Comparing the Performance of Multiple Linear Regression, Random Forest, and Artificial Neural Networks for the Prediction of Weather on Mars

by
Jared Frazier

A thesis presented to the Honors College of
Middle Tennessee State University in partial fulfillment of
the requirements for graduation from the University Honors College

Fall 2021

Thesis Committee:

Dr. Salvador Barbosa, Thesis Director

Dr. Joshua Phillips, Second Reader

Dr. Mary Evins, Thesis Committee Chair

Comparing the Performance of Multiple Linear Regression, Random Forest, and Artificial Neural Networks for the Prediction of Weather on Mars

by Jared Frazier

APPROVED:

_____

Dr. Salvador Barbosa, Thesis Director
Associate Professor, Computer Science

_____

Dr. Joshua Phillips, Second Reader
Associate Professor, Computer Science

_____

Dr. Mary Evins, Thesis Committee Chair
Research Professor, University Honors College

Dedication

To my beloved parents and brother, without whom nothing I have ever accomplished would be possible.

Acknowledgments

I would like to acknowledge all the phenomenal faculty at MTSU who have facilitated and encouraged my academic and personal growth. These individuals have pushed me to give my best efforts and consistently challenged me intellectually: Dr. Mengliang Zhang, Dr. Greg Van Patten, Dr. Sal Barbosa, Dr. Josh Phillips, Dr. David Butler, Dr. Jamie Burriss, and Ms. Laura Clippard. Additionally, this work was supported by an MTSU Silver Undergraduate Research Experience and Creative Activity Grant.

Abstract

This thesis explores several machine learning methods for time series forecasting for weather prediction on Mars. The colonization of Mars has been proposed and funded by both public and private organizations like the National Aeronautics and Space Administration (NASA) and the aerospace corporation Boeing. The colonization of Mars has many challenges, one of which is the reliable prediction of weather. Traditional weather prediction techniques, such as numerical weather prediction, are not feasible on Mars given the lack of infrastructure needed for such powerful methods. In this thesis, several machine learning methods were implemented to circumvent these computational requirements: multiple linear regression (MLR), random forest (RFs), and artificial neural networks (ANNs). The work done for this thesis will inform the research questions of future atmospheric informaticians investigating the colonization of Mars and will serve as a strong baseline for model performance and methodology. Code and data are freely available at https://github.com/jfdev001/mars-ml-mtsu-honors-thesis.

Table of Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

1. National Aeronautics and Space Administration: NASA
2. Multiple Linear Regression: MLR
3. Random Forest: RF
4. Artificial Neural Network: ANN
5. Recurrent Neural Network: RNN
6. Convolutional Neural Network: CNN
7. Mean Absolute Error: MAE
8. Mean Squared Error: MSE
9. Root Mean Squared Error: RMSE
10. Kelvin (unit of temperature): K
11. Pascal (unit of pressure): Pa
12. Gated Recurrent Unit: GRU
13. Long-Short Term Memory: LSTM
14. Cross Validation: CV
15. Rover Environmental Monitoring Station: REMS
16. Rolling-Origin-Recalibration Cross Validation (aka Walk-Forward Validation): RORCV
17. Mars Recurrent Neural Network: MarsRNN
18. Autoregressive: AR
19. Mars Autoregressive Net: MarsARNet
20. Unstacked Iterative Joint Model: USJ
21. Unstacked Stacked Iterative Model: USI

22. Stacked Joint Model: SJ

23. Stacked Iterative Model: SI

24. Bayesian Optimization: BOPT

25. Hyperparameter: HP

26. Variance Inflation Factor: VIF

I. BACKGROUND

At any given moment, a devastating cosmic event could wipe all life on Earth from existence. In combination with pressures humanity places on Earth's biosphere, extinction may be inevitable [1]. Moreover, an understanding of extra-terrestrial climatic conditions is critical for future unmanned missions [2]. Going beyond our domain, further from the sun, and to the terrestrial planet Mars may be one way to reduce the possibility of human extinction [3].

Despite this lofty goal, the hostile Martian weather conditions differ vastly from those on Earth, and the ability to predict those conditions would be invaluable for successful colonization and further exploration. In particular, the extremely wide range of temperatures (-225 °F to 70°F) are a significant barrier to implementing human infrastructure [1], [4]. Further, traditional weather prediction techniques (e.g., numerical weather prediction) are computationally expensive and are not always stable due to the volatile physical conditions of the Earth's atmosphere [3], [5]. While such techniques have steadily improved over the past few decades, they require significant technological infrastructure such as super-computing facilities, weather satellites, and other telemetry instruments that are not present on Mars [6].

Supervised machine learning is the process whereby computers are given training inputs (the specific values of independent variables) and the *known* outputs corresponding to these inputs in order to learn a rule, or function, that maps inputs to

outputs [7]. A simple example of a supervised learning problem pertaining to weather prediction might be the following: given a set of input values for humidity (70%, 20%, 30%) and a corresponding set of outputs for whether it will rain (*Will Rain, Won't Rain, Will Rain*), then the task of supervised learning is to learn a rule for predicting whether it will rain or not based on the humidity. Supervised machine learning is resistant to the incomplete understanding of atmospheric conditions that introduces uncertainties to numerical weather prediction and is therefore ideal for the even less understood Martian atmosphere [8]. Machine learning has also successfully been used for weather forecasting in a number of studies [9]–[16]. Additionally, a recent study implemented many machine learning methods—which are like those utilized in this thesis—using a smaller subset of NASA's Curiosity Rover data for the analysis of weather; however, the study does not enumerate data preparation, cross validation, or hyperparameter tuning techniques [13].

Despite known non-linear—in particular, sporadic or quasi-unpredictable—responses in weather, it has also been demonstrated that there is some degree of linearity that is present between certain input variables and certain output variables (e.g., the percent of dry days for a given year and total annual rainfall, respectively) [16]. Therefore, a multiple linear regression (MLR) model is also appropriate since it was unknown whether the current system's (Mars' Gale Crater) predictor variables and the response variables would demonstrate a linear relationship [10].

Random forest (RF) models—which combine decision trees to produce an average output—have been successfully implemented for the prediction of both severe and normal weather conditions [17], [11]. Therefore, this model is a suitable choice for the volatile Martian atmosphere.

Finally, artificial neural networks (ANNs) are strong candidates for weather forecasting since they can capture the non-linear output of future weather conditions from past weather conditions [18], [9]. Additionally, the developer has much more control over the design of ANNs for a given problem and therefore ANNs can be implemented with much greater agency than either RFs or MLR.

II. THESIS STATEMENT

The objectives of this study are to (1) implement MLR, several ANN architectures such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), and RF models to predict mean ambient air temperature in a region of Mars known as the Gale Crater; and (2) use the root mean squared error (RMSE), mean squared error (MSE), and mean absolute error (MAE) as metrics to evaluate which model can most accurately generate a 7-sol forecast for the mean ambient air temperature on Mars when given 28 prior sols of weather data. This study is important not only in comparing the predictive capabilities of various machine learning algorithms but is also a unique study since to our knowledge none of the NASA Mars Science Laboratory Teams have attempted similar efforts [19]. This thesis is relevant to atmospheric scientists, applied mathematicians, and machine learning engineers as it compares a breadth of techniques on a novel system. Finally, this thesis is relevant to the author's academic and personal growth since the author plans to attend graduate school for Computational Science, and the techniques learned are applicable to computational experimentation in general.

III. METHODOLOGY

A: Research Framework

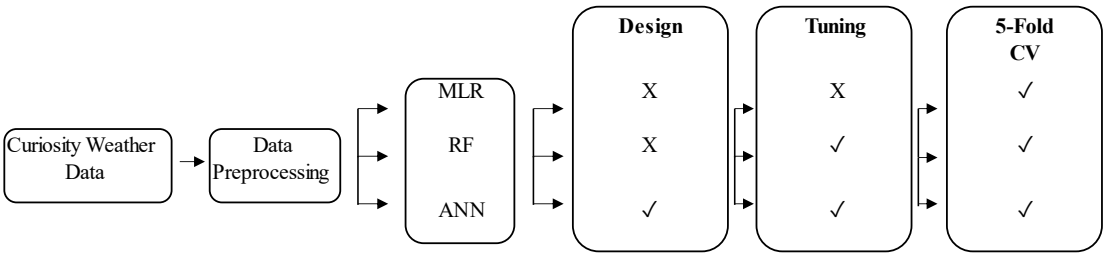


Figure 1. The research framework. The letter X indicates a non-applicable step for a model while a check mark ✓ indicates an applicable step for a model.

This study consists of several distinct phases depicted in the above figure: (1) data collection and preprocessing; (2) designing of models; (3) tuning of models; and (4) 5-fold cross validation (CV) and subsequent comparison of the performance metrics of all models for forecasting mean ambient air temperature in Mars' Gale Crater. Explicit tuning does not occur for MLR in this study since this model is most heavily impacted by the selection of inputs (i.e., data). This study is not an investigation into the different regression algorithms themselves, but rather it is an applied study of their comparative performance on the selected dataset.

B: Dataset

Table I. Features and targets for study.

Feature*	Description	Target
Ground Temperature (K)	The brightness temperature measured by a thermopile on boom 1.	Mean Ambient Air Temperature (K)
Boom Air Temperatures (K)	*Separate* local air temperatures for REMS booms 1 and 2.	
Local Relative Humidity (%)	Local relative humidity at humidity sensor.	
Atmospheric Pressure (Pa)	Pressure.	
Ambient Air Temperature** (K)	Estimated ambient air temperature.	
28 x 17 Total Features, 7 x 1 Total Targets		

*The minimum, maximum, and mean of a feature each of 28 sols prior.
**Only the minimum and maximum of this feature were computed.

The weather data used in this study are available through NASA's Planetary Data System (PDS). Data for 2837 Martian days (sols) were collected via the Rover Environmental Monitoring System (REMS) onboard NASA's Curiosity Rover. Note, Martian sols are approximately equivalent in duration to Earth days, and thus the word sol and day are often used interchangeably. Each row of a data product representing one sol is composed of different columns for time references, wind sensor products, ground temperature products, air temperature products, ultraviolet sensor products, humidity sensor products, and pressure sensor products [20].

Sampling for each row is taken at 1 Hz maximum, with baseline operation of 5 minutes every hour. While each sol therefore contains time series data, only medium-

range (3-7 days) forecasting was be attempted in this study. Consequently, Table I lists the relevant features (independent variables), which are estimators (minimum, mean, of a maximum) of a given sol, that were used in this study. The estimators of a given sol were selected because the same estimators were used in the datasets of several previous studies [21]–[23].

C: Preprocessing for Time Series Forecasting

Time series data are data that are sequential with respect to time. There are many common examples of time series data, such as fluctuating sea level, counts of sunspots, and daily closing values for stock market indices [24]–[26]. In all cases, some variable is always considered with respect to the time at which such information was collected. In weather prediction, the relevance and structure is quite intuitive since the reader has likely referred to weather forecasts in his or her daily life when making decisions about what to wear, whether to carry an umbrella, etc.

To understand the format of weather data, consider a single variable such as temperature. On the present-day *t*, the temperature might be 70 °F while the two previous days (days *t-1* and *t-2*) might be 67 and 68 °F, respectively. However, the temperature of the next 7 days is unknown. The expression that can represent the list, or window, of the next 7 days temperature is this: (*t+1, t+2, t+3, t+4, t+5, t+6, t+7*). Fundamentally, this is how time series data are divided and the "previous-day" window is called the time lag while the "future-day" window is called the forecast horizon.

For this study, data were windowed with a time lag of 28 (i.e., *t, t-1, t-2, …, t-27*) and a forecast horizon of 7 (i.e., *t+1, t+2, …, t+7*). The size of the time lag and forecast horizon are problem specific and often influenced by some interpretable concept such as quarters of a year (in the case of financial data) or, in this study's case, 1 previous month of data for a 7-sol forecast.

Moreover, it is important to divide data into a training, validation, and test set. If this procedure is not followed, then any evaluation of the model will be positively biased, that is the model will appear to be much better than it is. Data are split in the following proportions: 60% training, 20% validation, and 20% test. This is a typical splitting pattern in machine learning projects [27].

Differentiating the validation and test set is important. The validation set is "seen" multiple times by the model after it has been trained. The validation set is used to evaluate how well the model can generalize to new cases. However, since the same validation set is used multiple times after adjusting (i.e., optimizing) the hyperparameters (learning rate, model architecture, etc.) of the model, the generalization error for different models is only measured using a single validation set. Therefore, the test set contains data that *none* of the candidate models—each with different hyperparameters—has ever "seen," and this set is used at the very end of the study to evaluate and select the best candidate model [28], [29]. Moreover, at no point in the experimentation process do any models have access to the test set. This includes during transformation processes such as normalization and imputation, discussed shortly.

Min-max normalization was be applied to all features as given in [15]. This practice transforms data to a common scale to (1) allow for a model to learn more

efficiently; and (2) to make the comparison of models straightforward and constant [30]. For this study, the min-max normalization range is from zero to one, thus all variables (features and targets) were transformed to this range [31]. This follows the normalization procedure of previous work [15].

$$data = \frac{data - min\ (data)}{max(data) - min\ (data)} \quad (1)$$

As is the nature of data collection, some data are missing. Instruments can fail or data can be screened for anomalies that are subsequently removed. For this study, approximately 4% of sols were missing from when collection started on sol 1 to when collection ended on sol 2837. The task of generating values that are estimates of missing values is known as imputation. There are many techniques in missing value imputation; however, one of the most powerful imputation methods is known as the extra trees method—which is similar to RFs but instead "views" an entire input dataset rather than bootstrapped (random samples with replacement) samples [32]–[35]. While the extra trees method suffers from higher variance due to using the entire input dataset for training, imputation occurs more quickly and thus makes it preferable to the random forest implementation on which the Scikit-Learn implementation is based [33], [36], [37].

For multivariate imputation, often the iterative imputation method (in this case extra trees) will not converge to an optimal solution. For this study a max of 5 iterations was selected for the imputer and the effects of non-convergence were safely ignored

since Oberman et al. [38] showed that inferential validity can be achieved after 5 to 10 iterations.

The input to all algorithms was 28 days of feature data, and the output was the mean ambient air temperature for the next 7 sols (see Table I).

All data preprocessing and plotting were conducted using the Scikit-Learn, NumPy, SciPy, Matplotlib, and Pandas libraries for Python [33], [39]–[42].

D: Independent, Joint, and Iterative Models for Time Series Forecasting

When building models to make predictions on time series data, three possible strategies are available: independent, joint, and iterative models [43].

