Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Neural-Network Surrogate Workflow for Accelerating the Evaluation of Empirical Geophysical Models

Volodymyr Kashyn
volodymyr.kashyn@gmail.com

Main Astronomical Observatory of the National Academy of Sciences of Ukraine

Vasyl Choliy
Main Astronomical Observatory of the National Academy of Sciences of Ukraine

Research Article

Keywords: neural-network surrogates, empirical geophysical models, geophysical model acceleration, satellite orbit propagation, NRLMSISE and IRI models, IGRF and EGM models

Posted Date: August 14th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10433932/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License

Additional Declarations: No competing interests reported.

Abstract

This paper presents a neural-network surrogate workflow for accelerating the evaluation of empirical geophysical models used in Earth and space science applications. Four reference models are considered: NRLMSISE-00, IRI 2020, an IGRF-14-based magnetic-field model, and an EGM-96-based geopotential model. For each model, a surrogate was trained on data generated by the corresponding reference C + + implementation. The workflow covers input-domain sampling, spatial and temporal representation, input and output normalization, architecture selection, training, restoration of physical outputs, and evaluation of approximation accuracy and computational performance. Within the tested input domains, all surrogates achieved sub-percent mean approximation errors for the considered outputs, with the worst reported scalar/component mean absolute percentage error equal to 0.981%. They also substantially reduced execution time relative to the reference implementations. When both the reference C + + models and neural-network inference were restricted to a single CPU thread on the same AMD Ryzen 5900X, the best speedups for input batches of up to two million samples were 4.3× for NRLMSISE, 332× for IRI, 126× for the magnetic-field model, and 7.08× for the geopotential model. Relative to the same single-threaded C + + baselines, GPU inference on an NVIDIA RTX 4070 achieved speedups of 39×, 18188×, 921×, and 98.8×, respectively. These results show that neural-network surrogates can preserve sub-percent mean approximation accuracy while substantially reducing the cost of repeated geophysical-model evaluation. The workflow provides an efficient data-driven approach for large-scale Earth and space science applications, including satellite orbit propagation, observation modelling, and simulation.

1 Introduction

The Golosiiv station in Kyiv performs laser ranging observations of satellites and uses the Juliette software package for orbit prediction and processing (Choliy, 2021), which is not publicly available. Artificial satellites and their observations depend on several components of the near-Earth environment, which are relevant to force modelling and signal propagation. In orbit-related computations and observation modelling, empirical models are used to describe the neutral atmosphere, ionosphere, geopotential, and geomagnetic field. In Juliette, these components are represented by NRLMSISE-00 (NRL stands for US Naval Research Laboratory; MSIS for mass spectrometer and incoherent scatter radar; E for extended Earth surface) for the atmospheric component (Picone et al., 2002); IRI 2020 (The International Reference Ionosphere) for ionosphere (Bilitza et al., 2022); geopotential model based on EGM96 - Earth Gravitational Model 1996 (Lemoine et al., 1998) and Earth magnetic field model based on IGRF 14 - International Geomagnetic Reference Field coefficient set (Beggan et al., 2026). Juliette uses C + + implementations of these models, developed by the second author, that reproduce the outputs of the corresponding reference implementations (Choliy, 2021). The model generations considered in this study were selected because they correspond to the implementations currently integrated into the Juliette software and therefore define the practical computational workload addressed here. They should not be interpreted as the latest available generation of every model family. The objective is to evaluate the surrogate-development workflow against a fixed operational software baseline.

While the numerical integration and differentiation of the satellite trajectory are computationally efficient, the repeated evaluation of environmental models mentioned above introduces a significant performance bottleneck. Their relatively high execution cost dominates the overall execution time, especially in scenarios which require dense sampling or large-scale batch computations. This limitation motivated the exploration of optimization strategies, with a focus on neural network-based surrogate models. This is particularly important for applications such as Juliette, which involve repeated orbit propagation and the processing of large batches of observations, where the cumulative cost of environmental-model evaluation becomes significant. From an Earth science informatics perspective, the challenge is both to accelerate these repeated evaluations and to represent heterogeneous model-generated geophysical data in a form suitable for accurate and efficient data-driven approximation.

We consider neural networks as surrogate models to approximate the outputs of computationally expensive models while reducing evaluation time. In addition, neural networks are well suited for execution on GPUs (graphical processing units), while existing model implementations are difficult to port efficiently due to complex control flow, numerous conditional branches, and designs optimized for sequential CPU (central processing unit) execution rather than data-parallel computation. The resulting surrogates are differentiable almost everywhere, which may facilitate their future integration into gradient-based computational workflows. Neural networks general ability to approximate continuous functions is supported by the universal approximation theorem (Cybenko, 1989), although attainable accuracy and computational efficiency depend on the selected representation, architecture, training data, and optimization procedure.

In Earth system modelling, Lu and Ricciuto (2019) used singular-value decomposition and Bayesian optimization to build a neural-network surrogate for a regional ecosystem model with 42,660 outputs. Their results showed that representing the large output space with a small number of SVD coefficients allowed an accurate surrogate to be trained using substantially fewer expensive model simulations. Machine learning has also been used to develop data-driven models of the near-Earth environment. Acciarini et al. (2024) trained thermospheric-density models on precise-orbit-determination-derived satellite data and compared them with established empirical models. Pan et al. (2024) developed a thermospheric-density model using Swarm C observations and evaluated it against measurements from Swarm C, CHAMP, and GOCE, as well as against NRLMSIS 2.0. In ionospheric modelling, Dutta and Cohen (2023) presented an improved topside electron-density model trained on satellite measurements and compared its predictions with IRI and E-CHAIM, while Yang and Fang (2023) combined COSMIC radio-occultation observations with IRI-2016 outputs used as prior knowledge. These near-Earth studies are mainly trained on observationally derived data and aim to improve geophysical predictions, sometimes using established empirical models for comparison or as prior information. In contrast, the present study trains neural networks directly on outputs from selected reference-model implementations, with the goal of reproducing those outputs accurately while reducing evaluation time.

The NRLMSISE and IRI surrogate models were presented individually in our previous publications (Kashyn V., Choliy V., 2025; Kashyn V., Choliy V., 2026). The present study extends this work with new IGRF- and EGM-based surrogates and performs a cross-model analysis of all four cases. Its principal contribution is a workflow for developing and validating neural-network surrogates that accelerate the repeated evaluation of empirical geophysical models. The workflow covers model-derived dataset generation, input-domain sampling, spatial and temporal representation, distribution-aware normalization, model-specific

output handling, training, restoration of physical quantities, and accuracy and performance evaluation. The four cases are used to identify common workflow stages, distinguish model-specific design choices, and quantify the trade-off between approximation accuracy and computational acceleration.

2 Workflow for neural-network surrogate

This section describes the workflow used to convert empirical geophysical model implementations into accurate and computationally efficient neural-network surrogates. It covers input-domain definition and sampling, spatial and temporal data representation, input and output normalization, neural-network design, training, restoration of physical outputs, and accuracy and performance evaluation. These stages are common to the four case studies, whereas individual representations, architectures, loss functions, and training settings remain *model specific*. The reported choices should therefore be interpreted as tested design patterns for the models considered rather than as universally optimal configurations.

In general, the workflow described below can be summarized as follows: input-domain definition → sampling and dataset generation → spatial/temporal representation → normalization → architecture and output design → training and validation → defining output restoration → accuracy and runtime evaluation.

2.1 Reference models, inputs, and target outputs

This study considers four reference geophysical models: NRLMSISE-00 (hereafter NRLMSISE), IRI 2020 (hereafter IRI), an Earth magnetic field model based on IGRF-14 (hereafter IGRF), and a geopotential model based on EGM-96 truncated at degree and order 70 (hereafter EGM). For each reference model, a neural-network surrogate model was developed. Table 1 summarizes the input parameters used by the models and the corresponding outputs selected as approximation targets.

