ARTIFICIAL SATELLITES, Vol. 60, No. 3 – 2025
DOI: 10.2478/arsa-2025-0007

ACCELERATING ATMOSPHERE MODELING: NEURAL NETWORK ENHANCEMENTS FOR FASTER NRLMSISE CALCULATIONS

Volodymyr KASHYN, Vasyl CHOLIY

Main Astronomical Observatory, Kyiv, Ukraine

e-mails: volodymyr.kashyn@gmail.com, choliy.vasyl@gmail.com

ABSTRACT. NRLMSISE is an empirical model that allows us to predict temperatures and densities of the main atmospheric components. The model is widely used to evaluate atmospheric impacts on satellite orbits and laser beam refraction which come through the atmosphere, such as those used for Earth-satellite distance measurements. Model of the atmosphere is a valuable part of the Satellite Laser Ranging processing software like Kyiv Geodynamics (Juliette). Juliette is written in C++ and exploits the C++ clone of NRLMSISE written by the second author. The C++ version produces the same outputs as an official Fortran code.

Accurate modeling of atmospheric influences on satellite motion requires performing numerous calculations along satellite orbits or laser beam paths, which are computationally intensive. By decreasing calculation time of NRLMSISE, we would not only save the modeling time but also give a prospect for a wider application of the model due to lowering computational resource demands.

Our work demonstrates how the traditional NRLMSISE model can be effectively translated into a neural network. This conversion achieves significant performance gains on both CPU and GPU while maintaining acceptable accuracy when compared to the C++ implementation of NRLMSISE.

We demonstrate the process of moving NRLMSISE to a neural network, the resulting accuracy, ease of running the trained model on CUDA-enabled GPUs, and the obtained boost of performance on both CPU and GPU.

Keywords: atmospheric modeling, NRLMSISE, neural networks, machine learning, performance

1. INTRODUCTION

NRLMSISE-00, or further just NRLMSISE (NRL stands for US Naval Research Laboratory; MSIS for mass spectrometer and incoherent scatter radar, E for extended Earth surface), is an empirical atmospheric model developed by Mike Picone, Alan Hedin, Doug Drob, and colleagues. The model calculates the densities of different atmospheric gases and temperature at a specific location (Picone et al., 2002).

NRLMSISE takes the following parameters as inputs: year, day of the year (DOY), seconds past midnight, latitude, longitude, altitude, local apparent solar time, daily F10.7 solar flux for the previous day, 81-day average F10.7 solar flux (F10.7 81a), and day average magnetic index (Ap). The outputs include exospheric temperature, temperature at altitude, and the number densities of the main atmospheric gases: helium (He), oxygen (O), molecular oxygen (O_2), nitrogen (N), molecular nitrogen (N_2), argon (Ar), hydrogen (H), oxygen anomalous (O anom.), and total mass density.

NRLMSISE covers altitudes from ground to space (~1000 km) and has a wide range of applications, including evaluating atmospheric impacts on satellite orbits and assessing laser-beam propagation. For instance, NASA's Community Coordinated Modeling Center reports that NRLMSISE-00 is routinely employed not only to predict low-Earth-orbit satellite decay but also to calculate the column-air mass intersected by laser guide-star beams in adaptive-optics safety studies (NASA/GSFC CCMC, 2025).

An official Fortran implementation is available from the NASA/GSFC CCMC website (NASA/GSFC CCMC, 2025). NRLMSISE official implementation is ported on other programing languages and integrated into applications like Juliette in Satellite laser ranging station 1824 Golosiiv-Kyiv. Juliette is designed to predict satellite orbits, and C++ implementation of NRLMSISE, which provides the same outputs as the original Fortran one, is among other models used under the hood (Choliy, 2007). Although a single calculation is fast, processing hundreds of thousands or even millions of points can lead to significant computational delays, becoming a bottleneck. This exact issue was encountered in Juliette.

Deep learning techniques, particularly neural networks, are promising candidates to address this problem. Since technological progress keeps going on and graphical hardware becomes more powerful, this allows creating neural network surrogate models to mimic more complex models with a significant speed-up boost by skipping many intensive calculations. There are various such applications in physics, for instance, reactor-core diffusion solvers: modern research shows that convolutional neural network (CNN) surrogates can match traditional diffusion-solver accuracy while running faster on graphical processing unit (GPU) platforms (Zhang, Q. et al., 2021). Another example is from the field of mechanical stress and materials, where the first U-Net-based CNN surrogate was proposed for finite-element stress-field outputs, achieving ~500 times speed-ups with sub-percent errors (Khorrami et al., 2023).

It is worth noting that neural networks have an additional advantage over classic implementations that most of them can be run on GPU out-of-the-box, which natively support large matrix operations. This feature allows getting significant speed-ups by switching from central processing units (CPUs) to graphical processing units (GPUs) (Raina et al., 2009).

However, using neural networks we should always consider a trade-off between performance and errors. Given that NRLMSISE is already an empirical model, we set an acceptable threshold of mean relative percentage loss at less than 1% for each gas number density NRLMSISE output and less than 1% for total mass density compared to the official model outputs. By maintaining these thresholds, Juliette calculations remain stable.

There are existing neural network applications to NRLMSISE. However, their main ideas are to use neural networks and real observable data for improving the precision model rather than speeding up the computation time (Zhang, Y. et al., 2021). The most notable research in the field of current paper is thermoNET presented at the 29th International Symposium on Space Flight Dynamics, which is focused on thermosphere modeling (Izzo et al., 2024). It includes NRLMSISE replication through neural networks; however, its light physics-informed version (~1800 parameters) of NRLMSISE delivers the mean relative error 2.17% which does not satisfy our error requirements. The heavier (~257,000 parameters) variant reaches 0.98%.

While it is near to the required threshold, the latter one predicts only total density, and it has enormous number of parameters – this does not provide practical speed benefits. In practice, it remains computationally heavy for fast per-call evaluation.

