POLITECNICO DI TORINO

MASTER's Degree in COMPUTER ENGINEERING

MASTER's Degree Thesis

Synthetic Generation of Air Traffic Trajectories Using Deep Learning

Supervisors

Prof. Santa DI CATALDO

Prof. Massimiliano RUOCCO

Prof. Francesco PONZIO

Candidate

Mattia MILONE

APRIL 2025

Synthetic Generation of Air Traffic Trajectories Using Deep Learning

Mattia Milone

Abstract

This thesis explores the application of deep learning models to generate synthetic air trajectories, represented as sequences of sectors and time spent within each sector. The study compares different neural network architectures, including LSTM and Transformer, trained and evaluated on operational datasets. Additionally, the research investigates the effect of conditioning models with external variables, such as weather conditions, to simulate more realistic flight scenarios. Given the computational cost of training, parameter optimization was conducted using a progressive approach. While the study is exploratory, the results offer a foundation for future developments in air traffic simulation and management, aligning with the objectives of the European SESAR 3 program.

ACKNOWLEDGMENTS

A first thanks to my supervisors, Prof. Santa Di Cataldo, Prof. Massimiliano Ruocco, and Prof. Francesco Ponzio, for their guidance throughout this thesis.
Thanks also to Murad and Alessio for joining several meetings and always contributing.

Thanks to the people at Sintef, the research center in Trondheim where this project was conceived, who were very kind to me.

Thanks to Politecnico for being a tough ground where I could challenge and build my abilities, and for making me a bit more aware of who I am. This journey started in October 2019, and a major milestone was achieved in July 2022 with the bachelor's degree, and now we are at the end of it.
I want to particularly remember the 10 months at the Norwegian University of Science and Technology in Trondheim.
Thanks to the Erasmus project for this opportunity; it was an amazing adventure that gave me so much.

I want to thank all my friends and the kind people I encountered. Big thanks and love to you all.

Lastly, the best for last. Thanks to all my family, who has always supported me in every possible way. They are my best fanbase, whose aspiration is to see me healthy and achieving my goals.

In particular, thanks to my mother, my father, and my brother.

Table of Contents

List of Figures

List of Tables

Acronyms

ML	Machine Learning.
AI	Artificial Intelligence.
DL	Deep Learning.
ATM	Air Traffic Management.
TOS	Trajectory Option Sets.
GDPR	General Data Protection Regulation.
ANSP	Air Navigation Service Provider.
SDG	Sustainable Development Goals.
RNN	Recurrent Neural Network.
LSTM	Long Short Term Memory.
Seq2Seq	Sequence to Sequence.
NLP	Natural Language Processing.
ATC	Air Traffic Control.
FIR	Flight Information Region.
AUA	ATC Unit Airspace.

Chapter 1

Introduction

The management of air traffic (Air Traffic Management, ATM) represents an important element that enables air carriers to operate safely and efficiently through the multiple services provided [1]. In recent years, the increase in air traffic [2], a trend that is expected to continue to grow [3], has shown the need to develop new solutions to address challenges related to the optimization of aviation operations, the reduction of environmental impacts, and the improvement of the overall efficiency of the system. In this context, artificial intelligence (AI) and deep learning (DL), are emerging for the digital transformation of ATM. This thesis fits in the SESAR 3 program (Single European Sky ATM Research) [4], a strategic initiative of the European Union aimed at modernizing the ATM system and promoting smarter and more sustainable air transport.

1.1 Motivation

Modern society has entered an era dominated by data, in which huge amounts of information are used and generated every day. In recent decades, the analysis of this data has shown an unprecedented potential to improve various aspects of our daily life, with numerous examples of data-driven models that have transformed entire sectors [5]. For example, models for weather forecasting [6] [7], in medicine models for genomic research [8], diagnosis [9], and drug discovery [10]. In the discovery of new materials [11]. Furthermore, in the financial sector and public safety operations. Models to improve the efficiency of industrial production supply chain [12]. And many others.

Adopting DL models in ATM enables new cognitive services never considered before. Solutions vary from aircraft performance (e.g., flight state, parameters, and trajectory optimization) to human factors (e.g., fatigue assessment) [1]. So far, various data-based applications have been developed, including traffic-related mining, autonomous path planning, and popular route discovery. In the air traffic management domain, trajectory data have been used to identify traffic patterns, estimate the loss of separation probability, and gain operation experience [5].

The critical importance of data in developing more efficient technologies has been

widely demonstrated. However, there are three main challenges related to data: low quality, scarcity, and privacy concerns [13].

The quality of data is a crucial aspect for the effectiveness of the models used in air traffic management. In fact, if flight route data is incorrect, incomplete, or contaminated by noise, it compromises the reliability of the predictions and optimizations performed by the models. Synthetic data could deal well in these contexts because it can replace some real misleading data. In the aerospace field these cases can occur for different reasons. Measurement errors can be a result from a malfunctioning system or interference in communications between aircraft and control tower. An incomplete dataset may result from rare but possible events are not registered yet or in a low frequency. In this case synthetic data can populate the dataset with more scenarios making the dataset more complete. Moreover, in cases where data is noisy or obsolete, data generation can be useful by trying to understand the pattern that has to be preserved and other to be discarded and then populating a new dataset.

Another fundamental issue in the aviation sector is the scarcity of data. Real flight route data can be difficult and costly to obtain. Synthetic data can address this issue by generating a larger dataset from an existing set of records. The third case is data privacy. Data can be confidential and restricted by regulations (such as those imposed by the GDPR for the protection of personal data). Since synthetic data does not come directly from real-world sources, it can help address some of these issues. For example, an aviation company with confidential data could train a model and then release synthetic data generated from it.

Therefore, synthetic data can be generated to replicate the statistical properties of real data, reducing privacy risks and allowing for the filling of gaps in real datasets, providing an abundance of useful data for training robust models for air traffic management.

1.2 Problem definition

This work wants to be a first approach to synthetic trajectory generation. The main goal is to explore different methods and compare then. Firstly, generating trajectories as is and afterwards by conditioning the models with external variables.

The trajectories have a specific, easier representation than the ones we can usually find in ATM applications, they are described as sequences of big geographical areas called sectors, with the time spent in each of them.

This makes this first work a better way to start exploring. The aim is to get effective models capable of synthesizing, close to reality, events while maintaining the spatial and temporal restrictions. Moreover, checking their flexibility by testing them in different contexts, with different datasets.

1.3 SDG Contributions

This research represents a first approach in the field of flight route generation through deep learning techniques applied to synthetic data. Although the work is exploratory in nature, it opens the way for further developments and concrete applications that could have a direct impact on various Sustainable Development Goals (SDGs), particularly SDG 9 ("Industry, Innovation and Infrastructure") and SDG 13 ("Climate Action").

SDG 9 aims to improve sustainable modernization in industrial sectors, boost innovation, and build resilient infrastructure. The research presented is very pertinent in this regard, as it suggests the use of deep learning models to generate synthetic data to increase the effectiveness of air traffic planning and optimization models. Because it makes use of technology innovation to improve vital infrastructure, such as the air traffic management system.

SDG 13 promotes actions to combat climate change. In this context, the proposed research can contribute to reducing carbon emissions. In fact, better routes planning can reduce flight times, leading to a decrease in CO2 emissions. In this regard, SESAR report (2025) [4] highlights the analysis that considers how the adoption of advanced technologies for air traffic management can reduce sector emissions. Other studies [14] emphasize that advanced simulation technologies can effectively model the impact of sustainable measures in the aviation sector. The same synthetic data could be used in optimization architectures. It has also been shown [15] that optimizing air routes is one of the most effective strategies to reduce the climate impact, thus making a significant contribution to operational sustainability in the long term.

In conclusion, this work is structured as follow: Chapter 2 explores the theory of deep learning architectures, while Chapter 3 provides a detailed analysis of the state-of-the-art in artificial intelligence for trajectory problems and air traffic management in general. Chapter 4 focuses on data exploration. All experiments, results, and discussions are presented in Chapter 5. Finally, the conclusions are drawn in Chapter 6.

Chapter 2

Background theory

2.1 Recurrent Neural Networks

For time series and data issues requiring sequences, such as natural language processing [16], signal and voice processing [17], video processing [18] RNNs have been adopted frequently with excellent performance. Because it uses a specific type of recurrent topology to retain information about prior knowledge, an RNN is a neural network that processes sequential data over time, making it ideal for time-series data. As seen in Figure 2.1, these ring networks are referred to as recurrent because they carry out the same computations and operations for every component of an incoming data sequence. Furthermore, RNNs use the idea of "memory" to retain the states or data of earlier inputs to produce the subsequent sequence output, so capturing longer relationships.

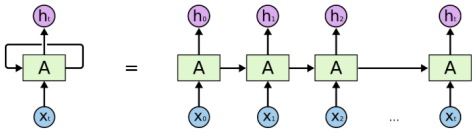


Figure 2.1: An unrolled RNN [19]
'A' is a NN

RNNs face limitations on the actual length of the input sequence due to the "vanishing gradient" problem: as gradients are backpropagated, they shrink exponentially, making it difficult for the network to learn long dependencies. This leads to poor performance on long sequences. Many solutions have been proposed to address this problem, two of which are LSTM [20] and GRU [21]. These are two types of recurrent networks designed in a more complex way than the standard RNN, which helps to improve performance on long sequences [22].

2.1.1 LSTM Networks

Long-short term memories (LSTM) were firstly introduced by Hochreiter & Schmidhuber (1997) [20]. As standard RNNs, LSTM has a form of chain of repeating modules of neural network. In standard RNNs, this repeating module have a very

simple structure, such as a single tanh layer. So, it means a NN with, for example, as activation function, a tanh, as the figure 2.2.

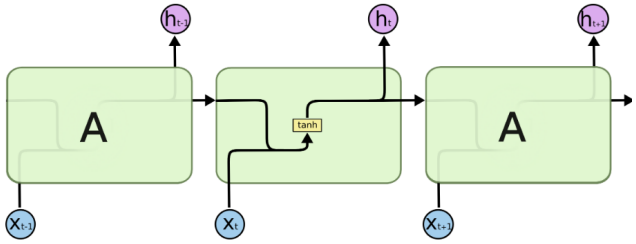


Figure 2.2: Simple RNN viewed as chain of a simple single model tanh [19]

LSTM is based on four modules, neural networks, that interact with each other in a specific manner.

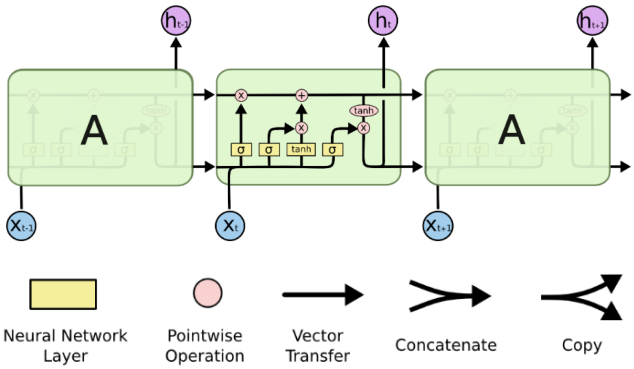


Figure 2.3: LSTM structure [19]

In Figure 2.3, each line carries an entire numerical vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, such as vector addition, while the yellow boxes represent learned neural network layers. Lines merging denote concatenation, while a forking line indicates that its content is copied and sent to different locations[19].

The architecture can ideally divided and described as 4 regions:

1. Cell State

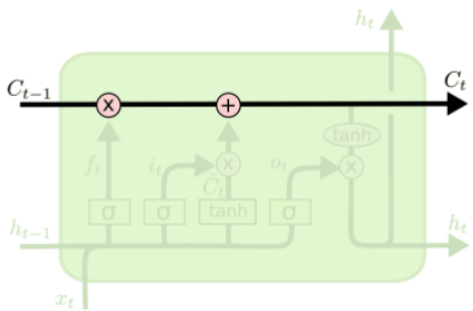
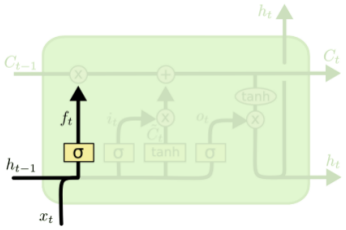


Figure 2.4: LSTM cell state [19]

The cell state is the horizontal line at the top of the diagram. LSTM can

remove or add information to the cell state, that is a numerical vector, thanks to the gates. Over time, the state is modified, keeping some old information, forgetting others, and starting to remember new ones.

2. Forget Gate

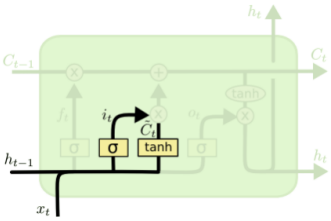


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.5: LSTM forget gate [19]

The forget gate decides what information will remain or be discarded in the cell state. The decision is made by a sigmoid layer that takes inputs $h(t-1)$ and x_t, and outputs a number between 0 and 1 for each element in the cell state $C(t-1)$. A value of 1 represents 'keep this completely,' whereas a value of 0 represents 'completely discard this'.

3. Input Gate

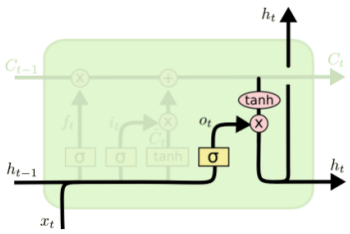


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 2.6: LSTM input gate [19]

The objective is to properly add new information to the old state $C(t-1)$. Here, the concatenation of the current input $x(t)$ and the previous hidden state $h(t-1)$ are passed parallelly through a neural network with a sigmoid function and a neural network with a tanh function. The two results are combined by multiplication and then added to the cell state, producing the current cell state $C(t)$.

4. Output Gate



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.7: LSTM output gate [19]

The output gate wants to deliver a filtered version of the cell state $h(t)$. This is done by multiplying the cell state $C(t)$ passed through a tanh function and the concatenation of the previous hidden state $h(t-1)$ and the current input $x(t)$ passed through a sigmoid function.

In this way, LSTM captures the context by getting relevant past information and updating it dynamically every cycle.

2.2 Attention Mechanism

This concept was first introduced in the Deep Learning field by Bandanau [23] applying it to seq2seq architectures that were built with RNN as LSTM and GRU.