Table 1
Reference models, inputs, and target outputs; TEC stands for total electron content

Reference model	Input features	Output parameters
NRLMSISE-00	Year; day of year (DOY), second of day, latitude; longitude; altitude; local solar time; daily F10.7, 81-day average F10.7, day average magnetic index Ap.	Number densities of atmospheric gases: He, O, O_2, N, N_2, Ar, H, O anomalous; exospheric and local temperatures; total mass density
IRI 2020	Year, DOY, universal time, latitude, longitude, altitude, current and previous day F10.7, 81-day and 365-day average F10.7; daily Ap index; thirteen 3-hour Ap index values; TEC integration range.	Electron density; electron density at F2 peak; neutral, ion, and electron temperatures; relative composition of O+, N+, H+, He+, O_2+, NO+; TEC; TEC percentage above F2 peak
IGRF-14 based magnetic field model	Year; DOY; latitude, longitude, altitude.	Magnetic field vector components in local geodetic North-East-Down frame: B_N, B_E, B_D
EGM-96 based geopotential model	Position in Cartesian coordinates (x,y,z).	Scaled non-central acceleration correction components in Cartesian coordinates: Δa_x, Δa_y, Δa_z

2.2 Dataset generation and input sampling

The general approach adopted in this work is to approximate established physical models rather than to rely directly on observational data. Accordingly, datasets are generated using existing implementations of these models, further referred to as reference (or original) models. Dataset construction requires adjusting a reference model implementation to enable systematic sampling over its input domain, triggering model execution with generated inputs, and storing input-results mapping, which form a dataset. One of the important steps in this process is an input sampling strategy that ensures proper coverage of the model domain.

The input space of empirical geophysical models can typically be partitioned into three main parts: time parameters, geodetic coordinates, and indices describing solar and geomagnetic activity. Time inputs may include combinations of calendar year, day of year (DOY), and time of day (e.g., seconds after midnight), depending on the specific model. Geodetic coordinates are given by longitude, latitude, and altitude. The third group includes model-dependent indices (such as solar flux and geomagnetic indices) that describe external forcing conditions.

A uniform distribution over *time parameters* is used, as it ensures unbiased coverage of the temporal domain in the absence of prior knowledge about regions of increased model complexity. It is worth mentioning that if time parameters consist of several parts (e.g. DOY and seconds after midnight), a complete datetime should be generated using uniform distribution and then it should be converted into required parts: DOY, seconds after midnight, etc. This avoids potential sampling biases that may arise when individual components are generated independently. For IRI the datetime range is taken from 1 Jan 2015 to 31 Dec 2019 (five-year interval) while for geomagnetic model it is taken from 1 Jan 2010 to 31 Dec 2019 (10 years range). For NRLMSISE, *year* was skipped as discussed below, and the geopotential-based model does not take time as an input parameter.

A more interesting case is a set of *geodetic coordinates*. Naive uniform generation of latitude, longitude and altitude could be enough, but in fact we get a poor sphere coverage with higher concentration on poles. It is shown in Fig. 1.

Having volume element in spherical coordinates:

$$dV = (R_{Earth} + h)^2 \cos(\varphi)\, dh d\varphi\, d\lambda$$

1

where here and further: R_{Earth} - radius of Earth (under selected spherical-Earth approximation); h - altitude; φ - latitude; λ - longitude.

Higher concentration on poles happens due to $\cos(\varphi)$, which affects angular distribution. As $|\varphi|$ approaches 90°, $\cos(\varphi)$ decreases, so uniform sampling in latitude and longitude produces a higher density of points near the poles. This effect can be avoided by sampling latitude in a way that compensates for the $\cos(\varphi)$ factor:

$$\varphi = \arcsin(v);\ \lambda = \pi u,$$

2

where: v and u - uniformly distributed values from − 1 to 1.

After applying Eq. (2), the angular part of the volume element is sampled uniformly. However, the radial factor remains: with $r = R_{Earth} + h$, the volume element is proportional to $r^2 dr$. Therefore, uniform sampling of altitude (h) would not produce a uniform distribution in the volume of the spherical shell. To obtain uniform volumetric sampling, the cumulative distribution of the radius must be proportional to r^3. For a sphere of radius R:

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

3

,

or, as in our case, for spherical shell bounded by $R_{min} = R_{Earth} + h_{min}$ and $R_{max} = R_{Earth} + h_{max}$:

$$r = \sqrt[3]{R_{min}^3 + w\left(R_{max}^3 - R_{min}^3\right)},$$

4

where: w - a uniformly distributed value from 0 to 1.

The altitude ranges used during dataset generation are $h_{min} = 500\ \text{km}$ and $h_{max} = 1500\ \text{km}$ for NRLMSISE; and $h_{min} = 80\ \text{km}$ and $h_{max} = 1500\ \text{km}$ for other models.

Fixed distribution of points on spherical shell is shown in Fig. 2.

The remaining set of parameters are *solar and geomagnetic activity indices*. There are two principal options here: either generate uniformly and independently or use historical data on generated time.

In the first option, the indices are sampled independently, typically from uniform distributions over predefined ranges. The bounds of these ranges should be estimated separately for each index. This approach treats the neural network primarily as an approximation of the reference model algorithm over a selected input domain, rather than over the empirical distribution of naturally occurring conditions. The advantage of this option is that we do not rely on historical data. A potential drawback is that we may spend resources on physically implausible combinations. However, this can be beneficial in some cases. For instance, NRLMSISE needs “year” input just to input a full datetime for selecting indices, and if “year” is skipped for preparing a deep learning model and NRLMSISE neural network is trained for any indices’ values (in their reasonable ranges) then the model may not need retraining for new time periods – basically, the model can be reused across future time periods without retraining, as long as the relevant indices remain within the sampled training ranges and no additional time-dependent inputs are required. As an example, this approach was used while creating NRLMSISE surrogate (Kashyn V., Choliy V., 2025).

The second option relies on the time parameters generated first, then a historical record is loaded from external dataset on that time. For instance, F10.7 and Ap indices can be searched at dataset provided by Geomagnetic Observatory Niemegk, GFZ covering from 1947 (Matzka et al. 2021). Here, a model learns only real dependencies, but it may require periodic retraining or dataset extension when new time periods introduce previously unseen index values or distributions. This approach was used for IRI neural network (Kashyn V., Choliy V., 2026).

The sampled spatial domain covered the full longitude and latitude ranges. Altitudes ranged from 500 to 1500 km for NRLMSISE and from 80 to 1500 km for IRI, IGRF, and EGM. IRI dates were sampled from 2015–2019 and IGRF dates from 2010–2019; NRLMSISE did not use year as an input, while EGM was time independent. The solar and geomagnetic indices were obtained from the corresponding historical records for IRI. The IRI TEC integration range was fixed at 80–1500 km.

2.3 Inputs representation for datetime and geodetic coordinates

In the most generic case models contain all three *datetime* parts, which have the following ranges: [1; 86400] for seconds in day, [1; 366] for days in year and years range for required time range. We do not recommend combining all time inputs in one “time” feature and remain separate to make it easier for neural networks to learn day-period dependency. However, the ranges above are not suitable as inputs for neural networks, because they have hard discontinuities: from 86400 to 1 and from 366 (or 365) to 1. To resolve this the ranges above can be rescaled to angles in ranges:

$$\theta_{DOY} = 2\pi \frac{DOY - 1}{Days\ in\ year}, \quad \theta_{Seconds} = 2\pi \frac{Seconds - 1}{86400},$$

5

where: θ_{DOY} - an angle representing day of year (DOY);

$\theta_{Seconds}$ - an angle representing seconds of the day.

$Days\ in\ year$ - either 365 in normal or 366 in leap years.

Then, neural networks can accept $\sin(\theta_{DOY})$; $\cos(\theta_{DOY})$; $\sin(\theta_{Seconds})$; $\cos(\theta_{Seconds})$ as inputs instead of raw values. The same approach can be used for "years" and other features in case they represent some period, e.g. a 12-year solar activity cycle.

For *geodetic coordinates*, raw longitude has a wraparound discontinuity at the -180° to 180° boundary. Sine and cosine encoding for latitude and longitude (denoted further as φ_1, φ_2 and λ_1, λ_2) were used for NRLMSISE as an input for neural networks, which also naturally represent longitude periodicity:

$$\varphi_1 = \sin(\varphi), \ \varphi_2 = \cos(\varphi), \ \lambda_1 = \sin(\lambda), \ \lambda_2 = \cos(\lambda),$$

6

Sine-cosine encoding (6) removes the wraparound problem for longitude. For latitude, which is not periodic, it mainly gives a bounded nonlinear input rather than fixing a real discontinuity. Still, the latitude-longitude representation has drawbacks, because distances are not represented uniformly over the globe: longitude changes near the equator ($\cos(\varphi) = 1$) represent larger physical distances; and smaller distances near poles ($\cos(\varphi) = 0$). So, in this case, instead of raw geodetic coordinates, a model consumes 5 values: two pairs of sine-cosine encoding of latitude and longitude plus altitude.