As discussed previously, a time series window for the future is called a forecast horizon. For independent models, one constructs a model that specifically learns to predict future values only at that time step. Both RF and MLR algorithms belong to this class of strategies, so seven models are constructed for each algorithm, one model for each day in the future. Neither MLR nor RFs are explicitly designed for multi-step predictions, which is the case for this study since multiple steps (sols) in the future are predicted. As a result, the independent model strategy is one strategy that can accommodate the problem formulation for this study.

Joint models are *singular* models that predict all values in the forecast horizon at once as opposed to constructing a *separate* model for each step in the forecast horizon (as is the case for independent models). The single model strategy was implemented with one

variation of RNNs in this study. RNNs that predict all steps in the forecast horizon at once are termed "joint" for this study.

Iterative models are those models that assume there is a relationship between sequential timesteps that cannot be ignored. This relationship is known as autocorrelation, and essentially requires that the model *iteratively* predict *t+1* then use *t-27, t-26, …, t* **and** *t+1* to predict *t+2*. The model then adds *t+2* to its internal list (or state) and predicts iteratively until the weather data for *t+7* are predicted. At the end of iterative prediction process, the desired target variable will be predicted in a time indexed list where each *t* is a timestep (day) like (*t+1, t+2, t+3, t+4, t+5, t+6, t+7*).

E: Multiple Linear Regression

The goal of multiple linear regression is to find a parametric function that minimizes the squared difference between the prediction $\hat{Y}$ for ambient air temperature and the actual value *Y* for the ambient air temperature. The function itself is a linear combination of *N* total independent variables X_n plus random error ε [22]. The parameters learned by MLR are the values β_n that correspond to each X_n. More commonly the set of independent variables is represented by an input vector $X^T = (X_1, X_2, \ldots, X_n)$ and $\hat{Y}$ is a function of each *X*. The relationship between $\hat{Y}$ and *X* is described in equation (2) from [44].

$$\hat{Y} = f(X) = \beta_0 + \sum_{n=1}^{N}(\beta_n X_n) + \varepsilon \quad (2)$$

Since there are $N = 476$ possible independent variables (features) in this study, then there are $2^N = 2^{476} = 1.95 \times 10^{143}$ possible parametric models that could be considered to produce the best estimated response $\hat{Y}$ of the actual response Y [45], [46]. This means that there are more combinations of MLR models for this study than there are atoms in the universe. To illustrate this concept with more manageable numbers, Table II demonstrates the number of possible models if there are only two features ($N = 2$) [45].

Table II. Possible MLR Models Given $N = 2$ Features.

Model	Equation
1.	$\hat{Y}_1 = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$
2.	$\hat{Y}_2 = \beta_0 + \beta_1 X_1 + \varepsilon$
3.	$\hat{Y}_3 = \beta_0 + \beta_2 X_2 + \varepsilon$
4.	$\hat{Y}_4 = \beta_0 + \varepsilon$

Testing each model combination would be impossible, and techniques such as backward elimination and principal component analysis are commonly used to select the most important features for a study [47]. However, while feature reduction techniques were employed originally for this study, they were abandoned due to failure of the dataset to meet assumptions that are required for MLR to be considered a robust and valid model [48]. This concept is elucidated in the discussion and two statistical tests are used to illustrate the dataset's failure to meet the required assumptions [49], [50].

MLR uses the method of ordinary least squares to adjust its parameters. In this way, MLR learns a function with a number of parameters equal to the number of features in the dataset. Such a function would essentially be Equation (2) but with β_0 through β_{476} possessing different values that would uniquely modify the input data to produce the desired output.

The implementation of the MLR model was via Scikit-Learn and statistical tests pertaining to the assumptions of MLR were carried out via Statsmodels [33], [51].

F: Random Forest

RFs are an ensemble learning method. The RF learning mechanism is to aggregate multiple individual decision trees for—in this study—regression problems and produce an average prediction [52]. A decision tree is simply a tree-like model of decisions and their possible consequences. The subsequent aggregation of all these individual decision trees into a forest yields greater accuracy and prevents overfitting associated with individual instances of decision trees [53].

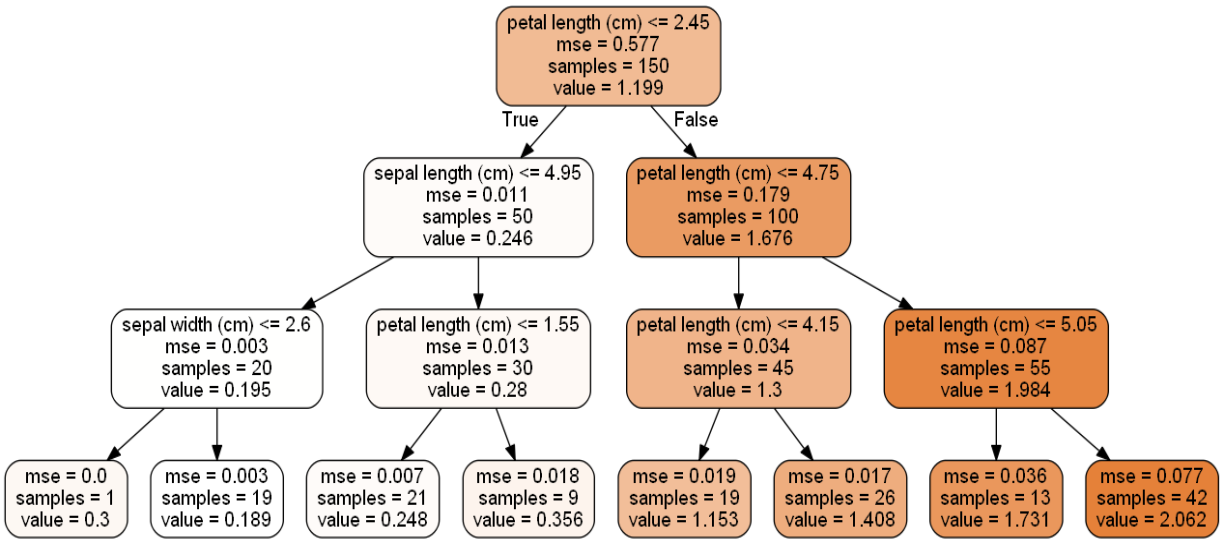


Figure 2. Decision tree regressor for canonical iris flower dataset.

Since the subunit of a random forest is a decision tree, Figure 2 depicts both a decision tree and the process by which it makes decisions. An adaptation of the well-known iris flower classification problem is used to illustrate a decision tree regressor. The adapted features for this problem are petal length, sepal length, and sepal width while the target for the decision tree is to predict the petal width. Note that this adaptation is somewhat trivial since if one possesses the iris flower, one can easily take the appropriate measurements without relying on a decision tree. Nevertheless, at each square (node) in the tree, information about a feature informs whether to proceed left or right down the tree. For example, if an iris has a petal length that is less than or equal to 2.45 cm, then the tree is traversed from the first node to the node on the left in the second row of nodes. This process continues until a prediction is made.

The learning criteria for decision trees, and consequently random forests, is minimizing the mean squared error ("mse" in the figure) between the current node's

prediction (value) and the known target value. Nodes make predictions by averaging the associated target values for the *n* samples of the node, and the ends (leaves or leaf nodes) of the decision tree have the lowest mean squared error.

A hyperparameter, or those parameters that are intrinsic to a particular model, that is also applicable to RFs for this study can be elucidated using Figure 2. The maximum depth of the decision tree is the maximum number of branches (arrows) until a leaf node is encountered. For Figure 2, the maximum depth is 3. This hyperparameter can be changed to increase or decrease the risk of overfitting (overlearning a problem and failing to generalize to new problems).

The random forest trains *n* decision trees known as *n* estimators and then averages the predictions across all decision trees. However, unlike decision trees, which select the best feature to split a node on based on the learning criteria for *all* features, the RF searches for the best feature among a random *subset* of features [34]. Moreover, the training set for the i^{th} decision tree is a bootstrap sample of the original training set [31]. Bootstrapping is the act of randomly sampling with replacement. The bootstrapping sampling method is known to provide a more diverse ensemble from which an accurate RF model prediction could arise [54].

RFs are a reputable choice for high-dimensional data and are reported to have ease of implementation for large datasets according to [55] and [23]. Additionally, RFs are capable of handling complex, non-linear relationships and are one of the most powerful machine learning algorithms available today [34].

The implementation of the RF model is available via Scikit-Learn [56]. The *n* estimators and max depth hyperparameters that were tuned for RFs are discussed in the results and discussion section.

G: Artificial Neural Networks

The bulk of this study was spent designing, tuning, and troubleshooting the class of machine learning algorithms known as artificial neural networks (ANNs).

ANNs are biologically inspired algorithms that, like the previous models discussed, learn parameters such that better predictions can be made [57]. ANNs are composed of neurons and can have multiple layers to learn some complex function such as the function that describes weather forecasting for a particular system. Figure 3 depicts the simplest form of an ANN, commonly referred to as a multilayer perceptron. ANNs are particularly useful as they do not require intimate knowledge of a system—like the complicated numerical methods and computational fluid dynamics used in atmospheric science—to make powerful predictions.

ANNs learn a function's parameters for a given system by performing gradient descent in error. The function that models the error between predictions and known targets is referred to as an objective, cost, loss, or error function [58]. The proverbial example of gradient descent is imagining a hiker trying to get the lowest point in a mountain range. Many routes will lead up the mountain, many routes will lead to spots that are lower than the hiker's current point, but not necessarily the lowest possible point in the mountain range. Thus, it is the task of the neural network to find such a path in a

(literally) unimaginable *N*-dimensional mountain range to the lowest point in said mountain range.

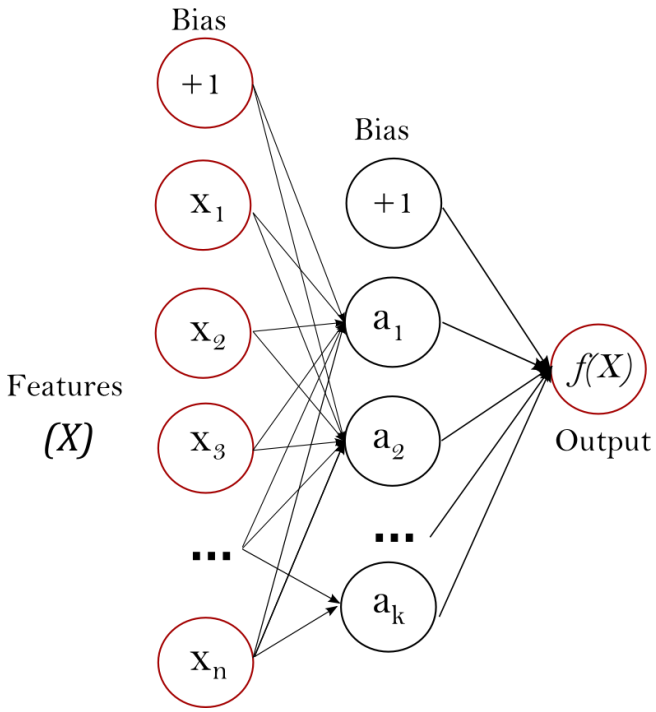


Figure 3. Feedforward neural network architecture. Single hidden layer neural network with *k* hidden neurons denoted by the black circles [33].

As the culmination of the study's complex models, a total of nine unique neural networks were implemented using TensorFlow and Keras [59], [60]. The mean squared error was used as the loss function for training of all neural networks (see Equation 3).

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (3)$$

The number of times the ANN "sees" the entire training dataset is called training epochs. The number of epochs used for all ANNs was 32. To prevent overfitting, which is learning parameters for a neural network such that the network performs poorly on unseen data, early stopping is implemented with TensorFlow callbacks. The early stopping value monitored was validation loss (the performance of a trained model after each epoch on the validation batches). The early stopping criteria was to halt model training if validation loss did not improve (i.e., decrease) after 4 consecutive epochs. The batch size, which is the subsample of data that is used to make parameter updates, was 28. The batch size was set to 28 because cross validation (described later) required that the data be divided into subsamples of 28 to meet the 28-day window corresponding to a single Martian month. The Adam optimizer was used for all ANNs. Other hyperparameters that apply to specific models for this study are listed as tables in the results and discussion.

H: Recurrent Neural Networks

As mentioned, weather data are time series data as they are collected at regular temporal intervals and have some sequential relationship. This time dependence is ideal for recurrent neural networks (RNNs) since RNNs can use their internal state to process variable length sequential input for the task of weather forecasting [61].

Several architectures exist for RNNs. The variations on the RNN are primarily based on changing its fundamental unit known as the *cell* in order to mitigate problems wherein long-sequences "confuse" the RNN and prevent it from learning a meaningful

function [62]. Thus, the LSTM and GRU cell were developed to alleviate instances where an RNN would fail for long-sequences.

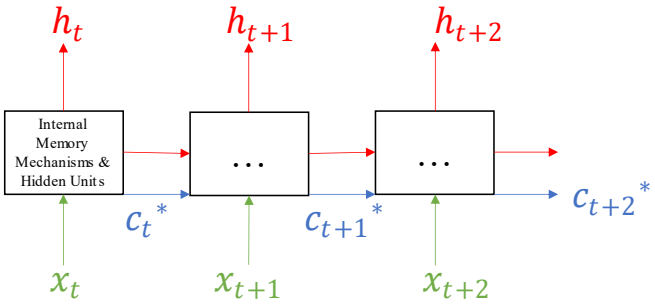
The number of cells in the RNN determines the output for a given time series. In its simplest form, an RNN has cells equal to the number of time steps in a given input matrix. A matrix, here, is simply a table where the number of the rows is the number of timesteps for a set of data and the number of columns is the number of features for the same data. Table III depicts example input to an RNN.

Table III. A sample time series window with two features.

Timestep	Relative Humidity (%)	Temperature (K)
0	0.97	300
1	0.53	275
2	0.75	286

The output from an RNN is one of three objects and is influenced by its own input at each timestep in the time series window. The first object, called the hidden state at a timestep *t* is denoted by h_{t+i} where *i* is from 0 to the total number of timesteps *T* in the time series window. Referring to Table III, *T = 3*. The h_{t+i} object is the direct result of the computations of the RNN cell and, as noted, there are three cells in Figure 3, one for each timestep in Table III. The network is recurrent in that it passes the h_{t+i} object to the next cell in the sequence of RNN cells. If the cell is an LSTM cell, a special memory state called the cell state c_{t+i} is passed to the next cell also. The final output of the RNN

is a matrix of hidden states consisting of a single hidden state per timestep, the last hidden state computed in the RNN sequence, and (in the case of LSTM cell) the last cell state computed in the RNN sequence.



