

Predicting Aircraft Trajectories

Recurrent Neural Networks for Time Series Forecasting

Lars Bogner

University of Oslo

<https://github.uio.no/larsbog/aiRtrafficNN>

(Dated: December 18, 2025)

Accurate prediction of aircraft trajectories is a key component of modern air traffic management, with direct implications for safety, efficiency, and airspace capacity. Traditional trajectory prediction methods are largely based on kinematic models and rule-based assumptions, which struggle to capture the complex, history-dependent dynamics observed in real flight operations. In this work, we investigate a data-driven approach to aircraft trajectory prediction using recurrent neural networks (RNNs) trained on large-scale Automatic Dependent Surveillance–Broadcast (ADS-B) data.

We formulate trajectory forecasting as a sequential learning problem and evaluate multiple recurrent architectures, including standard RNNs, Long Short-Term Memory (LSTM) networks, and Gated Recurrent Units (GRUs), in an offset many-to-many configuration. To ensure physically meaningful optimization, we introduce a geometrically motivated loss function based on the Haversine distance, augmented by a separately weighted altitude error term. Aircraft type and registration information are incorporated through a compact latent encoding learned via an autoencoder and combined with kinematic and temporal features.

Using a curated dataset of nearly 29 000 commercial flight trajectories over major Scandinavian routes, we perform an extensive hyperparameter study to assess the impact of architectural choices, temporal window sizes, and loss weighting. The results show that gated recurrent models significantly outperform vanilla RNNs, with LSTMs providing the most stable performance across configurations. Optimal predictions are obtained for initialization horizons of approximately 900 s and short prediction horizons of up to 150 s, achieving a minimum validation loss of 5.32 km. Notably, prediction accuracy degrades gradually for longer horizons, indicating robust learned representations of aircraft motion.

These findings demonstrate that recurrent neural networks, when combined with physically informed loss functions and appropriate preprocessing, offer a powerful and flexible framework for short- to medium-term aircraft trajectory prediction using publicly available surveillance data.

Contents		A. Conclusion	8
		B. Potential Future Work	9
I. Introduction	1	References	10
II. Methods	2	A. Hyperparameter Scan Results	10
A. Automatic Dependent Surveillance–Broadcast (ADS-B) Data	2		
1. Data Preprocessing	2		
B. Recurrent Neural Networks (RNNs)	3		
1. Long Short-Term Memory (LSTM)	3		
2. Gated Recurrent Units (GRU)	4		
C. Model Architecture	4		
1. Aircraft Encoding	4		
2. Trajectory Prediction Network	4		
3. Haversine Distance-Loss	5		
D. Tools and Usage of Artificial Intelligence	6		
1. Data Fetching and Parsing	6		
2. Model Implementation and Training	6		
3. Usage of Artificial Intelligence	6		
III. Results	6		
A. Dataset Overview	6		
B. Hyperparameter Optimization	7		
C. Window-Size dependent Accuracy	8		
IV. Conclusion	8		

I. INTRODUCTION

In airtraffic management, predicting the future trajectories of aircraft is crucial for ensuring safety, optimizing flight paths, and managing airspace efficiently. Current methods solely rely on radar data and basic kinematic models, which often fall short in capturing the complex dynamics of aircraft movements influenced by various factors such as weather conditions, air traffic, and pilot behavior. To address these challenges, we propose the use of Recurrent Neural Networks (RNNs), a powerful class of deep learning models designed for sequential data, to enhance the accuracy of aircraft trajectory predictions. RNNs are particularly well-suited for time series forecasting due to their ability to retain information from previous time steps, making them ideal for modeling the temporal dependencies inherent in flight data. By leveraging historical flight data, including position, velocity,

and altitude, our approach aims to provide more reliable and precise trajectory forecasts. This paper outlines the methodology for implementing RNNs in this context, discusses the data preprocessing steps, and evaluates the performance of the proposed model against traditional forecasting techniques.

Further uses of trajectory prediction include conflict detection and resolution, fuel efficiency optimization, and enhancing passenger experience through better predictions of arrival times. The integration of RNNs into air traffic management systems has the potential to revolutionize how we approach flight safety and efficiency in the aviation industry. Therefore, this field of research has seen a growing number of publications in recent years [1–3].

To optimize the performance of RNNs for trajectory prediction, we explore various architectures, including Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), which are known for their ability to mitigate the vanishing gradient problem commonly encountered in standard RNNs. We also investigate different input feature sets, and time window sizes to determine the most effective configuration for our specific application. In addition, we propose a geometric loss function that correctly captures the spatial distance between predicted and actual trajectories, irrespective of the position in the coordinate system.

We structure the paper as follows: In section II, we describe the dataset, preprocessing steps, and the architecture of the RNN model used for trajectory prediction. In section III, we present the results of our experiments, including performance metrics and extrapolations. Finally, in section IV, we summarize our findings and discuss potential future work in this area.

II. METHODS

A. Automatic Dependent Surveillance-Broadcast (ADS-B) Data

The dataset used in this study consists of ADS-B data collected from various aircraft over a specified period. ADS-B is a surveillance technology in which an aircraft determines its position via satellite navigation and periodically broadcasts it, enabling it to be tracked. The dataset includes information such as latitude, longitude, altitude, velocity, heading, and timestamp for each recorded position of the aircraft. These data points are transmitted at regular intervals, typically every second, on a frequency of 1090 MHz. Additionally, to the ground stations employed by air traffic control, ADS-B signals can also be received by other aircraft, enhancing situational awareness and safety. The data can also be received by hobbyists and researchers using relatively inexpensive equipment, contributing to a wealth of publicly available flight data for analysis and research purposes. Our work utilizes such publicly available ADS-B data,

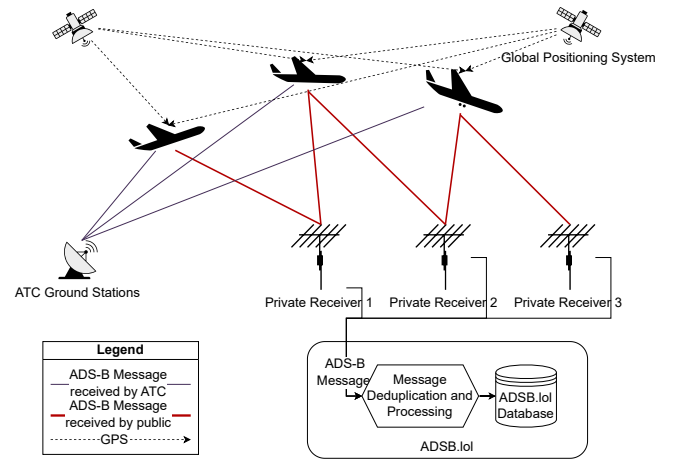


Figure 1: Working principle of ADS-B data transmission and reception.

provided by the ADSB.lol platform [4, 5].

