NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

KATARINA ELISABETH TOMÉ DYREBY

BSc in Physics Engineering Sciences

PREDICTION OF DEBRIS STATE BASED ON NEURAL ORDINARY DIFFERENTIAL EQUATIONS

MASTER IN BIG DATA ANALYTICS AND ENGINEERING

NOVA University Lisbon
September, 2025

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

PREDICTION OF DEBRIS STATE BASED ON NEURAL ORDINARY DIFFERENTIAL EQUATIONS

KATARINA ELISABETH TOMÉ DYREBY

BSc in Physics Engineering Sciences

Adviser: Cláudia Soares
Assistant Professor, NOVA University Lisbon

Examination Committee

Chair: João Moura Pires
Associate Professor, NOVA University Lisbon

Rapporteur: David Cabecinhas
Assistant Professor, Instituto Superior Técnico

MASTER IN BIG DATA ANALYTICS AND ENGINEERING

NOVA University Lisbon
September, 2025

Prediction of debris state Based on Neural Ordinary Differential Equations

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.29) [35].

Acknowledgements

I would like to begin by thanking my advisor, Cláudia Soares, for being an exceptional teacher. Her guidance helped me grow in many different ways throughout this year, I could not have asked for a better mentor.

The NOVA School of Science and Technology has been a second home to me for the past five years, both within the Computer Science Department and the Physics Department. During this time, I have learned immensely and am deeply grateful for the opportunities I was given.

I would also like to thank my sister, Mónica Dyreby, for being an outstanding role model whose example continually motivates me to strive for more. I hope to follow in her footsteps and make her proud.

Finally, it was at NOVA FCT that I met my closest friends Tiago Teles, Maria Eduarda, Diogo Paulico, Jacinta Sousa, João Bordalo, and David Antunes. Without them I would not be the person I am today, and I will be forever grateful for their friendship and support. I want to thank them for all the good times we had together, and I hope we can keep in touch for many years to come.

”

“Everythin’s got to end sometime. Otherwise, nothing would ever get started.”

— The Doctor, Doctor Who

Abstract

The space environment has become increasingly crowded, with space debris in Low Earth Orbit expected to grow exponentially in the coming years. Alongside the surge in satellite launches, this congestion raises collision risks and highlights the need for scalable avoidance measures to prevent further debris generation. Reliable orbit prediction is therefore essential for ensuring operational safety and prolonging satellite lifespans.

Satellite trajectories are shaped by external forces, some stochastic in nature and thus impossible to model perfectly. This introduces cumulative errors in orbital propagation, further compounded by limited knowledge of satellite properties and simplified force models.

This Master's thesis introduces an approach to Orbit Prediction using *Neural Ordinary Differential Equations*, with *Physics Informed Neural Networks* as a physics informed baseline and a *Multilayer Perceptron* as a non physics informed baseline for comparison. Relying on information about the space environment and the characteristics of satellites, the proposed approach also uses historical trajectory data to propagate a satellite's state vectors. In case of the Neural Ordinary Differential Equations this is done by modeling the underlying differential equation of motion governing the satellite's orbit and using an ordinary differential equation solver to propagate its trajectory forward in time. The results show that this approach effectively combines numerical integration techniques, which provide high accuracy, with Neural Networks, which capture complex patterns and generalize to unseen scenarios. As a result, it outperforms both baseline models by over 100 km, achieving a position mean RMSE of 0.25 km.

By successfully modeling orbit dynamics, this approach offers a high-fidelity solution for Orbit Prediction, ultimately contributing to safer and more sustainable satellite operations.

This thesis has contributed two publications: one presented at the International Astronautical Congress 2025, corresponding to the preliminary material from Chapters 5 to 6, and another at NeurIPS 2025, based on Chapter 4. In addition, a journal paper is in preparation covering the developed material from Chapters 5 to 6.

Keywords: Space debris, Low Earth Orbit, Orbit Prediction, Neural Ordinary Differential Equations

Resumo

As órbitas terrestres têm vindo a tornar-se cada vez mais congestionadas, prevendo-se que a quantidade de detritos em Órbitas Inferiores Terrestres cresçam exponencialmente nos próximos anos. O aumento do número de lançamentos de satélites intensifica este congestionamento e eleva o risco de colisões, reforçando a necessidade de medidas de Prevenção de Colisões escaláveis para evitar a produção de mais detritos. Assim, a previsão rigorosa de órbitas é essencial para garantir a segurança operacional no espaço e prolongar a vida útil dos satélites.

As trajetórias dos satélites são influenciadas por forças externas, algumas de natureza estocástica, o que torna impossível modelá-las de forma perfeita. Isto leva a erros cumulativos na propagação orbital, agravados pelo conhecimento limitado das propriedades individuais dos satélites e pelas aproximações simplificadas das forças externas.

Nesta dissertação de Mestrado, apresenta-se uma abordagem de Previsão de Órbitas utilizando *Neural Ordinary Differential Equations*, tendo como referência comparativa *Physics Informed Neural Networks* como modelo com conhecimento físico e um *Multilayer Perceptron* como modelo puramente baseado em dados. Para além de informação sobre o ambiente espacial, este método recorre também a dados históricos de trajetória para propagar os vetores de estado de satélites. No caso das *Neural Ordinary Differential Equations*, isto é feito com a equação diferencial de movimento que modela a trajoria do satélite e um integrador de equações diferenciais. Os resultados mostram que esta abordagem combina de forma eficaz métodos numéricos, que garantem elevada precisão, com Redes Neuronais, capazes de aprender padrões complexos e generalizar para cenários ainda não observados. Como resultado, supera ambos os modelos de referência em mais de 100 km, alcançando um RMSE de posição de 0.25 km.

Ao conseguir modelar as dinâmicas orbitais de forma mais precisa, esta abordagem oferece uma solução para a previsão de órbitas, contribuindo para operações de satélites mais seguras e sustentáveis.

Esta tese contribuiu com duas publicações: uma apresentada no International Astronautical Congress 2025, que correspondente ao material preliminar dos Capítulos 5 a 6, e outra na NeurIPS 2025, baseada no Capítulo 4. Além disso, está em preparação um artigo

que reúne o material dos Capítulos 5 a 6.

Palavras-chave: Detritos Espaciais, Órbitas Inferiores Terrestres, Previsão de Órbitas, Neural Ordinary Differential Equations

Contents

List of Figures

List of Tables

Acronyms

CAMs Collision Avoidance Manoeuvres *(pp. 1, 2)*
CDMs Conjunction Data Messages *(pp. 1, 2)*
CSpOC Combined Space Operations Center *(p. 2)*

ECI Earth-Centered Inertial *(p. 22)*
ESA European Space Agency *(pp. 1, 2, 10)*

GNSS Global Navigation Satellite System *(p. 1)*

J2000 Julian year 2000 *(p. 22)*

LEO Low Earth Orbit *(pp. 1–4, 7, 25, 30, 32, 44)*

ML Machine Learning *(pp. 4, 8–10)*
MLP Multilayer Perceptron *(pp. 5, 34, 37, 44–46, 49–60, 63)*
MSE Mean Squared Error *(pp. 5, 39, 42)*

NN Neural Network *(pp. 4, 9, 10, 12, 16, 37, 40–42, 44, 49, 51, 63)*
NODE Neural Ordinary Differential Equation *(pp. 4, 6, 10, 13, 15–19, 22, 32, 33, 37, 40–42, 44, 46, 48–51, 55, 56, 59–64)*
NORAD North American Aerospace Defense Command *(pp. 23, 25)*

ODE Ordinary Differential Equation *(pp. 10, 12–18, 34, 38–44, 46, 49, 50, 58, 59, 63)*
OP Orbit Propagation *(pp. 4, 6–8, 10, 32)*

PINN Physics Informed Neural Network *(pp. 5, 6, 9, 10, 19, 20, 22, 32–34, 37, 39, 40, 44, 46, 48–51, 53–60, 63, 64)*
PoC Probability of Collision *(p. 2)*

ResNet Residual Neural Network *(pp. 12, 13, 16, 18)*
RMSE Root Mean Square Error *(pp. 9, 34, 45, 46, 49–57, 59, 61, 62)*
RNN Recurrent Neural Network *(pp. 16, 17)*
RSO Resident Space Object *(pp. 2–4, 7, 8, 10, 32)*
RTN Radial-Transverse-Normal *(p. 22)*

SDP Simplified Deep Space Perturbations *(p. 7)*
SGP Simplified General Perturbations *(p. 7)*
SGP4 Simplified General Perturbations 4 *(pp. 7, 8, 10, 11)*
SINDy Sparse Identification of Nonlinear Dynamics *(pp. 9, 18, 63)*

TCA Time of Closest Approach *(pp. 1, 2)*
TLE Two-Line Element set *(pp. 7, 8, 10)*

USSC United States Space Command *(pp. 22, 23)*

1

Motivation and Problem Statement

Artificial satellites have become integral to modern life, enabling global communication, Global Navigation Satellite System (GNSS), accurate weather forecasting and many areas of scientific research. The increasing reliance on artificial satellites means that their safe and continuous operation is critical for maintaining these services. However, the rapid rise in satellite deployments has led to overcrowded orbits and heightened collision risks.

The first artificial satellite, Sputnik, was launched into Earth's orbit by the Soviet Union in 1957 with the aim to gather information about the atmosphere's density and test methods for orbital tracking. Since then, satellite launches have become more frequent. In 1978, *Kessler et al.* proposed a scenario, now known as the Kessler syndrome, where collisions between objects in space leads to a chain reaction — each collision generates more space debris, which subsequently raises the likelihood of further collisions [29]. As a result, the density of objects in the Low Earth Orbit (LEO)[1] grows exponentially, as seen in Figure 1.1, potentially rendering certain orbits unusable for satellites.

A significant rise of space debris in 2009, shown in Figure 1.1, followed the first in-orbit collision between two satellites, which alone created over 2300 fragments. Currently, with an estimated 1,036,500 objects larger than one centimeter in Low Earth Orbit, the reality of Kessler's scenario appears more plausible [20]. These objects travel at speeds of approximately 7.8 km/s, which is roughly nine times faster than a bullet. Consequently, even small pieces of space debris possess enough kinetic energy to cause significant damage to satellites.

With the increasing likelihood of collisions, accurate orbit and collision prediction is essential to control the buildup of space debris in LEO and ensure the safe operation of satellites. Satellite operator like the European Space Agency (ESA) covers the collision prediction, and Collision Avoidance Manoeuvres (CAMs) if needed, for several satellites in LEO. Whenever a collision is deemed possible between a Target (satellite) and a Chaser (space debris or another satellite), a conjunction alert is issued and Conjunction Data Messages (CDMs) are sent 7 days before the possible Time of Closest Approach (TCA) [41].

[1] Lower Earth Orbit extends up to 2000 km above Earth's surface

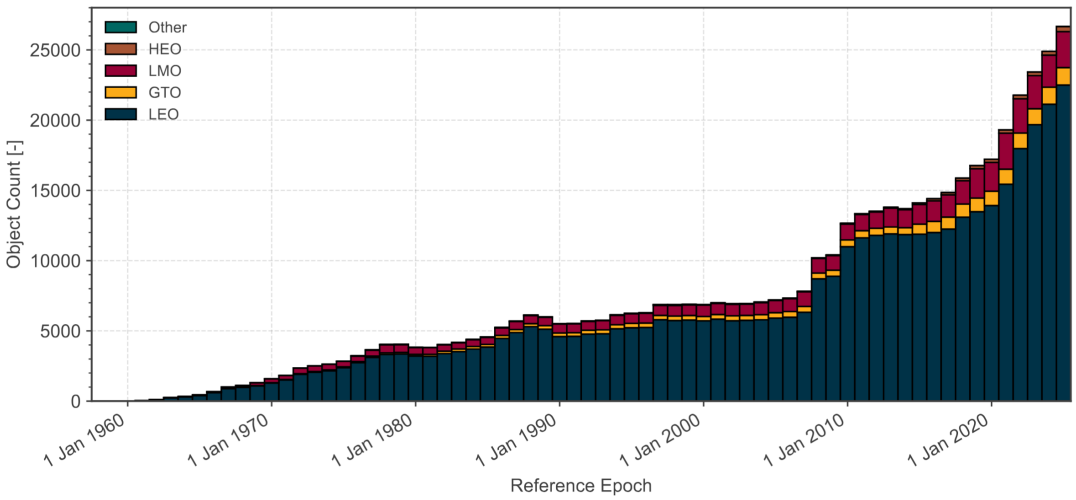


Figure 1.1: Evolution of the number of objects in Earth's orbits over the years [18]. Notably, there is a significant increase from 2009 onwards due to a collision between two satellites.

These messages, provided by the Combined Space Operations Center (CSpOC), include TCA, the position and velocity uncertainties of the objects and more. However, the CDMs are sent up to three times a day as the data about the collision is updated and becomes more accurate. This means an event could be classified as dangerous (or non-dangerous) on the last day before the closest approach. Consequently, CAMs are sometimes executed even when there was no real threat, wasting fuel and diminishing the satellite's lifespan[41].

The satellite operator then processes this data to estimate the Probability of Collision (PoC) by using algorithms such as the Alfriend–Akella algorithm [1] [41] [21] [8]. If the PoC exceeds a certain threshold (e.g., 10^{-5}), CAMs are considered. As a result, experts are required to be available at all times to investigate cases where, after processing and filtering conjunction alerts, a professional analysis is necessary to confirm the collision [4]. On average, ESA conducts two CAMs per satellite each year. This dependence on constant expert oversight reveals a vulnerability, as it does not scale efficiently with the exponential rise of objects in LEO.

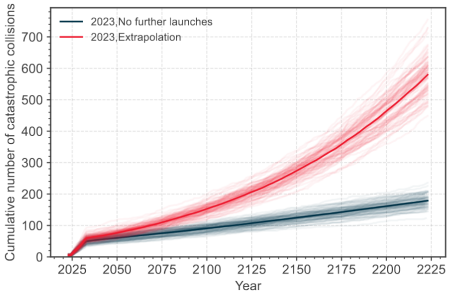
A Resident Space Object (RSO)'s equation of motion is influenced by a variety of forces Atmospheric Drag and Solar Radiation Pressure being the most relevant non-conservative and non-gravitational forces [66] [7].

Atmospheric Drag is a resistive force that opposes an object's motion, significantly affecting the trajectory of objects in LEO. Although the atmospheric density in LEO is much lower than at the Earth's surface, it still exerts a substantial force on orbiting objects. This resistive force acts as a natural cleaner of space debris, causing them to slowly lose velocity and gradually decay, eventually leaving orbit and re-entering Earth's atmosphere. The three-dimensional acceleration exerted by drag on a RSO, $\vec{a}_{\text{drag}} \in \mathbb{R}^3$ can be calculated using:

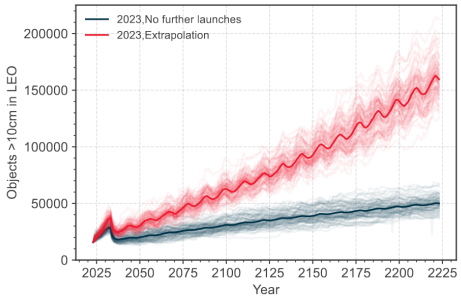
$$\vec{a}_{\text{drag}} = -\frac{1}{2}\rho C_D \frac{A}{m}\|\vec{v}\|\vec{v} \tag{1.1}$$

where ρ is the atmospheric density, $\vec{v}$ is the three-dimensional velocity vector of the RSO, A is the reference area of the satellite, m is the mass of the satellite and C_D is the drag coefficient. Drag has the effect of an RSO, that, in turn, loses energy and altitude, until it falls through Earth's atmosphere, burning while precipitating towards the surface of the planet. This process is call deorbiting.

Unfortunately, given the rapid accumulation of space debris in LEO, relying solely on atmospheric drag to de-orbit RSOs is not a viable long-term solution. As shown in Figures 1.2a and 1.2b, even if no more satellite launches were conducted from 2025 onwards, the amount of debris would still increase, as would the number of collisions.



(a) Projected number of catastrophic collisions in LEO: Red indicates if satellite launches continue at the current rate, and grey indicates if no further launches are conducted since 2023 [18].



(b) Projected number of objects (larger than 10cm) in LEO: Red indicates if satellite launches continue at the current rate, and grey indicates if no further launches are conducted since 2023 [18].

Figure 1.2: Projections related to LEO: (a) Number of catastrophic collisions, and (b) Number of objects larger than 10 cm.

Solar Radiation increases the amount of energy deposited into Earth's upper atmosphere [43]. The absorption of high-energy UV and X-rays raises the kinetic energy of atmospheric particles, raising the temperature as well. As a result, particles move with greater velocity, causing the Thermosphere to expand. This expansion leads to a higher density of particles in LEO (greater atmospheric density). The increased density, ρ, increases the Atmospheric Drag, the absolute value of Equation (1.1). Despite the fact that the sun goes through an 11-year cycle, with periods of high and low activity, the effect is still stochastic because the radiation emitted by the Sun varies throughout the cycles.

Solar Radiation Pressure pushes the RSO away from the sun due to the exchange of momentum between the flux of incident photons emitted by the Sun and the object's surface [66]. The acceleration gained by the object, $\vec{a}_{srp}$ depends on the rate of absorption and reflection of these photons, which is determined by the properties of the object's surface, and the intensity of solar activity. This acceleration is a three-dimensional vector defined by:

$$\vec{a}_{\text{srp}} = -\frac{P_0 \cdot A \cdot C_r}{m} \left(\frac{\text{AU}}{\|\vec{r}_{\text{sat-sun}}\|} \right)^2 \hat{r}_{\text{sat-sun}}, \tag{1.2}$$

where C_r is the reflectivity coefficient, P_0 is the solar radiation pressure at 1 AU (4.56×10^{-6} N/m^2), A is the cross-sectional area of the satellite, m is the mass of the satellite, $\vec{r}_{\text{sat-sun}}$ is the vector from the satellite to the Sun, and $\hat{r}_{\text{sat-sun}}$ is the unit vector in that direction.

The Gravitational Forces between RSOs and nearby celestial bodies (the Earth, the Moon, and the Sun) play a significant role in its motion as conservative forces [68]. Adjustments to these forces account for the fact that celestial bodies are not perfectly spherical and that the Earth's tides cause deformations on the Earths surface. Nevertheless, these huge bodies of mass pull the object towards them, effectively deviating it from its original orbital path.

These forces cause disruptions to the motion of satellites in LEO. Consequently, satellites are occasionally required to fire propulsion systems to maintain their intended orbits, or to avoid collisions. By knowing the forces that act on a RSO, its acceleration can be calculated. Subsequently, its velocity and position can be modeled by solving differential equations. However, due to the complexity of accurately calculating all the forces involved, models that predict a RSO's position in LEO typically include only the most significant forces: Atmospheric Drag and Solar Radiation pressure yielding the law of motion [7].

$$\frac{d^2\vec{r}}{dt^2} = -\frac{\mu\vec{r}}{\|\vec{r}\|^3} + \vec{a}_{\text{drag}} + \vec{a}_{\text{SRP}}, \tag{1.3}$$

where μ is the Earth's gravitational parameter, $\vec{r}$ is the position vector of the RSO, $\vec{a}_{\text{drag}}$ is the acceleration due to atmospheric drag, and $\vec{a}_{\text{SRP}}$ is the acceleration due to solar radiation pressure.

Errors in modelling these forces can easily propagate, leading to significant inaccuracies of a RSO's trajectory over time. Therefore, more precise modeling of these forces is mandatory for accurate orbit and collision predictions.

Contributions. This work focuses on improving Orbit Propagation (OP) accuracy by modeling the coefficients of the forces that affect a satellite in orbit while propagating it in time. A Neural Ordinary Differential Equation (NODE) is going to be used to model the differential equation of motion of a RSO (in (1.3)), allowing for predictions that generalize across various LEO orbits and objects, while adhering to the laws of physics [10].

NODEs are a type of Neural Network (NN) that combines Machine Learning (ML) principles with differential equations, incorporating the laws of physics as prior information about the system. This allows NODEs to learn system dynamics from data in a robust and physics abiding manner. They model a system's differential equation, allowing for smooth predictions, and can be trained on irregularly sampled data, which is non-trivial with conventional methods. Solving the differential equation of motion with the learned parameters over time allows the model to propagate a satellite forward in its orbit, providing reliable orbit predictions.

The model was trained on realistic synthetic data generated by the Orekit library. Its performance was evaluated using Mean Squared Error (MSE) and compared to a Physics Informed Neural Network (PINN), which serves as a physics baseline, and a Multilayer Perceptron (MLP), serving as a data only baseline. The initial plan was to use Starlink data, but due mostly to inaccuracies in propagator data (Section 4.7), the Orekit library was chosen as the source of data. Only after comparing the models was the Starlink data used to test the model's capacity to learn under real-world conditions.

2

Related Work

This section reviews the most commonly used methods in Orbit Propagation (OP). It begins with a discussion of traditional methods, including numerical, analytical and hybrid approaches, and the challenges they face. Physics Informed Neural Networks (PINNs) and Neural Ordinary Differential Equations (NODEs) are then introduced as promising machine learning techniques for OP.

2.1 Numerical Methods

The equation of motion (1.3) governing a satellite in orbit is a second-order differential equation. By converting it into a system of first-order differential equations, as shown in Equation (2.1), and solving them given initial conditions, we can obtain the satellite's position at any given time, allowing us trace satellite's trajectory.

$$\begin{cases} \frac{d\vec{r}}{dt} = \vec{v} \\ \frac{d\vec{v}}{dt} = -\mu \frac{\vec{r}}{\|\vec{r}\|^3} + \vec{a}_{\text{drag}} + \vec{a}_{\text{SRP}} \end{cases} . \tag{2.1}$$

The most commonly used approach for approximating the solution of this system of differential equations are numerical methods. These methods are widely applied to OP due to their accuracy. However, they can be computationally expensive and time-consuming, particularly for long-term predictions.

The study of ordinary differential equations is a well-established field, with numerous numerical methods available for approximating their solutions. Some of these methods have been specifically developed for OP [42][28][5].

Numerical methods fall under two categories: single-step methods, such as explicit Runge-Kutta and Dormand-Prince, and multi-step predictor-corrector methods, such as Adams-Bashforth and Gauss-Jackson. The latter has been in use by United States' space surveillance centers since the 1960s [5][3][7][28]. They can use either fixed or variable step sizes, except for the Gauss-Jackson method, which only supports a fixed step size. Fixed step sizes stay constant throughout the integration process, whereas variable step

sizes adjust dynamically based on the integration accuracy needs, potentially reducing the computational cost.

Numerical solvers can also be classified as either explicit or implicit. Explicit solvers rely only on the current state $y(t)$ to compute the next state $y(t + \lambda)$. In contrast, implicit solvers require both the current state $y(t)$ and information about the next state's gradient in order to compute the value $y(t + \lambda)$. This means that implicit methods are slower per step because the solver must solve a non-linear equation to compute the next step. However, implicit methods are more stable than explicit ones, allowing them to use larger step sizes while maintaining accuracy [23]. To improve OP and computational efficiency, A-stable[1] and parallelizable methods have been developed [3].

Numerical methods for OP are precise but time-consuming. Despite advancements, long computation times remain a major drawback. Given the need for rapid decision-making, especially in collision avoidance, these methods are not well-suited for the rapidly expanding space environment.

2.2 Analytical methods

Analytical methods are commonly used in Orbit Propagation, with Simplified General Perturbations 4 (SGP4) introduced in 1970 by Cranford, being the most widely adopted [27]. SGP4 is a space propagation model that predicts a Resident Space Objects (RSOs) future orbital positions based on its current orbital state vectors (position and velocity). It does this by using a simplified representation of the space environment and the forces acting on the RSO to propagate those state vectors forward in time.

SGP4 is designed for satellites in lower altitudes, while satellites in higher orbits (with orbital periods longer than 225 minutes) use Simplified Deep Space Perturbations (SDP) models. The key difference between these models lies in the space environment considered during propagation. For instance, in Low Earth Orbit (LEO), atmospheric drag has a much greater influence on a satellite's motion than solar radiation pressure due to the high velocities at which these satellites travel. At these speeds, the effects of solar radiation are overshadowed by the atmospheric forces acting on the satellite [36]. However, as altitude increases, the atmosphere becomes progressively rarefied, reducing the impact of drag. In contrast, solar radiation pressure becomes more significant.

Two-Line Element sets (TLEs), which are made publicly available by Space-Track, contribute to the widespread use of all Simplified General Perturbations (SGP) methods. These TLEs are specifically formatted for SGP models and provide the mean orbital elements RSOs [27]. The mean values in TLEs have been processed to remove periodic variations in the force model therefore, to maintain accuracy, these same variations must then be reconstructed in exactly the same way they were removed. Using a different model

[1] A-stable numerical integrators have a bigger stability zone making them particularly well-suited for stiff problems.

can lead to less accurate results because it will handle periodic variations differently [27] [61].

Over the years, different versions of SGP4 have been released, each offering slight modifications to the propagator. These changes can lead to discrepancies in predictions, as each version will handle certain aspects of orbit propagation differently, creating compatibility issues. Since TLE data does not reveal which version of SGP4 was used to estimate the mean orbital parameters and remove periodic variations, users might be unknowingly working with an outdated or modified version of SGP4, which could yield results that differ from the original model's intended predictions [61]. To mitigate this, efforts have been made to provide official versions as part of a software library [13].

Recently, SGP4-XP, an updated version of SGP4, was introduced to better capture the dynamics of satellite orbits. This version includes improved lunar perturbation modeling and solar radiation pressure modeling across all orbits, while maintaining nearly the same speed as SGP4. These changes have resulted in more accurate predictions [13] [46].

While the SGP4 propagators are fast, their accuracy tends to degrade, especially at lower altitudes due to the increased Atmospheric Drag. Additionally, their accuracy decreases in long-term predictions due to the accumulation of errors over time [63]. This is exacerbated by the fact that SGP4 does not account for satellite thrust, despite the majority of space missions use thrust for orbit maintenance [61]. This, combined with the dependence on modeling the space environment and the characteristics of the RSO, limits the accuracy and reliability of the predictions.

To address these limitations, hybrid methods have emerged to mitigate errors from propagators using Machine Learning (ML). Peng et al, have improved propagators by using Artificial Neural Networks, Support Vector Machines, and Gaussian Processes [48][50] [49][47]. These methods aim to improve the SGP4 propagator by using Machine Learning models to correct inherent prediction errors, rather than directly forecasting the future state of the RSO. This eliminates the need to explicitly model the RSO accurately, for example its shape and area-to-mass ratio.

Similarly, a hybrid approach using SGP4 with an autoencoder and random forest was used to reduce distance errors in SGP4 propagation, along with other related studies [34][54][55].

This hybrid approach has demonstrated promising results in reducing errors in SGP4 predictions, however, it has higher computational costs compared to using the propagator alone, and is restricted to only using the specific input format required by the SGP4 model.

2.3 Physics Informed Machine Learning

Recent advancements on OP methods have seen a shift away from the SGP4 propagator altogether due to Machine Learning's growing popularity. Unlike Numerical or Analytical Methods, which require the explicit knowledge of the forces acting on the RSOs, Physics Informed Machine Learning models learn to infer this information directly from the data.

These models incorporate the underlying physics into their ML architectures to better model nonlinear dynamic systems.