In this paper, we describe the process of building a neural network based on NRLMSISE original algorithms, including density predictions of each gas and total mass density. The new model should satisfy three major requirements: the first one is meeting Juliette needs for error threshold (<1% means relative loss for each gas density and <1% for total mass density); the second one is that it should be faster on the same CPU environment while using only one thread comparing to the reference C++ implementation; and the third one is that the new model should be natively run on C++. Results of this paper show the mean relative loss and performance gains of the neural network NRLMSISE model compared to the original one in one-threaded CPU,
multithreaded CPU, and GPU environments. We emphasize the single-thread CPU case to isolate per-call computational cost under identical conditions: both NRLMSISE and the neural-network surrogate are executed on the same machine with single-threading enforced in this specific case.

2. DATA PREPARATION

2.1. Dataset generation

Since the goal is to replicate the algorithm of NRLMSISE based on the reference application, it is reasonable to use the existing NRLMSISE program, which is currently used in Juliette for dataset generation. It was decided to select all inputs of the original model except "year" because it is implicitly covered by solar and magnetic indices in the model.

In order to properly represent natural periodicity of day of year (DOY), seconds (SEC), local solar time (LST), longitude (Long), and latitude (Lat), their sine and cosine values are used. The following rescaling is used for DOY, SEC, and LST to represent them as angles in range $[0;2\pi]$:

$$\theta_{DOY} = 2\pi \frac{DOY-1}{365}, \quad \theta_{SEC} = 2\pi \frac{SEC}{86400}, \quad \theta_{LST} = 2\pi \frac{LST}{24} \tag{1}$$

All periodic inputs are selected for dataset generation, and their ranges are shown in Table 1.

Table 1. Periodic inputs selected for the model and their ranges for dataset generation

DOY Sin and Cos	Seconds Sin and Cos	Long Sin and Cos	LST Sin and Cos	Lat Sin	Lat Cos
[-1;1]	[-1;1]	[-1;1]	[-1; 1]	[-1;1]	[0;1]

The normal altitude range for NRLMSISE is from 0 (ground) to 1000 km; however, here altitude is used in the range from 500 to 1500 kilometers due to the specific NRLMSISE model usage in Juliette application, as it relies on the model's smooth parametric extrapolation for heights above 1000 km.

F10.7 adjusted, F10.7 81a (81-day average), and Ap historical values were extracted from the dataset provided by Geomagnetic Observatory Niemegk, GFZ covering 1947–2024 period (Matzka et al. 2021). The F10.7 series itself originates from the Canadian observatory network (Tapping, 2013).

Outliers for F10.7 and daily Ap were removed using a Hampel filter with 11-day sliding window, median ± 3 · 1.4826 MAD (median absolute deviation) (Hampel, 1974). The cleaned up F10.7 values show a minimum of 62 sfu (solar flux units) and a maximum of 458 sfu. Daily Ap has the range from 0 to 189 nT (nanotesla) after removing outliers. The range for F10.7 81a was taken as minimum and maximum values based on the archive. To make the neural network more tolerant against extremes, we applied a ∼10 % safety margin, yielding the final sampling as shown in Table 2.

Table 2. Non-periodic inputs selected for the model and their ranges for dataset generation before normalization

Alt, km	**F10.7, sfu**	**F10.7 81a, sfu**	**Ap, nT**
[500;1500]	[56;504]	[61;300]	[0;208]

Considering that the neural network should learn NRLMSISE program algorithms, but not observable physical phenomena, it was decided to generate each input value independently and randomly with uniform distribution, of course, preserving sine and cosine relations. This can lead to irrelevant input combinations from a physics standpoint, but it is acceptable since the focus is on learning algorithms of NRLMSISE. However, naive uniform distribution for longitude sine/cosine, latitude sine/cosine, and altitude resulted in poor sphere coverage, especially around the poles as shown on the plot in Figure 1.

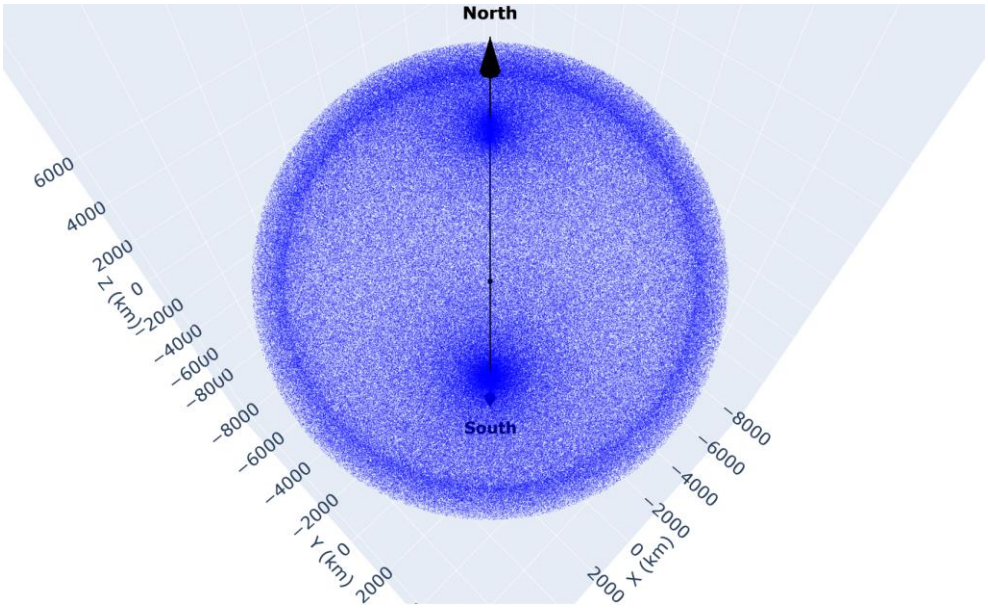

Figure 1. Distribution of input points based on naive uniform distribution of longitude, latitude, and altitude from 500 to 1500 km with the Earth radius taken as 6378 km. There are clearly visible cluster on poles, which should be removed.