An important consideration when working with ADS-B data is the potential for missing or erroneous data points due to signal loss, interference, or malicious activities. Especially the latter has raised concerns regarding the security and integrity of ADS-B data, prompting ongoing research into methods for detecting and mitigating such threats [6]. Such issues must be addressed before utilizing the data for trajectory prediction in safety-critical applications. For the purpose of this study, we assume that the dataset has been preprocessed to remove any erroneous data points and that it is representative of typical flight patterns. This is further aided by the fact, that we focus on commercial flights in airspace outside of current conflict zones.

The working principle of ADS-B relies on the aircraft’s onboard systems, which include a GPS receiver for accurate position determination and a transmitter for broadcasting the information. The data is formatted according to the ASTERIX standard, which defines the structure and content of the messages exchanged between aircraft and ground stations. This standardization ensures interoperability between different systems and facilitates the integration of ADS-B data into air traffic management systems. After receipt, the raw ADS-B data undergoes decoding and parsing to extract relevant information for further analysis. This process involves converting the binary messages into human-readable formats and organizing the data into structured databases and deduplicating entries from multiple receivers. This working principle is further illustrated in figure 1.

1. Data Preprocessing

Due to extremely large size of the raw ADS-B dataset, e.g. 1455 GiB for the year 2024, we limit our analysis to a subset of the data, specifically focusing on

flights within some major Scandinavian routes. We filter the data to include only trajectories, where the first and last recorded positions are within 20 km of the following airports: Oslo Gardermoen (ENGM), Stockholm Arlanda (ESSA), Bergen Flesland (ENBR), Stavanger Sola (ENZV), Umeå (ESNU), and Trondheim Værnes (ENVA). Additionally, the airports Gothenburg Landvetter (ESGG) and Tromsø Langnes (ENTC) are included. The altitude for the first and last recorded positions must be below 5000 ft to ensure that the data corresponds to takeoff and landing phases, respectively. To further refine the dataset, we filter on the aircraft type, including only commercial passenger jets such as the Airbus A320 family, Boeing 737 series, and Embraer E-Jets. This selection is based on the ICAO aircraft type designators found in the ADS-B data.

As neural networks allow for very good generalization given enough tunable parameters, this limit to specific routes and aircraft types is not expected to significantly impact the model's ability to learn trajectory patterns on larger scales. However, it does help to reduce the complexity of the dataset and allows for more focused evaluation of the model's performance on common commercial flight paths.

To prepare the data for training the RNN model, we perform several preprocessing steps. We normalize the latitude and longitude using min-max scaling to ensure that the input features are on a similar scale, which helps improve the convergence of the training process. We also convert the timestamps into relative time deltas to capture the temporal dynamics of the trajectories more effectively. Furthermore, we segment the trajectories into fixed-length sequences, where each sequence consists of a specified number of time steps. This segmentation allows the RNN to learn from shorter, manageable chunks of data while still capturing the overall trajectory patterns. Finally, we split the dataset into training, and test sets to evaluate the model's performance and generalization capabilities.

B. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a family of neural architectures explicitly designed for sequential and time-ordered data. Unlike feedforward networks, RNNs maintain an internal state that is propagated across time steps, enabling the model to represent temporal dependencies and history-dependent dynamics. Formally, given an input sequence $\{\mathbf{x}_t\}_{t=1}^T$, a generic RNN updates its hidden state according to

$$\mathbf{h}_t = \phi(\mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}), \quad (1)$$

where $\mathbf{h}_t$ denotes the hidden state at time t , $\phi(\cdot)$ is a nonlinear activation function, and $\mathbf{W}_x$, $\mathbf{W}_h$, and $\mathbf{b}$ are learnable parameters. Outputs $\mathbf{y}_t$ are typically obtained via

$$\mathbf{y}_t = g(\mathbf{W}_y \mathbf{h}_t + \mathbf{c}), \quad (2)$$

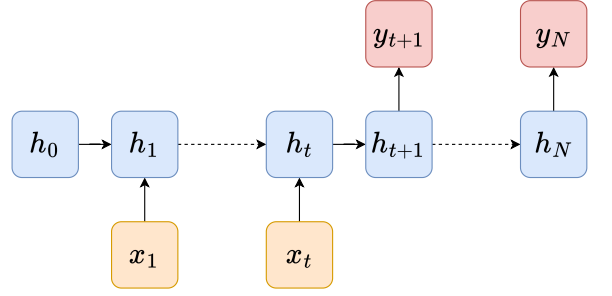


Figure 2: Generic Recurrent Neural Network (RNN) architecture in an offset many-to-many configuration. An initial conditioning phase updates the recurrent state without contributing to the loss, followed by a prediction phase in which outputs are produced and the loss is evaluated.

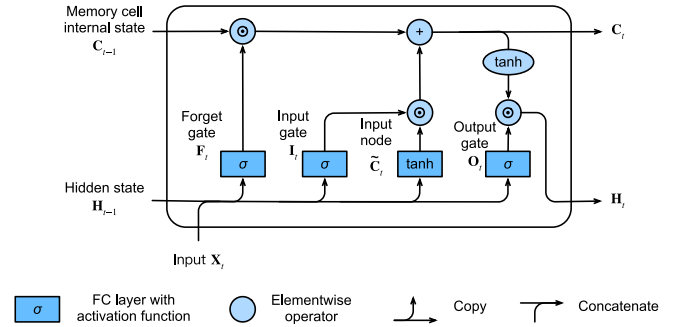


Figure 3: Long Short-Term Memory (LSTM) architecture [7].

with $g(\cdot)$ an output activation.

In practice, standard RNNs suffer from vanishing and exploding gradients when trained on long sequences. To address these limitations, gated architectures such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) introduce explicit mechanisms to regulate information flow over time. In this work, all recurrent models are employed in an *offset many-to-many* configuration: an initial conditioning phase updates the recurrent state without contributing to the loss, followed by a prediction phase in which outputs are produced and the loss is evaluated. A schematic overview of this setup is shown in figure 2.