- seq2seq are architectures that take sequence input and generate sequence output. That are made with an encoder and decoder, the encoder maps the input into a middle representation and then the decoder generates the output. An example are translator models that take a first sentence (sequence of string) and then generate the sentence of another language (order set of strings).

Attention mechanism helps to represent an input in a more meaningful way based on context. In seq2seq architecture helps the decoder to dynamically focus on more important information. That decreases the loss vanishing problem and overall increases model performance. Indeed, RNNs has a limited size of the hidden state. This means that all the information must be compressed in there, that can cause loss of information especially for longer sequences where old data are overwritten or faded off. This phenomenon is called bottleneck. Attention mechanism helps in that point. Attention mechanisms prevent this issue because the model accesses all the sequence directly without having to compress the information into the hidden state. Moreover, it makes the information to be dynamic and highlights the details that are more important at a specific time.

In figure 2.8 is represented is an example on how it could works. In the left side is a simple embedding that map each word of the input sequence individually while the attention mapping, on the right, add something else, each word of the input sequence interacts with each other to get new representation based also on local context. The attention mechanism can indeed be described as a dynamic function mapping, where the representation of a token is dynamically computed based on its context.

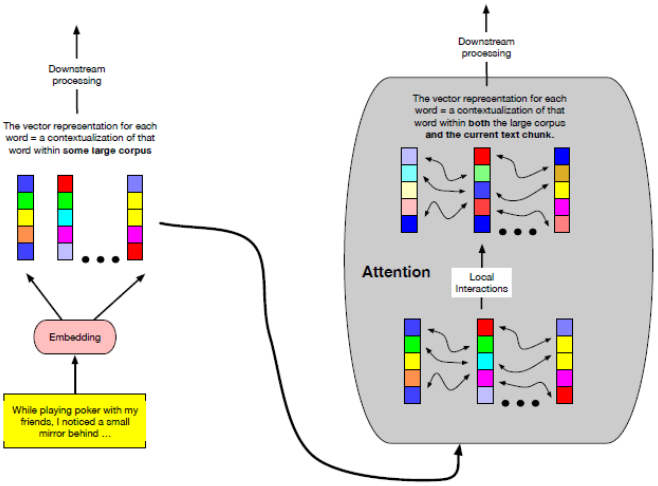


Figure 2.8

Later in 2017 was publish the article "Attention is all you need" [24] that formalize in a more generic way the attention mechanism with the following formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

Where:

- Q: **Query** matrix.
- K: **Key** matrix.
- V: **Value** matrix.
- d_k: Dimension of the key vector, used for scaling.
- The term $\frac{QK^T}{\sqrt{d_k}}$ computes the scaled dot-product attention.
- The final output is the weighted sum of the values V, using the attention weights.

The softmax function is used to convert a vector of values into a probability distribution making all the values sum to 1. The softmax formula is:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2.2}$$

Where:

- $\mathbf{z}$: Input vector (often represents the scores or logits, such as the result of the dot product in attention).
- z_i: The i-th element of the input vector $\mathbf{z}$.

The attention operation can also be seen as a retrieval process, that's why they name the 3 matrices key, value and query. In retrieval systems the query (the string

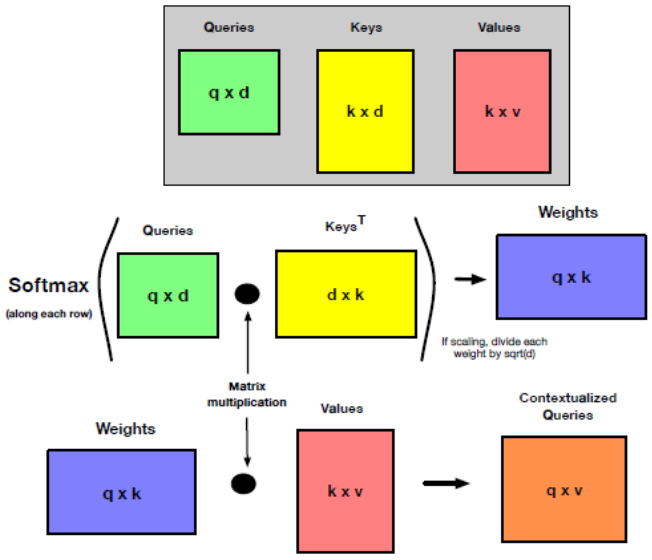


Figure 2.9: Attention mechanism: matrices view

prompt that we give through a searching bar) will be mapped into a set of keys associated with candidates in the database then will be presented to you with the best matches (values).

2.2.1 Self Attention

In the self-attention mechanism, the Value, Key, and Query all come from the same source. This allows the model to weigh the relationships between all the elements in the same sequence, including those that are far apart, in order to build a richer and more contextualized representation.

2.2.2 Cross Attention

In this case, a sequence of Queries (typically from the target side) interacts with a different sequence of Keys and Values (typically from the source side). This is used in situations where we want the model to focus on information from a different sequence, such as in Encoder-Decoder models, where one sequence generates another (e.g., translation).

In the training phase the goal is to correctly map the input sequence (source) to the output sequence (target), both of which are known, and the error leads to updates of the model's parameters.

In the inference phase the input sequence (source) is still present and provided to the model (as in the case of translation, where the source sentence is known). However, the target sequence (output) is not yet complete; thus, the model must generate it autoregressively, one token at a time.

2.2.3 Masked Attention

Used when it has to hide information from the model. For example, during the generation of a sequence, we shall not give any next expected token yet to the model (because we want it to guess them). This is implemented by adding, in the formula, a matrix, mask matrix, before the softmax function, with a large negative number to the tokens we want to mask. After the softmax, their score will be zero.

2.2.4 Multi-head Attention

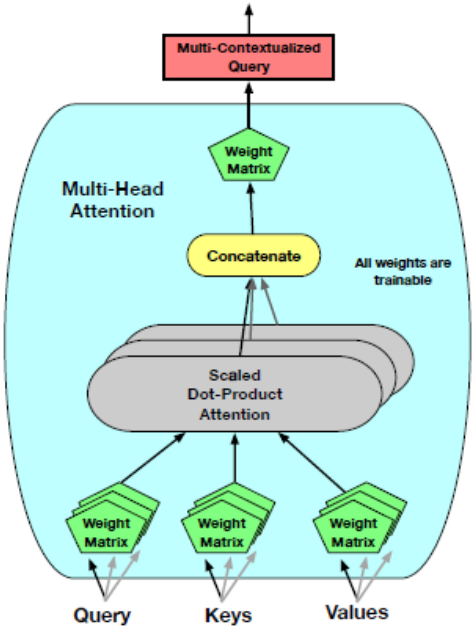


Figure 2.10: Multi head attention

It is called when multiple attention heads are used in parallel on the same input sequence. This is done to learn different patterns in the data, then the results from each head are concatenated and linearly transformed to obtain the final output.

2.3 Embedding

In ML embedding means to map data into a number vector of different dimensions. It's a must because deep neural networks need numerical input to be fitted in. To introduce Transformer, we need to know 2 different embeddings. Word2Vec [25] is used to transform words into vector and positional encoding is used to extract the token (i.e. words) order information. The first positional embedding developed by the authors of Transformer [24] was fixed, meaning it doesn't change during training and is defined as following:

$$p_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases}$$

where:

$$\omega_k = \frac{1}{10000^{2k/d}}$$

- t = position of the object in the input sequence $0 \leq t < L/2$
- d = dimension of the output space
- k = used for mapping to column indices $0 \leq k < d/2$ with a single value of k mapping both sin and cos

While learned positional embedding instead is simply a fully connected layer, they are learned by the model during training, exploiting the modern data-driven paradigm.

2.4 Transformer

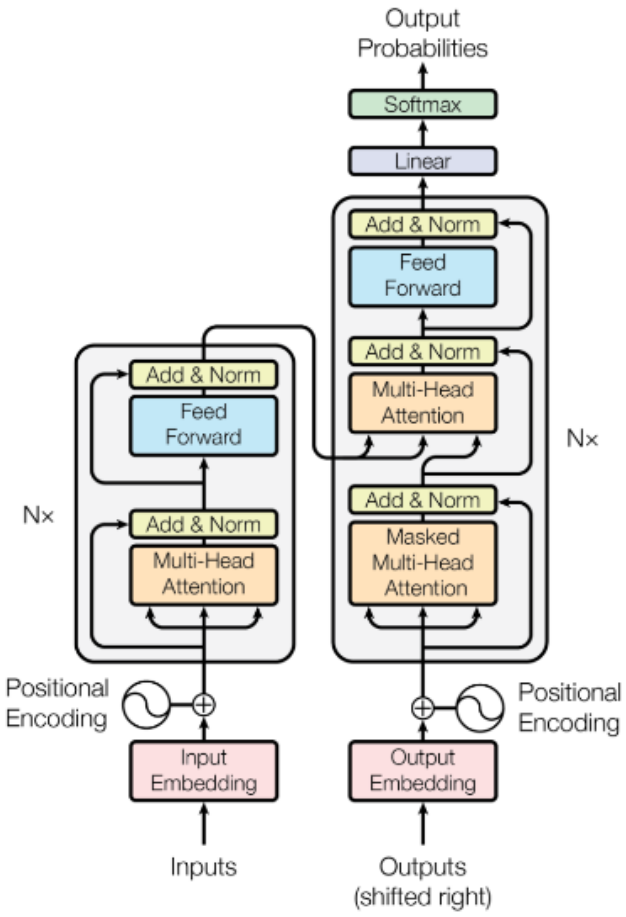


Figure 2.11: Transformer architecture [24]

Transformer architecture was designed to overcome the limitation of RNNs, which includes sequential processing. RNNs process tokens one at a time, making training

slow. Transformer comes as a basic structure as the seq2seq architecture, where there is an encoder and a decoder.

Encoder

The encoder consists of N layers, each composed of two sublayers: a self-attention mechanism and a feedforward network (as represented on the left side of the figure). The encoder processes the sum of different embeddings of the input. Positional encoding is crucial because there is no recurrence in the model; without it, there would be no way to capture order information explicitly. The first sublayer is a multi-head self-attention mechanism that captures relationships within the input. Multi-head attention enables the model to focus on different aspects of the input in parallel by applying separate linear projections to the queries, keys, and values. This results in multiple attention outputs (h), which are then concatenated to form a single output. The second sublayer is a fully connected feedforward network consisting of two linear transformations with Rectified Linear Unit (ReLU) activation in between. Additionally, each of these two sublayers is wrapped with a residual connection and a normalization layer to improve gradient flow and training stability.

Decoder

The decoder consists of N layers, each composed of three sublayers: a self-attention mechanism, a cross-attention mechanism, and a feedforward network (as represented on the right side of the figure). Similarly to the encoder, the input to the decoder is enriched with positional encoding to incorporate order information, as the model itself lacks recurrence. Unlike the encoder, which attends to all words in the input sequence regardless of order, the decoder attends only to previous words when generating a prediction. This ensures that the prediction for the word at position i depends solely on the words that precede it in the sequence. To enforce this constraint, the self-attention mechanism is masked in the decoder. Specifically, masking is applied over the values obtained from the scaled dot product of the queries (Q) and keys (K), preventing the decoder from accessing future positions (see figure: dot-product attention).

Chapter 3

Licterature Review

3.1 Introduction

In the ATM sector, but also in general applications related to route and trajectory management, the methods used can be divided into two categories: model-driven and data-driven.

The model-driven approach consists of building a mathematical or statistical model to solve a problem. In this approach, the focus is on the model and not on the data. These models can be very reliable, but in complex cases, they may struggle to find the pattern behind recorded events.

In contrast, in recent years, with the rise of machine learning, data-driven approaches started to be considered. These models rely on a large amount of data and aim to find the laws behind them. Still, these models can easily have a lack of interpretability.

In this chapter, I will introduce other studies related to deep learning applications for trajectory generation in general, as well as specific current applications in the ATM domain. The goal is to explore existing techniques, their strengths and weaknesses, while also understanding the ultimate applications of the projects.

3.1.1 Trajectory definition

Before fully stepping into state-of-the-art application, it's important to do a step back and define what its considered for trajectory.

- **Definition 1 (spatio-temporal point)** A spatio-temporal point p is a unique entity in the form of (o, t, l, f), which represent the access of moving object o to location l at timestamp t under the geographical coordinate system, and comes with an optional record attribute feature f.
- **Definition 2 (Trajectory)** A generalized trajectory T consists of a series of spatial-temporal point sequences (p1, p2, ..., pn) arranged in chronological order, which represents the movement information generated by moving objects in geographical space.

Based on the fundamental attributes of spatio-temporal points, trajectory can be extended into multiple forms.

- Firstly, with respect to object attributes, we can categorize them into Individual Trajectory, representing quasi-continuous tracking data of individual movements, and Group Trajectory, which denote the movements of a group of individuals during.
- Secondly, regarding time attributes, we can derive a spectrum of trajectories ranging from Sparse Trajectory (e.g., users' check-in data during travel) to Dense Trajectory (e.g., movement paths of vehicles equipped with GPS tracking systems) based on the dimension of sampling frequency.
- Thirdly, regarding location attributes, we can generate trajectories, also known as Raw Trajectory, by mapping coordinates (i.e. gps coordainates) to spatial embeddings. The new sequence generation of discretized tokens is referred to as Cell Trajectory. Further, trajectories composed of tokens with attribute features are called Semantic Trajectory.

Now, we can define our project more precisely. It works with trajectories considered at the individual level, rather than as a group. In terms of time attributes, the trajectory is dense, meaning there are no temporal gaps in the data. Regarding location attributes, the trajectory is cell-based, as it is represented as a sequence of spatial sectors, making it a discrete space problem.

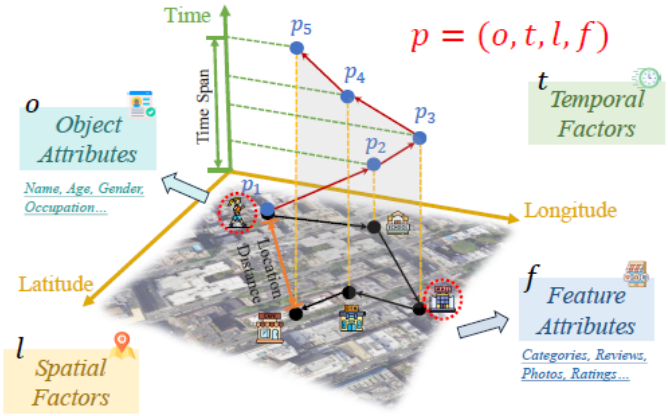


Figure 3.1: Illustration of a trajectory [26]

3.2 Deep learning solutions on trajectory generations in general

The following subsection is based on a 2024 survey [26] study that provides an extensive review of deep learning methods applied to trajectory application in general, as

well as trajectory generation applications.