For the IRI, IGRF, and EGM surrogates, the coordinates representation is changed towards using altitude and Cartesian unit vector (r_{dir}):

$$r_{dir} = (\cos(\varphi)\cos(\lambda), \ \cos(\varphi)\sin(\lambda), \ \sin(\varphi)), \ ||r_{dir}||_2 = 1,$$

7

This option was tested during the preparation of the IRI model and, at that stage, appeared to be a more convenient representation. The Cartesian direction vector avoids some coordinate-specific issues of latitude-longitude coordinates, such as longitude numerical discontinuity without additional sin-cos encoding and physical distance ambiguity near the poles. Altitude is kept as a separate input because the modeled quantities strongly depend on height. Within this approach, instead of raw geodetic coordinates, 4 values are passed into a model: three components of Cartesian direction unit vector and altitude.

However, this choice should not be interpreted as proof that the Cartesian representation is always better. After this change, the model architecture, preprocessing, and training settings continued to evolve. Therefore, a direct face-to-face comparison with the earlier geodetic representation would no longer isolate only the effect of the coordinate representation.

2.4 Input normalization

Once input features for training and their representation are selected, the next preprocessing step is numerical normalization of inputs. This step reduces differences in feature scales and helps keep the input values in ranges suitable for gradient-based optimization. Across the four surrogate models, three main normalization techniques were used: min-max scaling from 0 to 1, z-score standardization, logarithmic (log) transformation. The corresponding transformations are given by:

$$x_i'^{(k)} = \frac{x_i^{(k)} - min^{(k)}}{max^{(k)} - min^{(k)}}, \ z_i^{(k)} = \frac{x_i^{(k)} - \mu^{(k)}}{\sigma^{(k)}}, \ l_i^{(k)} = ln\left(x_i^{(k)} + 1\right),$$

8

where here and further:

$x_i'^{(k)}$, $z_i^{(k)} l_i^{(k)}$ – i-th sample of input k after min-max scaling, z-score standardization and logarithmic transformation (assuming only positive inputs);

$x_i^{(k)}$ – i-th sample of input k being normalized;

$min^{(k)}$, $max^{(k)}$ – min and max values of input k in training dataset;

$\mu^{(k)}$, $\sigma^{(k)}$ – mean and standard deviation of input k calculated on training dataset.

Min-max scaling can be used for sampled values with known fixed ranges. For instance, it is used in NRLMSISE for altitude and solar indices (generated uniformly for this model), because all these values have known min and max values, and the model is not intended to be extrapolated outside this range.

Z-score standardization is a common choice for neural networks inputs normalization, especially for features whose scale is determined empirically from the training data (unknown min-max ranges) or standard deviation and mean of input are far from 1 and 0 respectively. Good examples of such features are real historical solar and magnetic indices taken on generated datetime. Z-score can be also applied to altitude and year inputs, although they are fixed ranges. However, it is not expected to get valuable benefits compared to min-max, since year is uniformly sampled and altitude follows the spherical-shell sampling distribution. After min-max scaling, both inputs are already compact, approximately centered, and have order-one variance.

In some cases, there still can be wide ranges after applying z-score standardization which also can be skewed to positive or negative side. For instance, F10.7 index was within [-1;8] range for IRI model, while our activation function Hardswish (Howard, et al., 2019) has nonlinear transition region around [-3; 3]. So, we applied logarithmic scaling before z-score standardization.

Sin-Cos encoded features and Cartesian unit vector do not require additional normalization as their means are near to zero (~ +/-0.0002), standard deviations are ~ 0.7 and ~ 0.5 respectively, and their ranges are [-1;1] by definition. The summary for inputs is shown in Table 2.

Table 2
Inputs encoding and normalization summary. Specific choices per model in brackets are noted

Feature type	Examples	Encoding/representation	Numerical normalization
Non-cyclic time	year	scalar	min-max or z-score
Cyclic time	day of year, seconds of day	sin-cos encoding	no additional normalization
Altitude	altitude	scalar	min-max (NRLMSISE) or z-score (other models)
Geodetic angles	latitude, longitude	sin-cos encoding (NRLMSISE) or Cartesian unit vector (other models)	no additional normalization
Uniformly generated solar indices	F10.7	scalar	min-max
Historical solar/magnetic indices	F10.7, Ap/Kp	scalar	logarithmic first, then z-score

2.5 Output normalization

Although the models under consideration have similar input parameters, their output parameters vary, as it is shown previously in Table 1.

Outputs were analyzed and it was found that some of parameters in NRLMSISE and IRI have a difference in many orders of magnitude. For instance, NRLMSISE's Argon number density has a minimum ~ 10^{-40} and maximum ~ 10^{5} cm^{-3}, also 5th and 95th percentiles are ~ 10^{-14} and ~ 10^{1}. For IRI the extreme example is electron density with minimum 1 and maximum ~ 10^{6}. Such parameters are good candidates for logarithmic transformation before z-scoring, while direct z-score is a default option for other outputs. As a practical heuristic, logarithmic transformation was applied when the target distribution remained strongly asymmetric or covered several orders of magnitude in the original scale. Otherwise, z-score standardization alone was used. Formulas are used the same as for inputs (8). The summary for outputs is shown in Table 3.

Table 3
Normalization of outputs

Outputs group	Parameters	Normalization
Atmospheric number densities (NRLMSISE)	He, O, N_2, O_2, Ar, H, N, anomalous O	logarithmic transform + z-score
Temperatures (NRLMSISE and IRI)	exospheric, local temperatures; neutral, ion, electron temperatures	z-score
Electron density related (IRI)	electron density, NmF2, TEC	logarithmic transform + z-score
Ion composition % (IRI)	composition of O+, N+, H+, He+, O_2+, NO+	scaled to fractions (specific loss usage)
TEC % above NmF2	TEC % above NmF2	scaled to fractions + z-score
Magnetic field vector	North-East-Down (NED) coordinates	z-score
Acceleration correction vector	components of the vector	z-score

Ion composition is a combined parameter of 6 values which is handled in its own neural networks structure, so its values are just scaled to fractions.

More detailed analysis is described in separate papers of NRLMSISE (Kashyn V., Choliy V., 2025) and IRI (Kashyn V., Choliy V., 2026).

2.6 Neural-network architectures

Since execution speed is one of the main objectives of the surrogate models, the neural-network architecture was selected with computational efficiency in mind. Therefore, a classical multi-layer perceptron (MLP) was used as the primary architecture, because it is one of the simplest feed-forward neural-network structures and consists mainly of linear layers and activation functions, with optional normalization or dropout layers. In this work, preference was given to compact architecture, even if they required longer training or more careful hyperparameter tuning, since smaller models are expected to provide faster inference.

Selecting activation function is an important decision here. ReLU, Tanh, SiLU/Swish were considered as candidate activation functions. During training process and looking for best hyperparameters, we observed better results for SiLU/Swish (Ramachandran et al., 2017) over ReLU and Tanh. However, switching from ReLU to Swish slowed down NRLMSISE neural network model up to 20–25%. This performance drop was solved by switching to HardSwish (Howard, et al., 2019) activation function, which reduced the slowdown relative to Swish to approximately 15–20% for NRLMSISE model. HardSwish is a piecewise approximation of SiLU/Swish:

$$\mathrm{HardSwish}\,(x) = \begin{cases} 0 & x \leq -3, \\ x & x \geq 3, \\ \frac{x(x+3)}{6} & otherwise. \end{cases}$$

9

NRLMSISE, magnetic field and geopotential models are much simpler than IRI model and classic MLP with only linear layers and activation function was enough to prepare the surrogate model. The MLP structure used in this work for NRLMSISE, geomagnetic and geopotential models can be described as follows (referred further as “Simple MLP”):

$$\mathrm{Linear_j}\left(\bar{x}\right) = W_j\,\bar{x} + b_j,\ \ \mathrm{MLPBlock}_j\left(\bar{x}\right) = \mathrm{HardSwish}\left(\mathrm{Linear}_j\left(\bar{x}\right)\right), \mathrm{Output}\left(\bar{x_0}\right) = \mathrm{Linear}_{out}\left(\mathrm{MLPBlock}_{n-1}\left(\mathrm{MLPBlock}_{n-2}\left(\ldots\right.\right.\right.$$

10

where here and further: $\bar{x_0}$ – normalized inputs, $\bar{x_0}\,\epsilon\,\mathbb{R}^{in}$, $\mathrm{Output}\,\epsilon\,\mathbb{R}^{out}$, W_j, b_j – trainable weight matrices and bias vectors of j-th layer,

$W_0 \epsilon\,\mathbb{R}^{w\times\,in}$, $W_j \epsilon\,\mathbb{R}^{w\times\,w}(j>0)$, $W_{out}\,\epsilon\,\mathbb{R}^{out\times\,w}$, $b_j \epsilon\,\mathbb{R}^{w}$, $b_{out} \epsilon\,\mathbb{R}^{out}$, n – the ‘depth’ (number of hidden layers), w – the ‘width’ of hidden layers.