The volume element in spherical coordinates after switching to geographic coordinates is as follows:

$$dV = r^2 cos(\varphi)\, dr\, d\varphi\, d\lambda \tag{2}$$

where here and further in the formulas:

λ – the longitude;
φ – the latitude;
r – the altitude.

There is visible clustering near the poles on the spherical shell because the surface area element on a sphere is proportional to the $cos(\varphi)$ value. It is possible to eliminate that factor by using the following approach to generate latitude and longitude:

$$\varphi = arcsin(v);\ \lambda = \pi u \tag{3}$$

where v and u are the uniform distributed values from -1 to 1.

In this case, arcsine eliminates $\cos(\varphi)$ factor in Formula (2). Altitude also requires proper distribution, and it can be handled as follows:

$$r = R\sqrt[3]{w} \tag{4}$$

where:
w – the uniform distributed value from 0 to 1;
R – the sphere radius.

Formula (4) is for sphere; however, in fact, we have a spherical shell, so it should be modified to support the altitude range:

$$r = \sqrt[3]{R_1^3 + w\,(R_2^3 - R_1^3)} \tag{5}$$

where:
w – the uniform distributed value from 0 to 1;
R_1, R_2 – the lesser and greater radii of the spherical shell.

Earth radius (R_{Earth}) is taken as 6378 km and altitude range is 500..1500 km by the definition above, and this allows letting $R_1 = R_{Earth} + 500$ km and $R_2 = R_{Earth} + 1500$ km. The spherical shell built using this approach is shown in Figure 2, and it illustrates much better distribution without visible clusters. So, the strategy described above is used for generating altitude, longitude, and latitude in our dataset with a good spherical shell distribution.

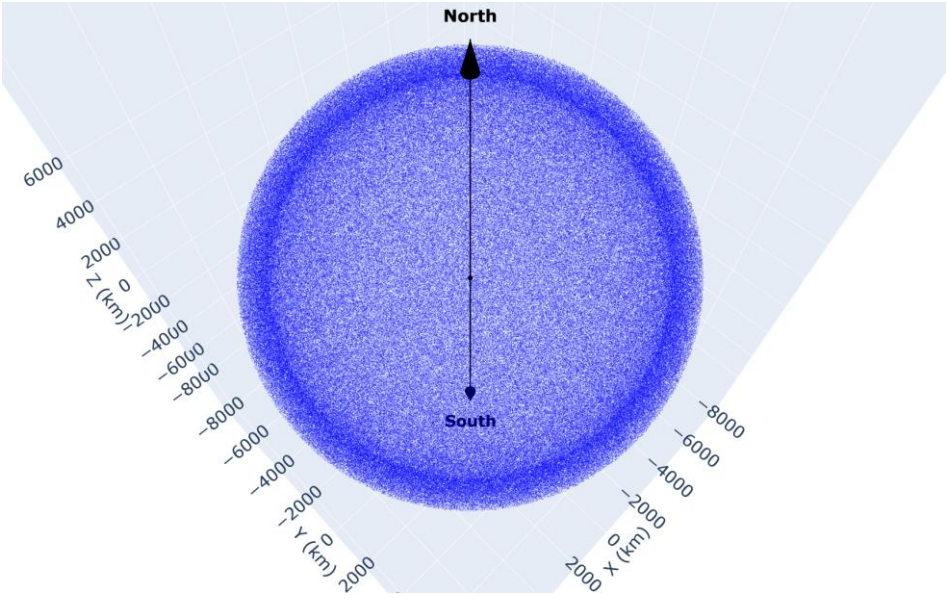

Figure 2. Distribution of input points based on proper distribution of longitude, latitude, and altitude from 500 to 1500 km with the Earth radius taken as 6378 km

The initial generated dataset for both training and evaluation contained two million records.

2.2. Input normalization

The total number of input features is 14 based on Tables 1 and 2. Ranges of cyclic parameters specified in Table 1 do not require further normalization, as they lie in a natural range [-1;1] for neural networks with Hardswish activation (Howard, 2019). However, non-periodic input parameters described in Table 2 require normalization. Since these parameters have strictly defined ranges and they are generated uniformly distributed, it was decided to use classic min-max normalization for inputs from Table 2 with the range [0;1]:

$$z_i^{(k)} = \frac{x_i^{(k)} - min(x^{(k)})}{max(x^{(k)}) - min(x^{(k)})} \quad (6)$$

where:

$z_i^{(k)}$ – the i[th] element of feature k, after normalization;
$x_i^{(k)}$ – the i[th] element of feature k, before normalization;
$min(x^{(k)}), max(x^{(k)})$ – the min and max values of feature k.

All input features now are in the [0; 1] or [–1; 1] range after normalization.

2.3. Output analysis and normalization

The total number of output features is 10. The list of outputs and their main characteristics are given in Table 3 for number densities and in Table 4 for temperatures.

Table 3. Quantile summary (cm^{-3}) of neutral number-density outputs used for network training. O(anom) is anomalous atomic oxygen.

	He	**O**	**N_2**	**O_2**	**Ar**	**H**	**N**	**O (anom)**
Min	2.99e+02	1.25e-08	1.19e-23	3.42e-29	1.11e-40	1.37e+03	6.84e-10	3.58e+01
5th	6.63e+04	5.70e+01	2.75e-05	1.24e-08	2.23e-14	8.56e+03	4.05e+00	1.00e+03
25th	2.16e+05	5.02e+03	1.05e-01	1.18e-04	3.29e-09	1.67e+04	3.58e+02	4.57e+03
50th	4.76e+05	1.01e+05	2.52e+01	5.58e-02	9.26e-06	2.96e+04	7.40e+03	1.38e+04
75th	1.08e+06	2.08e+06	5.06e+03	2.40e+01	1.86e-02	5.63e+04	1.16e+05	4.03e+04
95th	2.83e+06	3.77e+07	8.74e+05	7.98e+03	3.17e+01	1.56e+05	2.49e+06	1.69e+05
Max	1.04e+07	4.45e+08	9.67e+07	2.19e+06	1.40e+05	9.57e+06	2.65e+08	2.35e+06

Table 4. Quantile summary (K) of temperature outputs used for network training. Exospheric and altitude temperature quantile data are the same in taken precision.