Manzi et al. applied Symbolic Regression with Genetic Programming to discover the unknown drag component in orbital motion [39]. To simplify the model, they assumed drag was dependent on position and velocity rather than time. On noiseless data, their model found an expression that matched the exact missing drag component with an error of around 0.2%. However, Genetic Programming-based Symbolic Regression is computationally expensive and prone to overfitting. In a later work, Manzi et al. combined Genetic Programming based Symbolic Regression with Sparse Identification of Nonlinear Dynamics (SINDy) to address these issues [40].

Dawoodjee and Rajeswari [15] used Non-Linear Regression, a supervised ML model, for orbit prediction. They modeled the x coordinate of the satellite's position using a sinusoidal wave with a linear trend, fitting the data with a non-linear least squares function, extending this approach to the remaining coordinates and their respective velocities. Their method required approximately five days of historical data to predict satellite positions three days in advance with Root Mean Square Error (RMSE) of around 800 km for the x and y coordinates and approximately 300 km for the z coordinate.

SINDy has also been applied to uncover the underlying equation of motion of satellites in space by correctly modeling terms in the equation. The paper *Finding Real-World Orbital Motion Laws from Data* [24] demonstrates this by identifying the standard gravitational parameter of Earth using data and knowledge of orbital physics. It does so by using SINDy with a physics-based library of basis functions and a polynomial library. The physics-based library provided significantly better results, highlighting the importance of incorporating domain knowledge in such tasks. The findings were validated by propagating a satellite's trajectory and calculating the prediction errors using a realistic high-fidelity propagator. The results showed low errors for all positional predictions.

2.4 Physics Informed Neural Networks

Physics Informed Neural Networks, introduced by Lagaris et al. and popularized by Raissi et al. in 2019 are a specialized type of Neural Network (NN) that incorporate the dynamics of the physical process governing the data into their training [31][51]. However, the features learned by PINNs are not directly relatable to physical concepts, making them *black boxes*.

PINNs have gained popularity across a wide range of problems governed by differential equations. Additionally, since they require little data to converge to a solution, PINNs are well suited for fields where data is scarce or expensive to obtain. They work well with noisy data, as the physics-based constraints incorporated into their training help regularize the model's predictions. Furthermore, PINNs adapt to new data, making them

well-suited in environments where conditions fluctuate due to external factors. Such is the case with the trajectory of a RSO. See Section 3.2 for more details on PINNs.

Varey et al. [62] compared the performance of a traditional physics-based model and a PINN when propagating the orbit of a satellite with a thrust profile. By using PINNs, the model focuses on capturing deviations from the physical model using observed data by treating the unknown thrust component as a parameter to be learned by the NN. This approach closely resembles a NODE, as it does not incorporate a physics-based loss directly into the loss function. Instead, the physics is embedded within the NN and used as the function optimized during the forward pass. The network's outputs are passed to an Ordinary Differential Equation (ODE) solver, which computes the position and velocity. One day after the observations, the physics-based model had an error of 397 km, while the PINN had an error of 3.35 km, once again proving the benefits of using ML for OP.

2.5 Neural Ordinary Differential Equations

Neural Ordinary Differential Equations is a type of model that learns system dynamics by numerically solving an ODE during the forward pass, with the underlying function represented by a Neural Network. This allows them to model complex systems with high accuracy, even when the underlying dynamics are unknown, the data is noisy, and observation are sparse. For a detailed background on NODEs see Section 3.1. Recently, the European Space Agency (ESA) has recognized the potential of NODEs to address various space-related challenges [19].

One research team has taken the first steps toward applying NODEs to OP, using the SGP4 propagator as a baseline for comparison and TLE data as ground truth.In their approach, the NODE learns deviations from a simple gravitational model by incorporating unknown parameter through the Neural Network inside the ODE. As a result, the Neural Network only learns relatively small deviations from the main gravitational force, rather than having to relearn the entire acceleration profile from scratch. Additionally, they use SGP4 for data augmentation, generating synthetic data between TLE observations by propagating the satellite's trajectory [59].

The researchers also train multiple satellites simultaneously by treating the NODE as time-invariant, meaning the learned ODE does not explicitly depend on time. However, batch training is still possible without enforcing time invariance, provided that the start and end times, as well as the step sizes of observations, are the same. Furthermore, because the space environment varies across different regions, batch training should only be applied to satellites in similar orbital conditions, such as those with close altitude and inclination, which could be grouped into clusters accordingly. Doing batch training on satellites with different orbital conditions could lead to the NODE learning a single ODE that does not accurately represent any of the satellites in the batch.

Their results show that for the first three days of propagation, the NODE outperforms SGP4. However, beyond this period, the NODEs accuracy deteriorates, becoming less

precise than SGP4—even though SGP4 itself is known to accumulate significant errors over longer-term predictions.

3

Background

3.1 Neural Ordinary Differential Equations

Neural Ordinary Differential Equations are a general framework for incorporating ODE Solvers into a Neural Network architecture. They introduce a family of Neural Networks that are built upon the Residual Neural Network (ResNet) architecture, due to its stable gradient propagation for very deep models.

As the need for deeper networks increased, to allow the modeling of more complex functions and more complex features, issues like the vanishing and exploding gradients arose.

- Vanishing gradients occur when the gradients used during backpropagation become too small. This results in minimal updates to the network weights. As a result, adding more layers doesn't improve model performance since the gradients fail to propagate through the layers.
- Exploding gradients arise when the gradients become excessively large during backpropagation. This causes the weights to update too drastically, destabilizing the learning process. Similar to the vanishing gradient issue, adding more layers under these conditions won't better model performance, as it leads to unstable training.

ResNets were designed to tackle these issues through the use of *skip connections*, which allow layers to learn incremental changes necessary to adjust the input towards the desired output. The ResNet output is:

$$y = F(x) + x, \tag{3.1}$$

where, $F(x)$ represents the residual of a stack of layers computed by the NN, x represents the input to the stack of layers and y represents the output of the stack of layers. A skip connection is a shortcut that allows the input of a layer to bypass one or more intermediate layers and be directly added to the output. This means the network does not learn y directly but rather learns the correction $F(x)$ that modifies x.Therefore, by placing a

skip connection, the gradient can flow directly to the earlier layers even if the gradient becomes small in the previous layers. The gradient of the skip connection provides a direct contribution from the final layers, successfully avoiding the vanishing or exploding gradients problem and allowing the training of deeper networks.

Chen et al. [10] re-developed NODEs, interpreting the structure of a ResNet as a simple ODE Solver. In a ResNet, each layer's output is an iterative update from the previous layer, akin to one step in the Euler integration method, a simple ODE Solver:

$$y_{t+1} = y_t + \lambda f(t, y_t), \tag{3.2}$$

where y_{t+1} is the estimated value of the function in the next step, y_t is the value of the function in the current step, $f(t, y_t)$ is the gradient of the function at the current step and λ is the step size. Euler's method uses small discrete updates according to the direction of the gradient at the current position. By doing this process iteratively it arrives at the solution of the differential equation. While Euler's method solves differential equations, it is known for accumulating errors with each step. Since it was developed, much more robust solvers have been created.

By taking smaller steps (infinitesimally small) between layers while adding more layers, in the limit, we end up with a differential equation. This provides a continuous solution instead of a discrete approximation of the solution:

$$\frac{dy(t)}{dt} = F(y(t), t). \tag{3.3}$$

NODEs model an ordinary differential equation $F(y(t), t, \theta)$, as a Neural Network with parameters θ, by treating the network as a continuous transformation of the input over time. In other words, the Neural Network in NODEs define the vector field of the solution's gradient at any given point in time, see Image 3.1. A differential equation solver (such as Euler's method) approximates the solution, with each step requiring gradient information from the neural network. More recent ODE Solvers provide users with costumization, for example, controlling the numerical errors allowed during the integration process. Essentially, an ODE Solver approximates the integral over the Ordinary Differential Equation to find the solution:

$$\text{ODESolve}(y(t_0), F, t_0, t_1, \theta) \simeq y(t_0) + \int_{t_0}^{t_1} F(y(t), t, \theta)\, dt. \tag{3.4}$$

The parameters θ of the Neural Network $F(y(t), t, \theta)$ are not trained using the standard backprogation method typically used in traditional Neural Network. This is because numerical errors accumulate during the forward pass through the ODE Solver. Instead, the adjoint sensitivity method is used to compute gradients, treating the ODE Solver as a *black box*. This method involves solving a second ODE backward in time to calculate the gradients required for the loss function. By avoiding the need to store intermediate

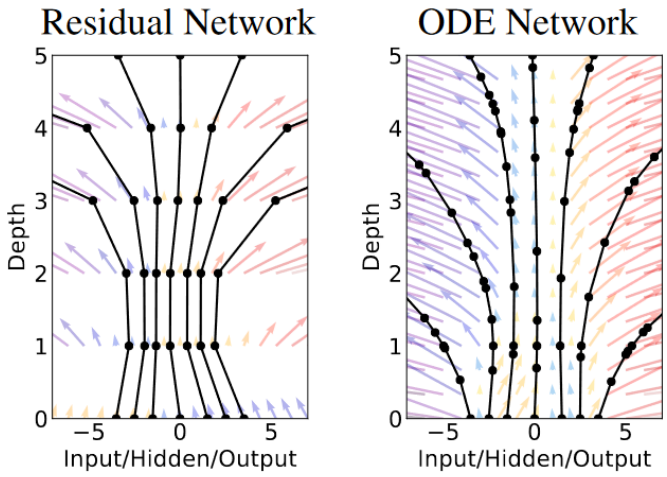


Figure 3.1: Visual example of the difference between a Residual Network and an ODE Network. In a Residual Network, the output of each layer is an iterative update from the previous layer. The gradient vector field of the solution is, by definition, discrete, as seen above in the colored arrows. In an ODE Network, the output is a continuous transformation of the input over time. The gradient vector field of the solution is continuous, as seen above in the colored arrows. This image was taken from the paper that introduced Neural Ordinary Differential Equations [10].

states during the forward pass, this approach is more efficient in terms of memory than traditional backpropagation [10].

The loss function of the Neural Ordinary Differential Equation is expressed as:

$$L(\text{ODESolve}(y(t_0), F, t_0, t_1, \theta)), \tag{3.5}$$

where t_0 and t_1 are the start and end times for the Initial Value Problem (IVP). ODESolve is the ODE Solver that numerically integrates the ODE over the time interval $[t_0, t_1]$. The loss function is minimized by adjusting the weights of the Neural Network that approximates the dynamics of the system.

The adjoint variable $a(t)$ is the gradient of the loss with respect to the state $y(t)$ at each instance in time, $\frac{\delta L}{\delta y(t)}$. The dynamics of the adjoint variable are themselves given by another ODE— The adjoint ODE:

$$\frac{\delta^2 L}{\delta y(t) \delta t} = \frac{\delta a(t)}{\delta t} = -a(t)^\top \frac{\delta F(y(t), t, \theta)}{\delta y}. \tag{3.6}$$

The gradient of the adjoint variable with respect to time, $\frac{\delta a(t)}{\delta t}$, describes how the gradient of the loss (with respect to the system's state variable) changes as we move backwards through time, just as normal backpropagation does. This is because the loss is defined at the output of the ODE network, therefore defined first with the final state $y(t_1)$. To determine how the loss depends on earlier states $y(t)$, we must integrate the adjoint ODE backwards in time, starting in t_1 to t_0.

To solve the integral of the function in Equation 3.6, the values of the state at every point in time, $y(t)$, must also be known. Instead of saving them during the forward pass, therefore saving memory, they are recomputed backwards in time with the adjoint ODE.

Finally, the gradient of the loss function with respect to the parameters, $\frac{\delta L}{\delta \theta}$, is calculated with another integral over time:

$$\frac{\delta L}{\delta \theta} = -\int_{t_1}^{t_0} a(t)^\top \frac{\delta F(y(t), t, \theta)}{\delta \theta}\, dt. \tag{3.7}$$

Automatic Differentiation[1] is used to compute the gradients of the vector field F with respect to the state variable y, in (3.6), and the parameters θ in (3.7).

In total, four integrals are evaluated during the training of a NODE.

- The original ODE that defines the NODE, see Equation 3.4.
- The adjoint ODE that defines the adjoint variable, see Equation 3.6.
- The integral of the state variable backwards in time.
- The ODE that defines the gradient of the loss with respect to the weights, see Equation (3.7).

The last three can all be computed with a single call to an augmented ODE solver:

$$\frac{d\,\mathrm{aug}(t)}{dt} = -\begin{bmatrix} a(t)^\top \dfrac{\delta F}{\delta y} \\ a(t)^\top \dfrac{\delta F}{\delta \theta} \\ a(t)^\top \dfrac{\delta F}{\delta t} \end{bmatrix} (y(t), t, \theta), \tag{3.8}$$

where the solutions are the loss gradients.

During the forward pass, the ODE solver outputs values of $y(t)$ at specific times were the solution is evaluated. These states are needed for the adjoint ODE. The ODE Solver operates continuously, and the states between consecutive observations are interpolated. However, the adjoint ODE must account for the solution at specific observation times, as required by the loss function, so it is split into segments between consecutive observation points.

At each of these segments, the adjoint variable must be updated to account for the current state. The integration backward in time is paused to add the direct contribution of the loss's dependence on the current state. After updating the adjoint variable, the adjoint ODE continues to be solved backward through time until the next segment, where new

[1]Automatic Differentiation calculates derivatives by recording the outcomes of each operation (such as additions and multiplications) during execution. It monitors how these operations impact the derivative calculations and uses this data to compute the derivatives in real-time.

information about the state becomes available. This process is repeated until the adjoint ODE has been solved for the entire time interval.

Defining models using ODE Solvers offers several advantages:

- Smoother transformation of the input
- The type of NN used to model the ODE can be chosen based on the problem at hand.
- The input for the model does not have to be sampled at regular intervals. This is not common with most time series models.
- The gradients of the loss function are not computed with backpropagation, using the chain rule. This eliminates the need to store intermediate values of the network during forward pass, reducing the memory requirements of the network.
- There are several ODE Solvers available. The choice of the solver can be used to control the trade-off between speed and accuracy. Modern ODE Solvers are designed to control numerical errors [23][10].

In the same paper that introduced NODEs, the authors compared the performance of an ODE-Net, a specific NODE model, to a ResNet on the MNIST dataset. The ODE-Net, with roughly one third the parameters of the ResNet, achieved similar performance. Essentially, a single ODESolve method replaced a ResNet with 6 layers. Additionally, the ODE-Net benefits from constant memory complexity during training.

The authors also present a timeseries modeling approach, a variational autoencoder, which treats time as a continuous variable rather than a discrete one. This framework surpasses traditional timeseries models by being inherently designed to handle irregularly sampled data.

The variational autoencoder is a generative model separated into two parts: an encoder and a decoder. The encoder maps the input data to a latent space, while the decoder maps the latent space back to the input space.

Instead of modeling the observed data directly, $x(t)$, it models the latent dynamics of the data. The latent state, $y(t)$, represents the underlying dynamics responsible for generating the observed data. In other words, the model assumes that there are hidden variables evolving over time, which influence the observed data. This latent state is governed by a differential equation that describes how the state evolves over time—the latent dynamics. This approach allows the model to better generalize to unseen data, as it is not directly dependent on the observable data. Additionally, by separating the latent dynamics from the noisy observations, the model becomes more robust to noise and irregularities in the data.

In the paper, the decoder used is a Recurrent Neural Network (RNN) that processes the observed data backwards in time. The model is used to obtain the initial latent state, $y(t_0)$, from the observed data, $x(t_1), x(t_2), ..., x(t_N)$.

The posterior is the conditional probability distribution that describes the probability of the latent variable, in this case $y(t_0)$, given the observed data. The RNN learns the parameters of the variational approximation of the posterior distribution over $y(t_0)$, $q_\phi(y(t_0)|x(t_1), ..., x(t_N)$. If, for example, the distribution where Gaussian, the model would learn the mean and variance. The initial latent state, $y(t_0)$, is then sampled from the learned posterior. Given this initial state, the ODE solver is used to integrate the latent dynamics, $f(y(t), \theta_f)$ over time, outputing the latent states, $y(t_1), y(t_2), ..., y(t_N)$. The latent dynamics approximate how the latent states evolves over time (forward or backward) using a Neural Network that is not explicitly dependent on time, but on the state.

After all the latent states are computed, a different model is used to generate the observed data, $x(t_1), x(t_2), ..., x(t_N)$. This is a generative model, the decoder, as it learns the underlying distribution of data allowing it to generate new samples:

$$x(t_i) \sim p(x|y(t_i), \theta_x). \tag{3.9}$$

The observed data at a given time, $x(t_i)$, is sampled from a distritbution that depends on the latent state at that time, $y(t_i)$, and the parameters of the model, θ_x. The model learns θ_x such that the distribution $p(x|y(t_i), \theta_x)$ accurately captures the relationship between the latent states and the observed data.

The paper showed that ODE Solvers, when incorporated as a model component, provide a roboust framework for a range of models, including timeseries and supervised learning. By using time as a continous variable, these models offer a new approach to handling irregularly sampled data and improving generalization.

Extensions of Neural Ordinary Differential Equations Since the introduction of NODEs models, several extensions have been proposed to improve their performance and address their limitations. One example of such limatations is their difficulty to handle stiff differential equations [67] [23]. This occurs when the eigenvalues of the Jacobian matrix of the ODE have widely varying magnitudes, with some being much larger than others.

The stability of the solver has a significant impact on the accuracy of the solution, even if the ODE itself is stable (i.e., the solution does not grow unbounded). A stable solver ensures that numerical errors do not amplify. If the solver is not stable, the solution may exhibit oscillatory behavior (due to eigenvalues with significant imaginary components) or grow exponentially with each step (due to amplification of numerical errors). If the step size does not fall within the solver's stability region (the range of step sizes for which the solver remains stable), the solver will not be stable.

Implicit methods, defined in 2.2, have a larger stability region than explicit methods. When using explicit methods in stiff problems, the step size must be very small to be in the stability region of the solver. This is computationally expensive, therefore, when the problem is stiff it is better to use implicit methods, which allows for larger step sizes.

To account for this, Westny et al. proposed a new initialization method for NODEs that improves the solver's stability [67].

In another paper, Fronk and Petzold focused on the interpretability of NODEs. They introduced Polynomial Neural Ordinary Differential Equations, which use a Deep Polynomial Neural Network (DPNN) instead of a regular Neural Network to model the underlying ODE of the system [22]. The motivation behind this approach is that traditional Neural Network are considered *black boxes*, as the models are not directly interpretable. Previous work had addressed this issue by combining SINDy with NODEs to make the model interpretable by providing the symbolic equations that describe the learned system [32].

The output layer of a DPNN is a polynomial function of the input layer[2]. Therefore, using DPNN allows the NODEs to become interpretable without requiring additional frameworks. The authors found that PNODEs were able to learn the vector field more effectively than conventional NODEs, even when the underlying function was not a polynomial. Additionally, PNODEs were able to better predict outside of the training region compared to conventional NODEs.

Another example of an extension to NODEs is Augmented Neural ODEs (ANODEs). Dupont et al. proposed a method to improve the expressiveness of NODEs by augmenting the state space with additional variables [16]. The authors argue that taking the continuous limit imposes constraints on NODEs, making them incapable of modeling certain functions.

The flow is defined as the influence of the initial position x on the hidden state as it evolves over time under the dynamics of the ODE. In other words, it is the solution of the ODE at time t, given the initial state. NODEs cannot represent functions whose flows intersect when starting from different initial positions. Such an intersection would mean that the solution at the point of intersection is not unique, which violates the uniqueness property of ODEs that follow the Lipschitz condition.

The *Picard–Lindelöf* theorem guarantees a unique local solution whenever the vector field f is continuous in t and follows the *Lipschitz* condition, i.e. there exists a constant $L \geq 0$ such that

$$\|f(t, \mathbf{h}_1) - f(t, \mathbf{h}_2)\| \leq L\|\mathbf{h}_1 - \mathbf{h}_2\|, \tag{3.10}$$

for all t in some interval I and all $\mathbf{h}_1, \mathbf{h}_2$ in a region $U \subset \mathbb{R}^d$[60].

This condition, along with the Picard-Lindelöf theorem, guarantees that a solution to an ODE exists and is unique for a given initial condition and region. This restriction, limits the expressiveness of NODEs to model certain functions. ResNets do not face this problem because they perform discrete updates, allowing trajectories to "jump" and avoid intersections. To overcome this issue, ANODEs augment the state space in which the ODE is learnt and solved. This augmentation avoids trajectory intersections by lifting points

[2]The polynomial introduces the necessary nonlinearity between layers, therefore, there is no need to add a nonlinear activation function, making the model interpretable [11].

into additional dimensions. The authors showed that ANODEs can learn more complex functions using simpler flows, and are more computationally efficient than NODEs.

3.2 Physics-Informed Neural Networks

Physics Informed Neural Networks were designed to address two different types of data-driven problems: the data-driven discovery of partial differential equations and the data-driven solution of partial differential equations [51] [52]. These problems are often encountered in fields like physics, where data is scarce and difficult to obtain, and the governing physical laws are only partially understood, with certain terms missing. PINNs work best when there is a balance between the available training data and the knowledge of the underlying physical principles.

Consider a differential equation of the form:

$$\frac{\delta u}{\delta t} + N(u, \lambda) = 0, \tag{3.11}$$

where $N(u, \lambda)$ is a nonlinear differential operator that depends on the solution u and unknown parameters λ. The goal is to find the solution u that satisfies the differential equation (representing laws of physics) along with any initial and boundary conditions, if available. PINNs approximate the solution using a Neural Network, $u_\theta(x, t)$, where θ represents it's trainable weights, as seen in Figure 3.2.

In order to enforce the laws of physics in the training of the Neural Network, the residual of the differential equation is defined as:

$$f = \frac{\delta u}{\delta t} + N(u, \lambda). \tag{3.12}$$

The partial derivatives needed for f are calculated using Automatic Differentiation.

The solution u and the unknown parameters λ are obtained by minimizing the loss function, which is defined as:

$$L = MSE_u + MSE_f, \tag{3.13}$$

where:

- Data loss (MSE_u) - the Mean Squared Error between the predicted solution and the observed data.
- Physics-Informed loss (MSE_f) - the Mean Squared Error of the residuals, f.

Therefore, complete compliance with the laws of physics requires the Physics-Informed Loss component, MSE_f, to be nearly zero. Achieving this increases the loss from the data, MSE_u, which can lead to noisier estimates. To address this trade-off, a regularization term is frequently introduced to the loss function.

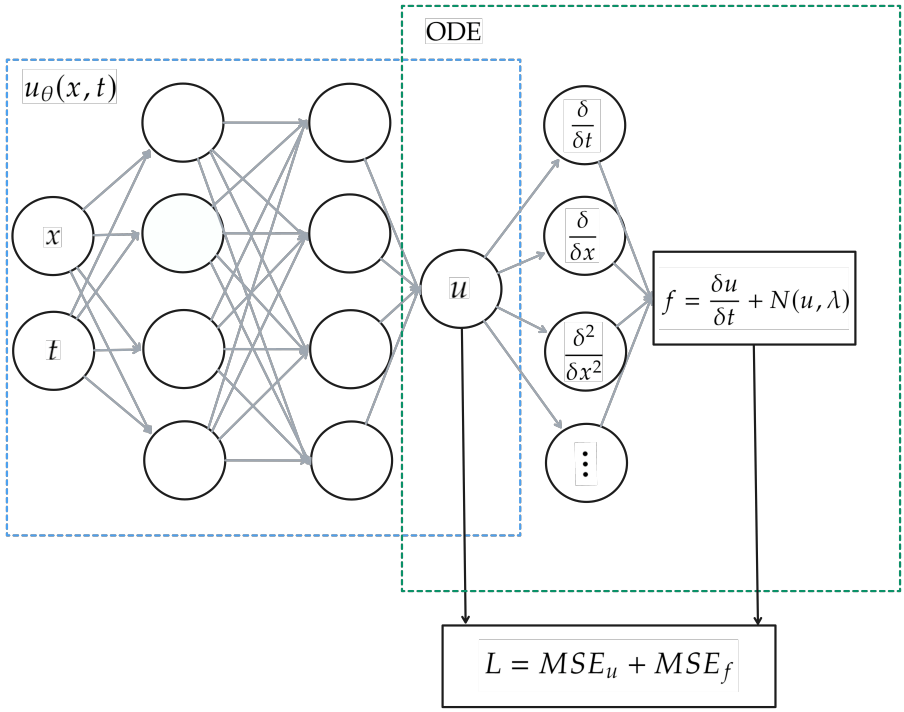


Figure 3.2: This diagram illustrates the architecture of a PINN. The blue rectangle encompasses the Neural Network, which consists of multiple dense layers. The green rectangle highlights the differential equation component of the network. This part takes the Neural Network's output and computes the partial derivatives using Automatic Differentiation. These derivatives are then used in the residual, f, which represents the governing laws of physics of the system. PINNs employ a dual loss function that combines data-driven model training and adherence to the laws.

By combining these two loss terms, PINNs mathematically express the objective that the solution not only fits the data but also adheres to the underlying physical laws, leading to solutions and the discovery of unknown parameters that are both accurate and physically consistent, even when the available data is limited or noisy.

Despite their advantages, PINNs face several challenges [65]. One notable issue is that since the physics constraints are only applied during the training phase through the loss function, no physics is actively enforced during future predictions. This leads to some uncertainty about whether the model has correctly learned the underlying differential equations for all possible input values. As a result, there is a risk that the model might generate significant errors in its predictions that go unnoticed.

Another challenge is the imbalance in back propagated gradients [64] [65]. Each of the two terms contributes gradients that are back propagated through the Neural Network to adjust its weights. However, the magnitudes of these gradients can vary significantly, leading to stiffness in the gradient flow, as some terms in the loss function change much faster than others.

The total gradient with respect to the trainable parameters θ is given by:

$$\frac{\delta L}{\delta \theta} = \frac{\delta L_u}{\delta \theta} + \frac{\delta L_f}{\delta \theta}, \tag{3.14}$$

where $\frac{\delta L_u}{\delta \theta}$ is the gradient from the data loss, and $\frac{\delta L_f}{\delta \theta}$ is the gradient from the physics-informed loss.

If the gradient from the physics-informed loss, is much larger than that from the data loss, the updates to θ will mainly be driven by the physics-informed loss, as the contributions from the other term will be small in comparison. This imbalance can cause the model to prioritize fitting the data at the expense of violating the physics or vice versa.

To address this issue, the learning rate must be carefully adapted, as small updates are required for the steep gradients in order not to overshoot the minima by changing the parameters too much, whereas larger updates are needed for the small gradients that reside in smoother regions. This can lead to slow convergence due to the unbalanced requirements of the learning rate.

4

Exploratory Data Analysis

The ephemeris data for this Master Thesis was chosen to closely reflect real world conditions, ensuring that the findings are applicable to practical scenarios. This data will later be used to train NODEs and PINNs to propagate satellite positions, allowing for a comparison of their performance.

The following section provides an overview of the data, including its source and how it was collected. Additionally, this section includes an Exploratory Data Analysis (EDA) on a subset of the data collected, which will help to better understand it.

4.1 Ephemeris Data

Ephemerides are files that contain predicted positions and velocities of objects in space at specific times. For this work, the ephemerides were obtained from the Space-Track website using their API, a platform managed by the United States Space Command (USSC) that offers publicly available data on space objects and their orbits, including satellites and debris. The ephemeris files used belonged to Starlink satellites, a constellation operated by *SpaceX* with over 6000 satellites. These files have been uploaded to Space-Track since May 2021 and provide propagated data spanning a three-day period.