Unlike other models, IRI is much more complex and has a specific output type – relative composition of ions. It was still possible to manage the model using MLP, but with some customizations. The MLP model was growing to handle IRI complexity and when number of hidden layers became more than 7, the training process became unstable. This was solved by adding batch normalization layers in each MLP block.

Another challenge was to handle the relative ion composition separately to preserve its behavior – 6 relative composition elements should be non-negative, and their sum be equal to 1 (the simplex constraint). Separate output heads were used to resolve it. At first, there were two heads – one for scalar outputs and the other for ion composition. However, later it was found that some scalar outputs have better loss while others have much worse loss, so several additional output heads were added to distribute neural network ‘effort’ to problematic scalar. The final structure is shown in Fig. 3, which consists of the common part (body or trunk) and several output heads.

At first, the trunk part consisted of MLPBlocks as defined in (10), however, as the depth was growing the training became unstable, so it was decided to use one-dimensional batch normalization layer (in the PyTorch implementation this corresponds to torch.nn.BatchNorm1d) which is referenced further as BatchNorm (Ioffe et al., 2015). This stabilized the training process and allowed to proceed with deeper MLP model. Heads 1–3 consist of MLPBlocks without normalization layers, as their depth is either 0 (just linear output) or 1. Head 4 is for relative ion composition which needs to satisfy the simplex constraint as mentioned above. IRI neural network structure can be defined as follows (referring further as “Multi-head MLP with normalization”):

$$\begin{aligned}
&\mathrm{TrunkBlock}_j\left(\bar{x}\right) = \mathrm{HardSwish}\left(\mathrm{BatchNorm}_j(\mathrm{Linear}_j\left(\bar{x}\right))\right), \\
&\mathrm{TrunkResult}\left(\bar{x_0}\right) = \mathrm{TrunkBlock}_{t-1}\left(\mathrm{TrunkBlock}_{t-2}\ldots\left(\mathrm{TrunkBlock}_0\left(\bar{x_0}\right)\right)\right), \\
&\mathrm{HeadResult}_1\left(\bar{x_0}\right) = \mathrm{Linear}_{h1,out}\left(\mathrm{MLPBlock}_{h1}\left(\mathrm{TrunkResult}\left(\bar{x_0}\right)\right)\right), \\
&\mathrm{HeadResult}_2\left(\bar{x_0}\right) = \mathrm{Linear}_{h2,out}\left(\mathrm{MLPBlock}_{h2}\left(\mathrm{TrunkResult}\left(\bar{x_0}\right)\right)\right),\ \mathrm{HeadResult}_3\left(\bar{x_0}\right) = \mathrm{Linear}_{h3,out}\left(\mathrm{TrunkResult}\left(\bar{x_0}\right)\right), \\
&\mathrm{HeadResult}_4\left(\bar{x_0}\right) = \mathrm{Softmax}\left(\mathrm{Linear}_{h4,\,out}\left(\mathrm{TrunkResult}\left(\bar{x_0}\right)\right)\right), \quad (11)
\end{aligned}$$

where, in addition to the notation introduced above:

$\mathrm{TrunkResult}\left(\bar{x_0}\right)\,\epsilon\,\mathbb{R}^{w_t}$, t – the trunk depth, number of trunk blocks, w_t – the trunk width, $\mathrm{MLPBlock}_{h1}: \mathbb{R}^{w_t} \rightarrow\ \mathbb{R}^{w_{h1}}$,
$\mathrm{MLPBlock}_{h2}: \mathbb{R}^{w_t} \rightarrow\ \mathbb{R}^{w_{h2}}$, $\mathrm{Linear}_{h\{x\},out}\,\epsilon\ \mathbb{R}^{out_{\{x\}}}$, $out_1,\ out_2,\ out_3,\ out_4$- output dimensions of corresponding heads.

Softmax is defined as follows:

$$\mathrm{Softmax}\left(x_i\right) = \frac{\exp\left(x_i\right)}{\sum_j \exp\left(x_j\right)}$$

12

The summary of neural networks architectures for all models is described in Table 4. More detailed rationale for NRLMSISE and IRI architectures can be found in separate papers of NRLMSISE (Kashyn V., Choliy V., 2025) and IRI (Kashyn V., Choliy V., 2026).

Table 4
Neural networks architectures of the surrogate models and their number of parameters

Surrogate model	Network type	Inputs	Depth	Width	Outputs	Total trainable parameters
NRLMSISE	Simple MLP	in=14	n=4	w=64	out=10	14 090
IRI	Multi-head MLP with normalization	in=27	trunk: t=11	w_t=256	n/a	805 645
			head 1: 1 $\mathrm{MLPBlock}$	w_{h1}=256	out_1=3	
			head 2: 1 $\mathrm{MLPBlock}$	w_{h2}=256	out_2=3	
			head 3: 0	n/a	out_3=1	
			head 4: 0 (Softmax)	n/a	out_4=6	
Magnetic field (IGRF based)	Simple MLP	in=7	n=4	w=64	out=3	13 187
Geopotential (EGM based)	Simple MLP	in=4	n=4	w=128	out=3	50 563

2.7 Loss and evaluation metrics

Here training and final evaluation loss functions used for our models are described.

Mean squared error (MSE), mean absolute error (MAE), per-element absolute percentage error (APE) and mean absolute percentage error (mean APE or MAPE) losses:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,\ MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|,\ APE_i = 100\% \bullet \left|\frac{y_i - \hat{y}_i}{y_i}\right|,\ MAPE = \frac{1}{N}\sum_{i=1}^{N} APE_i$$

13

where here and further:

N – number of samples, y_i, $\hat{y}_i$ – i-th sample of reference and predicted values.

All reference values used in the MAPE calculation were non-zero in the tested input domain.

Brier (Brier, 1950) and total variation distance (TVD) losses for relative ion composition:

$$Brier = \frac{1}{NK}\sum_{i=1}^{N}\sum_{j=1}^{K}(y_{ij} - \hat{y}_{ij})^2,\ TVD = \frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1}^{K}\left|y_{ij} - \hat{y}_{ij}\right|,$$

14

where:

N – samples number of dataset or resulting batch, K – number of components (for our ion composition, $K = 6$), y_{ij}, $\hat{y}_{ij}$ – j-th composition element of i-th sample.

The summary for loss function usages is outlined in Table 5.

Table 5
Loss functions

Model	Training and validation loss (scalar)	Main evaluation results loss (per output parameter)
NRLMSISE	Mean MSE	MAPE
IRI	Mean MAE + Brier	MAPE and TVD
Magnetic field (IGRF based)	Mean MSE	MAPE
Geopotential (EGM based)	Mean MAE	MAPE

2.8 Other hyperparameters and training process

The AdaMax optimizer (Kingma, et al., 2015) was used for the final surrogate model for NRLMSISE. It is an Adam-family optimizer based on infinity-norm variant of adaptive moment estimation. However, applying the same optimizer to the IRI surrogate led to convergence difficulties. This was resolved by switching to AdamW, an Adam-family optimizer with decoupled weight decay (Loshchilov, et al., 2019), which reduced the training loss. Therefore, AdamW was selected as the initial choice for magnetic and geopotential models, and both models were trained successfully to meet our loss requirements.

Based on results, AdamW can be a reasonable starting point for similar surrogate models. However, the weight-decay parameter should be tuned carefully, because in our experiments, even changes on the order of 10^{-4} noticeably affected the training behavior. Detailed optimizer parameters are described in Table 6.

Table 6
Optimizers and related parameters

Model	Optimizer	First moment (β_1)	Second moment (β_2)	Weight decay
NRLMSISE	AdaMax	0.8–0.9 (cyclic scheduler, inverse momentum)	0.999	0
IRI	AdamW	0.9	0.999	0.005; 0 for normalization layer parameters
Magnetic field (IGRF based)	AdamW	0.9	0.999	0.0001
Geopotential (EGM based)	AdamW	0.9	0.999	0.0001

Two types of learning rate (LR) schedules were used in our work: cyclical learning-rate schedule with inverse momentum coefficient (Smith, 2015; Smith, 2018) and one-cycle (“1cycle”) learning rate policy (Smith, 2017) with cosine annealing, but without cycling the first momentum. The schedulers setup per model is shown in Table 7.