Final RNN Output Shapes

$$H \in \mathbb{R}^{timesteps \times hidden\ units}$$

$$h_T = h_{last} \in \mathbb{R}^{hidden\ units}$$

$$c_T{}^* = c_{last}{}^* \in \mathbb{R}^{hidden\ units}$$

Figure 4. Unrolling of recurrent neural network over $T = 3$ timesteps. An asterisk * indicates an output that is only for the LSTM cell. Cells are depicted as black squares.

Cebeci [14] reported that both the Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) architectures are ideal for weather data. However, while Cebeci [14] reported that GRU-RNNs were computationally less expensive (faster) to train, LSTM-RNNs were deemed more accurate and therefore preferable for weather forecasting problems

RNNs are implemented in one of many ways for this study: stacked (neural network layers are stacked sequentially on one another to improve performance), RNNs using GRU or LSTM cells, and/or RNNs making iterative or joint predictions—as discussed previously. The iterative variation is implemented by using the last hidden states of the previous RNN to "warm-up" the states of the autoregressive (AR) net. The

AR neural net then iteratively predicts 7 more hidden states, 1 state for each step in the forecast horizon. These 7 hidden states are then passed to a densely connected (see Figure 3 for dense connections) layer that outputs the 7-sol forecast for the mean ambient air temperature. The iterative prediction architecture was inspired by a time series forecasting tutorial written by the contributors to TensorFlow [59]. The figure below depicts the variations of the RNN models designed for this study. Specific hyperparameters for this model are detailed in the results and discussion section.

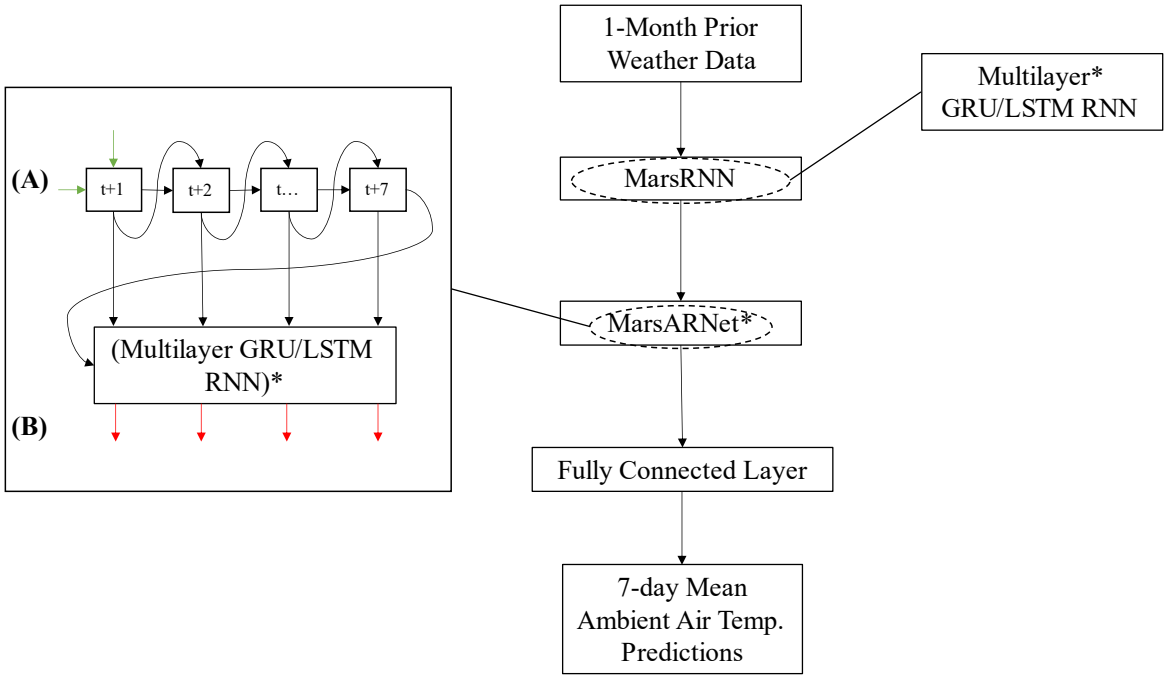


Figure 5. Recurrent neural network architecture. Items marked with an asterisk * indicate they are an optional hyperparameter for the architecture. **(A)** Depiction of the iterative prediction process. **(B)** The output from the MarsARNet is always of shape (sample_size, timesteps = 7, num_labels = 1).

I: Convolutional Neural Networks

CNNs are inspired by the way the eye processes visual stimuli [63]. The process can be applied to time series because much in the way visual perception is handled, sub-windows of a given time lag (the previous days of a month in this study) can be used to predict the forecast horizon. This method significantly reduces the number of parameters needed to perform such a prediction, and a reduction of model complexity favorably reduces model variance [64].

The convolution operation that is key to CNNs can be carried out in one of two ways: non-causal or causal [65]. Also, 1D convolution was used for this study because time series data, despite using a 3D input shape (samples, timesteps, dimensions), uses the 1D convolution operation. Whether the convolution is causal or non-causal, the convolution operation has two hyperparameters known as kernel and stride. The kernel is essentially the size of a sliding window over an input, and the stride is the number of steps in between subsequent sliding windows. The below figure shows how the time series window evolves over time.

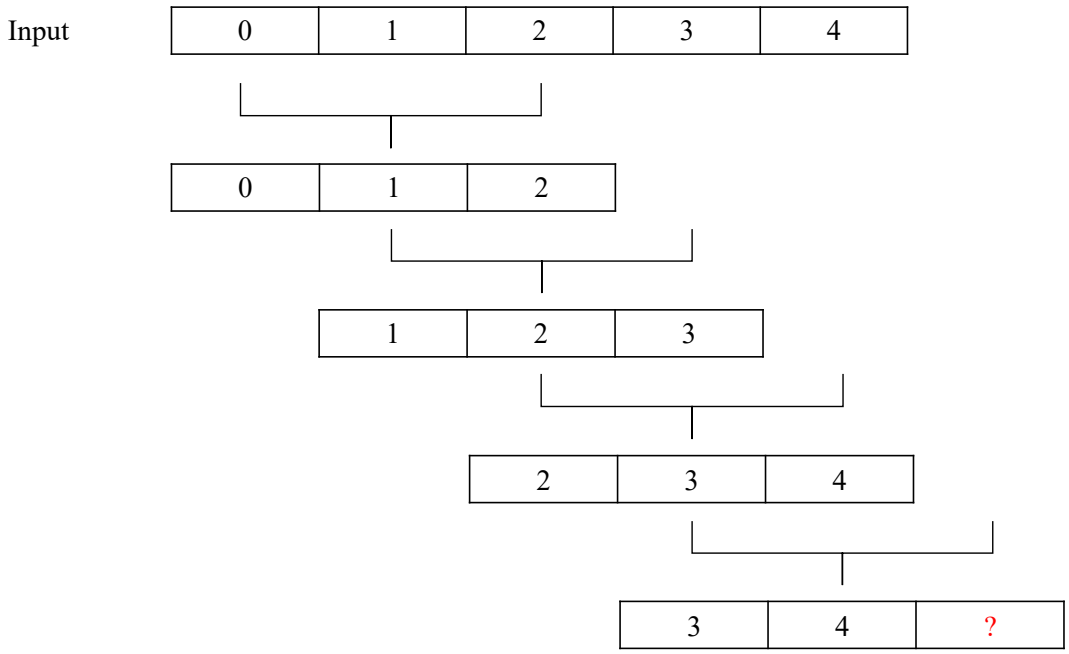


Figure 6. Convolution with kernel size 3 and stride length 1 on example time series data.

Below the input data in Figure 6, 4 sub-windows of length 3 appear. The size of the sub-window is determined by the kernel, and the index difference between sub-windows is determined by the stride. Since the stride in Figure 6 is 1, the difference between the first elements of adjacent sub-windows is 1. Note that the final sub-window has a red question mark "?" that denotes a failure of the remaining sub-window to slide over a value in the input data. If this "failed" sub-window is simply dropped from the convolution operation, this non-causal convolution employs *valid* padding [60]. If instead, the question mark is replaced with a value of 0, this non-causal convolution employs *same* padding [60].

A disadvantage of the non-causal convolution is that when employing valid or same padding, an input sequence can only be continued and not started. This means that in non-causal convolution, the output is dependent on future inputs and predictions can

only be made for future inputs in the input frame. In contrast, causal convolution left pads an input sequence with zeros to make sure the output at timestep *t* does not depend on the inputs at *t + 1* [59], [63]. This padding technique is particularly useful when one wants to prevent the network from "peeking" into the future—which one might intuitively want to prevent for an autoregressive problem.

For both types of convolutions, the time series is chopped into smaller windows and then passed on to deeper layers in the network

A common way to improve the performance of CNNs is to implement batch normalization, which standardizes outputs between layers, and max pooling, which subsamples hidden layer outputs to decrease computational complexity.

Many design considerations are readily apparent for CNNs, but for time series analysis, this study included convolutional dilation that was influenced by the dilation rate of the seminal WaveNet architecture [66]. As a result, the dilation rate was tuned to be a power of 2^n (e.g., 1, 2, 4, 8, …. 256, etc.) where the exponent *n* is determined by the number of convolutional layers in the network. Alternatively, the dilation rate could be held constant during tuning, though it has been shown that dilation improves model performance in a variety of tasks [67]–[70]. Moreover, the previous work in machine learning applications for mars using CNNs was also useful as a simple reference architecture [13]. For the current study, both non-causal and causal convolution techniques were tuned so valid, same, and causal padding were set as padding options. Batch normalization and max pooling were also implemented to regularize the CNN and thus improve its performance.

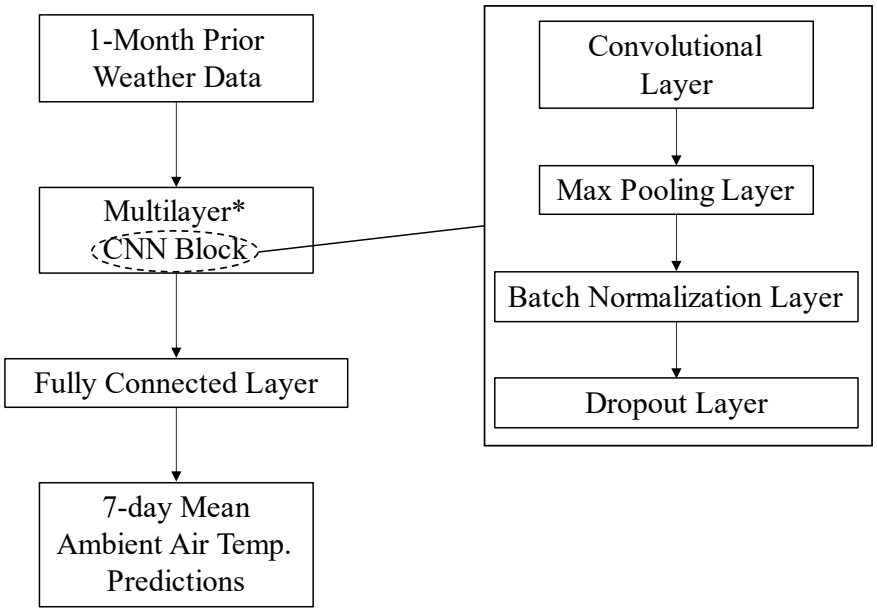


Figure 7. Convolutional neural network architecture. Items marked with an asterisk * indicate they are an optional hyperparameter for the architecture.

J: Cross Validation and Hyperparameter Tuning

The aim of cross validation (CV) is to (1) split data once or several times; (2) train a model with input and target outputs; and (3) evaluate how well the model performs on what is *initially* an "unseen" validation dataset [71]. Often, the splitting of data once or several times occurs randomly and involves the shuffling of data. Typical CV (k-fold CV) disregards the intrinsic time dependence of time series data and is therefore not ideal. For this study, rolling-origin-recalibration cross validation (RORCV) was be used to maintain time dependence as well as test models with adjusted hyperparameters [72].

Sometimes called walk-forward cross validation, the RORCV method maintains the temporal dependence of the data by adding the *N+1* validation set to the current

training set where *N* is the temporal split desired. The size of *N* is influenced by the data itself, and a size $N = 5$ was selected since this divides the Martian data into 5 windows of approximately 1 Martian year in length. In this way, sequential years are added to the training set in temporal "blocks" until no more blocks are available (i.e., year 1 predicts year 2, then year 1 **and** 2 predicts year 3, etc.). Refer to Figure 8 for a depiction of the walk-forward cross validation indexing process.

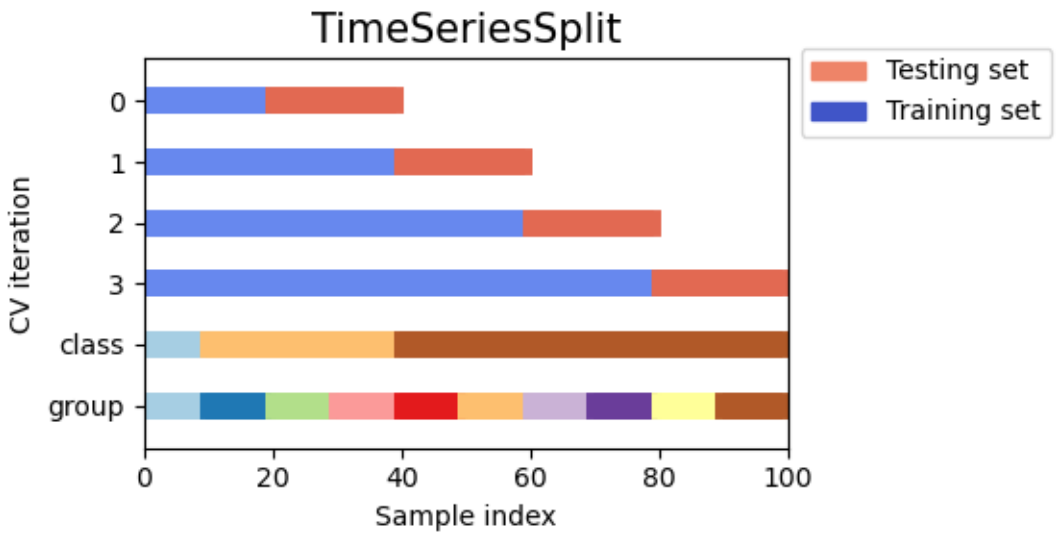


Figure 8. Data partitioning using walk-forward cross validation [33].