1. Long Short-Term Memory (LSTM)

The LSTM augments the recurrent state with a persistent cell state $\mathbf{c}_t$, which acts as an explicit memory. Information flow into and out of this memory is controlled by three gates: the input gate $\mathbf{i}_t$, forget gate $\mathbf{f}_t$, and output gate $\mathbf{o}_t$. The LSTM update equations are

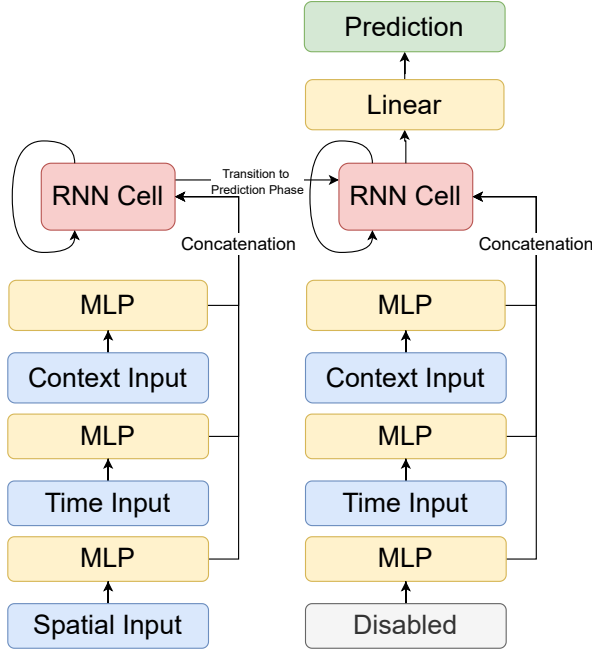


Figure 5: Recurrent Neural Network (RNN) architecture for aircraft trajectory prediction.

temporal features (e.g. time deltas), and contextual features (e.g. destination, aircraft encoding). All input features are first processed using separate Input Multilayer Perceptrons (I-MLPs) to transform them into a common feature space. Each I-MLP consists of a variable number of fully connected layers with GELU activation functions, allowing the model to learn complex feature representations. The outputs of the I-MLPs are then concatenated to form a unified input vector for the RNN layers. The RNN layers, which can be configured as either standard RNNs, LSTMs, or GRUs, process the sequential data and capture the temporal dynamics of the aircraft trajectories. The number of RNN layers and their hidden state sizes are hyperparameters that can be tuned based on the complexity of the dataset and the desired model capacity. Finally, the output from the last RNN layer is passed through a readout layer, which is a fully connected layer that maps the RNN’s hidden state to the predicted future positions of the aircraft. The readout layer produces outputs corresponding to the latitude, longitude, and altitude at each future time step in the prediction horizon.

While the contextual features are constant during a single trajectory, they are included at each time step to provide the RNN with relevant information throughout the sequence. This design choice allows the model to leverage contextual information effectively when making predictions, even though these features do not change over time. The temporal features are also present at each time step in the initialization phase and the predic-

tion phase, as they provide essential information about the timing of each recorded position and providing time stamps for the prediction outputs allows for easier evaluation against ground truth data. The spatial features, on the other hand, are only provided during the initialization phase, as they represent the historical positions of the aircraft that the model uses to inform its predictions.

3. Haversine Distance-Loss

To make the structure of the Haversine distance more explicit, we decompose the expression into a sequence of intermediate quantities that are subsequently combined into the final distance. We first define the differences in latitude and longitude,

$$\Delta\phi = \phi_2 - \phi_1, \quad \Delta\lambda = \lambda_2 - \lambda_1. \quad (13)$$

These angular separations enter the Haversine kernel through half-angle trigonometric terms. The central quantity of the formulation is the dimensionless Haversine argument

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right), \quad (14)$$

which can be interpreted as a measure of the squared chord length between the two points on the unit sphere. From this quantity, the corresponding central angle c between the points is obtained via

$$c = 2 \arcsin(\sqrt{a}). \quad (15)$$

Finally, the great-circle distance d follows by scaling the central angle with the Earth’s radius,

$$d = rc. \quad (16)$$

This decomposition highlights how latitudinal and longitudinal separations contribute separately to the overall distance, while preserving numerical stability for small angular differences. Such stability is particularly important in trajectory prediction settings, where successive waypoints are often closely spaced and accumulated errors in distance-based loss functions can otherwise bias model optimization over long flight paths. In our loss function, we compute the Haversine distance between the predicted positions and the ground truth positions at each time step in the prediction horizon. The overall loss is then defined as

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N d_i + \alpha \frac{1}{N} \sum_{i=1}^N (h_i^{\text{pred}} - h_i^{\text{true}})^2, \quad (17)$$

where N is the number of predicted time steps, d_i is the Haversine distance at time step i , h_i^{pred} and h_i^{true} are the predicted and true altitudes, respectively, and α is a weighting factor that balances the contribution

of altitude errors to the overall loss. This loss function effectively captures both horizontal and vertical discrepancies in the predicted trajectories, allowing the model to learn accurate representations of aircraft movements. We chose to deviate from the geometric distance, i.e., the Euclidean distance in three-dimensional space, as deviations in altitude are of different significance compared to horizontal deviations. For example, a 100m deviation in altitude is more critical than a 100m deviation in horizontal position when considering aircraft safety and trajectory accuracy. By separating the altitude error and applying a weighting factor, we can better tailor the loss function to reflect the practical importance of different types of errors in aircraft trajectory prediction.

The disadvantage of this approach is that for unphysical predictions, the Haversine distance can become ill-defined. For example, if the predicted latitude exceeds 90° or is less than -90° , the trigonometric functions in the Haversine formula may yield invalid results.

D. Tools and Usage of Artificial Intelligence

1. Data Fetching and Parsing

As to filter and parse the large amounts of raw ADS-B data, we utilize a combination Python scripts and a C++ parser. The Python scripts are responsible for downloading the raw data from the ADSB.lol archive on GitHub [4, 5] and dispatching the extraction and parsing tasks to the C++ parser. The C++ parser is optimized for performance and can efficiently handle the large volume of data, extracting relevant fields such as latitude, longitude, altitude, velocity, heading, and timestamp. In a first step, a basic filtering on the first and last recorded positions, as well as aircraft type, is performed to reduce the dataset size. The filtered data is then saved in a structured format in a CSV file, where each row corresponds to a single recorded position of an aircraft, along with its associated metadata. This structured dataset serves as the input for subsequent preprocessing and model training steps.

A second C++ program is used to filter the structured CSV data on the actual trajectories between airports as mentioned above. This program reads the CSV file, applies the trajectory filtering criteria, and outputs a new CSV file containing only the relevant trajectories. This two-step approach allows for efficient handling of the large ADS-B dataset while ensuring that only pertinent data is retained for model training.

