8-2021

Flight trajectory prediction for aeronautical communications.

Nathan T Schimpf
University of Louisville

FLIGHT TRAJECTORY PREDICTION FOR AERONAUTICAL COMMUNICATIONS

By

Nathan Schimpf
B.S. Electrical Engineering, University of Louisville, 2020

A Thesis
Submitted to the Faculty of the
University of Louisville
J.B. Speed School of Engineering
as Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Department of Electrical and Computer Engineering

July, 2021

FLIGHT TRAJECTORY PREDICTION FOR AERONAUTICAL COMMUNICATIONS

Submitted by: _____
Nathan Tyler Schimpf

A Thesis Approved On

_____
July 12th, 2021

by the Following Reading and Examination Committee:

_____
Hongxiang Li, Thesis Director

_____
Adel S. Elmaghraby

_____
Andre J. Faul

_____
Jacek M. Zurada

DEDICATION

To Etana, the life we build, and celebrations along the way.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Hongxiang Li, whose support and mentorship throughout my Master degree has been invaluable to me as a student and budding researcher. I would also like to recognize the assistance of the members of my thesis committee, Dr's. Adel Elmaghraby, Andre Faul, and Jacek Zurada; their support and feedback throughout this research meant a lot to me, and each of their courses built on that meaning throughout my Master's degree.

A special thanks is owed to Rafael Apaza, Eric Knoblock, Ruixuan Han, and Zhe Wang for sharing their experiences and help as I've been involved in this research and the larger scope of this project.

I would also like to thank NASA Glenn Research Center (specifically Rafael Apaza and Eric Knoblock) for making this research possible. Additionally, I am grateful to Kim Calden and Margo Pawlak at MIT Lincoln Laboratory for helping with the access of assorted data used in this project, as well as the University of Louisville and the Electrical and Computer Engineering department for providing the financial support for me pursue this work and continue as a student.

Finally, I would like to offer some personal thanks. To my partner, Etana, for her love, kindness, and patience. To my parents, for their love and support. And to my friends, particularly August, Brian, Gloria, and Kaleb, for the joy and kindness throughout the year.

Abstract

The development of future technologies for the National Airspace System (NAS) will be reliant on a new communications infrastructure capable of managing a limited spectrum among aircraft and ground systems. Emerging approaches to this spectrum allocation task mostly consider machine learning techniques reliant on aircraft and Air Traffic Control (ATC) sector data. Much of this data, however, is not directly available. This thesis considers the development of two such data products: the 4D trajectory (latitude, longitude, altitude, and time) of aircraft, and the anticipated airspace utilization and communication demand within an ATC sector. Data predictions are treated as a time series forecast challenge and addressed via the development of deep learning models with some form of recurrence. For each data product, relevant datasets are explored and an architecture search is conducted to identify and optimize a deep learning model. To this end, current efforts have primarily addressed trajectory prediction. Flight and weather data for the 4D trajectory prediction have been identified and preprocessed; initial comparisons of weather data have been used to hypothesize useful combinations; and initial model architectures have been identified for comparative development. Future work seeks to finalize training efforts toward trajectory prediction and address the task of airspace demand prediction.

Contents

List of Tables

List of Figures

I INTRODUCTION

With the proliferation of autonomous and unmanned aircraft, the Federal Aviation Administration (FAA) anticipates overhauling the National Airspace System (NAS). Because the NAS has a limited availability of radio spectrum that is predominantly allocated to static voice channels, a key aspect of redesigning the NAS includes the dynamic spectrum allocation for aeronautical communications, which the National Aeronautics and Space Administration (NASA) is currently investigating. In particular, authors in [4] describe an approach rooted in machine learning, reliant on data such as the anticipated location of aircraft and consequent communications demand on Air Traffic Control (ATC) sectors. While research has been previously conducted toward the development of these data products, the efforts have been preliminary.

In current research, development of an accurate 4D en-route trajectory prediction is considered a cornerstone for providing data items to these spectrum allocation frameworks. A sufficiently accurate trajectory prediction can serve to identify the sectors an aircraft will occupy, potentially indicating the available communication resources for each flight segment. The location of the aircraft can furthermore be used to estimate path loss and other channel quality metrics, providing insight for power budgeting and channel allocation engines. Finally, given a set of rules which define the occurrence of communications, spectrum demand can be inferred by comparing an accurate trajectory prediction against sector boundaries and the last-filed flight plan of the aircraft.

This thesis aims to develop such a trajectory prediction, by investigating deep learning mechanisms and available data. The structure of this thesis is as follows: Section II formulates the tasks of predicting each data product, identifies relevant datasets and deep learning frameworks, and discusses prior research on each task; Section III describes the computing resources, software setup, and methods of data acquisition for the tasks; Section IV describes the conducted efforts, including preprocessing, failed initial attempts, and conducted experiments; Section V discusses future efforts that could be conducted to continue the refinement and application of this research; finally, Section VI reiterates the scope, status, and conclusions of this thesis.

II BACKGROUND

This section discusses the tasks of trajectory and airspace demand prediction in greater detail. Both tasks are formulated, and a combination of data products and deep learning frameworks and mechanisms are introduced. Finally, a review of existing literature on both topics are discussed.

1 Problem Formulation

In this research, aircraft trajectory is a target output, which may support the estimation of communications within an ATC sector and aircraft channel qualities. Trajectory predictions are expected as time-varying products; however, the paradigm to accomplish this varies in research.

In the trajectory prediction scenario, each aircraft files a flight plan indicating the general path and cruising altitude to reach its destination; however, this plan is frequently modified during flight due to convective weather in the airspace. Trajectory prediction aims to train a model on these available flight plans and adverse weather data to estimate the flight's actual 4D coordinates. The generation of a predicted trajectory is accomplished in either a single-point forecast, multi-point forecast, or sequence-to-sequence paradigm. In single- and multi-point forecasts, prior aircraft position information is known and supplements flight plan and weather data to predict the next position or sequence of next positions. In a sequence-to-sequence paradigm, only the complete flight plan and weather data are used to create a complete trajectory prediction. For this research, trajectory prediction is developed under a sequence-to-sequence paradigm.

2 Data Products

Within the continental United States, flight data are collected via the FAA Air Route Traffic Control Center (ARTCC), while researchers typically access these data in aggregated locations such as the NASA Sherlock Data Warehouse [5]. Databases include flight plan and flight track information; while the flight track provides complete 4D information, flight plan messages only contain a cruising altitude and string of waypoints guiding the aircraft en-route. Flight plan messages are only provided as communications occur to modify these items. As a result, the last-filed flight plan (prior to departure)

must be interpreted by selecting initial messages and querying databases such as OpenNav [1].

A handful of weather databases are considered in existing literature to predict flight deviations. Each database provides regular updates to current and forecasted measurements of assorted products; Table 1 summarizes these databases. When collecting weather data for model training, a set of constraints are given: the weather products must be gridded, as this provides sufficient data for identifying trends related to convective weather; the weather products are also collected at their current measurement times (not forecast) to simplify data preprocessing and minimize potential forecast inaccuracies during model training. Finally, data products must be frequently updated to provide relevant forecasts. For this study, weather data from Massachusetts Institute of Technology Lincoln Labs' Corridor Integrated Weather Services (CIWS) and the National Oceanic and Atmospheric Administration (NOAA) High Resolution Rapid Refresh (HRRR) are considered, while NOAA North American Mesoscale products are neglected due to their update frequency.

Table 1: Summary of Weather Datasets

Weather Database	Used in	Relevant Variables	Update Period	Resolution
Corridor Integrated Weather Service (CIWS)	[2]	Vertically Integrated Liquid (VIL) Echo Top	Current 2.5 Min Forecast 5 Min	1.85 km (1 nmi)
North American Mesoscale (NAM)	[6]	Humidity Temperature Wind Speed (U) Wind Speed (V)	6 Hours	12 km (6.48 nmi)
Rapid Refresh (RAP) High Resolution Rapid Refresh (HRRR)	[7]	Humidity Temperature Wind Speed (U) Wind Speed (V)	1 Hour	RAP 13 km (7.01 nmi) HRRR 3 km (1.61 nmi)

CIWS provides a limited number of products designed in partnership with the FAA to support air traffic management [8]. Of particular interest are Echo Top and Vertically Integrated Liquid (VIL), radar measurements indicating the cloud height and total precipitation at all atmospheric levels, respectively. Neither measurement varies with altitude, though both strongly correlate to the presence of convective weather; while Echo Top is indicative of a lowest safe altitude for flight, VIL represents the amount and severity of precipitation in an area.

NOAA Rapid Refresh and HRRR collect over 14 general atmospheric measurements. These datasets are considered due to their public accessibility and providing the fastest update period of NOAA datasets. Most measurements in the datasets vary by latitude, longitude, and atmospheric pressure levels. For the purposes of this study, measurements of atmospheric temperature and Westerly (U) and Southerly (V) Wind Components are considered from the dataset.

3 Deep Learning Frameworks

Due to the variability and uncertainty of data present in both prediction tasks, machine learning frequently serves as a framework for research. This section will discuss the dominant frameworks used in research – Hidden Markov Models, Artificial Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks, and Self-Attention. Other statistical, adaptive, or machine learning techniques may appear in research and will be discussed as they appear.

A Hidden Markov Model

Prior to 2018, the Hidden Markov Model (HMM) was predominant in modelling forecast and regression techniques. Though the HMM was originally applied to language modelling, it is used in a variety of fields that use sequential data, including DNA modeling and flight prediction [9]. The HMM operates on a principle of stochastic chains, referred to as Markov chains. In an HMM, there are a set of potential states - of which a sequence may be given, as well as known probabilities to transition between each state. An HMM is used to predict a future state by computing the total probability of events leading up to each new potential state; effectively, this becomes a challenge of computing the maximum likelihood out of all possible sequences of events.

B Artificial Neural Network

The resurgence of deep learning has largely been characterized by the use of Neural Networks. Artificial Neural Networks (ANNs) were largely defined by 1995 but have become popular in the past 10 years due to advances in computing hardware [10]. Typical applications address classification and pattern recognition, including image processing and data categorization. In an ANN, individual neurons are interconnected in layers. Each neuron is trained to provide an output based on a weighted summation of all outputs from the previous layer; this summation is transformed via a nonlinear function referred to as an activation function. Typical ANN implementations are under a paradigm of supervised learning, where expected outputs are compared against computed outputs via a loss function; the reported error is used to update network weights via an optimizer.

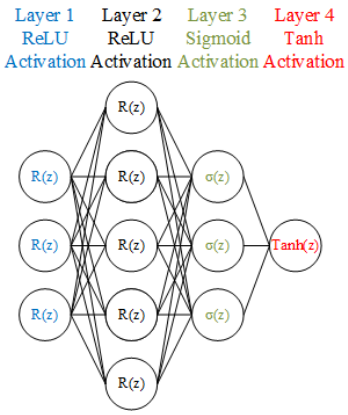

Figure 1: Illustration of an ANN

C Convolutional Neural Network

Convolutional Neural Networks (CNNs) present a model where data are operated on by a sweeping elementwise multiplication of an input matrix with a series of filter matrices (kernels) [10]. This operation is primarily modified by setting the size of these kernels, the rate that the kernels move over the input matrix (stride), and the padding of the input matrix. This convolution operation is frequently paired with pooling layers, which perform an operation (maximum, minimum, average, etc.) over fixed subsets to reduce data size. Typically, CNNs provide a method of abstraction and hierarchical behavior to larger deep learning frameworks. Most often, CNNs are used in computer vision tasks.

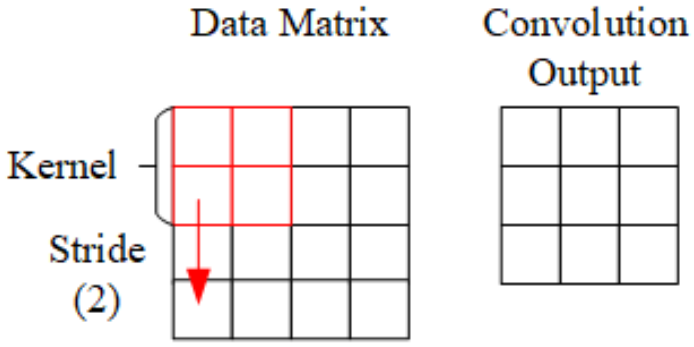

Figure 2: Convolution Dimensions

D Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a specific form of ANN, where prior predictions are fed back into the network alongside new information. In principle, this enables RNNs to retain a history of information and make predictions on sequences of data. However, RNNs suffer from challenges in error backpropogation known as vanishing and exploding gradients. During backpropogation, if a neuron is penalized such that its weights are close to zero (or greater than one), calculating the exponential impact of those weights on retaining sequence information decays (or grows) rapidly [10]. This is predominantly due to the shared weights within RNN cells and the nature of chain-differentiation when computing the gradient. Consequently, this problem is exacerbated by longer sequence lengths. Because of this problem, variants of the RNN are more commonly used in research.

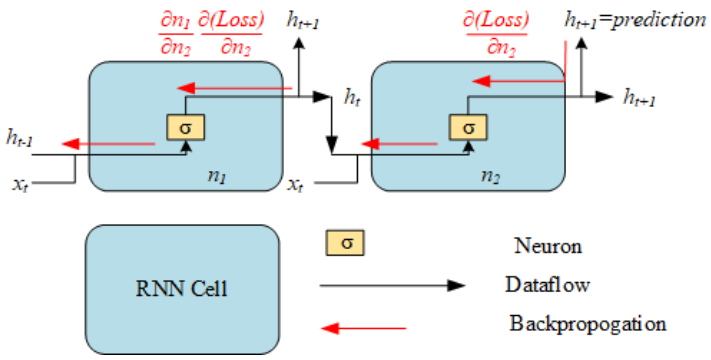

Figure 3: Vanishing Gradient in an Unrolled RNN

A popular alternative to traditional RNNs are gated variants such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU). These solutions address the vanishing gradient challenge by developing more complex cells in place of a simple neuron with associated recurrent state [10]. For example, LSTM cells implement two recurrent states: a cell state which directly stores and feeds forward historical information, and a hidden state which operates on prior and new information to generate outputs corresponding to the point in a sequence. For both new inputs and historic retention, the LSTM cells train neurons to regulate the relevance of data before their use in computing cell and hidden states. As a result, LSTM is able to operate on longer sequences of data with significantly greater effectiveness.

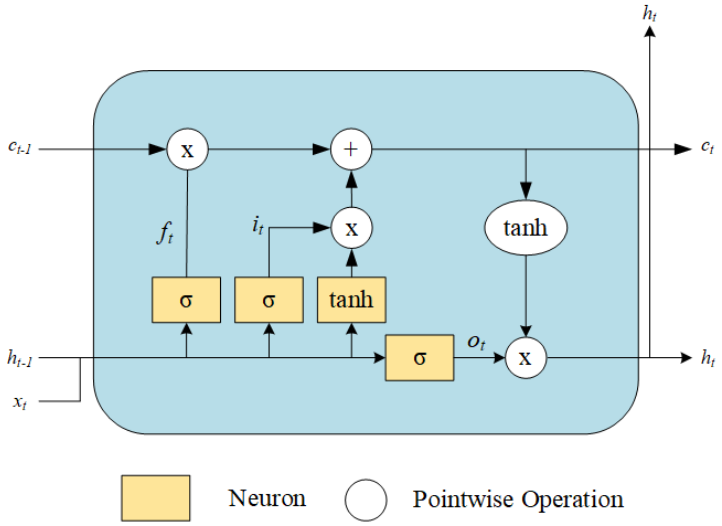

Figure 4: LSTM Cell Diagram

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \bigodot c_{t-1} + i_t \bigodot tanh(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \bigodot tanh(c_t)
\end{aligned}
$$

Similar in concept to LSTM, GRUs mitigate the vanishing gradient issue via gating mechanisms. However, their implementation does so with fewer neurons and only one recurrent state. Theoretically, this enables GRUs to be trained more quickly and achieve better performance than LSTM; however, research has found that the two cell designs perform similarly on a variety of tasks [11].

A more recent, alternative approach to the traditional RNN is through the constraint of layer connections, as posed by Independently Recurrent Neural Networks (IndRNN). In traditional RNNs (and gated variants), the recurrent states associated with a layer are connected to each cell in the respective layer; IndRNN proposed a restriction to these connections, such that each recurrent state is connected to only one cell in the layer [12]. This approach prevents multiple recurrent states from being harshly penalized or neglected as a result of one cell's weight update, solving the vanishing and exploding gradient problem so long as a sufficiently small optimizer learning rate is selected.

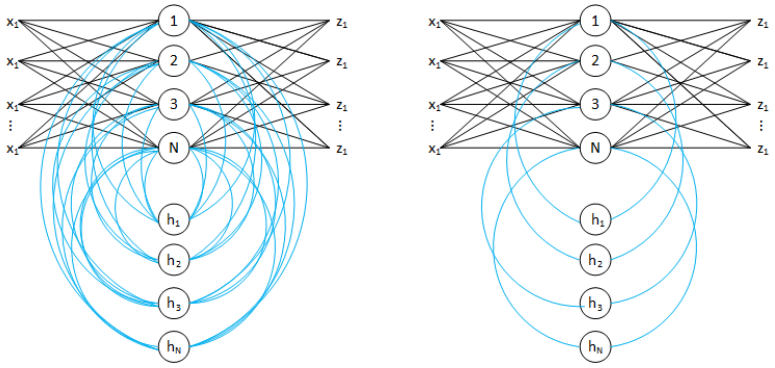

Figure 5: Visualization of Traditional RNN (left) and IndRNN (right) Layers

E Attention Mechanisms

While attention mechanisms have existed for years in the field of natural language processing, their application to other fields (such as computer vision) has only been considered recently [13]. For this research, soft self-attention is considered as in Figure 6 [10] and described by (1) - (5). A complete data sequence X is received and transformed into key K, query Q, and value V datasets. In the discussed attention, a sense of locality is embedded in the data by training a $softmax$ layer, whose input is a matrix multiplication of key and query datasets. Findings from [13] indicate that self-attention is a capable supplement following convolutional layers when extracting patterns from 2D data.

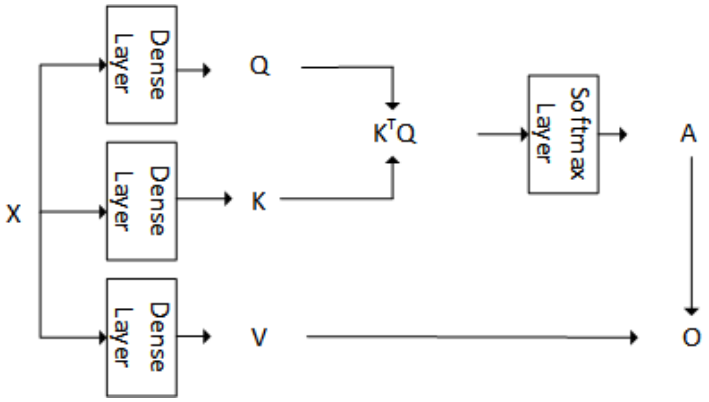

Figure 6: Functional Diagram of a Self-Attention Layer

$$Q = \sigma(\frac{W_Q^\intercal X}{\sqrt{d_{out}}}) \tag{1}$$

$$K = \sigma(W_K^\intercal X) \tag{2}$$

$$V = \sigma(W_V^\intercal X) \tag{3}$$

$$A = softmax(K^\intercal Q) \tag{4}$$

$$O = AV \tag{5}$$

4 Prior Research

Though the topic of flight prediction has been examined prior to 2016, research addressed specific aspects of the flight, such as descent and traffic merging ahead of arrival. Additionally, data sources for effective flight prediction - particularly aggregated GPS data - were not yet available. For the purpose of managing and informing communication cells throughout the NAS, many of these papers are therefore irrelevant. Four papers will be presented, with discussion focusing on their goals, selection of data sources, frameworks, and contributions.

In [7], the authors sought to develop a model that could correlate historic flight data with surrounding weather data. A Hidden Markov Model was developed, where flight coordinates were associated with those of the NOAA RAP dataset. Using the defined trajectory as observed emissions and regions of RAP coordinates as hidden states, the HMM was trained. 594 4D trajectories were collected for one flight (DAL2173) with identical arrival and departure locations (ATL to MIA). The authors were the first to formulate a complete trajectory prediction model and proposed the notion of surrounding weather data (feature cubes), a concept which has widely defined later model developments. The efficacy of this approach has yet to be matched, however it is unclear if the reported accuracy is a result of the learning approach or the heavy constraints placed on flight data collection.

A more complex, deep generative network is presented in [6]. At its core, this framework generates Gaussian Mixture Models using a sequence-to-sequence paradigm for Long Short-Term Memory (LSTM). The predictions of

these models are then filtered using a variety of techniques (Adaptive Kalman Filter, Beam Search, Rauch-Tung-Striebel Smoother). 3D Flight plans and 4D flight trajectories were recorded for 1,679 flights with identical arrival and departure airports (IAH to BOS). Weather data were collected from the NOAA NAM database, specifically U/V Winds, Air Temperature, and Convective Weather. While the results were not as compelling as those in [7], authors present several significant concepts, including a sequence-to-sequence paradigm and efficient methods of organizing and accessing weather data. Several reasons for these poorer results may be inferred: foremost, the NAM database is limited in resolution, as each data point is inter-spaced at 12 km and refreshed every 6 hours; the selected flight is infrequent, and collecting the number of flights in this paper may have required a significant range of seasons and consequent weather patterns; finally, the model itself may have been unnecessarily complicated by relying on the repeated generation and sampling of Gaussian Mixture Models.

Taking inspiration from the previous paper, the authors of [2] presented a convolutional-LSTM hybrid network to predict aircraft trajectories. This model presents a basis for hybrid-recurrent networks: weather features are extracted and represented through a series of convolutional and dense layers, while supplemented with the aircraft location prior to the provided cube. The combination of abstracted weather data and prior aircraft position are fed into an LSTM layer to predict the aircraft's position. LSTM layers were selected for recurrence to mitigate the vanishing gradient challenge of training traditional RNNs. Feature cubes were generated from Echo Top measurements, and flight data was collected for a total of 2,528 flights of identical arrival and departure points (JFK to LAX) over the dates November 1st, 2018 through February 5th, 2019. Initial research focused on 3D trajectory predictions (ignoring altitude), and reported efficacy in terms of improved error (described by Euclidean Norms) over that of the flight plan. The authors reported the accuracy of 47% of all flight plans were improved by their predictive model, on average by 12.3%. While this efficacy appears satisfying, this relative metric is not directly comparable to other research; the two papers prior reported efficacy in terms of a horizontal and vertical error (units of nautical miles and ft), with no reference to the error of related flight plans. As a result, this paper should be more critically contextualized in other research.

Finally, [14] discuss the prediction of aircraft based solely on prior Automatic Dependent Surveillance-Broadcast (ADS-B) data. Their research con-

siders single and multi-point forecasting of 4D trajectory using sequences of prior 4D data, as well as ground speed and heading information. Three models are presented for this task, one purely-convolutional, one purely-recurrent (LSTM), and one CNN-LSTM hybrid. The results reinforce the usefulness and importance of prior design choices in hybrid-recurrent networks, while also offering some qualitative understandings and intuition. Data were collected for approximately 397,000 flights from Qingdao to Beijing, providing the largest dataset of all considered in existing research. However, again, the metrics reported in this paper are not directly comparable to those described in prior research; those presented here are standard error metrics within deep learning (Mean Absolute Percentage Error, Mean-Squared Error, etc.), but not relevant to describing the usefulness of a trajectory prediction model.

Over the course of the existing body-of-research, several individual advances have been made; a core formulation and approach illustrated the viability of machine learning for this task in [7]; An initial deep learning framework was developed in [6] which, despite the limitations of used data, provided a framework and approach to the task; A more robust deep learning approach was formulated in [2] and validated with [14], which provided a starting point for research conducted in this thesis. However, each contribution mentioned considered different datasets, and frequently demonstrated the efficacy of an approach with different metrics. To advance the state of research on trajectory prediction, two contributions are noted in this thesis: 1) the generalization of the predictive task, experimenting with a variety of datasets, flights, and models to contextualize other research, and 2) a search for an accurate, generalizable deep learning model that suits this task.

III INSTRUMENTATION AND EQUIPMENT

All data processing and model development are performed on a workstation with AMD Ryzen Threadripper 1950X chipset and dual Nvidia RTX 2080 Graphics cards. To ensure stable hardware acceleration, development is performed on the Ubuntu 20.04 LTS release. Unless otherwise specified, software is developed in PyCharm projects, whose code and python environments are available at [15], [16].

As experimentation progressed, the selection of flights and dataset ranges varied significantly, and are specified for each experiment. Due to amount of data collected (particularly weather data), significant storage is necessary. CIWS Echo Top, Flight Plan, and Flight Track data are collected via Sherlock Data Warehouse in coordination researchers at NASA Glenn Research Center. CIWS VIL and miscellaneous products are collected in coordination with Lincoln Labs, but may be accessible via Sherlock as well. NOAA HRRR data are collected via NOAA's Google Cloud database.

IV TRAJECTORY PREDICTION TASKS AND EXPERIMENTS

The following section presents the experiment setup and results for all efforts made toward trajectory prediction. Flight and weather data were collected and preprocessed. Experiments to determine the optimal weather data and network structure were conducted in stages as well. The first four subsections discuss the data preprocessing and results related to comparing weather products, as well as initial results in comparing deep learning mechanisms; these subsections correspond to the efforts conducted and published in [17]. Continued efforts to address challenges with model tuning are described in the remaining subsection, including experiments to generalize flight and weather data, tune model hyperparameters, and provide finalized results of these improvement efforts.

For both experiments, the available data are split in a test-train ratio of 25%-75%. Presented results are determined from a 4-fold cross-validation. All model training is accomplished over 500 epochs, sampling the dataset with a batch size of 1. Error was calculated with a Mean Squared Error loss function, and models are currently adjusted using the Adam optimizer with a learning rate of $2 * 10^{-4}$.

1 Data Preprocessing

Preprocessing of flight and weather data was accomplished in several steps: the collection of individual, relevant flight data; the discretizing of 4D flight track data; the parsing and interpolation of 2D flight plans; and the collection of weather data surrounding the interpolated flight plan into usable feature cubes. The end products of preprocessing included a 4D flight track of historic GPS data at exactly 1-minute intervals; a 4D flight plan based on what is assumed as a last-filed flight plan at exactly 1-minute intervals; and a $NxZx20x20$ sequence of weather cubes, where N is the number of points recorded in the flight plan and Z is the number of relevant altitude levels for the select weather data. The following paragraphs provide a summary of these algorithms.