Table 7
Schedulers and their parameters

Model	Scheduler	LR ranges	Schedule duration
NRLMSISE	Cyclical LR	$\mathrm{Cyclerange}: [10^{-4}; 1.3 \bullet 10^{-2}]$	Ascent epochs: 1 000; Descent epochs: 10 000; ~5.5 cycles.
IRI	One-Cycle LR	Phase 1: $2 \bullet 10^{-5} \rightarrow 2 \bullet 10^{-3}$; Phase 2: $2 \bullet 10^{-3} \rightarrow 10^{-6}$.	Phase 1: 1 000; Phase 2: 9 000.
Magnetic field (IGRF based)	One-Cycle LR	Phase 1: $10^{-4} \rightarrow 1.5 \bullet 10^{-3}$; Phase 2: $1.5 \bullet 10^{-3} \rightarrow 10^{-6}$.	Phase 1: 5 000; Phase 2: 15 000.
Geopotential (EGM based)	One-Cycle LR	Phase 1: $10^{-4} \rightarrow 3 \bullet 10^{-3}$; Phase 2: $3 \bullet 10^{-3} \rightarrow 10^{-6}$.	Phase 1: 1 200; Phase 2: 10 800.

Here and further, one epoch is one full iteration over all samples from the training dataset and a batch denotes the subset of training samples processed during one optimization step before updating the neural network parameters. For each batch, the training loss was computed in normalized space, gradients were obtained by backpropagation, and model parameters were updated according to the selected optimizer and learning-rate schedule. Remaining training process parameters are described in Table 8. Since training time depends strongly on hardware, the reported values are provided mainly as reference measurements for the experimental setup used in this work. However, all surrogate models were trained in the same hardware and software environment, so these values can still be used for a relative comparison of training cost between the models.

Table 8
Batch sizes, number of epochs, training time

Model	Training dataset size, number of samples	Validation dataset size, number of samples	Batch size, number of samples	Total number of epochs	Training time, minutes
NRLMSISE	500 000	500 000	2 000	60 000	~ 55
IRI	3 000 000	500 000	8 000	10 000	~ 246
Magnetic field (IGRF based)	2 500 000	500 000	2 000	20 000	~ 78
Geopotential (EGM based)	5 000 000	1 000 000	4 000	12 000	~ 97

Training and validation datasets were generated as described in section 2.2 of this paper. The validation dataset was used for model selection and monitoring, and test data were used only for final accuracy evaluation. A model snapshot with the lowest validation loss was selected as the final surrogate model and evaluated in the Results section. Importantly, the test data reported in the Results section were not used for model training or selection.

The following setup was used: PyTorch 2.10, CUDA 13.0, Windows 11, CPU AMD Ryzen 9 5900X, GPU NVIDIA RTX 4070, SSD Kingston SA2000M81000G. CUDA Graphs were used during training. For the NRLMSISE surrogate, mixed-precision computations using FP16 and FP32 floating-point formats were applied. For the IRI and magnetic-field surrogates, TensorFloat-32 (TF32) operations were enabled in PyTorch by setting the float32 matrix-multiplication precision to “high”. The geopotential surrogate was evaluated using the default PyTorch settings, with TF32 disabled and float32 matrix-multiplication precision set to “highest”. Additional training process details and training curves for NRLMSISE and IRI can be found in their separate articles.

2.9 Outputs restoration

The neural networks were trained to approximate the normalized target outputs defined in Section 2.5, so the raw surrogate-model predictions are normalized quantities rather than physical values. To get real values the corresponding inverse transformations must be applied. Inverse operations for z-score and logarithmic transformation are the following:

$$x_i^{(k)} = z_i^{(k)}\sigma^{(k)} + \mu^{(k)}, \;\; x_i^{(k)} = \exp\left(l_i^{(k)}\right) - 1.$$

15

The same notation is used as in (8). For the outputs where logarithmic transformation was used before z-score, restoration should be performed in the reverse direction as well:

$$x_i^{(k)} = \exp\left(z_i^{(k)}\sigma^{(k)} + \mu^{(k)}\right) - 1,$$

16

where $\sigma^{(k)}$ and $\mu^{(k)}$ parameters of dataset which were calculated after logarithmic transformation.

3 Results

3.1 Test environment and common conditions

Separate datasets for results evaluations are used – test datasets. They are generated independently from training and validation datasets using the approach specified in section 2.2 and normalized using metrics achieved during training dataset generation in sections 2.2–2.5 (e.g. standard deviation, mean, etc.).

Loss and execution speed comparisons are done against the reference C + + implementations of models which are used in Juliette application. C + + reference applications are built using MSVC compiler in Release configuration, without debugging information and using optimizations towards execution speed (/O2). C + + reference applications are run without debugger attached and their execution time contains only evaluation of specified number of samples without writing them to file. Neural networks model timings include time of reading model parameters from filesystem, one feed-forward run of

model on specified number of samples and output restoration to original physical values. No CUDA warmup was performed before running models and CUDA synchronization is performed before reading elapsed time for all runs in Results section.

Environment used in tests: PyTorch 2.7–2.10 runtime, CUDA 12.6–13.0, Windows 11, CPU AMD Ryzen 9 5900X, GPU NVIDIA RTX 4070 12 GB, 64 GB RAM, SSD Kingston SA2000M81000G.

All approximation errors are outlined using loss functions described in section 2.7, “Evaluation results loss” row.

3.2 Surrogate-model approximation errors

All models except the geopotential model were evaluated using test datasets of 1,000,000 samples to estimate approximation errors. For the geopotential model, a test dataset of 2,000,000 samples was used, consistent with its two-times larger training and validation datasets. Outputs were restored to physical units and losses calculated as below. These test datasets and related neural network predictions are available at Zenodo repository (Kashyn V. and Choliy V., 2026).

NRLMSISE resulting errors are outlined in Tables 9 and 10.

Table 9
NRLMSISE loss results for each number density output; P95: 95th percentile

Loss statistics	He	O	N_2	O_2	Ar	H	N	O anomalous
Mean APE	0.171%	0.414%	0.560%	0.648%	0.850%	0.137%	0.401%	0.168%
Median APE	0.128%	0.312%	0.409%	0.475%	0.629%	0.106%	0.307%	0.135%
P95 APE	0.463%	1.112%	1.527%	1.760%	2.289%	0.364%	1.064%	0.432%

Table 10
NRLMSISE loss results for temperatures and restored total mass density; P95: 95th percentile

Loss statistics	Temperature exospheric	Temperature at height	Total mass density
Mean APE	0.047%	0.046%	0.245%
Median APE	0.035%	0.036%	0.169%
P95 APE	0.127%	0.129%	0.730%

Total mass density is not a direct output of surrogate model; it is restored based on number densities predicted by neural network.

IRI loss results are described in Table 11. The final mean IRI loss for scalar outputs is 0.409% (MAPE). The final IRI mean TVD loss for ion composition is 0.24%; median TVD: 0.15%; P95 TVD: 0.80%.

Table 11
IRI loss results for scalar outputs (MAPE)

Loss statistics	Electron density	NmF2	Temperature neutral	Temperature ion	Temperature electron	TEC	TEC top %
Mean APE	0.954%	0.384%	0.093%	0.466%	0.200%	0.392%	0.371%
Median APE	0.557%	0.285%	0.045%	0.129%	0.107%	0.272%	0.354%
P95 APE	2.733%	1.068%	0.207%	0.817%	0.641%	1.041%	0.875%

Neural network results for magnetic field and geopotential models are shown in Tables 12 and 13. Related vector-valued outputs were additionally evaluated using the relative percentage L_2 error, defined for each sample:

$$\epsilon_{rel,\,2}\left(\varvec{y}_{\varvec{i}}\right) = 100\% \bullet \frac{\left|\left|\varvec{y}_{\varvec{i}} - \widehat{\varvec{y}}_{\varvec{i}}\right|\right|_2}{\left|\left|\varvec{y}_{\varvec{i}}\right|\right|_2}.$$

Table 12
Component-wise APE and relative-vector-error distributions for the magnetic-field surrogate; P95: 95th percentile

Loss statistics	B_{North} APE	B_{East} APE	B_{Down} APE	$\epsilon_{rel,2}(\mathbf{B})$
Mean	0.081%	0.732%	0.257%	0.021%
Median	0.015%	0.073%	0.034%	0.016%
P95	0.117%	1.171%	0.354%	0.057%

Table 13
Component-wise APE and relative-vector-error distributions for the geopotential surrogate; P95: 95th percentile

Loss statistics	Δa_x APE, %	Δa_y APE, %	Δa_z APE, %	$\epsilon_{rel,2}(\varvec{\Delta \mathbf{a}})$
Mean	0.981%	0.850%	0.230%	0.053%
Median	0.033%	0.032%	0.024%	0.027%
P95	0.910%	0.847%	0.372%	0.181%

Component-wise APE values for the magnetic-field and geopotential surrogates should be interpreted with caution because APE can become large when the corresponding reference component approaches zero. The relative L_2 vector error is therefore more informative for evaluating these vector-valued outputs.