	Min	**5th**	**25th**	**50th**	**75th**	**95th**	**Max**
Temp.	4.51e+02	8.41e+02	1.03e+03	1.19e+03	1.37e+03	1.64e+03	2.24e+03

After analyzing these outputs, it is clearly visible that the ranges are very wide and span several orders of magnitude. For example, Ar's 25th percentile is on the order of 10^{-9}, while its 75th percentile is on the order of 10^{-2}. Such a dynamic range impedes stable gradient propagation,

therefore, to eliminate such difference, natural logarithm normalization can be used, which is shown below:

$$y_i^{(k)\prime} = ln(x_i^{(k)}) \tag{7}$$

where:

$y_i^{(k)\prime}$ – the i[th] sample of output feature k, after *ln* normalization;

$x_i^{(k)}$ – the i[th] sample of output feature k, before *ln* normalization.

The ln transformation compresses the range, which can help stabilize variance and make patterns in the data more apparent to the neural network.

Although the logarithmic transformation substantially compresses the dynamic range, the resulting values still extend from approximately –36 to +14 and exhibit high variance. To obtain targets that are numerically well conditioned for back-propagation, z-score standardization is applied further:

$$y_i^{(k)} = \frac{y_i^{(k)\prime} - \mu^{(k)}}{\sigma^{(k)}} \tag{8}$$

where:

$y_i^{(k)}$ – the i[th] sample of output feature k, after *ln* and z-score normalization;

$y_i^{(k)\prime}$ – the i[th] sample of output feature k, after *ln* normalization;

$\mu^{(k)}$ – the mean of output feature k;

$\sigma^{(k)}$ – the standard deviation of output feature k.

Some examples of output distributions of the first million records before and after normalizations are shown in Figure 3 and Figure 4. Plots of normalized data demonstrate new compressed and unit scaled ranges.

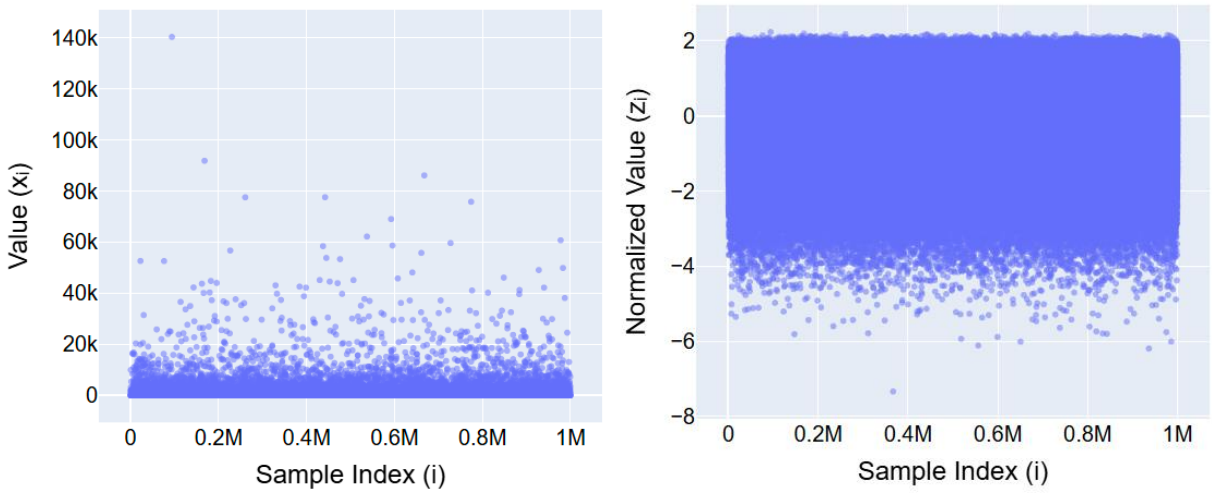

Figure 3. Distribution of argon (Ar) number densities before (left) and after (right) normalizations. Here, M stands for 1 million and k for 1 thousand.

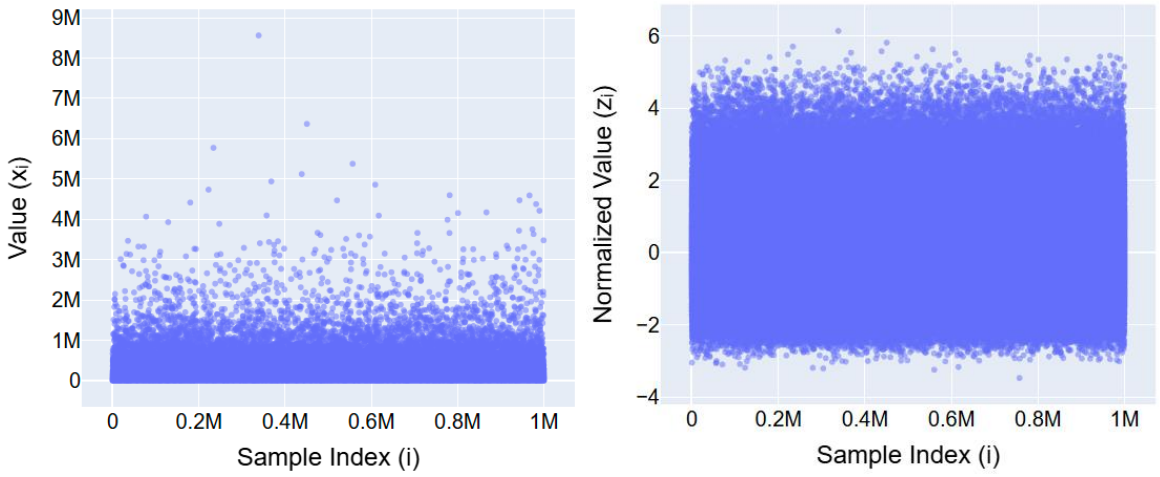

Figure 4. Distribution of hydrogen (H) before (left) and after (right) normalizations. Here, M stands for 1 million.