The collection of flight data are accomplished from Integrated Flight Format (IFF) data available at the NASA Sherlock Data Warehouse. IFF Data contain a complete set of messages from all active aircraft in the NAS in one day. Messages are all related to the aircraft's flight, including en-route GPS information, updates to an aircraft's flight plan, and general aircraft reports and information. To generate an initial set of files for each individual flight plan and flight track, a C-program was used to parse IFF data for messages specific to the relevant flights.

The resulting files for each flight, however, required some degree of cleaning. Flight track data were available at irregular intervals (between 1 second and 24 seconds), frequently with redundant messages. With the assumption of a constant airspeed, these flight tracks were linearly interpolated to provide 4D information at exactly 1 minute intervals.

The initial flight plan messages contain (among other things) a specified cruising altitude and list of waypoints and navigation aids resembling the general route of the aircraft through the NAS. Because these messages are provided as the flight is already beginning (taxiing, takeoff, en-route, etc.), creating a trajectory using the last-filed flight plan requires use of the earliest complete message. In order for the message to be useful in a deep learning model, this list must be interpreted into latitude and longitude coordinates. Interpretation is accomplished by querying the OpenNav website. From there, altitudes can be assigned based on the airport ground elevations and the cruising altitude after takeoff; initial and final timestamps are associated from the processed flight track; and the list of interpreted waypoints is interpolated to exactly 1-minute intervals.

As a final step, each dimension which will be used by the deep learning frameworks (latitude, longitude, altitude, weather measurements) is normalized to a value between 0 and 1. The scaling is individual for each dimension, and is performed to match data with the typical operating range of deep learning models.

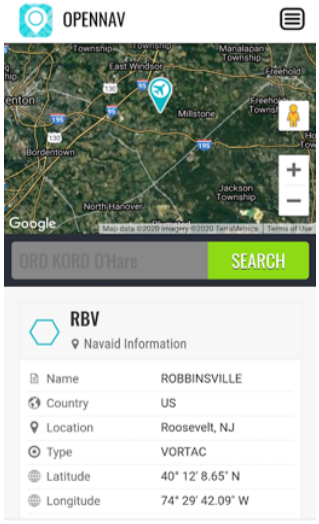

Figure 7: Sample Navigation Aid Query from [1]

The generation of weather cubes from a given flight plan is the most computationally intensive preprocessing task. For each point along the flight plan, a $Zx20x20$ set of gridded weather data is collected based on the projected heading of the aircraft, where Z is 1 for all 3D products (Echo Top, VIL) and 3 for all 4D products (HRRR measurements). This process is summarized in Algorithm 1 and visualized by Figure 8. At current, feature cubes have only been generated for 3D products.

Algorithm 1 Weather Cube Generation

Require: Flight Plan FP, Gridded Weather Database W

```
for time, alt, lat, lon in FP do
    find and open most-recent, valid file w in W
    calculate heading θ and orthogonal vectors θ⊥ from lat_i, lon_i, lat_{i−1}, lon_{i−1}
    locate the nearest latitude w_lat and longitude w_lon to lat_i, lon_i
    generate a 20-point axes X, Y along θ and θ⊥
    for x,y in X,Y do
        locate the nearest latitude wc_lat and longitude wc_lon
        if w varies by altitude then
            find containing altitude group w_alt
            for zϵ[−1, 1] do
                collect feature cube data wc_data at (wc_lat, wc_lon, w_alt + z)
            end for
        else
            collect feature cube data wc_data at (wc_lat, wc_lon)
        end if
    end for
end for
```

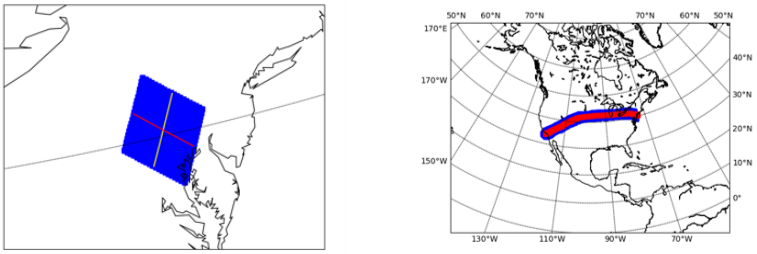

Figure 8: Selection of one feature cube (left) and all cubes (right, blue) along a flight plan (right, red)

2 Recreating the Initial Work

This section highlights the attempts to re-create the initial efforts to develop a deep-learning model, which approximated the scope of work in [2] and [18]. The two papers provided a basis for much of the initial efforts on this topic,

and comprised of three experiments summarized in Table XX. Notably, two of the three experiments did not incorporate altitude into trajectory predictions. To simplify the task of re-creating the research, experiment 2 is focused on: not only does this focus efforts to a 3D trajectory prediction at first, but also avoids the challenge of batch sizes for sequences of unequal length.

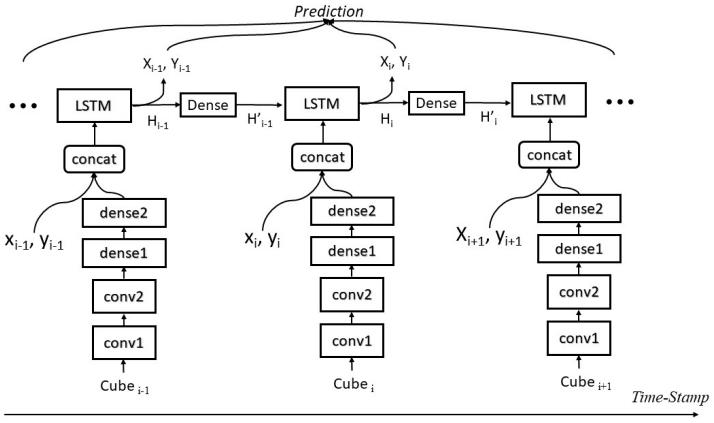

Figure 9: CNN-LSTM Model Presented in [2]

The development of the proposed model was notably simplified for the approach considered. The model proposed initially in [2] had been developed at a much lower level using TensorFlow, to incorporate several custom controls: the weather cube and flight plan information was designed to be fed into the model's LSTM forget gate, while prior trajectory points were fed through the model's LSTM input gate; additionally, dense layers were incorporate between the update of the LSTM's hidden states for each item flight sequence, as a way to briefly expand the dimensionality of the hidden state. Both of these behaviors required the design of a specific, customized form of an LSTM layer, which was beyond the knowledge and development time possible over the course of this thesis. Instead, the default LSTM layer within PyTorch was employed, where all useful data are provided to the input gates of the LSTM layer and no deep learning mechanism is incorporated into the updating of hidden states.

In initial efforts, data preprocessing and training focused on handling flights as sequences of data interpolated to a 1-second interval, while working with the complete 100 days of flight data. For a number of reasons, this approach was abandoned. First, generating such long sequences of data

required an analogously long period of time to pass the sequence through a deep learning model; in initial attempts (which trained the model in a time-series forecast paradigm), each epoch required approximately 46 minutes to process and backpropogate based on the 75% split of training data over the 100 days of collected flights.

Additionally, the model accuracy was significantly hindered by such long sequence lengths. As seen in the sample prediction of Figure 10, the model was unable to retain and meaningfully predict for all sequence points in the flight; at several instances, select points would be predicted wildly astray from the rest of the predicted flight path. This likely reflects the limits of LSTM memory retention: research has indicated that LSTM can only retain sequences of information no more than 1000 points; by contrast, a typical non-stop flight from New York to Los Angeles lasts roughly 6 hours (21,600 seconds) [12].

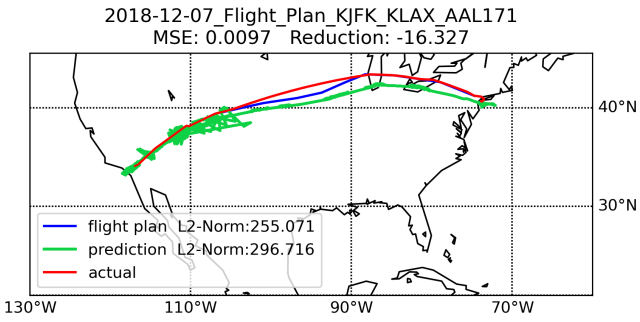

Figure 10: Sample Prediction Using 1-Second Interpolated Flight Data

To improve the training times and accuracy of recreating this model, data in all future experiments were interpolated at 1-minute intervals. This yielded a typical sequence length closer to 360 points, well within the limits of LSTM. Additionally, Model training shifted focus to a sequence-to-sequence paradigm; this limits the amount of training performed: in a time-series forecast, input data includes a window of past trajectory points, requiring

each flight to be used multiple times in training in, such that the window shifts throughout the flight duration. Simplifying to a sequence-to-sequence paradigm requires each flight to be used only once in training, greatly reducing the duration of each training epoch.

Finally, to insure training could be conducted and modified quickly, data were limited to flights in a two-week period as opposed to a 100 day period. The specific two-week period was determined by selecting a date range with the best reported coverage; from January 10th to Januray 24th, no Echo Top measurements were missing from available databases.

After re-considering the approach to trajectory prediction, models were able to be trained over the 14 days of 3D flight data. Figure 11 illustrates two sample flight plans and predictions. Some issues still arose during predictions as a result of shortcomings in preprocessing; specifically, flights and trajectories would occasionally include incomplete sequences, as seen on the right; this occurrence was reduced by revisiting the preprocessing algorithms, as well as setting up more stringent data filters prior to model training.

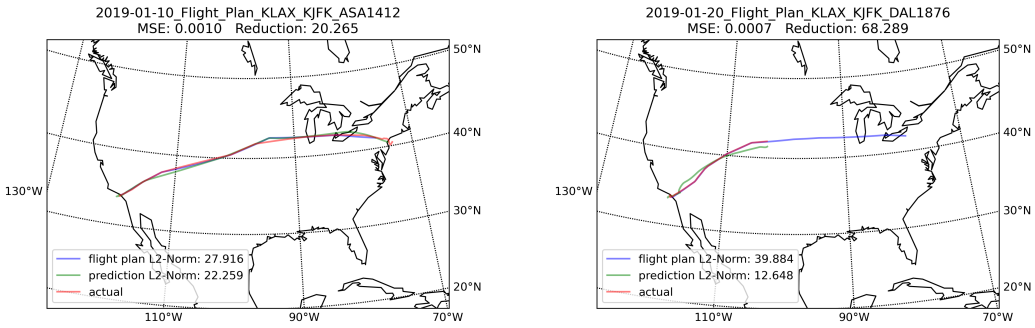

Figure 11: Sample 3D Trajectory Predictions, one of which is expected (left) and one incomplete (right)

Once 3D trajectory predictions were seen as functional, the incorporation of 4D data became somewhat trivial. Altitude information is directly available flight trajectory points, and can be inferred based on the cruising altitude provided in flight plan information; incorporating altitude into the hybrid-recurrent model was a matter of expanding the set of data provided from preprocessed flight plans. A sample 4D prediction is provided in Figure 12.

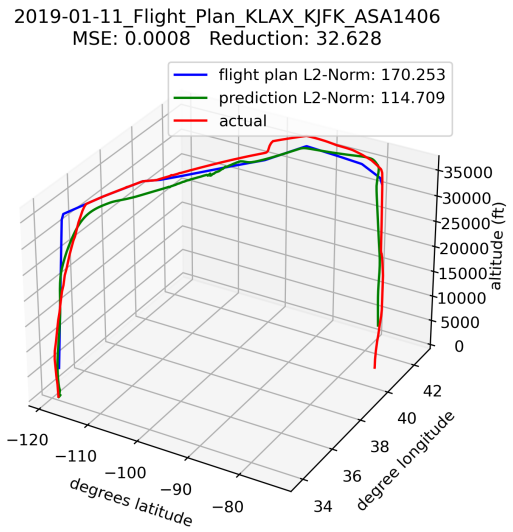

Figure 12: Sample 4D Trajectory Prediction

3 Weather Data Analysis

After building a foundation of data and deep learning models, the first experiment seeks to determine the optimal combination of weather datasets for predicting flight reroutes. Selected weather data include Echo Top and VIL products from CIWS, as well as atmospheric temperature and U/V Wind Components from NOAA HRRR. To limit the number of combinations considered, a cross-correlation of weather products is first performed using the products' complete coverage of the continental United States. The results of this cross-correlation will determine which combination of weather products supplement one another enough to justify the computational cost of providing additional data to the models. From here, the individual weather products and selected combinations thereof will be used to train a hybrid-recurrent model of fully convolutional layers and a single LSTM layer. For this section, all flight and weather data are collected in the two-week period from January 10th to 24th, 2019; flights travelling from Los Angeles to New York are collected, totaling 379 usable for training.

Due to the size and mismatch of coordinate space between some weather datasets, several constraints are placed for cross correlation. All weather product measurements have been normalized to a scale between 0 and 1.

When correlating between CIWS and HRRR products, a rectangular subset of data are selected to ensure a matching coordinate boundary. From there, HRRR data are interpolated to match the spatial resolution of CIWS data. Since CIWS data do not vary by altitude, its measurements are averaged over all relevant altitude levels (all levels for VIL, all levels containing or below the Echo Top measurement for Echo Top). Finally, the cross-correlation only considers the exact alignment of the two products, not any shifted forms which would require padding the boundaries of the dataset. Correlation coefficients are collected for each product over a two-week period, yielding the set of histograms in Figure 13.

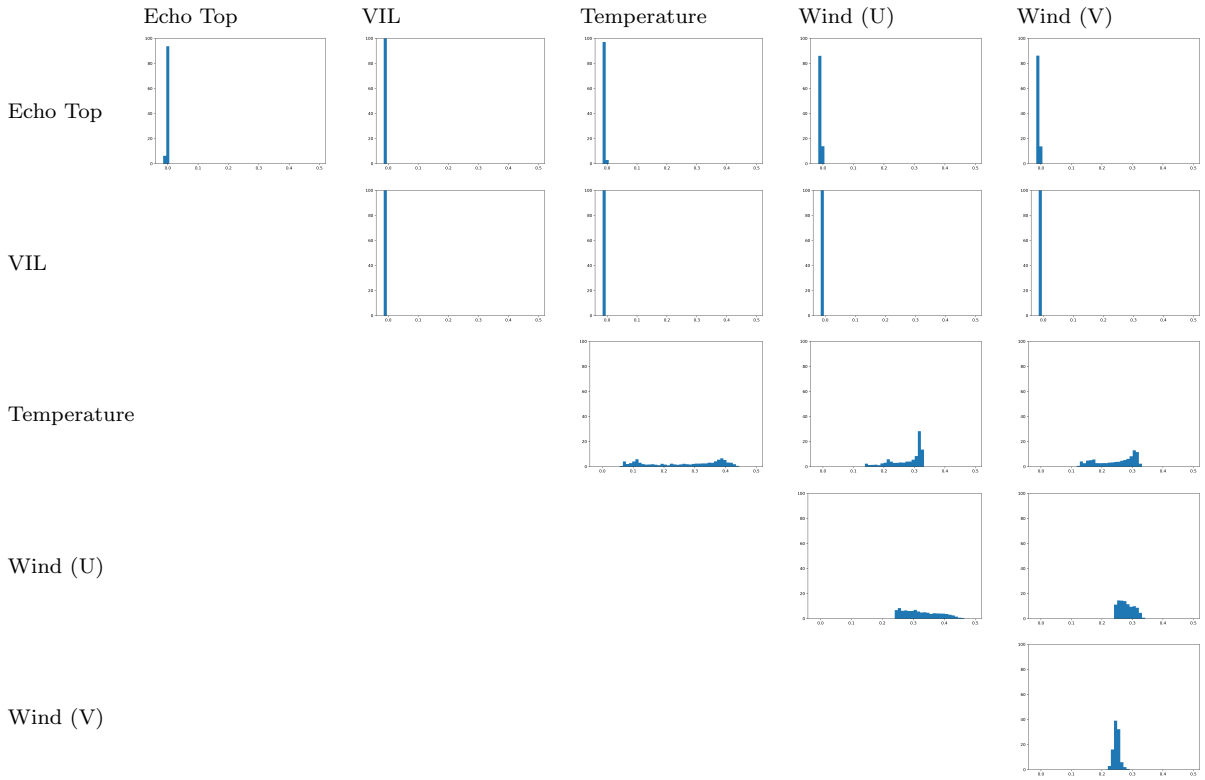

Figure 13: Histograms of Cross-Correlation Coefficients Ranging (0, .5)

Based on these results, Echo Top and VIL data were each found to be weakly correlated to other products; this is likely due to the sparsity of Echo Top and VIL data at any given time. In contrast, Temperature and Wind Components were all moderately correlated with one another; as a result, it

is anticipated that only one of the three NOAA products should be used. Because Temperature provided the lowest correlations of all NOAA products, it is predominantly considered in combinations of NOAA products. Therefore, the model training considers 8 combinations of datasets: Echo Top, VIL, Temperature, and V Wind Component will be considered individually; Combinations will also consider Echo Top and VIL, Echo Top and Temperature, and VIL and Temperature. Finally, Temperature and V Wind Component will be tested in combination, to verify the assumption of performance based on their higher correlation.

Table 2: Summary of Weather Product Performances

Product(s)	Horizontal Error (μ/σ in nmi)	Vertical Error (μ/σ in ft)	Improvement over Echo Top ($\mu_{Horiz}/\sigma_{Horiz}$ as percent)	Improvement over Echo Top (μ_{Vert}/σ_{Vert} as percent)
Echo Top	**50.017**	**1160.07**	**0**	**0**
	48.854	1420.26	0	0
VIL	**55.171**	**1230.23**	**-10.304**	**-6.048**
	67.276	1514.95	-37.708	-6.667
TMP	**52.983**	**1130.72**	**-5.931**	**2.530**
	60.901	1399.41	-24.659	1.468
U Wind (E/W)	**50.560**	**1128.17**	**-1.085**	**2.749**
	54.588	1420.57	-11.738	-0.022
V Wind (N/S)	**50.167**	**1097.16**	**-0.29**	**5.422**
	51.376	1390.80	-5.164	2.074
ET + VIL	**50.670**	**1118.72**	**-1.305**	**3.564**
	57.596	1365.45	-17.895	3.859
ET + TMP	**50.194**	**1156.50**	**-0.354**	**0.307**
	51.937	1424.41	-6.312	-0.292
VIL + TMP	**52.520**	**1248.81**	**-5.005**	**-7.650**
	65.513	1558.70	-34.101	-9.748
TMP + V Wind	**49.578**	**1128.25**	**0.877**	**2.743**
	51.764	1430.29	-5.957	-0.707

After training the products and product combinations above, prediction results are summarized in Table 2, with percent improvement comparisons drawn against Echo Top, which is treated as the default due its use in [2]. Several trends can be observed: Without considering combinations of mul-

tiple products, Echo Top provides the minimum horizontal error; this is expected, due to the nature of the measurement being tailored to air traffic management, alongside its sparsity. VIL performed notably worse than most products, despite its sparsity and correlation to convective weather; this reflects how VIL represents only the presence of liquid, not whether it is indicative of humidity, rainfall, snowfall, etc. - not all of which require flight adjustments. While no NOAA data product could provide an improved horizontal accuracy, the use of any of the three provided degrees of improvement to vertical accuracy predictions – likely because of data being altitude-varying. Notably, V Wind Components provided the greatest improvement in vertical accuracy; it is hypothesized this may reflect a skew in flight data, where a notable percent of arrivals and departures occurred with a significant north/south component (parallel to V Wind), while the en-route flight had a majority east/west component (parallel to U Wind).

No combinations of products provided sufficient improvements in accuracy to justify their additional data processing and model complexity. However, challenges of model reproducibility may have inhibited the usefulness of some products, especially with populous data items such as NOAA measurements. In particular, prior experiments without normalizing temperature data yielded horizontal accuracies much closer to (even surpassing) those of Echo Top, making temperature a viable choice for predictive modeling.

4 Initial Structure Comparison

The second experiment seeks to identify trends in component selection and usefulness among different hybrid-recurrent implementations. This section only considers a direct implementation of related deep-learning mechanisms with assumed hyperparameters; a more in-depth exploration of hyperparameters and overfitting follows.

Extending the general form of Figure 14, models were defined by the selection of weather feature extraction mechanism, recurrence mechanism, and number of recurrence layers. These parameters were assumed and modified from the network layout in [2], and are summarized in Table 3. Extraction mechanisms were defined in a depth of three layers, and included a purely convolutional design, a purely self-attention design, and a convolutional design with self-attention serving as a final layer. Recurrence techniques included LSTM, GRU, and IndRNN layers, always with a hidden state size of 100. Finally, the depth of recurrence was limited to 1 or 2 layers. For IndRNN

cells, an additional layer was included to allow for information sharing between neurons. All models were trained on Echo Top feature cubes and flight data. All models were initially trained with 379 flights from Los Angeles to New York, collected over the two week period of January 10th to 24th, 2019.

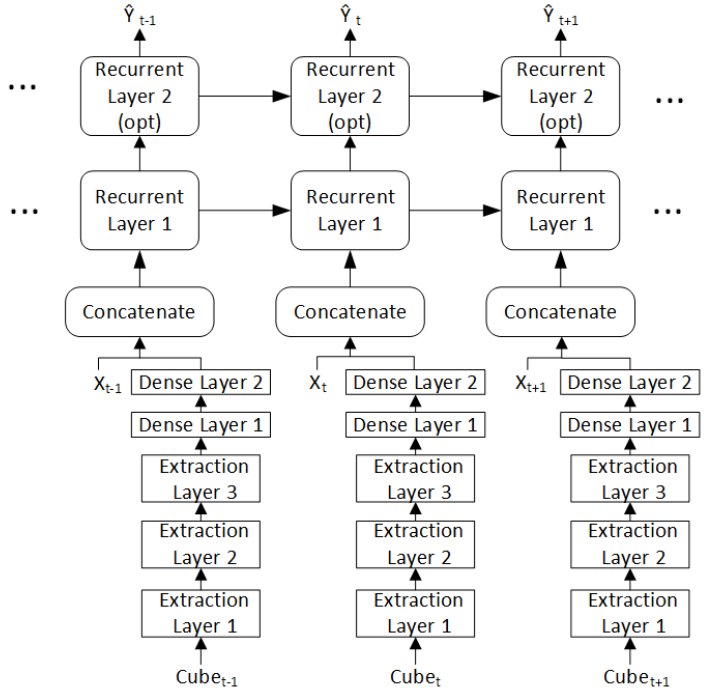

Figure 14: Layout of Hybrid-Recurrent Framework

Table 3: Default Model Parameters

Parameter Description	Parameter Value
Convolution Kernel Sizes	[6x6, 3x3, 1x1]
Convolution Stride Lenths	[2, 2, 1]
Convolution Filter Sizes	[1, 2, 4]
Attention Output Dimensions	[128, 36, 36]
Dense Layer Sizes	LSTM, GRU: [16, 3] IndRNN: [16, 97]
Recurrent Input Size	6
Recurrent Depth	GRU, LSTM: 1 or 2 Layers IndRNN: 2 or 3 Layers
Optimizer Learning Rate	$2x10^{-4}$

While some initial experimentation has been conducted, few conclusive results were drawn. Results are considered inconclusive due to prior tests yielding large variances in error. Table 4 is presented to indicate preliminary results and a general format for comparing effectiveness of architecture design, where horizontal and vertical error are standard metrics for general flight trajectory prediction error, and percent improvements are prescribed in comparison to the flight plan and a convolutional-LSTM design of 1 layer (a model inspired from [2] and used as a baseline for this task). The table does not detail all possible model combinations, with the intent that those presented would sufficiently describe the trends in usefulness of different recurrent and extraction components.

Table 4: Summary of Select Models' Performances

Model	Horizontal Error ($\boldsymbol{\mu}/\sigma$ in nmi)	Vertical Error ($\boldsymbol{\mu}/\sigma$ in ft)	Improvement over Flight Plan ($\boldsymbol{\mu_{Horiz}}/\mu_{Vert}$ as percent)	Improvement over CNN-LSTM1lay ($\boldsymbol{\mu_{Horiz}}/\mu_{Vert}$ as percent)
CNN-LSTM1lay	**63.5584**	**1160.27**	**39.5920**	**0**
	26.8905	1500.83	64.0127	0
CNN-LSTM2lay	**60.9995**	**1167.39**	**42.0241**	**4.0260**
	29.2265	1551.46	63.7919	-0.6135
CNN-GRU1lay	**59.8954**	**1120.04**	**43.0735**	**5.7632**
	28.0559	1399.75	65.2606	3.4676
CNN-GRU2lay	**47.2278**	**1156.16**	**55.1131**	**25.6938**
	22.9868	1332.40	64.1404	0.3548
CNN-IndRNN2lay	**119.1314**	**1219.99**	**-13.2263**	**-87.4361**
	63.1298	1682.68	62.1607	-5.1463
CNN-IndRNN3lay	**122.6245**	**1219.86**	**-16.5463**	**-92.9320**
	61.8804	1682.68	62.1645	-5.1355
CNN+SA-LSTM1lay	**59.3252**	**1178.57**	**43.6154**	**6.6603**
	29.5847	1546.38	63.4454	-1.5763
CNN+SA-LSTM2lay	**60.1143**	**1152.56**	**42.8654**	**5.4187**
	28.4656	1537.15	64.2520	0.6651
SA-LSTM1lay	**40.9453**	**804.73**	**61.0843**	**35.5785**
	23.7972	1054.89	75.0405	30.6436
SA-LSTM2lay	**56.2102**	**990.82**	**46.5760**	**11.5613**
	25.1934	1294.62	69.2687	14.6051

With large variances in mind, one trend still tends to appear, which is the usefulness of self-attention in this task. The use of self-attention, while potentially incrementing performance as a supplement to convolutional layers, notably improved performance of models when behaving as an outright replacement for convolutional layers. The improvements via fully attentional extraction may be caused by the globality of self-attention: since self-attention layers consider all features in relation to each other (as opposed to a fixed number of neighbors in one sequence element), attention mechanisms may be able to define complex filters that consider the interactions of data between each feature cube in the provided sequence. Consequently, the use of self-attention as a supplement to convolutional networks may be inhibited by convolutional extraction techniques: since convolutional layers can

only extract features within each cube individually, the patterns extracted may be counterproductive from a more global perspective.

An additional point should be made, that IndRNN performed notably poorly. When comparing results of IndRNN-based models, error was notably worse than that of the flight plan, and moreso in comparison to the baseline model. This may be a reflection of implementation challenges; the selected hyperparameters may have been limiting for the model. However, due to implementation challenges, this mechanism will be abandoned in remaining sections; despite being the least computationally-complex of the recurrent mechanisms tests, they require significant training time as GPU-accelerated variations of the mechanism are no longer supported.