2. Model Implementation and Training

The RNN model for aircraft trajectory prediction is implemented using the PyTorch deep learning framework [9], which provides a flexible and efficient platform for

building and training neural networks. The model architecture, including the input multilayer perceptrons (I-MLPs), RNN layers (LSTM or GRU), and readout layer, is defined as a custom PyTorch module. The training process involves defining a suitable optimizer, such as Adam or RMSprop, to update the model parameters based on the computed loss. The Haversine distance-loss function is implemented as a separate module to ensure modularity and ease of integration with the training loop. The training loop iterates over the training dataset, feeding batches of input sequences into the model, computing the loss, and performing backpropagation to update the model weights. To monitor the model's performance, we track metrics such as the average loss on after each epoch.

For preprocessing and postprocessing tasks, we utilize libraries such as NumPy [10] and Pandas [11] for efficient data manipulation and analysis. Matplotlib[12] is employed for visualizing the training progress and evaluating the model's predictions against ground truth trajectories. The entire implementation is structured in a modular fashion, allowing for easy experimentation with different model configurations, hyperparameters, and input feature sets.

All source code for data fetching, parsing, preprocessing, model implementation, training, and evaluation is made publicly available on GitHub at <https://github.uio.no/larsbog/aiRtrafficNN> to facilitate reproducibility and further research in the field of aircraft trajectory prediction using deep learning techniques.

3. Usage of Artificial Intelligence

For rapid prototyping and to overcome writer's block during the writing of this report, we made use of the AI language model ChatGPT-4 by OpenAI. The model was used to generate initial drafts of sections, suggest improvements in phrasing, and aid in the development of the data acquisition and preprocessing scripts. Care was taken to verify the accuracy and relevance of the generated content, ensuring that it aligned with the technical details and objectives of the study.

III. RESULTS

A. Dataset Overview

In a first study of the dataset, we analyze the distribution of flight trajectories, aircraft types, and registration countries. The dataset comprises a total of 28 796 flight trajectories, with a diverse range of aircraft types represented. In figure 6, we present a heatmap illustrating the density of flight trajectories across the selected Scandinavian routes.

We observe clear patterns in the flight paths, with higher densities along the straight routes between major airports. On the other hand issues due to missing data

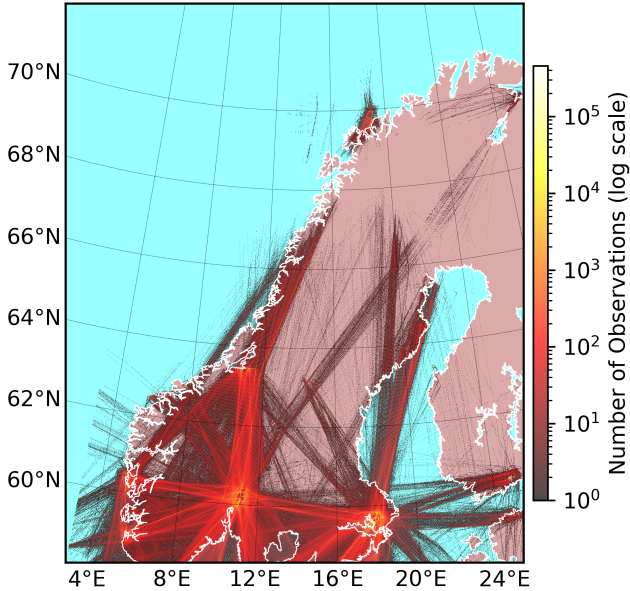


Figure 6: Heatmap of flight trajectory density across selected Scandinavian routes.

points and signal loss in the ADS-B data also become apparent, particularly in regions with low population density or challenging terrain. This behavior is mainly attributed to the community-based nature of ADS-B data collection, where reception quality can vary significantly based on the distribution of ground stations and environmental factors. While urban areas benefit from a higher density of receivers, rural and remote regions often suffer from sparse coverage, leading to gaps in the recorded trajectories. These data quality issues highlight the importance of robust preprocessing and filtering techniques to ensure the reliability of the dataset for trajectory prediction tasks. On the other hand we need to note, that low signal strength does not lead to erroneous data points, but rather to missing data points, as the positioning information is collected by the sender aircraft itself via GPS. As the data is transmitted digitally, interference or low signal strength will either lead to a complete loss of the message or a correct reception, but not to corrupted data.

Furthermore it becomes apparent that there is still a non-negligible spread in the actual flight paths taken by different aircraft on the same route. This variability can be attributed to factors such as weather conditions, air traffic control directives, and individual pilot preferences. Understanding and modeling this spread is crucial for developing accurate trajectory prediction models that can account for the inherent uncertainties in real-world flight operations. Figure 7 shows a zoomed-in view of the flight trajectories around Oslo Gardermoen Airport (ENGM), highlighting the complexity of approach and departure patterns in the vicinity of a major airport.

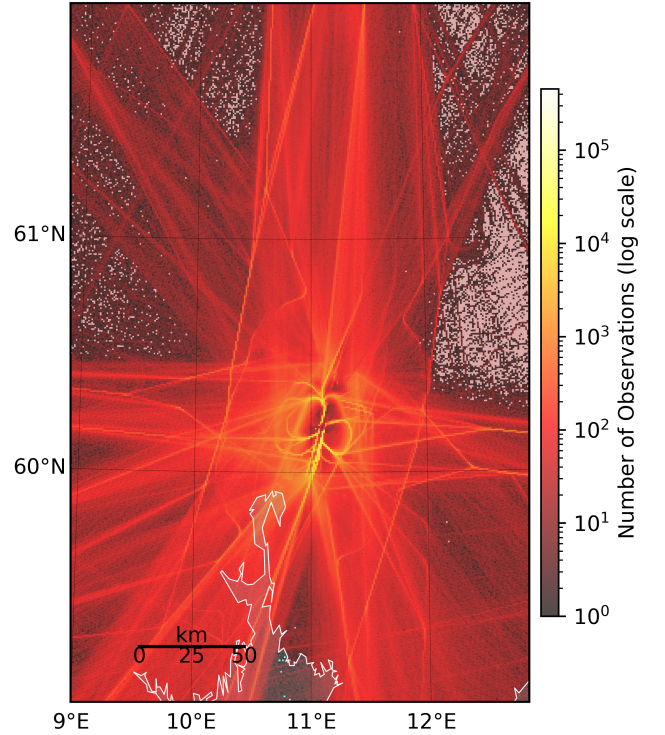


Figure 7: Zoomed-in heatmap of flight trajectory density around Oslo Gardermoen Airport (ENGM).