3.3 Execution time improvement

Tables 14–17 compare the execution times of the neural-network surrogates with those of the reference C + + implementations. Values in parentheses indicate the speedup factor relative to the corresponding single-threaded C + + baseline. The reported timings are means of 10 independent cold runs in environment mentioned in section 3.1, each performed in a newly started process and including model loading.

Table 14
NRLMSISE time improvements

Number of NRLMSISE samples calculated	C + + time, seconds	Neural network NRLMSISE time, seconds		
	CPU	CPU	CPU	GPU
	1 thread	1 thread	12 threads	CUDA
100 000	0.262	0.087	0.079	0.109
	(1× reference)	(3.0× boost)	(3.3× boost)	(2.4× boost)
500 000	1.314	0.310	0.192	0.109
	(1× reference)	(4.2× boost)	(6.8× boost)	(12.1× boost)
1 000 000	2.566	0.596	0.324	0.124
	(1× reference)	(4.3× boost)	(7.9× boost)	(20.7× boost)
2 000 000	5.420	1.268	0.683	0.138
	(1× reference)	(4.3× boost)	(7.9× boost)	(39.28× boost)

Table 15
IRI time improvements

Number of IRI samples calculated	C++ time, seconds	Neural network IRI time, seconds		
	CPU 1 thread	CPU 1 thread	CPU 12 threads	GPU CUDA
10 000	60.4 (1× reference)	0.224 (270× boost)	0.127 (476× boost)	0.209 (289× boost)
100 000	586.3 (1× reference)	1.82 (322× boost)	0.671 (874× boost)	0.224 (2 617× boost)
500 000	2 893.2 (1× reference)	8.85 (327× boost)	3.55 (815× boost)	0.323 (8 957× boost)
1 000 000	5 969.9 (1× reference)	17.97 (332× boost)	7.06 (841× boost)	0.436 (13 692× boost)
2 000 000	12 931.7 (1× reference)	39.22 (330× boost)	15.93 (812× boost)	0.711 (18 188× boost)

Table 16
Magnetic field (IGRF based) time improvements

Number of magnetic field model samples calculated	C++ time, seconds	Neural network magnetic field time, seconds		
	CPU 1 thread	CPU 1 thread	CPU 12 threads	GPU CUDA
10 000	0.721 (1× reference)	0.056 (12.9× boost)	0.055 (13.1× boost)	0.123 (5.86× boost)
100 000	7.167 (1× reference)	0.096 (74.7× boost)	0.076 (94.3× boost)	0.125 (57.3× boost)
500 000	35.802 (1× reference)	0.307 (117× boost)	0.183 (196× boost)	0.126 (284× boost)
1 000 000	72.222 (1× reference)	0.575 (126× boost)	0.319 (226× boost)	0.136 (531× boost)
2 000 000	144.615 (1× reference)	1.145 (126× boost)	0.650 (222× boost)	0.157 (921× boost)

Table 17
Geopotential (EGM based) time improvements

Number of geopotential model samples calculated	C++ geopotential model time, seconds	Neural network geopotential model time, seconds		
	CPU 1 thread	CPU 1 thread	CPU 12 threads	GPU CUDA
10 000	0.095 (1× reference)	0.063 (1.51× boost)	0.057 (1.67× boost)	0.126 (0.75× boost)
100 000	0.948 (1× reference)	0.179 (5.30× boost)	0.097 (9.77× boost)	0.128 (7.41× boost)
500 000	4.750 (1× reference)	0.705 (6.74× boost)	0.308 (15.4× boost)	0.136 (34.9× boost)
1 000 000	9.545 (1× reference)	1.360 (7.02× boost)	0.584 (16.3× boost)	0.151 (63.2× boost)
2 000 000	19.060 (1× reference)	2.691 (7.08× boost)	1.258 (15.2× boost)	0.193 (98.8× boost)

The achieved acceleration depends strongly on size of input. For small number of samples, fixed neural network overheads become noticeable: CPU inference may outperform GPU inference because GPU execution includes additional framework, kernel-launch, and, where applicable, data-transfer overheads. In addition, “cold” runs of the neural-network models include model-initialization and framework-setup costs, which were approximately 50–150 ms in our experiments. Therefore, if the reference implementation runs a small number of samples within a comparable time interval (< ~ 100ms), it may appear faster than the neural-network model. This overhead may be reduced in warm-run scenarios, where the model and framework are already initialized. For instance, the first “cold” run of geomagnetic model takes 56ms for 10 000 samples batch on CPU 1 thread, but next “warm” run of 10 000 samples takes only ~5ms, because it does not require framework initialization. However, all these effects are limited to very short, sub-second measurements and become negligible for the large-batch evaluation scenarios considered in this work.

The magnitude of acceleration also differs between models. The largest speedups were obtained for IRI, whose reference implementation is computationally expensive compared with a single feed-forward pass of the surrogate model. Smaller, but still significant, speedups were observed for NRLMSISE and the geopotential model, where the reference implementations are already relatively fast. Therefore, the reported speedups should be interpreted together with the computational cost of the corresponding reference model and the evaluated batch size.

3.4 Results summary

A summary of achieved results is shown in Table 18.

Table 18
Results summary; ×-multiplier denotes number of times neural network (NN) model is faster than the reference (ref. C++) one

Model **Result metric**	**NRLMSISE**	**IRI**	**Magnetic field**	**Geopotential**
Maximum speed up: NN CPU 1 thread vs ref. C + + 1 thread	4.3×	332×	126×	7.08×
Maximum speed up: NN CPU 12 threads vs ref. C + + 1 thread	7.9×	874×	226×	16.3×
Maximum speed up: NN GPU RTX 4070 vs ref. C + + 1 thread	39.3×	18 188×	921×	98.8×
Worst reported scalar/component, Mean APE	0.850%	0.954%	0.732%	0.981%
Additional model specific summary error	Total mass density MAPE: 0.245%	All scalars mean MAPE: 0.409%; ions TVD: 0.24%	Mean relative vector error: 0.021%	Mean relative vector error: 0.053%

The comparison indicates that the achievable speedup depends primarily on the computational cost of the reference implementation, whereas surrogate complexity is driven by the structure of the target outputs. IRI required the largest and most specialized architecture because of its heterogeneous scalar quantities and constrained ion-composition output, but it also produced the largest speedup because its reference implementation was the most expensive. The NRLMSISE, magnetic-field, and geopotential cases were represented by compact MLPs, although their normalization and output-restoration requirements differed. These results support a common surrogate-development workflow while showing that architecture, output constraints, and evaluation metrics can remain model specific.

4 Conclusion

This study presented a workflow for developing and evaluating neural-network surrogates intended to accelerate repeated evaluations of empirical geophysical models. The workflow was applied to four models representing different components of the Earth and near-Earth environment: NRLMSISE-00, IRI 2020, an IGRF-14-based magnetic-field model, and an EGM96-based geopotential model. The analysis identified both common workflow stages and model-specific choices in data representation, normalization, neural-network architecture, training, and output restoration.

All four surrogates achieved sub-percent mean approximation errors according to the model-appropriate evaluation metrics within the tested input domains. For every evaluated scalar or vector component, the mean APE was below 1%. The mean TVD for IRI ion composition was 0.24%, while the mean relative vector errors for the magnetic-field and geopotential models were 0.021% and 0.053%, respectively.

The surrogate models also demonstrated substantial execution-time improvements compared with the original C + + implementations. On the *same* single-threaded CPU setup, the best observed speedups were approximately 4.3× for NRLMSISE, 332× for IRI, 126× for the magnetic-field model, and 7.08× for the geopotential model. With multi-threaded CPU inference, the corresponding best speedups increased to approximately 7.9×, 874×, 226×, and 16.3×,

respectively. The largest improvements were obtained using GPU inference on an NVIDIA RTX 4070, where the best speedups reached approximately 39× for NRLMSISE, 18 188× for IRI, 921× for the magnetic-field model, and 98.8× for the geopotential model.

The results demonstrate that heterogeneous empirical geophysical models can be represented by compact neural-network surrogates with sub-percent mean approximation errors within the tested input domains and substantial reductions in evaluation time. From an Earth science informatics perspective, the workflow provides a structured method for transforming established model implementations and their generated data into efficient data-driven computational components. The speedups obtained are particularly relevant to large-scale Earth and space science applications requiring repeated model evaluations, including satellite orbit propagation, observation modelling, prediction, and simulation workflows.

At the same time, the surrogate models should be interpreted as approximations valid within the training domains used in this work. Their use outside the sampled input ranges would require additional validation or retraining. Future work may include testing the surrogates in full orbit-determination or orbit-prediction workflows, for instance, within Juliette application.