5 Continued Efforts in Network Exploration

Following initial attempts to propose useful deep learning mechanisms that would improve trajectory prediction, several questions and challenges remained. Initial efforts in comparing deep learning mechanisms relied on a set of assumed hyperparameters, which may have limited the performance of each model. Furthermore, variations in model accuracy between training sessions prevented a consistent comparison of models. A final, major concern is the fear of overfitting; 379 flights were split between training and testing in the initial results, all collected over a two-week period for one specific route. This is likely to provide a model heavily biased toward not only this route, but also this general flight direction and weather trends within this season.

This section aims to identify and address these problems in a number of steps. First, an initial attempt to tune model performance was attempted, which identified limited trends (including the degree of overfitting) while illustrating a need for more in-depth efforts. Follow-on efforts focused on data collection and parameters to address overfitting. Finally, model hyperparameters were experimented with, in order to identify optimal channel depths, recurrent depths and hidden state sizes, and optimizer selections.

For each of the following subsections, Ray Tune provided a basis and implementation for scheduling experiments and computer resources [19]. Ray is a collection of python libraries, which provide implementation of both scheduling and optimization algorithms for hyperparameter tuning in Pytorch and TensorFlow. While a variety of algorithms are available, few if any were used for this project, tended toward finding visual representations of hyperparameter trends; this chosen with then intent of allowing for a more

verbose comparison and analysis of hyperparameter selection.

A Model Tuning: Initial (Naive) Attempt

The first attempt to find optimal hyperparameter settings was admittedly naive. For each model type, a large search space was configured as defined in table 5. Note that this space incorporated dropout, an initial attempt to address potential overfitting. These layers were incorporated in select portions of the network, as illustrated in Figure 15. Using Asynchronous Hyperband, each search space was sampled 20 times, with additional training epochs (upto 300) progressively performed on models with the best validation loss. Again, data used for training consisted of the 379 flights over a two-week period.

Table 5: Hyperparameter Search Space of Initial Tuning Attempt

Hyperparameter	Sampling Distribution
Optimizer	gridSearch(Adam, RMSProp)
Extraction Channels	[1, randInt(1, 25), randInt(1, 25)]
Dense Layers	1-5 layers, each uniquely sampled by randInt(1, 48)
RNN Input	randInt(3, 20)
RNN Depth	randInt(1, 4)
RNN Hidden Size	randInt(10, 1000)
Dropout Rate	uniform(0, 0.5)
Learning Rate	logUniform($2 * 10^{-6}$,$2 * 10^{-2}$)

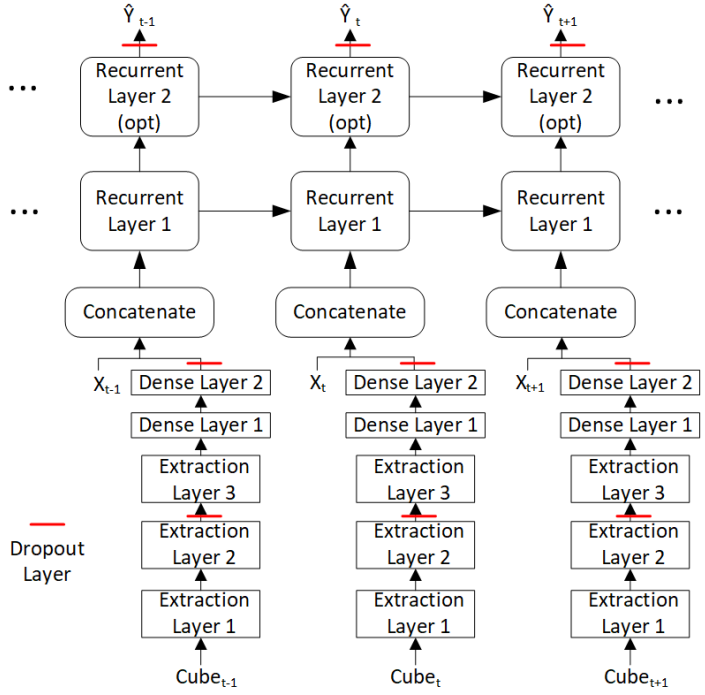

Figure 15: Hybrid-Recurrent Architecture, with Dropout Layer Placement

Several shortcomings became apparent with this approach. Primarily, the necessary number of samples to effectively represent the indicated search spaces would be excessive; this lead to an inability to recognize many trends in hyperparameter selection. In addition, there had been little consideration for the repeatability of these results; model weights were initialized with random distributions, each from a unique seed related to that model's training session. As a result, there was no way to guarantee that each model was training fairly, from a starting point of equal or approximate inaccuracy, and that the model could obtain more competitive parameters within its first epoch.

Table 6: Initial Tuning Attempt: Best Models by Training Loss

Model	Training Loss	Validation Loss	Iterations	Extraction Channels	Dense Layers	RNN Input Size	RNN Depth	RNN Hidden Size	Dropout Rate	Optimizer	Learning Rate
CNN-LSTM	0.00047	0.00476	262	[1, 13, 24]	[25]	19	2	601	$9.131 * 10^{-2}$	RMSProp	$2.615 * 10^{-4}$
CNN-LSTM	0.00055	0.00239	272	[1, 10, 10]	[30]	15	1	54	$7.740 * 10^{-2}$	RMSProp	$9.202 * 10^{-4}$
CNN-GRU	0.00061	0.01429	256	[1, 14, 21]	[20, 10]	15	4	718	$2.437 * 10^{-1}$	RMSProp	$2.558 * 10^{-4}$
CNN-GRU	0.00082	0.01346	249	[1, 5, 5]	[21, 10]	19	2	128	$1.935 * 10^{-1}$	RMSProp	$1.065 * 10^{-3}$
CNN-LSTM	0.00088	0.00188	283	[1, 5, 11]	[30]	15	2	462	$2.002 * 10^{-2}$	Adam	$1.758 * 10^{-4}$

Table 7: Initial Tuning Attempt: Best Models by Validation Loss

Model	Training Loss	Validation Loss	Iterations	Extraction Channels	Dense Layers	RNN Input Size	RNN Depth	RNN Hidden Size	Dropout Rate	Optimizer	Learning Rate
CNN-LSTM	0.00140	0.00128	129	[1, 5, 1]	[18, 9]	11	2	369	$6.456 * 10^{-3}$	Adam	$1.647 * 10^{-4}$
SA-LSTM	0.00172	0.00154	204	[1, 17, 5]	[13]	10	1	760	$8.596 * 10^{-2}$	Adam	$9.492 * 10^{-5}$
CNN-LSTM	0.00237	0.00161	64	[1, 17, 12]	[17, 31]	4	4	46	$7.858 * 10^{-2}$	Adam	$3.140 * 10^{-4}$
CNN+SA-LSTM	0.00197	0.00167	32	[1, 17, 20]	[19]	13	2	766	$2.347 * 10^{-2}$	Adam	$3.123 * 10^{-3}$
CNN-GRU	0.00136	0.00178	132	[1, 3, 11]	[12]	19	4	50	$1.220 * 10^{-2}$	Adam	$2.806 * 10^{-3}$

Even so, some trends were noticeable. The most glaring of these trends being the epochs. Even with the maximum number of epochs being less than those defined in the initial experiments, model training and validation losses were able to improve over prior attempts. Regarding model parameters, the assumed recurrent input size of 6 had been limiting; all but one of the best trained models in Tables 6 and 7 required a larger recurrent input size, most of which requiring a size larger than 10. Additionally, no models with useful validation losses had significant dropout (at or exceeding 20%). Finally, a notable trend appeared in better-performing models, where the recurrent network would balance either a larger hidden size or recurrent depth; infrequently, both would be present, but never were both absent. This provided some intuition for guiding later hyperparameter tuning.

Finally, the use of RMSProp exclusively provided the best training accuracies for this task, though with two caveats. First, none of the well-performing models associated with this optimizer incorporated attention; this may likely be a fluke, resulting from the small sample size in this experiment. Second, the models which used RMSProp in training each experienced severe overfitting, some by an order of magnitude. This result motivates both the use of RMSProp in further experiments, and the efforts to address overfitting more seriously in later experiments.

B Model Tuning: Addressing Overfitting

An initial attempt to address overfitting was conducted in this experiment, consisting of two steps: first, data was generalized by collecting a larger variety of flights within the two-week period. Second, hyperparameters to restrict model training were introduced and tested over ranges.

In order to generalize the available data within the two-week period, flights were collected to vary the general heading and region of coverage within the continental United States. In total, 1,468 flights were collected, with 734 of those flights considered useful for training. The varied flight collections include New York (KJFK) to Los Angeles (KLAX), Houston (KIAH) to Boston (KBOS), Atlanta (KATL) to Chicago (KORD), Atlanta to Miami (KMCO), and Seattle (KSEA) to Denver (KDEN). A general visual and detailed breakout of flights are prescribed in Figure 16 and Table 8.

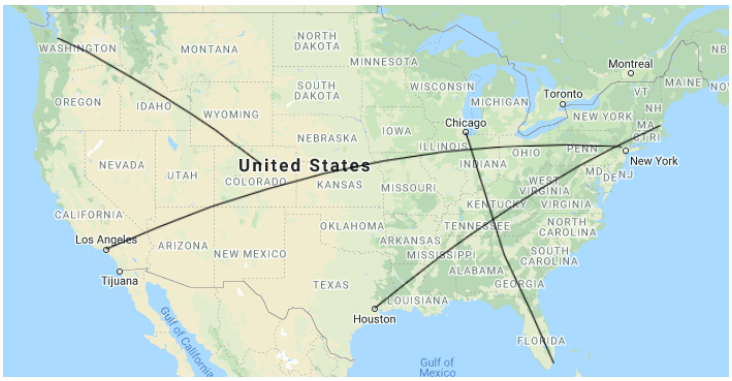

Figure 16: Coverage of Selected Flights for Data Generalization

Table 8: Breakout of Selected Flights for Generalization

Flight	General Heading	Nonstop Time	Estimated Flights in Two-Week Period
KLAX - KJFK	West-Southwest	5 hrs 30 mins	990
KIAH - KBOS	Northeast	3 hrs 45 mins	40
KATL - KORD	North-Northwest	1 hr 45 mins	313
KATL - KMCO	South-Southeast	1 hr 45 mins	388
KSEA - KDEN	Southeast	2 hrs 30 mins	238

In addition, tuning hyperparameters were configured that are traditionally incorporated to reduce overfitting. These included Batch Normalization, Dropout and Weight Regularization (referred to as Weight Decay in PyTorch). Batch Normalization was incorporated between each layer of the network, and could be configured in three manners: its absence, its presence without learned parameters, and its presence with learned parameters (Affine, incorporating β and γ in Equation 6). Dropout remains configured as described in Figure 15, with one of seven Dropout rates possible. Finally, Weight Regularization was an L-2 regularization as implemented by PyTorch's optimizers, with one of eight regularization rates considered. Each combination of the three hyperparameters was trained for 200 epochs, using the RMSProp optimizer with a learning rate of $2 * 10^{-4}$ and the generalized set of two-week flight data. To insure repeatability, training used a constant seed for weight initialization and forced deterministic algorithms, as discussed in [20]. Additionally, training was only conducted for the CNN-LSTM hybrid-recurrent architecture, as it is assumed that these parameters will be transferable to other architectures solving this task. For brevity in this experiment, it is assumed that these parameters will be generally transferable to other model architectures.

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta \tag{6}$$

From the results highlighted in Table 9 and Figure 17, batch normalization tended to have adverse effects on training. Without normalization, the model reached an error minima close to $2 * 10^{-3}$ in training, and $2.7 * 10^{-3}$ in validation. While overfitting is present, the degree of overfitting is negligible compared to the degree of accuracy; incorporation of un-trainable normalization did yield a point where validation loss improved over training loss, but with both yielding a best error magnitudes greater than models without normalization. Incorporating learned parameters only served to force a divergence of training and validation loss, with minima for both never reaching those even present in the prior un-trainable scenario.

Several reasons may exist for this, all resulting from a misunderstanding of Batch Normalization's use case and purpose. First and foremost, batch sizes are restricted to one in all training, in order to prevent model accuracy issues resulting from batching together data of varied lengths. The consequence of this, is that the expectations and variances calculated, are done

so over the sequence of input flight and weather data; incorporating batch normalization would only serve to skew the representation of existing data across the complete flight sequence. Alternative, more useful mechanisms may include Layer Normalization and Instance Normalization.

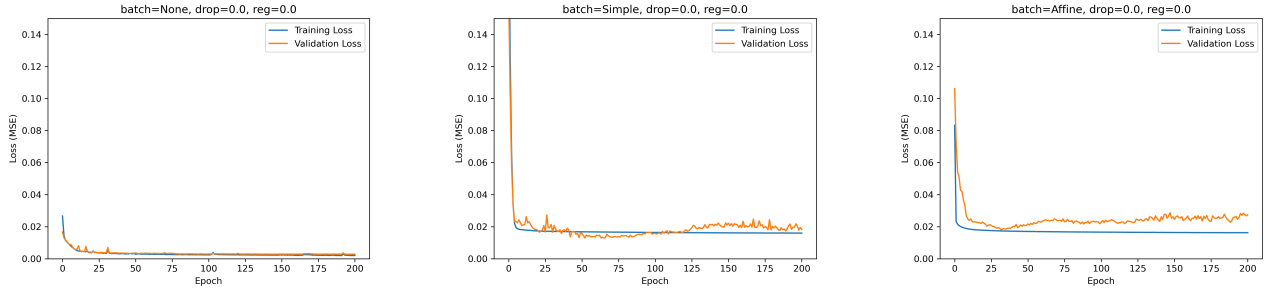

Figure 17: Training Plots of CNN-LSTM Model without Batch Normalization (left), with Batch Normalization (middle), and with Affine Batch Normalization (right)

Table 9: Final Losses for Batch Normalization Hyperparameters

BatchNorm	Training Loss	Validation Loss
None	0.002004	0.002747
Un-trainable	0.015965	0.018279
Trainable	0.016219	0.027436

The narrative for examining Dropout's usefulness to this problem is drastically different. As illustrated in Figure 18, Dropout rates greater than 1% served only to force a drastic divergence between training and validation losses. For Dropout rates at or less than 1%, however, an improvement can be noted: rates of 0.01% and 1% trained toward validation losses comparable to that without Dropout, while a rate of 0.1% provided marginal improvements at the cost of a raised training loss. While a rate of 0.01% approaches an absence of Dropout, as only a handful of weights are disconnected each iteration, this rate remains distinct due to the large number of epochs over which training occurred. Ultimately, a marginal amount of Dropout (.1%) is recommended in future models.

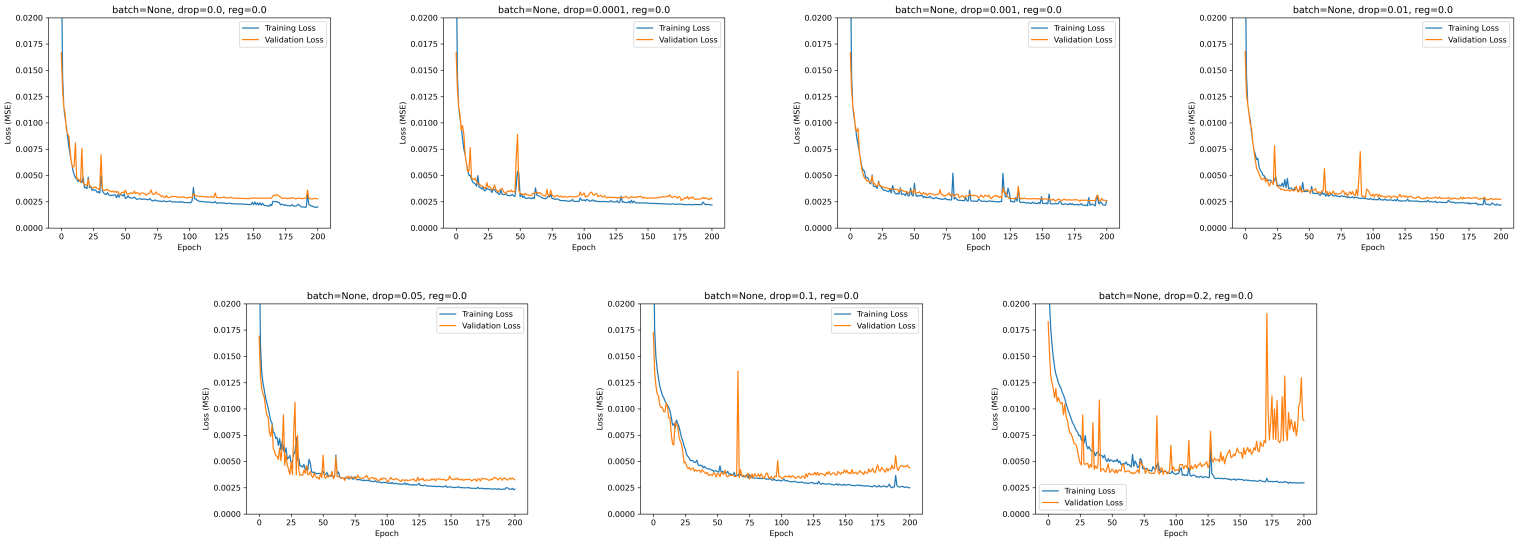

Figure 18: Training Plots of CNN-LSTM Model with Increasing Dropout Rates. From Left to Right: (Top) 0%, .01%, .1%, 1% (Bottom) 5%, 10%, 20%

Table 10: Final Losses for Dropout Hyperparameters

Dropout Rate	Training Loss	Validation Loss
0%	0.002004	0.002747
0.01%	0.002193	0.002781
0.1%	0.002607	0.002598
1%	0.002202	0.002755
5%	0.002369	0.003257
10%	0.002501	0.004381
20%	0.002967	0.008869

Finally, the incorporation of Weight Regularization yielded similar recommendations as that of Dropout. As summarized in Table 11 and Figure 19, minuscule amounts of regularization were able to improve validation loss at the cost of marginally worse training loss. However, the effects of Weight Regularization are distinct from those of Dropout. As regularization increased, the level of noise between training epochs decreased, with the model converging more smoothly toward its optimal solution. For all values, train-

ing and validation loss remain closely tied to one another. At the same time, the minimum loss of these optimal solutions continue to rise with increasing regularization, particularly beyond $1 * 10^{-4}$. Examining the final losses in Table 11 identifies a pocket of competitive-to-improved losses over a model without regularization, with penalty values from $1*10^{-8}$ to $1*10^{-6}$. The latter value is selected, despite providing a sub-optimal improvement, to insure a significant penalty.

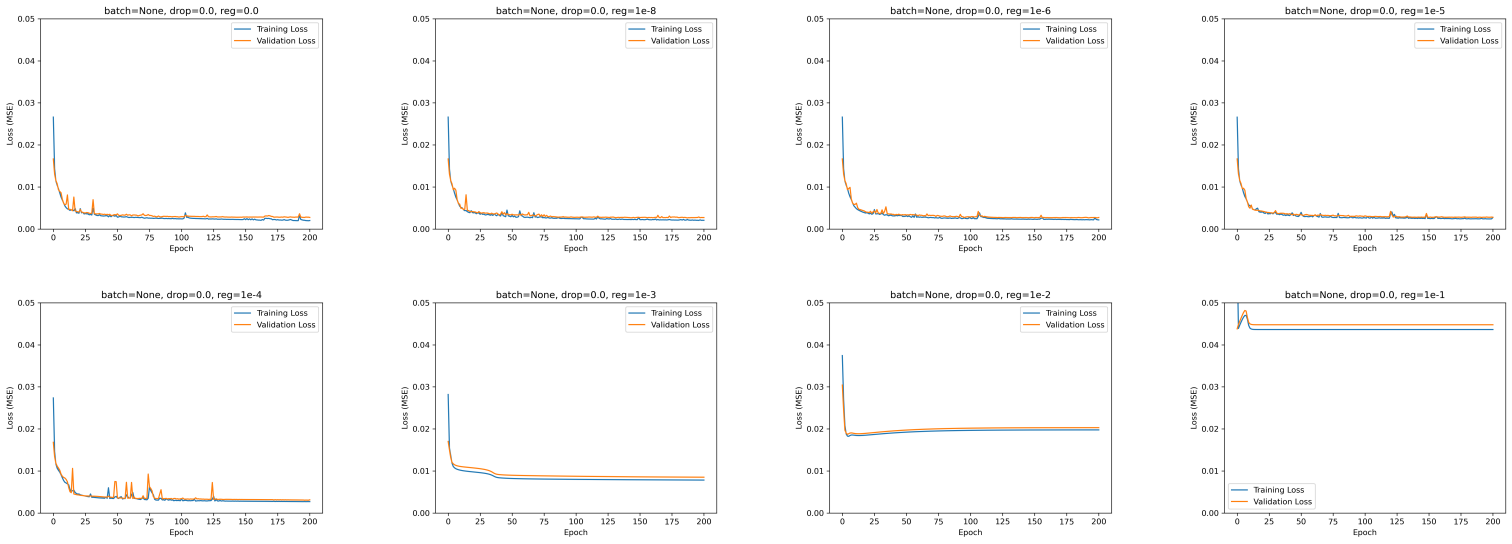

Figure 19: Training Plots of CNN-LSTM Model with Increasing Weight Regularization Rates. From Left to Right: (Top) 0, $1 * 10^{-8}$, $1 * 10^{-6}$, $1 * 10^{-5}$ (Bottom) $1 * 10^{-4}$, $1 * 10^{-3}$, $1 * 10^{-2}$, $1 * 10^{-1}$

Table 11: Final Losses for Weight Regularization Hyperparameters

Regularization Rate	Training Loss	Validation Loss
0	0.002004	0.002747
$1x10^{-8}$	0.002061	0.002716
$1x10^{-6}$	0.002155	0.002730
$1x10^{-5}$	0.002755	0.002785
$1x10^{-4}$	0.002704	0.003127
$1x10^{-3}$	0.007822	0.008506
$1x10^{-2}$	0.019760	0.020293
$1x10^{-1}$	0.043634	0.044770

Considering the total set of combinations, the top-performing models (based on validation loss) are summarized in Table 12. With consideration from the trends described above and the proposed parameters in the table, optimal hyperparameters for future training are selected as follows: no batch normalization will be incorporated, and a Dropout rate of 0.01% and Weight Regularization rate of $1 * 10^{-8}$ will be incorporated.

Table 12: Overfit Hyperparameter Tuning Attempt: Best Models by Validation Loss

Training Loss	Validation Loss	Dropout Rate	Weight Regularization	Batch Normalization
0.002607	0.002598	0.1%	0.0	None
0.002105	0.002633	0.01%	$1x10^{-8}$	None
0.002061	0.002716	0.0%	$1x10^{-8}$	None
0.002155	0.002730	0.0%	$1x10^{-6}$	None
0.002153	0.002745	0.1%	$1x10^{-6}$	None
0.0.002004	0.002747	0.0%	0.0	None

C Model Tuning: Weather Extraction Feature Sizes

This subsection begins the study of hyperparameters, specifically for tailoring them to the performance of different models. For this task, the dimensionality of each weather extraction mechanisms is varied. In a convolutional network, this can simply be accomplished by varying the number of filters, or channels, associated with a convolutional layer. Because attention mechanisms do not have this sort of parameter, the dimensionality is altered directly, assigning

a data dimension equivalent to the number of parameters that would be present in a comparable convolutional layer.

For this task, two models are considered: CNN-LSTM and SA-LSTM; it is assumed that the extraction channel configurations may be transferred to other models, regardless of recurrence type. The dimensionality of each model's hidden and output weather extraction layers were varied between 1 and 32 channels (or a multiple thereof, for attention layers), with all other parameters matching those described in Table 3; note one exception: due to a coding error, the LSTM input size for the SA-LSTM model was set to 10, instead of 6. Each model was sampled 200 times, to approximate the total set of combinations; samples were each trained for 50 epochs. Finally, in addition to prior efforts to insure repeatability, each model weight and bias was set to 0.5.

To present a clearer picture and eliminate outliers, Figure 20 presents a 2D scatter plot of the final training and validation losses with all varied channel depths, limiting the maximum visible loss to .005; a complete set of plots are available in Appendix I. From the illustrated plots, general regions of useful depths can be defined. Within the training 2D plot, a large segment is visible, where the hidden depth (Ch 2) requires a minimum of 9 and the output depth (Ch 3) requires a minimum of 15. However, this region is significantly trimmed in the validation loss plot, whose useful region requires a minimum hidden size of 25, and minimum output size of 20.

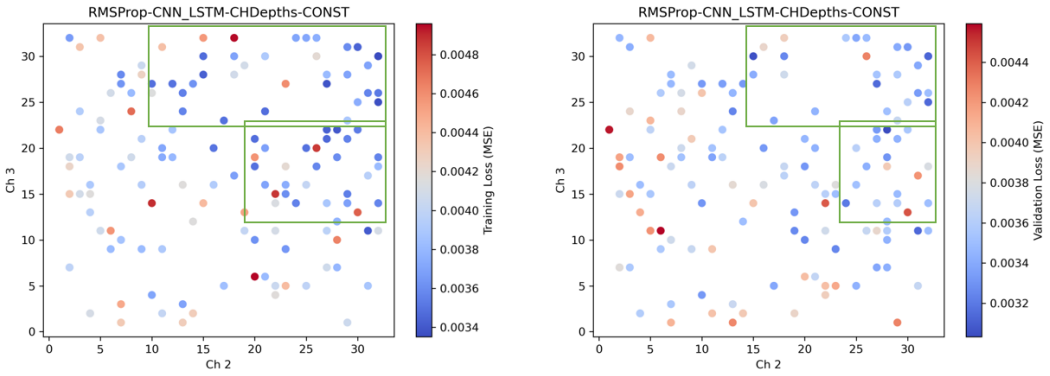

Figure 20: Training (left) and Validation (right) Scatter Plots of Training Losses Associated with Channel Depth Combinations, Limited to Losses no Greater Than 0.005

Examining the best training and validation loss models in Tables 13 and 14 identifies that optimal depths of 12 for both hidden and output layers. However, this value is toward the outermost region of the identified trends, and may not provide the best capability over larger data generalizations and more extensive training. Because of this sample being considered an outlier, and because the next-best model increasing validation loss by $2*10^{-5}$, hidden and output convolutional filter sizes of 28 and 22 were selected, respectively.