B. Hyperparameter Optimization

To evaluate the performance of different RNN architectures and hyperparameter configurations, we conduct a systematic hyperparameter optimization study. We explore various combinations of RNN types (LSTM and GRU), number of layers, hidden state sizes, RNN sizes and altitude loss weightings. The models are trained on the preprocessed dataset, and their performance is assessed using the Haversine distance-loss function described earlier. For each hyperparameter scan we define a grid of possible values and train models for each combination. The loss is then evaluated in pair wise comparisons with the definition

$$\mathcal{L}(A, B) = \min \mathcal{L}_{\text{validation}}(A, B, \theta) \forall \theta \in \Theta, \quad (18)$$

where A and B are two different hyperparameter configurations, $\mathcal{L}_{\text{validation}}$ is the validation loss after training, and Θ is the set of all other hyperparameter configurations. This approach allows us to identify the best-performing model configurations based on their relative performance across the validation set.

All results obtained in this section are featured in appendix A in the appendix. In the following, we summarize the key findings from the hyperparameter optimization study.

In a first run we compare different combinations of the following hyperparameters:

1. RNN Type: RNN, LSTM, GRU
2. Number of Nodes per Layer in the MLP: 16, 32, 64
3. Number of Nodes in the RNN: 32, 64, 128
4. Step Size between consecutive Data Points: 1, 5, 10, 30

The largest decline in accuracy is observed when using a simple RNN instead of the more advanced LSTM or GRU architectures. This behavior is expected, as standard RNNs are known to struggle with capturing long-term dependencies in sequential data due to vanishing gradients. Both LSTM and GRU models perform comparably well, with slight variations depending on the specific configuration of other hyperparameters. Increasing the number of nodes per layer in the RNN generally leads to improved performance, as it allows the model to learn more complex representations of the input data. However, this improvement comes at the cost of increased computational complexity and training time. When increasing the nodes in the MLPs, the converse effect is observed, as the increased complexity does not seem to add significant value to the model's ability to capture the relevant features from the input data. The step size between consecutive data points also plays a crucial role in model performance. Smaller step sizes (i.e., more frequent data points) provide the model with finer temporal resolution, which can enhance its ability to learn intricate trajectory patterns. However, this also increases the amount of data the model must process, potentially leading to vanishing influence of earlier time steps. The minimum validation loss is observed for a step size of 10, we choose the LSTM architecture with 64 nodes in the RNN and 16 nodes in the MLPs for further experiments.

In a second study, we investigate the impact of the number of hidden layers in the RNN and the I-MLPs, as well as a dropout probability to mitigate overfitting. The hidden layers in the I-MLPs are varied between 0 and 4 while the RNN hidden layers are varied between 1 and 5. The dropout rate is varied between 0% and 15% in steps of 5%. The results indicate that increasing the number of hidden layers does not significantly enhance model performance and may lead to vanishing gradients, particularly in deeper RNN configurations. The optimal configuration is found to be 0 hidden layers in the I-MLPs and 2 hidden layers in the RNN, with no dropout applied. We choose no dropout as the training and validation losses do not indicate overfitting in the current setup.

Lastly, we manipulate the weighting factor α in the Haversine distance-loss function to assess its influence on model performance. Simultaneously, we vary the initialization horizon and prediction horizon, in ranges of 300 to 1800 seconds and 150 to 900 seconds, respectively. The results show that a higher weighting on altitude errors of $1 \times 10^{-3} \text{ km ft}^{-2}$ leads to the best overall performance, as it encourages the model to prioritize accurate altitude

predictions alongside horizontal positioning. The optimal initialization horizon is found to be 900 s, providing sufficient context for the model to learn from past trajectory data. The best prediction horizon is determined to be 150 s, which is to be expected as the uncertainty in trajectory predictions generally increases with longer forecast periods. Most importantly it becomes apparent, that longer initialization horizons do not necessarily lead to better performance, as the additional information may not be relevant for the immediate future trajectory.

C. Window-Size dependent Accuracy

Instead of training separate models for different prediction horizons, we investigate the performance of a single model across varying initialization and prediction horizons. This approach allows us to assess the model's generalization capabilities and its ability to adapt to different temporal contexts. We train a single LSTM model with the previously identified optimal hyperparameters and evaluate its performance across a range of initialization horizons (from 300 s to 1800 s) and prediction horizons (from 50 s to 1800 s). As expected the best performance is observed around the previously identified optimal horizons of 900 s for initialization and 150 s for prediction. The model's accuracy decreases as the prediction horizon increases, reflecting the inherent uncertainty in forecasting longer-term trajectories. This decrease in accuracy is on the other hand quite moderate, indicating that the model is capable of maintaining reasonable performance even for extended prediction periods. The influence of the initialization horizon is more pronounced, with a sharp drop in accuracy for initializations for more than 1200 s. This behavior suggests that while additional historical context can be beneficial, there is a threshold beyond which the relevance of past data diminishes, potentially due to changes in flight dynamics or external factors not captured in the longer history. Overall, these results highlight the model's robustness and its potential applicability in real-world scenarios where varying temporal contexts are encountered. Especially the moderate decline in accuracy for longer prediction horizons is promising for practical applications in air traffic management and trajectory forecasting. The best obtainable validation loss for this study is 5.32 km for an initialization horizon of 900 s and a prediction horizon of 50 s.

IV. CONCLUSION

A. Conclusion

In this work, we investigated the use of recurrent neural networks for short- to medium-term aircraft trajectory prediction based on publicly available ADS-B data. By framing trajectory forecasting as a sequential learning problem and leveraging gated RNN architectures, we

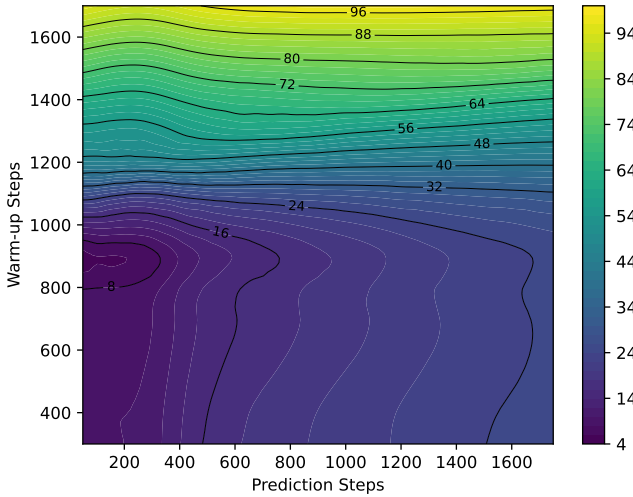


Figure 8: Contour plot of the validation loss as a function of initialization and prediction horizons for a single LSTM model.

demonstrated that data-driven models can capture non-trivial spatio-temporal structure in real-world flight trajectories beyond what is achievable with simple kinematic extrapolation.