Microdynamics route generation focuses on detailed individual movements, including visited locations, duration of stays, chosen routes, and timing. Early approaches use next-location prediction as as sequential task. One method, proposed by Mehmet et al. [27], uses a trajectory mixture model, though it requires large datasets and complex computations. Generative adversarial networks (GANs) introduced a shift, with Ouyang et al. [28] using grid-based trajectory data to create realistic paths, though balancing grid size and accuracy remains a challenge [29], [30]. Reinforcement learning (RL) offers another perspective, where trajectory generation is framed as a decision-making process, using a generator as an agent and a discriminator for evaluation [31], [32], [33]. Some methods transform trajectory data into images for processing, though this adds computational complexity [29]. More recently, denoising diffusion probabilistic models, like DiffTraj [34] and Diff-RNTraj [35], have emerged bringing in satisfying results.

DeepMove [36] is a deep learning model that uses an attentional recurrent neural network (RNN) with two attention mechanisms to recognize movement patterns over time, while GRUs process trajectory data for better predictions. VANext [37] improves on DeepMove by using variational attention, encoding recent movement features, and combining them with check-in preferences to enhance accuracy.

To address the challenge of sparse user movement data, Flashback [38] introduces an RNN-based approach that uses spatial and temporal information to discover hidden patterns with high predictive power. Another approach by Song et al. [39] implement a multi-task deep learning framework with stacked LSTMs to predict both future traffic patterns and user locations together.

Considering multiple factors on movement, MobTCast [40] applies transformer models to analyze temporal, semantic, social, and geographic contexts together. It also includes an auxiliary task to better understand geographic patterns and refine predictions. In recent research, location forecasting has expanded into two key areas: next POI (Point of Interest) recommendation and incomplete path prediction. POI recommendations help suggest new locations to users, especially in cold-start scenarios, while incomplete path prediction, useful in fields like food delivery and logistics, estimates a worker's upcoming stops based on their current, unfinished route.

3.3 Deep learning solutions on trajectory generation in ATM

In this part I will introduce the results of my research on studies that care about aircraft trajectory generation with deep learning that fits close to the same definitions of my problem so:

1. generation of single routes independently and not as a system of groups of

vehicles

2. dense trajectory so no missing pieces as can be for trajectory that comes from smartphone data

3. cell based trajectory, so the space threatened as sequences of area and not points

No study fully meets all these constraints, but the closest one is the first one I will discuss. That's because most of the other studies focus on solutions that represent trajectories in 4D (latitude, longitude, altitude, and time) rather than using a cell-based approach. Others, like the last study I'll mention does not fall under the deep learning category, as it relies on traditional machine learning techniques. But these remain a valuable reference and input for similar works.

3.3.1 Study 1 - STTN, waypoint sequence generation with Transformer

The paper under consideration , called "Sequence-to-sequence transfer transformer network for automatic flight plan generation" [41], addresses the problem of automatic flight plan generation, an activity traditionally managed by industry experts who manually modify sequences of waypoints—geographical points selected as references for air navigation—to ensure the safety and efficiency of air traffic.

The authors of the study implemented a sequence-to-sequence model based on a Transformer, called Sequence-to-Sequence Transfer Transformer Network (STTN), with the goal of mapping the input that is a pair of departure and arrival airports into an optimized sequence of waypoints. This model relies strongly on the effectiveness of attention mechanism that help to capture the relationships between airports and waypoints.

The process begins with the conversion of airports into numerical vectors through the airport embedding layer, which uses the SentencePiece method to segment airport names into subwords and obtain a compact and informative representation. Then, the airport information is processed by the airport pair encoder network, a network composed of six Transformer layers, where multi-head self-attention allows the model to understand the relationship between airports in terms of geographical distance, airspace restrictions, and commonly used routes.

Additionally, a relative positional embedding is introduced, improving the model's ability to understand the relative position of elements in the sequence. At the same time, the model includes a waypoint embedding layer, which transforms waypoints into numerical vectors. The last phase of the process is handled by the waypoint sequence decoder network, which generates waypoints sequentially. The decoder uses a self-attention and cross-attention architecture, allowing it to simultaneously use information about the departure and arrival airports and the waypoints already generated. The output at each stage is a probability vector, from which the most appropriate waypoint is then selected.

The authors trained STTN on a real dataset of flight plans collected in China between July and September 2019 that includes 19,238 flight plans, 389 airports, and 1,925 waypoints. BLEU and METEOR standard metrics were used in order to evaluate the model by measuring the similarity between the sequences generated and the real ones. The results showed that STTN significantly outperforms other sequence generation models, such as RNN, Seq2Seq, and standard Transformers.

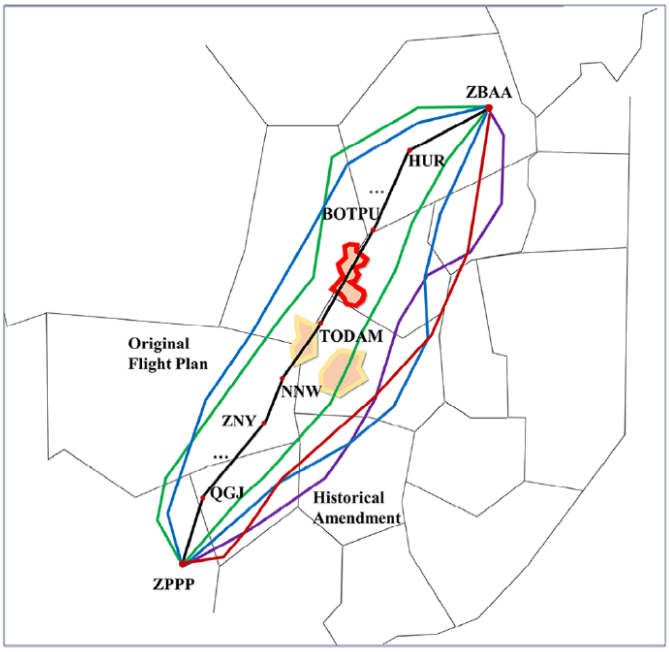


Figure 3.2: Waypoint trajectories. The black one is the official flight plan while the blu, red and purlple the historical modified one. The modification are made due to external factors (i.e. severe weather, congestion, rules) [41]

3.3.2 Study 2 - CTG-MMAT

The starting point of this research [5] is the observation that traditional trajectory optimization methods fail not to use historical air traffic data by not considering crucial aspects such as typical pilot maneuvers, ATC instructions, and the real operational conditions of aircraft.

The researchers developed a model called CTG-MMAT (Connection-Based Trajectory Generation with Majorization–Minimization-Based Adversarial Training). Instead of generating an artificial route from scratch, the model uses segments of real flights, connecting them with artificially generated intermediate part. This approach makes possible to maintain high fidelity of the trajectories improving diversity and flyability.

However, a key problem in this process is ensuring that the generated trajectory is flyable by aircraft. Here, the second component of the model comes into play: MMAT (Majorization–Minimization-Based Adversarial Training), a surrogate model which imitates the behavior of real operational restrictions, such as speed limits, altitude constraints, turn angles, and acceleration.

The trajectory generator produces new connections between flight segments, while the surrogate network evaluates which is in the flyability constrains. If a trajectory doesn't, it is discarded, and the model is refined to improve its generation.

This approach differs from traditional Generative Adversarial Networks (GANs) because it does not use a classic discriminator that distinguishes between real and synthetic trajectories. Instead, the surrogate model acts as a verifier that ensures trajectory to follow aeronautical rules.

An additional advantage of the model is its computational efficiency. Thanks to a simplified architecture, based on a few layers of dense neural networks, CTG-MMAT can be trained on standard hardware, reducing development costs and making it easily applicable in operational scenarios.

The result is a more controlled and targeted generation, which does not simply replicate a statistical distribution but takes into account the real operational needs of air traffic.

Tests conducted on real data from Guangzhou Baiyun Airport have shown that CTG-MMAT is capable of generating trajectories with high fidelity compared to real ones, while also incorporating typical air traffic controller maneuvers, such as deviations, holding patterns, and operational shortcuts.

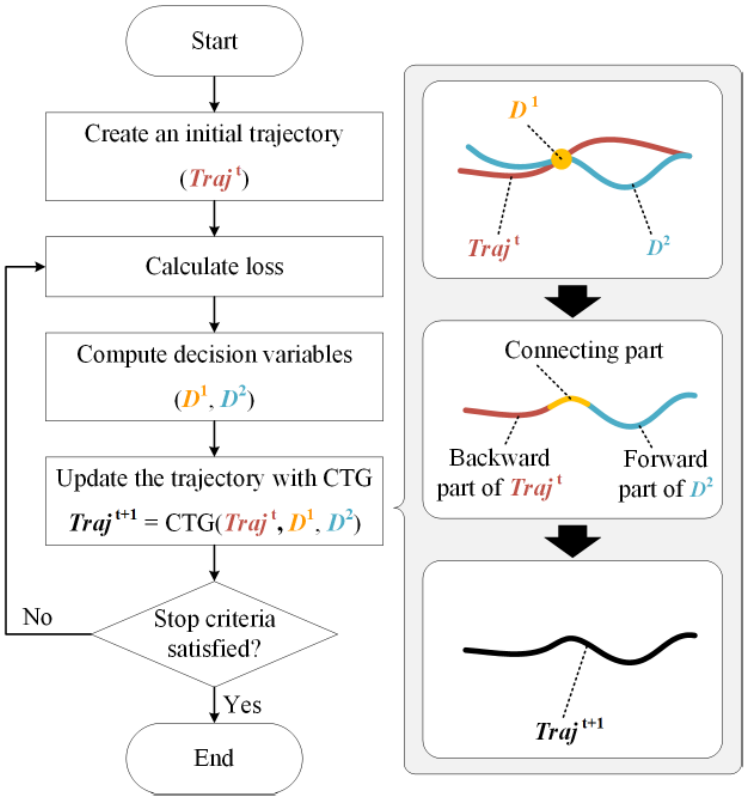


Figure 3.3: CTG it's an optimization loop where D1 and D2 are decision variable. Once peaked the trajectory is creare as concatenation and checked with some criteria. [5]

3.3.3 Study 3 - Trajectory prediction with time-frequency transformation

The research "Flight Trajectory Prediction Utilizing Time-Frequency Wavelet Transform" [42] introduces a novel method that uses the Discrete Wavelet Transform (DWT) to improve the next location forecasting in flight paths.

The Wavelet Transform-based Flight Trajectory Prediction (WTFTP) model presents an encoder-decoder neural network that examines flight trajectories in both the time domain and the frequency domain. The main concept involves breaking down the trajectory into parts at various frequency levels, enabling the model to differentiate between the overall flight patterns and finer local movements.

The operation of the model can be categorized into three primary stages. During the initial stage, the spatial history of the aircraft is handled through the Discrete Wavelet Transform, which breaks it down into low-frequency and high-frequency elements. The low-frequency elements reflect the overall motion of the aircraft, whereas the high-frequency elements capture more precise details, including consistent shifts in direction or altitude.

Afterwards, these data are handled by an encoder-decoder network. The encoder retrieves the time-related features of the trajectory, whereas the decoder produces the wavelet coefficients for upcoming trajectories. A crucial component of architecture is the Wavelet Attention module (WAtt), which assists in pointing the most significant information, enhancing the accuracy of the prediction.

Ultimately, with the help of the Inverse Discrete Wavelet Transform (IDWT), the model rebuilds the aircraft's future trajectory from the calculated wavelet coefficients.

Experimental results indicate that the WTFTP model exceeds the performance of traditional models like LSTM, CNN-LSTM, and Transformer. Specifically, the wavelet-driven method increases the accuracy by 35%.

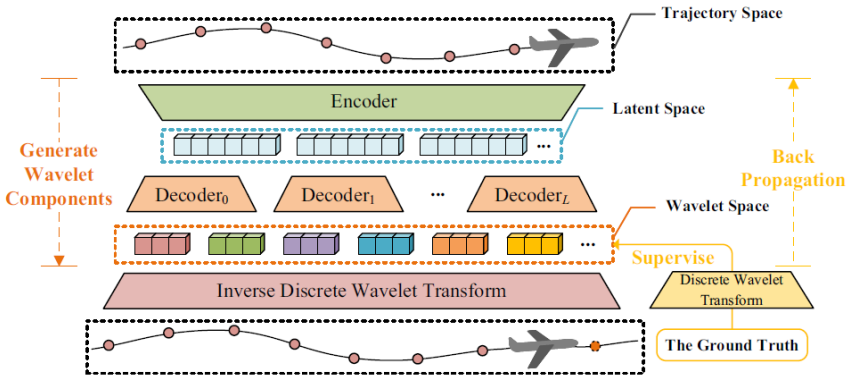


Figure 3.4: Model with wavelet transformation schema [42]

3.3.4 Study 4 - Generating trajectory with VAE

The research titled "Deep Generative Modeling of Aircraft Trajectories in Terminal Maneuvering Areas" [43] has the goal to produce a reliable generative model of 4D air trajectory that could be useful in airspace design, collision risk assessment, and Monte

Carlo simulations. This model was proposed based on a Variational Autoencoder (VAE) which is augmented with Temporal Convolutional Networks (TCN) and a Variational Mixture of Posteriors (VampPrior) to enhance the organization of the latent space. This specific configuration made the model of the architecture: VampPrior TC-VAE. This works as an encoder-decoder architecture where the first step is to represent at best the data input into the latent space, and then from it, having the opportunity to sample and generate data consistently and with enough variance. Because standard VAEs typically use a simple Gaussian prior that can lead to poor diversity, they chose to replace it with a learned mixture of posterior distributions. This model was trained on data from Zurich Airport, and the results demonstrate that it outperforms traditional Gaussian Mixture Models (GMMs) and standard VAEs in capturing the statistical distribution. The metrics used were the e-distance metrics and simulation tests in the BlueSky ATC simulator to check also the real flyability of the routes

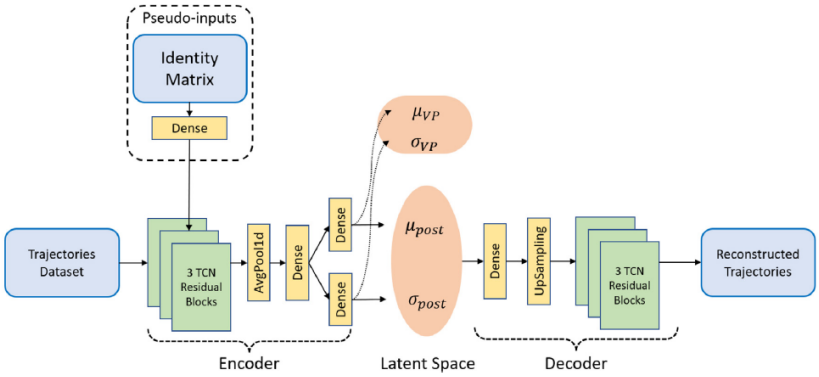


Figure 3.5: VampPrior TCVAE structure [43]

3.3.5 Conclusion

We have examined different methods for trajectory generation, each having unique techniques and goals. Study-3 implements an encoder-decoder architecture that uses a frequency transformation. It has the main goal of predicting the next point location, as said before, it can easily be seen as a potential generator if the location forecasting is iterated until the end of the route. Study-1 works with a Transformer that, given the start and end, creates the path, and Study-4 with Variational Autoencoder, Temporal Convolutional Networks, and Variational Mixture of Posteriors together.