Table 13: Best Training Results for Varied Channel Depths of CNN-LSTM Network

Training Loss	Validation Loss	Epoch Time	Hidden Channels	Output Channels
0.003351	0.003127	16.90	32	30
0.003351	0.003127	17.22	32	30
0.003365	0.003141	17.19	32	25
0.003386	0.003146	17.09	27	27
0.003397	0.003033	16.57	28	22

Table 14: Best Validation Results for Varied Channel Depths of CNN-LSTM Network

Training Loss	Validation Loss	Epoch Time	Hidden Channels	Output Channels
0.003397	0.003033	16.57	28	22
0.003472	0.003053	19.83	15	30
0.003351	0.003127	16.90	32	30
0.003351	0.003127	17.22	32	30
0.003365	0.003141	17.19	32	25

Once again, scatter plots for the final training and validation loss as a function of each model's channel sizes are generated for the SA-LSTM model; because of the model's greater general accuracy, the loss limitation was instead placed at .004, as seen in Figure 21. The best-performing models are much more scattered for self-attention. While large hidden and output dimensions are beneficial in training, they appear to facilitate overfitting in the validation dataset; instead, optimal channel depths consider a much smaller output size, as marked in the regions.

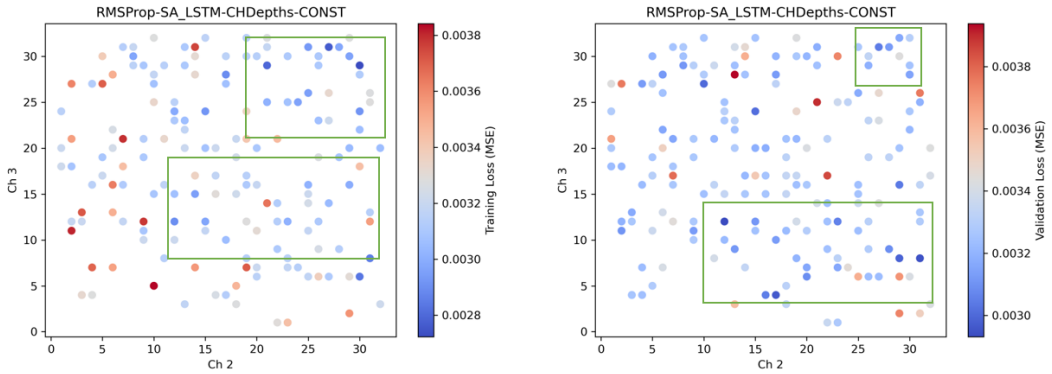

Figure 21: Training (left) and Validation (right) Scatter Plots of Training Losses Associated with Channel Depth Combinations, Limited to Losses no Greater Than 0.004

Considering the final loss values in Tables 15 and 16, a minimum validation loss is achieved with channel sizes os 31 and 8 for the hidden and output layers, respectively. Because this value falls within the visualized trends, and no other considerations are apparent, these channel sizes are selected for future attention-based models.

Table 15: Best Training Results for Varied Channel Depths of SA-LSTM Network

Training Loss	Validation Loss	Epoch Time	Hidden Channels	Output Channels
0.002723	0.003253	22.63	30	25
0.002763	0.003048	20.71	27	31
0.002793	0.003243	22.85	21	29
0.002836	0.003418	20.25	30	6
0.002856	0.002950	20.54	31	8

Table 16: Best Validation Results for Varied Channel Depths of SA-LSTM Network

Training Loss	Validation Loss	Epoch Time	Hidden Channels	Output Channels
0.002898	0.002932	20.62	12	12
0.002856	0.002950	20.54	31	8
0.003131	0.002965	20.26	17	4
0.004426	0.002990	20.36	29	8
0.003077	0.003006	20.04	10	27

D Model Tuning: Recurrent Hidden Layer Sizes v. Depths

This subsection presents the efforts of tuning RNN hyperparameters, which include RNN input size, number of layers, and the number of hidden states within each layer. The experiments to tune these parameters will primarily focus on the layers and hidden size, where both will be varied for each model and losses reported in scatter plots - much like with selection of feature extraction dimensionality. Instead of drawing 200 samples at random, a set of 80 samples are pre-defined using a grid search: networks will consider 1 to 4 recurrent layers, each with a common hidden state size (in increments of 50) between 50 and 1000. Because the RNN has the largest impact on complexity and training time, both a low number of samples and a smaller number of epochs (20) were used for this experiment.

Recall that (in Section A) an input size of 10 had been previously identified as preferred for recurrent neural networks over the prior default of 6. However, the experiments of this subsection were conducted with both, to validate the assumed performance improvements. Across all models, the smaller input size was preferred. For all but one experiment (CNN-LSTM), the minimum achievable loss was greater. While the larger input provided some relief in model complexity for the achievable results (particularly with the hidden size of LSTM models), this does not supersede the loss in accuracy. For brevity, figures with input size of 10 are provided in Appendix II.

Originally, only three models were considered for these experiments: CNN-LSTM, CNN-GRU, and SA-LSTM. A fourth model (SA-GRU) was added after the first three were initially executed, as the changes in desirable RNN parameters varied greatly - both when switching from LSTM to GRUs, and switching between CNNs and Self-Attention.

The optimal RNN parameters for each of the four model types is summarized in Table 17, and examined more closely below.

Table 17: Final Selections for RNN Hyperparameters

Model Type	Hidden Extraction Channel Size	Output Extraction Channel Size	RNN Input	RNN Hidden Size	RNN Depth
CNN-LSTM	28	22	6	1	1000
SA-LSTM	31	8	6	2	600
CNN-GRU	28	22	6	1	650
SA-GRU	31	8	6	2	600

D.1 CNN-LSTM Running the grid search across CNN-LSTM samples yielded the set of scatter plots in Appendix II. To minimize clutter, only a 2D view, limited to errors at or below 0.01 are presented in Figure 22. From these figures, it can be observed that this model benefits most from LSTM with either a large hidden size or multiple layers, occasionally both; this echoes initial trends from A.

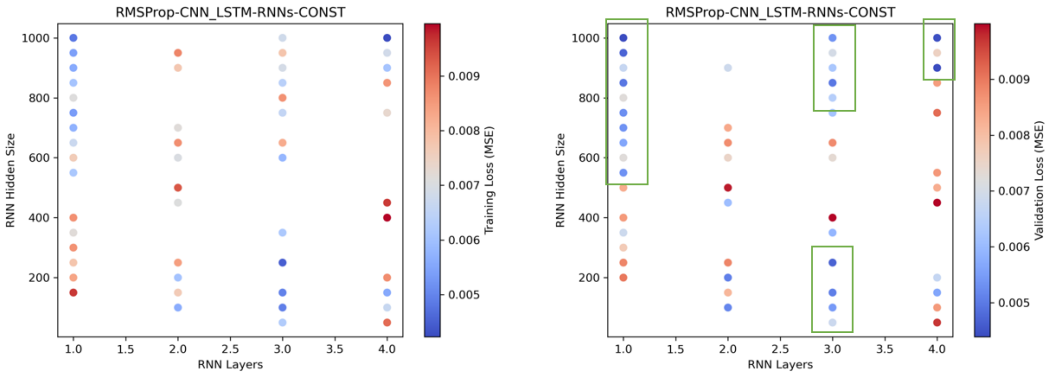

Figure 22: Training (left) and Validation (right) Scatter Plots of Training Losses Associated with CNN-LSTM RNN Parameter Combinations, Limited to Losses no Greater Than 0.01

Comparing these figures with the table of top-performing models (by validation loss) in Tables 18 and 19, it becomes apparent that a larger hidden state size has greater impact on accuracy. While the best-performing model

incorporates both a large hidden size and large depth, this utilizes more resources than can be reasonably provided - resulting in an epoch training time close of 64 seconds. To provide the greatest accuracy without significantly impacting training time, a single layer of LSTM with hidden size of 1000 is selected.

Table 18: Best Training Results for Varied RNN Hyperparameters of CNN-LSTM Network

Training Loss	Validation Loss	Epoch Time	RNN Depth	RNN Hidden Size
0.004225	0.004442	71.69	4	1000
0.004474	0.004683	24.16	3	250
0.004811	0.004389	23.24	1	1000
0.004933	0.005441	25.89	3	100
0.004951	0.005050	37.04	3	150

Table 19: Best Validation Results for Varied RNN Hyperparameters of CNN-LSTM Network

Training Loss	Validation Loss	Epoch Time	RNN Depth	RNN Hidden Size
0.006081	0.004387	64.82	4	900
0.004811	0.004389	23.24	1	1000
0.004225	0.004442	71.69	4	1000
0.004474	0.004683	24.16	3	250
0.005408	0.004701	22.54	1	950

D.2 SA-LSTM Considering the results for LSTM models with self-attention for an extraction layer provides a cleaner selection process. The 2D scatter plots presented in Figure 23 only consider samples with losses at or below 0.01; from this, we can observe that fewer models retain a useful accuracy as LSTM depth increases. Additionally, the best visible accuracies still require a moderate-to-large hidden layer size.

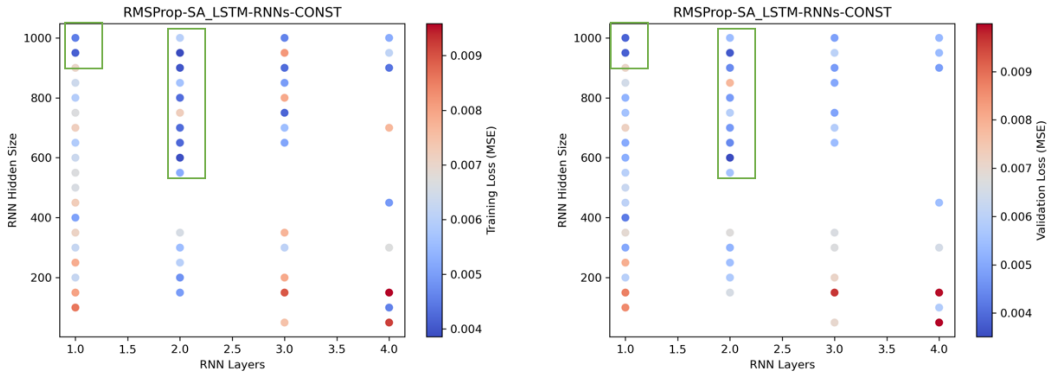

Figure 23: Training (left) and Validation (right) Scatter Plots of Training Losses Associated with SA-LSTM RNN Parameter Combinations, Limited to Losses no Greater Than 0.01

The results listed in Tables 20 and 21 tend to validate the previously-observed trends. As a result, a hidden size of 600 and depth of 2 layers was selected for SA-LSTM models. This reflects a capable and useful model; this parameter selection relays a balance of strong validation and training accuracy, while also not requiring significant resources (inferred from the training epoch time of 32 seconds).

Table 20: Best Training Results for Varied RNN Hyperparameters of SA-LSTM Network

Training Loss	Validation Loss	Epoch Time	RNN Depth	RNN Hidden Size
0.003858	0.003735	39.57	2	950
0.003910	0.003508	31.72	2	600
0.003994	0.004477	37.97	2	900
0.004147	0.003829	26.92	1	950
0.004197	0.004902	45.75	3	750

Table 21: Best Validation Results for Varied RNN Hyperparameters of SA-LSTM Network

Training Loss	Validation Loss	Epoch Time	RNN Depth	RNN Hidden Size
0.003910	0.003508	31.72	2	600
0.003858	0.003735	39.57	2	950
0.004147	0.003829	26.92	1	950
0.004535	0.003867	25.67	1	1000
0.005072	0.004236	27.87	1	400

D.3 CNN-GRU The results from testing CNN-GRU models have notable trends. Most apparently in the limited-view scatter plots of Figure 24 is the lack of effectively-trained models; roughly 25 models were able to achieve a final training loss that surpassed the limit of 0.01. Of those that did meet the threshold, most models required only one layer of recurrence and a moderate hidden-state size; in comparison to CNN-LSTM, GRU can achieve a compromised degree of accuracy for this specific task, albeit with a much more resource- and time-efficient model. With a longer model training time, this accuracy disparity may become negligible.

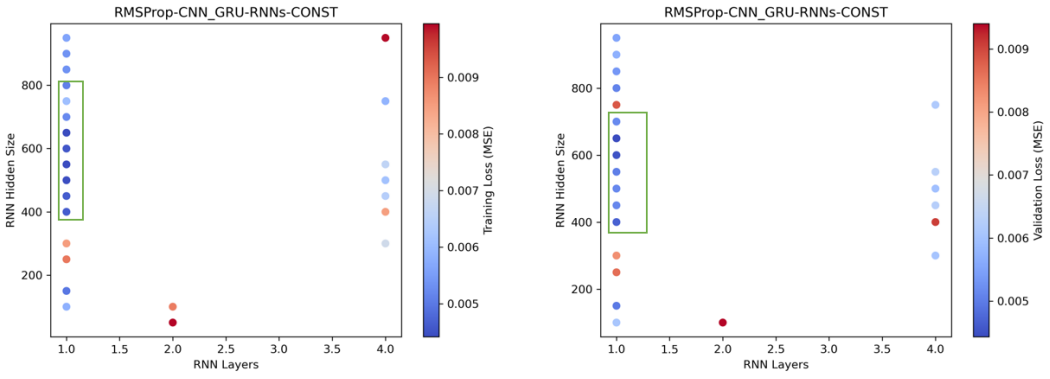

Figure 24: Training (left) and Validation (right) Scatter Plots of Training Losses Associated with CNN-GRU RNN Parameter Combinations, Limited to Losses no Greater Than 0.01

Considering the final loss values in Tables 22 and 23, most apparent samples validate the above trends, with exception for a single layer of 150 GRU cells. Going forward, a single GRU layer of size 650 will be used in

CNN-GRU models. This selection provided the best validation loss of the samples displayed; furthermore, the sample remained within the understood trends of this model type, and did not use an excess of computing resources.

Table 22: Best Training Results for Varied RNN Hyperparameters of CNN-GRU Network

Training Loss	Validation Loss	Epoch Time	RNN Depth	RNN Hidden Size
0.004417	0.004434	16.01	1	650
0.004432	0.005020	24.08	1	550
0.004446	0.005145	26.19	1	500
0.004549	0.004528	15.56	1	600
0.004613	0.005121	15.75	1	450

Table 23: Best Validation Results for Varied RNN Hyperparameters of CNN-GRU Network

Training Loss	Validation Loss	Epoch Time	RNN Depth	RNN Hidden Size
0.004417	0.004434	16.01	1	650
0.004549	0.004528	15.56	1	600
0.004632	0.004673	16.48	1	400
0.004959	0.004936	15.34	1	150
0.004432	0.005020	24.08	1	550

D.4 SA-GRU When utilizing self-attention to extract weather behaviors, the paradigm of useful GRU features shifts significantly. As seen in figure 25, an increase in models meeting the threshold loss can be noted; for training, 37 models achieved a loss no greater than 0.01. Additionally, the region of desirable models has shifted almost exclusively to the newly-visible set, whose recurrent depth is 2 and holds greatest interest in moderate hidden sizes.

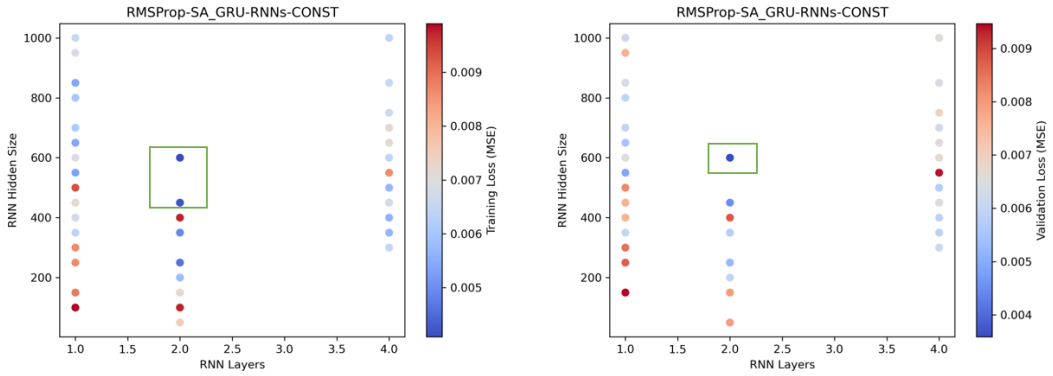

Figure 25: Training (left) and Validation (right) Scatter Plots of Training Losses Associated with SA-GRU RNN Parameter Combinations, Limited to Losses no Greater Than 0.01

Considering the finalized losses in Tables 24 and 25, a recurrent depth of 2 layers and hidden size of 600 was selected; coincidentally, this is identical to the hyperparameters for SA-LSTM. This sample provides the best training and validation loss, and remains within the limited trends for this model type. The compute utilization remains somewhat negligible, with an epoch training time of roughly 23 seconds.

Table 24: Best Training Results for Varied RNN Hyperparameters of SA-GRU Network

Training Loss	Validation Loss	Epoch Time	RNN Depth	RNN Hidden Size
0.004084	0.003586	23.45	2	600
0.004188	0.004547	23.67	2	450
0.004615	0.005259	23.00	2	250
0.004905	0.005755	25.04	2	350
0.005350	0.004908	24.88	1	550

Table 25: Best Validation Results for Varied RNN Hyperparameters of SA-GRU Network

Training Loss	Validation Loss	Epoch Time	RNN Depth	RNN Hidden Size
0.004084	0.003586	23.45	2	600
0.004188	0.004547	23.67	2	450
0.005350	0.004908	24.88	1	550
0.004615	0.005259	23.00	2	250
0.005382	0.005474	19.43	1	650

E Model Tuning: Optimizer Selection

Expanding on prior results, the question of an appropriate optimizer is revisited. For overfitting and hyperparameter tuning, RMSProp had been employed, as it significantly exacerbated the overfitting challenge. This section briefly considers RMSProp's usefulness against the presumed default optimizer, Adam. Due to time constraints, a learning rate of $2 * 10^{-4}$ is assumed to be useful. Each tuned model type is considered, with optimal parameters listed in Table 26. Each model is again initialized with all parameters set to 0.5, and trained for 50 epochs.

Table 26: Final Selections for Hyperparameters

Model Type	Hidden Extraction Channel Size	Output Extraction Channel Size	RNN Input	RNN Hidden Size	RNN Depth	Optimizer
CNN-LSTM	28	22	6	1	1000	RMSProp
SA-LSTM	31	8	6	2	600	Adam
CNN-GRU	28	22	6	1	650	RMSProp
SA-GRU	31	8	6	2	600	RMSProp

For all but one model type (SA-LSTM),both training and validation model performance remained better with the use of RMSProp as opposed to Adam; CNN-LSTM in particular reflects this, as Adam was unable to generate a convergent model whatsoever. no doubt, the preference for RMSProp is at least partially a reflection of the optimizer's use throughout parameter tuning. The training results for each optimizer are listed in Figure 26 for CNN-LSTM; Figure 27 for SA-LSTM; Figure 28 for CNN-GRU; and Figure 29 for SA-GRU.

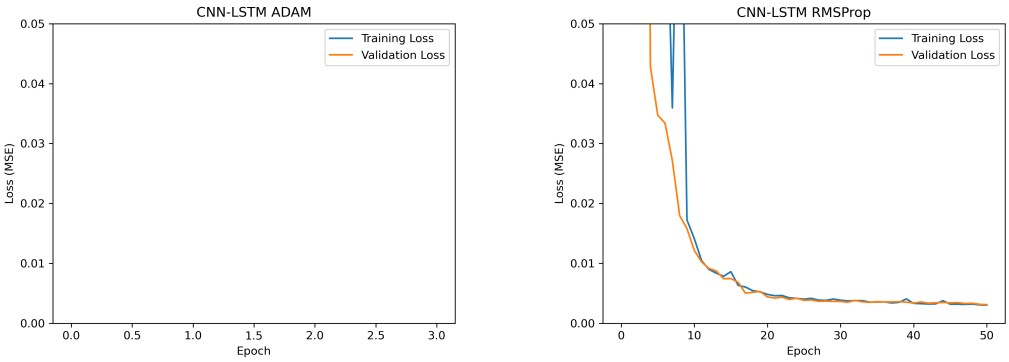

Figure 26: CNN-LSTM Training Plots using Adam (left) and RMSProp (right) Optimizers. Note that Adam Does Not Converge for the Given Model

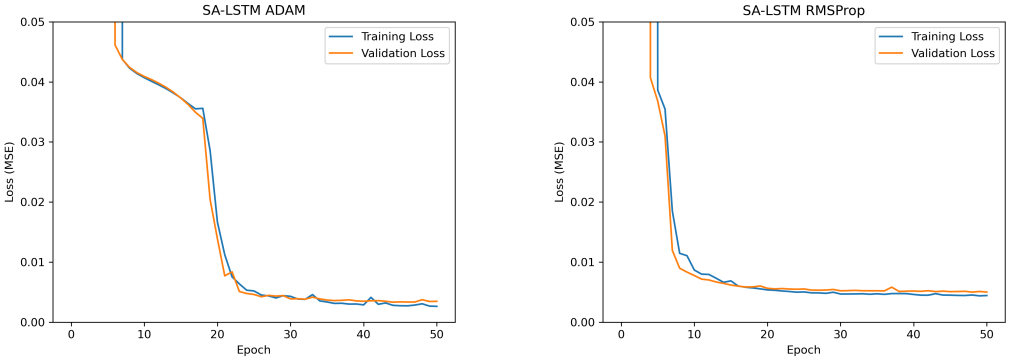

Figure 27: SA-LSTM Training Plots using Adam (left) and RMSProp (right) Optimizers

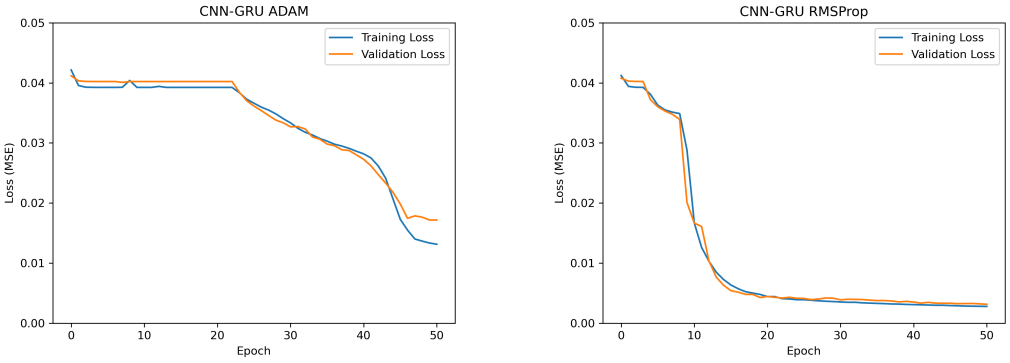

Figure 28: CNN-GRU Training Plots using Adam (left) and RMSProp (right) Optimizers

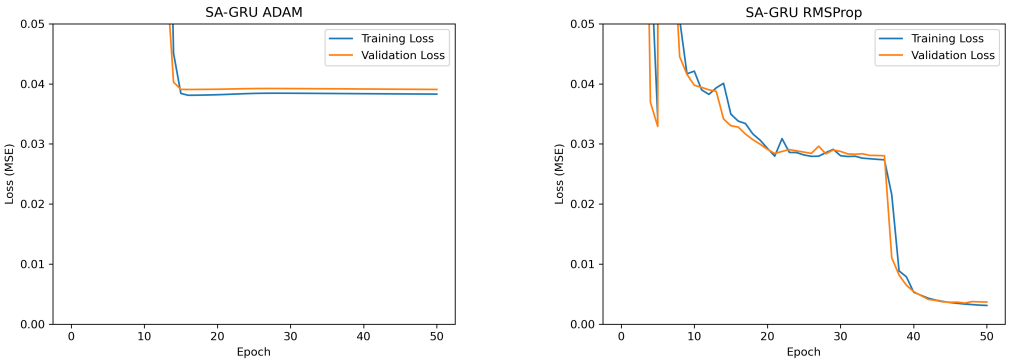

Figure 29: SA-GRU Training Plots using Adam (left) and RMSProp (right) Optimizers

From a brief discussion in [21], it is noted that adaptive optimizers such as RMSProp and Adam may poorly generalize models; the experiments conducted in [21] all are performed over hundreds of epochs, such that there are more optimizer steps than there are model parameters. As a result, it is hypothesized their findings may be applicable for this task. Over the course of 300 epochs, the tuned CNN-LSTM model was initialized and trained with a set of optimizers; Adam is included twice in these results, once of which without a weight regularization penalty, as a result of a coding error. For

each optimizer, 5 models are trained with random parameter initializations; each training session and the average of each optimizer is plotted in Figure 30. Due to a limited amount of time, default learning rates are assumed for each optimizer based on those present in Pytorch and Keras, and the weight decay selected in subsection B is transferred to each optimizer; the selected learning rates are listed in Table 27.

Table 27: Default Parameters of Tested Optimizers

Optimizer	Learning Rate	Weight Regularization	Momentum
SGD	0.01	$1 * 10^{-6}$	0.0
SGD+Momentum	0.01	$1 * 10^{-6}$	0.5
SGD+Nesterov	0.01	$1 * 10^{-6}$	0.5
Adam	0.001	$1 * 10^{-6}$	N/A
Adam with Initial Parameters	0.001	0.0	N/A
Adadelta	1.0	$1 * 10^{-6}$	N/A
Adagrad	0.01	$1 * 10^{-6}$	N/A
RMSProp	0.0001	$1 * 10^{-6}$	N/A

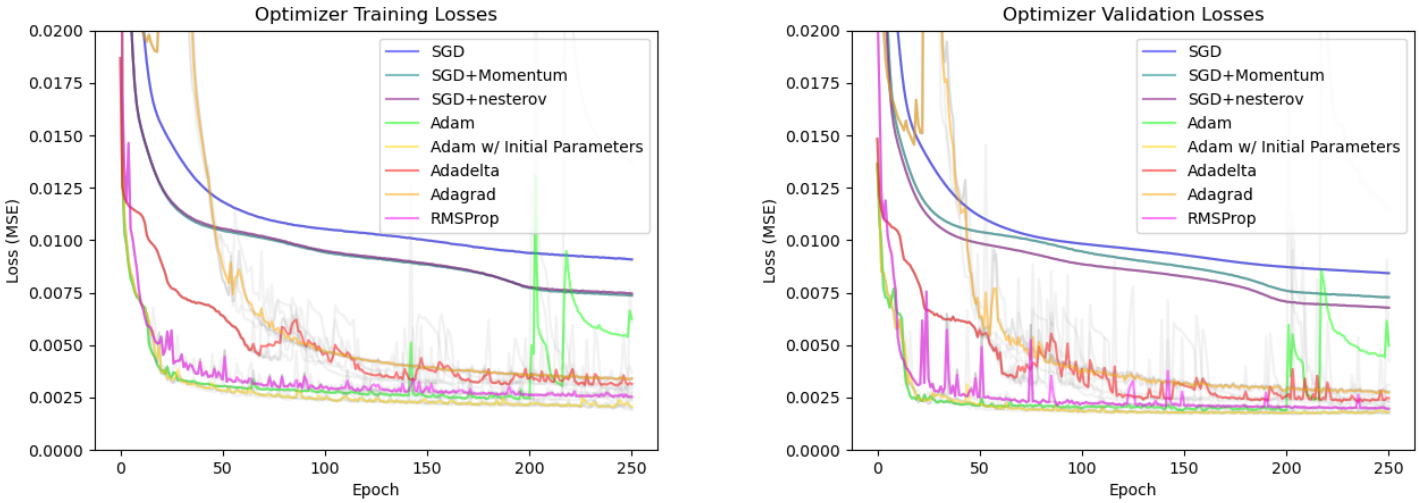

Figure 30: Training (left) and Validation (right) Losses of Optimizers with Default Parameters for Tuned CNN-LSTM Model

From these results, it is noted that adaptive optimizers are likely to remain useful in this task. It is possible Stochastic Gradient Descent (SGD) and its momentum variants (SGD+momentum, SGD+nesterov) may improve in accuracy with selection or scheduling of learning rates, as well as with additional training epochs; however, these possibilities could not be further explored. The selected optimizer is used in the revisited tuning results of Subsection C.