It is worth noting that total mass density is not included in the output feature list which will be processed by the neural network because this value is straightforwardly restored based on predicted number densities in the way it is done in NRLMSISE:

$$\rho_{Total} = \frac{1}{N_A} \cdot \sum_{i=1}^{8} n_i M_i \tag{9}$$

where:

N_A – Avogadro's constant (~ 6.022 x 10^{23} mol^{-1});

n_i & M_i – number density and molar mass of i^{th} gas species (Table 3).

3. NEURAL NETWORK

3.1. NRLMSISE and neural networks

Neural networks, especially feedforward architecture, have proven themselves to be powerful tools for modeling complex physical processes due to their ability to approximate any continuous function with arbitrary accuracy, if they contain sufficient neurons and layers (Hornik et al., 1989). This capability is formally established by the Universal Approximation Theorem (UAT) and its extensions. The original UAT states that even simple neural network architectures, such as multilayer perceptrons with at least one hidden layer, can approximate continuous functions on compact subsets of n-dimensional Euclidean space R^n to arbitrary precision, given enough hidden units (Cybenko, 1989).

The NRLMSISE model should employ segmented empirical fits and smooth interpolation techniques to ensure that its outputs vary continuously with respect to input parameters, such as altitude, solar flux, and geomagnetic indices, in order to illustrate real physical processes in the atmosphere. Although the use of piecewise functions may introduce localized regions of reduced smoothness (i.e., where the derivatives are not perfectly continuous), the overall behavior of the model is continuous and largely differentiable. This makes the NRLMSISE model a good candidate for approximation by neural networks.

By training a neural network on a sufficiently rich and representative dataset derived from the NRLMSISE algorithm, it can theoretically capture underlying functional mapping with high precision. The UAT provides a strong theoretical foundation for using neural networks as surrogate models of the NRLMSISE algorithm, potentially offering benefits such as reduced computational overhead and enhanced real-time performance.

3.2. Neural network representation

Neural networks are computational models built from interconnected neurons that transform input data through weighted sums and non-linear activation functions. A general feedforward multilayer perceptron (MLP) takes an input vector $\bar{x}$ and processes it through L layers, where each layer performs a linear transformation followed by a non-linear activation. For each layer l (with $l = 1, \ldots, L$ and $a^{(0)} = \bar{x}$), the computation is as follows:

$$a^{(l)} = \sigma^{(l)}(W^{(l)}a^{(l-1)} + b^{(l)}) \tag{10}$$

where:

- $W^{(l)}$ – the weight matrix;
- $b^{(l)}$ – the biases vector;
- $\sigma^{(l)}(\cdot)$ – continuous, non-polynomial activation function $R \rightarrow R$.

The mapping introduced by an L-layer neural network can be written as a nested composition of the layer-wise functions:

$$f_N(\bar{x}) = f^{(L)}(f^{(L-1)}(\ldots f^{(1)}(\bar{x})\ldots)) \tag{11}$$

where:

- f_N – the function implemented by a neural network whose capacity is N (e.g., the number of trainable parameters).

Additionally, UAT states that for any continuous function $f : K \rightarrow \mathbb{R}$ defined on a compact set $K \subset \mathbb{R}^d$ and for any tolerance $\varepsilon > 0$, there exists a neural network such that:

$$sup_{x \in K} \mid f_t(\bar{x}) - f_N(\bar{x})| < \varepsilon, \;\; N = N(\varepsilon) \tag{12}$$

where:

- f_t – the target function to approximate;
- f_N – the function implemented by a neural network whose capacity is N;
- ε – a user-chosen error tolerance.

These formulas capture the essence of multilayer perceptrons and underscore their capabilities, which is guaranteed by the UAT, to approximate any continuous function, such as those mapping the input parameters of the NRLMSISE model to its atmospheric outputs.

The process of training a feedforward neural network includes finding the optimal weights and biases that minimize a predefined loss function. This includes using methods like gradient descent and backpropagation so that network's output f_N closely matches the target values f_t across the training data as shown in formula (12) more formally.

3.3. Model architecture and training process

It is important to consider the model size during the architecture definition and training process, as our main goal is to achieve optimal performance of the neural network evaluation. Therefore, a balance must be kept between minimizing loss and ensuring efficient execution time of the model.

The global training strategy was as follows: first, the performance requirement was ignored, and the only resulting loss was considered. Then, once the required loss threshold was achieved for the reasonable number of parameters, the model was then optimized preserving the accuracy, mostly by increasing training time, decreasing the number of parameters, tuning hyperparameters, and using less computation intensive activation functions.

It is interesting that classic multilayer perceptron (MLP) neural networks performed well, and architectures like ResNet (modified for the regression problem) and custom structures, such as two-headed neural networks, do not provide any benefits, and their evaluation time is longer than that of a classic MLP for the achieved optimal loss. So, we focused on MLP neural networks.

The most optimal neural network architecture in terms of minimal relative loss and evaluation time is outlined in Table 5.