Hyperparameters are not parameters of the model itself, see Table II for a model with two parameters X_1 and X_2, but rather those "settings" (e.g., number of hidden layers, learning rate η, etc.) of the machine learning algorithm itself [73]. Comprehensive tables of the hyperparameters are available in the results and discussion section.

Hyperparameters can be changed (tuned) such that a model performs with a better desired performance metric. For this study, automatic hyperparameter tuning via Bayesian optimization was be used. Hyperparameters differ for each machine learning algorithm in this study; however, the MLR algorithm was not regularized and therefore has no hyperparameters to tune. RFs, RNNs, and CNNs have many hyperparameters that can be tuned but the following hyperparameters were selected for each model, respectively: (1) number of decision trees in forest and max depth of trees; (2) number of hidden units, number of hidden layers, GRU or LSTM unit for RNN cell, iterative or joint model, and layer normalization rate; (3) number of convolutional filters, filter increase rate between multiple layers, dropout rate, padding strategies, and dilation rate [74]. For all ANNs, the learning rate for the Adam optimizer was also tuned. Further, Bayesian optimization as an automatic hyperparameter tuning algorithm is a strong choice since the RF and RNN models will not be applied to other datasets and the current dataset is not of substantial size [75].

Bayesian optimization was implemented using Keras-Tuner [76]. The maximum number of trials for the Bayesian optimization was 10. As implemented in Keras-Tuner, Bayesian optimization initially selects a random set of hyperparameters from a list of choices for each hyperparameter. After this initial selection, which hyperparameters to explore further is determined by training and validating via 5-fold time series cross validation a given model and updating the Bayesian optimization's search parameters based on the average validation loss for a particular model. This process repeats for a total of 10 (i.e., max number of trials) times such that 10 combinations of hyperparameters are tested and the best set of hyperparameters can be reported.

K: Performance Metrics and Final Evaluation

MAE, MSE, and RMSE were used to compare the different models in this study. These metrics are the most common for time series forecasting [77].

The MAE is used to define the absolute difference between actual values of the mean ambient air temperature *y* and a given model's prediction $\hat{y}$ over a sample of size *N* [13].

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \quad (4)$$

The MSE (see Equation 4) is like MAE except the difference in model predictions and actual values of the mean ambient air temperature are squared.

The RMSE is simply the square root of the MSE and is commonly reported in forecasting domains.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \quad (5)$$

For each model, 5-fold cross validation allows for the computation of confidence intervals, and this strategy was used for both the hyperparameter tuning and final evaluation of the models. Confidence intervals (CIs) are the range of estimates for an unknown parameter—in this case the MAE, MSE, RMSE—and the likelihood that the unknown parameter falls within that range [78]. For this study, 95% confidence intervals were used.

IV. RESULTS AND DISCUSSION

Table IV. Metrics for all models' performance for 7-sol mean ambient temperature forecasting. Sorted in ascending order along MAE.

Models	Metrics (K)		
	MAE	MSE	RMSE
gru_unstacked_iterative	2.31	10.98	3.25
lstm_stacked_iterative	2.39	11.58	3.38
cnn	2.76	14.98	3.87
mlr	2.88	15.78	3.93
lstm_stacked_joint	3.22	15.96	3.99
rf	3.23	15.53	3.92
gru_stacked_joint	4.1	25.29	4.99
lstm_unstacked_iterative	4.24	28.15	5.25
gru_unstacked_joint	4.28	27.38	5.19
gru_stacked_iterative	4.3	25.03	5
lstm_unstacked_joint	6.26	58.58	7.51

Table IV depicts the performance evaluation for all the models considered for this study. In order of design, the algorithms considered for this study were MLR, RF, CNN, GRU-unstacked-joint, GRU-unstacked-iterative, GRU-stacked-joint, GRU-stacked-iterative, LSTM-unstacked-joint, LSTM-unstacked-iterative, LSTM-stacked-joint, and LSTM-stacked-iterative. The statistical parameters used for the evaluation were MAE, MSE, and RMSE.

Therefore, a total of eleven models were implemented for this study. All metrics were computed by taking the mean of the metrics computed during 5-fold walk-forward

cross validation on the training set and validation[1] set. MLR and RF do not have epoch nor batch size hyperparameters, but for the ANNs the batch size was 28 to reflect the number of sols in a Martian month, the number of epochs was 32 with the same early stopping parameters as mentioned in the artificial neural networks section of the methods, and the hyperparameters for all models will be discussed in the proceeding sections. Table XII shows the mean epoch and 95% confidence intervals on which early stopping occurred. Notably, the LSTM unstacked iterative model has a negative lower bound for the confidence interval due to 4 out of the 5 cross validation folds terminating around 6 epochs, while 1 of the folds terminating near 30 epochs. This high variance led to the physically uninterpretable lower bound for the CI.

A: Multiple Linear Regression

Table V. Assumption violations for linear regression.

Gauss-Markov Property	**VIFs > 5**	**Reject Goldfeld-Quandt H_0**
Multicollinearity	True	--
Homoscedasticity	--	False

From Table IV, the performance of MLR with respect to MAE and relative to all other models made MLR the fourth best performing model. MLR's value for MAE was 2.88 K and Table X in the appendices shows that the 95% CI is in (2.45, 3.30). Also from Table IV, the performance of MLR with respect to RMSE and relative to all other models made MLR the fifth best performing model. MLR's value for RMSE was 3.93 K and

[1] This is not the final evaluation of models but should approximate the test set's error.

Table XII in the appendices shows that the 95% CI is in (3.13, 4.73). MSE and RMSE are proportional, and values of MSEs are described both in the appendices and in Table IV.

The RMSE indicates poorer model performance than the MAE; however, and this will hold true for all discussions in this thesis regarding metrics, the RMSE is more sensitive to the magnitude of errors and MAE is considered a more unambiguous measure of inter-model performance [79].

Despite the performance of MLR relative to other models, the robustness and statistical validity of MLR are questionable. MLR is subject to several key assumptions known as the Gauss-Markov assumptions, and failure to meet one or more of these assumptions about the distribution of errors, normality, and/or linearity can jeopardize the integrity of the model [49].

Table V shows two assumptions that the MLR model was unable to meet: no multicollinearity among features and the variance of errors is the same (aka homoskedasticity).

Multicollinearity was tested by using variance inflation factors (VIFs) for each feature in the dataset. If the VIF for a given feature is greater than five, then there exists multicollinearity (or correlation) between that feature and at least one other feature in the dataset [51]. For all 476 (28 * 17) features, the VIF was greater than five. Therefore, all features in the dataset had multicollinearity with at least one other feature. Multicollinearity can be alleviated by using backwards elimination or dimension reduction techniques such as principal component analysis; however, to maintain a constant dataset across models, these techniques were not performed [80], [81].

Homoscedasticity is the property that the variance of the residuals is a uniform distribution, and the null hypothesis of the Goldfeld-Quandt test is that variance in one subsample of residuals is larger than in the other subsample [51], [82]. If the null hypothesis is rejected, then the alternative hypothesis is that the variance of residuals is homoscedastic (or uniform). Table V shows that the null hypothesis was not rejected, therefore it was concluded that the distribution of the residuals is non-uniform and therefore heteroscedastic.

Given that MLR violates at least two core assumptions of the model, its performance relative to the other models should be only tentatively accepted.

B: Random Forests

Table VI. Hyperparameter choices for random forest.

HP	**BOPT Choices**	**HP Selected**
max_depth	8, 16, 32	32
min_samples_leaf	NA	1
min_samples_split	NA	2
n_estimators	64, 128, 256	128

From Table IV, the performance of RF with respect to MAE and relative to all other models made RF the sixth best performing model. RF's value for MAE was 3.23 K and Table X in the appendices shows that the 95% CI is in (2.84, 3.62).

From Table VI, the best hyperparameter for the maximum depth of the i^{th} decision tree in the random forest was 32 while the number of decision trees in the forest (aka "n_estimators") was 128. It is unsurprising that the RF model outperformed about half of

the ANNs since RFs are notoriously powerful and, since RFs are an independent value model, the propagation of errors over time is expected to be minimized compared with iterative neural network models [83].

Despite the intuition that the independent value model should have less error propagation over time compared with the iterative models, the RF does not appear to have substantially less variance. The evolution of all metrics over the 7-sol forecast is depicted in the figures in the appendices. Figure 13 shows the evolution of MAE with 95% confidence intervals over each timestep. While the CIs for the RF appear narrower in Figure 13 when compared to the LSTM-unstacked-iterative model in Figure 16, the MAE over time for the LSTM-stacked-iterative model in Figure 18 appears to have a narrower CI than the MAEs for the RF. These observations indicate that the RF avoids the error propagation pitfalls of shallow, iterative LSTM models, but is not quite comparable to the deeper, iterative LSTM models.

C: Artificial Neural Networks

Table VII. Hyperparameter choices for convolutional neural network.

HP	BOPT Choices	HP Selected
cnn_activation_function	NA	relu
num_layers	1, 2, 4, 8	2
padding	valid, same, causal	valid
strides	NA	1
dilation_rate	1, 2	2
dropout_rate	0.0, 0.16, 0.32, 0.64	0.32
filter_increase_rate_per_layer	1, 2	2
filters	32, 64, 128	128
kernel_size	NA	3x3
use_max_pooling	NA	true
pool_size	NA	2
pool_padding	NA	valid
dense_size	NA	7
dense_activation	NA	none
learning_rate	1e-2, 1e-3, 1e-4	1e-2

From Table IV, the performance of the CNN with respect to MAE and relative to all other models made the CNN the third best performing model. The CNN's value for MAE was 2.76 K and Table X in the appendices shows that the 95% CI is in (2.53, 2.99).

Table VII shows the best hyperparameters for the CNN. Unsurprisingly, the dilation rate and filter increase rate per layer were the highest of the available BOPT choices. The dilation rate agrees with the principles of many architectures that were discussed under the convolutional neural networks section of the methods. Moreover, the filter increase rate of 2 means that for subsequent layers, the number of filters is doubled. This means for the first convolutional block, there were 128 filters, and the subsequent block had 256 filters. It is unexpected that causal padding was not the optimal padding strategy. The causal padding, which has had success on long sequence time series as

discussed in the methods section, was not the selected padding method. Instead, valid padding was selected. One explanation for this might be that the time lag was not large enough for causal padding to be effective at improving model performance. The time lag was only 28 sols, which is a short sequence length compared to more complicated sequences that are typically handled with causal padding.

Table VIII. Hyperparameter choices for LSTM model variants.

HP	**BOPT Choices**	**HP Selected for LSTM Model Variant**			
		USJ	USI	SJ	SI
autoregressive	NA	false	true	false	true
dropout_rate**	0.0, 0.16, 0.32	0.0	0.32	0.0	0.0
rnn_cell	NA	lstm	lstm	lstm	lstm
rnn_layers*	1, 2, 4, 8	1	1	2	4
rnn_size	32, 64, 128	32	128	32	32
learning_rate	1e-2, 1e-3, 1e-4	1e-3	1e-4	1e-3	1e-2
dense_size	NA	7	7	7	7
dense_activation	NA	none	none	none	none

*****For the unstacked models, rnn_layers = 1 but for stacked models the choices are 2, 4, 8.
******Not recurrent (i.e., intra-layer) dropout but rather inter-layer dropout.

From Table IV, the order of best model performance on MAE for the LSTM variants: stacked-iterative > stacked-joint > unstacked-iterative > unstacked-joint. The obvious observation here is that the deeper network outperforms the shallow network. The respective mean MAE values (in Kelvin) and 95% CIs for these models are as follows: 2.39 in (2.13, 2.66); 3.22 in (3.08, 3.37); 4.24 in (2.66, 5.82); and 6.26 in (3.53, 9.00).

Of the LSTM models, the LSTM-stacked-iterative model was the second-best performing model with respect to all models, while the LSTM-unstacked-joint model was

the worst-performing model with an upper bound on the confidence interval of ca. 1.5 times more than the upper bound of the second worst-performing model.

With far fewer hyperparameters, interpretation of the LSTM model variants in Table VIII and their respective performance in Table IV is more straightforward. Even when the unstacked-iterative model has more hidden neurons per single layer compared with the stacked models, and the unstacked-iterative model has the slowest learning rate, it cannot outperform the stacked models.

Table IX. Hyperparameter choices for GRU model variants.

HP	**BOPT Choices**	**HP Selected for GRU Model Variant**			
		USJ	USI	SJ	SI
autoregressive	NA	false	true	false	true
dropout_rate**	0.0, 0.16, 0.32	0.32	0.16	0.16	0.32
rnn_cell	NA	gru	gru	gru	gru
rnn_layers*	1, 2, 4, 8	1	1	8	8
rnn_size	32, 64, 128	32	32	32	64
learning_rate	1e-2, 1e-3, 1e-4	1e-4	1e-2	1e-4	1e-3
dense_size	NA	7	7	7	7
dense_activation	NA	none	none	none	none

*****for the unstacked models, rnn_layers = 1 but for stacked models the choice are [2, 4, 8].
******not recurrent (i.e., intra-layer) dropout but rather inter-layer dropout.

From Table IV, the order of best model performance on MAE for the GRU variants: unstacked-iterative > stacked-joint > unstacked-joint > unstacked-iterative. The respective mean MAE values (in Kelvin) and 95% CIs for these models are as follows: 2.31 in (1.74, 2.89); 4.1 in (2.84, 5.37); 4.28 in (2.90, 5.65); and 4.30 in (3.94, 4.66).

Of the GRU models, the GRU-unstacked-iterative model was the best performing model with respect to all models, while the remaining GRU models were in the bottom half of the model performance.

GRU RNNs and LSTM RNNs have been shown to outperform one another on some tasks while paradoxically performing so similarly as to be ambiguous on others [14], [84]–[87]. To illustrate the overlapping intervals of all models, refer to Figure 9 that compares cross validated model performance using MAE.

Despite the GRU-unstacked-iterative model performing the best with respect to mean MAE across all cross validation folds, the CIs for the top 4 models all overlap. This indicates that extensive hyperparameter tuning and careful cross validation were not sufficient to demonstrate a clear best-performing model. This overlap might be resolved by using *N*, 5-fold walk-forward cross validation with the random seed for weight initialization set to different values for each CV replicate so that the neural networks weight initialization and data shuffling is reproducible. Unfortunately, with deeper models with more parameters, repeating cross validation can become computationally expensive, though this avenue of experimentation could be pursued in the future.

What remains of the results section are the performance metric graphs for the MAE, MSE, and RMSE. The appendices also contain the remaining CI tables and error propagation tables over time for each model.