6 Final Model Evaluations

After completing all model tuning for the four models in 26, a series of finalized model trainings and evaluations is conducted. This section considers several distinct evaluations to improve performance: re-training sets of default models with the parameters in Table 3, this time using the generalized two-week data, to illustrate the effect of generalizing model data; and training the sets of tuned models from Table 26 on generalized two-week data, in order to illustrate the effects of a careful hyperparameter selection. Results in this section are briefly described in terms of mean and standard deviation of trajectory errors; for a more comprehensive view of the model assessments, see Appendix III.

A Data Generalization Results

For this comparison, three models (CNN-LSTM, SA-LSTM, CNN-GRU) are considered with the associated default parameters described in Table 3. These models are trained with the 734 flights collected in the generalized two-week flight data over 500 epochs and using a 4-fold cross validation. From this, the trajectorywise error results are listed in Table 28. This table lists error for each model in two separate forms: one, which considers all flights in the validation set, and one which considers only the subset of flights from New York to Los Angeles within each validation set. Any available, comparable error from the initial training in 4 are listed alongside these errors, as well as percent improvements over the initial training.

Table 28: Summary of Data Generalization Model Performance

Case	Initial Horizontal Error (μ/σ in nmi)	Initial Vertical Error (μ/σ in ft)	Case Horizontal Error (μ/σ in nmi)	Case Vertical Error (μ/σ in ft)	Improvement over Initial Horiz. Error (μ/σ as percent)	Improvement over Initial Vertical Error (μ/σ as percent)
CNN-LSTM	**63.5584**	**1160.27**	**48.5837**	**1525.08**	**23.561**	**-31.442**
Total Data	26.8905	1500.83	63.1849	2121.23	-134.971	-41.337
CNN-LSTM	**63.5584**	**1160.27**	**29.5634**	**711.24**	**53.486**	**38.700**
KJFK-KLAX	26.8905	1500.83	67.8611	1673.58	-152.361	-11.510
CNN-GRU	**59.8954**	**1120.04**	**44.5988**	**1486.57**	**25.539**	**-32.725**
Total Data	28.0559	1399.75	40.8868	2032.41	-45.733	-45.198
CNN-GRU	**59.8954**	**1120.04**	**26.3954**	**686.36**	**55.931**	**38.720**
KJFK-KLAX	28.0559	1399.75	46.1633	1597.22	-64.541	-14.107
SA-LSTM	**40.9453**	**804.73**	**53.1642**	**1565.15**	**-29.842**	**-94.493**
Total Data	23.7972	1054.89	67.7009	2183.16	-184.49	-106.956
SA-LSTM	**40.9453**	**804.73**	**33.4991**	**732.47**	**18.186**	**8.980**
KJFK-KLAX	23.7972	1054.89	72.7729	1708.79	-205.804	-61.987

From the table, several trends can be noted. In general, training with identical model hyperparameters over the generalized flight dataset yielded models which were more accurate in terms of average error, but varied in accuracy significantly more than initial models. This is especially noted in comparisons of horizontal error, though the trend is also present when comparing model evaluation over the restricted set of flight between New York and Los Angeles.

For all models trained with the generalized data set, their vertical error was notably worse when evaluated with the total set of flights than with the subset of flights. This is certainly a reflection of the nature of that subset. Recalling from Table 8, flights from New York to Los Angeles are the longest duration considered at over 5 hours; as a result, they have the longest flight duration at a cruising altitude, where a model is most likely to accurately predict the flight's elevation. Since the evaluation of the total set of flights considers aircraft spending more of their route in a climb or descent, each model's shortcomings in predicting altitude are heavily depicted. It should be noted that these shortcomings may be a reflection of naive data preprocessing approaches, which do not accurately represent the climb and descent of an aircraft when generating a 4D flight plan.

Generally speaking, SA-LSTM is somewhat of an outlier in the results presented. The SA-LSTM model trained with generalized flight data failed both to maintain a relative standing as the best model of those tested with a given set of data, as well as to improve on its accuracy in a meaningful over the initial SA-LSTM model. While it is true that the model improves in mean horizontal error once evaluations are restricted to flights from KJFK to KLAX, the variance in error notably large. It is believed that the shortcomings of SA-LSTM are a reflection of the assumed model hyperparameters. Self-attention parameters were not available from prior research, nor thoroughly tested; as a result, it is both possible and likely that the self-attention layers lacked the necessary complexity to generalize for this task.

It is hypothesized that the increase in error variance is a reflection of the model's inability to generalize along with the newly-available data. Assumptions for default hyperparameters were made based on the model in [2], which focused on a convolutional-recurrent design for flights from New York to Lose Angeles exclusively; as a result, both convolutional-recurrent designs (CNN-LSTM, CNN-GRU) may be hindered by limitations in network size, while these limitations are further exacerbated with models utilizing self-attention. A supplemental reason for the variance still assumes the models faced challenges with generalization, but specifically considers the selection of optimizer as a reason for this challenge. Discussions in [21] indicate that the use of adaptive optimizers such as RMSProp and Adam may prevent models from effectively generalizing to support new data, particularly for smaller data sizes such as this situation.

B Model Tuning Results

For this comparison, the models using tuned hyperparameters in Table 26 are trained with the two-week generalized data over 500 epochs, also with a 4-fold cross-validation. Their trajectorywise errors are reported in Table 29, with reference to the errors of the complete validation sets listed in Table 28.

Table 29: Summary of Initial Model Tuning Performance

Model	Default Horizontal Error ($\boldsymbol{\mu}/\sigma$ in nmi)	Default Vertical Error ($\boldsymbol{\mu}/\sigma$ in ft)	Tuned Horizontal Error ($\boldsymbol{\mu}/\sigma$ in nmi)	Tuned Vertical Error ($\boldsymbol{\mu}/\sigma$ in ft)	Improvement over Default Horiz. Error ($\boldsymbol{\mu}/\sigma$ as percent)	Improvement over Default Vertical Error ($\boldsymbol{\mu}/\sigma$ as percent)
CNN-LSTM	**48.5837**	**1525.08**	**55.9422**	**1531.31**	**-15.146**	**-0.408**
	63.1848	2121.23	40.8471	1973.87	35.353	6.947
CNN-GRU	**44.5988**	**1486.57**	**56.6427**	**1575.88**	**-27.005**	**-6.008**
	40.8868	2032.41	39.0383	1973.20	4.521	2.913
SA-LSTM	**53.1642**	**1565.15**	**47.3686**	**1722.35**	**10.901**	**-10.044**
	67.7009	2183.16	34.4715	2041.98	41.697	6.466
SA-GRU	**46.6859**	**1646.01**	**NaN**	**NaN**	**NaN**	**NaN**
	44.1792	2232.96	NaN	NaN	NaN	NaN

Before discussing the results in greater detail, an experimental error should be noted. For this task, it was intended that all models would be initialized with weights and biases set to 0.5. However, this yielded results with significant error; most notably, the final model in Table 29 (SA-GRU) was unable to converge to a useful model implementation. Due to a limitation of time, only one set of models (those using the default hyperparameters in Table 3) was able to be re-trained with a random initialization; as a result, it is likely that the tuned model results represent a poor implementation for this task.

Even so, the tuned models indicate their ability to address the main concern previously encountered: a poor ability to generalize, resulting in large error variances. For all models and each error, the standard deviation was able to be reduced, even if marginally. Were the models retrained with a random initialization, it is expected that the tuned models would find significant improvements in error variance.

Special concern should be given to SA-LSTM, as the only model which can represent the possibilty of improvements in the use of self-attention for a more generalized dataset. While the model's mean horizontal error (47.4 nmi) does not improve over the initial SA-LSTM's accuracy, it does improve over that presented by the generalized model; furthermore, the accuracy is in-line with and outperforming that of other tuned models considered in this experiment. From this, it can be assumed that the parameters for self-attention have been meaningfully prescribed for this task, and that self-attention still remains a

viable and competitive deep learning mechanism for improving trajectory prediction.

C Revisited Model Tuning

After disabling the constant parameter initialization and selecting an optimizer based on a larger collection of tests, the four models were re-trained with otherwise identical setup to the prior tuning comparison: models were trained over 500 epochs with a four-fold cross-validation using the generalized flight data over a 2-week period. This time, each model was trained using the Adam optimizer, with a learning rate of 0.001 and no weight regularization.

Table 30: Summary of Final Model Tuning Performance

Model	Default Horizontal Error (μ/σ in nmi)	Default Vertical Error (μ/σ in ft)	Tuned Horizontal Error (μ/σ in nmi)	Tuned Vertical Error (μ/σ in ft)	Improvement over Default Horiz. Error (μ/σ as percent)	Improvement over Default Vertical Error (μ/σ as percent)
CNN-LSTM	**48.5837**	**1525.09**	**47.9694**	**1489.36**	**1.170**	**2.343**
	63.1848	2121.23	36.8354	1985.36	41.702	6.405
CNN-GRU	**44.5988**	**1486.57**	**NaN**	**NaN**	**NaN**	**NaN**
	40.8868	2032.41	NaN	NaN	NaN	NaN
SA-LSTM	**53.1642**	**1565.15**	**48.0979**	**1424.11**	**9.530**	**9.011**
	67.7009	2183.16	36.6018	1830.96	45.936	16.133
SA-GRU	**46.6859**	**1646.01**	**NaN**	**NaN**	**NaN**	**NaN**
	44.1792	2232.96	NaN	NaN	NaN	NaN

The results presented in Table 30 show some improvement, with notable limitations. neither GRU-based model could consistently converge; this is likely a reflection of the choice of optimizer and learning rate, which may have allowed for rapid changes to the model, causing an exploding gradient problem. From the LSTM models that did converge, however, accuracies are somewhat more in-line with desired results: CNN-LSTM was able to improve its horizontal error (7.97 and 4.01 nmi in mean and standard deviation, respectively), as well as SA-LSTM notably (298 and 211 ft in mean and standard deviation). Even so, these improvements are marginal; initial results in Table 4 found models capable of predicting with desirable accuracy - particularly SA-LSTM. Additional optimizer tuning will likely reduce error,

specifically horizontal error. Large vertical error may be a reflection of poor flight plan processing, as previously referenced in subsection A.

V FUTURE WORK

A number of extensions are relevant to the task of trajectory prediction, and are highlighted in this section.

1 Trajectory Prediction Refinement

Needless to say, this work on 4D trajectory prediction is not comprehensive. Additional efforts may include further weather data assessment and data generalization, as well as further attempts to tune and improve architecture design.

Regarding data: this body of work predominantly focused on a two-week period of flights, regardless of whether the collection of flights was generalized. Selection of flight and weather data should consider varied time periods, covering different seasons within the year as well as the same date ranges from prior years, to insure the model is trained for a more general set of weather patterns. As of the submission of this thesis, processing was completed for the generalized set of 5 flights over 100 days (November 1st, 2018 through February 5th, 2019), but could not be used in training as a function of time.

Additionally, other weather products and combinations may still be considered; of particular interest is air pressure, as it is closely related to temperature and air movements, and consequently may reflect wind turbulence between altitudes. Combinations should be revisited, particularly with the gained knowledge of model reproducibility, to re-assess the accuracy of weather products and combinations.

Because the airspace has far more combinations of departure and arrival locations than could realistically be incorporated into this dataset, it is important to accentuate the possibility of unknown flight routes. In future research, a number of departure/arrival pairs should be included in validation data - and excluded from training data - to better represent this challenge.

A final note on data: as experiments with data generalization were conducted, the flaws in simplistic flight plan developments became apparent - especially in representing altitude, climb, and descent. Two approaches should be considered to alleviate this challenge. A preferred approach would be the coordination to understand systems behavior that dictates climb and descent phases of a flight; this knowledge should be incorporated into pre-processing, if possible. However, this may be impractical due to knowledge

accessibility or complexity; it may be appropriate to limit the scope of trajectory prediction to the duration of a flight at cruising altitude.

From an architecture standpoint, several parameters may be useful which were not considered in the scope of this thesis. As mentioned in B, normalization should be revisited, as other scopes of normalization may prove useful for this task. Additionally, in-model preprocessing techniques may be useful, such as incorporating matrix decomposition into weather behavior extraction in order to efficiently locate and learn patterns in weather data. On the aspect of generalization, attention mechanisms may still be explored; in computer vision, they are used to identify regions of interest within an image, to reduce the computation performed by a model when recognizing images; this principle might be applied to the gridded data of the continental United States, potentially allowing the prediction model to be generalized for multiple flights without scaling memory usage associated with each flights' set of feature cubes. A more in-depth assessment of optimizer choice and tuning is also worth considering, especially given the intended number of training epochs and the possibility that adaptive optimizers may poorly generalize the model [21]. Finally, tuning may be revisited with algorithmic optimizers, particularly Bayesian-assisted techniques such as Bayesian-Optimized HyperBand, which may learn and adjust hyperparameters throughout a training process [22].

2 Applying Trajectory Prediction

As referenced in Section I, trajectory prediction is seen as a cornerstone to providing data inputs for spectrum allocation. In current research, it is hoped that an accurate 4D trajectory may help to localize flights within airspace sectors (identifying what resources are available for assignment), as well as estimating path loss from standard equations for air-to-ground communications [23].

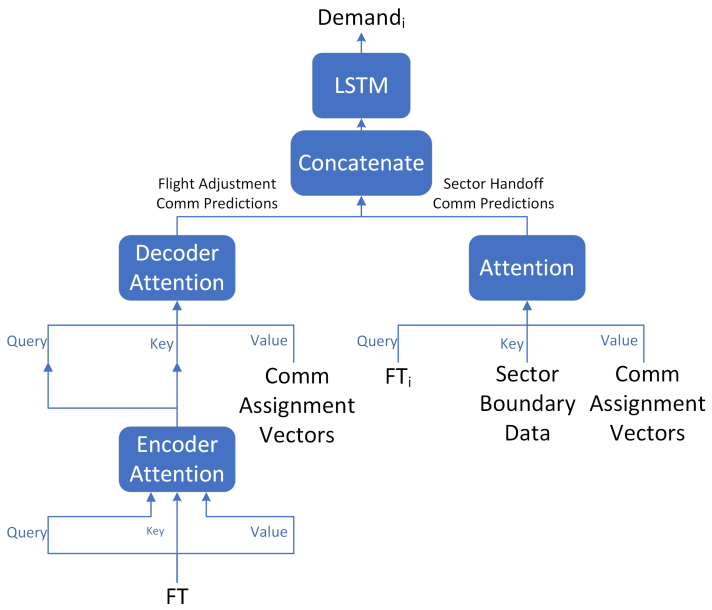

Figure 31: Notional Architecture for Communication Demand Prediction

Of greatest consequence, though, would be using an accurate trajectory predictions to infer communication demand. Limited communication data are available for this task, however a set of rules are generally known to affect and require communications. These rules include aircraft handoff between ATC sectors, as well as a set of flight adjustments (i.e. changes to altitude, flight path, and airspeed). Consequently, it is believed that a deep learning model may be developed to infer flight communications from predicted trajectories, flight plans, ATC sector boundaries, and available weather data. A notional architecture is provided in Figure 31, though significant effort must be continued for this task.

VI CONCLUSIONS

Future spectrum allocation techniques in the NAS are anticipated to rely on machine learning. Research efforts are attempted to develop deep learning architectures for the accurate prediction of aircraft locations and may continue to investigate consequent communication demand.

A general analysis of weather products found Echo Top as the best holistic product for model training, providing average horizontal and vertical accuracies of 50 nautical miles and 1160 ft, respectively. NOAA weather products, specifically temperature and wind components, were found to marginally improve vertical error (2-5% reduction), but would require sacrificing the sparsity and limited dimensionality of data.

By incorporating attention mechanisms into existing hybrid-recurrent architectures, model accuracy has the potential of increasing by over 60% - predicted trajectories resided within 41 nautical miles horizontally and 805 ft of elevation, on average. This result outperforms reported accuracies in state-of-the-art deep learning research, and is competitive against prior machine learning approaches as well.

The author's greatest perceived contribution is the unification and generalization of data and challenges within trajectory prediction; existing literature addressed unique flights, datasets, and models, with reported errors that were difficult to contextualize in a literature review alone. When training comparable models over limited and generalized datasets, it was not uncommon for error variances to increase by more than a factor of 2. Incorporating tuned hyperparameters helped some models to maintain or marginally improve on initial errors, however most vertical error increased by approximately 22% - on average and standard deviation. By starting to consider a variety of data, this thesis hopes to illustrate the complexity and importance of generalizing the trajectory prediction challenge in state-of-the-art research.

I APPENDIX A: COMPLETE EXTRACTION CHANNEL TUNING FIGURES

1 CNN-LSTM

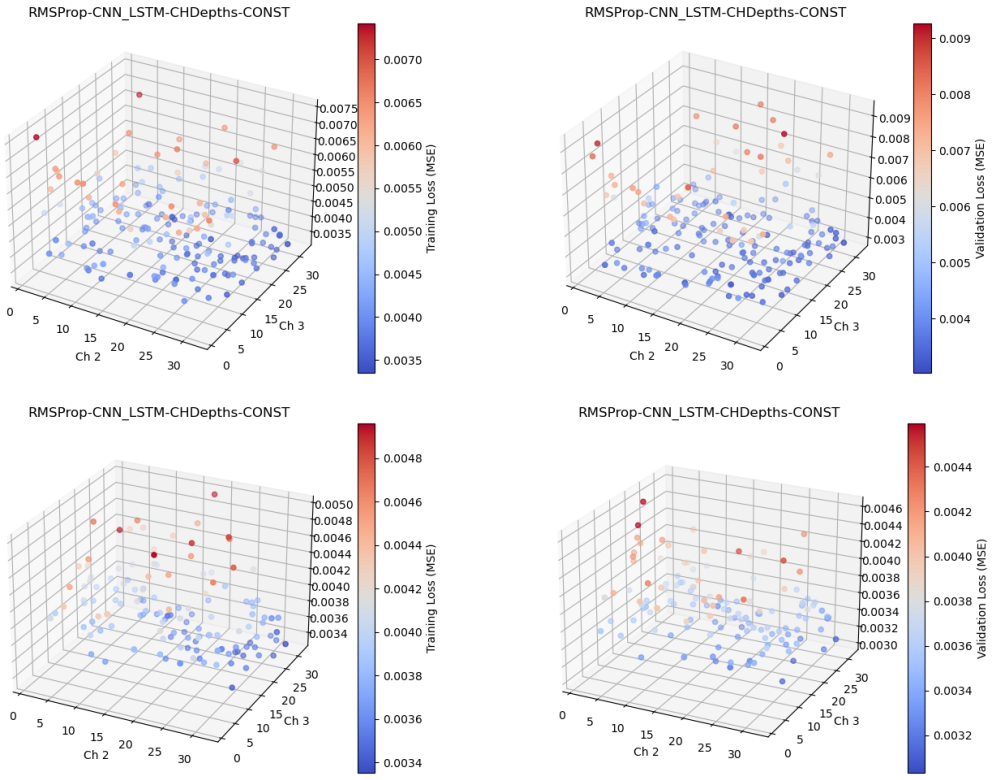

Figure 32: Training (left) and Validation (right) 3D Scatter Plots of Varied Channel Depths for CNN-LSTM Model, with Complete (top) and Limited (bottom) Sample Views

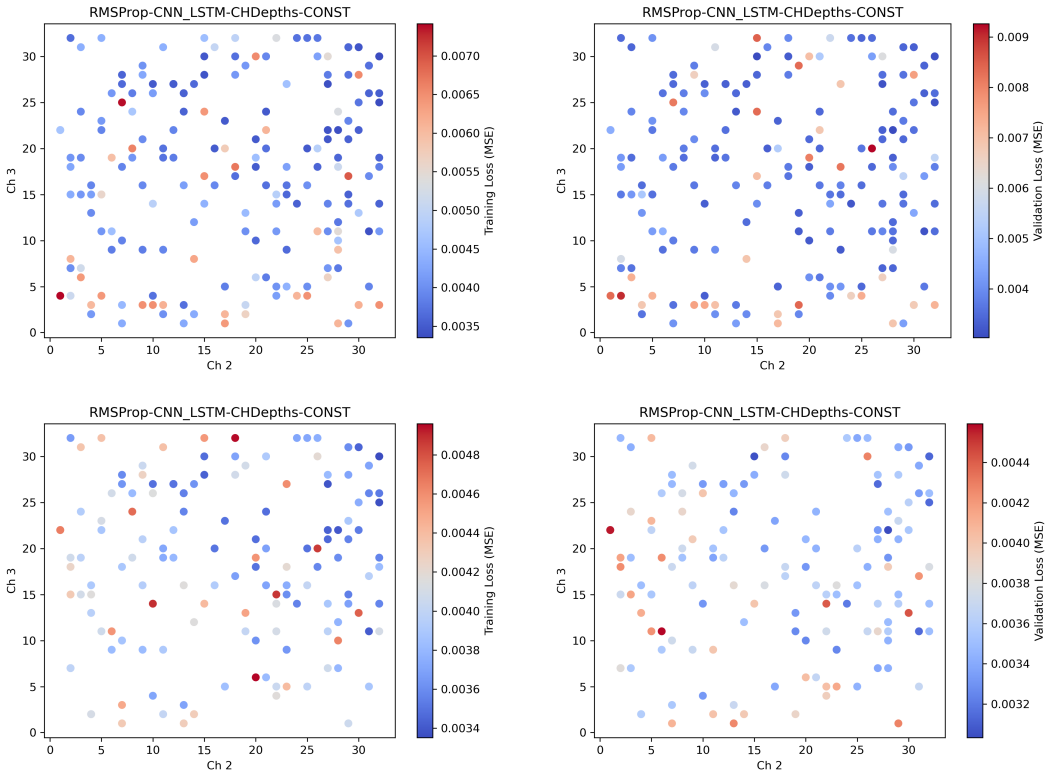

Figure 33: Training (left) and Validation (right) 2D Scatter Plots of Varied Channel Depths for CNN-LSTM Model, with Complete (top) and Limited (bottom) Sample Views

2 SA-LSTM

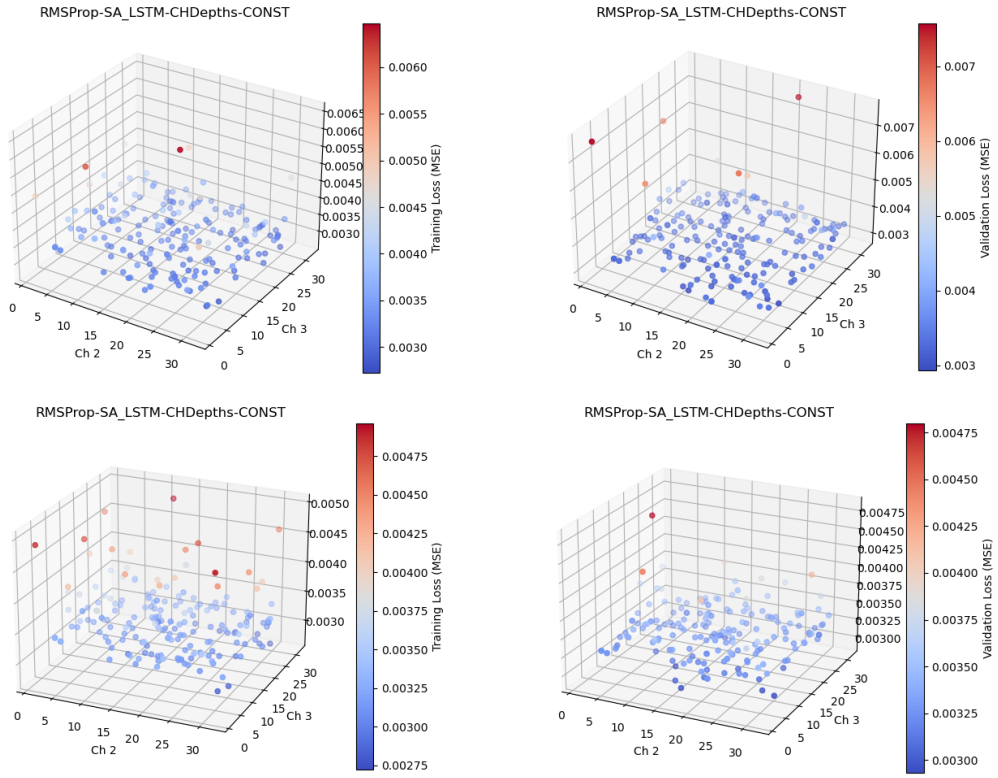

Figure 34: Training (left) and Validation (right) 3D Scatter Plots of Varied Channel Depths for SA-LSTM Model, with Complete (top) and Limited (bottom) Sample Views