Initially, SiLU-Swish (Elfwing et al., 2018) was used to introduce non-linearity; however, it includes Sigmoid function, which is computationally heavy. In order to maintain accuracy gains and get evaluation speed enhancements, about 10–20%, Hardswish activation is used, the piece-hard analog of Swish (Howard, et al., 2019):

$$Hardswish(x) = \begin{cases} 0, & x \leq -3 \\ x, & x \geq +3 \\ x(x+3)/6, & otherwise \end{cases} \quad (13)$$

The first models were trained on the 2 million dataset (1 million training + 1 million validation), while later it was safe to drop the training dataset to 0.5 million records without noticeable drawbacks, using 0.5 million of other records as the validation dataset during training.

Since the current task is regression one, mean squared error (MSE) is used as a loss function for training. It is smoothly differentiable and scales trivially from one-to-many outputs, making optimization stable and implementation simple.

Table 5. Neural network structure. Biases enabled in each layer.

Layer index	Layer type	Input shape	Output shape	Parameters	Activation function
0	Dense	14	64	960	Hardswish
1	Dense	64	64	4160	Hardswish
2	Dense	64	64	4160	Hardswish
3	Dense	64	64	4160	Hardswish
4	Dense	64	10	650	No activation
Total parameter numbers				14 090	

Three Adam-family optimizers were tried: Adam (Kingma, et al., 2015), AdamW (Loshchilov, et al., 2019), and AdaMax (Kingma, et al., 2015) on the proposed network architecture. AdaMax converged fastest and achieved the lowest final validation loss. Core parameters to tune within AdaMax are learning rate (LR), momentum coefficient (β_1) and variance coefficient (β_2).

The following training parameters were used along with the structure defined in Table 5:

- learning rate strategy during training:
    - use a cyclical learning-rate schedule with inversed momentum coefficient, where the global learning-rate is allowed to oscillate between two boundary values for a specific number of cycles (Smith, 2015; Smith, 2018);

- for finding the first guess for the two boundary values, Leslie N. Smith search algorithm was used (Smith, 2017). After applying it and some further empirical tuning, the LR boundaries were taken: [0.0001, 0.013] for LR and [0.8, 0.9] for inversed momentum;
- other parameters: number of steps up per cycle is 1000; number of steps down per cycle is 10 000; β_2 = 0.999; learning rate upper range is halved each cycle;

- batch size: 2000;
- training dataset size: 0.5 million records;
- validation dataset size: 0.5 million records;
- stop condition: 60 000 epochs (full passes over 0.5 million samples).

Although SiLU and Hardswish perform similarly, the optimal training parameters are quite different. While a batch size of 512 is nearly optimal for SiLU (Swish), switching to the piecewise version of HardSwish required additional tuning of both the learning rate and the batch size, with a batch size of 2000 proving to be the best fit.

PyTorch, a deep learning library, was used to implement the neural network using the parameters defined above. The training process was done on CUDA (Compute Unified Device Architecture by Nvidia Corporation)-enabled GPU, and also, the training loop was moved to CUDA graphs to have the training speed up to 10 times due to eliminating CPU bottlenecks.

The neural network model was trained and compared against C++ reference NRLMSISE program in the same environment, which consists of AMD Ryzen 9 5900x CPU and Nvidia RTX 4070 GPU. All comparisons are done using x64 compiled programs in release configuration with maximum optimizations favoring speed and without debugger attached on Windows 11. The evaluation time of the neural network model listed in Results section includes reading the model from file, computing number densities and temperatures, restoring the normalized predicted output data to original ranges, and calculating the total mass density.

The training time of the neural network with the specified parameters and environment above is 50–60 minutes after 60 000 epochs. We used many epochs to attain the target loss without enlarging the model, thereby keeping fast evaluation of the model. The NRLMSISE neural network model should be trained only once and then used without any changes for any input data in the ranges specified in Tables 1 and 2. The edge cases which might require re-training are changing input parameter ranges, changes of NRLMSISE model internal empirical parameters, or significant changes of NRLMSISE code.

Also, it is verified that native Pytorch's C++ library LibTorch allows running the same models trained by Python interface in C++ code. Loss and execution time are the same.

4. RESULTS

In order to restore results obtained from neural networks, the inverse of the output normalization operations must be applied:

$$x_i^{(k)} = exp(y_i^{(k)} . \sigma^{(k)} + \mu^{(k)}) \tag{14}$$

where:

$y_i^{(k)}$ – the predicted feature k value for i[th] record;

$x_i^{(k)}$ – the restored predicted feature k value for i[th] record;

$\sigma^{(k)}$ and $\mu^{(k)}$ – the standard deviations and mean of output feature k achieved during dataset generation.

It is important to preserve $\sigma^{(k)}$ and $\mu^{(k)}$ for each feature k obtained from the training dataset.

4.1. Loss

Table 6 and Table 7 contain final relative loss for each restored NRLMSISE output separately. Figure 5 illustrates validation loss trends during training: mean squared error (MSE) for normalized outputs and mean relative loss (REL) among all restored outputs.