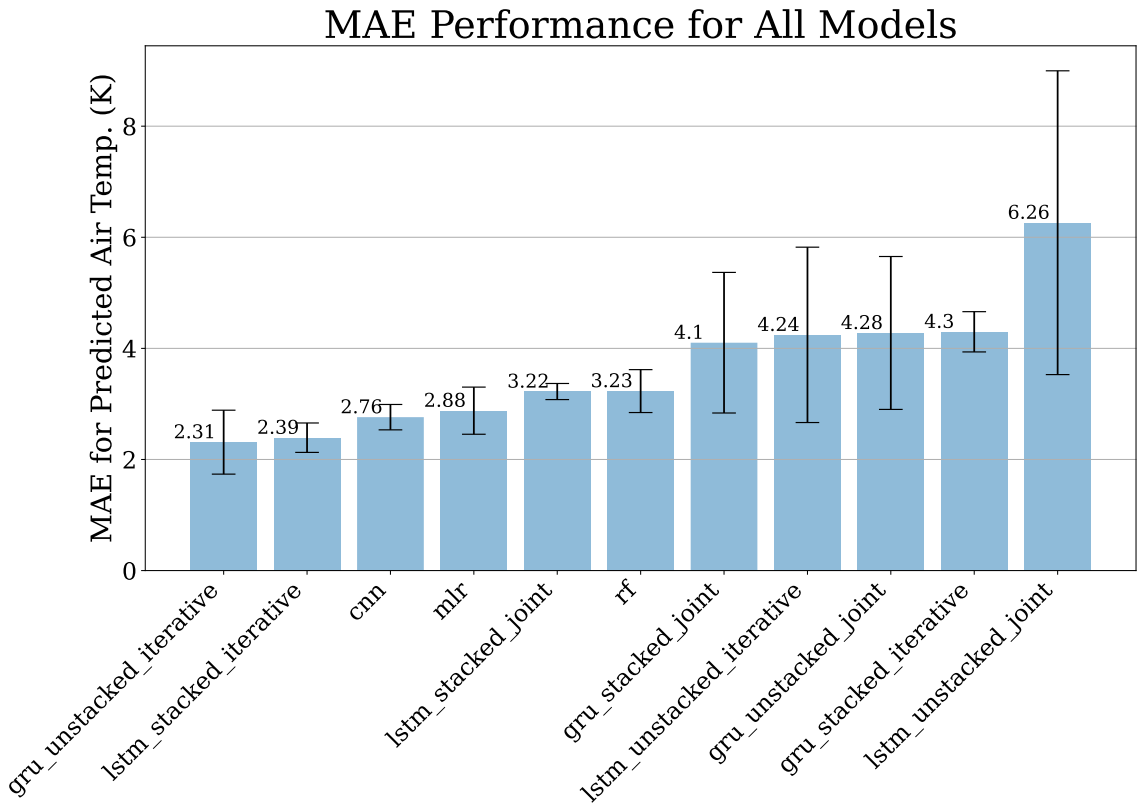


Figure 9. MAE for all models with 95% confidence interval.

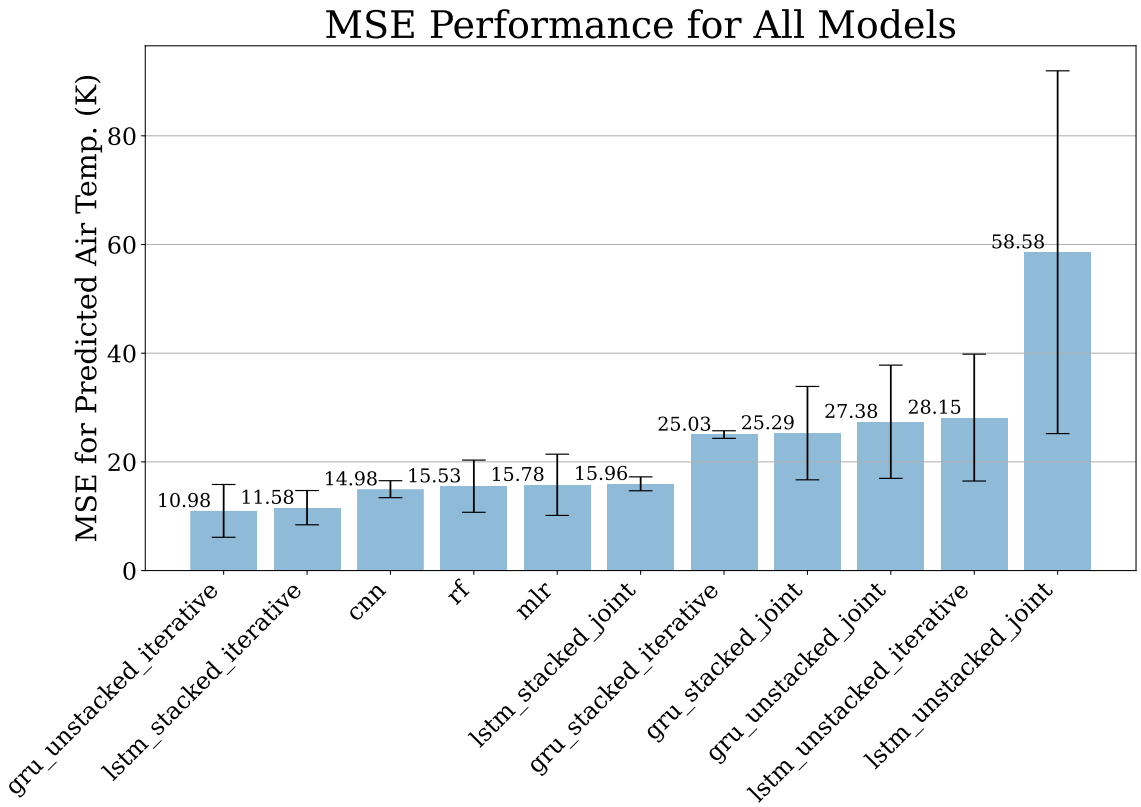


Figure 10. MSE for all models with 95% confidence interval.

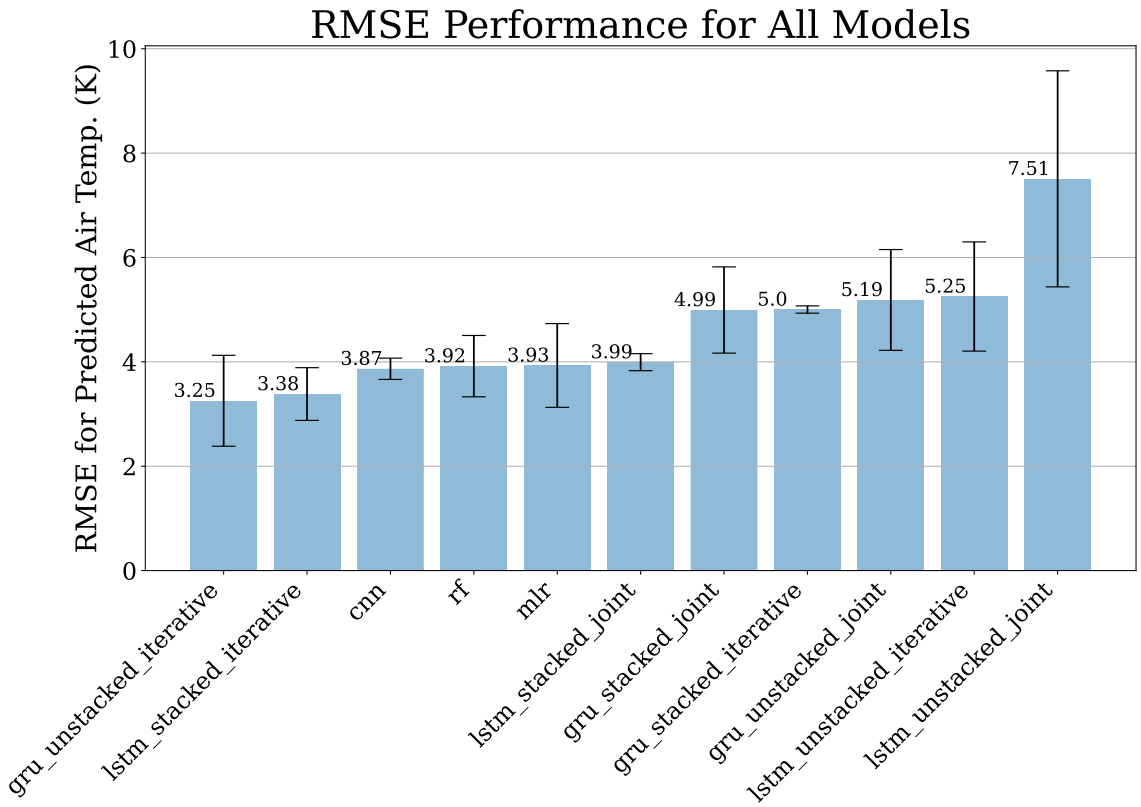


Figure 11. RMSE for all models with 95% confidence interval.

V. CONCLUSIONS AND FUTURE WORK

Eleven machine learning models were implemented for the prediction of a 7-sol forecast for mean ambient air temperature on Mars. Of those models, multiple linear regression, random forest, several variants of recurrent neural nets, and finally convolutional nets were explored. Bayesian optimization was used to hyperparameter tune models while walk-forward cross validation was utilized to maintain temporal dependencies between cross validation blocks. The best performing model based on MAE was the GRU-unstacked-iterative RNN; however, the top performing models had significant overlap in their confidence intervals. The work presented in this thesis is progress toward more thorough studies in computational atmospheric sciences, particularly those that would focus on space exploration. This work also makes the code base used to develop and test these models freely available to the public for research and recreational purposes (https://github.com/jfdev001/mars-ml-mtsu-honors-thesis).

The techniques learned for this thesis are critical for all future computational studies that the author will conduct. Future work could entail de-seasonalizing, de-trending, or implementing graph/attention-based models for time series forecasting.

REFERENCES

[1] M. Buchanan, “Colonizing Mars,” *Nature Physics*, vol. 13, no. 11, p. 1035, Nov. 2017, doi: 10.1038/nphys4311.

[2] S. P. Faulk, J. M. Lora, J. L. Mitchell, and P. C. D. Milly, “Titan’s Climate Patterns and Surface Methane Distribution Due to the Coupling of Land Hydrology and Atmosphere,” *Nature Astronomy*, vol. 4, no. 4, pp. 390–398, 2020, doi: 10.1038/s41550-019-0963-0.

[3] I. Levchenko, S. Xu, S. Mazouffre, M. Keidar, and K. Bazaka, “Space Exploration: Mars Colonization: Beyond Getting There,” *Global Challenges*, vol. 3, no. 1, p. 1970011, Jan. 2019, doi: 10.1002/gch2.201970011.

[4] D. James, “Mars Facts,” *NASA Quest*. NASA, Jun. 2013. [Online]. Available: https://web.archive.org/web/20130607140708/http://quest.nasa.gov/aero/planetary/mars.html

[5] R. A. Pielke, “Examples of Mesoscale Models,” in *International Geophysics*, Elsevier, 2013, pp. 427–500. doi: 10.1016/B978-0-12-385237-3.00013-X.

[6] P. Bauer, A. Thorpe, and G. Brunet, “The Quiet Revolution of Numerical Weather Prediction,” *Nature*, vol. 525, no. 7567, pp. 47–55, 2015, doi: 10.1038/nature14956.

[7] D. Michie, “Memo Functions and Machine Learning,” *Nature*, vol. 218, no. 5136, pp. 19–22, Apr. 1968, doi: 10.1038/218019a0.

[8] M. G. Schultz *et al.*, “Can Deep Learning Beat Numerical Weather Prediction?,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 379, no. 2194. Royal Society Publishing, Apr. 05, 2021. doi: 10.1098/rsta.2020.0097.

[9] M. W. Gardner and S. R. Dorling, “Artificial Neural Networks (The Multilayer Perceptron): A Review of Applications in the Atmospheric Sciences,” *Atmospheric Environment*, vol. 32, no. 14–15, pp. 2627–2636, Aug. 1998, doi: 10.1016/s1352-2310(97)00447-0.

[10] T. R. V. Anandharajan, G. A. Hariharan, K. K. Vignajeth, R. Jijendiran, and Kushmita, “Weather Monitoring Using Artificial Intelligence,” in *2016 2nd International Conference on Computational Intelligence and Networks (CINE)*, Jan. 2016, pp. 106–111. doi: 10.1109/CINE.2016.26.

[11] A. J. Hill, G. R. Herman, and R. S. Schumacher, “Forecasting Severe Weather with Random Forests,” *Monthly Weather Review*, vol. 148, no. 5, pp. 2135–2161, May 2020, doi: 10.1175/mwr-d-19-0344.1.

[12] N. Anusha, M. S. Chaithanya, and G. J. Reddy, “Weather Prediction Using Multi Linear Regression Algorithm,” *IOP Conference Series: Materials Science and Engineering*, vol. 590, p. 12034, Oct. 2019, doi: 10.1088/1757-899x/590/1/012034.

[13] I. Priyadarshini and V. Puri, “Mars Weather Data Analysis Using Machine Learning Techniques,” *Earth Science Informatics*, vol. 14, no. 4, pp. 1885–1898, Dec. 2021, doi: 10.1007/s12145-021-00643-0.

[14] Y. E. Cebeci, “A Recurrent Neural Network Model for Weather Forecasting,” Sep. 2019. doi: 10.1109/ubmk.2019.8907196.

[15] M. Hossain, B. Rekabdar, S. J. Louis, and S. Dascalu, “Forecasting the Weather of Nevada: A Deep Learning Approach,” Jul. 2015. doi: 10.1109/ijcnn.2015.7280812.

[16] S. Cramer, M. Kampouridis, A. A. Freitas, and A. K. Alexandridis, “An Extensive Evaluation of Seven Machine Learning Methods for Rainfall Prediction in Weather Derivatives,” *Expert Systems with Applications*, vol. 85, pp. 169–181, Nov. 2017, doi: 10.1016/j.eswa.2017.05.029.

[17] S. Dhamodaran, Ch. K. C. Varma, and C. D. Reddy, “Weather Prediction Model Using Random Forest Algorithm and GIS Data Model,” in *Innovative Data Communication Technologies and Application*, Springer International Publishing, 2020, pp. 306–311. doi: 10.1007/978-3-030-38040-3_35.

[18] N. Jones, “How Machine Learning Could Help to Improve Climate Forecasts,” *Nature*, vol. 548, no. 7668, p. 379, Aug. 2017, doi: 10.1038/548379a.