A systematic comparison of architectures confirmed that gated models, specifically LSTMs and GRUs, substantially outperform vanilla RNNs, consistent with their ability to retain relevant temporal information over extended horizons. While both LSTM and GRU architectures exhibited comparable predictive performance, the LSTM showed slightly more stable behavior across hyperparameter settings and was therefore selected for further analysis. The hyperparameter studies further revealed that model capacity is primarily driven by the recurrent layers rather than the input MLPs, and that excessively deep recurrent stacks or long initialization windows can be detrimental due to diminishing relevance of historical context.

A key contribution of this study is the use of a geometrically motivated loss function based on the Haversine distance, augmented by a separately weighted altitude error term. This formulation respects the spherical geometry of the Earth while allowing altitude deviations to be penalized in a physically meaningful way. The results indicate that an appropriate balance between horizontal and vertical errors is crucial for stable training and improved predictive accuracy, particularly in terminal flight phases where altitude dynamics are most pronounced.

The trained model achieves its best performance for initialization horizons of approximately 900s and short prediction horizons of 50s to 150s, with a minimum validation loss of 5.32 km. Importantly, the degradation in accuracy for longer prediction horizons is gradual rather than catastrophic, suggesting that the learned representations capture robust aspects of aircraft motion. This robustness, combined with the model’s ability to general-

ize across varying window sizes without retraining, makes the approach well suited for practical air traffic management applications such as conflict detection, short-term flow prediction, and trajectory-based operations.

Several limitations remain. The study assumes clean and reliable ADS-B data and does not explicitly address missing data, adversarial manipulation, or uncertainty quantification in the predictions. Moreover, external factors such as weather, wind fields, and air traffic control interventions are only implicitly encoded through historical trajectories rather than being modeled explicitly. Incorporating such information, along with probabilistic or ensemble-based prediction methods, represents a natural direction for future work.

Overall, this work demonstrates that recurrent neural networks, combined with physically informed loss functions and careful preprocessing, provide a viable and flexible framework for aircraft trajectory prediction. As richer datasets and contextual information become available, such models have the potential to play a central role in next-generation, data-driven air traffic management systems.

B. Potential Future Work

While the results presented in this study demonstrate the viability of recurrent neural networks for aircraft trajectory prediction, several directions remain open for further improvement and extension.

A first and essential step is the expansion toward a more general and diverse dataset. The present analysis focuses on selected Scandinavian routes and a limited set of commercial aircraft types, which simplifies the learning problem but restricts generalization. Extending the dataset to include a wider range of airspaces, flight phases, aircraft categories, and traffic densities would allow for a more comprehensive assessment of model robustness and scalability. Such an extension would also enable the investigation of regional and procedural differences in air traffic operations.

Further gains are expected from more systematic feature engineering. While the current model primarily relies on kinematic and basic contextual inputs, additional temporal and operational features could provide valuable information. Examples include explicit encoding of time-of-day, day-of-week, seasonal effects, and airline-specific identifiers, which may capture operational preferences, scheduling constraints, or standardized procedures. These features could help the model distinguish between structurally different trajectory patterns that are otherwise similar in purely geometric terms.

Incorporating meteorological information represents another critical avenue for improvement. Weather effects such as wind speed and direction, temperature, pressure, and convective activity are among the dominant external drivers of trajectory deviations. Integrating gridded weather data or outputs from numerical weather predic-

tion models would allow the network to explicitly account for environmental forcing, thereby improving prediction accuracy, particularly for longer horizons and in adverse conditions.

The inclusion of information about nearby aircraft is also a promising direction. Aircraft trajectories are not independent, especially in dense airspace or during approach and departure phases, where separation constraints and air traffic control interventions play a major role. Modeling local traffic context—such as relative positions, velocities, and headings of surrounding aircraft—could enable the prediction model to capture interaction effects and collective traffic dynamics that are currently only indirectly reflected in the data.

From an architectural standpoint, attention-based models offer a compelling alternative or complement to recurrent networks. Attention mechanisms can enable the model to selectively focus on the most relevant parts

of the input history or contextual information, potentially alleviating the performance degradation observed for long initialization horizons. Hybrid architectures combining recurrent layers with temporal or spatial attention, or fully transformer-based approaches, may provide improved scalability and interpretability.

Finally, a stronger focus on data quality and robustness is essential for operational relevance. Although ADS-B data are generally reliable, issues such as missing messages, uneven spatial coverage, and intentional or unintentional data corruption remain significant challenges. Future work should investigate methods to explicitly handle such imperfections, including anomaly detection, confidence-aware training, data imputation strategies, and robustness-enhancing loss functions. Addressing these issues is a prerequisite for deploying learning-based trajectory prediction models in safety-critical air traffic management environments.

-
- [1] Z. Shi, M. Xu, and Q. Pan, IEEE Transactions on Intelligent Transportation Systems **22**, 7242 (2021), ISSN 1558-0016.
 - [2] Z. Shi, M. Xu, Q. Pan, B. Yan, and H. Zhang, in *2018 International Joint Conference on Neural Networks (IJCNN)* (2018), pp. 1–8, ISSN 2161-4407.
 - [3] K. Zhang and B. Chen, *Phased Flight Trajectory Prediction with Deep Learning* (2022), 2203.09033.
 - [4] *Adsblob/globe_history_2024: 2024 Historical data for all aircrafts traces known to adsb.lol. Openly licensed.*, https://github.com/adsblob/globe_history_2024.
 - [5] *Adsblob/globe_history_2025*, ADSB.lol (2025).
 - [6] M. Riahi Manesh and N. Kaabouch, International Journal of Critical Infrastructure Protection **19**, 16 (2017), ISSN 1874-5482.
 - [7] *10.1. Long Short-Term Memory (LSTM) — Dive into Deep Learning 1.0.3 documentation*, https://d2l.ai/chapter_recurrent-modern/lstm.html.
 - [8] *10.2. Gated Recurrent Units (GRU) — Dive into Deep Learning 1.0.3 documentation*, https://d2l.ai/chapter_recurrent-modern/gru.html.
 - [9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library* (2019), 1912.01703.
 - [10] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al., Nature **585**, 357 (2020).
 - [11] T. pandas development team, *Pandas-dev/pandas: Pandas*, Zenodo (2025).
 - [12] J. D. Hunter, Computing in Science & Engineering **9**, 90 (2007).

Appendix A: Hyperparameter Scan Results

This appendix presents the complete results of the hyperparameter optimization studies summarized in the main text. The figures referenced below provide a visual overview of validation performance across the explored configurations. All plots are based on validation losses obtained after training each model to convergence on the same preprocessed dataset, using the Haversine distance–loss function defined earlier.