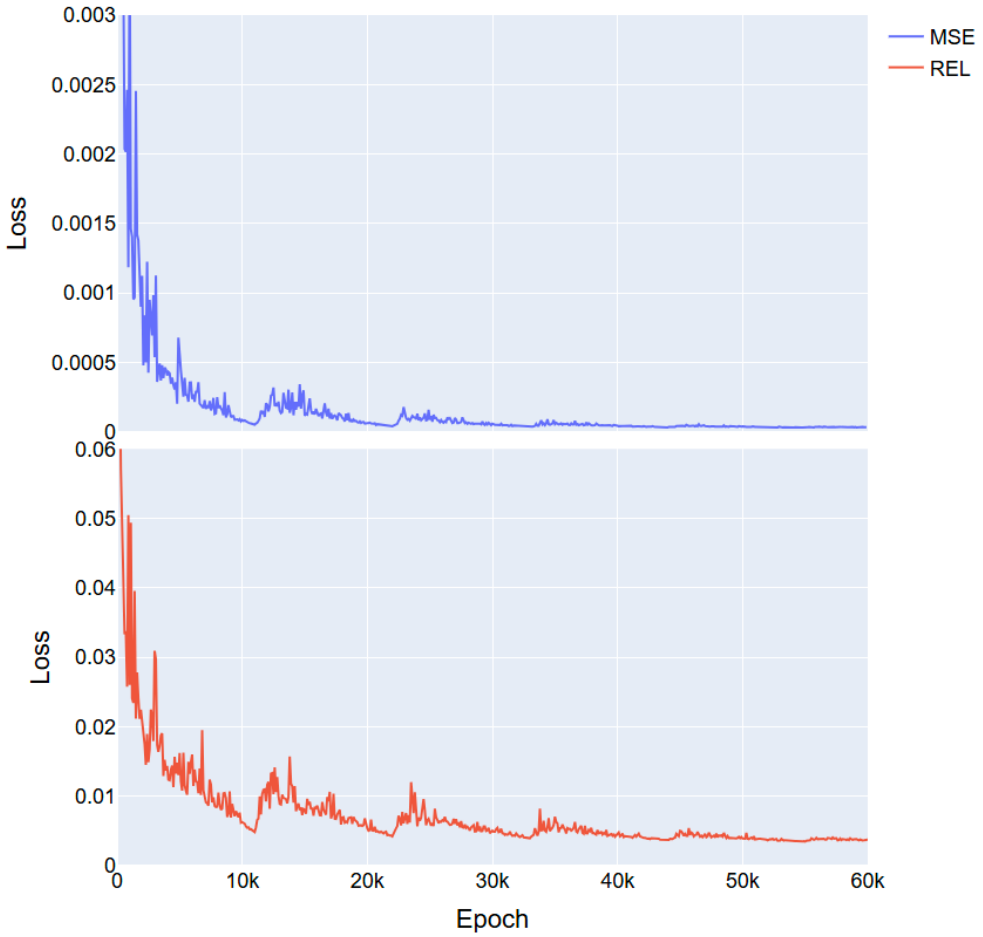

Figure 5. Validation loss on test dataset vs. epochs: MSE and REL (mean relative loss of all output features). Here, "k" means thousand.

Table 6. Final relative percentage loss for each restored number density output.

He	O	N_2	O_2	Ar	H	N	O (anom)
0.171%	0.414%	0.560%	0.648%	0.850%	0.137%	0.401%	0.168%

Table 7. Final relative percentage loss for restored temperatures and total mass density

Temp. exospheric	Temp. at height	Total mass density (ρ_{Total})
0.0469%	0.0475%	0.245%

"Saw-tooth" spikes seen on Figure 5 are normal for cyclical learning rate. They are visible near the top of each cycle where LR reaches its max value.

The losses illustrated in Tables 6 and 7 are calculated on the extended 1 million testing dataset, which did not participate in the training process. The relative percentage loss is calculated as follows:

$$rel_{ik} = \frac{|p_{ik} - t_{ik}|}{|t_{ik}|}, \qquad REL_k = 100\% \times \frac{1}{N}\sum_{i=1}^{N} rel_{ik}\,, \qquad REL = \frac{1}{K}\sum_{k=1}^{K} REL_k$$

where:

- rel_{ik} – the relative loss for i[th] sample of k[th] output feature;
- p_{ik} – the predicted output value for i[th] sample of k[th] output feature;
- t_{ik} – the truth/expected output value for i[th] sample of k[th] output feature;
- REL_k – the relative percentage loss for k[th] output feature (shown in Tables 6 and 7);
- REL – mean relative percentage loss of all output features (shown in Figure 5);
- N, K – numbers of samples and output features, respectively.

As mentioned before, total mass density is not a part of the learning process; during evaluation phase, it is restored based on formula (9). After getting numbers from NRLMSISE and placing them into (9), we got the exact formula we used to restore total mass density (g/cm^3):

$$\rho_{Total} = 1.66 \cdot 10^{-24}\left(4 \cdot n_{He} + 16 \cdot n_O + 28 \cdot n_{N_2} + 32 \cdot n_{O_2} + 40 \cdot n_{Ar} + n_H + 14 \cdot n_N\right) \quad (15)$$

where:

- n_k – the predicted number density (1/cm^3) by the neural network for gas k (Table 6).

Anomalous oxygen is not included in (15), as it is not used in GTS7 routine of NRLMSISE, which is the point of interest within Juliette. Total mass density loss is calculated compared to reference data in the generated testing dataset.

4.2. Execution time

Execution times and achieved boost for neural networks are outlined in Table 8.

In CPU single-threaded tests conducted on the same machine, the neural network model achieved performance between 3 and 4.5 times faster than the reference NRLMSISE model. When multithreading was enabled for the neural network NRLMSISE model, speed increased up to 9.7 times on the same CPU compared to the original model. In a GPU CUDA environment using an Nvidia RTX 4070, performance improvements of up to 250 times were observed. It is worth noting that switching between CPU and GPU evaluation is very straightforward, just changing one parameter in the program.

Table 8. Evaluation times of compared models for different data. M stands for million.

Number of NRLMSISE calculations	C++ NRLMSISE time, seconds	Neural network NRLMSISE time, seconds		
	CPU 1 thread	CPU 1 thread	CPU 12 threads	GPU CUDA
100 000	0.262 (1× reference)	0.087 (3.0× boost)	0.079 (3.3× boost)	0.142 (1.85× boost)
500 000	1.314 (1× reference)	0.310 (4.2× boost)	0.192 (6.8× boost)	0.143 (9.19× boost)
1M	2.566 (1× reference)	0.596 (4.3× boost)	0.324 (7.9× boost)	0.157 (16.3× boost)
1M × 10 times	25.598 (1× reference)	5.646 (4.5× boost)	2.709 (9.5× boost)	0.166 (154.2× boost)
1M × 100 times	258.448 (1× reference)	57.424 (4.5× boost)	26.68 (9.7× boost)	1.034 (250.0× boost)

4.3. Conclusions

In this work, we successfully replicated the NRLMSISE model using a feedforward neural network. The neural network model meets relative loss requirements: the worst mean relative loss is for argon, and it is 0.850%, which is smaller than the 1% threshold, and the best loss is 0.137% for hydrogen. The most important feature total mass density has 0.245% relative loss, which is better than 1% requirement.