[19] National Aeronautical and Space Administration, “Mars Science Laboratory Team Papers,” 2020.

[20] J. Gomez-Elvira, “Mars Science Laboratory Rover Environmental Monitoring Station RDR Data V1.0,” *NASA Planetary Data System*. 2013.

[21] A. G. Salman, B. Kanigoro, and Y. Heryadi, “Weather Forecasting Using Deep Learning Techniques,” in *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, 2015, pp. 281–285. doi: 10.1109/ICACSIS.2015.7415154.

[22] M. A. I. Navid, “Multiple Linear Regressions for Predicting Rainfall for Bangladesh,” *Communications*, vol. 6, no. 1, p. 1, 2018, doi: 10.11648/j.com.20180601.11.

[23] S. Karthick, D. Malathi, and C. Arun, “Weather Prediction Analysis Using Random Forest Algorithm,” *International Journal of Pure and Applied Mathematics*, vol. 118, no. 20, pp. 255–262, 2018.

[24] J. S. Löfgren, R. Haas, and H.-G. Scherneck, “Sea Level Time Series and Ocean Tide Analysis from Multipath Signals at Five GPS Sites in Different Parts of the World,” *Journal of Geodynamics*, vol. 80, pp. 66–80, 2014, doi: https://doi.org/10.1016/j.jog.2014.02.012.

[25] Z. Pala and R. Atici, “Forecasting Sunspot Time Series Using Deep Learning Methods,” *Solar Physics*, vol. 294, no. 5, p. 50, 2019, doi: 10.1007/s11207-019-1434-6.

[26] B. LeBaron, W. B. Arthur, and R. Palmer, “Time Series Properties of an Artificial Stock Market,” *Journal of Economic Dynamics and Control*, vol. 23, no. 9, pp. 1487–1516, 1999, doi: https://doi.org/10.1016/S0165-1889(98)00081-5.

[27] Y. Xu and R. Goodacre, “On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning,” *Journal of Analysis and Testing*, vol. 2, no. 3, pp. 249–262, Jul. 2018, doi: 10.1007/s41664-018-0068-2.

[28] A. Geron, “The Machine Learning Landscape,” in *Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed., O’Reilly Media Inc., 2019, pp. 30–31.

[29] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996. doi: 10.1017/cbo9780511812651.

[30] Y. Lecun, L. Bottou, G. Orr, and K.-R. Müller, “Efficient BackProp,” Aug. 2000.

[31] S. G. K. Patro and K. K. Sahu, “Normalization: A Preprocessing Stage.” 2015.

[32] R. J. A. Little and D. B. Rubin, *Statistical Analysis with Missing Data.*, 2nd ed. John Wiley & Sons, Inc., 2002.

[33] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[34] A. Geron, “Ensemble Learning and Random Forests,” in *Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed., O’Reilly Media, Inc., 2019, pp. 192–198.

[35] P. Aznar, “What is the Difference Between Extra Trees and Random Forest?,” *Quantdare*, Jun. 2020. https://quantdare.com/what-is-the-difference-between-extra-trees-and-random-forest/

[36] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely Randomized Trees,” *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006, doi: 10.1007/s10994-006-6226-1.

[37] D. J. Stekhoven and P. Bühlmann, “MissForest—Non-Parametric Missing Value Imputation for Mixed-Type Data,” *Bioinformatics*, vol. 28, no. 1, pp. 112–118, Jan. 2012, doi: 10.1093/bioinformatics/btr597.

[38] H. I. Oberman, S. van Buuren, and G. Vink, “Missing the Point: Non-Convergence in Iterative Imputation Algorithms,” Oct. 2021. [Online]. Available: http://arxiv.org/abs/2110.11951

[39] C. R. Harris *et al.*, “Array Programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020, doi: 10.1038/s41586-020-2649-2.

[40] P. Virtanen *et al.*, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.

[41] J. D. Hunter, “Matplotlib: A 2D Graphics Environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007, doi: 10.1109/MCSE.2007.55.

[42] W. McKinney, “Data Structures for Statistical Computing in Python,” in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 56–61.

[43] D. M. Kline, “Methods for Multi-Step Time Series Forecasting Neural Networks,” in *Neural Networks in Business Forecasting*, IGI Global, 2011, pp. 226–250. doi: 10.4018/978-1-59140-176-6.ch012.

[44] T. Hastie, R. Tibshirani, and J. Friedman, “Linear Methods for Regression,” in *The Elements of Statistical Learning*, 2009, pp. 43–49. doi: 10.1007/978-0-387-84858-7_3.

[45] J. Chan, “STAT 3022: Variable Selection: Backward and Forward.” University of Sydney, 2020.

[46] J. Hintze, “User’s Guide III: Regression and Curve Fitting.” NCSS Statistical Software, 2007.

[47] G. Heinze, C. Wallisch, and D. Dunkler, “Variable Selection - A Review and Recommendations for the Practicing Statistician,” *Biometrical Journal*, vol. 60, no. 3, pp. 431–449, Jan. 2018, doi: 10.1002/bimj.201700067.

[48] M. A. Poole and P. N. O’Farrell, “The Assumptions of the Linear Regression Model,” *Transactions of the Institute of British Geographers*, no. 52, pp. 145–158, 1971, [Online]. Available: http://www.jstor.org/stable/621706

[49] M. Verbeek, “An Introduction to Linear Regression,” *A Guide to Modern Econometrics*. Wiley, pp. 15–19, 2017.

[50] M. Verbeek, “Heteroskedasticity and Autocorrelation,” *A Guide to Modern Econometrics*. Wiley, pp. 97–120, 2017.

[51] S. Seabold and J. Perktold, “statsmodels: Econometric and Statistical Modeling with Python,” 2010.

[52] T. K. Ho, “Random decision forests.” doi: 10.1109/icdar.1995.598994.

[53] T. Hastie, R. Tibshirani, and J. Friedman, “Random Forests,” in *The Elements of Statistical Learning*, Springer New York, 2008, pp. 587–604. doi: 10.1007/978-0-387-84858-7_15.

[54] K. Fawagreh, M. M. Gaber, and E. Elyan, “Random Forests: From Early Developments to Recent Advancements,” *Systems Science & Control Engineering*, vol. 2, no. 1, pp. 602–609, Oct. 2014, doi: 10.1080/21642583.2014.956265.

[55] E. W. Fox, J. M. ver Hoef, and A. R. Olsen, “Comparing Spatial Regression to Random Forests for Large Environmental Data Sets,” *PLOS ONE*, vol. 15, no. 3, p. e0229509, Mar. 2020, doi: 10.1371/journal.pone.0229509.

[56] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[57] I. Goodfellow, Y. Bengio, and A. Courville, “Deep Feedforward Networks,” *Deep Learning*. MIT Press, p. 165, 2016.

[58] I. Goodfellow, Y. Bengio, and A. Courville, “Numerical Computation,” *Deep Learning*. MIT Press, p. 80, 2016.

[59] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” Mar. 2016. [Online]. Available: http://arxiv.org/abs/1603.04467

[60] F. Chollet, “Keras.” GitHub, 2015. [Online]. Available: https://github.com/fchollet/keras

[61] A. Tealab, “Time Series Forecasting Using Artificial Neural Networks Methodologies: A Systematic Review,” *Future Computing and Informatics Journal*, vol. 3, no. 2, pp. 334–340, Dec. 2018, doi: 10.1016/j.fcij.2018.10.003.

[62] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[63] A. Geron, “Deep Computer Vision Using Convolutional Neural Networks,” in *Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed., O’Reilly Media, Inc., 2019, pp. 445–452.

[64] A. Geron, “Training Models,” in *Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed., O’Reilly Media, Inc., 2019, p. 134.

[65] R. S. Srinivasamurthy, “Understanding 1D Convolutional Neural Networks Using Multiclass Time-Varying Signalss,” 2018.

[66] A. van den Oord *et al.*, “WaveNet: A Generative Model for Raw Audio,” Sep. 2016. [Online]. Available: http://arxiv.org/abs/1609.03499

[67] Y. Li, X. Zhang, and D. Chen, “CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes,” Feb. 2018. [Online]. Available: http://arxiv.org/abs/1802.10062

[68] Y. Li, M. Liu, K. Drossos, and T. Virtanen, “Sound Event Detection Via Dilated Convolutional Recurrent Neural Networks,” in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 286–290. doi: 10.1109/ICASSP40776.2020.9054433.

[69] G. Lin, Q. Wu, L. Qiu, and X. Huang, “Image Super-Resolution Using a Dilated Convolutional Neural Network,” *Neurocomputing*, vol. 275, pp. 1219–1230, 2018, doi: https://doi.org/10.1016/j.neucom.2017.09.062.

[70] A. Hatamizadeh, H. Hosseini, Z. Liu, S. D. Schwartz, and D. Terzopoulos, “Deep Dilated Convolutional Nets for the Automatic Segmentation of Retinal Vessels,” May 2019. [Online]. Available: http://arxiv.org/abs/1905.12120

[71] S. Arlot and A. Celisse, “A Survey of Cross-Validation Procedures for Model Selection,” *Statistics Surveys*, vol. 4, no. none, Jan. 2010, doi: 10.1214/09-SS054.

[72] C. Bergmeir and J. M. Benitez, “On the Use of Cross-Validation for Time Series Predictor Evaluation,” *Information Sciences*, vol. 191, pp. 192–213, May 2012, doi: 10.1016/j.ins.2011.12.028.

[73] I. Goodfellow, Y. Bengio, and A. Courville, “Machine Learning Basics,” in *Deep Learning*, MIT Press, 2016, p. 96.

[74] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, “Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization,” *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019, doi: https://doi.org/10.11989/JEST.1674-862X.80904120.

[75] K. Swersky, J. Snoek, and R. P. Adams, “Multi-Task Bayesian Optimization,” *Advances in Neural Information Processing Systems*, Jan. 2013.

[76] T. O’Malley *et al.*, “Keras Tuner.” 2019.

[77] A. Botchkarev, “A New Typology Design of Performance Metrics to Measure Errors in Machine Learning Regression Algorithms,” *Interdisciplinary Journal of Information, Knowledge, and Management*, vol. 14, pp. 45–76, 2019, doi: 10.28945/4184.

[78] S. Bates, T. Hastie, and R. Tibshirani, “Cross-Validation: What Does It Estimate and How Well Does It Do It?,” Apr. 2021. [Online]. Available: http://arxiv.org/abs/2104.00673

[79] C. Willmott and K. Matsuura, “Advantages of the Mean Absolute Error (MAE) Over the Root Mean Square Error (RMSE) in Assessing Average Model Performance,” *Climate Research*, vol. 30, no. 1, pp. 79–82, 2005, doi: 10.3354/cr030079.

[80] C. Agostinelli, “Robust Stepwise Regression,” *Journal of Applied Statistics*, vol. 29, no. 6, pp. 825–840, 2002.

[81] A. Z. Ul-Saufie, A. S. Yahya, and N. A. Ramli, “Improving Multiple Linear Regression Model Using Principal Component Analysis for Predicting PM10 Concentration in Seberang Prai, Pulau Pinang,” *International Journal of Environmental Sciences*, vol. 2, no. 2, pp. 403–410, 2011.

[82] W. Greene, “Heteroskedasticity,” *Econometric Analysis*. Wiley, pp. 223–225, 2003.

[83] G. Bontempi, S. ben Taieb, and Y. A. le Borgne, “Machine Learning Strategies for Time Series Forecasting,” in *Lecture Notes in Business Information Processing*, 2013, vol. 138 LNBIP, pp. 62–77. doi: 10.1007/978-3-642-36318-4_3.

[84] F. Shahid, A. Zameer, and M. Muneeb, “Predictions for COVID-19 with Deep Learning Models of LSTM, GRU and Bi-LSTM,” *Chaos, Solitons & Fractals*, vol. 140, p. 110212, 2020.

[85] S. Yang, X. Yu, and Y. Zhou, “LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example,” in *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, 2020, pp. 98–101.

[86] R. Fu, Z. Zhang, and L. Li, “Using LSTM and GRU Neural Network Methods for Traffic Flow Prediction,” in *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, 2016, pp. 324–328.

[87] S. Khandelwal, B. Lecouteux, and L. Besacier, “Comparing GRU and LSTM for Automatic Speech Recognition,” Jan. 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01633254

APPENDICES

Table X. MAE for models with 95% confidence interval.

Models	CI Bounds		
	lower	mean	upper
gru_unstacked_iterative	1.74	2.31	2.89
lstm_stacked_iterative	2.13	2.39	2.66
cnn	2.53	2.76	2.99
mlr	2.45	2.88	3.3
lstm_stacked_joint	3.08	3.22	3.37
rf	2.84	3.23	3.62
gru_stacked_joint	2.84	4.1	5.37
lstm_unstacked_iterative	2.66	4.24	5.82
gru_unstacked_joint	2.9	4.28	5.65
gru_stacked_iterative	3.94	4.3	4.66
lstm_unstacked_joint	3.53	6.26	9

Table XI. MSE for models with 95% confidence interval.

Models	CI Bounds		
	lower	mean	lower
gru_unstacked_iterative	6.13	10.98	15.84
lstm_stacked_iterative	8.43	11.58	14.73
cnn	13.42	14.98	16.54
rf	10.73	15.53	20.33
mlr	10.15	15.78	21.42
lstm_stacked_joint	14.68	15.96	17.24
gru_stacked_iterative	24.33	25.03	25.73
gru_stacked_joint	16.7	25.29	33.88
gru_unstacked_joint	16.97	27.38	37.8
lstm_unstacked_iterative	16.48	28.15	39.83
lstm_unstacked_joint	25.2	58.58	91.97

Table XII. RMSE for models with 95% confidence interval

Models	CI Bounds		
	lower	mean	lower
gru_unstacked_iterative	2.38	3.25	4.13
lstm_stacked_iterative	2.88	3.38	3.89
cnn	3.66	3.87	4.07
rf	3.33	3.92	4.51
mlr	3.13	3.93	4.73
lstm_stacked_joint	3.83	3.99	4.16
gru_stacked_joint	4.17	4.99	5.82
gru_stacked_iterative	4.93	5	5.07
gru_unstacked_joint	4.22	5.19	6.15
lstm_unstacked_iterative	4.21	5.25	6.3
lstm_unstacked_joint	5.44	7.51	9.58

Table XIII. Number epochs before termination for early stopping of neural network training with 95% confidence interval.