The ephemerides are in the Modified ITC format, containing fields such as position (km) and velocity (km/s), along with 21 covariance values representing the lower half of a triangular covariance matrix for position and velocity. The position and velocity are given in the MEME (Mean Equator Mean Equinox) *Julian year 2000 (J2000)* reference frame, whereas the covariance values are expressed in the *UVW* frame.

The MEME J2000 frame is an Earth-Centered Inertial (ECI) reference system. Its axis are defined using Earth's equator and the position of the equinox at the start of the J2000, corresponding to 12:00 Terrestrial Time on January 1, 2000. In this frame, the Z-axis is perpendicular to the mean equatorial plane, pointing north, while the X-axis points to the mean position of the vernal equinox, and the Y-axis is orthogonal to both.

Unlike an ECI frame, the UVW frame (also known as the Radial-Transverse-Normal (RTN) frame) is local and non-inertial meaning it is tied to the moving object rather than

a "fixed" reference point. The U-axis points radially from the object toward Earth's center, the V-axis is tangential to the orbit in the direction of motion, and the W-axis is normal to the orbital plane.

4.2 Ephemeris Structure

The files downloaded via the API are provided as ZIP archives. A typical filename follows the format: "SpaceX_Ephemeris_552_SpaceX_2024-12-12UTC05_21_06_17.zip".

The timestamp in the filename indicates when the file was uploaded. The third field ("552") remains the same across all ZIP files, while the last two digits represent the ZIP file's sequence number. Each ZIP archive contains around 500 ephemeris files in TXT format.

The ephemeris files uploaded to Space-Track must follow a specific naming convention. The general format can be seen in Table 4.1.

Format Component	**Example Value**
DataType	`MEME`
Catalog number	`44714`
CommonName	`Starlink-1008`
DayTimeGroup	`3250450`
Operational/Special	`Operational`
MetaData	`1416372660`
Classification	`UNCLASSIFIED`
File Extension	`.txt`

Table 4.1: Naming convention of ephemeris files uploaded to Space-Track. The right column provides an example of each field corresponding to a Starlink file name.

The field 'DayTimeGroup' combines the year, day of the year, hour, minute, and second when the ephemeris file was created. In the example provided in Table 4.1, the ephemeris file was created on the 325th day of 2024 at 04:50:00 UTC, which corresponds to November 20, 2024, at 4:50 AM UTC (the year itself is not explicitly included in the field.).

The 'MetaData' field contains a Unix 1980 timestamp. However, this field is generally meant to include additional details provided by the satellite operator, such as the maneuver status or planned maneuver options. The "Catalog Number" refers to the North American Aerospace Defense Command (NORAD) ID, a unique identifier assigned to each satellite by the USSC.

The first six rows of an ephemeris file contain metadata, including the creation timestamp, the start and stop times of the ephemeris data, the step size of the propagation (60 seconds), the source of the ephemeris and the reference frame for the covariance values.

After the metadata, each row begins with a timestamp in the format `yyyDOYhhmmss.sss` , which represents the year, day of the year, hour, minute, second, and fractional second. This is followed by the satellite's position and velocity values, along with 21 covariance values. An example of the first rows of an ephemeris file can be found in Figure 4.1.

Ephemeris File Example

```
created: 2024-11-27 21:31:00 UTC
ephemeris_start: 2024-11-27 21:01:42 UTC
ephemeris_stop: 2024-11-30 21:01:42 UTC
step_size: 60
ephemeris_source: blend
UVW
2024332210142.000 1610.4600892826 4515.3801199790 -5005.1720357675
-5.2714368810 4.7762919984 2.6147591009
 5.0465629211e-07 -3.7206693046e-07 7.3026178382e-07
-1.6385748692e-10 1.3145644397e-10 1.1593950291e-06 8.5334766282e-10
-8.5637710168e-10 -1.2116766053e-12 1.9406263737e-12
-4.5209435293e-10 3.7885490507e-10 1.6256036509e-12 -7.8392787388e-13
 4.7630769688e-13 -5.0508494707e-13 -2.9203033426e-13
 1.5733729931e-09 -5.9415670085e-16 2.2745483306e-15 5.5351490230e-12
```

Figure 4.1: An example of an ephemeris file, which contains time-stamped position, velocity and 21 covariance values belonging to a propagation from a satellite. The file includes metadata such as creation time, start and stop times, step size, and source.

The ephemeris source is labeled as "blend", which could mean the use of either a combination of propagators or of multiple data sources. According to Liu et al., Starlink uses two different propagators. The ephemeris data for the first two days of propagation accounts for Earth's nonspherical gravitational effects up to J20, while the third and final day of propagation only accounts for this effect up to J2.
As discussed in Section 1, Earth is not a perfect sphere but rather an oblate spheroid with variations in mass distribution. This results in a non-uniform gravitational field, meaning that the gravitational force experienced by a satellite varies depending on its position. To account for these variations, Earth's gravitational potential is expanded into spherical harmonics, where each J-term (J2, J3, J4, etc.) represents a different aspect of these irregularities. Therefore, a propagation that only accounts for the J2 term is a simplified version of Earth's gravitational force. The transition from J20 to J2 means that short-term predictions prioritize accuracy, while longer-term calculations prioritize computational efficiency with a less detailed model [33].

4.3 Data Collection

The files for each satellite were uploaded every 8 hour, three times a day, and removed by Space-Track after 1 day of their upload. This meant that it was crucial to obtain the files as soon as they arrived, to make sure none were missing. Stalink has over 6000 satellites, and over time the files accumlate very quickly, with each file containing 2MB of data. In order to avoid an issue of space, the ephemerides collected for were for the first 1000 satellites in the constellation, ranging from NORAD ID 44714 to 48110. This collection started on the 27th of November 2024.
The files were collected using the Space-Track API through a Python script that automatically downloaded them one hour after they were uploaded. This process was scheduled using crontab to run the script on machines provided by the university's cluster.
Up until January 7th, the files were downloaded without any issues. However, after this date, data collection became inconsistent. After two weeks of automatically collecting the data, the files were not being downloaded correctly due to errors with the "requests" library. Upon investigation, it was discovered that some machines in the cluster were running different versions of Python, with some versions not supporting the "requests" library. Furthermore, there was a three-day period during which the files were not uploaded to Space-Track. As a result, the data for the first two days is missing. On the third day, however, the data was retrieved directly from the Starlink website, which serves as a mirror of the files hosted on Space-Track. After the three-day interruption, uploads to Space-Track resumed as normal. However, each file name now included a unknown string at the beginning of the file name.

4.4 Related Work on Starlink Ephemeris Data

There has been limited research analyzing Starlink's ephemeris data, likely because the data is deleted one day after publication, requiring researchers to actively collect it in real time. Despite being publicly available since 2021, only a handful of studies have focused on it. Notably, Liu et al. analyzed the ephemeris data to detect maneuver strategies of Starlink satellites as well as their frequency [33].
Satellites in LEO are subject to more atmospehric drag compared to those in higher-altitude orbits. As stated in Section 1, atmospheric drag causes satellites to gradually lose altitude, leading them into lower orbits until they eventually re-enter the atmosphere. To counteract this effect, satellites must perform regular low-thrust maneuvers to maintain their operational altitude.
Since atmospheric drag depends a lot on altitude, Liu et al. seperated their maneuver extraction techniques into different categories. In their research, they identified maneuvers by analyzing semi-major axis changes using Starlink ephemeris data. They found that deorbiting and climbing orbit satellites maneuver every 0.1 days (2.4 hours), while parking satellites maneuver about every 0.25 days (6 hours). Starlink's version 1.0 satellites perform

maneuvers approximately every 2 days, while their other satellites perform a maneuver roughly once a day. Notably, these maneuvers alter satellite trajectories, leading to discrepancies in orbit propagation.
Other researchers have also studied Starlink's ephemeris data, however, the research is not in English which limits the accessibility to its findings [14].

4.5 Exploratory Analysis

Each ephemeris file contains propagated orbital data covering a three-day period. Given this structure, analyzing the data over a three-day window is the most appropriate way to understand the behavior of the satellites.

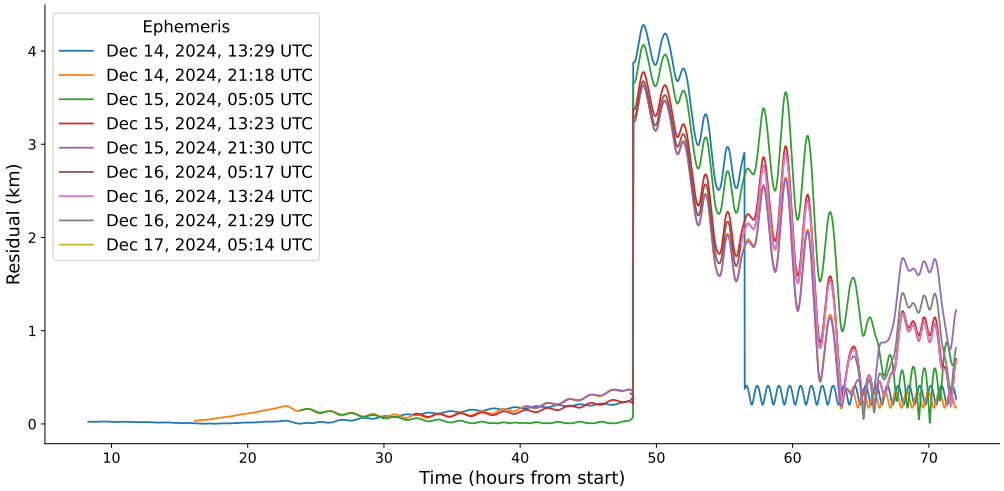


Figure 4.2: Comparison of predictions between the first ephemeris file, starting on December 14, 2024, at 4 AM UTC, with the subsequent ephemeris files to form a complete three-day prediction for the satellite 44917. The residuals represent the distance between the corresponding points in each comparison.

The data in these files is propagated by an unknown propagator that uses undisclosed features. This means the files do not contain raw satellite data but rather the output of a propagator that processes the raw data to predict the satellites' positions and velocities. As the propagation progresses through time, errors accumulate, and the data becomes less accurate. Consequently, the most accurate and reliable data is the one closest to the upload time—meaning that for any given three-day period, the most accurate data comes from the first 8 hours of consecutive ephemeris files. All further analysis of the ephemeris data is done using only the most accurate data.

New files are uploaded approximately every 8 hours, therefore there is an overlap of around 62 hours when consecutive ephemeris files intersect. These overlapping regions between files allows us to analyze accuracy of Starlink's propagator by comparing predictions across different upload times. As seen in Figures A.1 and 4.2, the residuals between

later files and the first file increase over time, becoming more pronounced after the 48-hour mark. This sharp increase in residuals beyond 48 hours aligns with the findings of Liu et al., which indicate that the first two days of propagation are more accurate than the third day due to the use of a simplified gravitational model. In other words, the sharp increase is due to the transition from a more accurate model to a less accurate one.

Figure A.2 shows the evolution of satellite positions over a three-day period starting on December 14, 2024. The position vectors (x, y, z) are oscillatory in nature, which is expected since their orbits are elliptical. As a result, when describing the trajectory in Cartesian coordinates, trigonometric functions must be used, leading to this periodic behavior. The same behaviour can also be seen in the evolution of the magnitude of the velocity in Figure A.5.

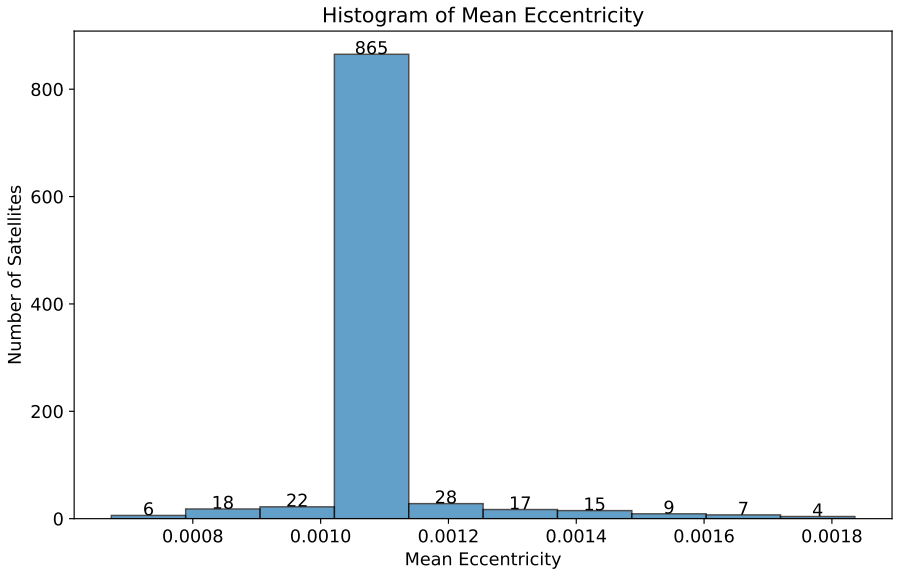


Figure 4.3: Histogram of the mean eccentricity of 991 observed satellites observed between December 14 and December 17. These eccentricities were calculated with the ephemeris data.

The eccentricity of a satellite's orbit quantifies its deviation from a circular trajectory. An eccentricity of 0 corresponds to a perfectly circular orbit, while values greater than 0 but less than 1 correspond to an elliptical orbit. As shown in Figure 4.3, the eccentricity of Starlink satellites is close to 0, meaning their orbits are nearly circular but still slightly elliptical. Consequently, Earth is positioned at one of the focal points of the orbit, causing the satellites to move closer to and farther from Earth at different points in their orbit.
Figure 4.4 show the evolution of a satellite's orbit over the same three-day period. As time progresses, the orbit gradually shifts, creating a twisting effect. The visual changes in the satellite's orbit result from external forces acting on the satellite (as discussed in Section 1), as well as adjustments made through satellite maneuvers.

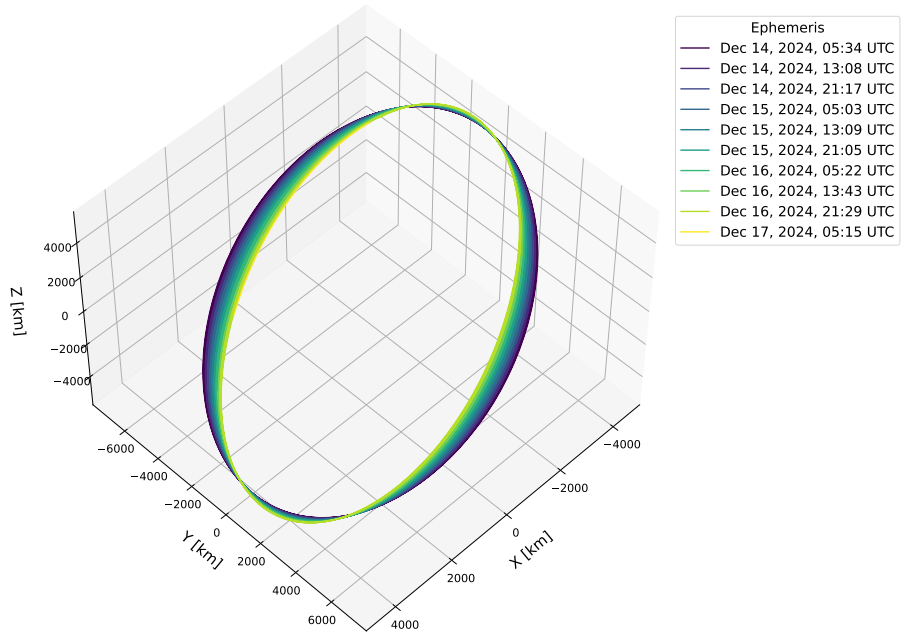


Figure 4.4: Visualization of the satellite's orbit evolution over three days for satellite 44751. To display the most accurate three-day trajectory, a combination of consecutive ephemeris files was used, selecting only the most accurate segment from each ephemeris file, roughly the first 8 hours of each file.

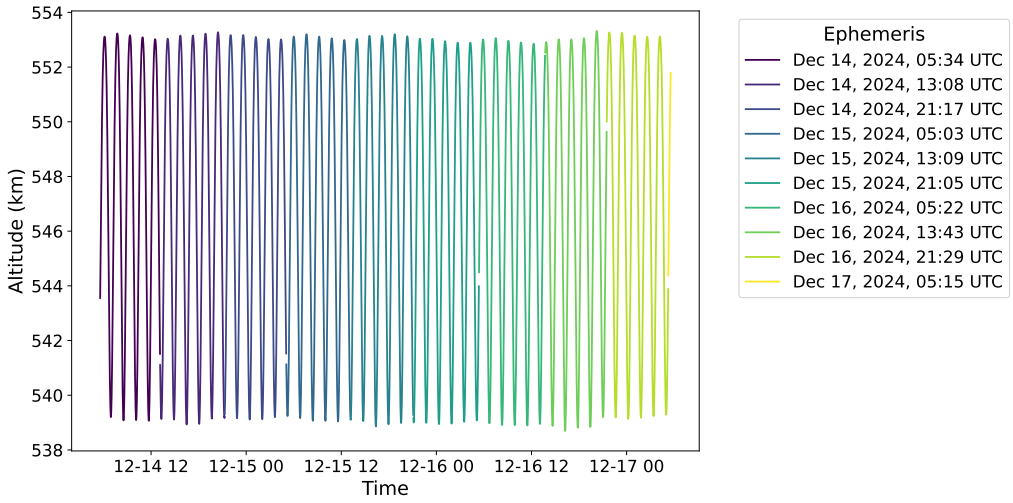


Figure 4.5: Altitude variation of satellite 44751 over three days, starting on December 14, 2024. The oscillatory pattern reflects the satellite's movement between apogee (maximum altitude) and perigee (minimum altitude) as it follows its elliptical orbit around Earth. Since the plotted altitude is derived from a combination of ephemeris files, using only the most accurate segments from each file, a discontinuity occurs at the transitions between files when one ends, and another begins.

A satellite's position reaches its maximum distance at apogee—the farthest point in its orbit from Earth—and its minimum distance at perigee, the closest point to Earth. This variation is reflected in the oscillatory pattern of a satellite's altitude over time, as shown

in Figure 4.5.
When the satellite reaches perigee, its radius is smaller, meaning that, according to the conservation of angular momentum, its velocity must be higher. Even if angular momentum is not conserved due to external forces (e.g., atmospheric drag and solar radiation pressure), this relationship still approximately holds. This causes the satellite's velocity to increase as it approaches perigee and decrease as it approaches apogee as seen in Figure A.5.

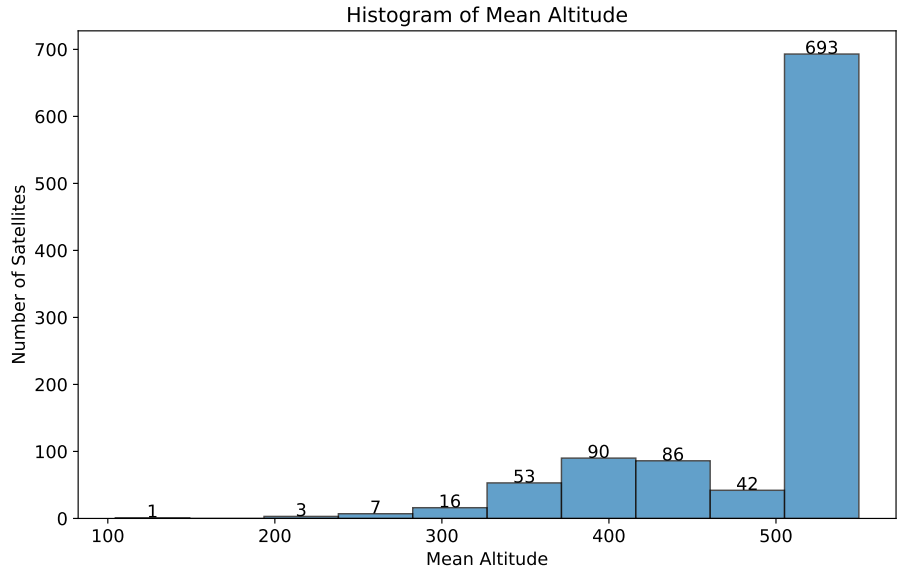


Figure 4.6: Histogram of the mean altitudes of 991 satellites observed between December 14 and December 17. These altitudes were calculated with the ephemeris data.

Histogram 4.6 illustrates the mean altitude of 991 Starlink satellites over 3 days while Figure 4.5 shows a detailed view of how the altitude of an individual satellite fluctuates over the same period. As expected, most satellites operate above 500 km, with a few falling below this range. According to Starlink, their satellites are deployed at altitues bellow 350 km and are boosted to 550 km to start their mission, with none exceeding 600 km. This is done to ensure that inactive satellites or debris will naturally deorbit within five years, aligning with current space regulations. Therefore, satellites at altitudes below 350 km are likely in the process of deorbiting.
Additionally, histograms of the mean orbital period and mean inclination are shown in Figures A.8 and A.9, respectively. The orbital period is the time it takes for a satellite to complete one full orbit around Earth, whereas the inclination is the angle between the satellite's orbital plane and Earth's equatorial plane. The inclination of Starlink satellites is approximately 53ºC, aligning with Starlink's deployment strategy. For most satellites, the orbital period is around 93 minutes, meaning each satellite completes approximately 15 orbits per day.

4.5.1 Deorbiting Satellites in LEO

This section analyzes the behavior of deorbiting satellites, which exhibit distinct orbital characteristics compared to operational satellites. While operational satellites maintain relatively stable orbits, deorbiting satellites undergo continuous orbital decay driven by a combination of controlled thrust maneuvers and atmospheric drag.
As satellite launches continue to increase, concerns about the sustainability of the space environment have led space agencies to establish guidelines aimed at reducing space debris. One such guideline is the *25-year rule*, which states that spacecrafts in LEO must deorbit within 25 years of completing their mission [12] [18]. Recently, stricter rules have been adopted by some space agencies that require satellites to deorbit within 5 years of completing their mission instead.
There are several ways to deorbit a satellite, but in LEO, the most common method is to rely on atmospheric drag to make the satellite re-enter Earth's atmosphere and burn up. However, in some cases, satellites may need to be actively deorbited using propulsion systems. According to a report issued by Starlink, atmopheric drag is enough to naturaly deorbit one of their satellites in 5 years or less [58]. In the same statement, SpaceX announced that they would perform the deorbiting of 100 of their oldest Starlink satellites, each taking approximately six months to deorbit using controlled propulsive maneuvers. They also stated that the satellites would completely burn up during re-entry. The environmental impacts of burning up satellites during re-entry are not fully understood, but recently, it has become a growing concern.

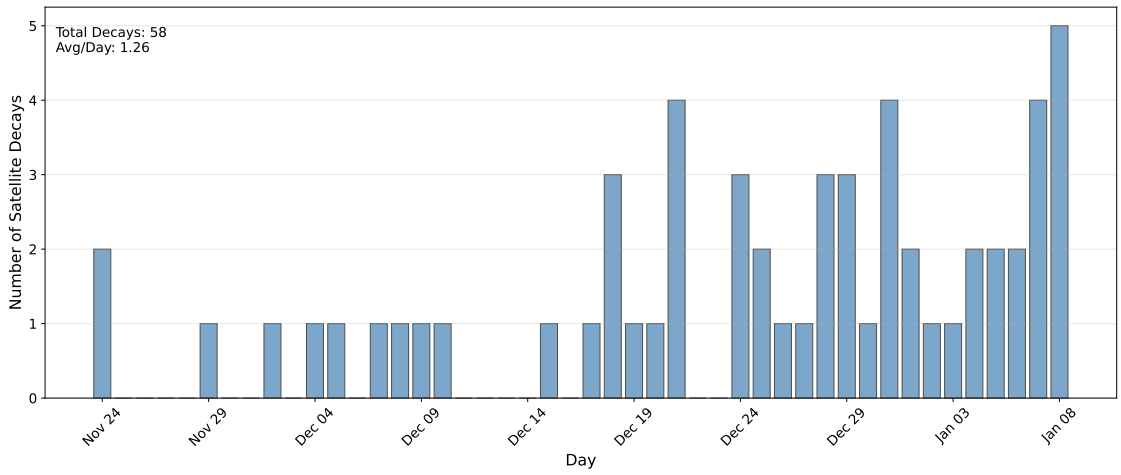


Figure 4.7: This bar graph illustrates the number of satellites that decayed each day from November 22, 2024, to February 4, 2025. In total 58 satellites of the collected satellites decayed during the time period.

Between December 21, 2024, and February 3, 2025, Starlink deorbited satellites, at times removing as many as five per day, resulting in a total of 58 deorbited satellites, as shown in Figure 4.7. The evidence of these deorbiting events is present in the gathered ephemeris data.

As previously mentioned, consecutive ephemeris files can be used to evaluate the accuracy of Starlink's propagation models. It is assumed that Starlink uses two different propagation model, one for the first two days of propagation and one for the last day. However, some satellites did not follow this expected pattern. Instead of a sharp increase in residuals after the 48-hour mark, their residuals gradually increased over time. This behavior is observed in satellites undergoing deorbiting. One possible explanation is that when deorbiting a satellite, Starlink changes their propagator to account for the continuous thrust required for deorbiting. This distinct behavior is illustrated in Figure 4.8 which show the residual of satellites 44760.

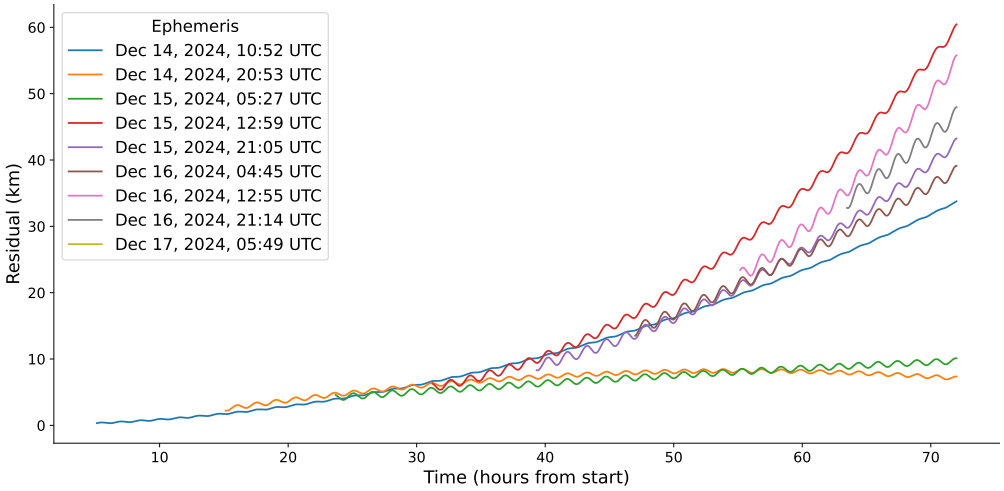


Figure 4.8: Comparison of predictions between the first ephemeris file, starting on December 14, 2024, at X UTC, with the subsequent ephemeris files to form a complete three-day prediction for the deorbiting satellite 44760. The residuals represent the distance between the corresponding points in each comparison.

