Predicting Future Orbital States Across Three Different Orbit Types Using Artificial Neural Networks

Nayana Jayakumar
nayanajaypillai@gmail.com

ABSTRACT

With the rapid growth in space exploration and commercialization, the number of Resident Space Objects (RSO's) has significantly increased, raising the risk of orbital collision. As of September 2025, there are around 23030 satellites in Earth's orbit with more than 650 estimated collisions, according to the European Space Agency.

This data highlights the need for improved methods to accurately predict satellite trajectories and mitigate the collision risk. However, classical propagators, like SGP4, suffer from uncertainty due to perturbations. Most of the Machine Learning (ML) research done in this area explores satellites and debris orbiting the earth in Low Earth Orbits (LEO), Geosynchronous Earth Orbits (GEO) or both, with varying success . These researches were done using either Recurrent Neural Networks (RNNs) like Long Short Term Memory (LSTM) or Reinforcement Learning (RL) models.

This project proposes Astraeus, a set of six Artificial Neural Network (ANN) models that can generalize across LEO, Medium Earth Orbits (MEO), and GEO; based on an input of the Keplerian elements. Using Two-Line Element (TLE) data from Space-Track, Keplerian elements were converted into position and velocity state vectors using SGP4 for training. Astraeus learns from historical orbital data to predict future satellite state vectors without relying on analytical approximations. Results show that Astraeus achieved an overall prediction accuracy of 94%, with position vectors exhibiting lower percent error than velocity components. These findings demonstrate the potential of machine learning to enhance satellite navigation and space traffic management systems by improving orbit prediction across multiple orbits.

INTRODUCTION

This introduction first addresses the problem, traditional propagation methods, then ML propagation methods before introducing Astreaus and why it is necessary in this field.

An estimated 140 million pieces of space debris orbit the Earth, according to the European Space Agency [1]. To mitigate collision risk and maintain satellite functionality, it is necessary to identify the orbital

paths of each Resident Space Object (RSO). Thus, the growth in RSOs surrounding the Earth creates a need for accurate and precise orbit prediction methods.

Traditional orbit propagation methods are analytical, semi-analytical, and numerical. To start off, analytical propagators use pre-defined equations of motion to predict state vectors but are less accurate than their numerical counterparts and less computationally demanding. Some examples are SGP4 and TwoBody. Meanwhile, numerical propagators, or Integrators, take in forces and the initial state to produce the state of a satellite. They are highly accurate but computationally demanding [2]. Meanwhile, semi-analytical propagators have numerical accuracy and analytical speed; they express the orbital state as a component with slow dynamics and another that collects fast variations [3]. However, semi-analytical propagators struggle with LEO because these perturbations have a lot of atmospheric drag. Some have tried to remedy the situation by adding an atmospheric model to help in the spatial density and decay rate [4].

A study by Saika Aida and Michael Kirschner claims that the problem with other analytical propagators, specifically SGP4, is that they struggle with accuracy. SGP4 constructs mean orbital elements by taking away periodic variances, and then the model must reconstruct them perfectly to obtain a good prediction, leaving room for error if not handled properly. Furthermore, SGP4 struggles over longer propagation periods in the long track directions and imperfect two-line element data (TLE) leads to further inaccuracy in Orbit Determination (OD) [5].

The limitations of traditional orbital propagators leaves a gap in OD that ML methods can fill using less computing power while being more cost-efficient and precise. The success of ML models in this field offers a solution to the problem collisions of RSOs pose; if achieved, they can be implemented in newer satellites to mitigate collision risk.

Long Short Term Memory Models (LSTM) has been utilized in orbit prediction, with one study [6] using a novel hybrid approach with two ML estimators, specifically random forest and the autoencoder in order to reduce errors in SGP4 caused by incomplete perturbation calculations and low-order series expansions. However, another study evaluates the potential of different ML algorithms (like KNeighbor's regressor, Random Forests regressor, Linear Regression, and Polynomial Regression) on LEO Swarm-A's satellite data and MEO GNSS' satellite data, noting that a pure ML approach, particularly Polynomial Regression, shows the most promise in predicting orbital paths [7]. Meanwhile, another study used a Recurrent Neural Network (RNN) algorithm (LSTM) along with Bayesian search technique for optimized high parameter search to solve the gaps in outdated faulty TLE datasets with Root Mean Squared Error reduction of (RMSE) 54% in position and 73% in velocity for LEO satellites [8].

Using data from the International Laser Ranging Service, Caldas et al use machine learning and time series techniques in OP, generating low-positioning errors with low computational cost. Interestingly, this study uses exogenous variables as its features, of which there are 27. Since SGP4 struggles with forces, the exogenous variables were picked in their relevance to the nonconservative forces acting on the

satellite. There are 8 features of Space Weather Drivers, 3 Velocity features, 6+6 Keplerian(a, e, i, ω, Ω, ν) and Equinoptical(p, f, g, h, k, l), 1 Total Mass Density feature, and 3 Gravity Field Functionals' features [9]. On the other hand, another study uses a Time Delay Neural Network (TDNN) for orbital state prediction in LEO satellites. It calculates the error of SGP4's propagation in comparison with a high precision propagator, then the ML model can be used to apply the past corrections from ephemeris errors onto future orbital propagations [10].

Another approach by [11] involves the use of Physics-Informed-Neural-Networks (PINNs) which managed to outperform the best-fit physics model when predicting the orbital state and low-amplitude anomalous acceleration for GEO satellites. PINNs are often used because they combine physics with Deep Neural Networks (DNNs) so the ML model doesn't have to learn astrodynamics, becoming computationally less costly.