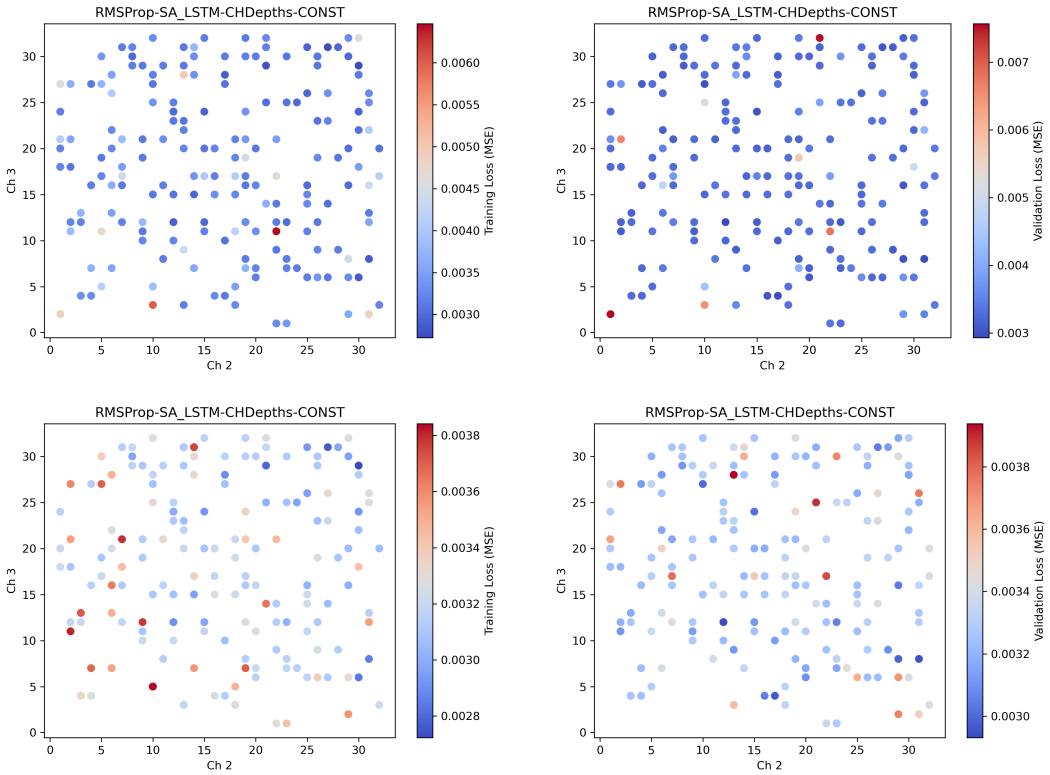

Figure 35: Training (left) and Validation (right) 2D Scatter Plots of Varied Channel Depths for SA-LSTM Model, with Complete (top) and Limited (bottom) Sample Views

II APPENDIX B: COMPLETE RNN HYPERPARAMETER TUNING FIGURES

1 CNN-LSTM

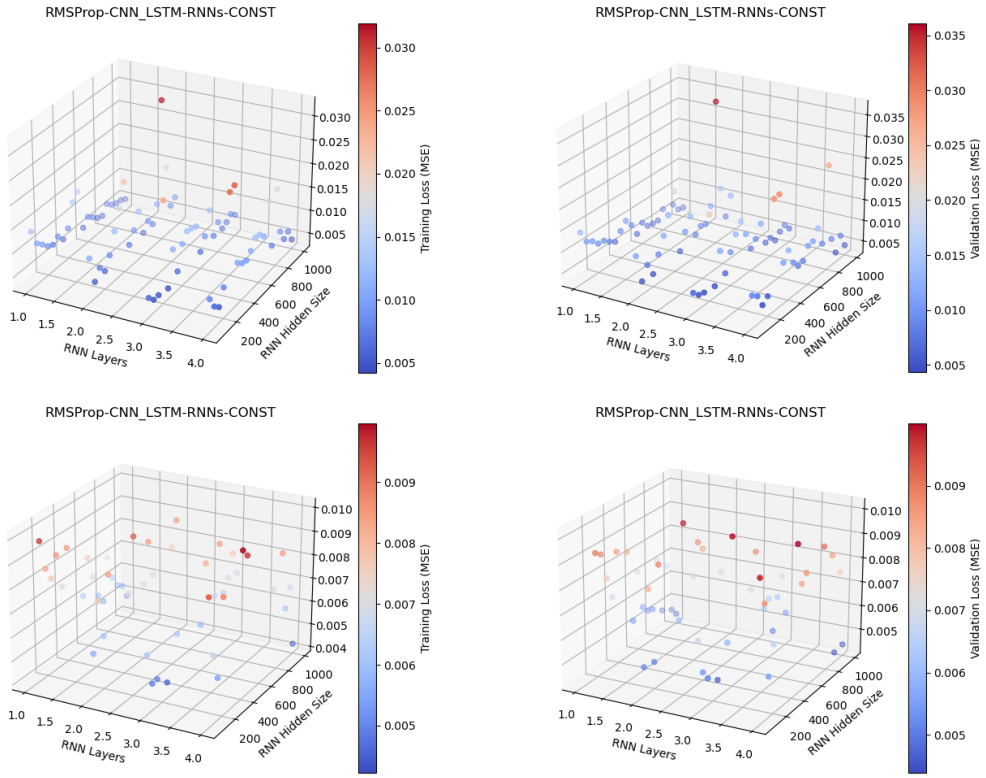

Figure 36: Training (left) and Validation (right) 3D Scatter Plots of Varied RNN Hyperparameters for CNN-LSTM Model with Input Size of 6, with Complete (top) and Limited (bottom) Sample Views

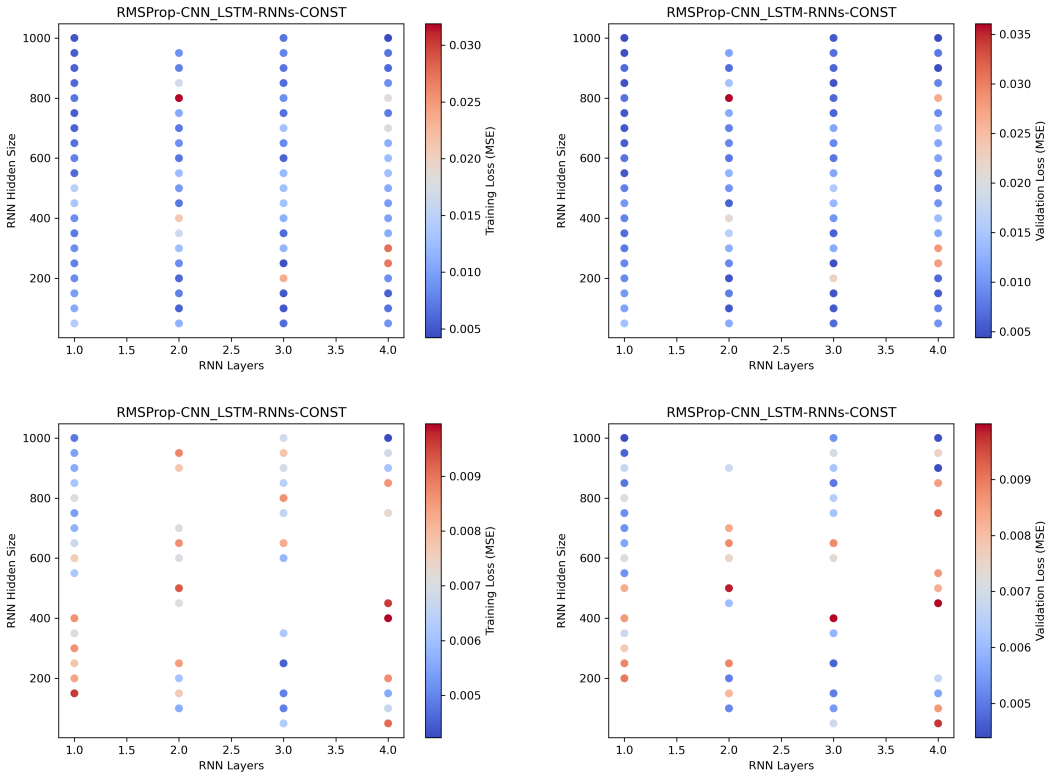

Figure 37: Training (left) and Validation (right) 2D Scatter Plots of Varied RNN Hyperparameters for CNN-LSTM Model with Input Size of 6, with Complete (top) and Limited (bottom) Sample Views

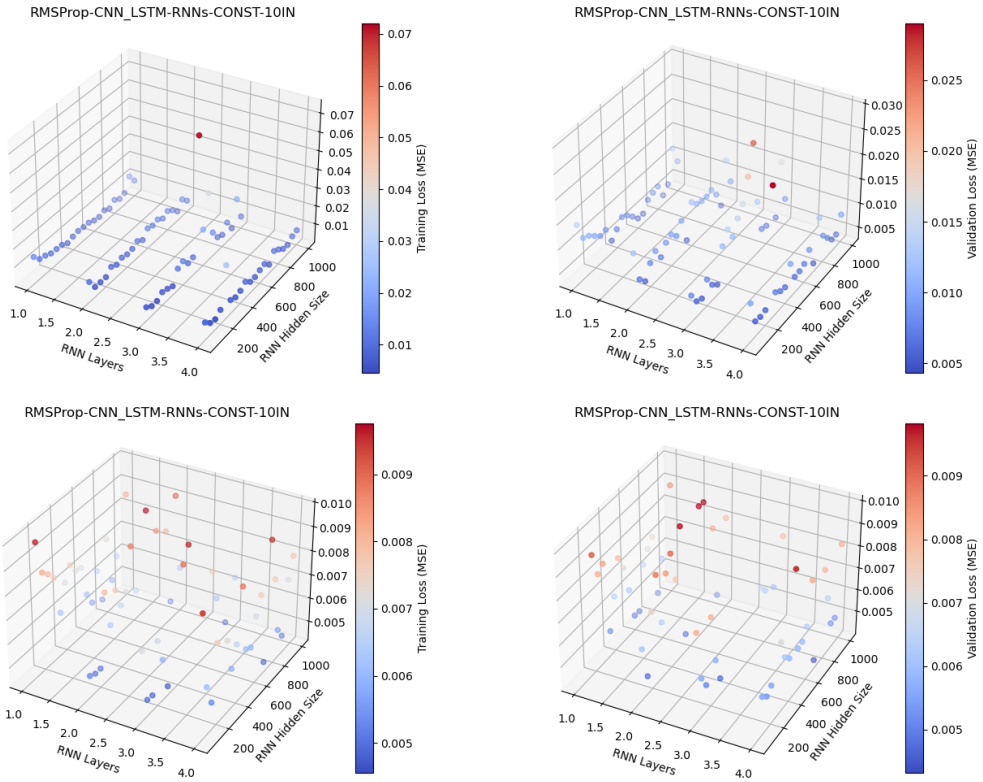

Figure 38: Training (left) and Validation (right) 3D Scatter Plots of Varied RNN Hyperparameters for CNN-LSTM Model with Input Size of 10, with Complete (top) and Limited (bottom) Sample Views

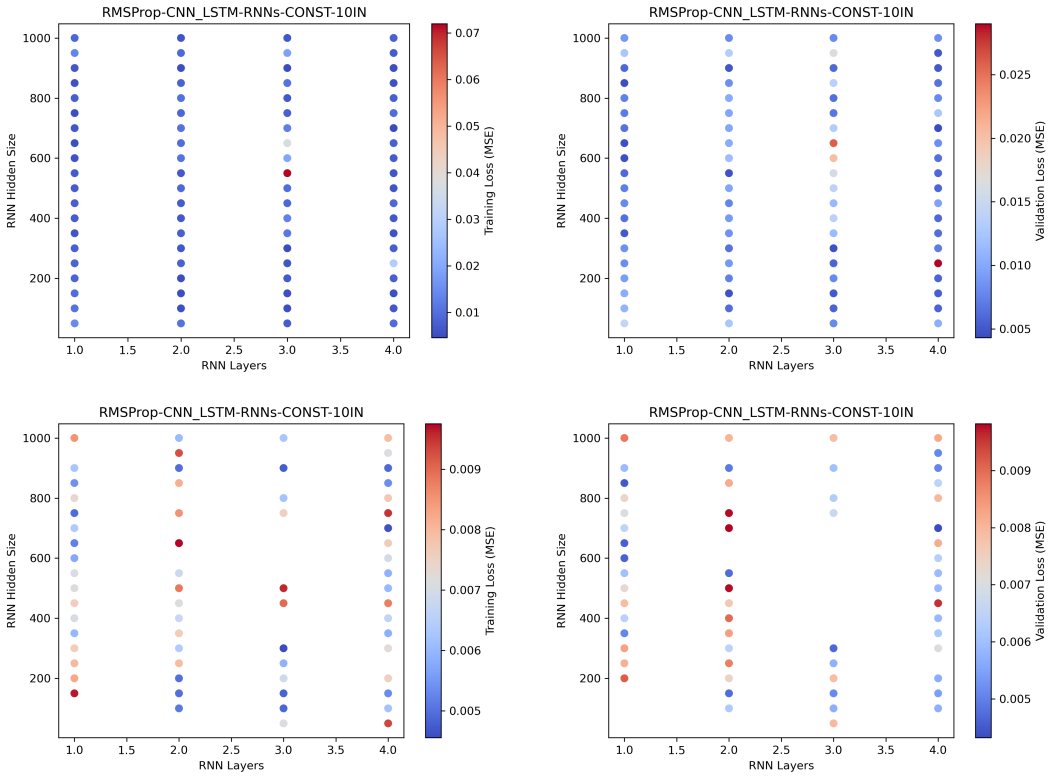

Figure 39: Training (left) and Validation (right) 2D Scatter Plots of Varied RNN Hyperparameters for CNN-LSTM Model with Input Size of 10, with Complete (top) and Limited (bottom) Sample Views

2 SA-LSTM

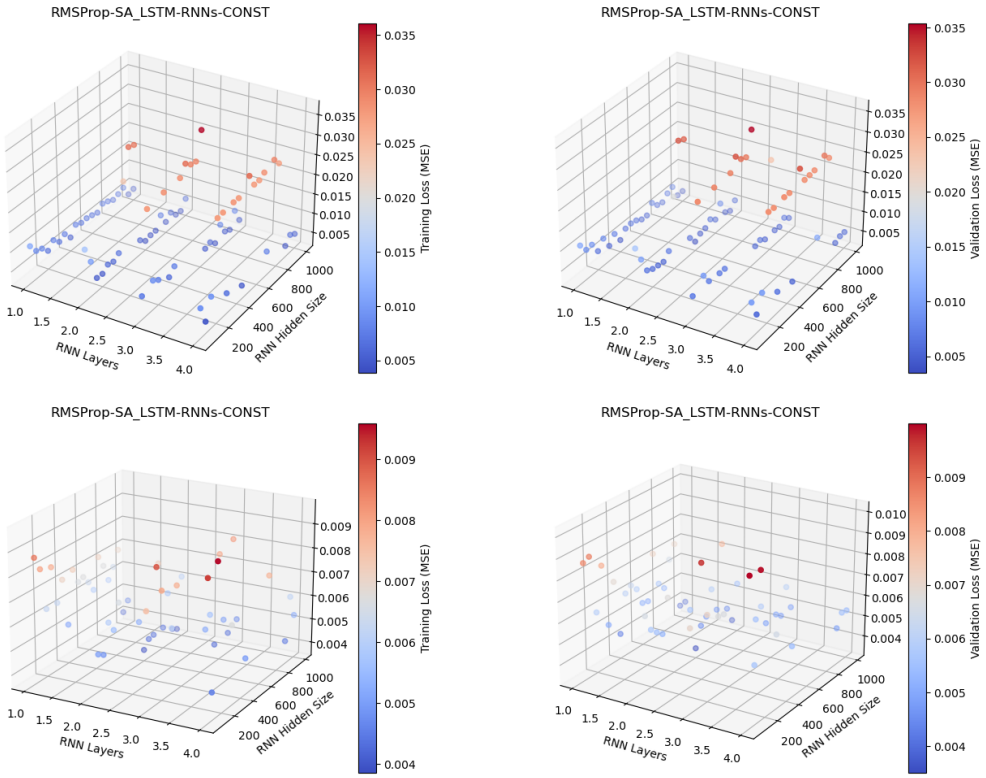

Figure 40: Training (left) and Validation (right) 3D Scatter Plots of Varied RNN Hyperparameters for SA-LSTM Model with Input Size of 6, with Complete (top) and Limited (bottom) Sample Views

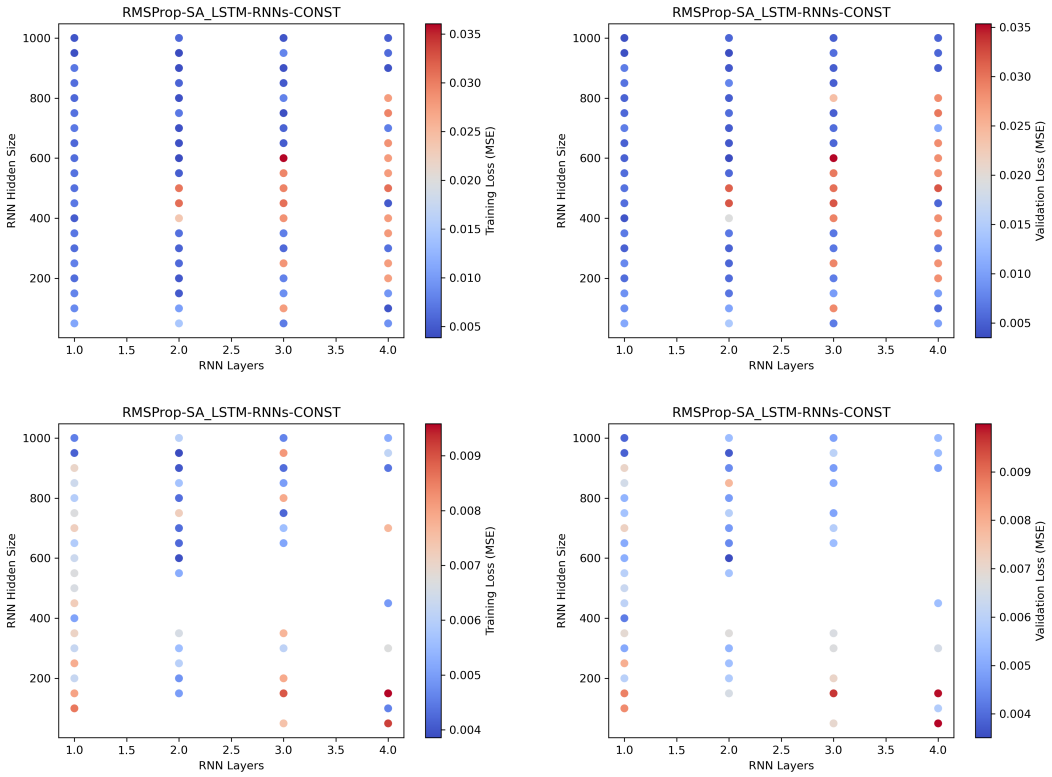

Figure 41: Training (left) and Validation (right) 2D Scatter Plots of Varied RNN Hyperparameters for SA-LSTM Model with Input Size of 6, with Complete (top) and Limited (bottom) Sample Views

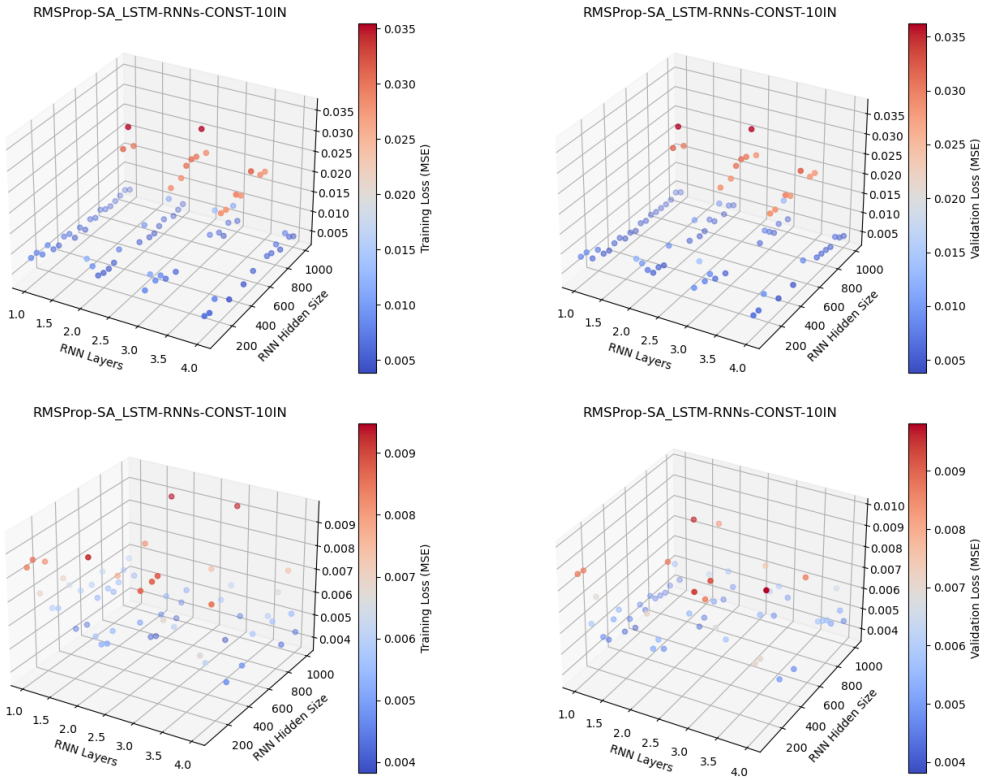

Figure 42: Training (left) and Validation (right) 3D Scatter Plots of Varied RNN Hyperparameters for SA-LSTM Model with Input Size of 10, with Complete (top) and Limited (bottom) Sample Views

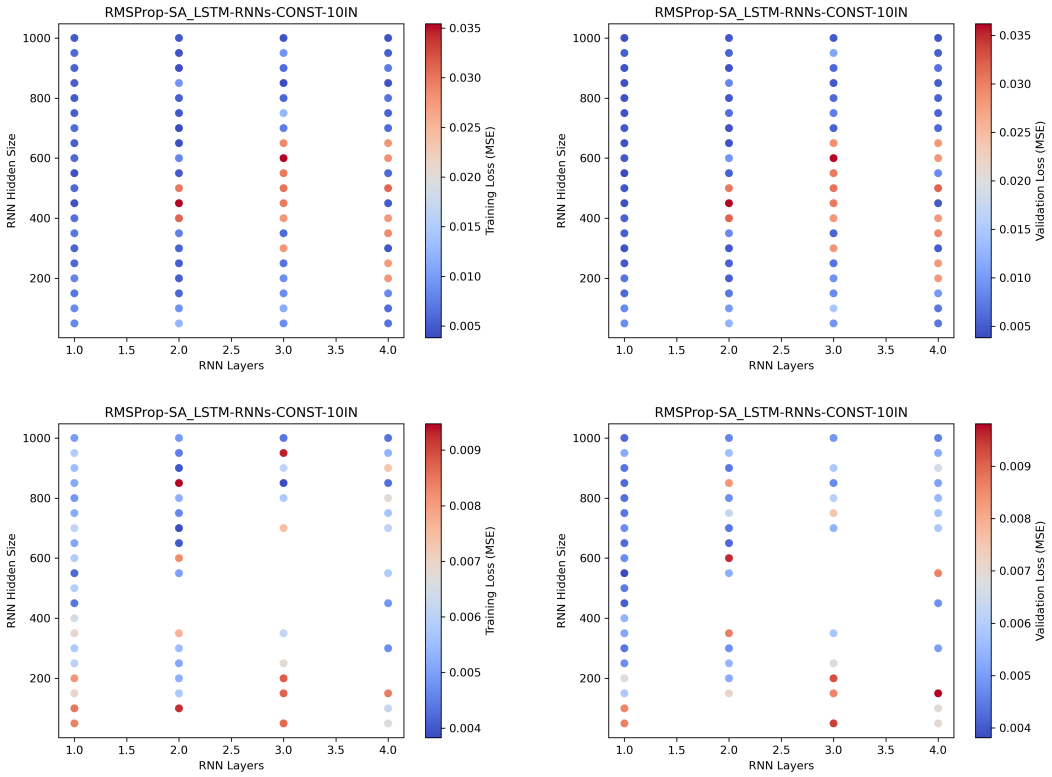

Figure 43: Training (left) and Validation (right) 2D Scatter Plots of Varied RNN Hyperparameters for SA-LSTM Model with Input Size of 10, with Complete (top) and Limited (bottom) Sample Views

3 CNN-GRU

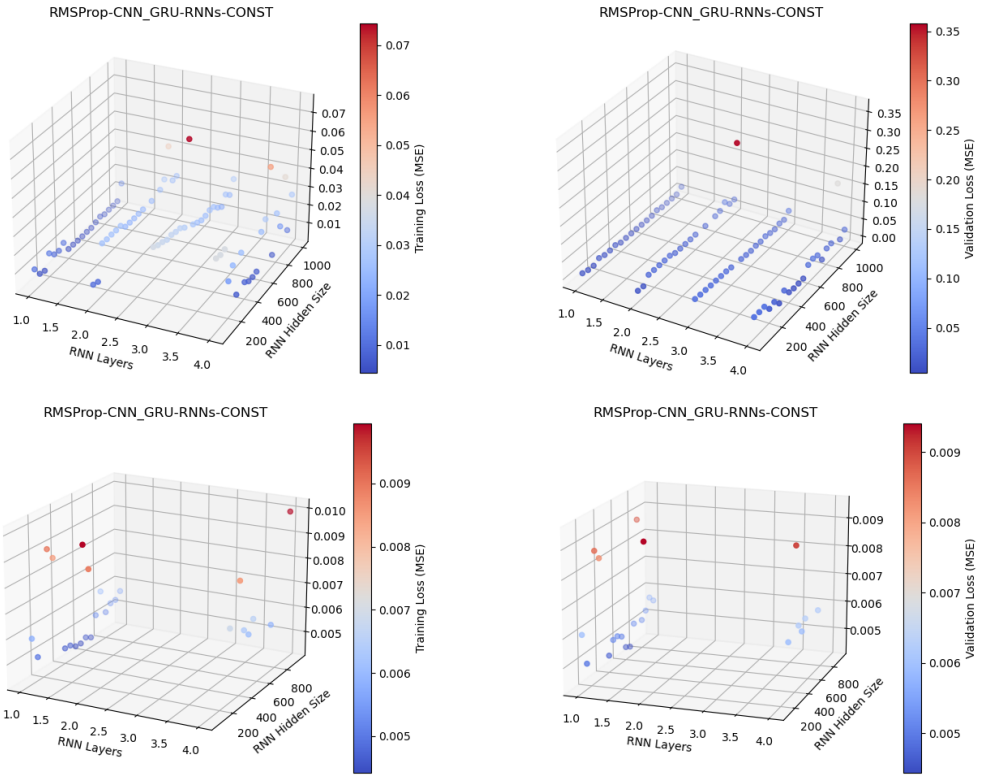

Figure 44: Training (left) and Validation (right) 3D Scatter Plots of Varied RNN Hyperparameters for CNN-GRU Model with Input Size of 6, with Complete (top) and Limited (bottom) Sample Views