The altitude of a satellite that completed its deorbiting process and was officially classified as decayed on December 15, can be seen in Figure A.4. It shows negative values after the expected decay date, suggesting that the propagator continues to compute the satellite's trajectory as if the satellite were still active. Before decay, the altitude has the expected oscillatory behavior, with an increase in wave frequency as the satellite nears the end of its life. This behaviour makes sense because as the satellite is getting closer and closer to Earth, the orbital period is becoming shorter and shorter. Additionally, as seen in Figure A.7, the magnitude of the velocity increases as the satellite approaches decay.
The altitude of satellite 44760, which was in the process of deorbiting but had not yet fully decayed, is shown in Figure A.3. As expected, its altitude is steadily decreasing. Meanwhile, its velocity, shown in Figure A.6, is increasing due to the additional thrust from propulsion.

4.6 Covariance Analysis

Examining the covariance of satellites, everytime a new ephemeris file is released the covariance values of the propagation are naturally reduced, as seen in Figures A.13,A.14 and A.15. This is expected, as newer ephemeris files are based on more recent and reliable data, leading to higher confidence in the predicted satellite position and velocity.
However, the covariance of ephemeris files seem to have sharp increases at specific time intervals, sometimes at the 18-hour mark, othertimes at the 28-hour mark, but almost always at the 48-hour mark as seen in Figures A.10 and A.11. These sudden changes in covariance could be attributes to a change in propagator used to generate the data. This is further supported by the covariance trends in deorbiting satellites. Unlike operational satellites, deorbiting satellites display a smooth and continuous increase in covariance over time, without the abrupt jumps, as seen in Figure A.12. This pattern aligns with their residual profiles, as shown in Figure4.8, where residuals also increase gradually rather than experiencing sharp deviations.
There appear to be multiple different covariance profiles across different satellites. This could mean that Starlink uses different propagators with different features, or a different combination of propagators for some satellites in its constellation. It could also be an indicator of a manourver, however, it would be unlikely that Starlink would not propagate their satellites with information about scheduled manouvers.

4.7 Conclusion

The uncertainty surrounding the propagators used in Starlink's ephemeris data, which appear to vary between satellites and even across different ephemeris files for the same satellite, presents a major challenge for using this data to train NODEs and PINNs. Additionally, the frequent maneuvers performed on these satellites also further complicate the integrity of the propagation.
Ideally, data used for training should reflect real-world orbital dynamics, meaning that satellites should primarily experience the natural forces present in the space environment without excessive external interference. By changing propagators and adding maneuvers, Starlink's orbits are not consistent and modelling them will bring incorrect results.
Despite these limitations, the dataset remains valuable for studying Starlink's orbital characteristics, maneuvering patterns, and the general behavior of satellites in LEO. While it may not be the best fit for OP, it provides useful insight into how commercial satellite constellations are managed and maintained.
Instead of using Starlink's data, a more suitable approach would be to generate synthetic data that accurately reflects the unactuated behavior of RSOs in LEO. For this, we will use Orekit, an industry-standard library for orbital mechanics [38]. Orekit provides a high-fidelity propagator capable of modeling perturbative forces such as atmospheric drag, solar radiation pressure, third-body gravitational influences, detailed gravitational

effects and more. It has been widely used for precise orbit determination [37],[45][9], propagation, and uncertainty analysis [2].
By generating synthetic data with Orekit, we can ensure that the data reflects real-world orbital dynamics while avoiding the inconsistencies and operational interventions present in the Starlink ephemeris data. This synthetic dataset will then be used to train the NODEs and PINNs, maintaining the original plan.

5

Model Formulation

This section starts by describing the training data, and the preprocessing steps applied to it, including the normalization of input features to improve model training. Then, it continues by describing the implementation of the three models used in this work. It begins with a Multilayer Perceptron (MLP) that is implemented to be a data driven baseline model, which relies solely on the available data without incorporating any physical knowledge of the system. The MLP serves as a comparison point to evaluate the effectiveness of the physics informed approaches. Then it follows the application of Physics Informed Neural Networks to the problem of satellite trajectory prediction, both as a standalone approach and as a baseline for evaluating the performance of Neural Ordinary Differential Equations. After the PINN model is described, the section discusses the implementation details of the Neural ODE.

All three models are used to address the forward problem, predicting the future states of a satellite based on known dynamics.

5.1 Data Representation and Preprocessing

The data used as input to the models is synthetic satellite trajectories generated using Orekit. As discussed in Chapter 4, the public Starlink data contained inconsistencies, leading to the decision to generate synthetic data instead. Orekit was used to propagate the orbits of Starlink satellites, from their initial position and velocity.

To ensure the synthetic data accurately reflected the characteristics of the real Starlink orbits, a parameter search was made to identify the optimal Orekit propagation parameters for each satellite. These parameters, included the drag coefficient, the area to mass ratio, and the reflectivity coefficient. They were selected to minimize the RMSE between the propagated orbits and the real Starlink data, ensuring that the resulting dataset accurately reflected real world conditions.

The satellites selected for propagation were chosen based on orbital stability, ensuring that they followed typical Starlink orbital patterns. Stability was assessed by analysing the evolution of the semi major axis over time. As shown in Figure 5.1, stable satellites (highlighted in blue tones in the Figure) exhibit an approximately constant semi major axis. These stable satellites were used generate the synthetic dataset, whereas satellites displaying noticeable orbital decay (highlighted in orange tones in the Figure) were excluded.

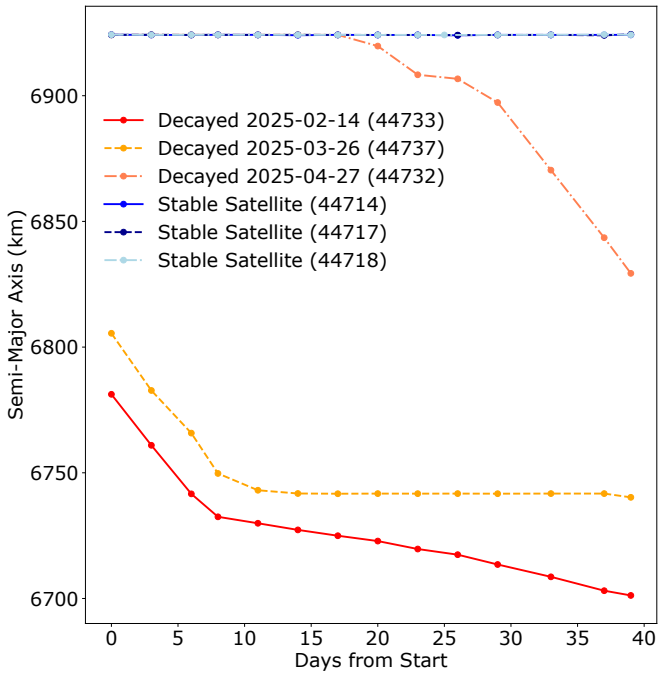


Figure 5.1: Evolution of satellites' semi major axis during the data collection period. Blue lines show stable satellites with minor manoeuvre induced variations, while orange lines indicate satellites undergoing deorbiting and eventual re-entry.

To this end, 300 orbits were generated from different satellites at different times. This number was chosen to balance the requirements of the models, as discussed in Chapter 6, with the available computational resources, since each orbit required approximately 40 minutes to optimise the parameters.

The synthetic data was then split into training, validation, and test sets, with the training set containing 80% of the data, the validation set 10%, and the test set 10%. This split ensures that the models are trained on a diverse range of scenarios while still being able to evaluate their performance on unseen data. Increasing the number of training orbits improved validation performance up to a certain degree as will be discussed in Chapter 6. However, the full 170 orbits were used for the normalization paramaters, even if not all of them were used to train the models.

5.1.1 Input Features

The input features used in the models include the satellite's initial position and velocity vectors, the elapsed time from the initial state to the prediction point, space weather parameters such as the geomagnetic activity index A_p and the solar radio flux indices $F_{10.7}$ and $F_{10.7}A$. These last two indices quantify the solar radio flux at a wavelength of 10.7 cm, and represents its 81 day average respectively. The A_p is an index for disturbances in Earth's magnetic field, and quantifies the strength of geomagnetic storms. Together, all three indices are used as proxies for solar and geomagnetic activity affecting atmospheric density and, consequently, satellite drag. The angular momentum derived from the initial state is also included, along with the past twenty position and velocity vectors to incorporate recent trajectory history.

5.1.2 Normalizing values

Before using the data for training, all input variables were normalized to improve stability and convergence. This normalization step is typically performed so that the inputs are within a similar range, ensuring that each feature contributes proportionally to the loss function.

All features were rescaled to be in the interval $[-1, 1]$. The normalization was performed as follows:

$$u' = 2 \cdot \frac{u_{\text{dataset}} - u_{\min}}{u_{\max} - u_{\min}} - 1, \tag{5.1}$$

$$\tag{5.2}$$

where u' is the normalized variable. The minimum and maximum values used for normalization correspond to the minimum and maximum values of each feature in the training dataset.

In order to be able to compare the normalized outputs of the model with values in the physics domain, the outputs need to be de-normalized. The formulation of the chain rules is as follows:

$$\frac{du_i'}{dt'} = \frac{du_i'}{du_i} \cdot \frac{du_i}{dt} \cdot \frac{dt}{dt'} \tag{5.3}$$

After calculating the needed derivatives, the domains can now be compared.

$$\frac{du_i'}{dt'} = \frac{t_{\max} - t_{\min}}{u_{i,\max} - u_{i,\min}} \cdot \frac{du_i}{dt} \tag{5.4}$$

This ensures that all variables are rescaled consistently while preserving their physical interpretation.

The only feature not normalized between $[-1, 1]$ is the time variable. Instead, it is normalized to the range $[0, 2\pi]$ so that it no longer represents the absolute time, but rather the angular position in the orbit. If the requested state corresponds to an angle beyond the training domain ($e.g., \theta = 3\pi$), the model can be called iteratively. First, it is used to predict the trajectory up to the last 20 minutes of a full orbit, which then serves as the new initial condition for the subsequent prediction up to $\theta = \pi$.
An alternative to normalizing the variables is to nondimensionalize them. This method scales them using characteristic quantities of the system so that the resulting variables are dimensionless.

5.2 Multilayer Perceptron Implementation

The first model implemented is a standard MLP that predicts the satellite's position and velocity at a given time step based solely on the available data. This model serves as a baseline for comparison with physics-embedded data-driven models, such as the PINN and the NODE.

The MLP architecture consists of an input layer, several fully connected hidden layers, and an output layer. The input layer receives the normalized features described in Section 5.1.2, while the output layer produces the predicted position and velocity vectors at the specified time. Additionally, past accelerations are also provided as input. The inclusion of the past acceleration information benefited only the MLP and the PINN, likely due to their more data-driven nature.

5.3 Physics Informed Neural Networks Implementation

The Physics Informed Neural Networks approach integrates the governing physical laws of satellite motion directly into the learning process, allowing the model to leverage both data and prior physical knowledge.

The PINN model F takes the input, described in Section 5.1.2, with an addition of past accelerations. It processes the input through a series of NNs and outputs the satellite's position and velocity at each time step, forming the state vector:

$$\begin{aligned} \vec{r}(t) &= [x, y, z] \\ \vec{v}(t) &= [v_x, v_y, v_z]. \end{aligned} \tag{5.5}$$

The outputs, $r(t)$ and $v(t)$, represent the satellite's position and velocity at time t. The time derivatives of the state vector are then computed using Automatic Differentiation:

$$\begin{aligned} \frac{d\vec{r}}{dt} &= [v_x, v_y, v_z] \\ \frac{d\vec{v}}{dt} &= [a_x, a_y, a_z]. \end{aligned} \tag{5.6}$$

These derivatives are used to build the physics loss function governed by the ODE of the system in Equation (5.7). Because the model outputs position and velocity in a normalized domain, the derivatives in Equation (5.6) are also normalized. To compare with the right-hand side of Equation (5.7), expressed in the physics domain, with the left-hand side, normalized, the Formula (5.4) is used.

$$\frac{d}{dt}\begin{bmatrix}\vec{r}(t)\\ \vec{v}(t)\end{bmatrix} = \begin{bmatrix}\vec{v}(t)\\ -\dfrac{\mu}{\|\vec{r}(t)\|^3}\cdot\vec{r}(t)\end{bmatrix} \tag{5.7}$$

Although satellite's motion is originally a second-order differential equation, it is reformulated as a system of two first-order ODEs, as discussed in Section 3.1. This eliminates the need to compute second-order derivatives during backpropagation, thereby reducing memory usage, computational complexity, and numerical instabilities.
As outlined in Section 1, satellites are subject to several non-conservative forces, whose accelerations can be incorporated into the ODE dynamics. These include perturbations from Earth's oblateness (J_2, J_3), Atmospheric Drag, Solar Radiation Pressure (SRP), and Third-Body effects from the Moon and the Sun [56]. The corresponding acceleration terms are defined below.

J2 Perturbation

$$\vec{a}_{J_2} = J_2 \cdot \frac{1}{|\vec{r}|^7} \cdot \begin{bmatrix} x(6z^2 - 1.5(x^2+y^2)) \\ y(6z^2 - 1.5(x^2+y^2)) \\ z(3z^2 - 4.5(x^2+y^2)) \end{bmatrix} \tag{5.8}$$

J3 Perturbation

$$\vec{a}_{J_3} = J_3 \cdot \frac{1}{|\vec{r}|^9} \cdot \begin{bmatrix} xz(10z^2 - 7.5(x^2+y^2)) \\ yz(10z^2 - 7.5(x^2+y^2)) \\ 4z^2(z^2 - 3(x^2+y^2)) + 1.5(x^2+y^2)^2 \end{bmatrix} \tag{5.9}$$

where J_2, J_3 are the second and third harmonic coefficients of Earth's gravitational potential.

Atmospheric Drag

$$\vec{a}_{\text{drag}} = -\frac{1}{2} C_D \frac{A}{m} \rho \, \|\vec{v}\| \vec{v}, \tag{5.10}$$

where C_D is the atmospheric drag coefficient, A is the satellite cross-sectional area (m^2), m the satellite mass (kg), ρ the atmospheric density (kg/m^3), and $\vec{v}$ the velocity vector. ρ

was estimated using the *NRLMSISE-00* model in Python, with solar activity inputs and geomagnetic indices from public datasets.[1,2]

Solar Radiation Pressure (SRP)

$$\vec{a}_{\text{SRP}} = -\frac{P_0 \cdot A \cdot C_r}{m} \left(\frac{\text{AU}}{\|\vec{r}_{\text{sat-sun}}\|} \right)^2 \hat{r}_{\text{sat-sun}}, \tag{5.11}$$

where P_0 is the solar radiation pressure at 1 AU (4.56×10^{-6} N/m^2), C_r the reflectivity coefficient, and $\hat{r}_{\text{sat-sun}}$ the unit vector from the satellite to the Sun.

Third-Body Perturbation (Sun or Moon)

$$\vec{a}_{\text{body}} = \mu \left(\frac{\vec{r}_{\text{body}} - \vec{r}_{\text{sat}}}{\|\vec{r}_{\text{body}} - \vec{r}_{\text{sat}}\|^3} - \frac{\vec{r}_{\text{body}}}{\|\vec{r}_{\text{body}}\|^3} \right), \tag{5.12}$$

where μ is the gravitational parameter of the third body, $\vec{r}_{\text{body}}$ the body's position relative to Earth, and $\vec{r}_{\text{sat}}$ the satellite's position in the same inertial frame.

These perturbations are added to the right-hand side of Equation (5.7), yielding

$$\vec{v}(t) = -\frac{\mu}{\|\vec{r}(t)\|^3} \cdot \vec{r}(t) + \vec{a}_{J_2} + \vec{a}_{J_3} + \vec{a}_{\text{drag}} + \vec{a}_{\text{SRP}} + \vec{a}_{\text{body}}. \tag{5.13}$$

Other non-conservative forces exist but are orders of magnitude smaller. The inclusion of these acceleration terms is intended to guide the model toward the underlying physics. While not strictly necessary (the model could approximate them from data), incorporating them into the loss acts as a regularization mechanism that could improve stability.

The loss function for the PINN combines a data-driven term with a physics-informed term, as described in Section 3.2. The data loss is computed as the Mean Squared Error (MSE) between the model predictions and the reference satellite states. The physics loss is computed as the MSE between the left and right-hand sides of the governing ODE, evaluated at collocation points.

Collocation points correspond to values of the input variables where physics constraints are enforced, ensuring that the model satisfies the governing equations across the entire domain rather than only at labelled points. The physics loss at collocation points is defined as:

$$\mathcal{L}_{\text{ODE}} = \frac{1}{N_c} \sum_{i=1}^{N_c} \left\| \frac{dr}{dt}(t_i) - v(t_i) \right\|^2 + \left\| \frac{dv}{dt}(t_i) - a(t_i) \right\|^2. \tag{5.14}$$

[1] https://spaceweather.gc.ca/solar_flux_data/daily_flux_values/fluxtable.txt
[2] https://www-app3.gfz-potsdam.de/kp_index/Kp_ap_Ap_SN_F107_since_1932.txt

The original PINN setup by Raissi et al. [51] only includes boundary conditions in the data loss term, leaving the dynamics to be learned in a self-supervised way from collocation points. Adding more data points to the data loss can, however, help anchor the model to the physics, especially when data is noisy or sparse.

The total loss is:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{ODE}} \mathcal{L}_{\text{ODE}} + \lambda_{\text{Data}} \mathcal{L}_{\text{Data}}, \tag{5.15}$$

with coefficients λ controlling the relative importance of each term. If one dominates, the optimization becomes skewed. Proper tuning ensures physically consistent solutions while improving accuracy and convergence. The initial conditions of the satellite state are enforced separately as boundary conditions to ensure the model starts from the correct state.

5.4 Neural Ordinary Differential Equations Implementation

Similarly to the setup described in Section 3.1, the NODE predicts the satellite's position and velocity at a given state by integrating the differential equation of motion over time. However, unlike PINNs, the NODE model does not explicitly incorporate the physics into the loss function. Instead, it learns the system dynamics directly from the data by approximating the time derivatives of the state vector, which are then integrated by an ODE solver. This allows the model to enforce the physics at every time step during training, validation, and testing, rather than only at discrete collocation points during training.

The NODE consists of two main components:

1. **ODE Solver:** Numerically integrates the dynamics over time, allowing for error tolerances and providing a trade-off between accuracy and runtime.
2. **NN:** Approximates the time derivative of the state vector given the current state and other input variables.

Another important distinction between the PINN and NODE models is what the NN outputs. In the PINN, the NN outputs the state values, $r(t_i)$ and $v(t_i)$, while the original NODE outputs the time derivatives of the state (instantaneous velocity and acceleration), $\frac{dr}{dt}(t_i)$ and $\frac{dv}{dt}(t_i)$. However, predicting both $\frac{d\mathbf{r}}{dt}$ and $\frac{d\mathbf{v}}{dt}$ was found to be unnecessary.
Since the derivative of position is the velocity, it can be set equal to the current velocity provided by the ODE solver, allowing the network to focus solely on predicting the time derivative of the velocity. This is possible because the solver reaches the solution at t_f by computing intermediate states at earlier times, progressively stepping forward from the initial condition. By contrast, for a PINN to predict only the acceleration, the output

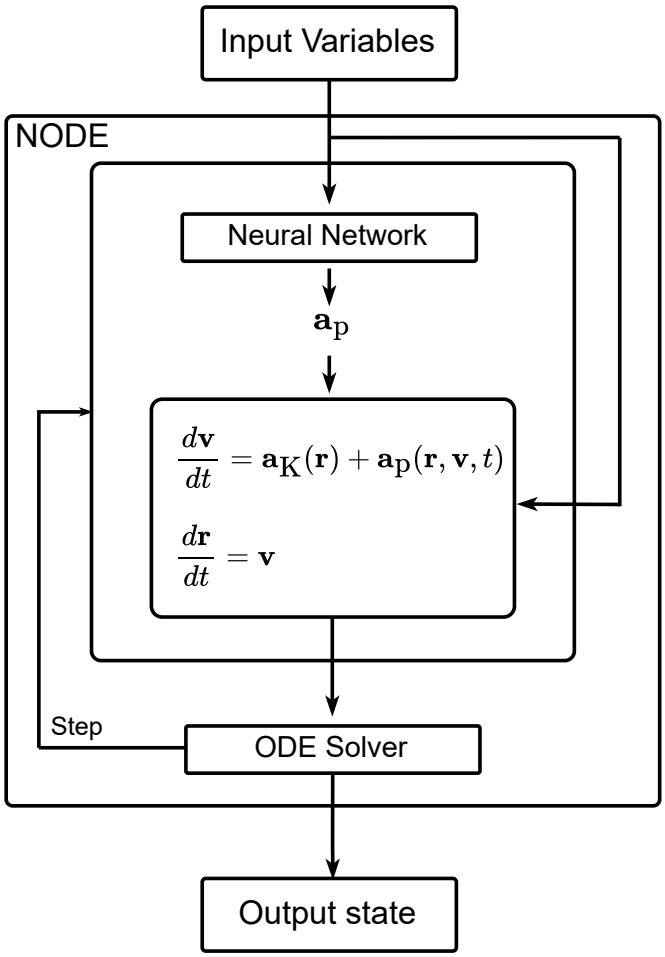


Figure 5.2: Neural ODE architecture used for satellite trajectory prediction. The input variables consist of space weather features, the satellite's initial position, velocity and angular momentum, and the elapsed time. The NN outputs the non-conservative accelerations, a_p, which are then added to the two-body acceleration, a_K. This output is integrated over time using an ODE solver to produce the predicted state at a future time step. The solver queries the NN at each integration step to obtain the time derivatives, which are then used to update the state vector. The process is repeated until the prediction time is reached. The output of the NODE is the predicted position and velocity of the satellite at the specified time step.

would need to be integrated twice (first to obtain velocity and then position), which is not recommended due to additional numerical instabilities introduced by repeated integration.

To accelerate learning, the Keplerian two-body acceleration was added to the predicted acceleration. This anchors the model in the dominant two-body dynamics, leaving it to learn only the non-conservative forces. As a result, the network outputs only three values corresponding to the components of the perturbative acceleration.

The NODE predicts only the perturbative acceleration $\mathbf{a}_{\text{pert}}$, such that:

$$\frac{d\mathbf{r}}{dt} = \mathbf{v}, \quad \frac{d\mathbf{v}}{dt} = \mathbf{a}_{\text{Kep}}(\mathbf{r}) + \mathbf{a}_{\text{pert}}(\mathbf{r}, \mathbf{v}, t), \tag{5.16}$$

where $\mathbf{a}_{\text{Kep}}(\mathbf{r}) = -\frac{\mu}{\|\mathbf{r}\|^3} \cdot \mathbf{r}$ is the Keplerian acceleration, and $\mathbf{a}_{\text{pert}}$ represents the non-conservative accelerations learned by the NN. The non-conservative accelerations can also be added to the right-hand side of the ODE, as described in Section 3.2. The gradient outputs of the NN are computed whenever the solver queries the network for time derivatives. The solver then integrates these derivatives over time to produce the predicted state at a future time step:

$$\begin{aligned} \mathbf{r}(t + \Delta t) &\approx \mathbf{r}(t) + \int_t^{t+\Delta t} \frac{d\mathbf{r}}{dt}(\tau)\, d\tau, \\ \mathbf{v}(t + \Delta t) &\approx \mathbf{v}(t) + \int_t^{t+\Delta t} \frac{d\mathbf{v}}{dt}(\tau)\, d\tau. \end{aligned} \tag{5.17}$$

This loop is repeated until the prediction time is reached.

The NODE is trained using a purely data-driven loss function, without explicit physics-based constraints. The loss is defined as the MSE between the predicted and reference satellite states:

$$\mathcal{L}_{\text{NODE}} = \lambda_r \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{r}_{\text{pred}}(t_i) - \mathbf{r}_{\text{true}}(t_i) \right\|^2 + \lambda_v \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{v}_{\text{pred}}(t_i) - \mathbf{v}_{\text{true}}(t_i) \right\|^2, \tag{5.18}$$

where $\lambda_r > \lambda_v$ is chosen to emphasize position accuracy, which is more challenging to learn than velocity.
The process of backpropagation through the ODE solver is handled using the adjoint method, as discussed in Section 3.1.

5.4.1 Numerical Integration of ODEs

As mentioned before, NODEs rely on ODEs solvers to compute the solution of the differential equation defined by the neural network. The choice of solver can significantly impact the model's performance. There are several types of ODEs solvers, each with its own advantages and disadvantages, with some requiring substantially more computational effort to achieve the same level of accuracy.

In this work, multiple ODE solvers were tested to evaluate the impact of solver choice on model performance. The PyTorch library offers a variety of solvers that can be divided into three categories: fixed-step explicit solvers, adaptive-step explicit solvers, and adaptive-step implicit solvers, divided in Table 5.1 [6] [25].

Chapter 2 briefly introduced numerical integration methods for solving ODEs, including the differences between explicit and implicit schemes. Explicit methods compute the next state based solely on the current state, making them generally faster and simpler to implement, but often less stable. Implicit methods, on the other hand, involve solving an equation at each step that depends on the next state, which improves stability but increases computational cost. Adaptive methods adjust the step size based on estimated error, allowing for more accurate solutions at the cost of runtime overhead. Fixed-step methods use a constant step size throughout integration, which can be less efficient for problems with rapidly changing dynamics. Table 5.1 summarises the solvers available in the PyTorch library that were tested in this work, along with their main characteristics. Not all solvers were used as the remaining solvers were too computationally expensive.

Table 5.1: ODE solvers tested in this work, classified by type, order, and step control. The first 6 methods are fixed-step explicit solvers, the next 3 are adaptive-step explicit solvers, and the last one is an adaptive-step implicit solver.

Solver	**Method Type**	**Order**	**Step Control**
Euler	Explicit (Runge–Kutta)	1	Fixed
Midpoint	Explicit (Runge–Kutta)	2	Fixed
Heun2	Explicit (Runge–Kutta)	2	Fixed
Heun3	Explicit (Runge–Kutta)	3	Fixed
RK4	Explicit (Runge–Kutta)	4	Fixed
Fixed Adams	Explicit (Multistep)	Variable	Fixed
Dopri5	Explicit (Runge–Kutta)	5(4)	Adaptive
Explicit Adams	Explicit (Multistep)	Variable	Adaptive
Implicit Adams	Implicit (Multistep)	Variable	Adaptive

The accuracy of an ODE solver is characterised by its order, which dictates how numerical error scales with the integration step size. High order methods provide greater accuracy for a given step size but require more function evaluations per step, increasing computational cost. The order of Adams methods depends on the number of steps used, and therefore they are classified as variable order methods. Dopri5 is denoted as 5(4) because at each step it advances the solution with a fifth order Runge–Kutta method, while simultaneously computing a fourth order Runge–Kutta estimate to control the local error and adapt the step size.

6

Model Comparison and Results

This chapter compares the performance of three distinct models for satellite trajectory prediction. The first is a standard MLP that relies solely on the available data, without incorporating any physical knowledge of the system. The second is a PINN, which serves as a physics-based baseline by embedding the governing orbital dynamics into the learning process. The third is a NODE, which learns the system's underlying dynamics in a continuous-time framework.

The objective of this comparison is to evaluate how well each model can predict the motion of a satellite in LEO given a fixed training dataset, and to determine which approach offers the best balance between accuracy, generalisation, and adherence to the physical laws governing orbital motion. The section begins with the selection of hyperparameters for each model, detailing how the number of orbits, the number of collocation points (for the PINN), and the choice of ODE solver (for the NODE) were determined.