The representation of the routes is different. In Study-1, the trajectory is categorical as a sequence of waypoints, while in the others, they are mostly 4D points.

The goal and, therefore, the metrics used vary a lot. Study-4 and Study-2 focus mostly on generating flyable routes, and so Study-4 uses the BlueSky simulator as a metric, while Study-2 implements a specific surrogate model in addition to the generator to filter data that do not follow aeronautical rules. Both have the intention to generate data that can then be useful for analysis and simulations. Study-1's primary goal is to optimize the path to reach the destination, while Study-3's primary

goal is to develop an application to analyze air traffic by predicting the next step flight positions.

Chapter 4

Data exploration

The data I have been working on are from Eurocontrol, an intergovernmental organization dedicated to supporting European aviation. They release traffic datasets covering all historic commercial flights in four fixed, sample months of specific years, while remaining within the restrictions of use placed on. 'Historic' means a delay of two years before release. 'Commercial' means excluding military, state and general aviation flights. In this project the table I have been working on are 2 with the same format, the FIR and AUA dataset.

Term	**Description**
ECTL ID	Unique numeric identifier for each flight in Eurocontrol PRISME DWH
Sequence Number	Numeric sequence number of the airspace entered by the flight in chronological order
FIR ID	Identifier of the FIR (Flight Information Region)
Entry Time (UTC)	Time when the flight entered the airspace (in UTC)
Exit Time (UTC)	Time when the flight exited the airspace (in UTC)

Table 4.1: FIR flight table

Term	**Description**
ECTL ID	Unique numeric identifier for each flight in Eurocontrol PRISME DWH
Sequence Number	Numeric sequence number of the airspace entered by the flight in chronological order
AUA ID	Identifier of the AUA
Entry Time (UTC)	Time when the flight entered the airspace (in UTC)
Exit Time (UTC)	Time when the flight exited the airspace (in UTC)

Table 4.2: AUA flight table

The difference is in the way the air space is divided.
The FIR (Flight Information Region) is a geographical subdivision in which

an aeronautical authority provides Flight Information Service (FIS) and Alerting Service (ALRS). Every portion of the Earth's atmosphere is included in a specific FIR. The airspace of smaller countries is incorporated into a single FIR, while for larger countries, the airspace is divided into a certain number of regional FIRs. These regions never overlap between themselfs. FIRs can be further subdivided into Upper Information Regions (UIR) to manage high-altitude traffic.

On the other hand, AUA means Air Traffic Control Unit Airspace. These airspace area are assigned to a specific air traffic control unit (ATC unit). These regions are into a FIR and can overlap depending on operational needs. AUA are flexible and can be redefined to improve traffic management.

The difference is that FIRs have a macro-informative purpose within their area, describing conditions and issuing alerts. AUA, on the other hand, have detailed operational management to ensure the safe separation of the single aircraft and facilitate delicate maneuvers such as takeoff and landing.

FIRs are well-defined and internationally recognized areas, whereas AUA are not; they can overlap and change over time. Therefore, a map of FIRs is available, but not of AUA.

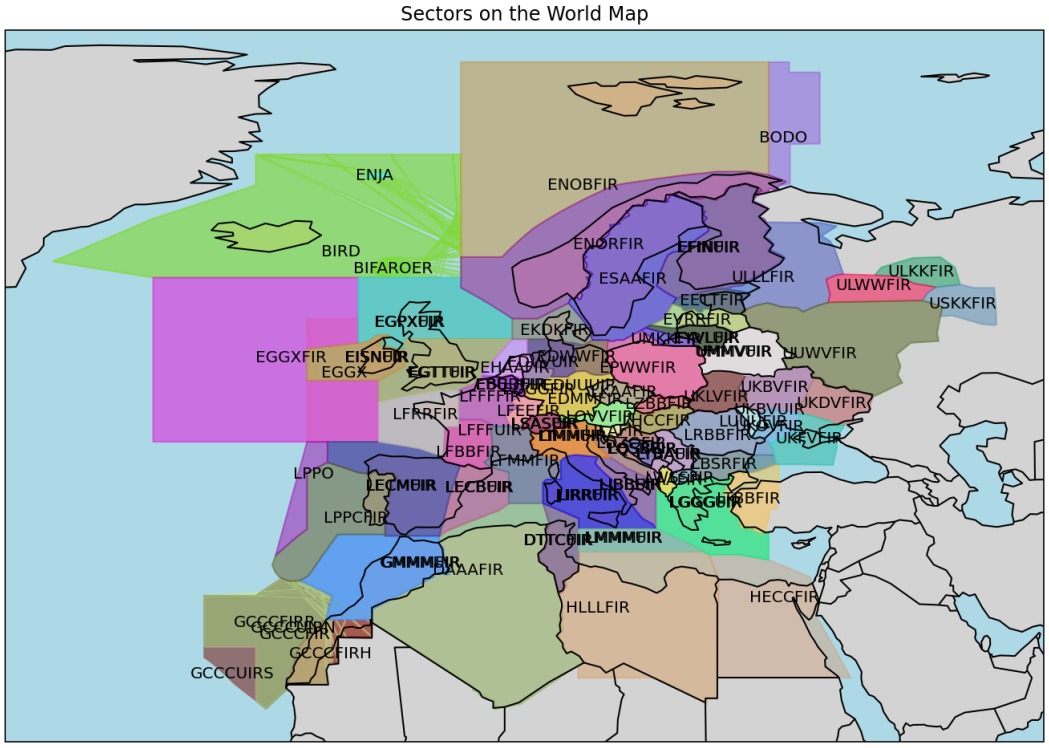


Figure 4.1: EU FIR sectors

Of all the available data, the models will be based on 10,000 flight routes. Specifically, some of these routes were recorded by Eurocontrol in December 2022. Only European routes were selected, and then a random sample was taken to reach 10,000 routes.

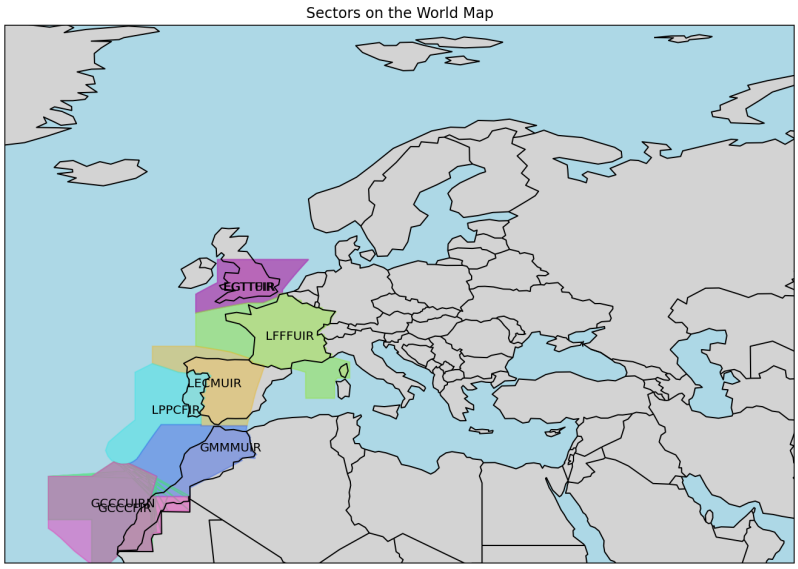
Among these 10,000 routes, there are about 1,300 unique routes, meaning they have the same departure and arrival points.

4.1 Data visualization

To effectively visualize the trajectory, both spatial and temporal information must be represented.

- Temporal Representation: An inter-row plot is used to display the time spent in each sector (Figure 4.2).
- Spatial Representation: The Python library Cartopy has been utilized to map the trajectory in space (Figure 4.3).

At Eurocontrol, a table is available that maps each FIR sector to the geographic points forming the polygons of the area. This dataset enables a full visualization of the trajectory.



TAXI_OUT,0.0;EGTTFIR,9.43;EGTTUIR,3.67;LFFFUIR,58.5;LECMUIR,50.3;LPPCFIR,61.0;GMMMUIR,55.9;GCCCUIRN,21.2;GCCCFIR,13.1;TAXI_IN,0.0

Figure 4.2: trajectory - space

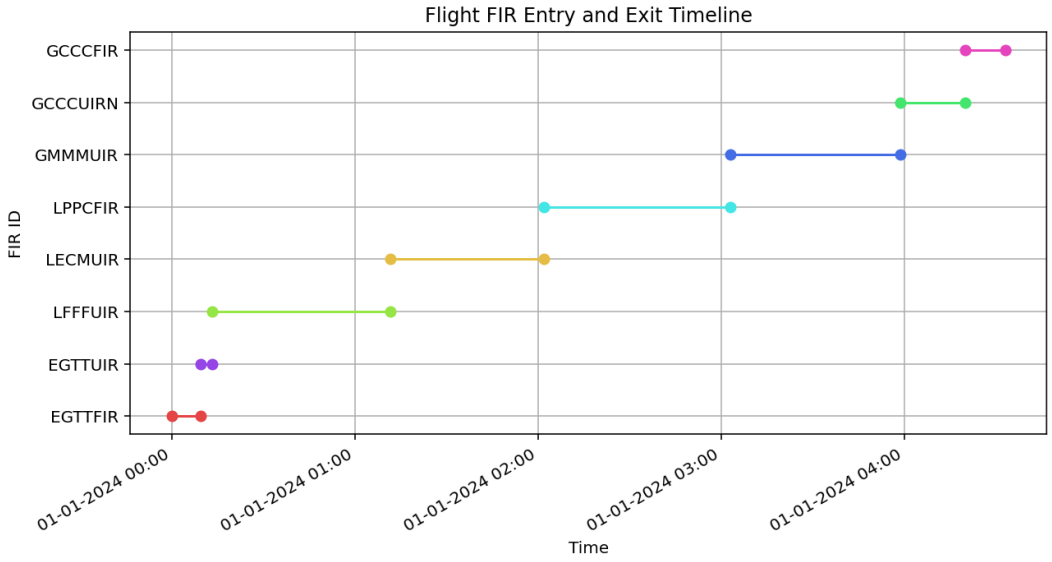


Figure 4.3: trajectory - time

4.2 Data selection

To make the study more relevant to the European airspace, it was decided to select only the sectors located in Europe or in nearby areas. Each sector is defined as a set of points that, joined in order, form a 2D polygon parallel to the Earth's surface. To determine which sectors to include, an imaginary 2D polygon was drawn, representing a large area covering the European continent and the surrounding zones. Only the trajectories composed of sectors with at least one vertex inside this polygon were selected. This operation was carried out using the Shapely library. This process was first applied to the FIR dataset, and then the same trajectories were filtered in the AUA dataset to ensure comparability when training the model in both cases. This was done by looking at the same trajectory ID identification.

Chapter 5

Experiments

5.1 Problem 1: Trajectory generation

The main objective of the project is to develop models capable of generating flight routes based on the available datasets (see Data chapter). The routes can be represented as discrete sequences of airspace sectors, each corresponding to a specific area of the airspace and identified by a unique string. A key aspect of the trajectory is the time spent within each sector. The generation of trajectories can be divided into two distinct problems:

- Sequential categorical generation: concerns the sequence of sectors crossed, similar to a Natural Language Processing (NLP) problem.
- Sequential continuous numerical generation: concerns the time spent in the sectors, comparable to a time series problem.

The two problems were initially addressed separately, starting with the generation of the sequence of sectors and then integrating the time component as well. The entire project was developed in Python, using PyTorch [44] as the main library for implementing deep learning architectures. For this first phase, the data were extracted from Eurocontrol FIR datasets (see Data Exploration chapter). Empirical considerations related to computational costs led to the selection of a subset of the available data. Moreover, it was decided to focus the analysis on trajectories within the European continent and surrounding areas, in line with the objectives of the SESAR 3 project. This choice is based on the hypothesis that greater geographical specificity can lead to more relevant results concerning the project's objectives.

5.1.1 Data pre-processing

The nature of the data is tabular, now we focus on three features, the following:

Flight Id	Sequence number	FIR Id

Where I recall that the Flight Id identifies the flight within the Eurocontrol database ecosystem, the FIR Id is the string that identifies the sector, and the

Sequence number indicates the order in which the aircraft of that flight is in that specific sector. Starting from this format, I applied a transformation to obtain sentences, meaning sequences of words. Here an example:

Flight Id	Sequence number	FIR Id
1	0	ABCD
1	1	REFT
1	2	EREF
2	0	YUYT

We can see two trajectory with the respectively Id number 1 and 2:

- ABCD->REFT->EREF
- YUYT

Data to be processed by neural networks needs to be numeric, which is why tokenization is applied as a pre-processing step. Tokenization consists of viewing text as sequences of elements called tokens—in this case, sectors—and then mapping each token into a unique integer number. The result is a dictionary capable of converting tokens into numbers and vice versa, thus transforming phrases into tokens and back, serving as a bridge between human and machine language. This was done simply by reading through the entire formatted dataset of "phrases" and adding new tokens to the dictionary with an incremental index. During the training phase, we want our model to learn which "word" to generate given some previous words. To achieve this, the data were prepared so that each input sequence corresponded to an output label. Specifically, from each sequence, 0-gram combinations were created. By 0-gram, I mean splitting a sequence of elements into multiple pairs of sets, where one serves as the input (x) and the other as the output (y).