However, many of these studies use complex models (iTransformer and Prophet from [9]) where overfitting becomes more likely and the authors note that their method cannot match the accuracy and precision of numerical propagators. Additionally, many existing works typically address one or two types of orbits; ANN exploring MEO, GEO, and LEO has not been researched enough. This is where Astraeus comes in with its simple but promising pure ML, ANN approach. This paper attempts to generalize across three different orbits: LEO, MEO, and GEO. It aims to respond to the research question: "To what extent can an ANN model predict future orbital states across three different orbits based on historical data compared to traditional OD methods?"

METHODS

Data and Preprocessing

The dataset obtained is a combination of data from Space-Track and transformed with SGP4. Space Track is a website maintained by the US Space Force, and it contains orbital data for satellites and space debris. A subset of representative satellites orbiting in LEO, MEO and GEO were identified for this experiment. For instance, satellites like STARLINK and ONEWEB were used for LEO, while BEIDOU and GSAT were used for GEO, and COSMOS and GALILEO satellite data were used for MEO. Each satellite has a propagation period of roughly one month, from either October to November of 2025 or December 2025 to January 2026.

Table 1: List of satellites included in dataset

Orbit Class	Satellite Types	Total Satellites Included	Total Observations Across All Satellites
LEO	Starlink, Oneweb, Envisat	5	419
GEO	Echostar, TDRS, Beidou, IRNSS	5	379
MEO	Cosmos, Galileo, Viking, OPS, Navstar	12	403

As seen in Table 1, each orbital class has a somewhat similar amount of observations - the goal was to ensure that each type had roughly ~ 400 observations, with fluctuations due to varying amounts of observations given by SpaceTrack per satellite. This is also why there are different amounts of satellites included in each dataset; SpaceTrack's MEO satellites have considerably less observations each when compared to LEO and GEO satellites. Thus, it should be noted that in the final results, the GEO class satellites are more likely to present error, since they had the least amount of observations, and the ML architecture will have a bias towards LEO satellites.

It was important to get the amount of observations per orbit class as similar as possible, to prevent bias. Initially, I tried to ensure that the number of satellites per class was the same, but this led the first results to favor LEO satellites and fail when predicting states of MEO or GEO satellites, which were discovered to have more data values compared to MEO and GEO. Additionally, the TLE data was transformed by SGP4 into Cartesian coordinates in order to simplify the prediction process for an ANN, but the original TLE data given by Space-Track didn't have uniform time intervals. This resulted in different amounts of data for each type of orbit. This is why more emphasis was placed on ensuring an equal distribution of observations, not number of satellites, per orbit class.

The TLE data does not directly include the position and velocity vector components which were the predictions or labels for the model. To transform the elements into the position and velocity vector components the SGP4 Python library was utilized. The position and velocity vector components for the given timestamps of TLE data are calculated in kilometers, and kilometers per second in the Earth Centered Inertial Frame (ECI). It is known that as an analytical propagator, SGP4 struggles with accuracy over a long period of time. This study used SGP4 and acknowledges its approximation errors, and keeps them in mind when analyzing the results of the ML model.

Then, the data was preprocessed in order to feed it into the ML algorithm. It was cleaned up of any NaN values and outlier data, and then combined into one dataset of all satellites' TLE. Each type of data, LEO,

MEO, and GEO, was given a different class using the one-hot encoding technique and added on to the dataset. For instance, LEO = 1, MEO = 2, GEO = 3.

Because this Deep Learning Neural Network aims to predict position and velocity vectors based on Keplerian elements at a certain state, timestamps were removed as well and data was shuffled for training and validation purposes. Some of the Keplerian elements are already time-dependent, specifically the mean anomaly, Right Ascension of Ascending Node (RAAN), and the argument of periapsis. Mean anomaly in particular is assumed to increase linearly when predicting across observations, further justifying the removal of timestamps as a feature.

The subsequent dataset used for model training and validation is a combination of all randomly shuffled satellite data, with the Keplerian elements, orbit classifier, and propagated Cartesian coordinates and velocity components in one combined dataframe. Generally, the models taken in the Keplerian elements and orbit classifier as features, or inputs, and the position and velocity components are used to compare with the model's prediction and produce loss and accuracy values. The train-test split used was 70-30, meaning that 70% of data was used for training and the rest was used for validation. The model learns the pattern across 70% of the data, and then it is validated on the unseen 30% to check if it is really learning patterns in the data.

Generally, for the reader's understanding moving forward, this paper proposes six conditional neural networks with teacher forcing. The six models correspond to each of the position (x,y, z) components and velocity components (v_x, v_y, v_z) respectively.

Feature Engineering

Similar to [9], this paper also uses exogenous variables, which are used to predict the values of other variables in the model [12]. In this case, the exogenous variables are the eccentricity, inclination, RAAN, argument of periapsis, mean anomaly, and semi-major axis also referred to as e, i, , ω, M, a respectively. Mean anomaly is used instead of true anomaly to avoid propagating it using Kepler's Equation to ensure simpler preprocessing and less chance of errors, which the propagation with SGP4 has already increased. Additionally, since time stamps are not included and mean anomaly is approximated to increase linearly, it is a better fit for an ANN model which uses a linear activation function. These exogenous variables were coded into features for each one of the six ANN models. In addition to the Keplerian elements, an orbit classifier was included to assist with generalization.