Models	CI Bounds		
	lower	mean	lower
gru_unstacked_joint	3.81	6.2	8.59
lstm_unstacked_joint	2.64	6.2	9.76
gru_stacked_iterative	3.74	6.6	9.46
lstm_stacked_joint	3.07	7	10.93
gru_stacked_joint	4.25	7.6	10.95
lstm_stacked_iterative	6.24	8	9.76
cnn	5.37	8.2	11.03
gru_unstacked_iterative	2.05	10.8	19.55
lstm_unstacked_iterative	-0.78	11.8	24.38
gru_unstacked_joint	3.81	6.2	8.59
lstm_unstacked_joint	2.64	6.2	9.76

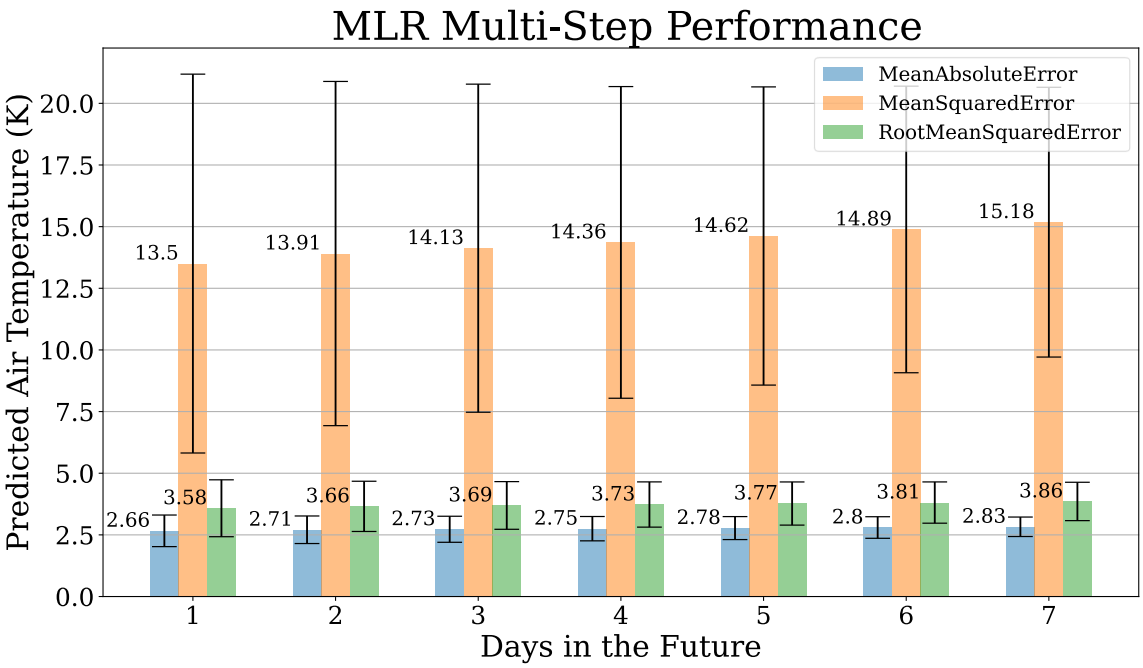


Figure 12. Performance of MLR at each timestep (sol) for all metrics with 95% confidence intervals.

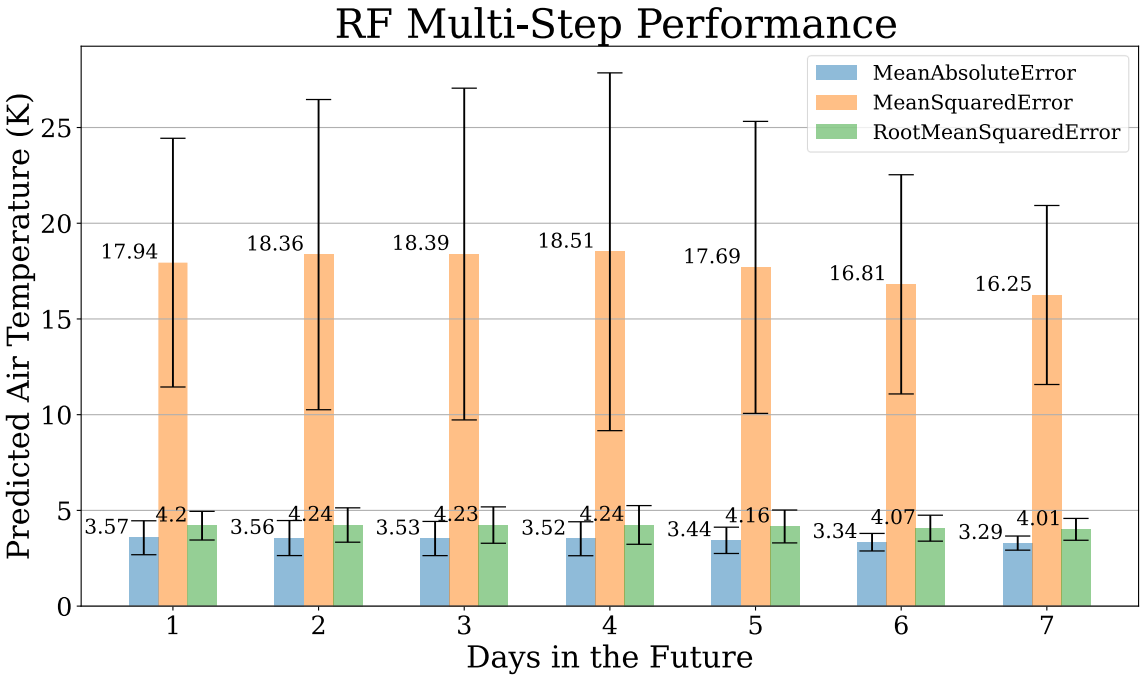


Figure 13. Performance of RF at each timestep (sol) for all metrics with 95% confidence intervals.

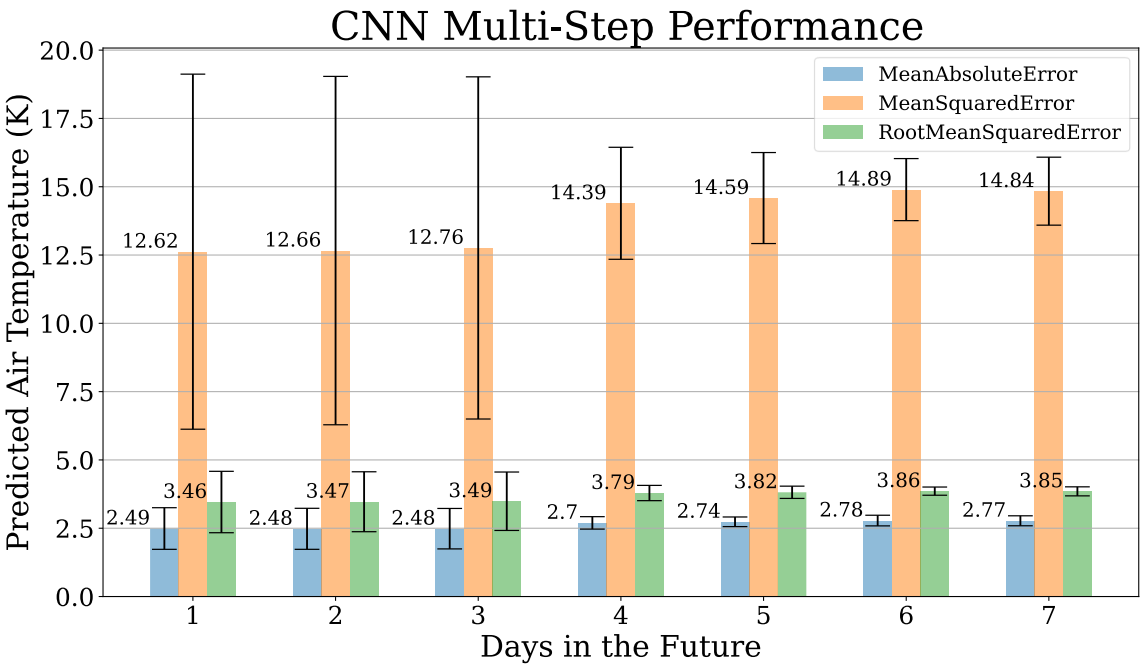


Figure 14. Performance of CNN at each timestep (sol) for all metrics with 95% confidence intervals.

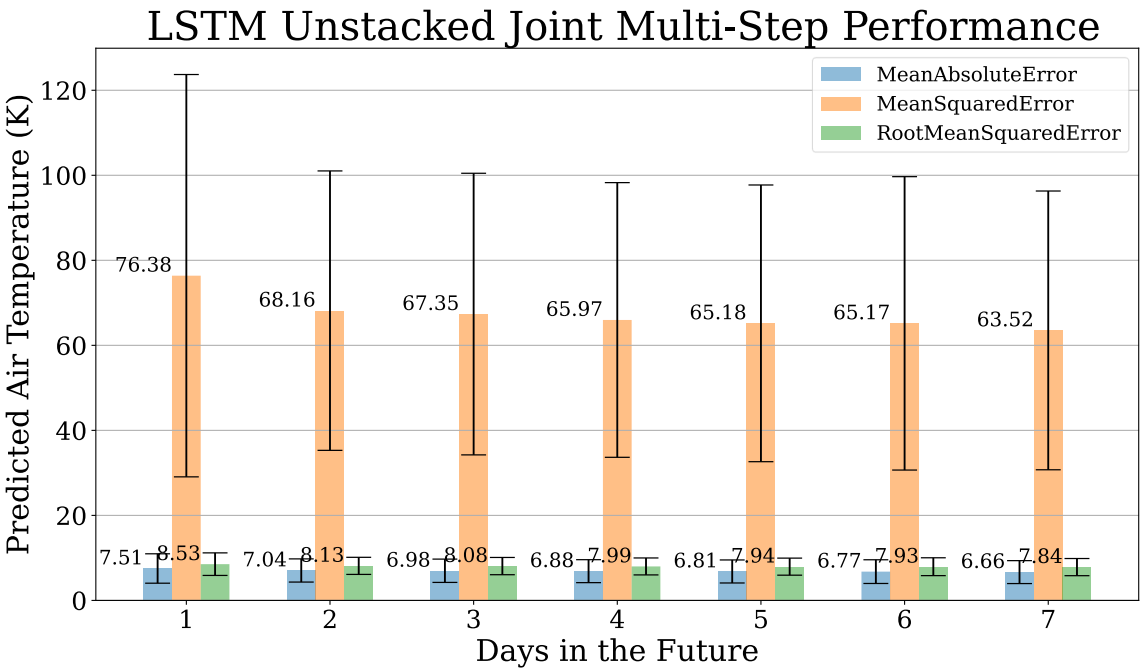


Figure 15. Performance of LSTM Unstacked Joint Model at each timestep (sol) for all metrics with 95% confidence intervals.

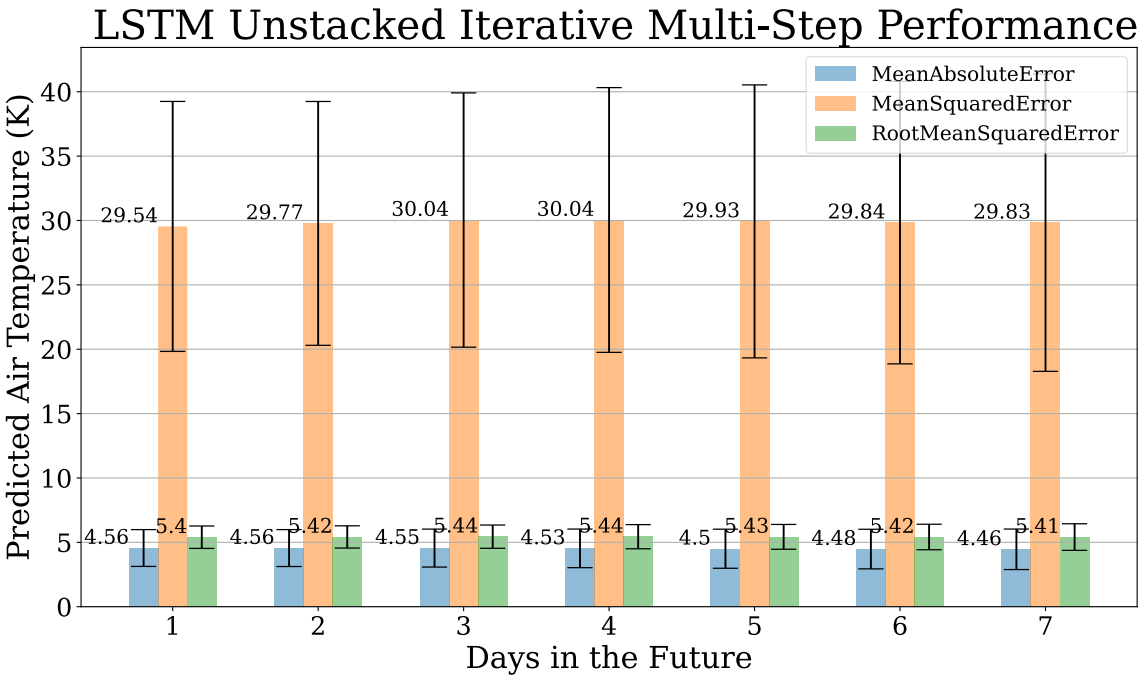


Figure 16. Performance of LSTM Unstacked Iterative Model at each timestep (sol) for all metrics with 95% confidence intervals.