6.1 Model Architectures and Training

After defining each model in the previous Chapter, several architectures were tested to find the best performing configuration. For each model, a hyperparameter search was conducted to determine the optimal architecture and training parameters. The hyperparameters considered included the number of layers, number of neurons per layer, learning rate, and regularisation techniques such as dropout and activation function.

Two activation functions were tested: the Gaussian Error Linear Unit (GELU) and the hyperbolic tangent (Tanh) [26]. Both are smooth and fully differentiable, which guarantees that first order derivatives of the NN output exist and are well defined. This property is particularly important for the PINN, since its loss function involves differential operators. The GELU is defined as

$$\mathrm{GELU}(x) = x \cdot \Phi(x) = \tfrac{1}{2} x \left(1 + \mathrm{erf}\left(\tfrac{x}{\sqrt{2}}\right)\right), \tag{6.1}$$

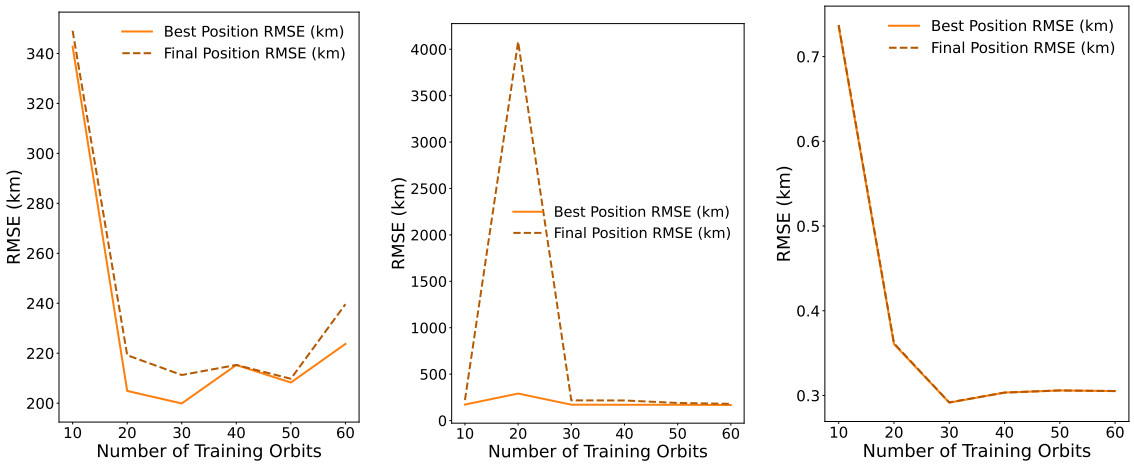
where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution and $\text{erf}(\cdot)$ is the *error function*, given by

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt. \tag{6.2}$$

The Tanh is defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{6.3}$$

The number of training orbits was treated as an important parameter, since it directly influences both model accuracy and training time. Figures 6.1a show how the performance of validation orbits evolve with the number of training orbits for each model.



(a) MLP (b) PINN (c) NODE

Figure 6.1: Evolution of the RMSE for the different models as the number of training orbits increases.

For all three models, increasing the number of training orbits initially reduces overfitting on the training data, corroborated by the high RMSE values, and improves generalisation as the model is exposed to a broader set of trajectories. However, beyond a certain number of orbits the addition of more diverse orbits introduces conflicting patterns.

In the case of the MLP in Figure 6.1, the optimal number of training orbits was found to be around 30. Beyond this point, the validation RMSE starts to increase again, indicating that the model is struggling to reconcile the conflicting patterns in the data. This is further illustrated in Figure 6.2, which shows the distribution of RMSE values across different validation orbits as the number of training orbits increases. As more than 30 orbits are added, the spread of RMSE values narrows, outliers with significantly higher errors begin to appear, and the mean RMSE becomes higher. This shows that while the model may perform the same on average, it struggles with certain orbits due to the

increased complexity and variability in the training data. Additionally, the model reaches a performance plateau, where adding more epochs does not lead to improvement, as seen in the dotted line in Figure 6.1.

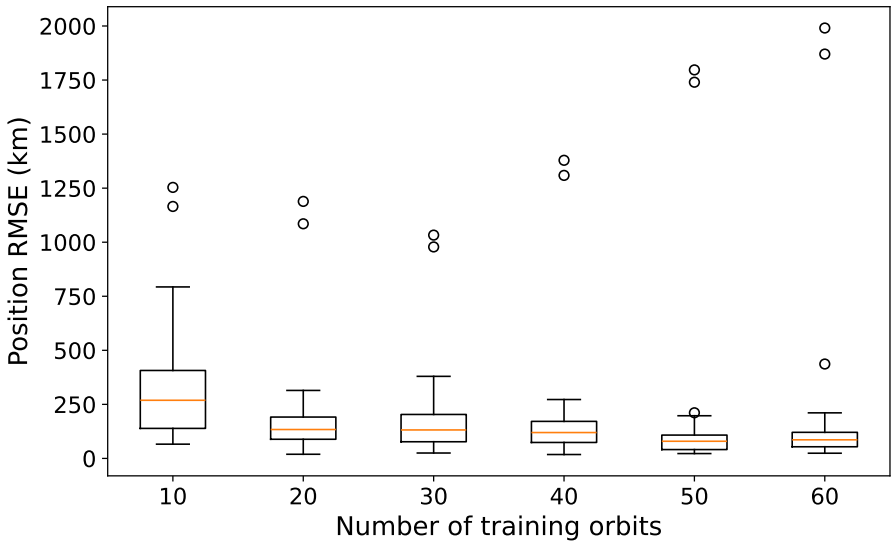


Figure 6.2: Box plot showing the distribution of RMSE values for validation orbits across different numbers of training orbits for MLP model.

For the PINN model, the increase in orbits does not lead to an immediate improvement. Instead, when 20 orbits are used, the model's gradients explode and it struggles to learn. However, as more orbits are added, the model eventually stabilizes and starts to improve again, as shown in Figure 6.1b. The optimal number of training orbits for the PINN was found to be 40, where the validation RMSE reaches its lowest point and there are fewer outliers, as seen in Figure 6.3. Similar to the MLP, the PINN also reaches a performance plateau with the number of epochs (Figure 6.1).

The forces described in Section 3.2 were added to the ODE in the PINN loss iteratively, but they did not lead to improvements in performance (see Figure 6.4). As such, only the gravitational force without the J_2 and J_3 perturbations was used in the final model. This is likely due to the increased stiffness and complexity introduced by these additional terms, which make the residuals more difficult to minimise.

Finally, for the NODE model, the best-performing epoch always coincided with the last epoch, as shown in Figure 6.1. The model with the validation set reaches its lowest RMSE with 30 orbits. Given this, and the fact that no outliers are present in Figure 6.5, 30 orbits was selected as the optimal number of training orbits for the NODE.

Multi Layer Perceptron The MLP was trained with 30 training orbits, 5 hidden layers with 256 neurons each, and GELU activations (no dropout). The training ran for 10,000 epochs using the Adam optimizer with a One-Cycle learning rate (LR) schedule.

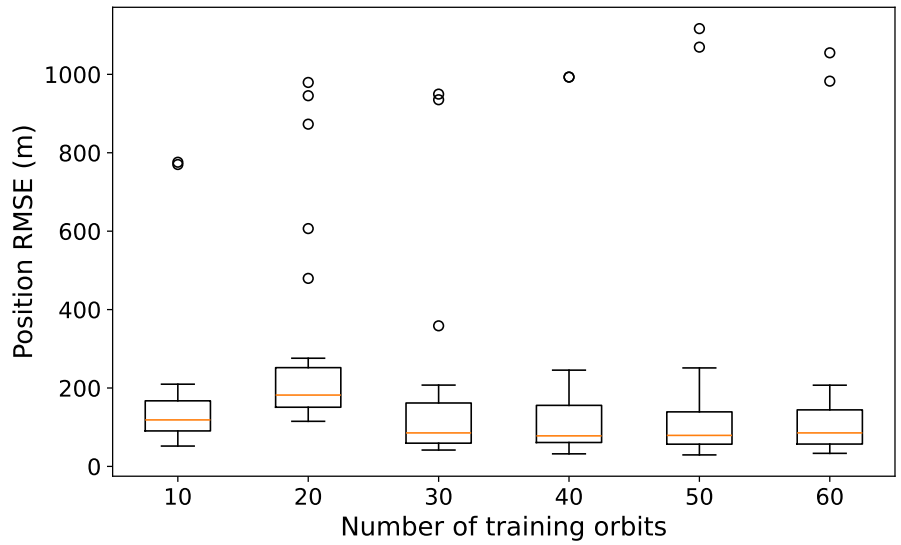


Figure 6.3: Box plot showing the distribution of RMSE values for validation orbits across different numbers of training orbits for PINN model.

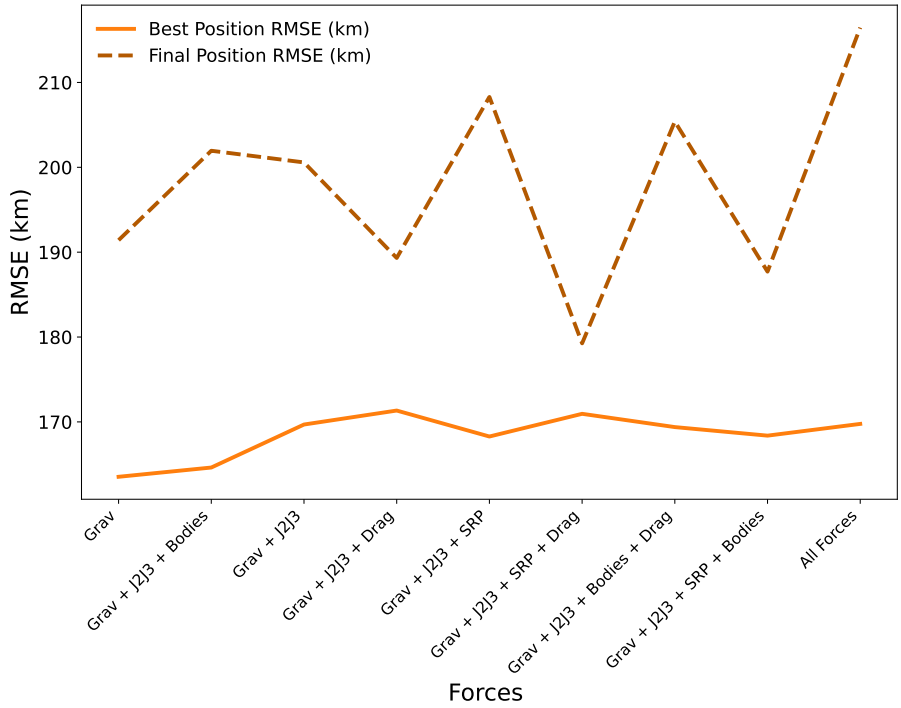


Figure 6.4: Evolution of the RMSE for the PINN model as the forces/corrections are iteratively added to the ODE. The forces that were iteratively added where: the gravitational force (Grav), the J_2 perturbation and J_3 perturbation (j2j3), solar radiation pressure (SRP), third bodies (Sun and Moon) (Bodies) and atmospheric drag (Drag). The gravitational force alone showed to the best RMSE.

The One-Cycle learning rate scheduler [57] adjusts the learning rate in two distinct phases. Initially, it increases from a low starting value to a defined peak, allowing the model to explore the loss surface. It then decreases sharply to a much smaller final value, for a

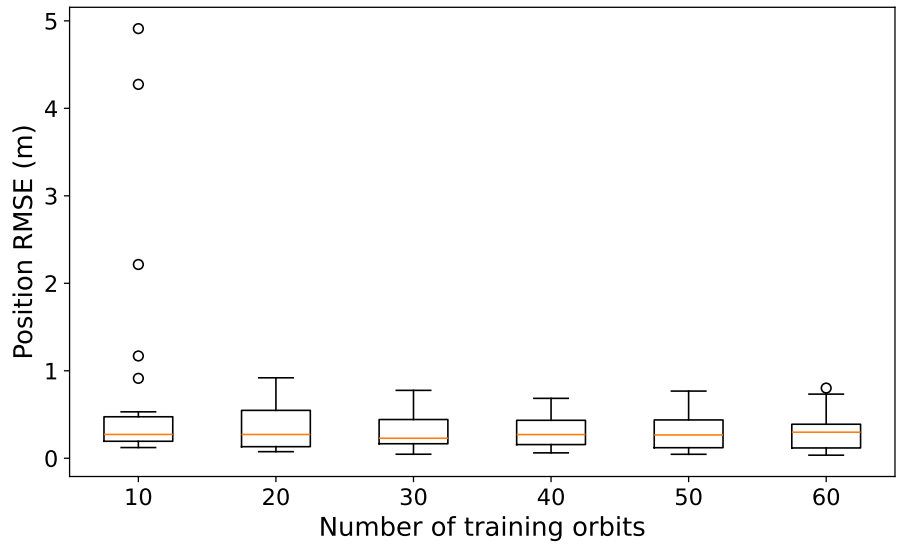


Figure 6.5: Box plot showing the distribution of RMSE values for validation orbits across different numbers of training orbits for NODE model.

more precise convergence. This has been shown to improve training speed by allowing the model to escape shallow local minima and saddle points early in training.

Physics Informed Neural Network The PINN model was the most difficult to optimise on the validation data, often encountering gradient issues, with extremely large gradients preventing the model from learning. For the collection points, a cumulative learning strategy was used, starting with no collocation points and gradually increasing them to 100 during training. This approach allowed the model to first learn the data patterns before being constrained by the physics, leading to better overall performance. The number of collocation points was also increase to see which would benefit the model more. Figure B.1 shows that over 100 collocation points did not lead to improvements, and as such it was kept at 100.

The final PINN architecture consisted of 5 hidden layers with 256 neurons each, using the GELU activation function. The Tanh activation function resulted in exploding gradients. Training was performed for up to 25,000 epochs with the AdamW optimizer and a cosine annealing schedule with linear warm-up.

This combination of scheduler is similar to how the One-Cycle learning rate scheduler works. However, it increases the learning rate linearly during the warm-up phase, and then decreases it following a cosine curve. The One-Cycle scheduler was tested and did not lead to improvements.

The loss function combined data fidelity, initial condition enforcement, and physics residual terms, with weights $\lambda_{\text{Data}} = 4.0$, $\lambda_{\text{IC}} = 3.0$, and $\lambda_{\text{ODE}} = 1.0$. Physics enforcement was introduced after 7000 epochs and ramped up smoothly to avoid destabilising early training. Although the PINN model served as a baseline for the NODE in this work, the training required significantly more care and effort to achieve stable performance.

Neural Ordinary Differential Equation Lastly, the NODE architecture had 3 layers, each with 256 neurons and Tanh activations, together with a dropout rate of 0.05. Training was performed for 1000 epochs with the Adam optimizer and a One-Cycle learning rate scheduler, the same as the MLP. Numerical integration of the latent ODE was tested with several different ODE solvers: RK4, Euler (with different step sizes), Explicit Adams, Implicit Adams, Midpoint, Dopri5, Heun2, and Heun3, as shown in Figure 6.6. The details of these ODEs can be found in Table 5.1.
The best-performing solver was Heun3 with a step size of 0.066, achieving a position RMSE of 0.282 km. The second, third, and fourth best solvers all had very similar RMSE values: Implicit Adams (0.288 km), RK4 with a step size of 0.066 (0.292 km), and Euler with a step size of 0.01 (0.292 km). The worst results came from the Euler solver with a step size of 0.06, which achieved a position RMSE of 116.285 km. This value is not shown in the plot to allow for a clearer visualisation of the trade-off between training time and position RMSE. The only solver that did not converge was Explicit Adams, likely due to its limited stability region.

The step size of the ODE solver was chosen to match the training data, which for each orbit corresponds to 96 points evenly spaced from 0 to 2π. This gives a step size of approximately 0.066. Low order methods such as Euler are more sensitive to step size, which explains their poor performance of 116 km when using a higher step size.
In general, higher order solvers achieve greater accuracy with fewer steps but require more function evaluations per step. Adaptive solvers such as Dopri5, or variable order solvers such as Adams methods, can take many additional internal steps to satisfy error tolerances, which explains their significantly higher computational cost during training. In the case of Implicit Adams, the tolerances were set to be a relative tolerance of 10^{-6} and an absolute tolerance of 10^{-8}, whereas for Dopri5 they were set to be 10^{-4} and 10^{-4} respectively. These values set how precisely the solver must approximate the trajectory at each integration step. The use of different tolerances for these ODEs solvers was necessary because Dopri5 is a higher order solver that takes many internal steps, and with the same tolerances as Implicit Adams, the training time would have been extremely long.

Initially, the assumption was that solvers with higher error could perform similarly to those with lower RMSE if trained for longer, however, this was not the case. As shown in Figure 6.7, all solvers reach a performance plateau.
Additional physical forces were incrementally incorporated into the NODE's ODE function, following the same iterative approach used for the PINN. Initially, training the NODE without any external accelerations, including no gravitational force, resulted in an unstable NN that failed to learn, with RMSE consistently remaining around 1500 km. Introducing the central gravitational force led to an improvement, reducing the RMSE to approximately

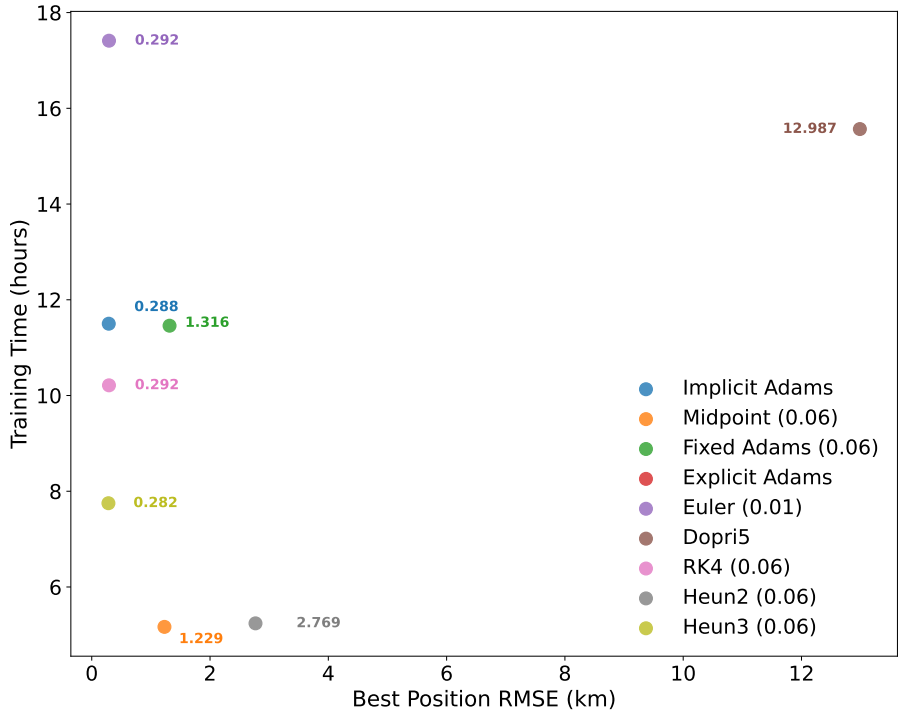


Figure 6.6: Figure showing how the postion RMSE changes when different numerical ordinary differential equarion solvers are used. The numbers next to each solver name indicate the step size used. A description of each solver can be found in Table 5.1. The best performing solver was Heun3 with a step size of 0.066 it achieved a position RMSE of 0.282 km. The second best was implicit adams, which achieved a position RMSE of 0.288 km. The explicit adams solver did not converge, and as such appears in the legend but not in the plot.

4.9 km. Incorporating the J_2 and J_3 perturbations further reduced the RMSE to 0.287 km. In contrast, including third-body perturbations (from the Moon and Sun) increased the RMSE to 1.4 km, although this was the only additional force that allowed the model to train at all. Both solar radiation pressure and atmospheric drag caused the NODE to diverge.

The final ODE used in the NODE model included the central gravitational force along with the J_2 and J_3 perturbations, and was solved using the Heun3 solver with a step size of 0.066.

6.2 Results

This section presents the results of each model, for the architectures described in the section above, starting with the simplest one, the MLP, followed by the PINN and finally the NODE. The results are calculated over a test set of 26 unseen satellite orbits.

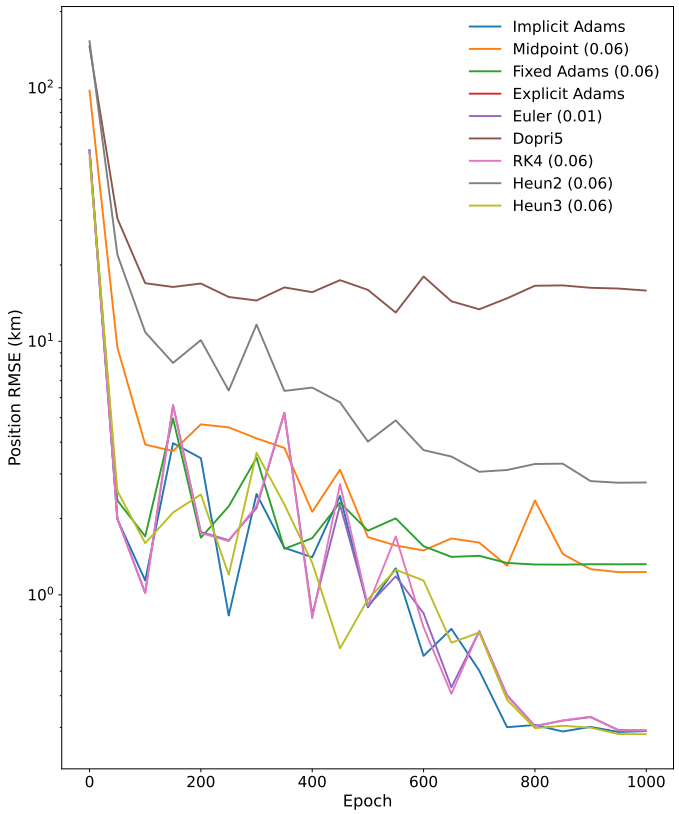


Figure 6.7: Evolution of the Position RMSE as the number of epochs progress for different solvers. The best performing solver reaches a plateau after 950 epochs, obtaining a position RMSE of 0.288km.

6.2.1 Multilayer Perceptron

The first model to be evaluated was a standard feedforward NN, the MLP. Its performance across the test orbits is summarised in Figure 6.8, which shows the distribution of position RMSE values. A detailed summary of its performance is presented in Table 6.2, along with its PINN and NODE counterparts. The performance of the model on each test orbit is shown in Table 6.1.

Overall, the mean position RMSE across the 26 satellites is 200.56 km with a standard deviation of 259.7 km, while the velocity errors average 0.214 km/s with a standard deviation of 0.279 km/s. The high standard deviation indicates significant variability in performance across different orbits, as is seen in Figure 6.8 and in Table 6.1.

The position RMSEs for the individual components are of 195.9 km for the x component, 213.7 km for the y component, and 172.0 km for the z component, whereas the velocity RMSEs are 0.213 km/s, 0.224 km/s, and 0.188 km/s for the x, y, and z components, respectively. In terms of dominant error contribution, the x position accounts for 61.5% of the error, while the y and z components each contribute 19.2%. For the velocity, the x component dominates with 57.7%, followed by 34.6% from the y component and 7.7% from the z component. The best orbit (ID 47378), seen in Figure B.2, reached a position

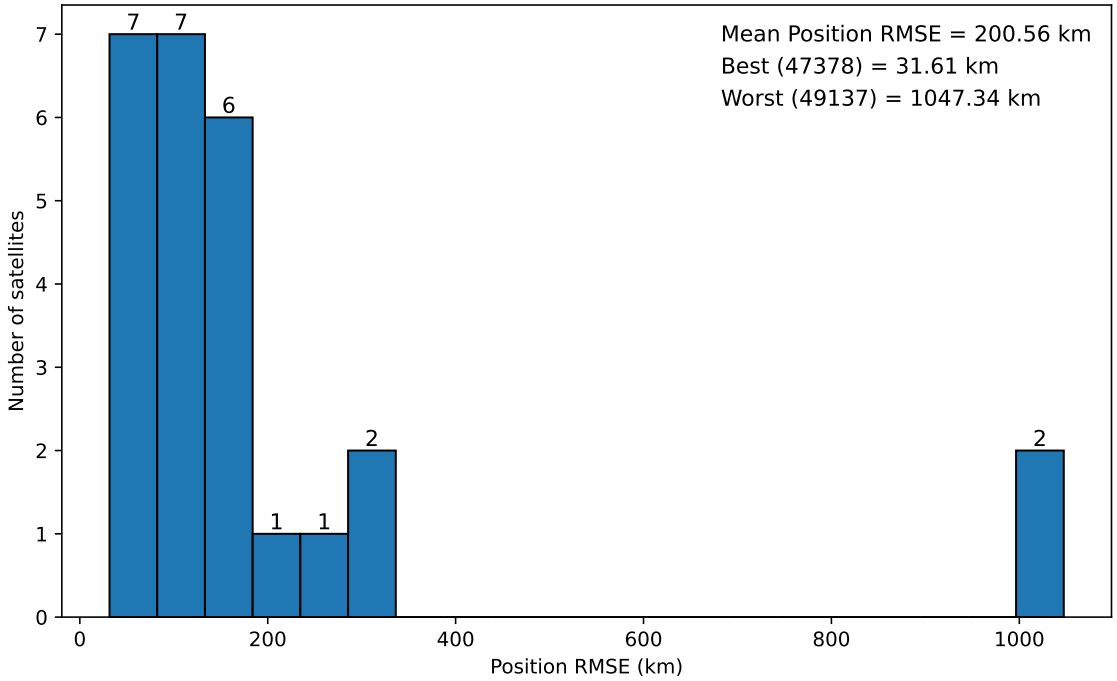


Figure 6.8: Histogram of the RMSE values for the MLP model on the test set. The mean RMSE is 200.56 km.

RMSE of only 31.6 km and a velocity RMSE of 0.038 km/s. The evolution of the error for each component over time is shown is Figure 6.9 for a better understanding of how the error accumulates. In contrast, the worst case (ID 49137), seen in Figure B.3, and associated error evolution Figure B.8, showed 1047.3 km in position error and 1.12 km/s in velocity error. The correlation between position and velocity errors was high, with an R^2 value of 0.9991, which indicates a very strong correlation between the two metrics.

Adding more epochs does not improve this model, as its performance has already plateaued. This can be seen in the evolution of the training and validation losses shown in Figure B.11. The validation loss decreases early in training but then stagnates, while the training loss continues to fall. After roughly epoch 400, the network no longer extracts useful patterns from the data and instead overfits, as indicated by the widening gap between training and validation losses.

The high RMSE values were expected for this model. Although the MLP can approximate nonlinear relationships, the absence of physical constraints makes it prone to overfitting and poor extrapolation outside the training domain. This can be seen by looking at the spread of the RMSE values, which range from 31.6 km to 1040.50 km in Figure 6.8. This is likely because the model did not train with orbits that are similar to the worst performing test orbits, and because the model does not have any knowledge of the underlying physics, it cannot generalise well to unseen scenarios.