The neural network model provides a notable boost of computation speed compared to the reference C++ NRLMSISE program: on the same CPU one-threaded environment, we got 4.5 times boost. While using multithreaded CPU, we got up to 9.7 times boost, and using GPU CUDA-enabled environments, we got ~250 times boost compared to the original program.

Having these results, we can affirm that replacing NRLMSISE's branch- and transcendental-heavy evaluations with a compact dense network, the surrogate reduces computation overhead, which enhances the model's applicability in real-time and computationally intensive environments, including Juliette. Moreover, the NRLMSISE neural network model can be easily integrated into existing C++ applications using the Libtorch library, deployed as a web-service or used in other environments where neural networks are supported.

Acknowledgments: The authors thank Vitaliy Zhaborovskyy and other colleagues at 1824 Golosiiv-Kyiv station for their valuable feedback and helpful discussions.

REFERENCES

Choliy V. (2007). New GNSS processor (Juliette) for geodynamic and atmospheric tasks. *Geophysical Research Abstracts*, v.9, EGU General Assembly 2007.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control Signal Systems 2*, 303–314 (1989). https://doi.org/10.1007/BF02551274

Elfwing, S., Uchibe, E. & Doya, K. (2018) ‘Sigmoid-weighted linear units for neural network function approximation in reinforcement learning’, *Neural Networks*, 107, pp. 3–11. doi: 10.1016/j.neunet.2017.12.012

Hampel, F.R., 1974. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69 (346), pp. 383–393. https://doi.org/10.1080/01621459.1974.10482962

Hornik, K., Stinchcombe, M. and White, H. (1989) ‘Multilayer feedforward networks are universal approximators’, *Neural Networks*, 2(5), pp. 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019, November 20). Searching for MobileNetV3 [Preprint]. arXiv. https://arxiv.org/abs/1905.02244

Izzo, D., Acciarini, G. & Biscani, F. (2024) ‘NeuralODEs for VLEO simulations: Introducing thermoNET for Thermosphere Modeling’. *Proceedings of the 29th International Symposium on Space Flight Dynamics (ISSFD 2024)*, Darmstadt, Germany, 22–26 April 2024. ESA/ESOC. Available at: https://issfd.org/ISSFD_2024/ISSFD2024_3-2.pdf (Accessed 16 July 2025)

Khorrami, M.S. et al. (2023) ‘An artificial neural network for surrogate modeling of stress fields in viscoplastic polycrystalline materials’, *npj Computational Materials*, 9, 37. https://doi.org/10.1038/s41524-023-00991-z

Kingma, D.P. and Ba, J. (2015) ‘Adam: A method for stochastic optimization’, *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, San Diego, CA, USA, 7–9 May. Available at: https://arxiv.org/abs/1412.6980

Loshchilov, I. and Hutter, F. (2019) ‘Decoupled weight decay regularization’, *Proceedings of the 7th International Conference on Learning Representations (ICLR 2019)*, New Orleans, LA, USA, 6–9 May. Available at: https://arxiv.org/abs/1711.05101

Matzka, J., Bronkalla, O., Tornow, K., Elger, K. and Stolle, C., 2021. Geomagnetic Kp index. V. 1.0. *GFZ Data Services*, https://doi.org/10.5880/Kp.0001

NASA/GSFC CCMC (2025) *NRLMSISE-00 empirical atmosphere model*, Fortran source code. Available at: https://ccmc.gsfc.nasa.gov/modelweb/models/nrlmsise00.php (Accessed: 5 July 2025).

Picone, J. M., A. E. Hedin, D. P. Drob, and A. C. Aikin, NRLMSISE-00 empirical model of the atmosphere: Statistical comparisons and scientific issues, *J. Geophys. Res.*, 107(A12), 1468, doi:10.1029/2002JA009430, 2002.

Raina, R., Madhavan, A. & Ng, A.Y. (2009) ‘Large-scale deep unsupervised learning using graphics processors’, *Proceedings of the 26th International Conference on Machine Learning*, pp. 873-880.

Smith, L.N. (2015) ‘Cyclical learning rates for training neural networks. arXiv preprint arXiv:1506.01186. Available at: https://arxiv.org/abs/1506.01186.

Smith, L.N. (2017) ‘Cyclical learning rates for training neural networks’, *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV 2017)*, Santa Rosa, CA, 24–31 March. Piscataway, NJ: IEEE, pp. 464–472. doi: 10.1109/WACV.2017.58

Smith, L.N. (2018) ‘A disciplined approach to neural-network hyper-parameters: Part 1 – learning rate, batch size, momentum and weight decay’. arXiv preprint arXiv:1803.09820. Available at: https://arxiv.org/abs/1803.09820

Tapping, K.F. (2013) 'The 10.7 cm solar radio flux (F10.7)', *Space Weather*, 11 (7), pp. 394–406. doi: 10.1002/swe.20064

Zhang, Q., Zhang, J., Liang, L., Li, Z. & Zhang, T. (2021) 'A deep-learning-based surrogate model for estimating the flux and power distribution solved by diffusion equation', *EPJ Web of Conferences*, 247, 03013. https://doi.org/10.1051/epjconf/202124703013

Zhang, Y., Yu, J., Chen, J., & Sang, J. (2021). An Empirical Atmospheric Density Calibration Model Based on Long Short-Term Memory Neural Network. *Atmosphere*, 12(7), 925. https://doi.org/10.3390/atmos12070925

Received: *2025-04-18*

Reviewed: *2025-07-03 (undisclosed name); 2025-07-03 (undisclosed name)*

Accepted: *2025-10-01*