PyTorch library was used to support the model implementation and since many PyTorch model-functions require input of consistent shape, all the input sequences were padded to match the length of the longest one. Specifically, the input data is structured in three dimensions:

(seqLength, batchSize, channels)

Padding tokens are added to shorter sequences until they match the longest ones. Additionally, each token sequence is expanded by adding special Start and End tokens at the beginning and end, respectively. This allows the model to generate sequences independently from any starting point and determine when the trajectory reaches its destination.

5.1.2 Architecture

The architectural approaches considered have been two. Since each sector must correspond to the time spent in it, there were two possible approaches for a model to "predict it".

In the first approach, two separate models have been used: one dedicated to generating the sequence of sectors and the other to generating the corresponding times. The result is therefore a structure with two models in cascade, see Figure 5.1.

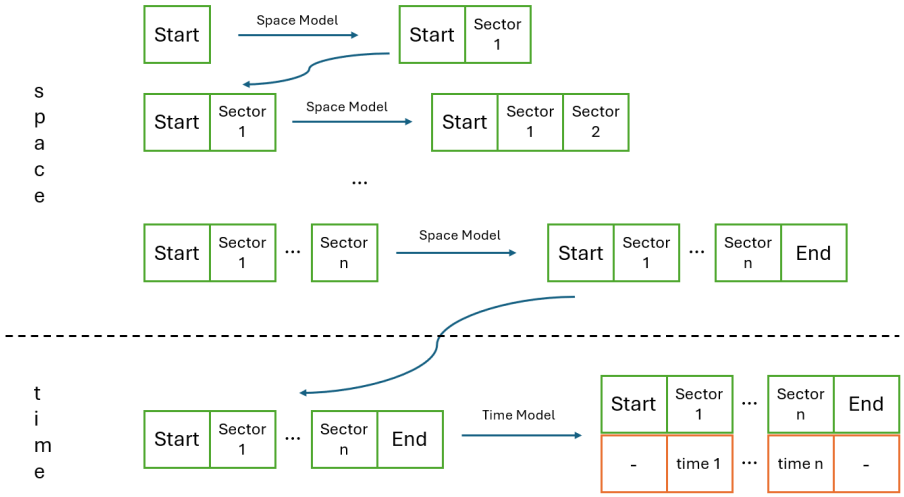


Figure 5.1: Cascade architecture scheme where firstly the space sequence is generated and then the time is assigned to each sector

In the second approach, instead, a single model was chosen, which, at each step, directly generates the pair (sector, time), see Figure 5.2.

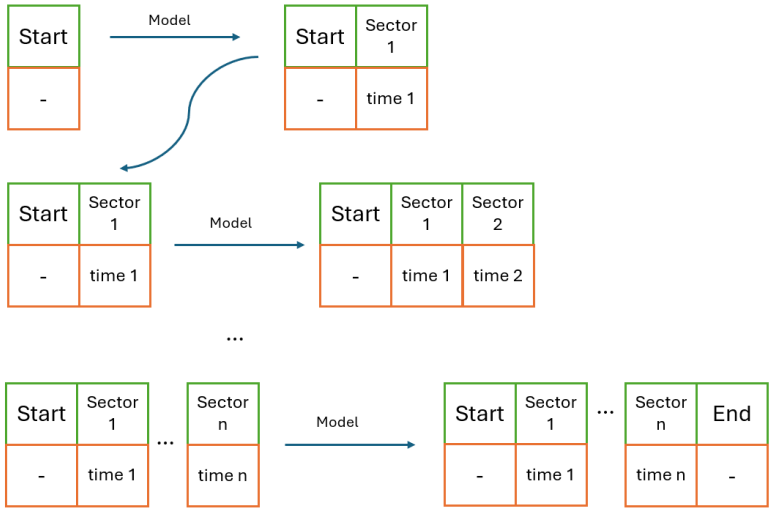


Figure 5.2: Step-by-step architecture scheme where at each iteration the pair (next-sector,time) is added

In the end, for the first architecture, there will be four models: one for generating the sequence of sectors with LSTM, one for generating the times with LSTM, one for generating the sequence of sectors with Transformer, and finally, one for generating the times with Transformer. For the second architecture, only one model has been implemented, and the model is based on Transformer.

For spatial sequence generation models, the first step was to apply an embedding to the input. This transforms the categorical information about the sectors into

dense representations with a smaller dimension. Embedding helps to capture hidden relationships between the sectors, such as their spatial proximity or how often they appear in sequence. Specifically, the Embedding function from PyTorch was used. This function initializes vectors randomly, and during training with backpropagation, it learns the best way to represent similar inputs with similar vectors. This process is like Word2Vec, but without using a pre-existing dataset.

After this step, the transformed input is passed to a LSTM or to a Transformer-based architecture.

For each initial sequence of tokens, the model produces an output with the same size as the token dictionary. Each entry in this output represents the probability of that token being the next one in the sequence. To get this result, the output goes through a feed-forward layer, which changes its size to match the dictionary size. Then, a Softmax function 5.1 transforms it into a probability distribution.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{5.1}$$

Basically, the goal is to have a model that, given an input sequence, can estimate the probability that a certain sector is the next one in the trajectory or that the sequence should end. For time assignment models, instead, the output is not a single value, but a sequence of numerical values. Here too, the input first passes through an embedding layer and is then processed by an LSTM or a Transformer. However, the output is not just a vector of times but a probability density vector. During the first prototyping phase, it became clear that generating single values led to an overly rigid embedding, where each input was linked to a fixed numerical vector, limiting the time variability of the trajectory. To overcome this limitation, the models were designed to learn the time distribution for each sector, given a specific trajectory. This was implemented using the Reparameterization Trick, which allows the model to obtain a probabilistic representation and capture the intrinsic uncertainty in the time spent within each sector, see Figure 5.3.

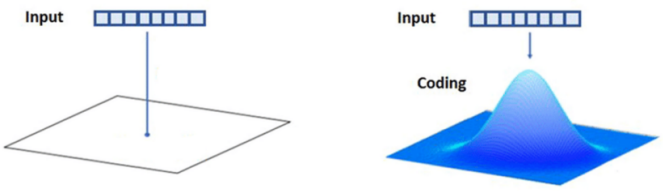


Figure 5.3: Left: mapping from input to a point and Right: mapping to a distribution

The reparameterization trick allows backpropagation of the gradient through random variables. Instead of directly sampling from the observed times, the model learns the parameters of a distribution, such as the mean and standard deviation for a Gaussian distribution. The output of a model like an LSTM or Transformer is not directly a deterministic value, but the parameters of a distribution (such as μ and σ

for a normal distribution), which represent the expected stay time for each sector.

Instead of directly sampling from a normal distribution with learned parameters, the following formula is applied.

$$z = \mu + \sigma \cdot \epsilon \tag{5.2}$$

where ϵ is random noise sampled from a standard normal distribution $\mathcal{N}(0, 1)$. The final architecture takes advantage of all the previous considerations, resulting in a model that, given an input vector that is the concatenation of a sector vector and a time vector, generates both the probability distribution of the next token and the parameters for sampling the next time.

5.1.3 Train and generation

The training phase involves updating the model parameters through backpropagation, based on the impact each parameter has on the output loss. The loss is calculated using a loss function that is specifically selected for the task at hand. Its purpose is to define a way to compare the actual output with the expected one. The generation function is then designed to adapt trained models to produce synthetic trajectories.

LSTM Path | Transformer Path
Both the LSTM and Transformer models share the same functions for training and generation. The loss function used is cross-entropy loss, which compares the predicted sector with the actual observed sector at each step in the trajectory. Given a target sector and a predicted probability distribution over possible sectors, the cross-entropy loss is computed as follows:

$$\mathcal{L}_{\text{sector}} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{5.3}$$

where:

- y_i represents the **true probability distribution** over the sectors, a one-hot encoded vector where only the correct sector has a probability of 1.
- $\hat{y}_i$ represents the **predicted probability distribution** over all possible sectors, obtained after applying the softmax function to the model's output.

The generation algorithm starts with an initial set of tokens, which can simply be the <start> token. This set is provided to the model, which then generates distributions for the next token. A challenge arises here: selecting the most probable token each time results in a lack of variability. To address this, top-p sampling was introduced. Top-p sampling involves selecting the next token from a subset of the most probable tokens, rather than always choosing the single most likely one. This method introduces variability while maintaining a high likelihood of selecting a plausible next step. The tokens are sorted in descending order by their probabilities. Starting from the top, the probabilities are cumulatively summed until they exceed

a predefined threshold p (e.g. p=0.9). Only the tokens within this cumulative probability range are considered for the next step. A token is then sampled according to their relative probabilities.

Given the sorted token probabilities $P(x_1) \geq P(x_2) \geq \cdots$, top-$p$ sampling selects the smallest set S such that:

$$\sum_{x \in S} P(x) \geq p$$

The next token is sampled from this set S. The threshold p is a parameter that heavily influences the model's performance. Once the next token is sampled from the most probable ones, it is appended to the previous token, and the procedure is repeated until the end token is predicted.

LSTM Time | Transformer Time

For time generation, given the full sequence of sectors, the loss function used is the Gaussian Negative Log-Likelihood Loss (NLL):

$$\mathcal{L}_{\text{time}} = \frac{1}{2}\left[\log(2\pi\sigma^2) + \frac{(t-\mu)^2}{\sigma^2}\right] \tag{5.4}$$

where t is the actual observed time, while μ and σ are the outputs of the model, which learns the time distributions by modeling them as Gaussian distributions. The generation part involves obtaining the distributions for each sector from the model for a given trajectory and then sampling from them.

The sequence generation and the time generation are then considered together as a single trajectory generation model.

Path&Time Transformer

This model has to deal with picking the next sector and time. So, it provides both the next token distribution and the parameters of the time Gaussian distribution. It's a combination of the previous ones.

$$\mathcal{L} = \lambda_{\text{sector}}\mathcal{L}_{\text{sector}} + \lambda_{\text{time}}\mathcal{L}_{\text{time}} \tag{5.5}$$

The loss function in this case it's a weighted sum of the loss equations seen before, the cross-entropy (5.3) for the sector choice and the Gaussian NLL (5.4) for the time. The weights λ_{sector} and λ_{time} had been chosen then empirically taking the ones with the best performance.

5.1.4 Evaluation metrics

The metrics to evaluate how good the model perform has to match the final goal of the project, in this case to get sufficient realistic generated trajectory. To measure so I had implemented 4 metrics:

- **feasibility**: it's an accuracy metrics. It makes the ratio between the number trajectories generated that are 'feasibile' over the total. I defined a trajectory

as 'feasibile' if it's a sequence of sectors which are bordering. To implement this in the project it was needed to retrive the geographical coordinates of each sector and that check if the 2 sequential sectors were sharing at least one edge.

- **converability**: it's also a ratio between the number of sectors used in the generated set over the total number of different ones in the training set. This wants to check how broadly the model is learning. Indeed, a low ratio has been experinced that means the models is overfocusing on few patterns, same type of trajectories. Instead, a close to 1 coverability means it's using almost all possible sectors in the traning so, having a lot a more variance.

For the time, the 2 metrics focus more on the feasibility of the data. It want to check if the time generated makes sense with respect to the trajectories given in the training phase. They are similar and make an accuracy estimation between the time considered feasible and the total. In this case a time is considered feasibile if it's in between the range minimum and maximum seen in the training set.

- **trajectory feasibility**: check if the total time of a specific sequence of sectors is in between the times seen in the learned set
- **sector feasibility**: check if the time spent in the sector in a specific condition is in between the range min-max seen in the training set.

ex. a specific trajectory that goes sectors A;B;C;D has a fesaibile time if is between the minimum and maximum times ever seen in the training set for the same trajectory A;B;C;D

5.1.5 Experimental settings

In machine learning there is a distinction between model parameters and hyperparameters. The firsts consists on internal model value that are estimated from the data during training while the latter cannot be modified by the value but is up to the developer to set them. This choice process is called hyperparameter tuning and can affect critically the overall performance.

The most important hyperparameters in a deep learning environment are:

- Learning rate: It controls the step size at which an optimization algorithm updates the model's weights during training. A larger learning rate can speed up learning but may reduce precision in finding an optimal solution, while a smaller learning rate ensures more precise convergence but may slow down training.
- Batch size: it refers to the number of training samples processed before the model updates its weights. It affects training speed, memory usage, and model performance.
- Number of layers: total count of hidden and output layers in a neural network. It determines the model's depth and its ability to learn complex patterns. More layers increase computation.

- Number of Hidden Neurons: refers to the total neurons in the hidden layers of a neural network. It impacts the model's capacity to learn patterns, with more neurons enabling greater complexity but increasing computation.
- Activation function: determines whether a neuron should be activated by applying a mathematical transformation. It can introduce non-linearity, allowing the network to learn complex patterns.
- Dropout: it's a regularization technique in deep learning that randomly deactivates a subset of neurons during training to prevent overfitting and improve generalization.
- Number of Epochs: it's the number of times the entire training dataset is passed through the model during training. More epochs can improve learning but may lead to overfitting.
- Number of heads (transformer): it refers to the number of attention mechanisms running in parallel within a multi-head self-attention layer. More heads allow the model to capture different aspects of input relationships. It increases computation.
- Threshold p-sampling: In our NLP scenario of generating the sector sequence, it controls the maximum cumulative probability that must be reached to select the next token, thus influencing the creativity of the model's choices. This is only during the generation phase and does not influence the computational cost.

5.1.6 Optimization

The optimization of the hyper-parameters is always a time-consuming task. The model has to run multiple times with different configuration. Taking into consideration the resources available the "stair step" optimization method had been chosen. It consisted of having a queue of parameters to optimize, so after getting the best one at the top of the line, it was kept and then moved to find the best for the next one.