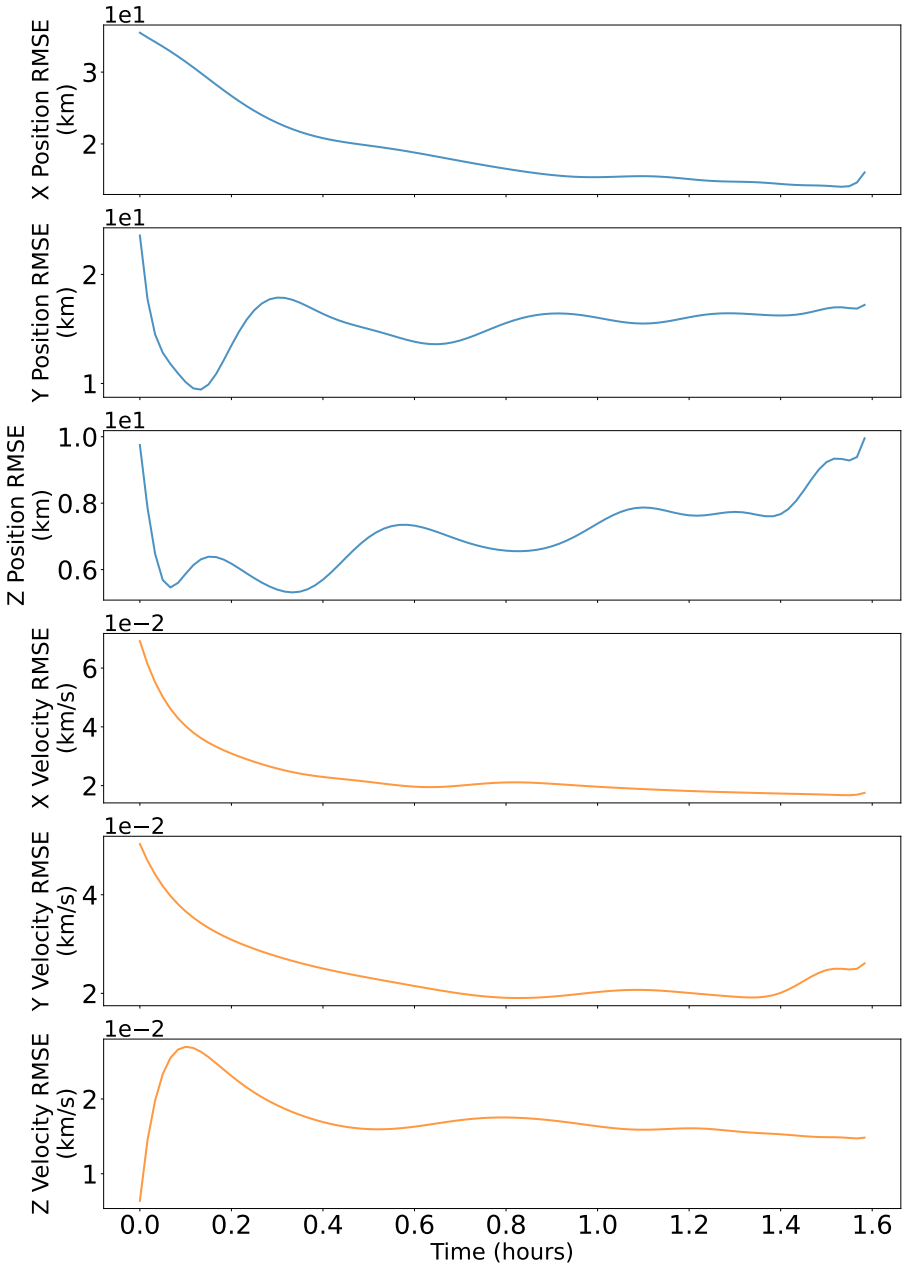


Figure 6.9: Evolution of the RMSE for the best performing test orbit using the MLP model.

6.2.2 Physics Informed Neural Network

Similarly to the MLP, the PINN has a broad distribution of high RMSE values, as seen in Figure 6.10.

The mean RMSE is smaller, compared to the MLP, with an error of 176 km. Figure B.4 shows the predicted orbit versus the true orbit for the best performing test orbit using the PINN model, which belongs to satellite 50834, and has a position RMSE of 36.4 km and velocity RMSE of 0.032 km/s. In contrast, Figure B.5 shows the worst performing test orbit using the PINN model, which belongs to satellite 49137 and has a position RMSE of 1040.12 km and a velocity RMSE of 1.09 km/s. Similarly to the MLP, the model

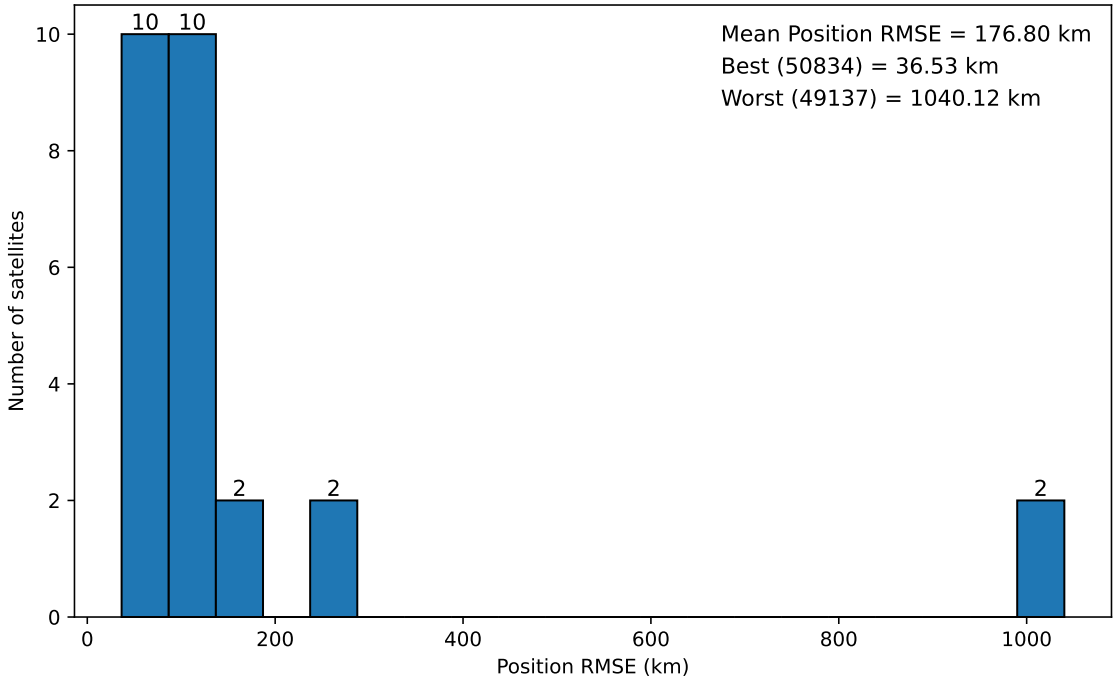


Figure 6.10: Histogram of the RMSE values for the PINN model on the test set. The mean position RMSE is 176 km.

still fails to generalize reliably across all test orbits, despite having information about the physics. This makes sense because the physics is enforced only during training as a form of regularization, as such the model can produce predictions during testing and validation that are inconsistent with the underlying dynamics.

In this model, like in the MLP, the dominant source of error across most satellites is the x position, which accounts for 50% of the total error contribution. The y position follows with 26.9%, while the z position contributes 23.1%. On average, the position RMSE is 176.8 km with a standard deviation of 259.0 km, while the velocity RMSE is 0.186 km/s with a standard deviation of 0.272 km/s. The component wise RMSE are: mean values of 181.5 km for the x position, 182.2 km for the y position, and 157.2 km for the z position, while the velocity RMSE are 0.178 km/s for the x component, 0.197 km/s for the y component, and 0.175 km/s for the z component. For the velocity error distribution, the y component is the most dominant with 46.2%, followed by the x component with 42.3% and the z component with 11.5%. The correlation between position and velocity errors is high, with an R^2 value of 0.9992.

The PINN also plateaus in validation performance early in training, as shown in Figure B.12. This is followed by an increase in validation loss and a subsequent slight decrease. Although the training loss continues to decrease, it remains above the validation loss for most of the training process.

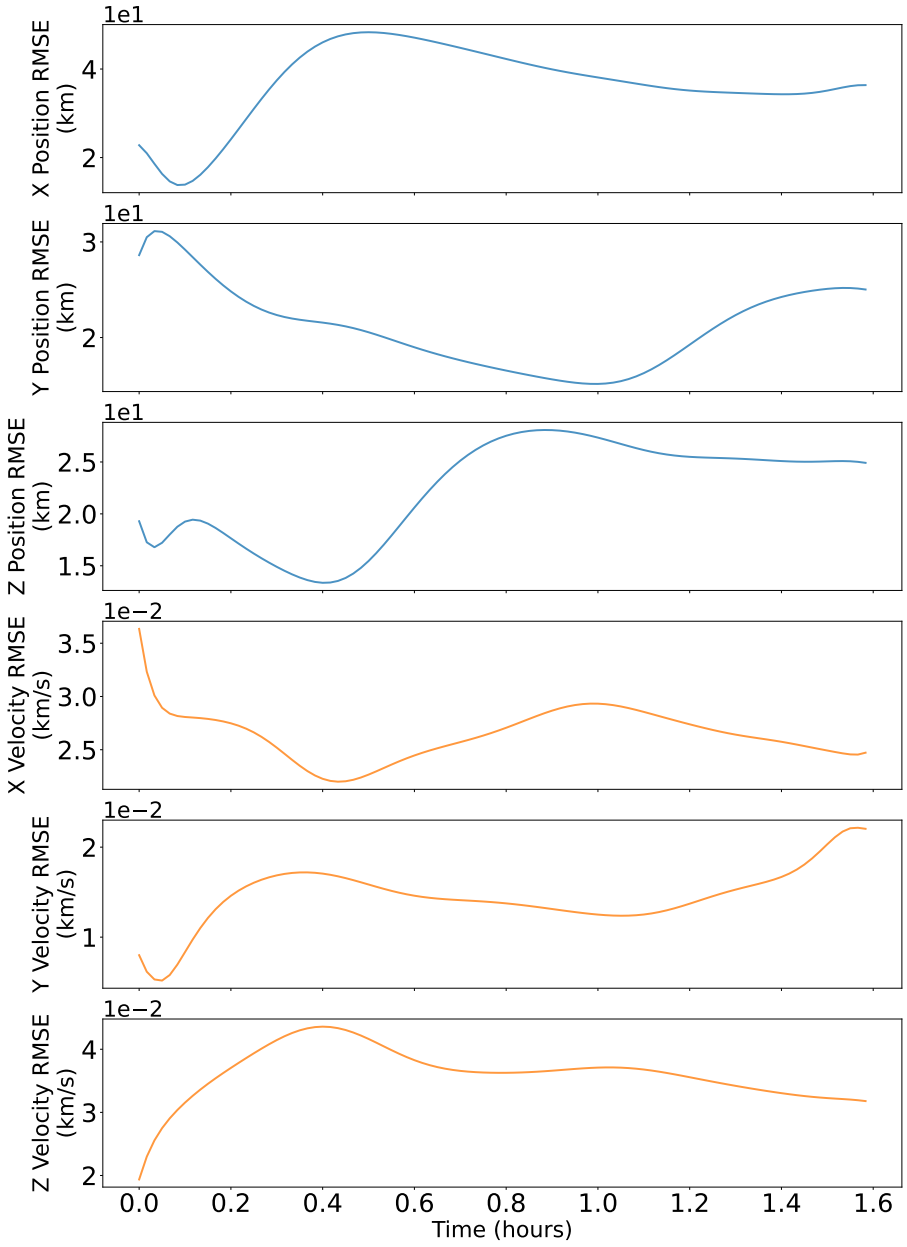


Figure 6.11: Evolution of the RMSE for the worst performing test orbit using the PINN model.

6.2.3 Neural Ordinary Differential Equation

The NODE model outperforms both the MLP and the PINN. The RMSE values across the test orbits are shown in Figure 6.12. Not only are the RMSE values significantly lower, but the spread of the RMSE values is also much tighter, with all satellites having errors below 1 km.

For the NODE model, the mean position RMSE across the 26 satellites is only 0.25 km with a standard deviation of 0.14 km, while the velocity errors average 2.37×10^{-4} km/s

with a spread of 1.30×10^{-4} km/s. The position RMSE broken down by component has mean values of 0.26 km in the x direction, 0.23 km in the y direction, and 0.24 km in the z direction. The velocity errors show similar consistency, with average RMSE of 2.43×10^{-4}, 2.25×10^{-4}, and 2.37×10^{-4} km/s for the x, y, and z components, respectively. In terms of dominant error contribution, similarly to the MLP and the PINN, the x position errors are most frequently the largest, accounting for 38.5% of the satellites, while the y and z components each dominate in 30.8% of the cases. For velocity, the x component again dominates with 53.8%, followed by 26.9% from the y component and 19.2% from the z component.

The best orbit (ID 48558), shown in Figure B.6, achieved a position RMSE of only 0.064 km, whereas the worst case (ID 48466), seen in Figure B.7, reached 0.60 km in position error. Overall, the correlation between position and velocity errors exceeds an R^2 of 0.977, the smallest of the three models.

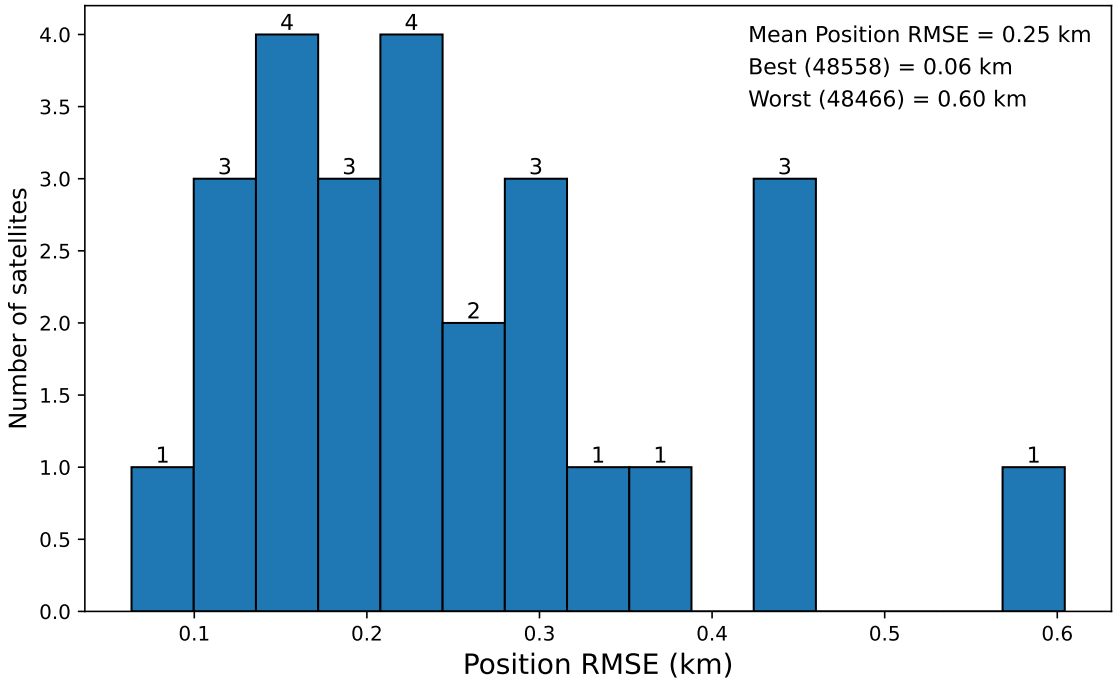


Figure 6.12: Histogram of the RMSE values for the NODE model on the test set. The mean RMSE is 0.25 km.

Lastly, the NODE decreases smoothly in both training and validation losses throughout training, as shown in Figure B.13. The training loss remains slightly above the validation loss, indicating that the model is not overfitting. Nonetheless, the validation loss reaches a plateau after approximately 800 epochs.

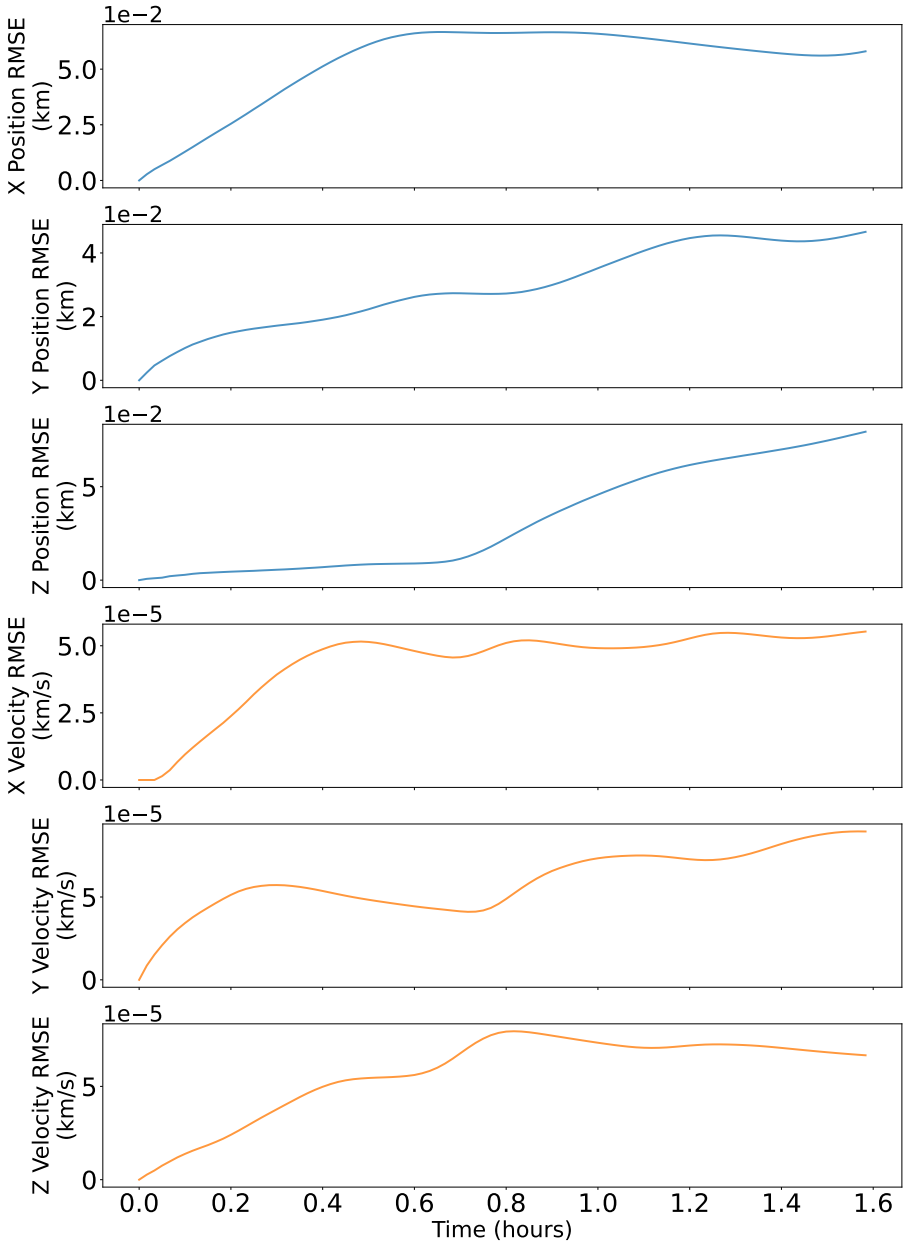


Figure 6.13: Evolution of the RMSE for the best performing test orbit using the NODE model.

6.3 Comparison of Models

The MLP, as the simplest of the three models, relies entirely on data to approximate the trajectory, providing a useful benchmark for evaluating the added value of incorporating physical knowledge into the learning process. It serves to show the challenges of predicting complex orbital dynamics when only data-driven representations are used.
As shown in Table 6.2, the MLP has a slightly higher mean RMSE than the PINN. This shows that adding physical knowledge to the model improves performance, however, the improvement is not as significant as one might expect. This could be due to the challenges

Table 6.1: Position RMSE (km) across 26 test satellites for the three models (MLP, PINN and NODE). For each model the best (green) and worst (red) result is highlighted in bold.

Satellite ID	MLP	PINN	NODE
44725	248.0	132.8	0.25
45412	153.3	69.2	0.45
45590	141.2	156.5	0.22
45674	57.5	108.6	0.13
46353	101.4	92.8	0.11
47371	83.5	91.6	0.45
47378	**31.6**	154.2	0.22
47628	40.8	124.3	0.34
47836	159.4	68.4	0.44
47904	329.6	264.7	0.35
48277	108.6	89.3	0.18
48292	136.1	70.5	0.19
48295	76.9	59.1	0.15
48382	39.6	123.6	0.21
48466	136.5	46.8	**0.60**
48482	201.8	97.2	0.14
48558	113.6	74.4	**0.06**
48659	107.5	89.6	0.28
48671	132.4	60.9	0.24
49136	1040.5	1028.6	0.21
49137	**1047.3**	**1040.1**	0.16
49763	322.6	280.0	0.29
49752	55.8	105.4	0.15
50199	129.6	70.5	0.28
50815	144.8	61.3	0.11
50834	74.5	**36.5**	0.30

associated with training PINNs, such as the complexity of the loss landscape and the difficulty in balancing the data and physics terms in the loss function.

Minimising the PINN loss remains notoriously difficult. Even though the problem is formulated correctly and the architecture of the model is suitable, the shape of the loss landscape makes training PINNs extremely hard. Computing the derivatives required by the ODE residual amplifies numerical noise during backpropagation, often causing gradients to explode or vanish. Additionally, adding the differential operators in the loss leads to a Hessian whose eigenvalues span many orders of magnitude. This results in the loss landscape being ill-conditioned, with very large valleys and very flat surfaces, making optimizers stall in saddle points. On top of this, the non-convexity of the loss adds to the lack of guidance in finding the minimum [53] [30]. This difficulty in optimisation can be seen by comparing the loss figures of the MLP and the PINN, in Figures B.11 and B.12. The MLP, while still somewhat noisy, shows a clear downward trend in training loss, whereas the PINN's training loss is much more erratic.

While the MLP avoids the numerical difficulties associated with computing derivatives in the loss function, it suffers from a different set of limitations. Since it relies purely on data and does not incorporate any physical constraints, it lacks the ability to generalise beyond the range of its training data and can produce predictions that violate the underlying laws of motion. However, despite the fact that it has no physics prior, the MLP was able to outperform the PINN in 6 of the 26 test orbits, see Table 6.1.
Another challenge with PINNs arises from the way the physics is enforced. The collocation points are the only points where the physics is enforced, which is then used in the loss. This means that only during training does the physics play a role as a regularization term. During evaluation and testing, the model can predict anything, even physically inconsistent results. Out of the three models, the PINN is the hardest one to optimise.

The NODE model avoids the complex loss landscape of PINNs by learning the underlying dynamics of the system without requiring explicit derivatives in the loss function. Another advantage of the NODE is that the solver enforces the velocity as the derivative of position, allowing the model to predict only the acceleration. This avoids the unstable double integration required in PINNs, allowing the model to predict only three values instead of six. The problems it faces are instead related to the numerical integration of the ODEs, which can be sensitive to the choice of solver. To achieve high accuracy, it may require small time steps, increasing computational cost. However, regardless of the ODE solver chosen, the NODE model consistently outperforms both the MLP and PINN in terms of accuracy and generalization. The model is able to provide results below 1 km in position RMSE, with some orbits only having a few metres of error.

In terms of implementation, the MLP is the most straightforward, requiring only input normalisation. The PINN introduces additional complexity, as it incorporates physics-based constraints into its architecture, but its simplicity is overshadowed by the multiple challenges it faces during training. The NODE had a simpler formulation since it does not need the explicit ODE function. It requires the use of a numerical ODE solver to integrate the learned dynamics.

The results show that incorporating physical knowledge improves performance but it is not the sole factor. The NODE model outperforms both the MLP and PINN, achieving the lowest RMSE values across the test orbits. This suggests that learning the underlying dynamics in a continuous-time framework, and allowing it to be applied during training and testing, is more effective than simply adding physics as a regularisation term in the loss function. Lastly, a table comparing the results of all the tested satellites for the three models is shown in Table 6.1. Looking at the table, we see that while both the MLP and PINN struggled the most with the same satellites, 49136 and 49137 (shown in red), the NODE had no problem with those orbits.

Interestingly, all three models struggled most with the x component of the position, despite their differing overall accuracies. A possible explanation for this could be that the x component is more sensitive to perturbations and external forces, leading to more variability between orbits. Additionally, the NODE model had the weakest correlation between position and velocity errors, with an R^2 value exceeding 0.977, whereas the MLP and PINN had R^2 values of 0.9991 and 0.9992 respectively.

6.4 Conclusion

Across the 26 satellites, the NODE consistently achieved the lowest position errors, clearly outperforming both the purely data-driven MLP and the physics-regularised PINN, as summarised in Table 6.2. The MLP and PINN perform similarly, with very large standard deviations indicating inconsistency and unreliability across different orbits. In contrast, the NODE is not only more accurate but also more consistent, with errors tightly concentrated around the mean.

Table 6.2: Summary of position RMSE (km) across 26 test satellites for each model.

Model	**Best RMSE (km)**	**Worst RMSE (km)**	**Mean RMSE (km)**	**Std. Dev. (km)**
MLP	31.6	1047.3	200.6	259.7
PINN	36.5	1040.1	176.8	259.0
NODE	0.06	0.60	0.25	0.14

The three models can be contrasted as follows:

- **MLP:** Provides a simple baseline but lacks any physical constraints. Errors range from tens to over a thousand kilometres, reflecting poor generalisation beyond the training set.

- **PINN:** Incorporating physics lowers the mean error relative to the MLP, but optimisation is fragile due to exploding gradients and an ill-conditioned loss landscape. Because physics is only enforced at collocation points, predictions at test time can still deviate significantly from the underlying dynamics.

- **NODE:** Achieves both accuracy and consistency, with all orbits predicted within 1 km. Training is more stable than for the PINN, since the solver enforces velocity as the derivative of position. The model only needs to learn acceleration, reducing the output dimension from six to three.

Taken together, these results show that embedding dynamics through numerical integration, rather than via differential penalties in the loss, provides a more effective and reliable strategy for satellite trajectory prediction.

6.5 Neural Ordinary Differential Equations on Real-World Data

The initial analysis of the Starlink dataset in Chapter 4 revealed that the data was highly error-prone. Daily manoeuvres, an unknown and varying propagator, unmodelled physical forces acting on the satellites, and internal estimation errors all contribute to its unreliability. For this reason, the NODE model was trained and evaluated using simulated data generated with Orekit. Nevertheless, given the strong performance of the NODE on the Orekit dataset, it was of interest to assess its capability on the Starlink data and determine whether it could handle such noisy conditions.

To this end, the same NODE architecture and configuration used for the Orekit experiments was trained directly on the Starlink orbits. The Starlink orbits themselves were divided into two different types: decaying orbits and stable orbits. This was done by looking at the semi-major axis of each orbit, the same way as in Chapter 5. If the semi-major axis remained relatively the same [1] during the collection period it was considered stable. The orbits that had a decreasing semi-major axis and decayed after the collection of the data were classified as decaying orbits. This way, two models could be trained, one for each type of orbit, to understand if the NODE is able to learn under different orbital regimes.