In addition, a learning principle called teacher forcing, typically applied in Recurrent Neural Networks, was applied to improve performance and create stability in subsequent models. Teacher forcing is used in RNNs to provide the previous true prediction the model made as a feature when making the following prediction [13]; this general principle has been applied onto sequential models rather than within one model. Astraeus consists of six stacked models. Each consecutive model takes in the ground truth predicted value from the previous model as an additional feature. To clarify, the Keplerian Elements and

orbit classifier are input into the X model, and the following Y model takes in the same features as well as the true X values. See Figure 1 depicting the first three model inputs to clarify understanding.

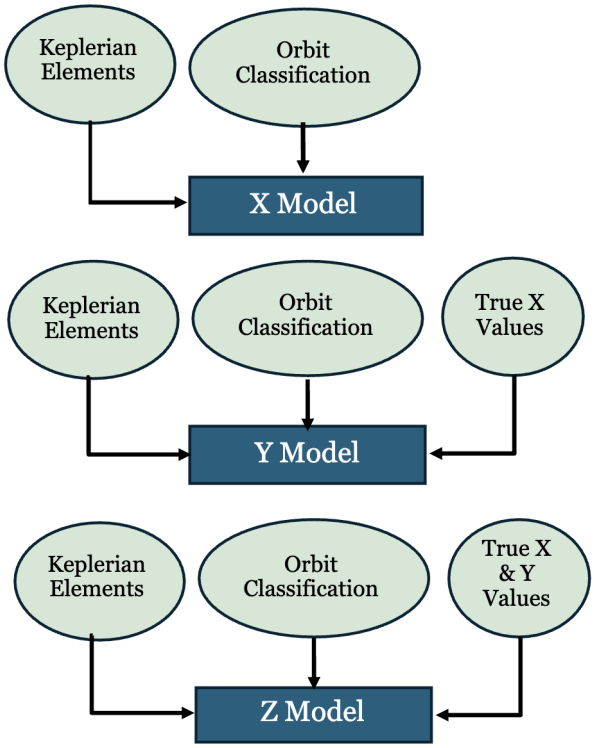

Figure 1: Input Structure of Astraeus

Six ANN Models

Instead of outputting the six components of an orbital state using a single model, this study uses six feed-forward ANN models with conditional neural networks and teacher forcing. Each of the six feed-forward ANN models corresponds respectively to x, y, z, v_x, v_y, and v_z; in other words, the Cartesian coordinates. The x, y, z parameters correspond to the position component while the v_x, v_y, v_z correspond to the velocity component, making up an orbital state.

The models progress like so: x, y, z, v_x, v_y, v_z, according to what each one predicts. Each model consists of fully connected (Dense) layers varying from 128 nodes to 16 nodes, and have either 3 layers or 4 layers. For all six models, a RobustScaler was used to prevent outliers from creating extreme loss on the position vectors. Typically, StandardScalers are used, where

$$Xscaled = \frac{X - \mu}{\sigma}$$

The X is the original value, μ is the mean of the feature, and σ is the standard deviation. It's used for normal datasets. However, the dataset that was fed to the model in this project consists of varying numbers; for instance, the position vectors are in thousands of kilometers, and the velocity is likewise in kilometers per second, but the velocity values are very small, less than 1. In addition, the features are also similar in that e is dimensionless from 0 to 1, inclination, RAAN, argument of perigee, and mean anomaly is in degrees. StandardScaler is also very sensitive to outliers and datasets with changing scales, which is where RobustScaler helped to improve the loss and accuracy using mean and interquartile range (IQR) in addition to scaling the position and velocity vectors separately.

In addition, a dynamic learning rate was used, because one that was too small can slow convergence to the optimal prediction, while a learning rate that's too big can destabilize the model, leading to overshooting. When the default learning rate was used for predicting position vectors in particular, losses began at nine figures and would not come down; considering that the loss used is MSE, that means that the model was approximately off by four or five figures of kilometers. Specifically, ReduceLROnPlateau was used which reduces the learning rate when the model stagnates, preventing overfitting [14]. When using ReduceLROnPlateau in combination with the Robust Scaler and batch size twenty, the loss dropped from 9 figures to 7 figures.

Now, moving on to the overall structure of the ANNs. To demonstrate with the X-Model, the input vector contains the Keplerian elements as well as the orbit classification one-hot encoded:

$$u = [e,\ i,\ a,\ M,\ \Omega,\ \omega,\ c_q]^T$$

Where q = 1, 2, or 3. Thus, there are 7 inputs to begin with, and four hidden layers, before the output layer, comprising one node. Let h_l represent the hidden layers, where l = 1, 2, 3, or 4, W represent the weights matrix, and b represent the bias vector. Thus,

$$h_l = uW_l + b_l$$

Each hidden layer has the ReLu activation function, described as

$$f(x) = max(0,\ x)$$

where x is the input. In this study, the hidden layer outputs

$$h_l = max(0,\ uW_l + b_l)$$

The output layer has a linear activation function, represented mathematically as $f(x) = x$. Therefore, the model outputs

$$O = h_1W + h_2W + h_3W + h_4W$$

Where O is the output, and the weights are calculated from the last hidden layer. As the model has made the prediction, it calculates the error, using the cost/loss function Mean Squared Error (MSE),

$$L = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y_i}),$$

where Y_i is the true value and $\hat{Y}_i$ is the prediction.
The weights are updated as such (as in the standard Stochastic Gradient Descent),

where w is the weight, α is the learning rate, and $\nabla L(w)$ is the gradient of the loss function. α, ReduceLROnPlateau, can be visualised as the following:

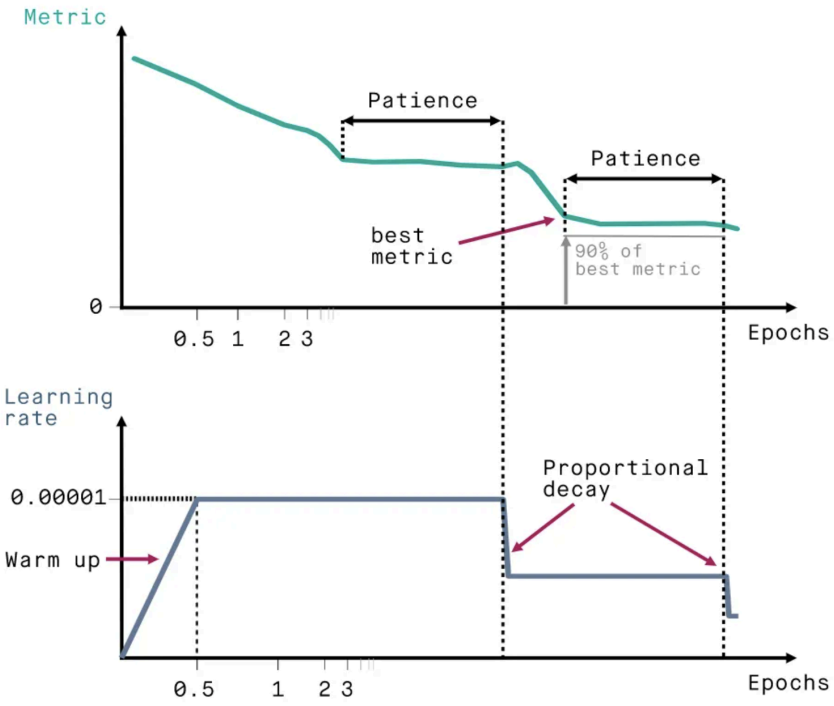

Figure 2: ReduceLROnPlateau Visualization [15]

ReduceLROnPlateau reduces the learning rate when the model stagnates for longer than the patience number allows; so long as the learning rate improves the metric quantity, it will be kept constant. Patience is the number of epochs with no improvement. [15]

The same model structure repeats for the Y Model, Z Model, V_x Model, V_y Model, and V_z Model, with differences only in the number of hidden layers and the addition of features due to teacher forcing. Each model was trained for 400 epochs and split into a validation and training dataset with a 70-30 split.

The final data model pipeline is shown below:

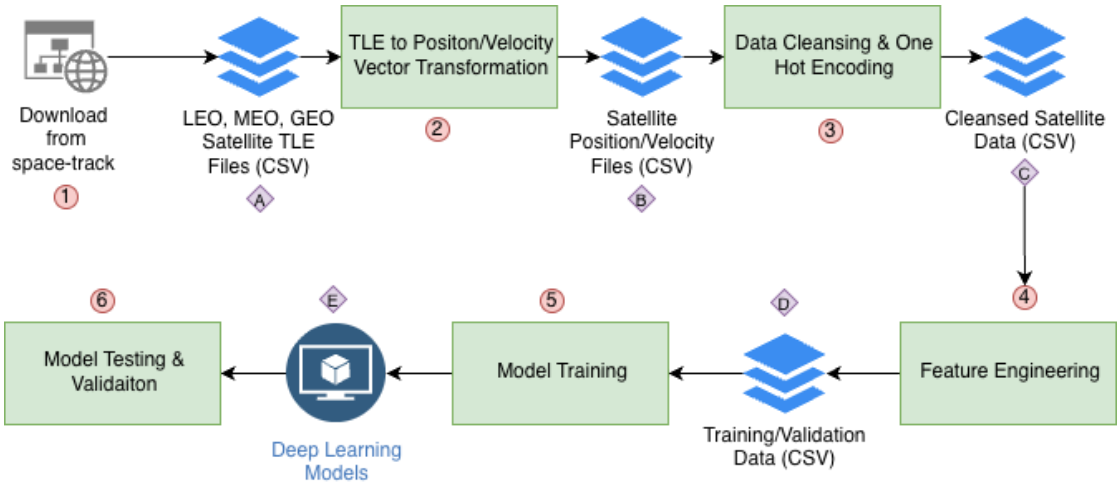

Figure 3: Astraeus Data Model Pipeline

RESULTS

The models' performance during training can be seen here.