The Adam algorithm (Adaptive Moment Estimation) technique was implemented, specifically to optimize the learning rate in order to converge to the model's best solution more effectively and quickly. Adam is a merger of two techniques, Momentum and RMSProp. Momentum, during the training of a model, considers the past gradients to accelerate convergence and bypass local minimum solutions. RMSProp, on the other hand, adapts the learning rate for each parameter based on the average of recent gradients.

During the optimization process, it was observed that optimizing for a single was leading to the other to decrease drastically. Therefore, the goal was to find a balance between all the metrics. When considering feasibility and coverability, it's clear that a model with limited flexibility often produces a high ratio of feasible routes, but these routes tend to share similar patterns. On the other hand, focusing too much on

coverability can lead to a larger number of generated routes that lack feasibility.For example, changing the generation value, which determines the threshold in the top-p sampling, was seen that a higher value, that narrows down the token choices, was making the most probable tokens more likely to be selected reducing variability and making coverability to go down, while a lower value allows less probable tokens to be chosen, increasing pattern variability.

Another aspect seen during the hyperparameter optimization was that adding complexity to the model, for example by adding more layers or heads in the Transformer doesn't have the direct consequence to get better performance.
The table shows the selected parameters that were chosen to achieve a balanced overall best-performance of the models.

	Learning rate	**Batch size**	**n. layers**	**n. epochs**	**Dropout**	**n. heads**	**Top-p threshold**	**Hidden size**	**Results**
LSTM path	0.0005	110	4	1000	0.2	-	0.90	128	Acc: 96.667% Cover: 68/86 = 0.8006
Transformer path	0.0005	110	6	500	0.1	8	0.95	128	Acc: 95.133% Cover: 0.953
LSTM time	0.0005	110	4	1000	0.2	-	-	64	Time traj:58.1% Time sector:86.0%
Transformer time	0.0005	110	6	1100	0.1	8	-	64	Time traj:64.6% Time sector:86.5%
Transformer path&time	0.0005	110	6	2000	0.2	4	0.95	128	Acc: 92.167% Cover: 86/86=1.0 Time traj: 58.3% Time sector: 72.9%

Table 5.1: Parameter models optimization

5.1.7 Hardware settings and GPU

Graphics Processing Units (GPUs) were born for graphics and images processing. Now they have also been used in deep learning for their ability to handle parallel computations well. This feature can make the training process of a deep learning model much faster by parallelizing matrices operation. Indeed, matrices results are combinations of independent computation, than can be possibly parallelized. A really wide range of operations in deep learning pass through matrices iteration from the forward and backpropagation pass that are the core of each architecture to attention-mechanism.

These experiments were carried out on a workstation equipped an NVIDIA GeForce RTX 3090 GPU well integrated thanks to the support of PyTorch library and CUDA.

5.1.8 Results

The experiments consisted of training the models, running them, obtaining evaluation metrics to assess their performance, and then comparing the results. The models were trained and optimized on the same dataset, which consisted of 10,000 European trajectories.

To add more information to the study, we also tried applying the same architecture to different data sources. In this case, we used another dataset containing the same

trajectory but in a different representation. The first dataset used is the FIR dataset (see chapter on data), and the second is the AUA dataset (see chapter on data). The format of the data is the same, but the trajectory is represented as a sequence of sectors, which differ between the two representations, as the airspace is 'divided' differently.

The models were optimized only on the first dataset, and then the same parameters were applied to the second dataset. The models were tested on the generated test set, and during the development process, its size was also adjusted, from several hundred trajectories to 10,000 trajectories. For the optimization process, based on previous empirical results, it was decided to evaluate the models on 3,000 trajectory generation datasets. The results metric is the average of 3 different generations.

	Best Feasibility		**Best Coverability**		**Best Balance**	
	Feasib.	**Coverab.**	**Feasib.**	**Coverab.**	**Feasib.**	**Coverab.**
FIR						
LSTM path on FIR	98.8%	75.5%	92.8%	82.5%	96.6%	80.0%
Transformer Path on FIR	99.9%	88.0%	90.1%	100%	97.1%	96.3%
Transformer Path&Time on FIR	99.2%	80.4%	92.1%	100%	96.3%	94.5%
AUA						
LSTM path on AUA	91.7%	69.6%	89.4%	73.2%	90.5%	70.3%
Transformer Path on AUA	99.3%	72.1%	91.3%	85.0%	95.9%	81.0%
Transformer Path&Time on AUA	97.6%	70.1%	93.2%	81.4%	95.5%	76.6%

Table 5.2: Path models comparison

This table shows the 3 models that generate sequences of sector modelled on FIR dataset and AUA dataset. For each of them there are 3 results the best Feasibility possibly get without considering the coverability, the best coverability measured without considering the feasibility and the overall performance considering a balanced result where Feasibility and coverability tend to be optimal at the same time.

	Time Trajectory	**Time Sector**
FIR		
LSTM time on FIR	58.1%	86.0%
Transformer time on FIR	**64.6**%	**86.5**%
Transformer Path&Time on FIR	58.3%	72.9%
AUA		
LSTM time on AUA	59.3%	**84.3**%
Transformer time on AUA	**62.1**%	83.0%
Transformer Path&Time on AUA	56.5%	70.4%

Table 5.3: Time models comparison

The table shows for each of the time models the two performances on FIR and AUA dataset. The Time Sector and Time Trajectory column correspond to the two metrics expressed previously in the chapter.

The results show that by adopting these methods it's possible to generate sets of feasible trajectories with some variance, measured with the coverability metrics, and consistent time. Transformers result in better performance than LSTM because its complexity can capture more and more complex patterns. Between the two types of architectures—one that concatenates the path and time into a single sequence and another that generates sector-time pairs at each step—both show similar performance, but the first performs slightly better. This is likely because, in the first approach, the model decides the time while already having the spatial information. In contrast, the second approach predicts the probability distribution for the next token and its corresponding time at each step, but the next token is selected randomly with the top-p method sampling afterward. This can result in a mismatch between the chosen time and the selected token, because, again, the time spent in the current sector is chosen without really knowing already the sector picked but just the tokens distributions.

For the time models it was possible to see that the two metrics were in a direct relationship compared to the performances, if one was increasing the other follow. It's possible to see that the percentage of time for the entire trajectory is way higher compared to the metrics that check the time of the single sector, that can mean that overall, the model does correct choices, sometimes mismatch the time for the sector compromising the metrics of the total trajectory time.

5.2 Model Conditioning

In machine learning the concept of 'conditioning' plays a big role giving models more flexibility and realism to the generated data. Conditioning consists of adding external information as new variables to the model building making it aware of this extra information. In this chapter we will analyze conditioning techniques applied to trajectory generation, in particular features that affect the time.

5.2.1 Experiment 1: weather conditioning

In this first experiment, was tried to inject weather conditions data to conditions the time generated from the models. The time spent in the last sector was conditioned based on the weather recorded at the landing airport. This is because there is a direct relationship between weather conditions and the delays an airplane may experience. Adverse weather conditions can lead to longer flight routes or more delicate and slower maneuvers, such as during landing, where specific conditions make the process more complex and, consequently, more time-consuming [45].

5.2.1.1 Data

For the weather conditions data, I did not have access to any proprietary dataset, so I had to rely on publicly available data online. I used Meteostat [46], a free and open-source weather data provider that offers historical meteorological information from various global weather stations. Meteostat provides access to variables such as temperature, wind speed, humidity, and atmospheric pressure, which are relevant for analyzing weather conditions affecting air traffic. Despite its advantages, using publicly available weather data comes with limitations, such as potential gaps in records, lower resolution compared to proprietary datasets, and variations in data collection methods between different weather stations. These factors were considered during the experimental setup. Meteostat requires to query information given point location or station ID but I am working on discrete trajectory where the data given aren't point but extended area. To first overcome this issue and give it a try I decided to make a preliminary data study allowing me the possibility to cross the Eurocontrol sector dataset with the Eurocontrol point dataset. In the latter for each trajectory there are registered a set of GPS points at a specific datetime. So, from now on I have the airport time-location. From that it was possible to query Meteostat in that point. The idea was firstly to look to see if there was a correlation between the time spent in the last sector and the weather conditions at the arrival airport that considerably would have affected the landing procedure. The free feature set provided by Meteostat are the following:

- Temp: Air Temperature
- Dwpt: Dew point

- Rhum: Relative humidity
- Prcp: Total precipitation
- Snow: Snow depth
- Wdir: Wind (From) direction
- Wspd: Average wind speed
- Wpgt: Wind peak gust
- Pres: Sea-level air pressure
- Tsun: Total sunshine duration
- coco: Weather condition code

Not all the data was available at the specific arrival airport at the specific time, so I had to make larger the time range by getting the closest point from 60 minutes in advance and later. The same with the point location, it was search for the closest point into 10 km diameter, by cutting the location point to 2 decimals.

Still some trajectory remained with Nan values and so just discarded. Then were plotted each time spent in a sector coming from a specific other sector with the feature in analysis.

5.2.1.2 Result and discussion

The preliminary data analysis did not reveal any correlations. The plots (see Appendix B) indicate complete randomness between the time spent in a sector and the specific weather conditions at the destination airport. The main reasons are:

1. Sector Size – Airspace sectors cover vast areas, ranging from 50,000 km2 to several hundred thousand or even a million km2. While weather conditions may impact only landing times so much information are lost due to the large spatial scale.
2. Weather Data Limitations – Obtaining precise weather data for a specific location and time was challenging. In many cases, it was necessary to get values by looking at nearby times and locations because the exact one where not available.
3. Granularity of the Problem – This analysis is too high-level for such a detailed problem. A more meaningful approach would involve studying entire trajectories, including specific waypoints, maneuvers, aircraft models, and vehicle characteristics.

A possible solution to try could be to create global data and weather information about the sector in the specific range of time. This solution requires a lot of time and resources so I moved into the next part of experiment to try to understand if the models built so far can genuinely be conditioned.

5.2.2 Experiment 2: artificial variable conditioning

To test this out I created an ad hoc variable with a correlation. The is with the time spent in the last sector of a trajectory.

The variable created is a Boolean based on the statement "the time of a last sector C that succeed a specific sector B is higher than the 80% of the times of the same configuration". If it is true, the variable associated with the trajectory is 1 otherwise it is 0.

So, what Is this variable? This variable doesn't stand for a real meaning, it's just a variable with a correlation with another one, the time. Because in this setting problem was difficult to find any correlation in data, make the models learn them and them verify any learning of them it was easier to prepare a set and then just go for it.

To make the variable more understandable it could be an example of a case where there is this variable that means 'extreme weather' in the sector and so, indeed, the time of travel is higher than usual. The architecture was adapted in a way that can get external input and then went into an internal embedding process with the 'standard' input. For each model the binary condition variable was concatenated with the input after their embedding first phase. The rest of the architecture remains the same as before.

In this experiment the FIR dataset was used.

5.2.2.1 Results

The model now incorporates the Boolean variable and to check if it really learned the correlation that has to be with the time spent in the last sector I performed 2 metrics:

1. Accuracy: measure the percentage of cases where the model correctly assigns a time over or under the treshold of 80% in corresponds to 1 or 0.
2. Mean Squared Error (MSE): measure how far the prediction times are from the treshold.

The Mean Squared Error (MSE) is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where:

- n is the total number of data points,
- y_i represents the actual values,
- $\hat{y}_i$ represents the predicted or reference values (e.g., the threshold),
- $(y_i - \hat{y}_i)^2$ is the squared error for each data point.

MSE measures how far the observed values are from the expected values. A lower MSE indicates a smaller error.

Result table:

Model	Accuracy	MSE
LSTM-cascade	61%	9.4
Transformer-cascade	**74**%	**4.5**
Transformer-stepByStep	71%	5.3

Table 5.4: Comparison of model on variable conditioning

The result shows the capacity of the models to learn patterns from the external input, making the conditioning possible. Also, in this case the transformer model has better metrics than the LSTM thanks to his ability to get more complex patterns. This suggests that in future work, when incorporating data that has correlations with the current dataset, conditioning could be an effective approach.

Chapter 6

Conclusions

This work has explored innovative methods for the generation of air trajectories through the application of deep learning models, with particular attention to LSTM and Transformer networks. One of the key aspects of the research has been the division of the problem into two main components:

1. The generation of the spatial sequence of the trajectories, treated as an NLP problem in which each air sector has been modeled as a token in a sentence.
2. The temporal assignment to each sector, addressed as a time series prediction in which the travel time of each segment has been predicted based on the characteristics of the route.

To face this challenge, two different architectures have been developed:

- A two-model architecture, in which the spatial sequence and times are generated separately.
- A unified architecture, in which a single model is responsible for the simultaneous generation of sectors and time.

From the experiment results it has shown that it is possible to generate feasible air trajectory with a good variance. The best model performance is the one that before generate the space and then assign the time for sectors and in particular the Transformer one, at higher computational costs, performed better then the same but LSTM architecture.

6.1 Main contributions

This research offers a significant contribution to the scientific community in several ways:

- Demonstration of the feasibility of an NLP approach for the generation of air trajectories.
- Comparative analysis between LSTM and Transformer , highlighting the advantages and disadvantages of each technology.

- Definition of metrics for the evaluation of the generated trajectories, with an approach that considers both spatial and temporal aspects.
- Demonstration of the models flexibility by testing them between different datasets (FIR and AUA), suggesting that the approach can be extended to other operational scenarios.
- Experimentation with conditioning using external variables.

These elements lay the foundations for future studies and possible applications in the aeronautical sector.

6.2 Limitations and Challenges

During the development of the project some difficulties were faced based on:

1. Representation of the trajectory: managing this discontinue type of trajectory was a easier way to look at the problem but at the same time it creates issue. Especially:
    - Lack of variability: the synthetic generator value is higher when they learn patters and their points of variance, to play with them and creates more 'version' of it. In this space representation, there is little variance.
    - evaluation metrics: because of this specific problem, it wasn't possible to use standardized metrics but it was necessary to define new ones.
2. Computational cost: It required significant amount of time and dedicated hardware to complete this experiment, especially all the optimization process.

6.3 Future Perspectives

This works want to be a lunch base for other studies and application. It was to lead making aware of which approaches and architectures can works better. Which parameters and resources are needed for the process. The following results could range between close to real trajectory generation for different application from route planning, to data used for traffic analysis or to feed any sort of model like emission reduction optimization.