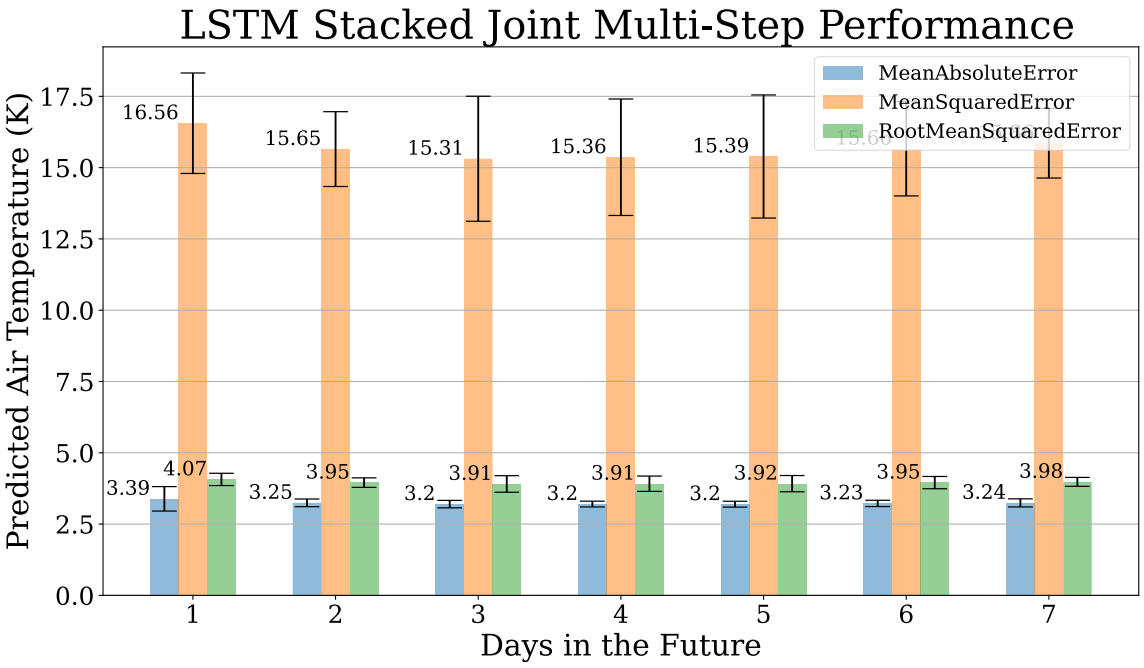


Figure 17. Performance of LSTM Stacked Joint Model at each timestep (sol) for all metrics with 95% confidence intervals.

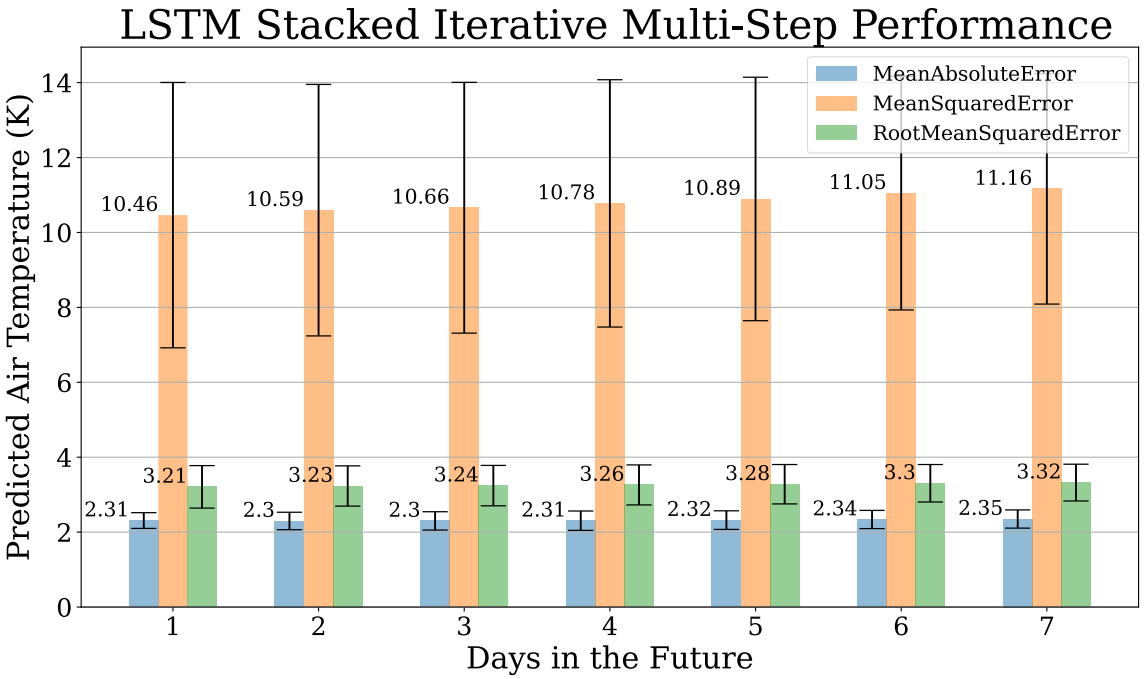


Figure 18. Performance of LSTM Stacked Iterative Model at each timestep (sol) for all metrics with 95% confidence intervals.

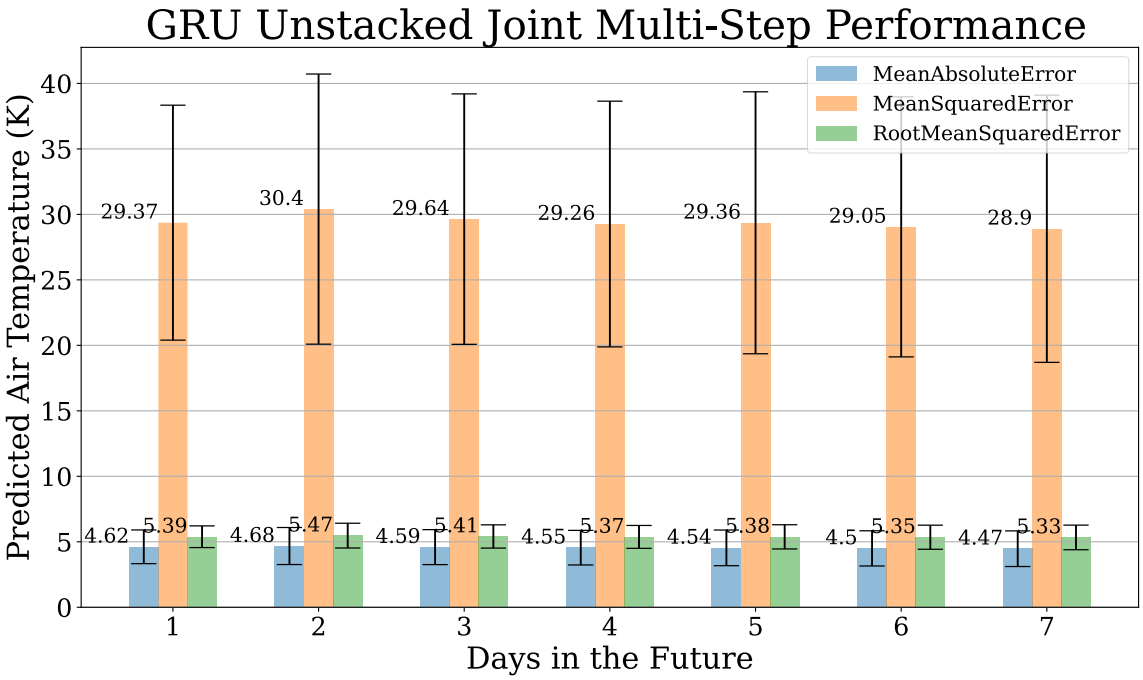


Figure 19. Performance of GRU Unstacked Joint Model at each timestep (sol) for all metrics with 95% confidence intervals.

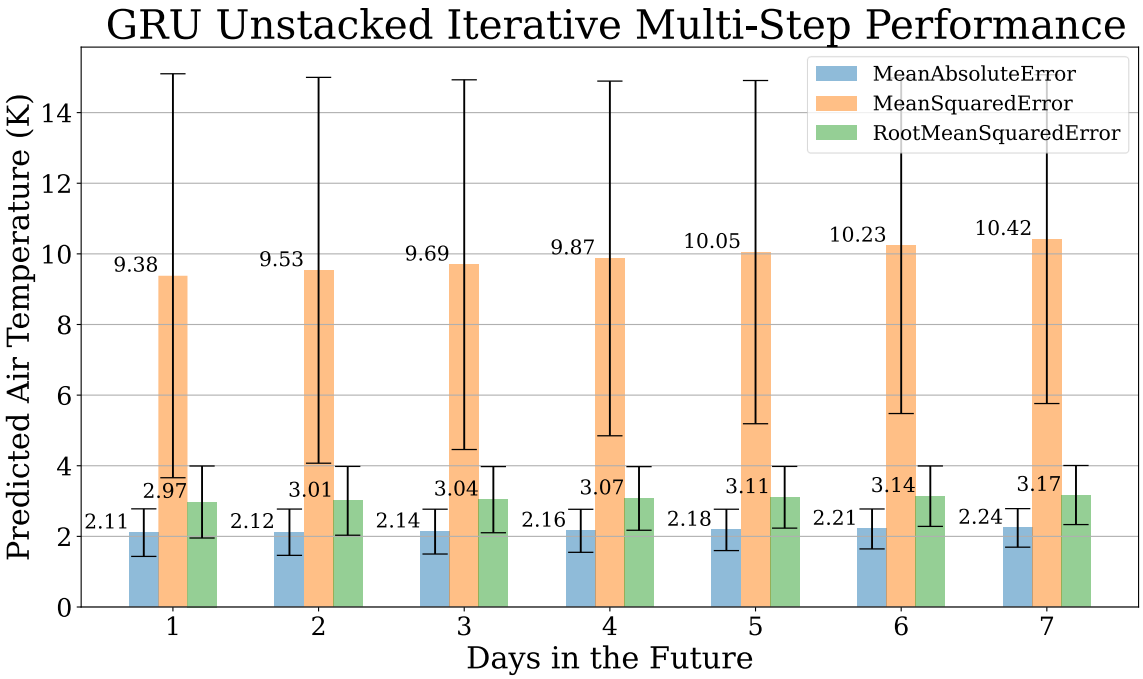


Figure 20. Performance of GRU Unstacked Iterative Model at each timestep (sol) for all metrics with 95% confidence intervals.

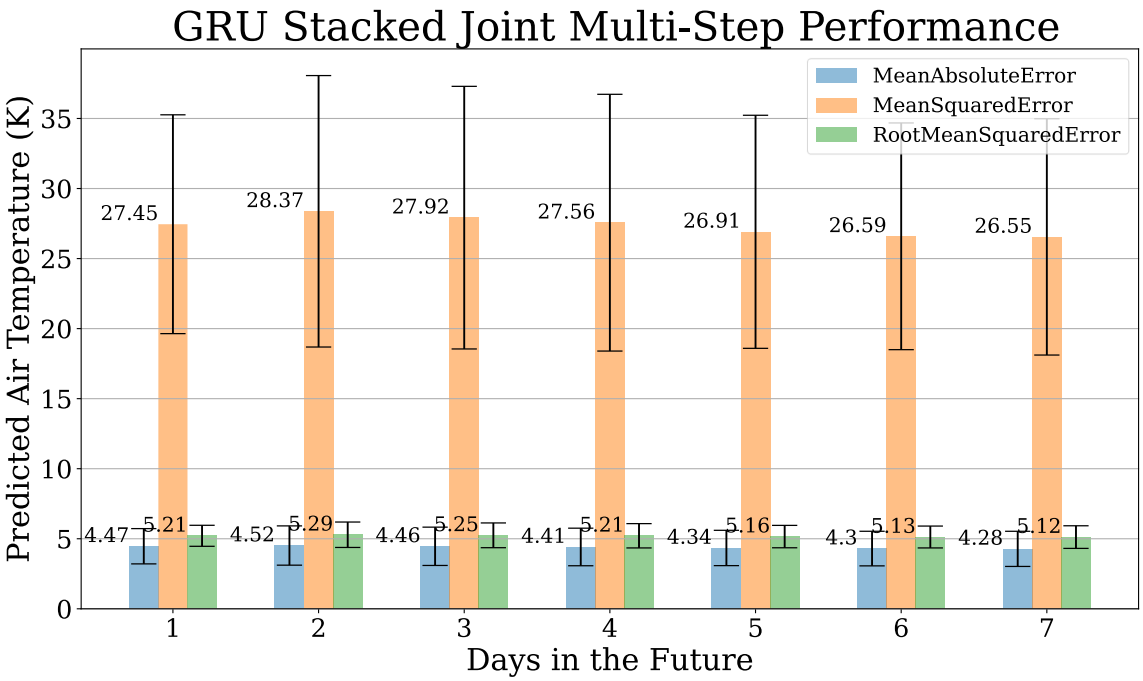


Figure 21. Performance of GRU Stacked Joint Model at each timestep (sol) for all metrics with 95% confidence intervals.

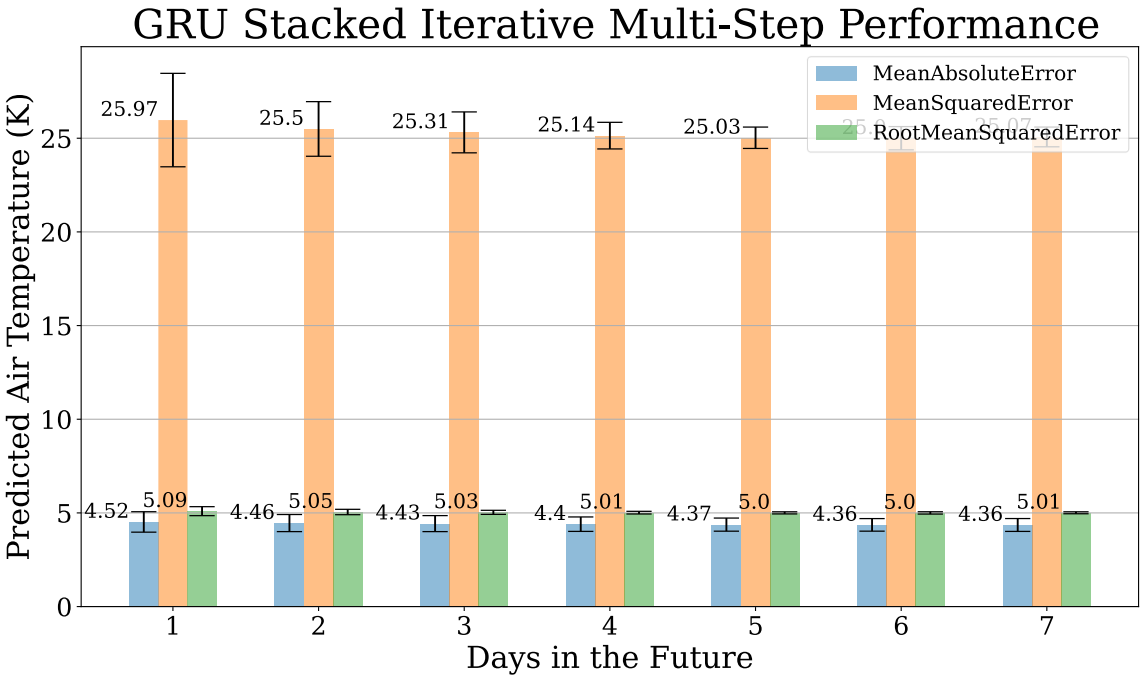


Figure 22. Performance of GRU Stacked Iterative Model at each timestep (sol) for all metrics with 95% confidence intervals.