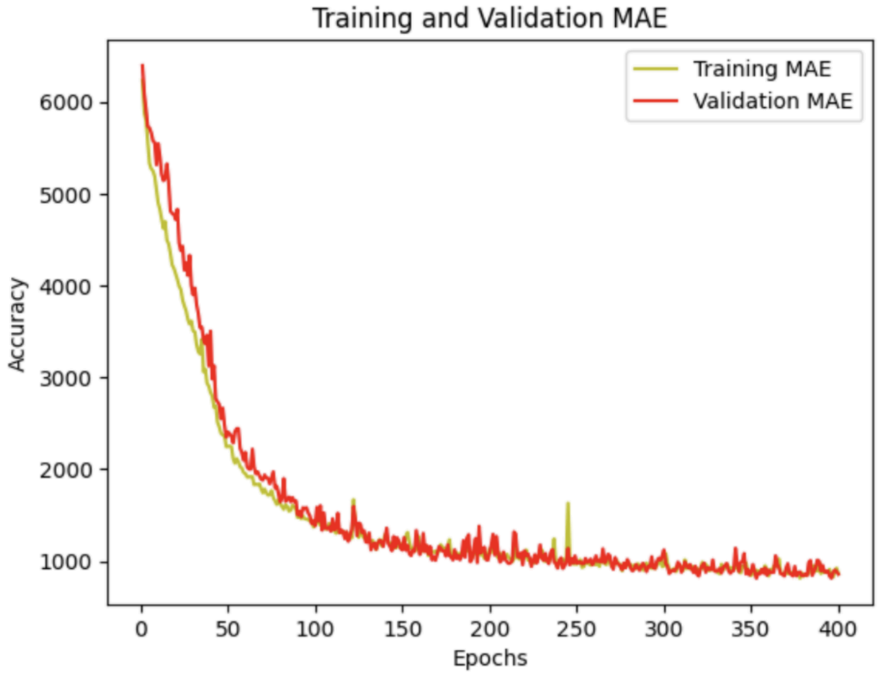

Figure 4: Training and Validation Accuracy Curve for X Model

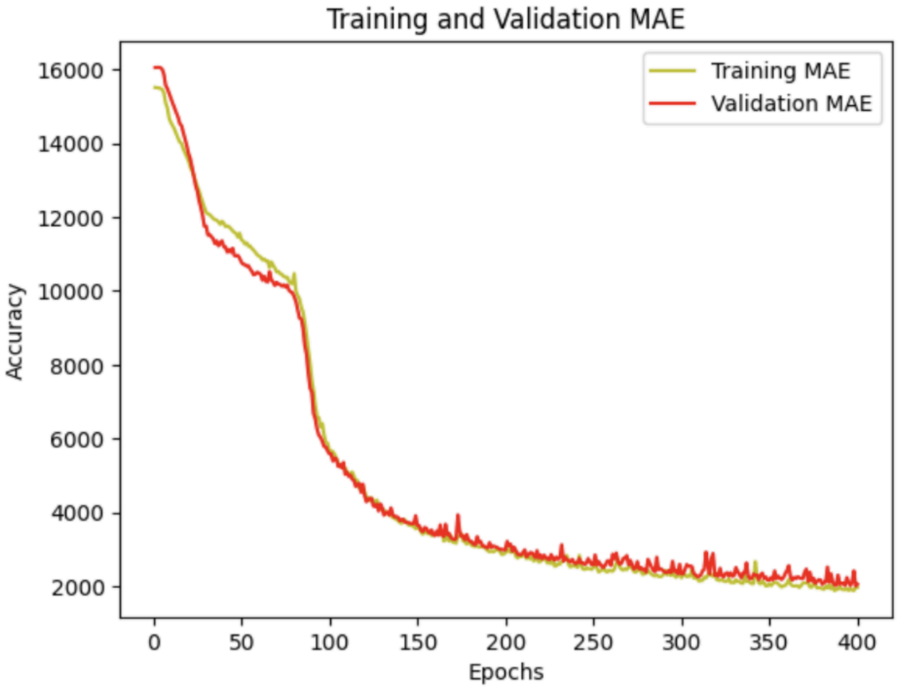

Figure 5: Training and Validation Accuracy Curve for Y Model

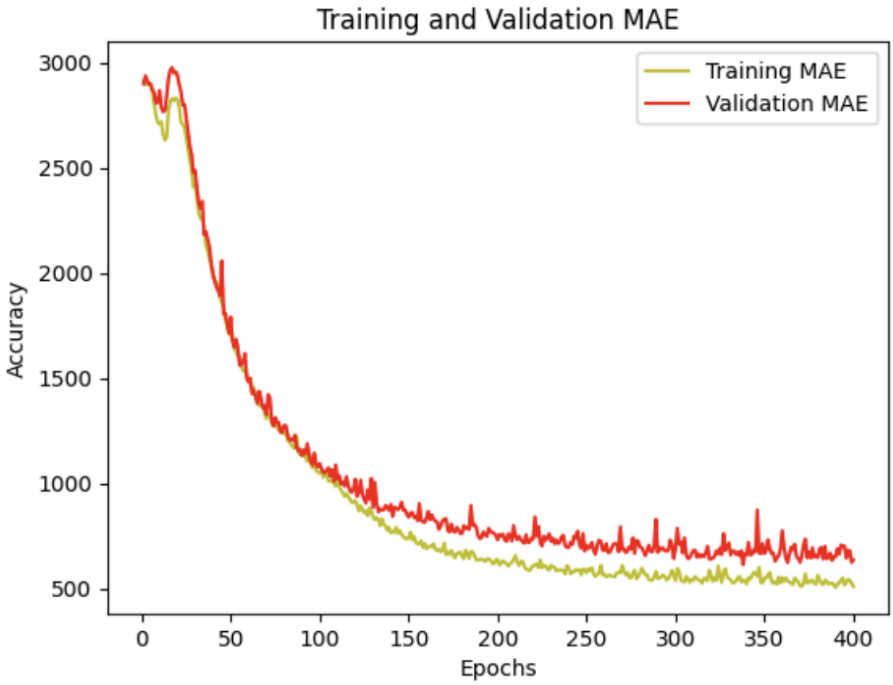

Figure 6: Training and Validation Accuracy Curve for Z Model

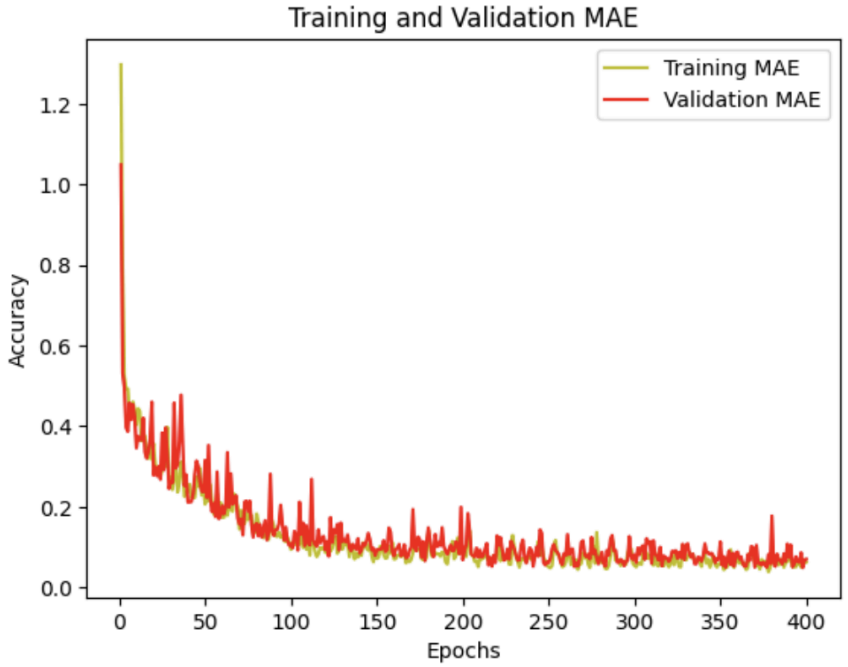

Figure 7: Training and Validation Accuracy Curve for V_x Model

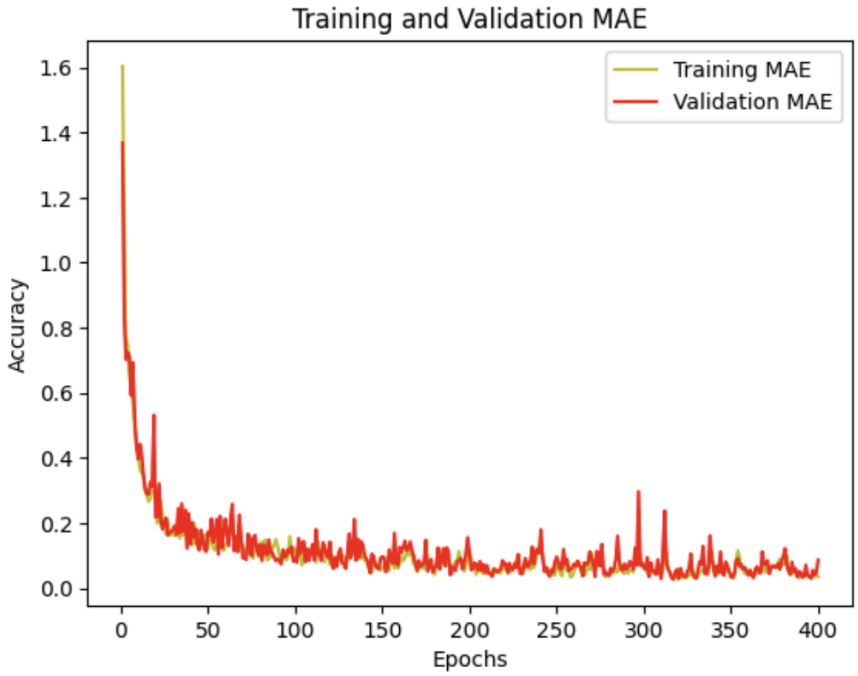

Figure 8: Training and Validation Accuracy Curve for V_y Model

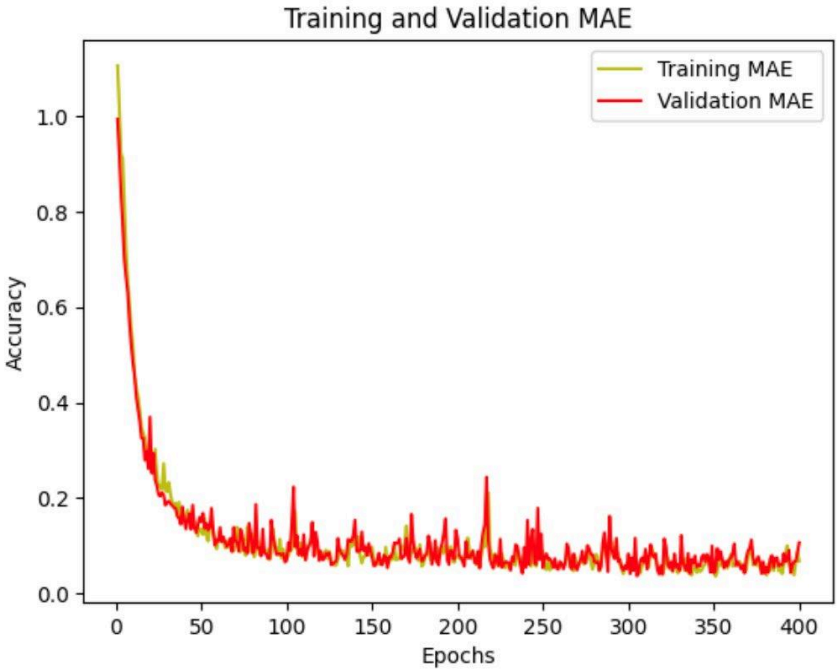

Figure 9: Training and Validation Accuracy Curve for V_z Model

Looking through the graphs of the training sets, the convergence seen throughout the X model until the V_y model demonstrates proper learning generally. Unfortunately, the graphs for the V_z model indicate outliers in the data which are affecting convergence as well as overfitting; despite numerous attempts at standardization and normalization, the model struggles to predict V_z components. Particularly, the position component models show great aptitude, while the velocity components struggle. In terms of scale, the position models are predicted in kilometers, and then due to loss calculations squared to 8 figures; thus, no concern should be presented as part of the capability of the position models. However, with the velocity models, whose values are in kilometers per second and are thus very very small (specifically to the order of 10^-1 or less), such values in loss and accuracy demonstrate limitations of a pure ML method to determine orbital state. This is seen in the following sample predictions made by the trained models on new, unseen data.

Table 2: X Model Sample Values

Predicted X Values	Actual X Values	Residual	RMSE	R^2
309.512329	381.951917	72.439588	475.394	0.998
2555.017822	2166.317128	-388.700694		
-23305.240234	-22390.705220	914.535014		
-953.850098	-1113.611952	-159.761854		
9842.331055	9508.018379	-334.312676		

Table 3: Y Model Sample Values

Predicted Y values	Actual Y Values	Residuals	RMSE	R^2
-7515.877930	-7549.814942	-33.937012	148.789	0.999