Figure 9 shows the results of the initial hyperparameter scan comparing different RNN architectures and core model sizes. The figure summarizes pairwise validation-loss comparisons for simple RNN, LSTM, and GRU architectures, as well as different RNN hidden-state sizes, MLP widths, and temporal step sizes. Each data point corresponds to the minimum validation loss achieved for a given configuration, marginalized over all remaining hyperparameters according to the definition in equation (18). The plot was generated by evaluating the full hyperparameter grid and aggregating the best-performing validation losses for direct comparison.

The second study, visualized in figure 10, focuses on architectural depth and regularization effects. Validation losses are shown for varying numbers of hidden layers in the RNN and the I-MLPs, as well as different dropout probabilities. This plot was produced by fixing the best-performing configuration from the first scan and systematically varying only depth and dropout. The reported losses correspond to the minimum validation loss observed for each setting, enabling a direct assessment of stability and diminishing returns in deeper architectures.

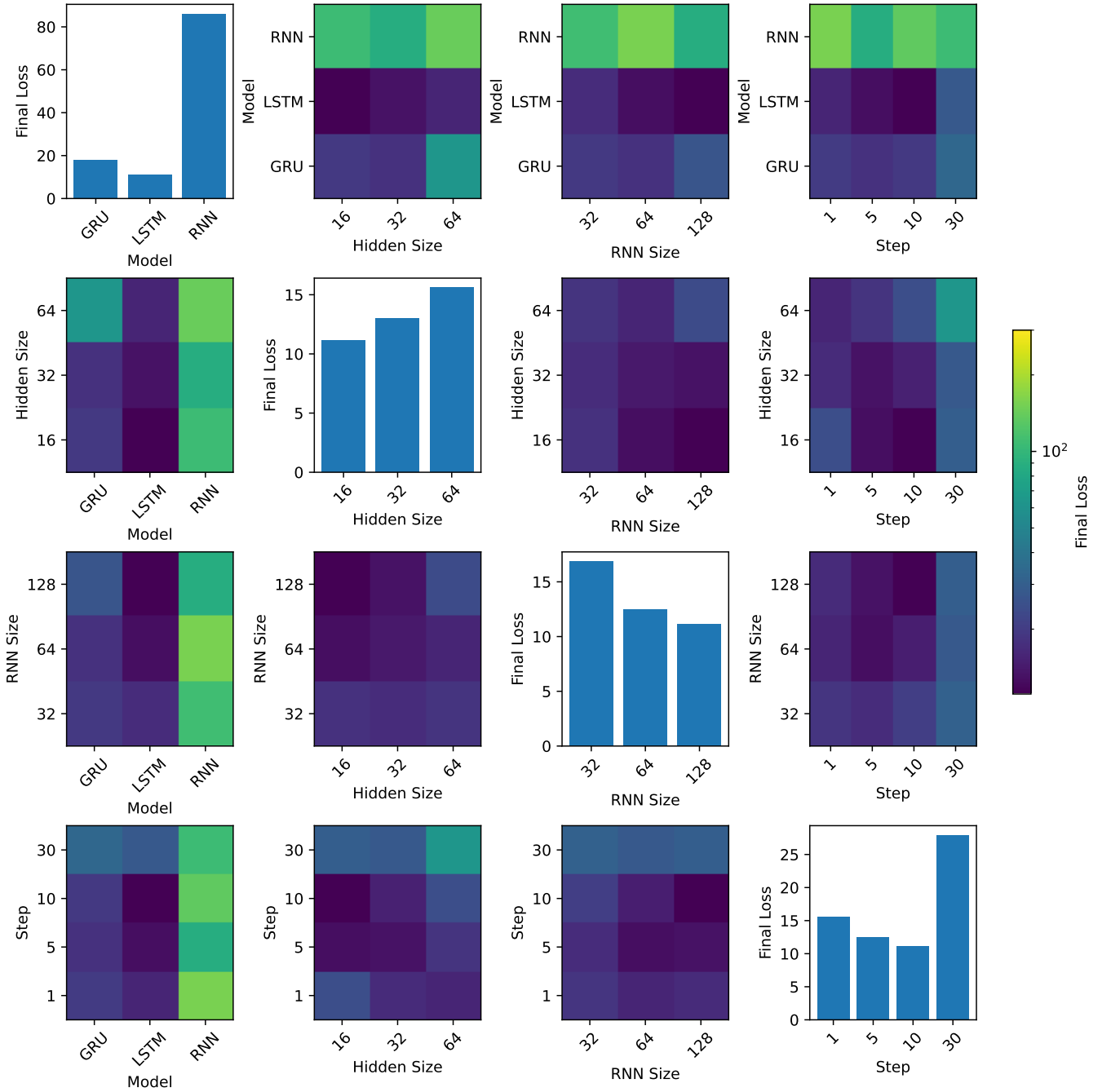


Figure 9: Initial hyperparameter scan comparing RNN type, RNN size, MLP width, and temporal step size.

Finally, figure 11 presents the results of the scan investigating the influence of the altitude-loss weighting factor, initialization horizon, and prediction horizon. The figure displays the minimum validation loss obtained across all tested combinations for each parameter value. It was generated by jointly varying the loss weighting and temporal horizons and selecting the best-performing configuration for each parameter according to the pairwise comparison criterion.

Taken together, figures 9 to 11 provide the empirical basis for the hyperparameter choices adopted in the final model configuration used throughout this work.