6.5.1 Stable Orbits

Like in the previous experiments using NODEs, 26 test orbits were used to evaluate the model, while the remaining orbits were used for training and validation.
The model achieved an average position RMSE of 0.32 km, which is slightly higher than the 0.25 km obtained using Orekit. All errors remained under 1 km, with the best satellite achieving 0.07 km RMSE, while the worst satellite had a position RMSE of 0.88 km. The histogram of position RMSE for all test orbits is shown in Figure B.14, while the best and worst satellite trajectories are shown in Figure B.15 and Figure B.16.
In contrast to the Orekit results, the y component of the position was the hardest to model, with 50% of the satellites having the largest error in this component. The x component was the second hardest to model, with 46.7% of the satellites having the largest error in this component. The z component was the easiest to model, with only 3.3% of the satellites having the largest error in this component.

6.5.2 Deorbiting Orbits

As expected, the model trained with deorbiting orbits performed worse than the model trained with stable orbits. The average position RMSE was 0.50 km, less than double the error of the stable orbits. The best satellite achieved a position RMSE of 0.11 km, while the worst satellite had a position RMSE of 1.86 km, making this the only NODE model to have a satellite with an error above 1 km. The histogram of position RMSE for all test orbits is

[1] Semi-major axis values vary due to non-conservative forces and manoeuvres.

shown in Figure B.17, clearly showing that an outlier had the biggest error. Overall, this NODE is the one with the widest spread of position RMSE values. The best and worst satellite trajectories are shown in Figure B.18 and Figure B.19.
In this case, the y component of the position was again the hardest to model, with 40% of the satellites having the largest error in this component. The x and z components were equally hard to model, with 30% of the satellites having the largest error in each component.

6.5.3 Comparison

The results of the NODE on the Starlink stable orbits are summarised in Table 6.3. Comparing with the results obtained using Orekit data, the performance of the NODE is slightly worse, with an increase in average position RMSE from 0.25 km to 0.32 km on stable orbits. While the NODE can learn from the noisy Starlink data, its performance is hindered by the inherent errors and inconsistencies present in the observations. Nevertheless, the model still captures the overall orbital dynamics, as evidenced by the relatively low RMSE values across all satellites. This suggests that NODEs have potential for orbit prediction even in challenging real-world scenarios.

Table 6.3: Summary of NODE prediction errors on Starlink data, separated into stable and deorbiting orbit categories. Reported values correspond to position RMSE in km.

Type	**Best RMSE (km)**	**Worst RMSE (km)**	**Mean RMSE (km)**	**Std. Dev. (km)**
Stable	0.07	0.88	0.32	0.13
Deorbiting	0.11	1.86	0.50	0.41

The results on the deorbiting orbits were worse, which is likely due to the more complex dynamics involved in deorbiting, which are harder for the NODE to learn from noisy data. Starlink's own propagation of deorbiting satellites is also less accurate, which may have contributed to the higher errors.
This analysis also shows that when applied to real-world data, the NODE model should focus more on improving the prediction of the y component of the position, as it was the hardest to model in both stable and deorbiting orbits. This could be due to a specific force affecting more the y component, or simply because the data is noisier in this component.

7

Future Work

This dissertation focused on the development of NODEs for satellite trajectory prediction, and compared their performance with two other models: a standard MLP and a PINN. In addition, a brief analysis of Starlink's publicly available data was carried out to better understand the data collection process and the challenges associated with working with real-world satellite observations, highlighting its applicability for training NNs. Initially, it was thought that Starlink data would be too noisy and error prone to be useful for training, but the results show that NODEs can learn from this data and achieve reasonable accuracy in orbit prediction, be it for stable orbits or deorbiting satellites. The results that came from this dissertation serves as a foundation for further research in this area, with several avenues for future exploration.

An interesting direction for future work would be to extract acceleration profiles from real-world satellite trajectories and iteratively subtract known forces in order to isolate the residuals. Using the NODE, the NN output represents the learnt non-conservative acceleration acting on the satellite. By systematically removing contributions from well-characterised forces, such as atmospheric drag and solar radiation pressure, we can obtain a residual acceleration profile. This residual can then be modelled as a function of time, for example through sparse identification such as SINDy. Such an approach may provide new insights into non-conservative forces that are not explicitly accounted for *a priori*, thereby improving understanding of the dynamics influencing satellite motion.
Another promising avenue would be to investigate the robustness of the NODE architecture under various types of noise applied to the input data, to assess how well the model can generalize in the presence of observational errors.

Finally, a more in-depth analysis of the different ODEs solvers and their stability regions could be carried out to better understand their impact on the training and prediction performance of NODEs. It would be interesting to explore how different solvers affect the model's ability to learn and generalise from the training data, in the context of satellite trajectory prediction and other problems. Applying the different ODE solvers

with greater knowledge and understanding of how they work could help identify which methods are most appropriate for specific classes of problems

7.1 Conclusion

This research demonstrates the promise of NODEs for satellite trajectory prediction, showing that they can learn and reproduce the underlying orbital dynamics. Although PINNs seem like methods that would be well suited for this task, they were severely limited by their architecture.
NODEs performed well on both simulated data and error prone real-world observations from Starlink satellites, which included unmodelled forces, internal estimation errors, and covered both stable and deorbiting cases. This model shows great potential for further development and application in the field of orbit prediction.

From this research, we had the opportunity to present this work at the International Astronautical Congress (IAC) 2025 in Sydney, Australia, including an expanded analysis of Starlink satellites and their suitability as training data for the NODE model discussed in this dissertation [17]. This dissertation also contributed to a publication at NeurIPS 2025, focusing on the analysis of the Starlink data [44].

Bibliography

[1] M. R. Akella and K. T. Alfriend. "Probability of Collision Between Space Objects". en. In: *Journal of Guidance, Control, and Dynamics* 23.5 (2000-09), pp. 769–772. ISSN: 0731-5090, 1533-3884. DOI: 10.2514/2.4611 (cit. on p. 2).

[2] A. Andrea and L. Maisonobe. "Automatic differentiation for propagation of orbit uncertainties on Orekit". In: 2016-11 (cit. on p. 33).

[3] J. Aristoff and A. Poore. "Implicit Runge-Kutta Methods for Orbit Propagation". en. In: *AIAA/AAS Astrodynamics Specialist Conference*. Minneapolis, Minnesota: American Institute of Aeronautics and Astronautics, 2012-08. ISBN: 978-1-62410-182-3. DOI: 10.2514/6.2012-4880 (cit. on pp. 6, 7).

[4] B. Bastida Virgili et al. "CREAM—ESA's Proposal for Collision Risk Estimation and Automated Mitigation". In: *First International Orbital Debris Conference*. Vol. 2109. 2019, p. 6031 (cit. on p. 2).

[5] M. M. Berry and L. M. Healy. "Implementation of Gauss-Jackson Integration for Orbit Propagation". In: *The Journal of the Astronautical Sciences* 52.3 (2004-09), pp. 331–357. ISSN: 2195-0571. DOI: 10.1007/BF03546367 (cit. on p. 6).

[6] J. C. Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016 (cit. on p. 42).

[7] F. Caldas and C. Soares. "Machine learning in orbit estimation: A survey". In: *Acta Astronautica* 220 (2024-07), pp. 97–107. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2024.03.072 (cit. on pp. 2, 4, 6).

[8] J. S. Catulo, C. Soares, and M. Guimarães. *Predicting the Probability of Collision of a Satellite with Space Debris: A Bayesian Machine Learning Approach*. 2023. arXiv: 2311.10633 [cs.LG] (cit. on p. 2).

[9] B. Cazabonne et al. "A Semi-analytical Approach for Orbit Determination based on Extended Kalman Filter". In: 2021-08 (cit. on p. 33).

[10] R. T. Chen et al. "Neural ordinary differential equations". In: *Advances in neural information processing systems* 31 (2018) (cit. on pp. 4, 13, 14, 16).

[11] G. G. Chrysos et al. "Deep Polynomial Neural Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. ISSN: 1939-3539. DOI: 10.1109/tpami.2021.3058891 (cit. on p. 18).

[12] I.-A. S. D. C. Committee. *Space Debris Mitigation Guidelines*. Tech. rep. IADC-02-01, Revision 3. IADC, 2021 (cit. on p. 30).

[13] D. Conkey and M. Zielinski. "Assessing Performance Characteristics of the SGP4-XP Propagation Algorithm". In: *Advanced Maui Optical and Space Surveillance Technologies Conference, Maui, HI, USA*. 2022 (cit. on p. 8).

[14] Z. Dapeng and S. Hongxin. "STUDY ON THE ORBITAL CHARACTERISTICS OF STARLINK SATELLITES BASED ON PRECISE EPHEMERIS". In: *MECHANICS IN ENGINEERING* 44.6 (2022), pp. 1268–1278. ISSN: 1000-0879. DOI: 10.6052/1000-0879-22-355 (cit. on p. 26).

[15] I. E. Dawoodjee. "Establishing a regression baseline for predicting satellite motion". In: 2021 (cit. on p. 9).

[16] E. Dupont, A. Doucet, and Y. W. Teh. "Augmented neural odes". In: *Advances in neural information processing systems* 32 (2019) (cit. on p. 18).

[17] K. Dyreby, F. Caldas, and C. Soares. *Analyzing Data Quality and Decay in Mega-Constellations: A Physics-Informed Machine Learning Approach*. 2025. arXiv: 2510.11242 [astro-ph.EP] (cit. on p. 64).

[18] ESA Space Debris Office. *ESA's Annual Space Environment Report*. Tech. rep. Darmstadt, Germany: European Space Agency, 2023 (cit. on pp. 2, 3, 30).

[19] European Space Agency. *Neural Ordinary Differential Equations (NODEs) for Space Applications*. Accessed: 10-Feb-2025. 2025. URL: https://www.esa.int/gsp/ACT/projects/neural_ode/ (cit. on p. 10).

[20] European Space Agency. *Space debris by the numbers*. https://www.esa.int/Space_Safety/Space_Debris/Space_debris_by_the_numbers. Accessed: 2024-05-28 (cit. on p. 1).

[21] R. Ferreira, C. Soares, and M. Guimarães. *Probability of Collision of satellites and space debris for short-term encounters: Rederivation and fast-to-compute upper and lower bounds*. 2023. arXiv: 2311.08978 [astro-ph.EP] (cit. on p. 2).

[22] C. Fronk and L. Petzold. "Interpretable polynomial neural ordinary differential equations". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 33.4 (2023-04). ISSN: 1089-7682. DOI: 10.1063/5.0130803 (cit. on p. 18).

[23] C. Fronk and L. Petzold. "Training stiff neural ordinary differential equations with implicit single-step methods". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 34.12 (2024) (cit. on pp. 7, 16, 17).

[24] J. Funenga et al. *Finding Real-World Orbital Motion Laws from Data*. 2023. arXiv: 2311.10012 [cs.LG] (cit. on p. 9).

[25] E. Hairer, G. Wanner, and S. P. Nørsett. *Solving ordinary differential equations I: Nonstiff problems*. Springer, 1993 (cit. on p. 42).

[26] D. Hendrycks and K. Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG] (cit. on p. 44).

[27] F. R. Hoots and R. L. Roehrich. *Spacetrack report number 3: Models for propagation of NORAD element sets*. 1980 (cit. on pp. 7, 8).

[28] B. A. Jones and R. L. Anderson. "A survey of symplectic and collocation integration methods for orbit propagation". In: *AAS/AIAA Spaceflight Mechanics Conference*. 2012, pp. 1–20 (cit. on p. 6).

[29] D. J. Kessler. "Collision Frequency of Artificial Satellites: The Creation of a Debris Belt". In: *Journal of Geophysical Research* 83.A6 (1978), pp. 2637–2646 (cit. on p. 1).

[30] A. Krishnapriyan et al. "Characterizing possible failure modes in physics-informed neural networks". In: *Advances in neural information processing systems* 34 (2021), pp. 26548–26560 (cit. on p. 58).

[31] I. Lagaris, A. Likas, and D. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations". In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 987–1000. ISSN: 1045-9227. DOI: 10.1109/72.712178 (cit. on p. 9).

[32] K. Lee, N. Trask, and P. Stinis. "Structure-preserving sparse identification of nonlinear dynamics for data-driven modeling". In: *Mathematical and Scientific Machine Learning*. PMLR. 2022, pp. 65–80 (cit. on p. 18).

[33] A. Liu et al. "Maneuver strategies of Starlink satellite based on SpaceX-released ephemeris". In: *Advances in Space Research* 74.7 (2024), pp. 3157–3169. ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2024.06.038 (cit. on pp. 24, 25).

[34] Z. Y.-C. Liu et al. "Improved orbital propagator integrated with sgp4 and machine learning". In: (2021) (cit. on p. 8).

[35] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[36] Y.-z. Luo and Z. Yang. "A review of uncertainty propagation in orbital mechanics". In: *Progress in Aerospace Sciences* 89 (2017), pp. 23–39. ISSN: 0376-0421. DOI: https://doi.org/10.1016/j.paerosci.2016.12.002 (cit. on p. 7).

[37] L. Maisonobe et al. "Multi-satellites Precise Orbit Determination, an adaptable open-source implementation". In: *2018 SpaceOps Conference*. DOI: 10.2514/6.2018-2622 (cit. on p. 33).

[38] maisonobe et al. *CS-SI/Orekit: 12.2.1*. Version 12.2.1. 2024-12 (cit. on p. 32).

[39] M. Manzi and M. Vasile. "Discovering Unmodeled Components in Astrodynamics with Symbolic Regression". In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. 2020, pp. 1–7. DOI: 10.1109/CEC48606.2020.9185534 (cit. on p. 9).

[40] M. Manzi and M. Vasile. "Orbital anomaly reconstruction using deep symbolic regression". In: *71st International Astronautical Congress*. 2020 (cit. on p. 9).

[41] K. Merz et al. "Current collision avoidance service by ESA's Space Debris Office". In: *7th European Conference on Space Debris*. 2017, p. 219 (cit. on pp. 1, 2).

[42] O. Montenbruck. "Numerical integration methods for orbital motion". en. In: *Celest. Mech. Dyn. Astron.* 53.1 (1992) (cit. on p. 6).

[43] V. U. J. Nwankwo et al. "Atmospheric drag effects on modelled low Earth orbit (LEO) satellites during the July 2000 Bastille Day event in contrast to an interval of geomagnetically quiet conditions". en. In: *Annales Geophysicae* 39.3 (2021-05), pp. 397–412. ISSN: 1432-0576. DOI: 10.5194/angeo-39-397-2021. (Visited on 2024-05-24) (cit. on p. 3).

[44] A. Oliveira et al. "OrbitZoo: Real Orbital Systems Challenges for Reinforcement Learning". In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025 (cit. on p. 64).

[45] T. Paulet and B. Cazabonne. "An Open-Source Solution for TLE Based Orbit Determination". In: *Proceedings of the 8th European Conference on Space Debris*. ESA Space Debris Office Publication. Darmstadt, Germany: ESA Space Debris Office, 2021-04 (cit. on p. 33).

[46] T. Payne et al. "Improvements to the SGP4 propagator (SGP4XP)". In: *Proceedings of the 2022 AMOS Technical Conference*. 2022 (cit. on p. 8).

[47] H. Peng and X. Bai. "Artificial Neural Network–Based Machine Learning Approach to Improve Orbit Prediction Accuracy". In: *Journal of Spacecraft and Rockets* 55 (2018-06), pp. 1–13. DOI: 10.2514/1.A34171 (cit. on p. 8).

[48] H. Peng and X. Bai. "Comparative evaluation of three machine learning algorithms on improving orbit prediction accuracy". en. In: *Astrodynamics* 3.4 (2019-12), pp. 325–343. ISSN: 2522-008X, 2522-0098. DOI: 10.1007/s42064-018-0055-4 (cit. on p. 8).

[49] H. Peng and X. Bai. "Gaussian Processes for improving orbit prediction accuracy". In: *Acta Astronautica* 161 (2019-05). DOI: 10.1016/j.actaastro.2019.05.014 (cit. on p. 8).

[50] H. Peng and X. Bai. "Improving orbit prediction accuracy through supervised machine learning". In: *Advances in space research* 61.10 (2018), pp. 2628–2646 (cit. on p. 8).

[51] M. Raissi, P. Perdikaris, and G. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045 (cit. on pp. 9, 19, 40).

[52] M. Raissi, P. Perdikaris, and G. E. Karniadakis. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. 2017. arXiv: 1711.10566 [cs.AI] (cit. on p. 19).

[53] P. Rathore et al. *Challenges in Training PINNs: A Loss Landscape Perspective*. 2024. arXiv: 2402.01868 [cs.LG] (cit. on p. 58).

[54] J. F. San-Juan et al. "Hybrid SGP4 orbit propagator". In: *Acta Astronautica* 137 (2017-08), pp. 254–260. ISSN: 0094-5765. DOI: 10.1016/j.actaastro.2017.04.015 (cit. on p. 8).

[55] J. F. San-Juana et al. "Hybrid SGP4 propagator based on machine-learning techniques applied to GALILEO-type orbits". In: *69th International Astronautical Congress, Bremen, Germany*. Vol. 4. 2018 (cit. on p. 8).

[56] H. Schaub and J. L. Junkins. *Analytical mechanics of space systems*. Aiaa, 2003 (cit. on p. 38).

[57] L. N. Smith and N. Topin. "Super-convergence: Very fast training of neural networks using large learning rates". In: *Artificial intelligence and machine learning for multi-domain operations applications*. Vol. 11006. SPIE. 2019, pp. 369–386 (cit. on p. 47).

[58] SpaceX. *Commitment to Space Sustainability*. Accessed: 2025-01-29. 2024. URL: https://api.starlink.com/public-files/Commitment%20to%20Space%20Sustainability.pdf (cit. on p. 30).

[59] S. Subramanian et al. "Orbit Propagation from Historical Data using Physics-informed Neural ODEs". In: 2023-12. DOI: 10.1109/ICMLA58977.2023.00248 (cit. on p. 10).

[60] G. Teschl. *Ordinary differential equations and dynamical systems*. Vol. 140. American Mathematical Soc., 2012 (cit. on p. 18).

[61] D. A. Vallado et al. "Revisiting spacetrack report 3: rev 2". In: AIAA-2006-6753-Rev2. AIAA Astrodynamics Specialists Conference and Exhibit . . . 2006 (cit. on p. 8).

[62] J. Varey et al. "Physics-Informed Neural Networks for Satellite State Estimation". In: *2024 IEEE Aerospace Conference*. IEEE, 2024-03, pp. 1–8. DOI: 10.1109/aero58975.2024.10521414 (cit. on p. 10).

[63] R. Wang, J. Liu, and Q. Zhang. "Propagation errors analysis of TLE data". In: *Advances in Space Research* 43.7 (2009), pp. 1065–1069 (cit. on p. 8).

[64] S. Wang, Y. Teng, and P. Perdikaris. "Understanding and mitigating gradient flow pathologies in physics-informed neural networks". In: *SIAM Journal on Scientific Computing* 43.5 (2021), A3055–A3081 (cit. on p. 20).

[65] S. Wang et al. *An Expert's Guide to Training Physics-informed Neural Networks*. en. arXiv:2308.08468 [cs]. 2023-08. DOI: 10.48550/arXiv.2308.08468 (cit. on p. 20).

[66] Y. Wang et al. "Improving Precise Orbit Determination of LEO Satellites Using Enhanced Solar Radiation Pressure Modeling". en. In: *Space Weather* 21.1 (2023-01), e2022SW003292. ISSN: 1542-7390, 1542-7390. DOI: 10.1029/2022SW003292 (cit. on pp. 2, 3).

[67] T. Westny et al. *Stability-Informed Initialization of Neural Ordinary Differential Equations*. 2024. arXiv: 2311.15890 [cs.LG] (cit. on pp. 17, 18).

[68] Y. Zhuang and L. Wang. "ANALYSIS OF THE GRAVITY MODELS IMPACT ON LEO SATELLITE ORBIT PREDICTION". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLVI-3/W1-2022 (2022-04), pp. 307–313. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-XLVI-3-W1-2022-307-2022 (cit. on p. 4).

A

Additional Figures for Exploratory Data Analysis

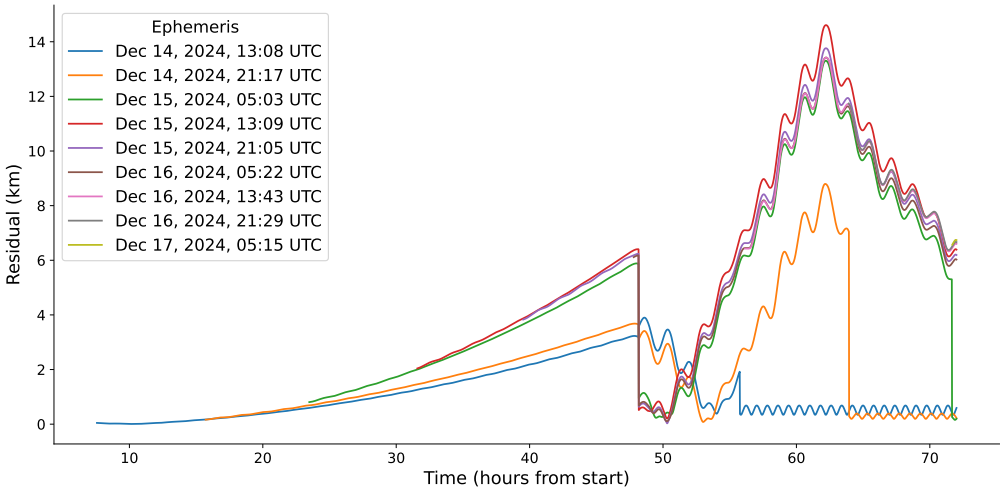


Figure A.1: Comparison of predictions between the first ephemeris file, starting on December 14, 2024, at 4 AM UTC, with the subsequent nine ephemeris files to form a complete three-day prediction for the satellite 44751. The residuals represent the distance between the corresponding points in each comparison.

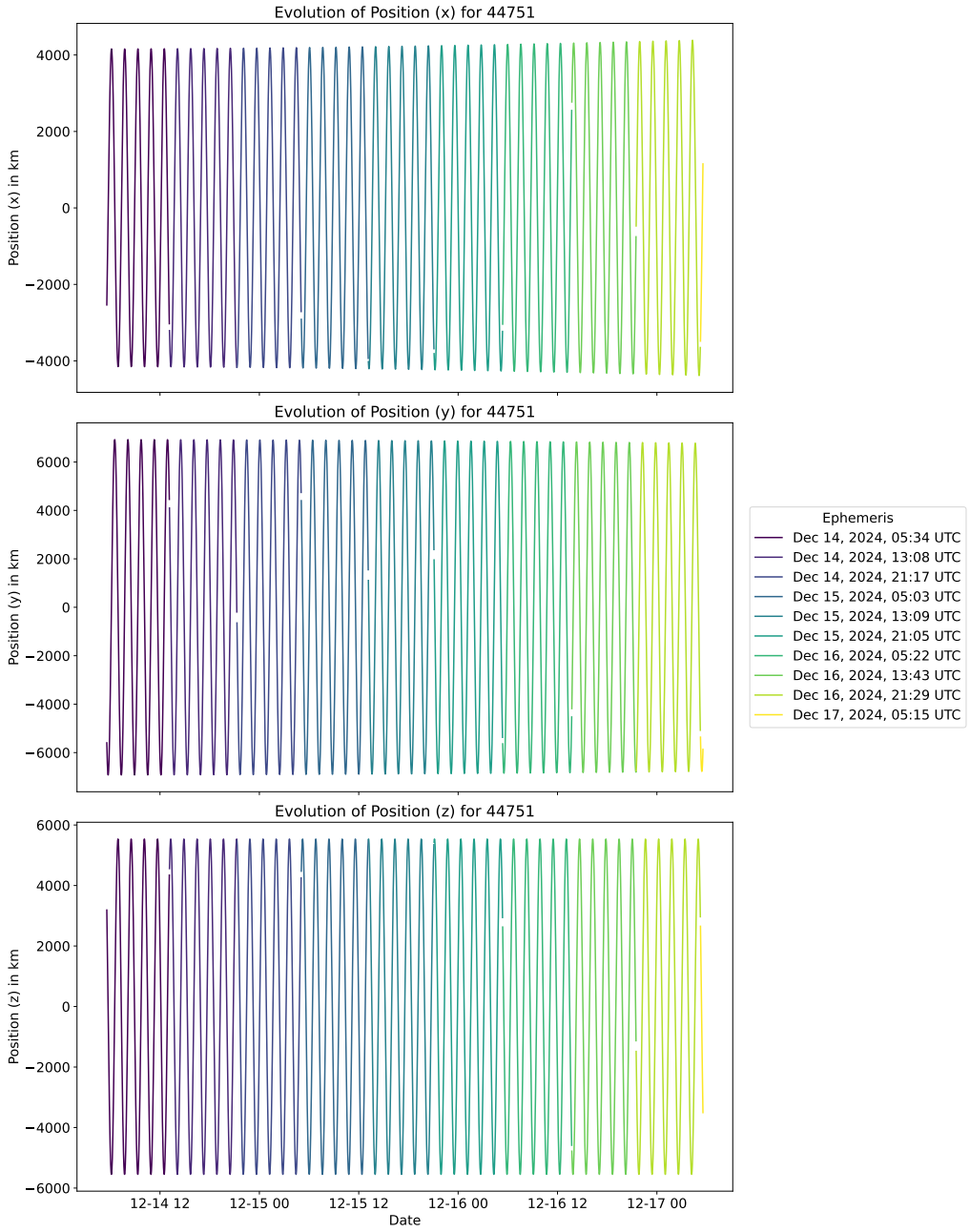


Figure A.2: Visualization of the satellite's x, y, and z positional evolution over three days for satellite 44751. Since the plotted position is derived from a combination of ephemeris files, using only the most accurate segments from each file, a discontinuity occurs at the transitions between files when one ends, and another begins.

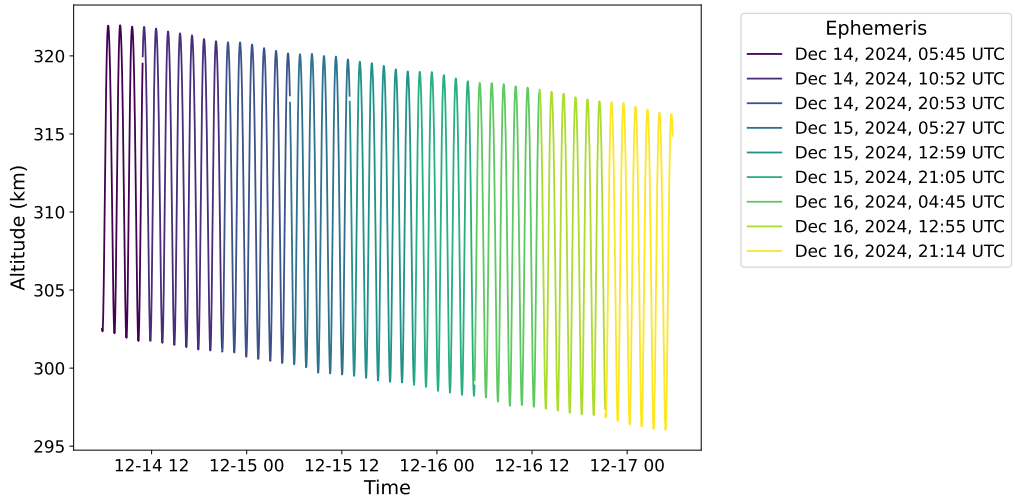


Figure A.3: Altitude variation of the deorbiting satellite 44760 over three days, starting on December 14, 2024. The oscillatory pattern reflects the satellite's movement between apogee (maximum altitude) and perigee (minimum altitude) as it follows its elliptical orbit around Earth. Because it is deorbiting, the altitude is decreasing. Since the plotted altitude is derived from a combination of ephemeris files, using only the most accurate segments from each file, a discontinuity occurs at the transitions between files when one ends, and another begins.