6.4 Final Conclusion

This research has demonstrated that the use of deep learning techniques for air trajectory generation is a promising and flexible approach, capable of adapting to different operational conditions. The comparative analysis between different architectures has highlighted that dividing the problem into two subcomponents (space and time) allows for more accurate and adaptable results. The most innovative

aspect of this study is the general vision of the problem, which makes it a starting point for further research rather than a closed and definitive solution. The defined metrics and explored methodologies provide a solid foundation on which to build more advanced, more adaptable, and more integrable models into air traffic management systems. The future of this research could lead to increasingly sophisticated models, capable of generating optimized trajectories in real-time, with a significant impact on aeronautical operations and air traffic control

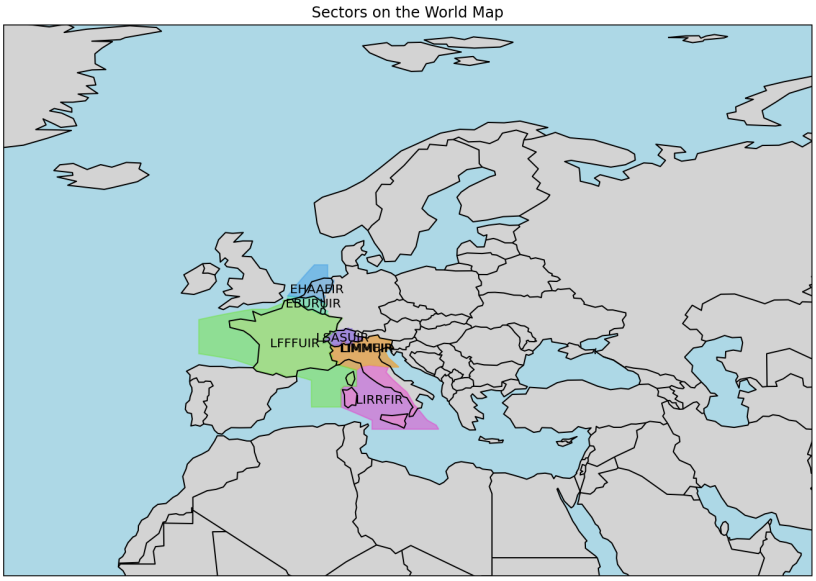
Appendix A

Appendix A

A.1 Plot generated trajectory

Following some plots from the different models trained on FIR dataset:

1. LSTM path + LSTM time
2. Transformer path + Transformer time
3. Transformer path&time



TAXI_OUT,0.0;LIMMFIR,1.79;LIRRFIR,0.613;LIMMFIR,10.9;LIMMUIR,2.12;LSASUIR,9.64;LFFFUIR,18.6;EBURUIR,4.92;EHAAFIR,22.1;TAXI_IN,0.0

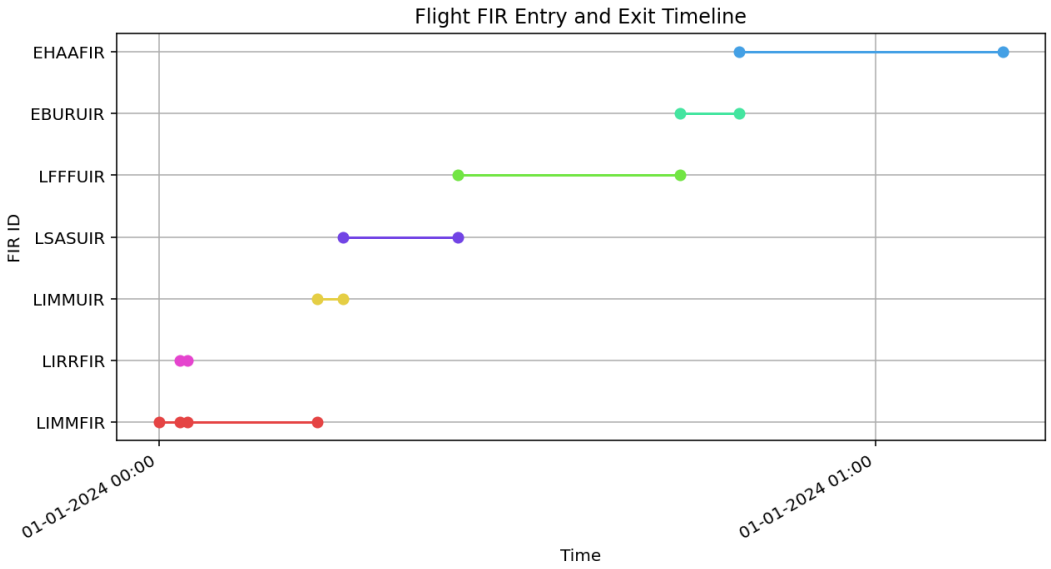
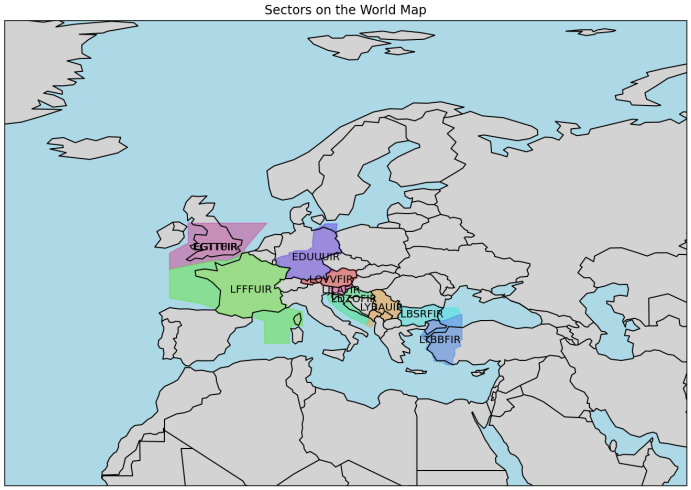


Figure A.1: LSTM path + LSTM time



TAXI_OUT,0.0;EGTTFIR,9.38;EGTTUIR,5.82;LFFFUIR,39.6;EDUUUIR,22.6;LOVVFIR,15.4;LJLAFIR,7.34;LDZOFIR,20.5;LYBAUIR,22.7;LBSRFIR,28.4;LTBBFIR,20.3;TAXI_IN,0.0

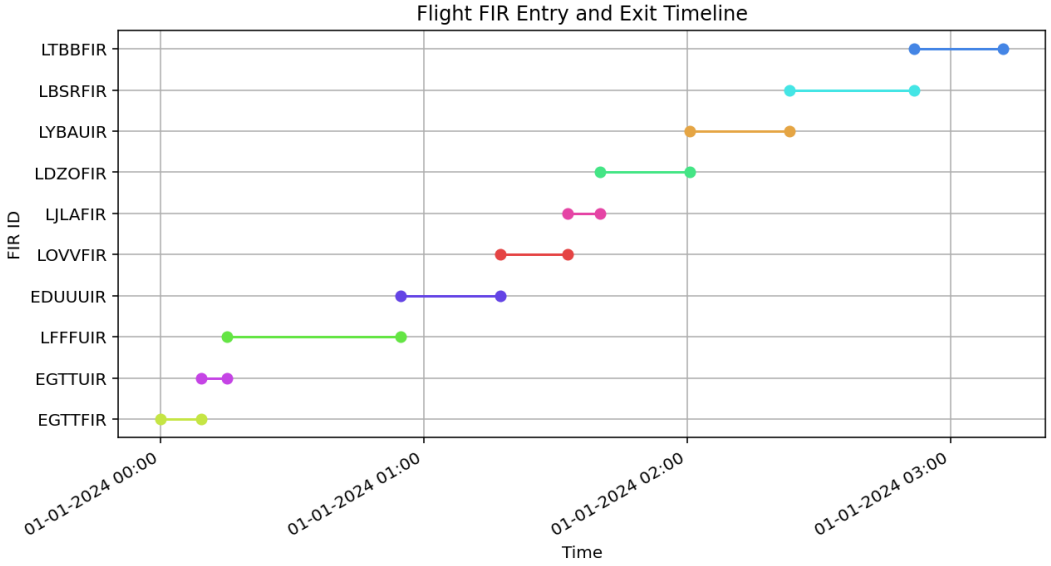
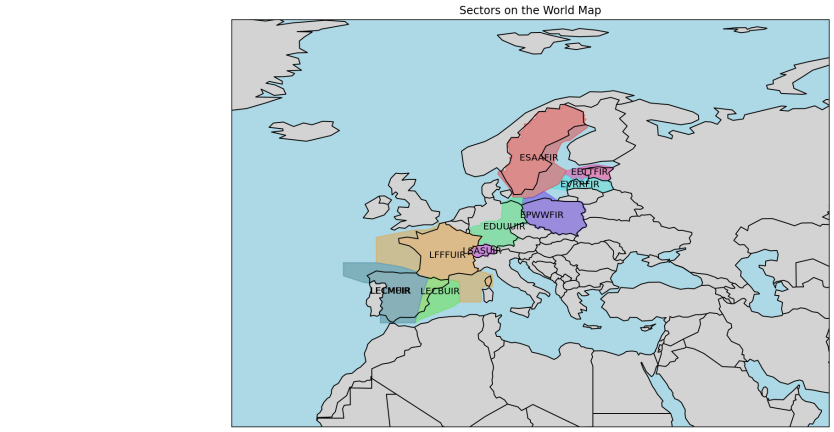


Figure A.2: Transformer path + Trasformer time



TAXI_OUT,0.0;EETTFIR,23.1;EVRRFIR,-0.524;ESAAFIR,22.5;EPWWFIR,25.4;EDUUUIR,50.8;LSASUIR,-0.115;LFFFUIR,33.8;LECBUIR,0.00221;LFFFUIR,0.1;LECMUIR,23.0;LECMFIR,19.6;TAXI_IN,0.0

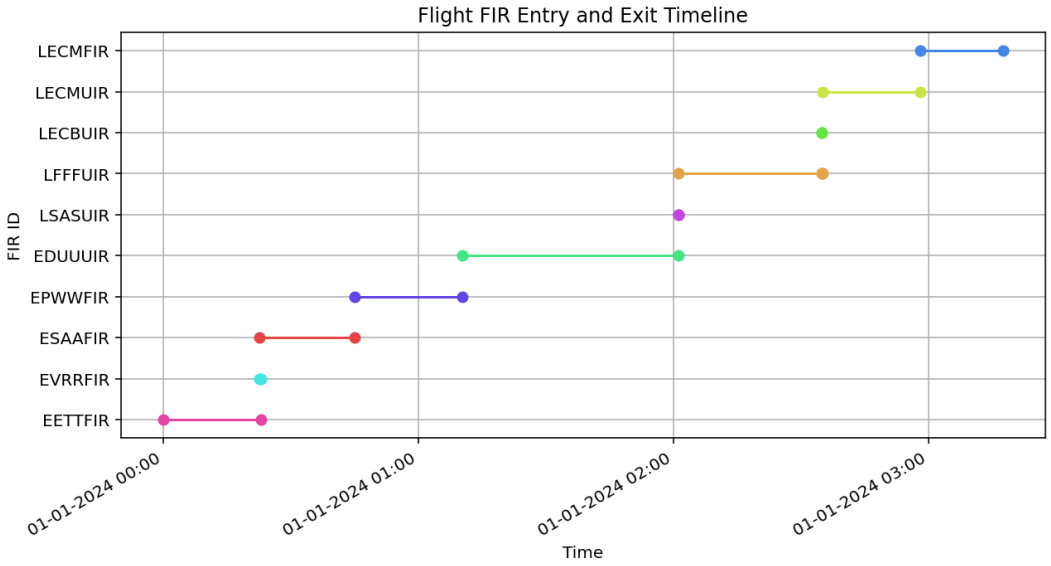


Figure A.3: Transformer path&Time

Appendix B

Appendix B

B.1 Weather correlation with time plots

As stated in Chapter 5, Section 5.2.1.2, no correlation was found between the time spent in the landing sector and the weather variable at that specific time. The plot suggests a lack of direct relationship. As shown in the following plots, the relationship is quite random; there is no clear, defined variation in the time based on changes in the weather variable considered.
On the x-axis we have the time spent while on the y-axis the value of the condition variables:

- red: wind speed
- green: snow
- yellow: precipitation
- blue: coco category

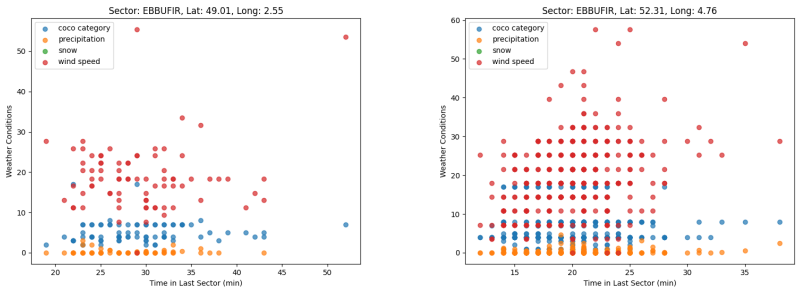


Figure B.1

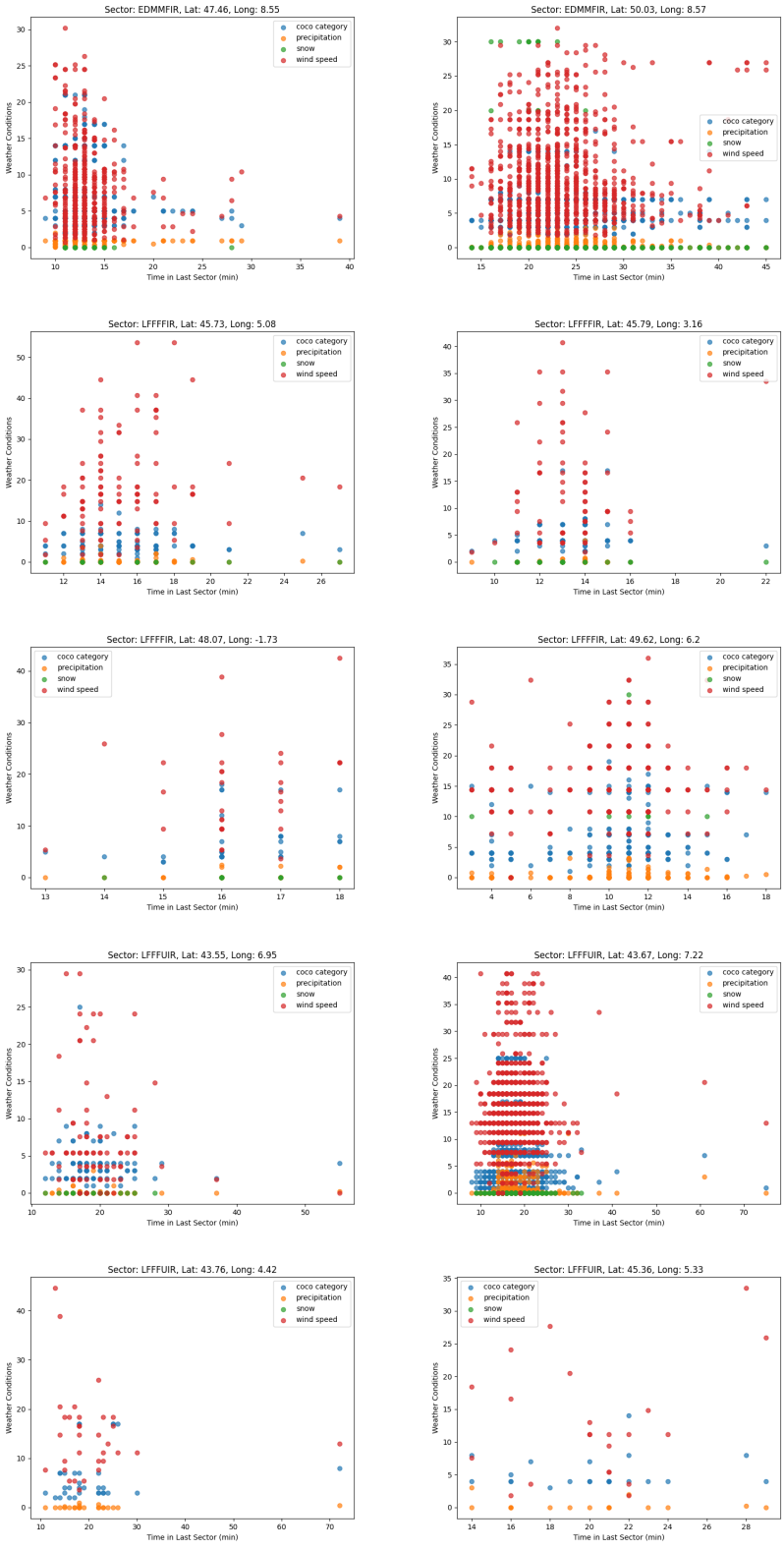

Sector: EDMMFIR, Lat: 47.46, Long: 8.55
Sector: EDMMFIR, Lat: 50.03, Long: 8.57
Sector: LFFFFIR, Lat: 45.73, Long: 5.08
Sector: LFFFFIR, Lat: 45.79, Long: 3.16
Sector: LFFFFIR, Lat: 48.07, Long: -1.73
Sector: LFFFFIR, Lat: 49.62, Long: 6.2
Sector: LFFFUIR, Lat: 43.55, Long: 6.95
Sector: LFFFUIR, Lat: 43.67, Long: 7.22
Sector: LFFFUIR, Lat: 43.76, Long: 4.42
Sector: LFFFUIR, Lat: 45.36, Long: 5.33
coco category
precipitation
snow
wind speed
Weather Conditions
Time in Last Sector (min)

Figure B.2

Bibliography

[1] Euclides Carlos Pinto Neto, Derick Moreira Baum, Jorge Rady de Almeida Jr, Joao Batista Camargo Jr, and Paulo Sergio Cugnasca. "Deep learning in air traffic management a survey on applications, opportunities, and open challenges". In: *Aerospace* 10.4 (2023), p. 358 (cit. on p. 1).

[2] swissinfo. *Mai così tanti passeggeri aerei in Europa*. it. Dec. 2019. URL: https://www.swissinfo.ch/ita/mai-cos%C3%AC-tanti-passeggeri-aerei-in-europa-43-rispetto-a-2010/45417432 (cit. on p. 1).

[3] Airbus. *Global Market Forecast 2024 Mapping Demand for the Future of Air Transport*. English. Accessed: 2025-03-03. May 2024. URL: https://www.airbus.com/en/products-services/commercial-aircraft/global-market-forecast (cit. on p. 1).

[4] SESAR Joint Undertaking. *MP 7 1 Expected Benefits and Impact*. Accessed: 2025-03-03. 2025. URL: https://www.sesarju.eu/node/4842 (cit. on pp. 1, 3).

[5] Xuhao Gui, Junfeng Zhang, Xinmin Tang, Daniel Delahaye, and Jie Bao. "A Novel Aircraft Trajectory Generation Method Embedded with Data Mining". In: *Aerospace* 11.8 (2024) (cit. on pp. 1, 17, 18).

[6] Zied Ben Bouallegue et al. "The rise of data-driven weather forecasting". In: *Bulletin of the American Meteorological Society* 105.6 (2024), E864–E883 (cit. on p. 1).

[7] Google DeepMind. *GenCast Predicts Weather and the Risks of Extreme Conditions with State-of-the-Art Accuracy*. Accessed: 1 March 2025. Feb. 2025. URL: https://deepmind.google/discover/blog/gencast-predicts-weather-and-the-risks-of-extreme-conditions-with-sota-accuracy/ (cit. on p. 1).

[8] AlphaFold. Accessed: 1 March 2025. URL: https://alphafold.ebi.ac.uk/ (cit. on p. 1).

[9] Kaustav Bera, Nathaniel Braman, Amit Gupta, Vamsidhar Velcheti, and Anant Madabhushi. "Predicting cancer outcomes with radiomics and artificial intelligence in radiology". In: *Nature reviews Clinical oncology* 19.2 (2022), pp. 132–146 (cit. on p. 1).

[10] Kit-Kay Mak, Yi-Hang Wong, and Mallikarjuna Rao Pichika. "Artificial intelligence in drug discovery and development". In: *Drug discovery and evaluation: safety and pharmacokinetic assays* (2024), pp. 1461–1498 (cit. on p. 1).

[11] Jiazhen Cai, Xuan Chu, Kun Xu, Hongbo Li, and Jing Wei. "Machine learning-driven new material discovery". In: *Nanoscale Advances* 2.8 (2020), pp. 3115–3130 (cit. on p. 1).

[12] Yanlei Yin, Lihua Wang, Dinh Thai Hoang, Wenbo Wang, and Dusit Niyato. "Sparse Attention-Driven Quality Prediction for Production Process Optimization in Digital Twins". In: *IEEE Internet of Things Journal* (2024) (cit. on p. 1).

[13] Yingzhou Lu, Minjie Shen, Huazheng Wang, Xiao Wang, Capucine van Rechem, Tianfan Fu, and Wenqi Wei. "Machine learning for synthetic data generation: a review". In: *arXiv preprint arXiv:2302.04062* (2023) (cit. on p. 2).

[14] Thomas Planès, Scott Delbecq, Valérie Pommier-Budinger, and Emmanuel Bénard. "Simulation and evaluation of sustainable climate trajectories for aviation". In: *Journal of Environmental Management* 295 (2021), p. 113079 (cit. on p. 3).

[15] Abolfazl Simorgh and Manuel Soler. "Climate-optimized flight planning can effectively reduce the environmental footprint of aviation in Europe at low operational costs". In: *Communications Earth & Environment* 6.1 (2025), p. 66 (cit. on p. 3).

[16] Yoav Goldberg. "A primer on neural network models for natural language processing". In: *Journal of Artificial Intelligence Research* 57 (2016), pp. 345–420 (cit. on p. 4).

[17] Thad Hughes and Keir Mierle. "Recurrent neural networks for voice activity detection". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 7378–7382 (cit. on p. 4).

[18] Feng Liu, Zhigang Chen, and Jie Wang. "Video image target monitoring based on RNN-LSTM". In: *Multimedia Tools and Applications* 78 (2019), pp. 4527–4544 (cit. on p. 4).

[19] colah. *Recurrent Neural Networks explanation*. English. Accessed: 2025-03-03. Aug. 2015. URL: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/` (cit. on pp. 4–7).

[20] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 4).

[21] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014) (cit. on p. 4).

[22] Alex Sherstinsky. “Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network”. In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306 (cit. on p. 4).

[23] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014) (cit. on p. 7).

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 8, 10, 11).

[25] Kenneth Ward Church. “Word2Vec”. In: *Natural Language Engineering* 23.1 (2017), pp. 155–162 (cit. on p. 10).

[26] Wei Chen et al. “Deep learning for trajectory data management and mining: A survey and beyond”. In: *arXiv preprint arXiv:2403.14151* (2024) (cit. on p. 14).

[27] Mehmet Ercan Nergiz, Maurizio Atzori, and Yucel Saygin. “Towards trajectory anonymization: a generalization-based approach”. In: *Proceedings of the SIGSPATIAL ACM GIS 2008 International Workshop on Security and Privacy in GIS and LBS*. 2008, pp. 52–61 (cit. on p. 15).

[28] Kun Ouyang, Reza Shokri, David S Rosenblum, and Wenzhuo Yang. “A non-parametric generative model for human trajectories.” In: *IJCAI*. Vol. 18. 2018, pp. 3812–3817 (cit. on p. 15).

[29] Xingrui Wang, Xinyu Liu, Ziteng Lu, and Hanfang Yang. “Large scale GPS trajectory generation using map based on two stage GAN”. In: *Journal of Data Science* 19.1 (2021), pp. 126–141 (cit. on p. 15).

[30] Nan Xu, Loc Trinh, Sirisha Rambhatla, Zhen Zeng, Jiahao Chen, Samuel Assefa, and Yan Liu. “Simulating continuous-time human mobility trajectories”. In: *Proc. 9th Int. Conf. Learn. Represent.* 2021, pp. 1–9 (cit. on p. 15).

[31] Yu Wang, Tongya Zheng, Yuxuan Liang, Shunyu Liu, and Mingli Song. “Cola: Cross-city mobility transformer for human trajectory simulation”. In: *Proceedings of the ACM Web Conference 2024*. 2024, pp. 3509–3520 (cit. on p. 15).

[32] Seongjin Choi, Jiwon Kim, and Hwasoo Yeo. “TrajGAIL: Generating urban vehicle trajectories using generative adversarial imitation learning”. In: *Transportation Research Part C: Emerging Technologies* 128 (2021), p. 103091 (cit. on p. 15).

[33] Yuan Yuan, Jingtao Ding, Huandong Wang, Depeng Jin, and Yong Li. “Activity trajectory generation via modeling spatiotemporal dynamics”. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2022, pp. 4752–4762 (cit. on p. 15).

[34] Yuanshao Zhu, Yongchao Ye, Shiyao Zhang, Xiangyu Zhao, and James Yu. "Difftraj: Generating gps trajectory with diffusion probabilistic model". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 65168–65188 (cit. on p. 15).

[35] Tonglong Wei, Youfang Lin, Shengnan Guo, Yan Lin, Yiheng Huang, Chenyang Xiang, Yuqing Bai, and Huaiyu Wan. "Diff-rntraj: A structure-aware diffusion model for road network-constrained trajectory generation". In: *IEEE Transactions on Knowledge and Data Engineering* (2024) (cit. on p. 15).

[36] Jie Feng, Yong Li, Chao Zhang, Funing Sun, Fanchao Meng, Ang Guo, and Depeng Jin. "Deepmove: Predicting human mobility with attentional recurrent networks". In: *Proceedings of the 2018 world wide web conference*. 2018, pp. 1459–1468 (cit. on p. 15).

[37] Qiang Gao, Fan Zhou, Goce Trajcevski, Kunpeng Zhang, Ting Zhong, and Fengli Zhang. "Predicting human mobility via variational attention". In: *The world wide web conference*. 2019, pp. 2750–2756 (cit. on p. 15).

[38] Dingqi Yang, Benjamin Fankhauser, Paolo Rosso, and Philippe Cudre-Mauroux. "Location prediction over sparse user mobility traces using rnns". In: *Proceedings of the twenty-ninth international joint conference on artificial intelligence*. 2020, pp. 2184–2190 (cit. on p. 15).

[39] Xuan Song, Hiroshi Kanasugi, and Ryosuke Shibasaki. "Deeptransport: Prediction and simulation of human mobility and transportation mode at a citywide level". In: *Proceedings of the twenty-fifth international joint conference on artificial intelligence*. 2016, pp. 2618–2624 (cit. on p. 15).

[40] Hao Xue, Flora Salim, Yongli Ren, and Nuria Oliver. "MobTCast". In: *Advances in Neural Information Processing Systems* 34 (2021) (cit. on p. 15).

[41] Yang Yang, Shengsheng Qian, Minghua Zhang, and Kaiquan Cai. "Sequence-to-sequence transfer transformer network for automatic flight plan generation". In: *IET Intelligent Transport Systems* 18.5 (2024), pp. 904–915 (cit. on pp. 16, 17).

[42] Zheng Zhang, Dongyue Guo, Shizhong Zhou, Jianwei Zhang, and Yi Lin. "Flight trajectory prediction enabled by time-frequency wavelet transform". In: *Nature Communications* 14.1 (2023), p. 5258 (cit. on p. 19).

[43] Timothé Krauth, Adrien Lafage, Jérôme Morio, Xavier Olive, and Manuel Waltert. "Deep generative modelling of aircraft trajectories in terminal maneuvering areas". In: *Machine Learning with Applications* 11 (2023), p. 100446 (cit. on pp. 19, 20).

[44] pytorch. *Python Library ML Developers*. English. Accessed: 2025-03-03. URL: `https://pytorch.org/` (cit. on p. 26).

[45] Chia-Wei Hsu, Chenyue Liu, Zhewei Liu, and Ali Mostafavi. "Unraveling extreme weather impacts on air transportation and passenger delays using location-based data". In: *Data Science for Transportation* 6.2 (2024), p. 9 (cit. on p. 37).

[46] meteostat. *Python Library Meteostat Developers*. English. Accessed: 2025-03-03. URL: `https://dev.meteostat.net/python/` (cit. on p. 37).