A detailed investigation of the sources of the observed speedups, including the gains observed in the same single-threaded CPU comparison, was outside the scope of this paper. However, preliminary profiling of the IRI C + + reference implementation indicated that mathematical operations accounted for a large fraction of the execution time. This suggests that part of the acceleration may be related to replacing repeated model-specific numerical computations with compact neural-network inference.

Declarations

Declarations

Competing interests

The authors have no competing interests to declare that are relevant to the content of this article.

Funding

No funding was received for conducting this study.

Author Contribution

V. K.: methodology, implementation, data curation, investigation, validation, formalanalysis, writing - original manuscript draft;V. C.: conceptualization of the study, provision of reference modelimplementations, supervision, writing - review and editing manuscript.

Data Availability

The independent test datasets used to calculate the reported approximation-error statistics are available in Zenodo at https://doi.org/10.5281/zenodo.21444921. The repository includes the input parameters, outputs of the corresponding reference C++ implementations, restored neural-network predictions, column descriptions, physical units.

References

1. Acciarini G, Brown E, Berger T, Guhathakurta M, Parr J, Bridges C, Baydin AG (2024) Improving thermospheric density predictions in low-Earth orbit with machine learning. Space Weather 22:e2023SW003652. https://doi.org/10.1029/2023SW003652
2. Beggan CD, Kloss C, Amblard P et al (2026) International geomagnetic reference field: the fourteenth generation. Earth Planet Space 78:127. https://doi.org/10.1186/s40623-025-02360-0
3. Bilitza D, Pezzopane M, Truhlik V, Altadill D, Reinisch BW, Pignalberi A (2022) The International Reference Ionosphere model: a review and description of an ionospheric benchmark. Rev Geophys 60. https://doi.org/10.1029/2022RG000792. e2022RG000792
4. Brier GW (1950) Verification of forecasts expressed in terms of probability. Mon Weather Rev 78(1):1–3. https://doi.org/10.1175/1520-0493(1950)078%3C0001:VOFEIT%3E2.0.CO;2
5. Choliy V (2021) Kinem Phys Celest Bodies 37(5):66–73. https://doi.org/10.15407/kfnt2021.05.066. Coordinates of the pole and length of the day from reprocessing of LEO SLR observations
6. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals Syst 2:303–314. https://doi.org/10.1007/BF02551274
7. Dutta S, Cohen MB (2023) Topside electron density modeling using neural network and empirical model predictions. Space Weather 21. https://doi.org/10.1029/2023SW003501. e2023SW003501
8. Howard, Andrew & Pang, Ruoming & Adam, Hartwig & Le, Quoc & Sandler, Mark & Chen,Bo & Wang, Weijun & Chen, Liang-Chieh & Tan, Mingxing & Chu, Grace & Vasudevan, Vijay& Zhu, Yukun. (2019). Searching for MobileNetV3. 1314–1324. https://doi.org/10.1109/ICCV.2019.00140.

9. Ioffe S, Szegedy C (2015) ‘Batch normalization: Accelerating deep network training by reducing internal covariate shift’, Proceedings of the 32nd International Conference on Machine Learning, 37, pp. 448–456. Available from https://proceedings.mlr.press/v37/ioffe15.html
10. Kashyn V, Choliy V (2025) Accelerating atmosphere modeling: neural network enhancements for faster NRLMSISE calculations. Artif Satell 60(3):121–136. https://doi.org/10.2478/arsa-2025-0007
11. Choliy V (2026) Accelerating geophysical models via Neural Networks: A case study of NRLMSISE and IRI, International Association of Geodesy Symposia, Proceedings of the IAG Scientific Assembly 2025, Rimini, Italy. Accepted for publication
12. Kashyn V, Choliy V (2026) Geophysical reference-models test datasets and neural-network surrogate predictions [Data set]. Zenodo. https://doi.org/10.5281/zenodo.21444921
13. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization, Proc. 3rd Int. Conf. on Learning Representations (ICLR 2015), San Diego, CA, USA, 7–9 May. Available at: https://arxiv.org/abs/1412.6980
14. Lemoine FG, Kenyon SC, Factor JK, Trimmer RG, Pavlis NK, Chinn DS, Cox CM, Klosko SM, Luthcke SB, Torrence MH, Wang YM, Williamson RG, Pavlis EC, Rapp RH, Olson TR (1998) The Development of the Joint NASA GSFC and the National Imagery and Mapping Agency Geopotential Model EGM96. NASA/TP-1998-206861. NASA Goddard Space Flight Center, Greenbelt, MD
15. Loshchilov I, Hutter F (2019) Decoupled weight decay regularization, in: Proc. 7th Int. Conf. on Learning Representations (ICLR 2019), New Orleans, LA, USA, 6–9 May. https://doi.org/10.48550/arXiv.1711.05101
16. Lu and Ricciuto (2019) Efficient surrogate modeling methods for large-scale Earth system models based on machine-learning techniques. Geosci Model Dev 12:1791–1807. https://doi.org/10.5194/gmd-12-1791-2019
17. Matzka J, Bronkalla O, Tornow K, Elger K, Stolle C (2021) Geomagnetic Kp index. V. 1.0. https://doi.org/10.5880/Kp.0001. GFZ Data Services
18. Pan Q, Xiong C, Lühr H, Smirnov A, Huang Y, Xu C et al (2024) Machine learning based modeling of thermospheric mass density. Space Weather 22. https://doi.org/10.1029/2023SW003844. e2023SW003844
19. Picone JM, Hedin AE, Drob DP, Aikin AC (2002) NRLMSISE-00 empirical model of the atmosphere: statistical comparisons and scientific issues. J Geophys Res 107(A12):1468. https://doi.org/10.1029/2002JA009430
20. Ramachandran P, Zoph B, Le QV (2017) Searching for activation functions. arXiv preprint. https://doi.org/10.48550/arXiv.1710.05941
21. Smith LN (2015) Cyclical learning rates for training neural networks. arXiv preprint arXiv. https://doi.org/10.48550/arXiv.1506.01186. :1506.01186
22. Smith LN (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. https://doi.org/10.1109/WACV.2017.58
23. Smith LN (2018) A disciplined approach to neural-network hyper-parameters: Part 1 – learning rate, batch size, momentum and weight decay. arXiv preprint arXiv. https://doi.org/10.48550/arXiv.1803.09820. :1803.09820
24. Yang D, Fang H (2023) A low-latitude three-dimensional ionospheric electron density model based on radio occultation data using artificial neural networks with prior knowledge. Space Weather 21:e2022SW003299. https://doi.org/10.1029/2022SW003299