14148.458984	14419.315580	270.856596		
41116.113281	41190.200410	-74.087129		
40875.167969	40715.391180	-159.776789		
16463.845703	16392.027990	-71.817713		

Table 4: Z Model Sample Values

Predicted Z Values	Actual Z Values	Residuals	RMSE	R^2
970.626343	1256.441434	285.815091	317.787	0.997
-3224.451416	-3170.733558	53.71785		
-1607.773560	-1509.514188	98.259372		
-2113.797363	-2129.608642	-15.811279		

13675.857422	13035.183690	-640.673732		

Table 5: V_x Model Sample Values

Predicted Vx Values	Actual Vx Values	Residuals	RMSE	R^2
0.706595	0.867611	0.161017	0.220	0.965
0.086211	0.264798	0.178587		
1.415349	1.025873	-0.389476		
-1.969180	-1.787604	0.181576		
-1.448797	-1.420272	0.028525		

Table 6: V_y Model Sample Values

Predicted V_y Values	Actual V_y Values	Residuals	RMSE	R^2
-2.411333	-2.165450	0.245884	0.113	0.995
2.333641	2.310217	-0.023424		
-1.221420	-1.166807	0.054613		
-2.217680	-2.232269	-0.014589		
0.647194	0.649871	0.002677		

Table 7: V_z Model Sample Values

Predicted V_z Values	Actual V_z values	Residuals	RMSE	R^2
0.658077	0.651122	-0.006955	0.014	0.999
0.164243	0.150360	-0.013883		
7.244984	7.249333	0.004349		

7.082108	7.079072	-0.003036		
7.221674	7.249306	0.027632		

Afterwards, the accuracy was calculated for each model using Normalized Mean Accuracy Error (NMAE), as well as Root Mean Squared Error (RMSE) and the R^2 value. The following table demonstrates the values.

Table 8: Overall Analysis of Astraeus' Performance

Model Type	Percent Error	RMSE	R^2
X Model	3.031%	4378.799	0.955
Y Model	3.706%	3919.077	0.955
Z Model	1.372%	3379.282	0.749
V_x Model	4.323%	0.107	0.996
V_y Model	4.807%	0.151	0.995
V_z Model	5.867%	0.607	0.970

CONCLUSION

The model was evaluated based on its ability to make predictions compared to the ground truth values calculated by SGP4. For each of the output components (x, y, z, v_x, v_y, v_z) a separate ANN model was built. Loss was calculated using MSE and the model's accuracy was calculated based on Mean Absolute Error (MAE).

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \hat{Y}_i\right|$$