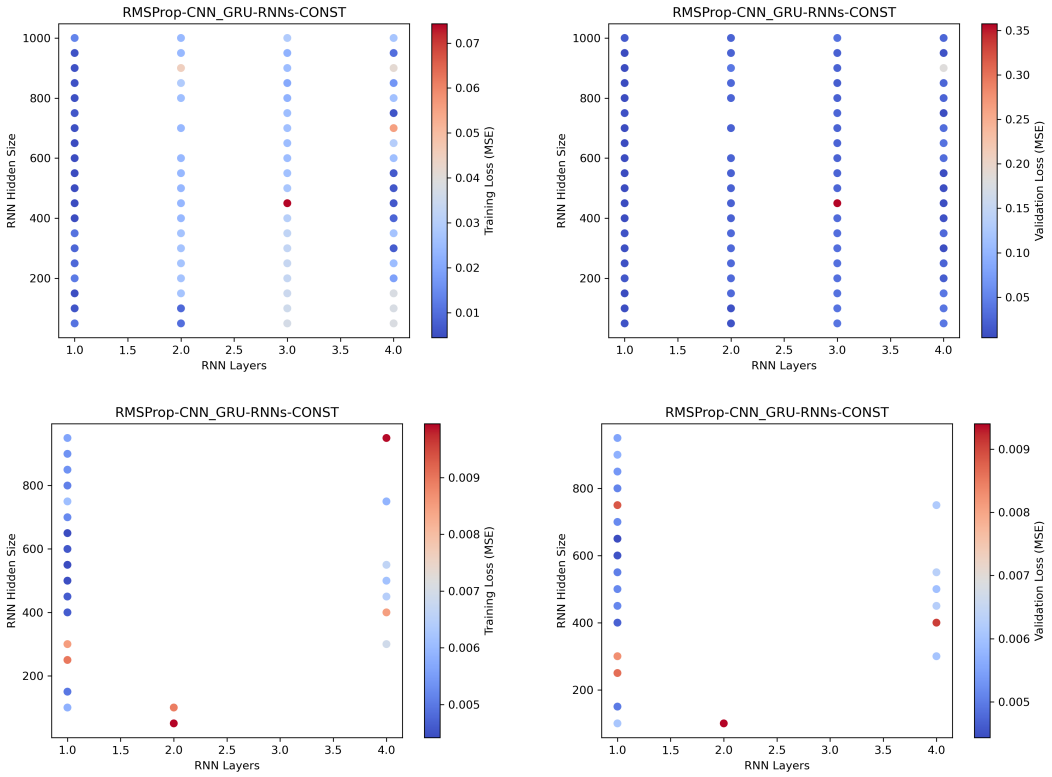

Figure 45: Training (left) and Validation (right) 2D Scatter Plots of Varied RNN Hyperparameters for CNN-GRU Model with Input Size of 6, with Complete (top) and Limited (bottom) Sample Views

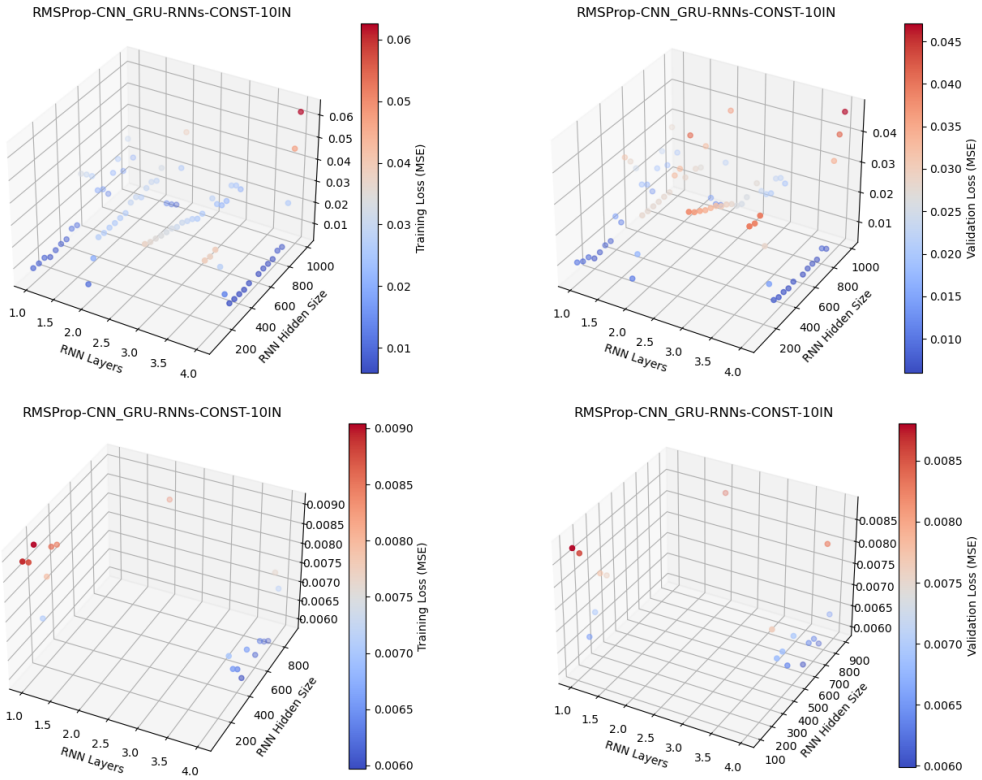

Figure 46: Training (left) and Validation (right) 3D Scatter Plots of Varied RNN Hyperparameters for CNN-GRU Model with Input Size of 10, with Complete (top) and Limited (bottom) Sample Views

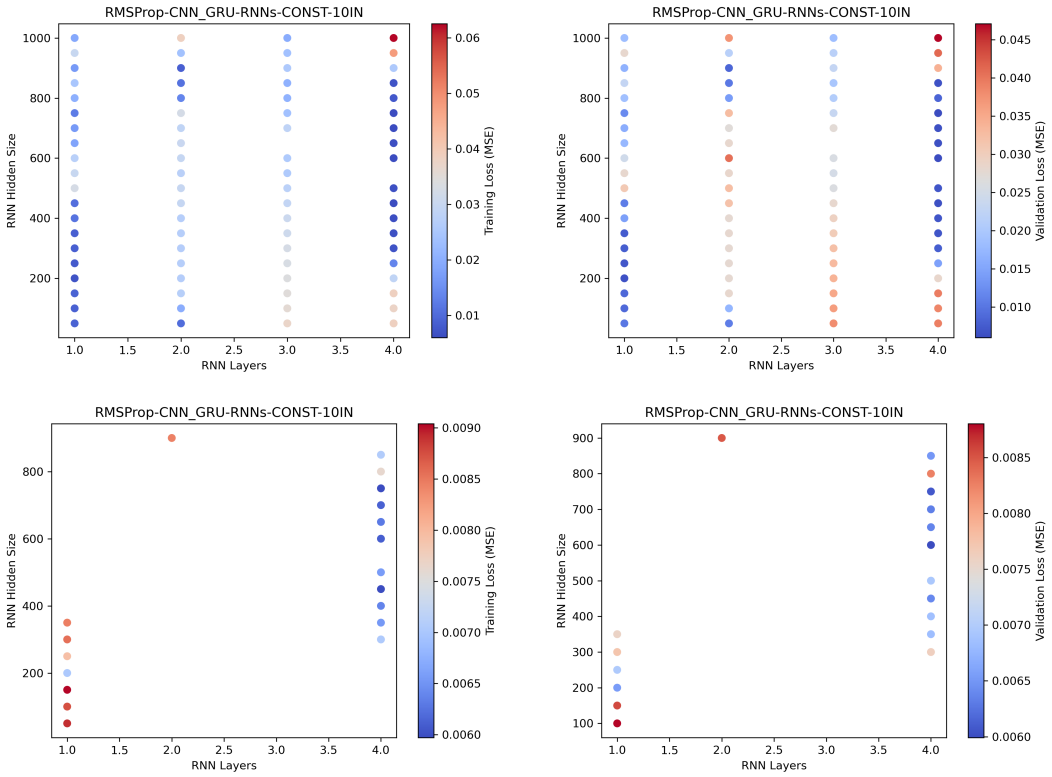

Figure 47: Training (left) and Validation (right) 2D Scatter Plots of Varied RNN Hyperparameters for CNN-GRU Model with Input Size of 10, with Complete (top) and Limited (bottom) Sample Views

4 SA-GRU

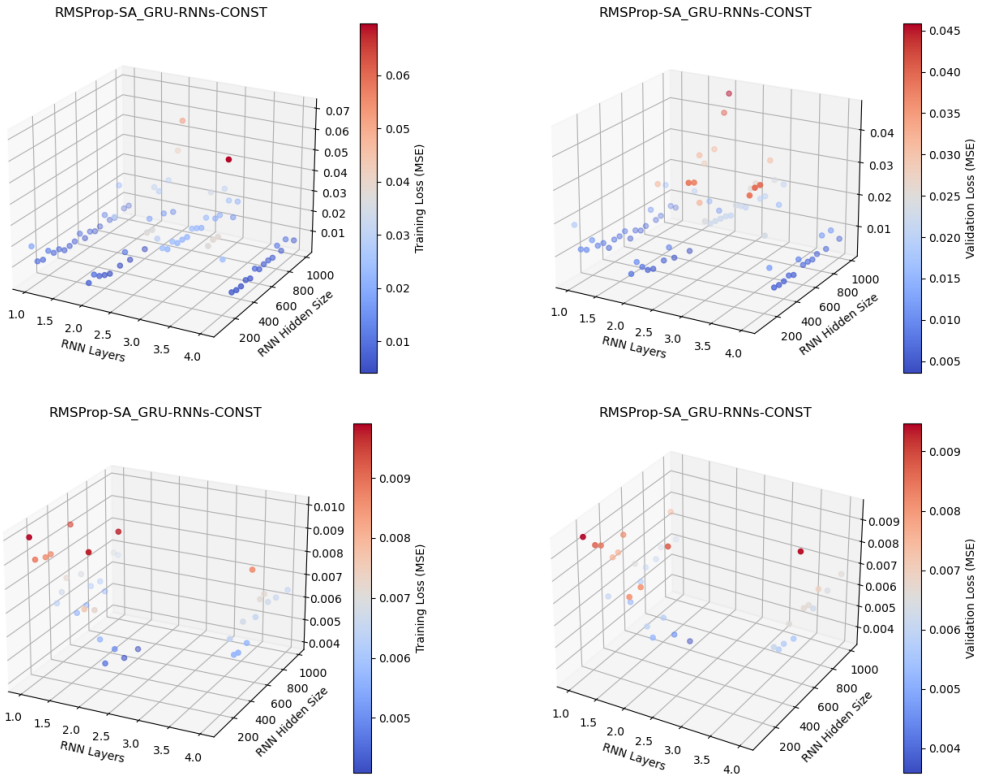

Figure 48: Training (left) and Validation (right) 3D Scatter Plots of Varied RNN Hyperparameters for SA-GRU Model with Input Size of 6, with Complete (top) and Limited (bottom) Sample Views

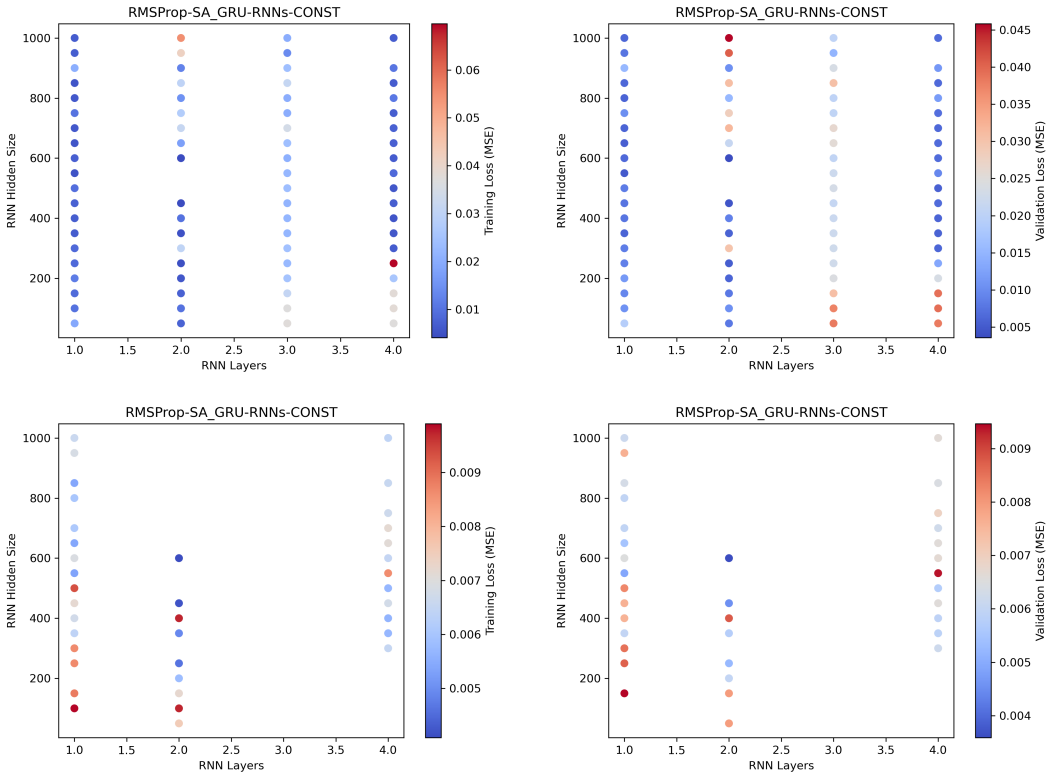

Figure 49: Training (left) and Validation (right) 2D Scatter Plots of Varied RNN Hyperparameters for SA-GRU Model with Input Size of 6, with Complete (top) and Limited (bottom) Sample Views

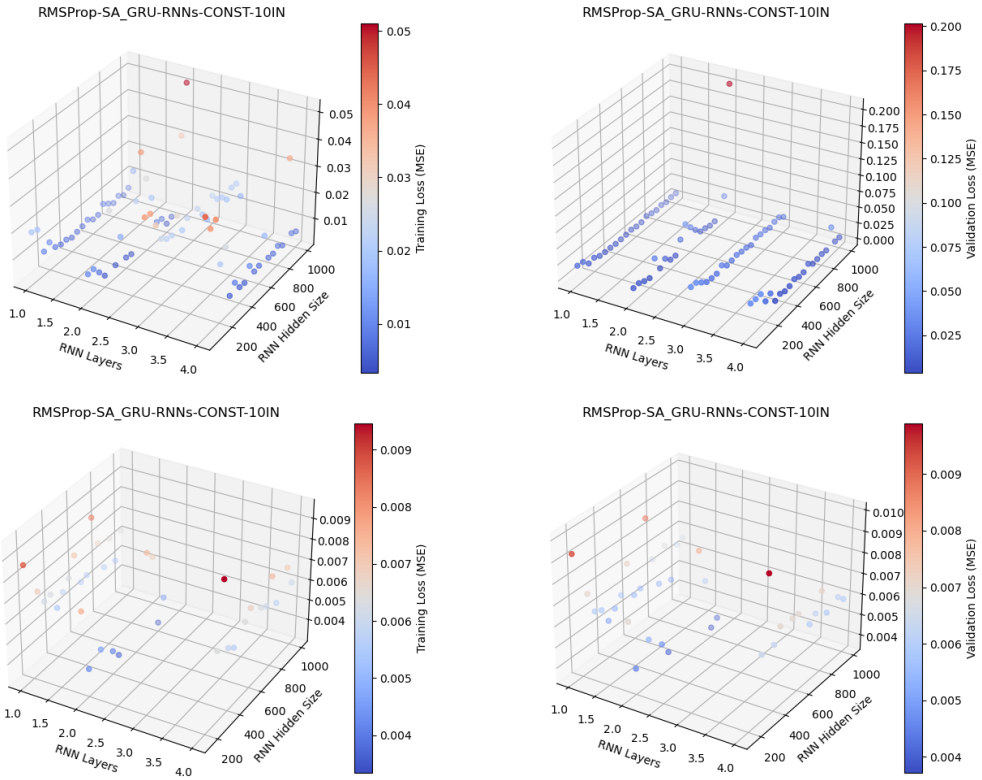

Figure 50: Training (left) and Validation (right) 3D Scatter Plots of Varied RNN Hyperparameters for SA-GRU Model with Input Size of 10, with Complete (top) and Limited (bottom) Sample Views

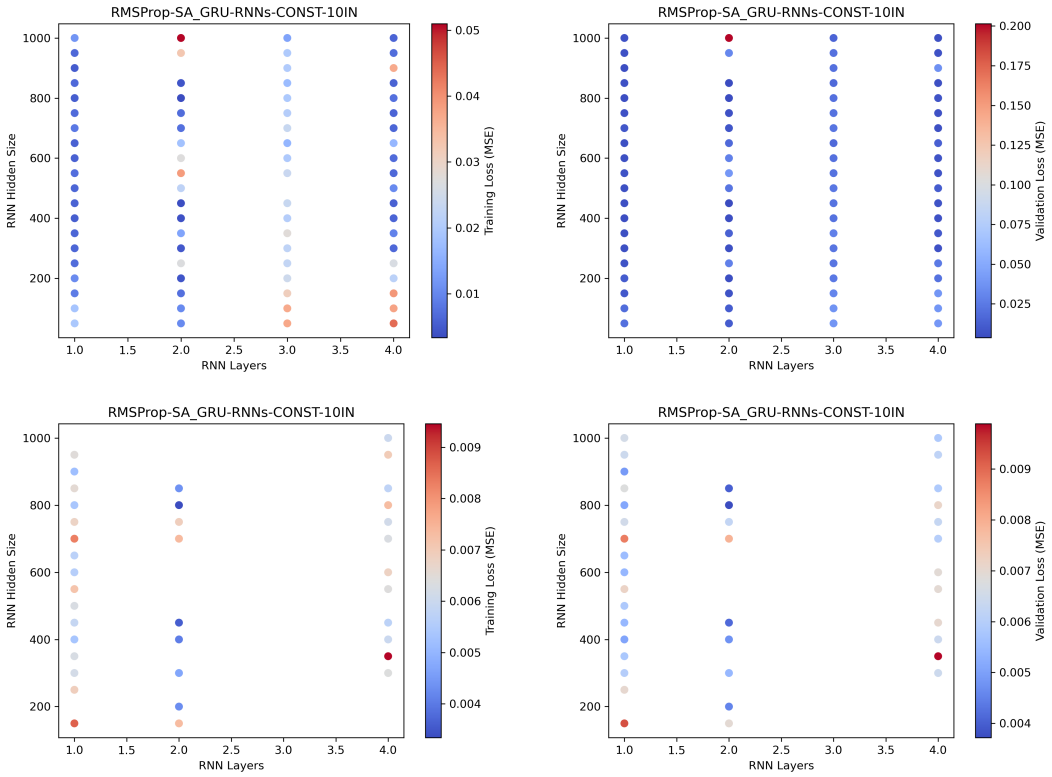

Figure 51: Training (left) and Validation (right) 2D Scatter Plots of Varied RNN Hyperparameters for SA-GRU Model with Input Size of 10, with Complete (top) and Limited (bottom) Sample Views

III APPENDIX C: EVALUATION VISUALS OF TRAINED MODELS

1 Initial Trained Model Plots

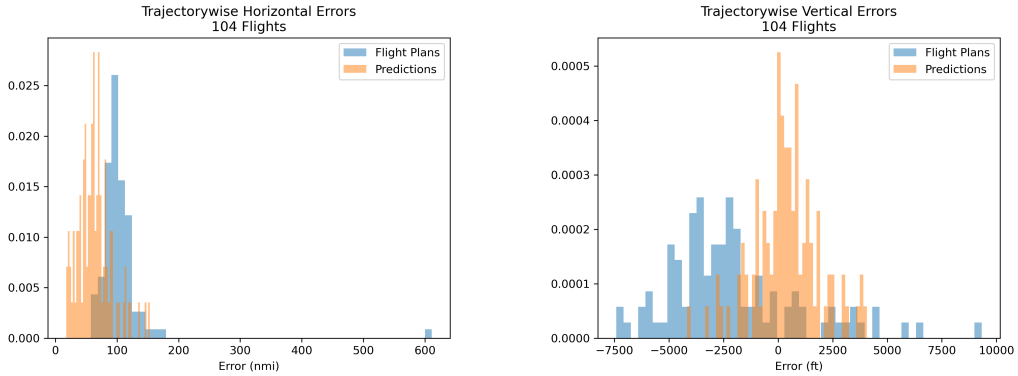

Figure 52: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-LSTM1lay Model

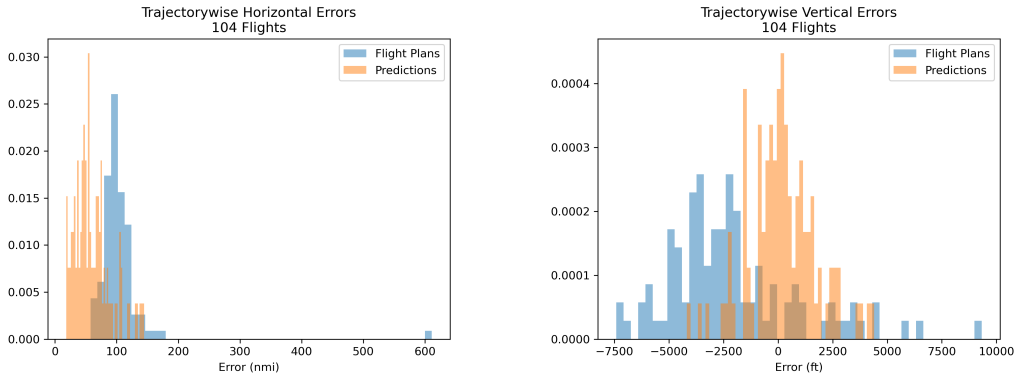

Figure 53: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-LSTM2lay Model

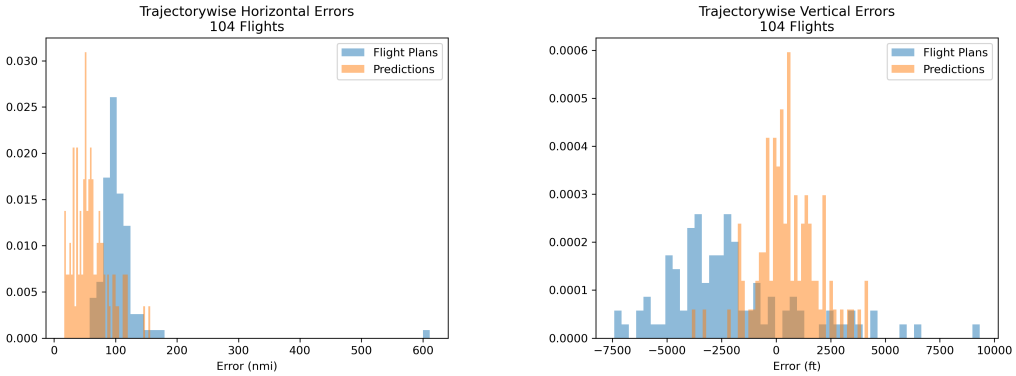

Figure 54: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-GRU1lay Model

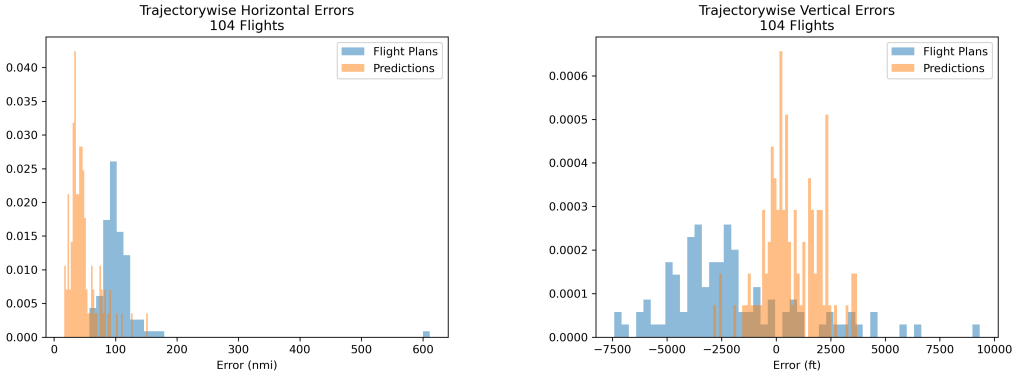

Figure 55: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-GRU2lay Model

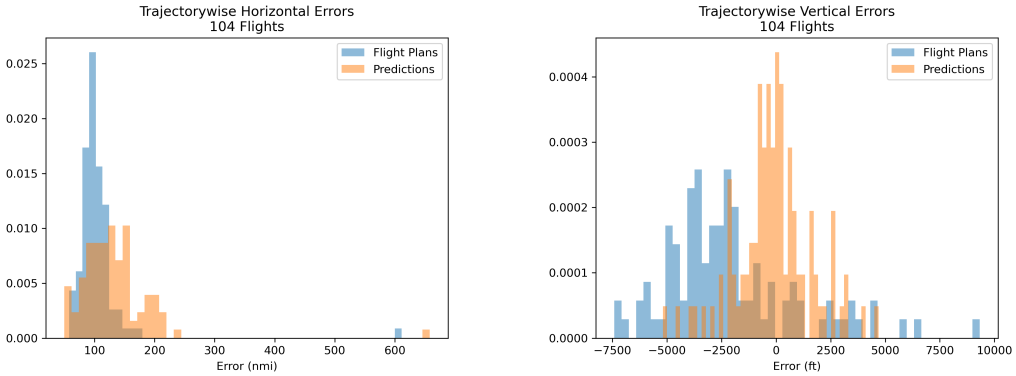

Figure 56: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-IndRNN2lay Model

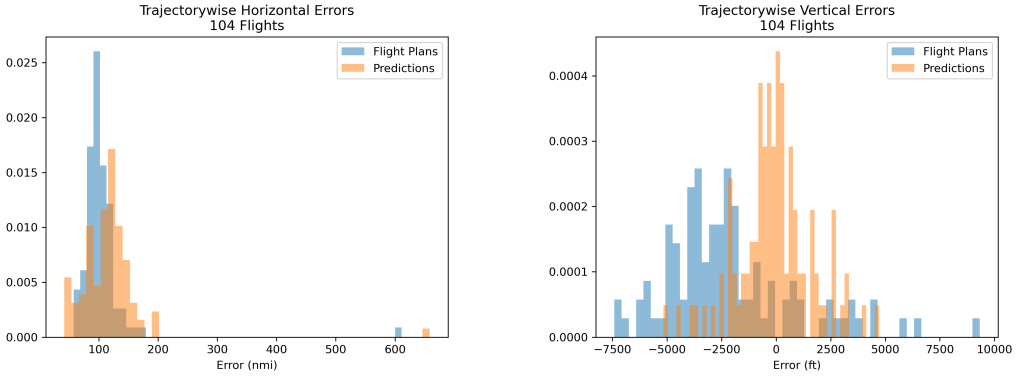

Figure 57: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-IndRNN3lay Model

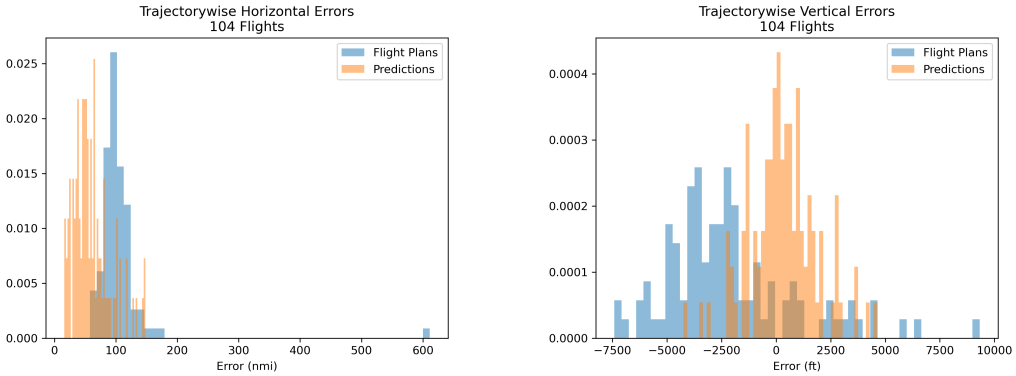

Figure 58: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN+SA-LSTM1lay Model