Figures

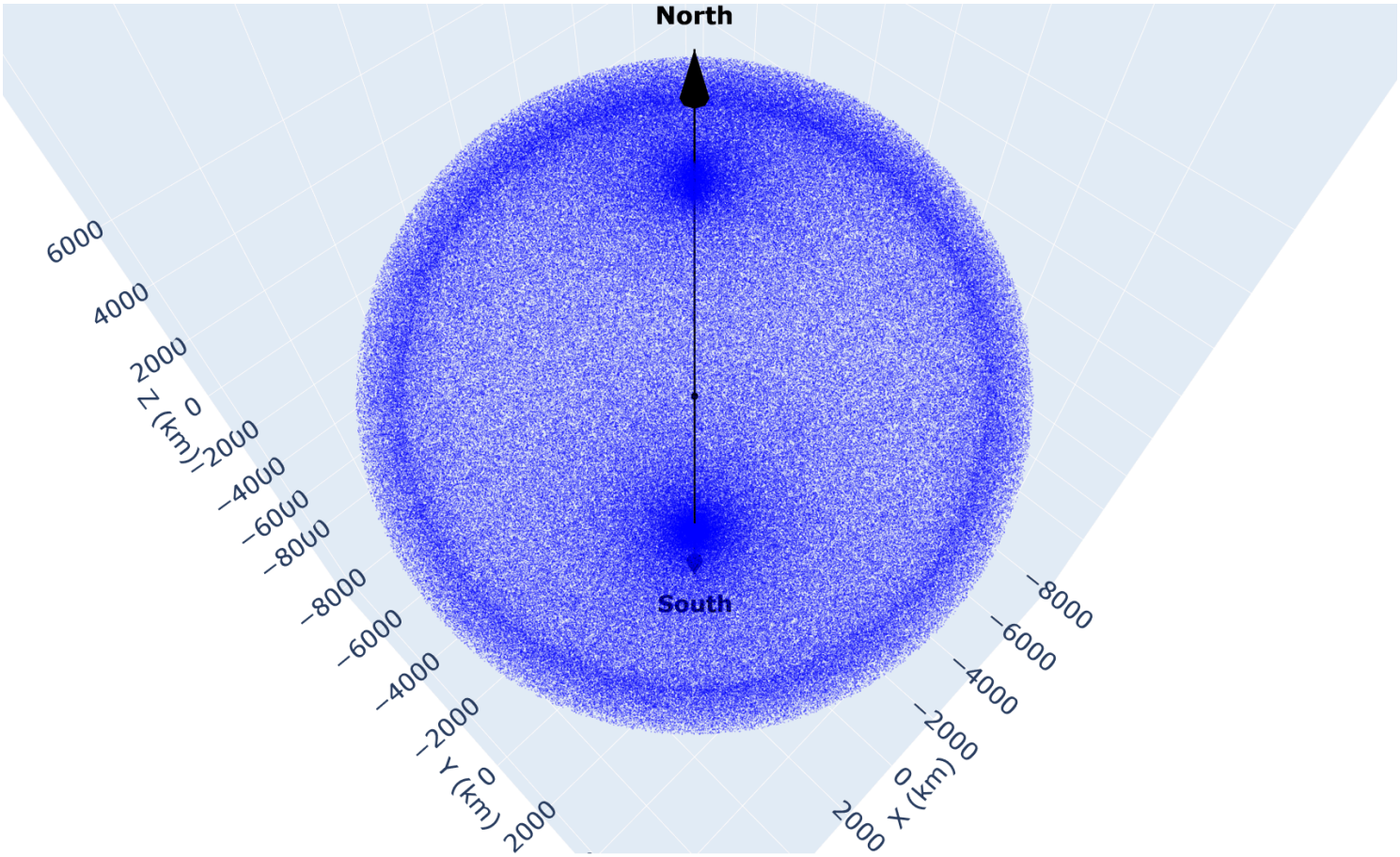


Figure 1

Distribution of points with higher concentration on poles when using naive uniform distribution for longitude, latitude, and altitude

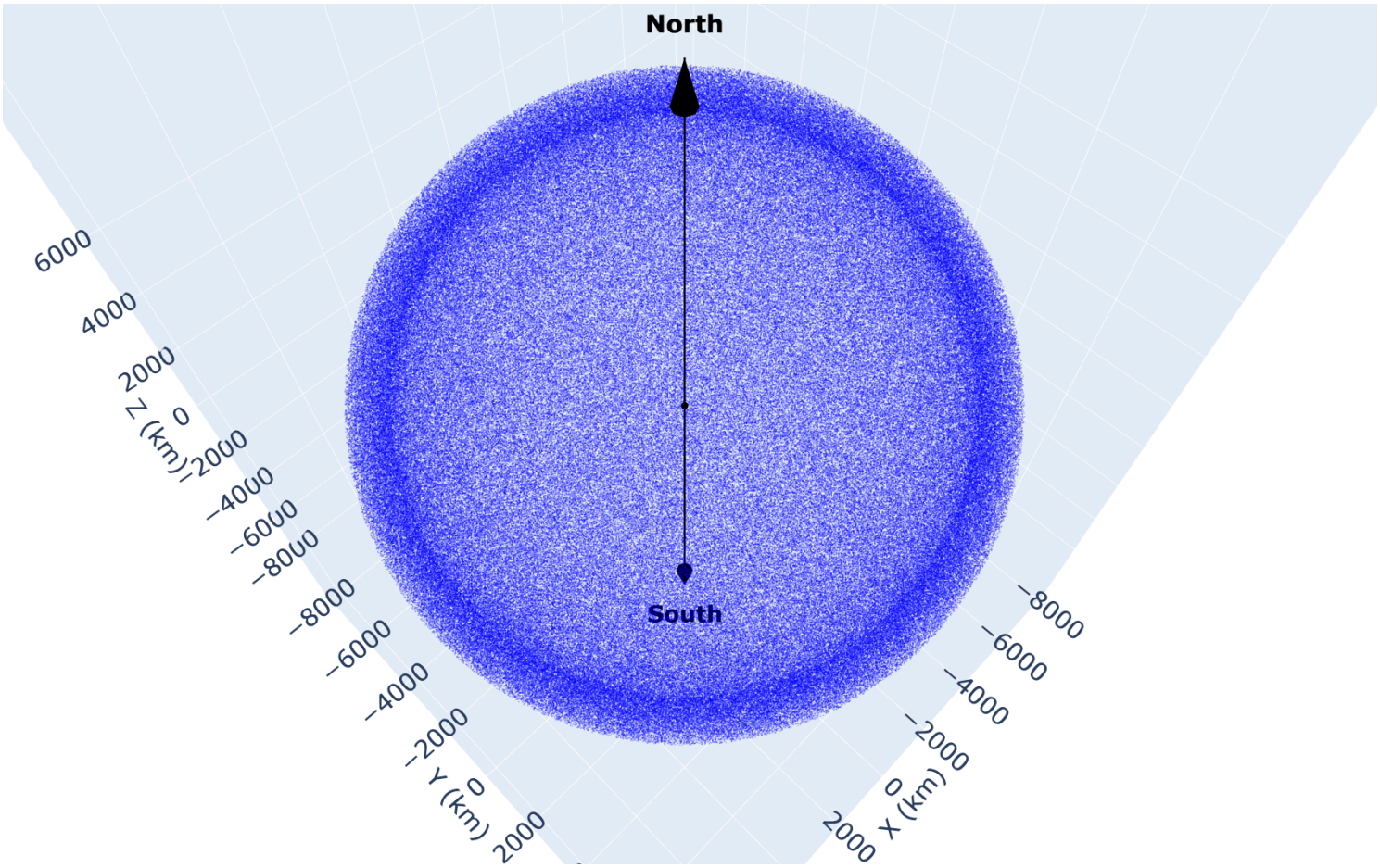


Figure 2

Uniform distribution of points for longitude, latitude, and altitude on a spherical shell

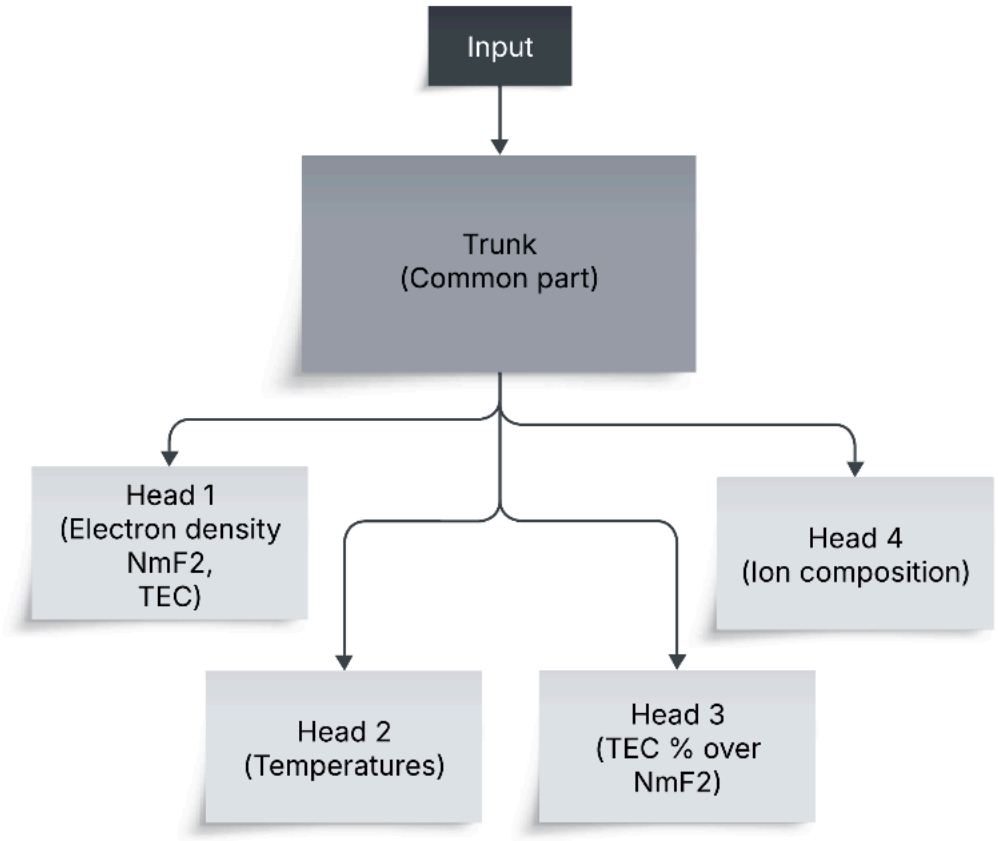


Figure 3

IRI schematic neural network structure (Kashyn V., Choliy V., 2026)