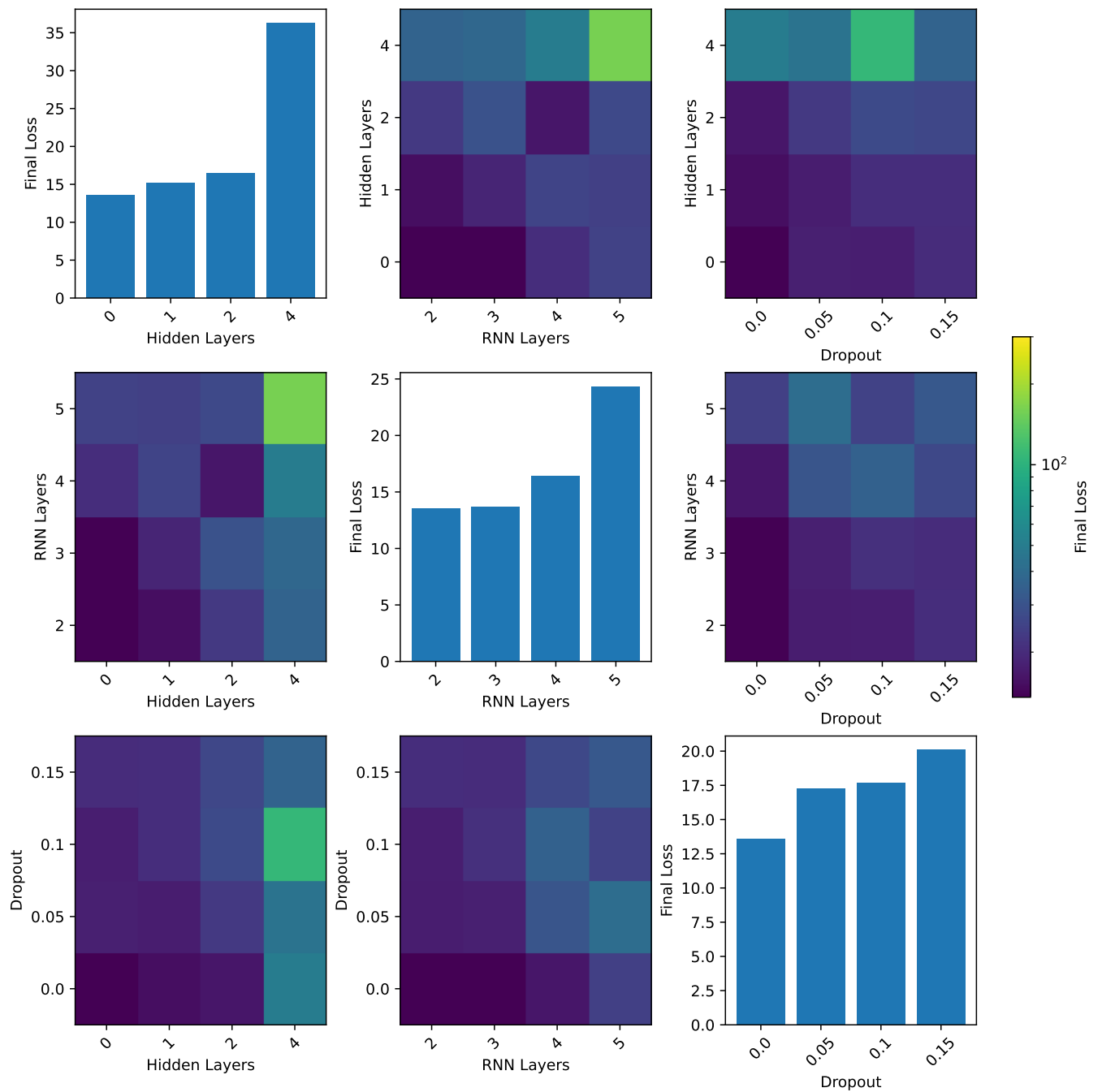


Figure 10: Hyperparameter scan over RNN and I-MLP depth and dropout probability.

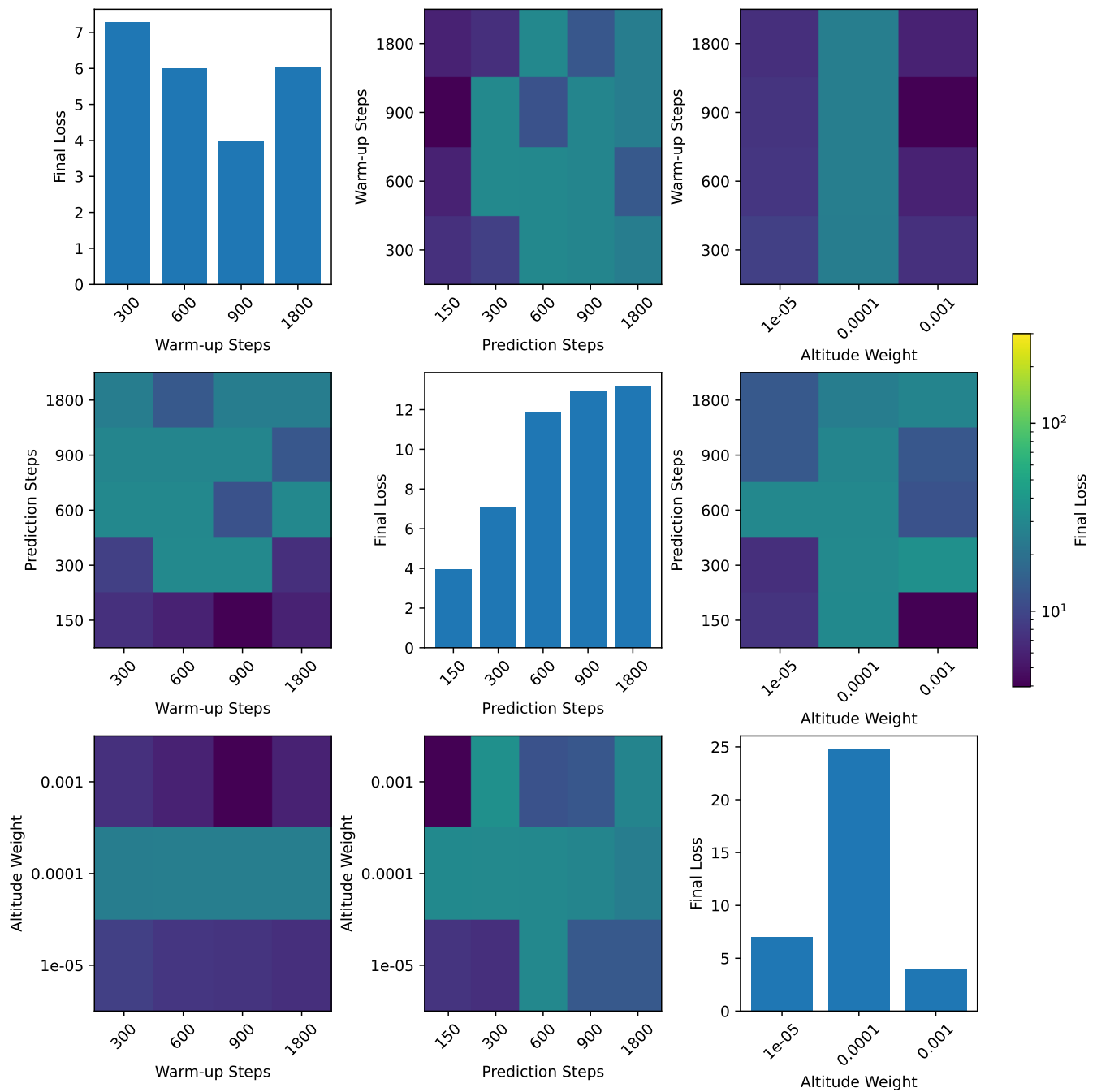


Figure 11: Validation-loss minima as a function of altitude-loss weighting, initialization horizon, and prediction horizon.