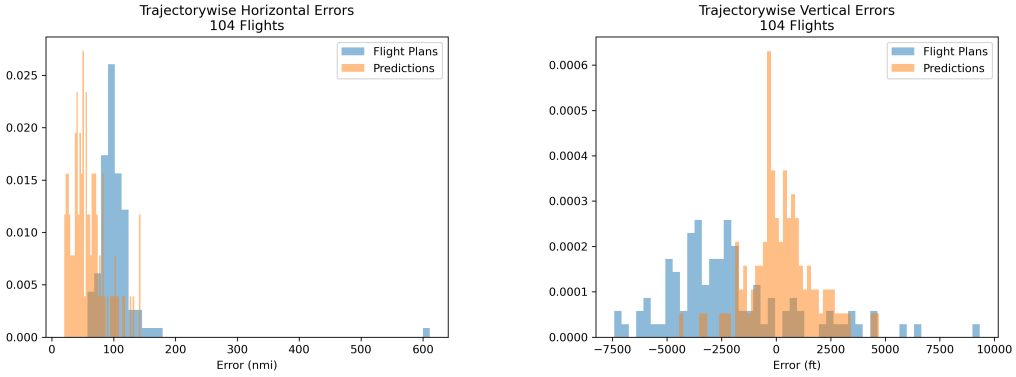

Figure 59: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN+SA-LSTM2lay Model

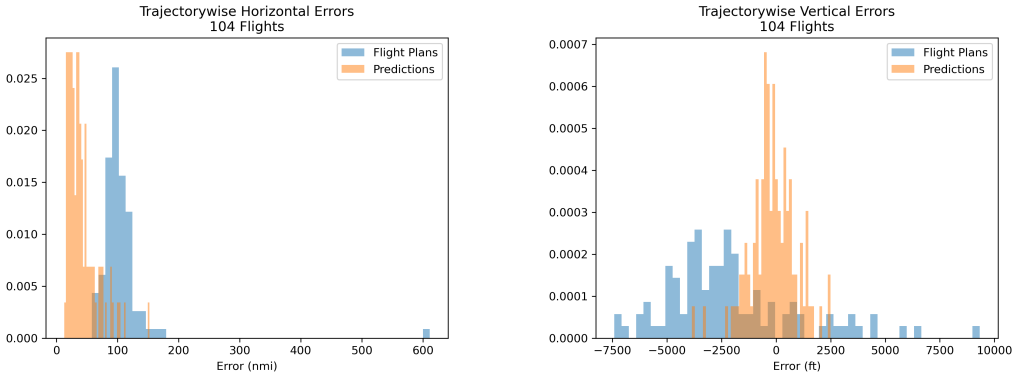

Figure 60: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for SA-LSTM1lay Model

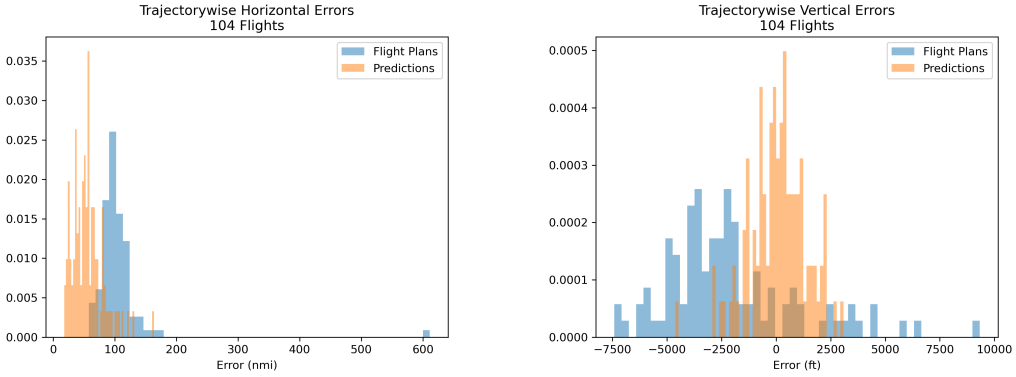

Figure 61: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for SA-LSTM2lay Model

2 Data Generalization Model Plots

A CNN-LSTM

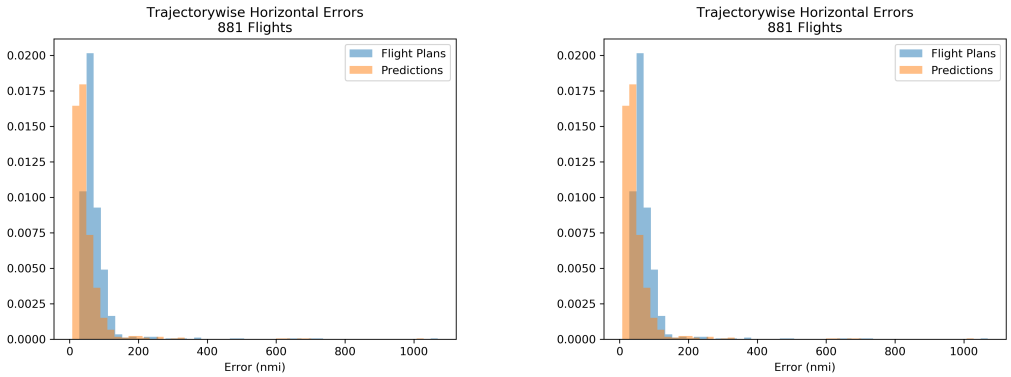

Figure 62: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-LSTM Generalized Model

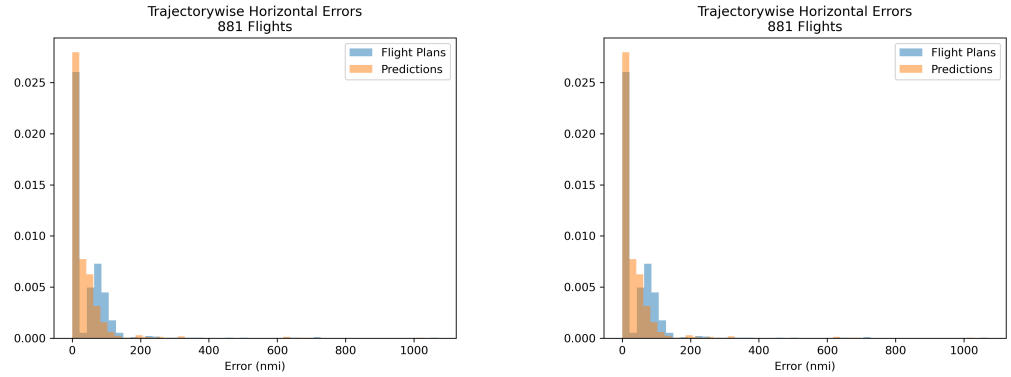

Figure 63: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-LSTM Generalized Model, KJFK-KLAX Flight Subset

B CNN-GRU

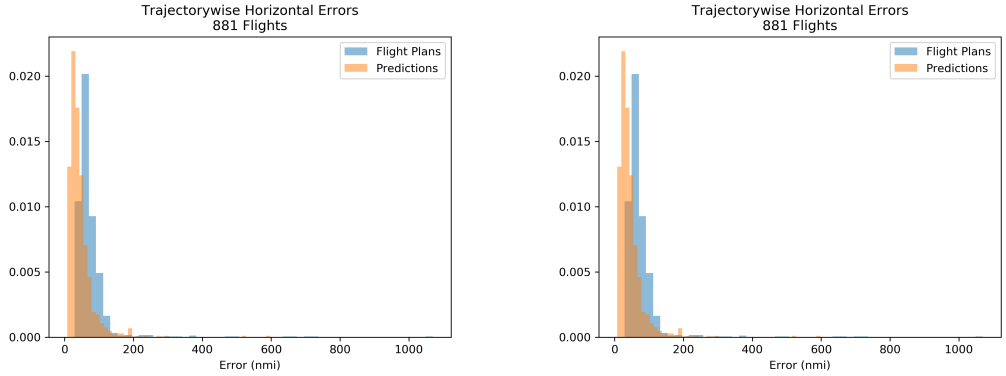

Figure 64: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-GRU Generalized Model

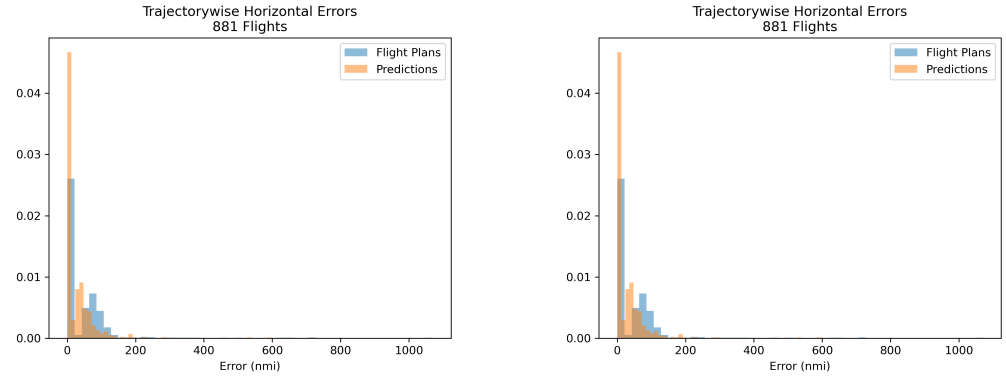

Figure 65: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-GRU Generalized Model, KJFK-KLAX Flight Subset

C SA-LSTM

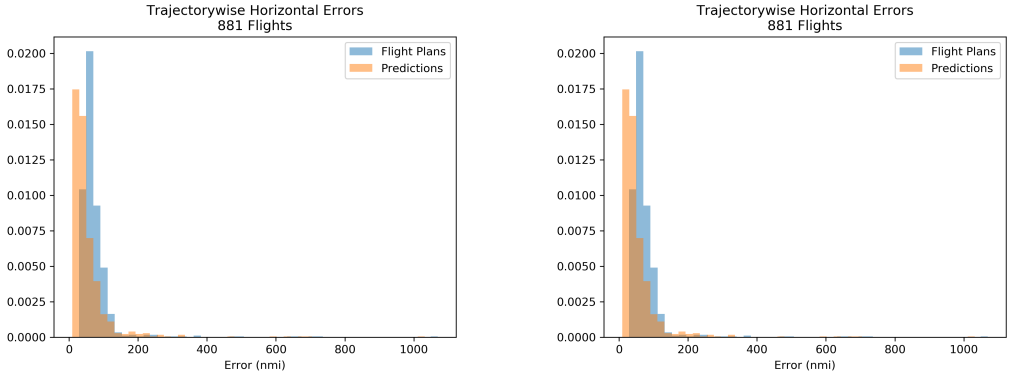

Figure 66: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for SA-LSTM Generalized Model

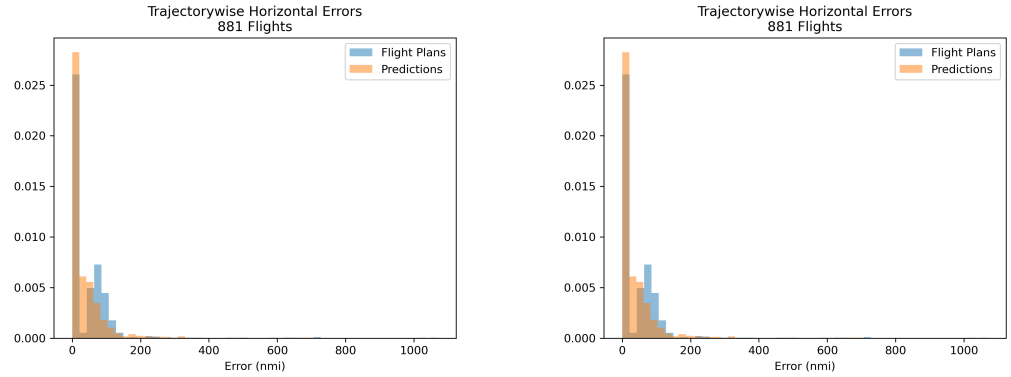

Figure 67: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for SA-LSTM Generalized Model, KJFK-KLAX Flight Subset

D SA-GRU

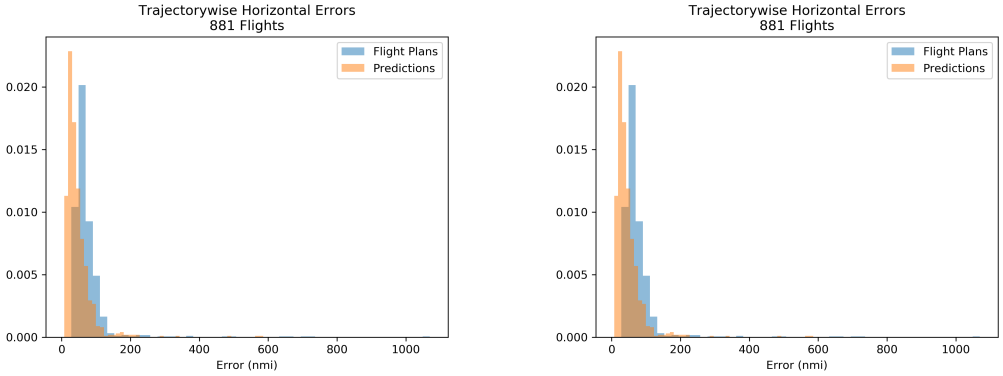

Figure 68: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-LSTM Generalized Model

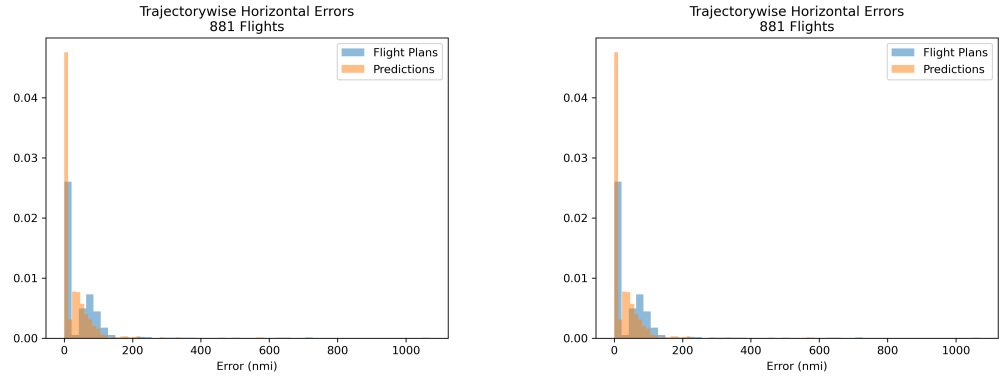

Figure 69: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for SA-GRU Generalized Model, KJFK-KLAX Flight Subset

3 Model Tuning Model Plots

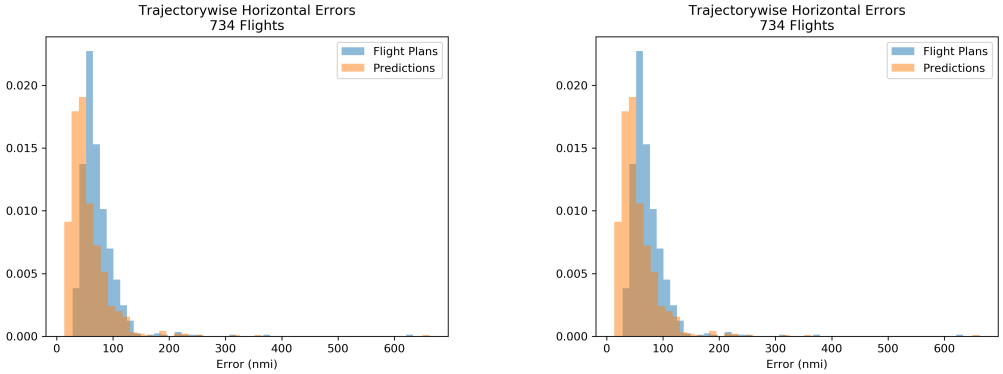

Figure 70: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-LSTM Generalized and Tuned Model

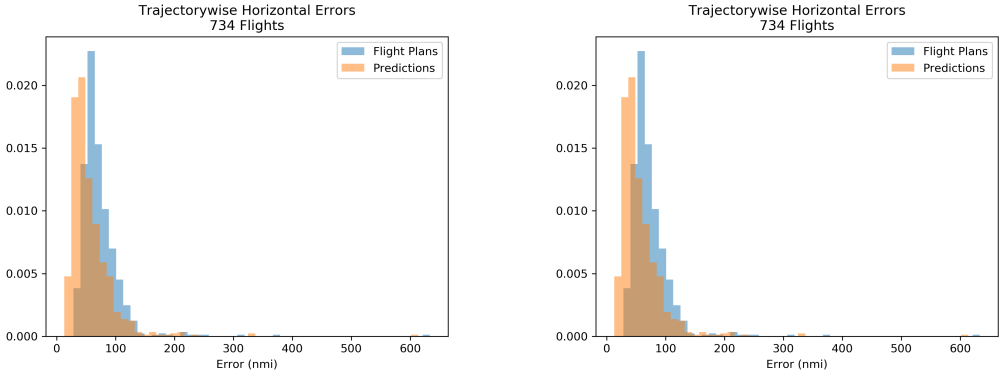

Figure 71: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for CNN-GRU Generalized and Tuned Model

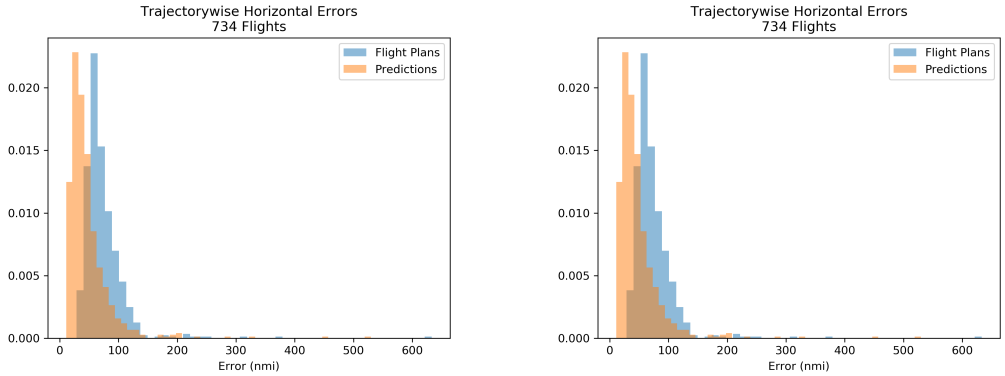

Figure 72: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for SA-LSTM Generalized and Tuned Model

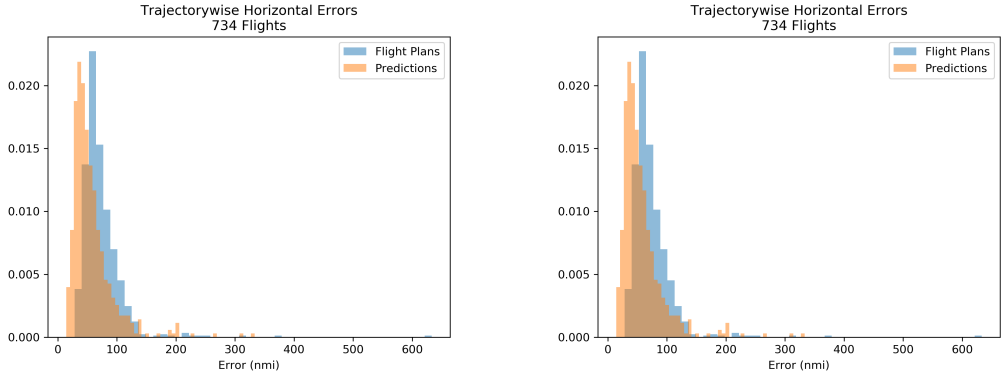

Figure 73: Trajectorywise Horizontal (left) and Vertical (right) Error Histograms for SA-GRU Generalized and Tuned Model

Bibliography

[1] OpenNav, "Opennav aeronautical database," in *http://opennav.com*. Accessed 2020-09-23.

[2] Y. Pang, H. Yao, J. Hu, and Y. Liu, *A Recurrent Neural Network Approach for Aircraft Trajectory Prediction with Weather Features From Sherlock*. 2019.

[3] Massachusetts Institute of Technology, "Corridor integrated weather services web display," in *https://ciws.wx.ll.mit.edu/*, 2021. Accessed 2021-04-27.

[4] R. D. Apaza, E. J. Knoblock, and H. Li, "A new spectrum management concept for future nas communications," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pp. 1–7, 2020.

[5] M. M. Eshow, M. Lui, and S. Ranjan, "Architecture and capabilities of a data warehouse for atm research," in *2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC)*, pp. 1E3–1–1E3–14, 2014.

[6] Y. Liu and M. Hansen, "Predicting aircraft trajectories: A deep generative convolutional recurrent neural networks approach.," *arXiv preprint arXiv:1812.11670*, 2018.

[7] S. Ayhan and H. Samet, "Aircraft trajectory prediction made easy with predictive analytics," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), p. 21–30, Association for Computing Machinery, 2016.

[8] J. Klingle-Wilson, D.; Evans, "Description of the corridor integrated weather system (ciws) weather products," tech. rep., Lincoln Laboratory, Massachusetts Institute of Technology, 2005.

[9] V. A. Petrushin, "Hidden markov models: Fundamentals and applications," in *2000 Online Symposium for Electrical Engineers (OSEE)*, 2000.

[10] Y. Lecun and A. Canziani, "Deep learning: Ds-ga 1008, spring 2020," in *https://atcold.github.io/pytorch-Deep-Learning/*, 2021. Accessed: 2020-11-15.

[11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.355v1*, 12 2014.

[12] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, "Independently recurrent neural network (indrnn): Building a longer and deeper rnn," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5457–5466, 2018.

[13] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, "Stand-alone self-attention in vision models," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[14] L. Ma and S. Tian, "A hybrid cnn-lstm model for aircraft 4d trajectory prediction," *IEEE Access*, vol. 8, pp. 134668–134680, 2020.

[15] N. Schimpf, "Weather preprocessing," in *Github*, https://github.com/schimpfen/Weather-Preprocessing, 2021.

[16] N. Schimpf, "Flight track prediction," in *Github*, https://github.com/schimpfen/Flight-Track-Prediction, 2021. Development in-progress.

[17] N. Schimpf, E. J. Knoblock, Z. Wang, R. D. Apaza, and H. Li, "Flight trajectory prediction based on hybrid-recurrent networks," in *2021 IEEE Cognitive Communications for Aeronautical Applications Workshop (CCAAW)*, pp. 1–6, 2021.

[18] Y. Pang, N. Xu, and Y. Liu, "Aircraft trajectory prediction using lstm neural network with embedded convolutional layer," in *Conference of the PHM Society 11*, PHM 2019, (Scottsdale, AZ, USA), pp. 1–10, Prognostics and Health Management Society (PHM), 2019.

[19] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[20] PyTorch, "Reproducibility," in *https://pytorch.org/docs/stable/notes/randomness.html*. Accessed 2021-6-7.

[21] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," 2018.

[22] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," 2018.

[23] International Civil Aviation Organization, *ICAO Handbook on Radio Frequency Spectrum Requirements for Civil Aviation: Volume II - Frequency Assignment Planning Criteria for Aeronautical Radio Communication and Navigation Systems*, first edition ed., 2017.

[24] B. Sridhar, T. Soni, K. Sheth, and G. Chatterji, "Aggregate flow model for air-traffic management," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 4, pp. 992–997, 2006.

[25] D. Sun and B. Sridhar, "Traffic flow management using aggregate flow models and the development of disaggregation methods," 08 2009.

[26] Y. Cao and D. Sun, "Link transmission model for air traffic flow management," *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 5, pp. 1342–1351, 2011.

[27] M. Albasman and J. Hu, "An approach to air traffic density estimation and its application in aircraft trajectory planning," pp. 706–711, 05 2012.

[28] W. Chan, M. Refai, and R. DeLaura, *An Approach to Verify a Model for Translating Convective Weather Information to Air Traffic Management Impact*. 2007.

[29] E. J. Knoblock, R. D. Apaza, H. Li, Z. Wang, R. Han, N. Schimpf, and N. Rose, "Investigation and evaluation of advanced spectrum management concepts for aeronautical communications," in *Proceedings of the Integrated Communications, Navigation, and Surveillance Conference, ICNS 2021*, 2021.

IV VITA

Nathan Schimpf was raised in Fort Thomas, Kentucky. They attended the University of Louisville, earning a Bachelor of Science in Electrical Engineering, with High Honors, in 2020. They remained at the University of Louisville to complete their Master of Engineering in Electrical Engineering.

While an undergraduate, Nathan supported research on physical-layer wireless systems by developing hardware implementations. They were awarded an NSF I-CORPS grant through the university to explore the viability of a startup supporting these implementations. They also remained active with the university's IEEE student branch, serving as technology chair and branch chair.

Nathan is currently interning with NASA Glenn Research Center virtually. Following the internship, they will be resuming studies at the University of Louisville, starting in the university's PhD program.