Errors, like predictions, are in either kilometers, or kilometers per second based on if the model outputs a position vector or a velocity vector.

When the default learning rate was used for predicting position vectors in particular, losses began at nine figures and would not come down; considering that the loss used is MSE, that means that the model was approximately off by four or five figures of kilometers.

In addition to using ReduceLROnPlateau, a Robust Scaler and batch size of 20 was applied. Before implementing these hyperparameters, the model struggled to learn and often began overfitting. After implementing these hyperparameters, accuracy curves converged as is seen in Figures 4 - 9 in the results section. In addition, the loss began at nine figures but reduced to seven figures; thus implying that the model's predictions are off by three or four figures, a far better number that suggests the model's capability for learning.

The model also applies the principle of teacher forcing, often seen in Recurrent Neural Networks. Providing the models actual values of features rather than predicted values of the features helps to stabilize it and reduce error accumulation. In addition, the conditional neural network served to help with model generalization. Before applying it, this contributed to the large loss values observed.

The results have shown that the model possesses capability in predicting the orbital state vectors given Keplerian elements, a classification one hot encoded, and ground truth predicted values as features. In particular, the model shows great promise in predicting the position vector rather than the velocity vector, as is supported by Table 8's reported percent uncertainties. Yet, the abnormally high R^2 values for the velocity could be due to the scale and range of values for velocity; velocity is in kilometers per second, and because the values are abnormally small the produced R^2 is smaller in comparison to the position R^2 values. The NMAE metric is more accurate in this area considering that it accounts for the scale of the data.

However, the model's limitations should be noted. The time period of propagated data was roughly one month per satellite, and the data the model was tested upon was also one month. This means that the

model can predict orbital states across short time periods; it has not been tested on larger datasets. Furthermore, the data the model is using to validate its predictions has been transformed by SGP4, thus, the model's predictions are susceptible to SGP4 driven errors.

In the future, the next steps would be attempting to train the model on chronological train-test split to see if it improves prediction accuracy. Initially, when considering a chronological approach, I opted to use random shuffling instead because half the Keplerian elements were already time dependent, and I wanted the model to be able to learn the dataset properly. The model would have struggled if given multiple sets of chronological datasets of different types of satellite orbits. It was observed that Astraeus would have a hard time converging if I did it in that manner. Yet, a chronological dataset is more relevant when making a model and applying it to real life scenarios.

All in all, this study has found that deep neural networks, specifically ANN, can be used to predict orbital state vectors. Pure ML-models can achieve success and show continuing capability in doing so. This paper in particular demonstrates the aptitude of ANN models to operate on low data under the properly fixed hyperparameters and learning principles. In particular, the transferred use of teacher forcing from RNNs to ANNs can be used in other such models to train them.

REFERENCES

[1] "ESA Space Environment Report 2025."
https://www.esa.int/Space_Safety/Space_Debris/ESA_Space_Environment_Report_2025

[2] "Spacecraft propagators." https://ai-solutions.com/_help_Files/spacecraft_propagators.htm

[3] P. Cefola, J. F. San-Juan, S. Setty, and R. Proulx, "Review of the Draper Semi-Analytical Satellite Theory (DSST)," 2019. [Online]. Available:
https://issfd.org/ISSFD_2019/ISSFD_2019_AIAC18_San-Juan-Juan_F%C3%A9lix.pdf

[4] A. Horstmann, E. Stoll, and Technische Universit¨at Braunschweig, Institute of Space Systems, *INVESTIGATION OF PROPAGATION ACCURACY EFFECTS WITHIN THE MODELING OF SPACE DEBRIS*. ESA Space Debris Office, 2017. [Online]. Available:
https://conference.sdo.esoc.esa.int/proceedings/sdc7/paper/399/SDC7-paper399.pdf

[5] Aida, Saika & Kirschner, Michael. "Accuracy Assessment of SGP4 Orbit Information Conversion into Osculating Elements". 2013

[6] Z. Y.-C. Liu, S. Tarlow, M. Akbar, Q. Donnellan, and D. Senkow, "Improved Orbital Propagator Integrated with SGP4 and Machine Learning," *Proc. Of Small Satellite Conference*, Art. no. SSC21-IX–02, Aug. 2021, doi: 10.26077/aspb-yk57.

[7] K. Selvan, A. Siemuri, F. S. Prol, P. Välisuo, and H. Kuusniemi, "Machine learning for LEO and MEO satellite orbit prediction," *Proceedings of the Satellite Division's International Technical Meeting (Online)/Proceedings of the Satellite Division's International Technical Meeting (CD-ROM)*, pp. 3556–3571, Oct. 2024, doi: 10.33012/2024.19765.

[8] M. S. Mahmud, Z. Zhou, and B. Xu, "LSTM-Driven prediction of orbital parameters for accurate LEO opportunistic navigation," *Proceedings of the Satellite Division's International Technical Meeting (Online)/Proceedings of the Satellite Division's International Technical Meeting (CD-ROM)*, pp. 3542–3555, Oct. 2024, doi: 10.33012/2024.19869.

[9] F. Caldas and C. Soares, "Precise and Efficient Orbit Prediction in LEO with Machine Learning using Exogenous Variables," *arXiv (Cornell University)*, Jul. 2024, doi: 10.48550/arxiv.2407.11026.

[10] J. Saroufim, P. El-Kouba, S. Hayek, and Z. M. Kassas, "Machine Learning-Driven Long-Term Ephemeris error correction for improved LEO PNT," *Proceedings of the Satellite Division's International Technical Meeting (Online)/Proceedings of the Satellite Division's International Technical Meeting (CD-ROM)*, pp. 974–982, Oct. 2025, doi: 10.33012/2025.20470.

[11] J. Varey, J. D. Ruprecht, M. Tierney, and R. Sullenberger, "Physics-Informed neural networks for satellite state estimation," *arXiv (Cornell University)*, Mar. 2024, doi: 10.48550/arxiv.2403.19736.

[12] N. Van Otten, "Endogenous vs Exogenous Variables Explained With Examples & Why It's Important For Machine Learning," *Spot Intelligence*, Oct. 31, 2023. https://spotintelligence.com/2023/04/19/endogenous-exogenous/#Exogenous

[13] R. J. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," in Neural Computation, vol. 1, no. 2, pp. 270-280, June 1989, doi: 10.1162/neco.1989.1.2.270.

[14] R. Banjade, "Learning Rate Schedulers," *Medium*, Jan. 28, 2024. https://medium.com/%40rabin.banjade/learning-rate-schedulers-f940a6ff3b01 (accessed May 31, 2026).

[15] "ReduceLROnPlateau | CloudFactory Computer Vision Wiki," *CloudFactory Computer Vision Wiki*. https://wiki.cloudfactory.com/docs/mp-wiki/scheduler/reducelronplateau

ABOUT THE AUTHOR

Nayana Jayakumar is a junior at Hillsborough High School in Tampa, Florida. She aims to pursue a career in the aerospace industry after getting an engineering and maths degree from university.