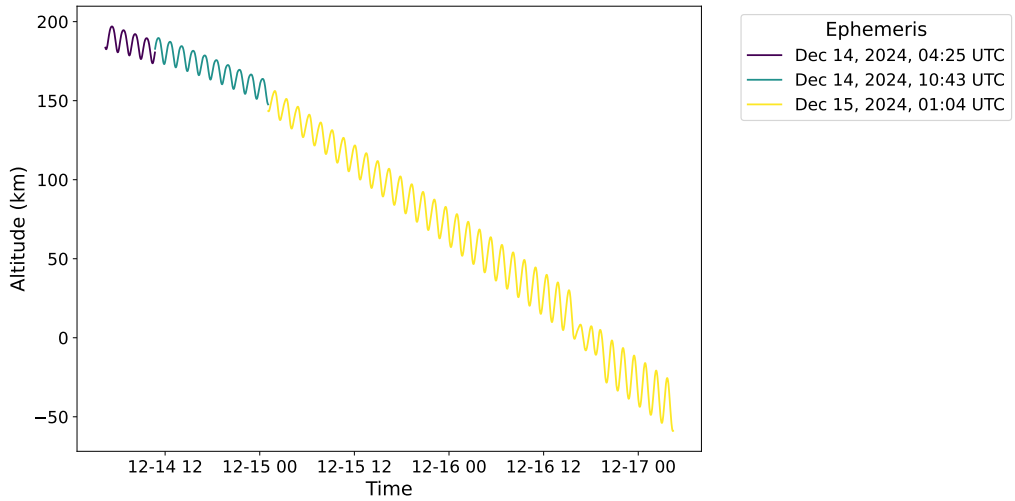


Figure A.4: Altitude variation of the decayed satellite 45049 over three days, starting on December 14, 2024. The oscillatory pattern reflects the satellite's movement between apogee (maximum altitude) and perigee (minimum altitude) as it follows its elliptical orbit around Earth. The satellite decayed on the 15th of December, hence the negative values in altitude after it decayed. Since the plotted altitude is derived from a combination of ephemeris files, using only the most accurate segments from each file, a discontinuity occurs at the transitions between files when one ends, and another begins.

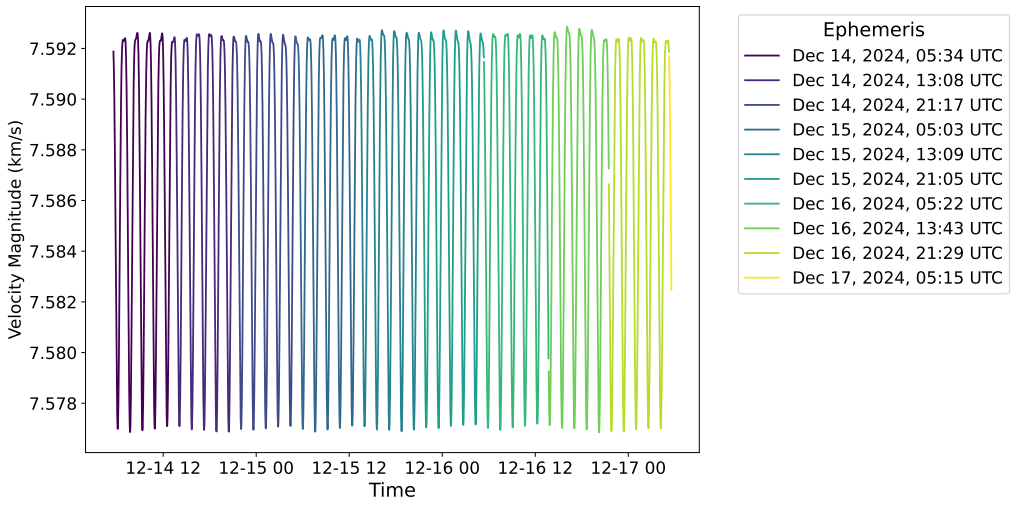


Figure A.5: Velocity magnitude of the satellite 44751 over three days, starting on December 14, 2024. The periodic variations correspond to the satellite's motion along its elliptical orbit, where velocity increases near perigee and decreases near apogee. Since the plotted velocity is derived from a combination of ephemeris files, using only the most accurate segments from each file, a discontinuity occurs at the transitions between files when one ends, and another begins.

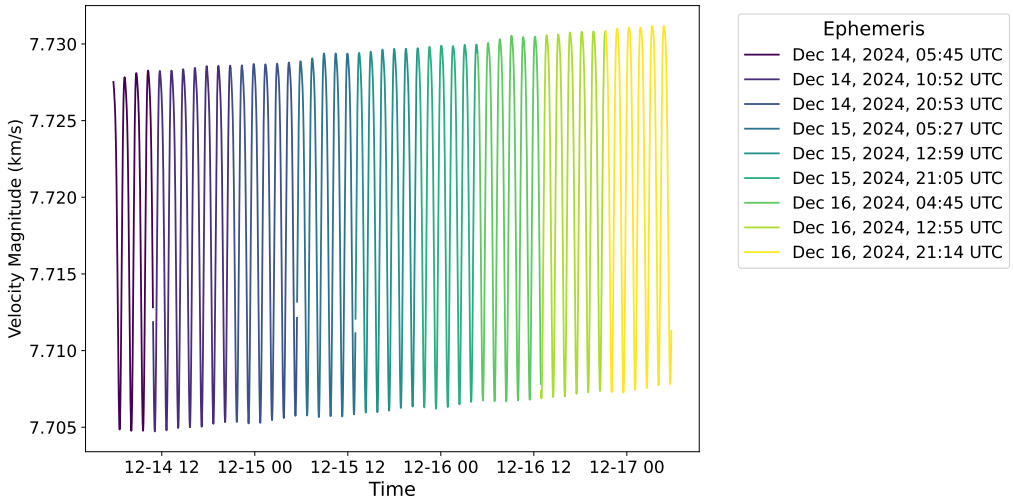


Figure A.6: Velocity magnitude of the deorbiting satellite 44760 over three days, starting on December 14, 2024. The periodic variations correspond to the satellite's motion along its elliptical orbit, where velocity increases near perigee and decreases near apogee. The satellite is deorbiting, hence the increase in velocity. Since the plotted velocity is derived from a combination of ephemeris files, using only the most accurate segments from each file, a discontinuity occurs at the transitions between files when one ends, and another begins.

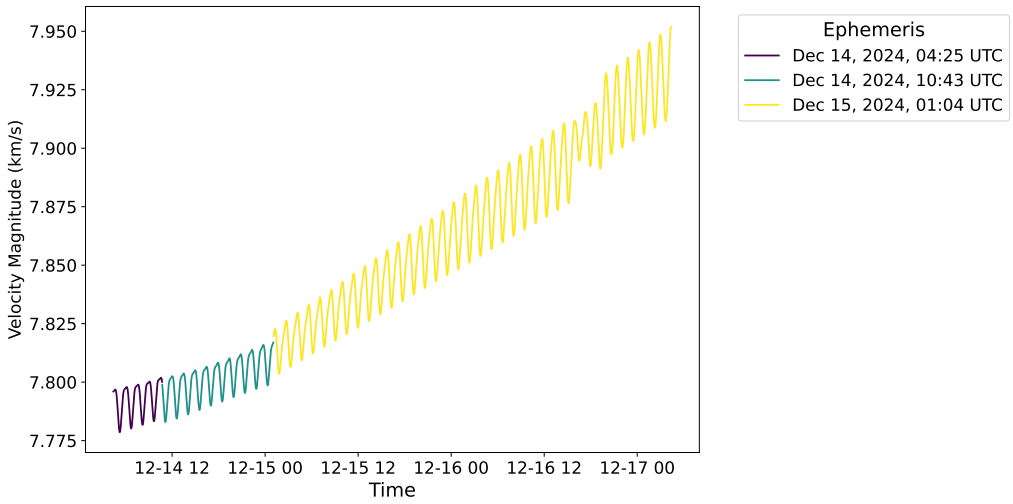


Figure A.7: Velocity magnitude of the decayed satellite 45049 over three days, starting on December 14, 2024. The periodic variations correspond to the satellite's motion along its elliptical orbit, where velocity increases near perigee and decreases near apogee. The satellite decayed on the 15th of December, hence increase in velocity. Since the plotted velocity is derived from a combination of ephemeris files, using only the most accurate segments from each file, a discontinuity occurs at the transitions between files when one ends, and another begins.

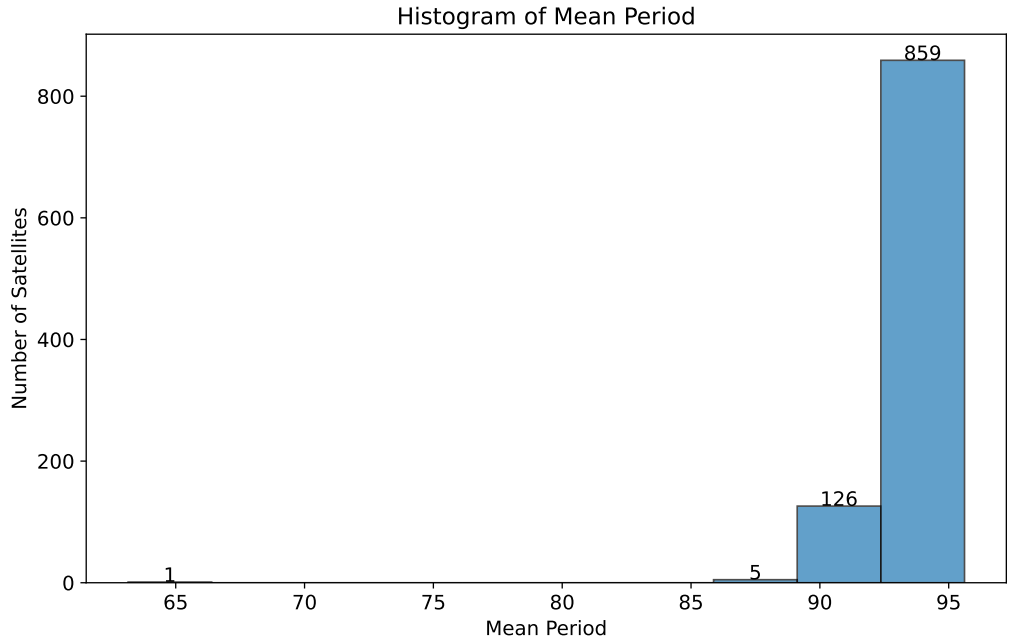


Figure A.8: Histogram of the mean orbital period of 991 satellites observed between December 14 and December 17. These eccentricities were calculated with the ephemeris data.

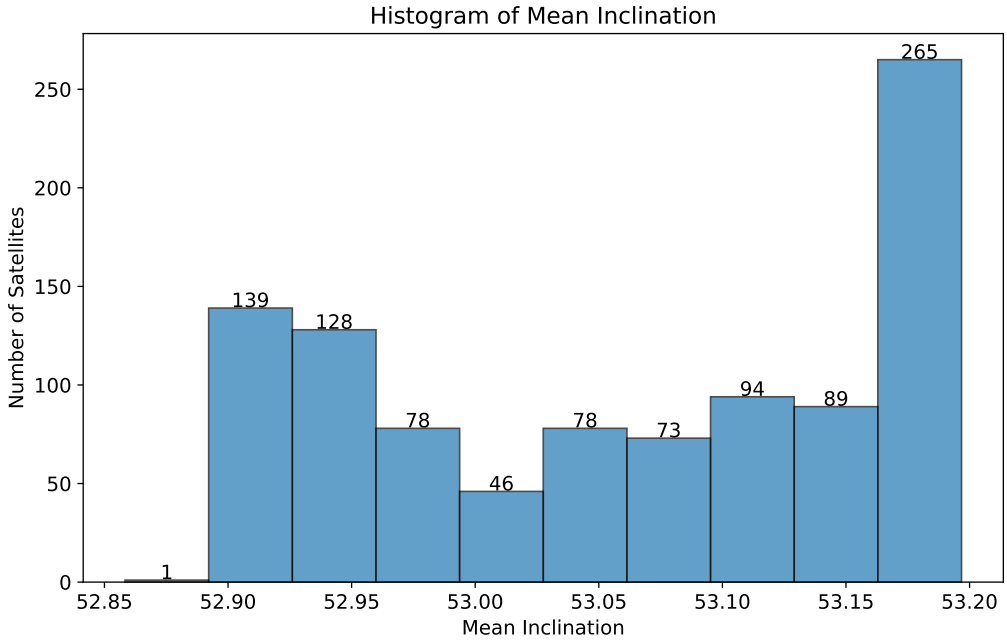


Figure A.9: Histogram of the mean inclination of 991 satellites observed between December 14 and December 17. These eccentricities were calculated with the ephemeris data.

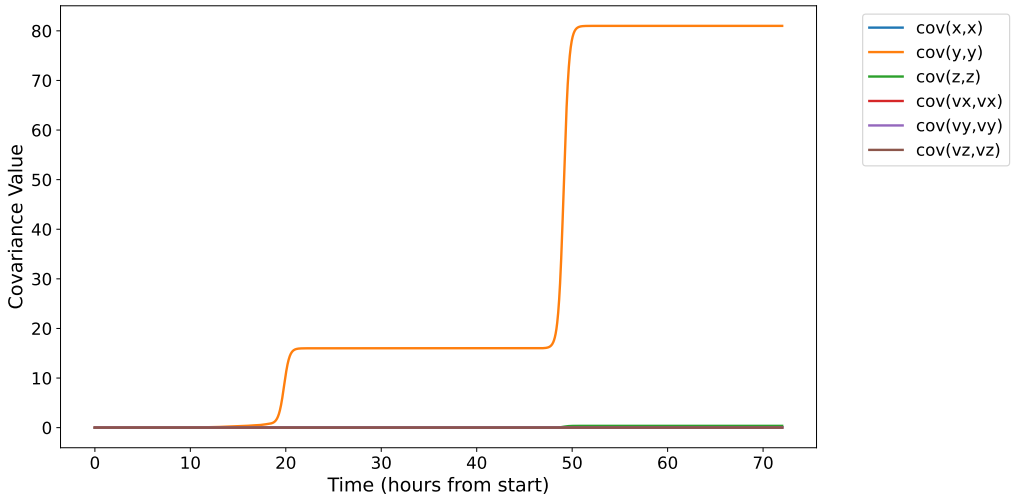


Figure A.10: Evolution of the covariance values for satellite 44751.

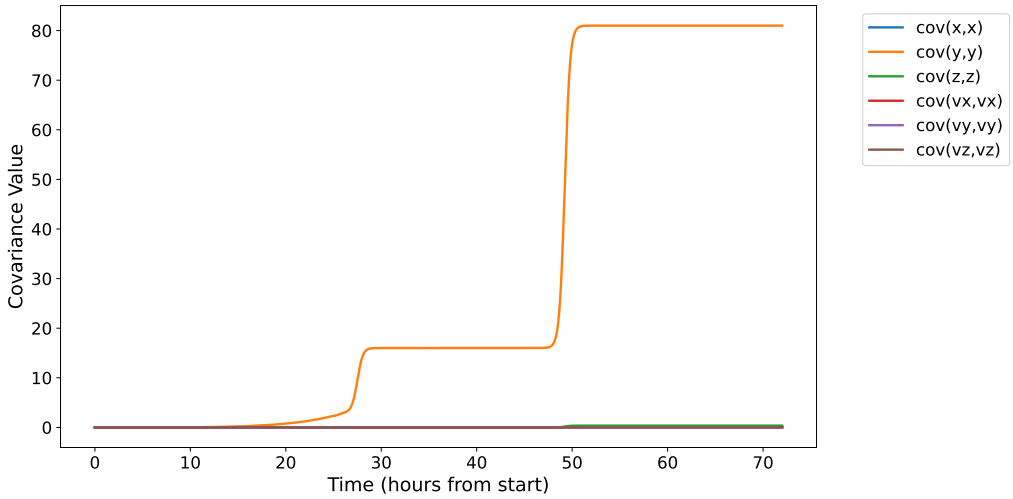


Figure A.11: Evolution of the covariance values for satellite 44917 during one ephemeris file.

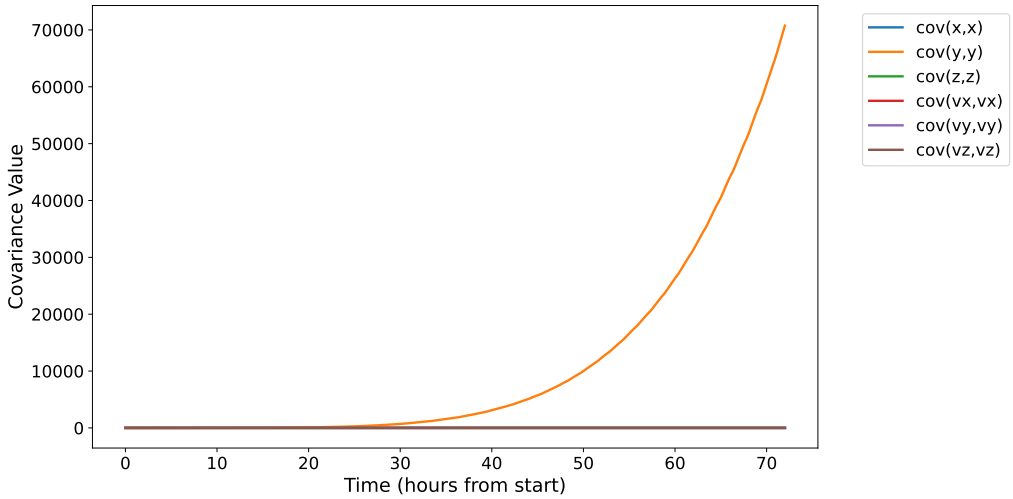


Figure A.12: Evolution of the covariance values for satellite 44760 during one ephemeris file.

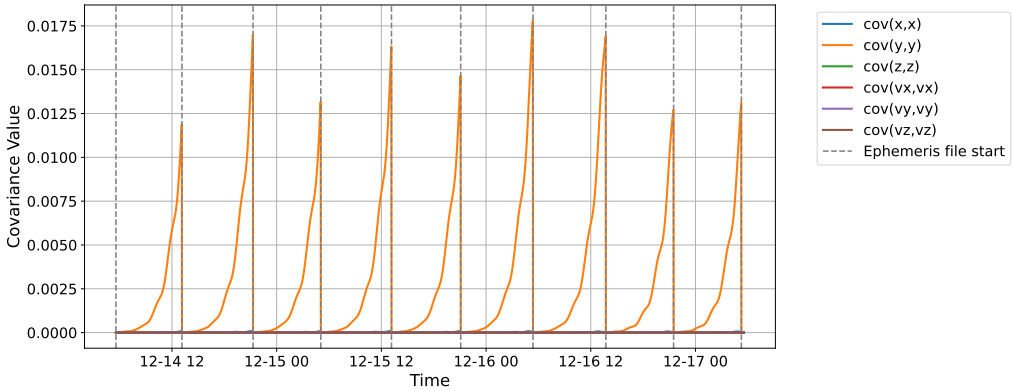


Figure A.13: Evolution of the covariance values for satellite 44751. The covariance values are updated everytime there is a new ephemeris file. The gray line indicates when a new ephemeris file was created.

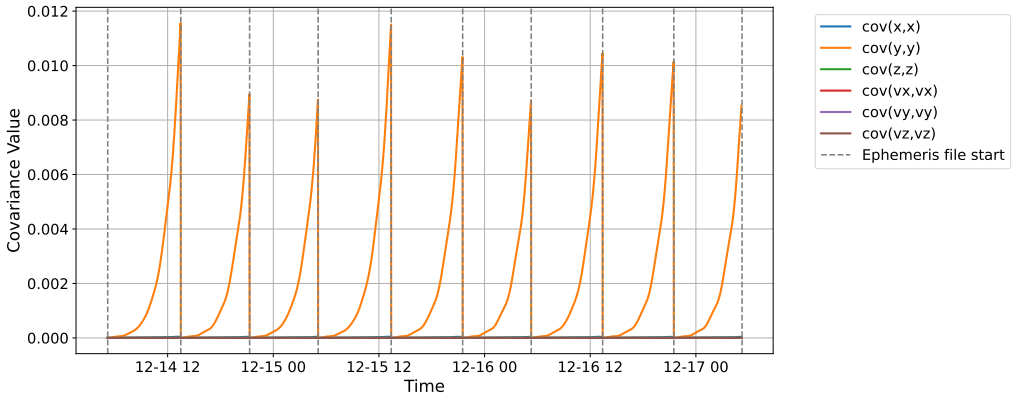


Figure A.14: Evolution of the covariance values for satellite 44917. The covariance values are updated everytime there is a new ephemeris file. The gray line indicates when a new ephemeris file was created.

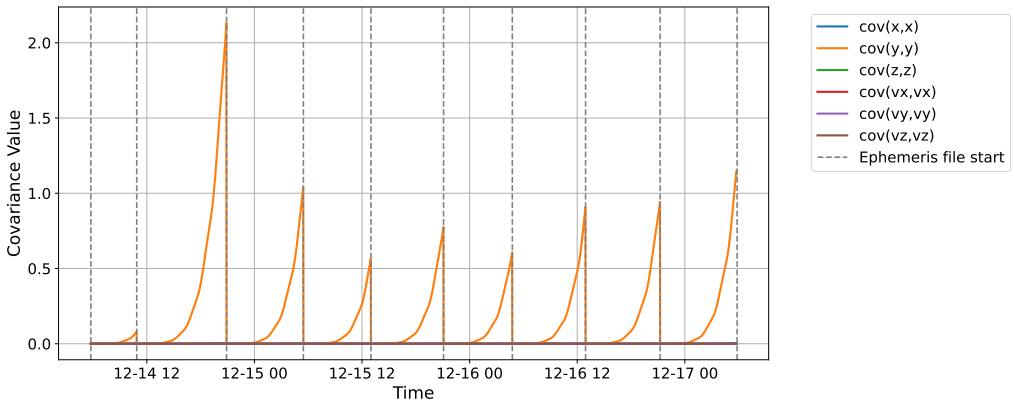


Figure A.15: Evolution of the covariance values for satellite 44760. The covariance values are updated everytime there is a new ephemeris file. The gray line indicates when a new ephemeris file was created.

B

Additional Figures for Model Evaluation

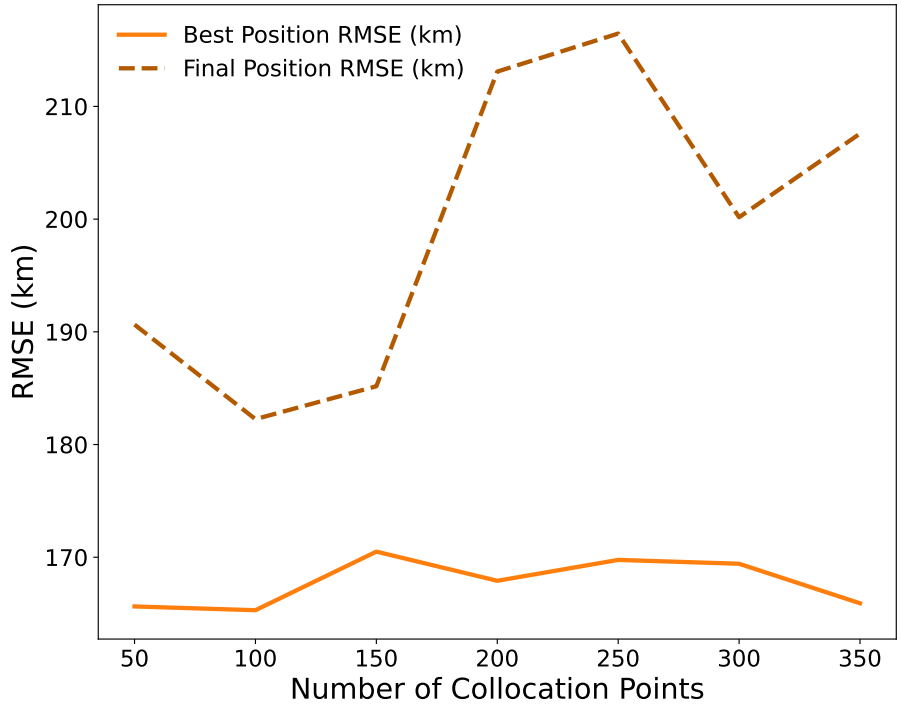


Figure B.1: Evolution of the RMSE for the PINN model as the number of collocation points increases.

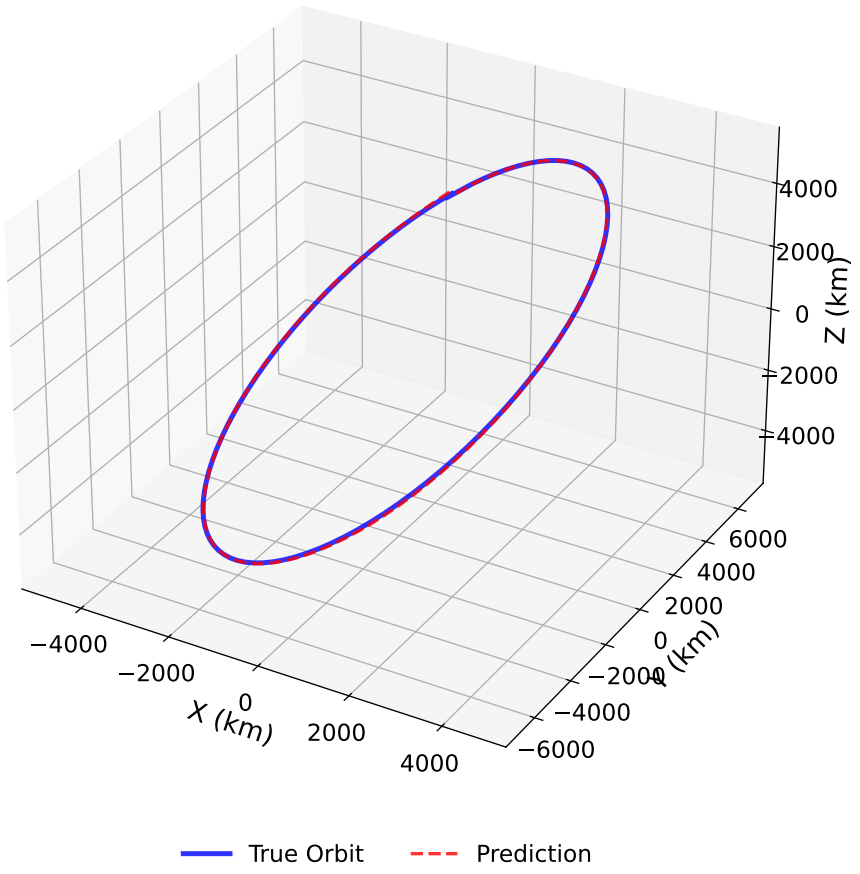


Figure B.2: Plot showing the predicted orbit in 3D (in red) versus the true orbit (in blue) for the best performing test orbit using the MLP model. The position RMSE for this orbit is 31.6km. The evolution of the error for each component over time is shown is Figure 6.9 for a better understanding of how the error accumulates over time.

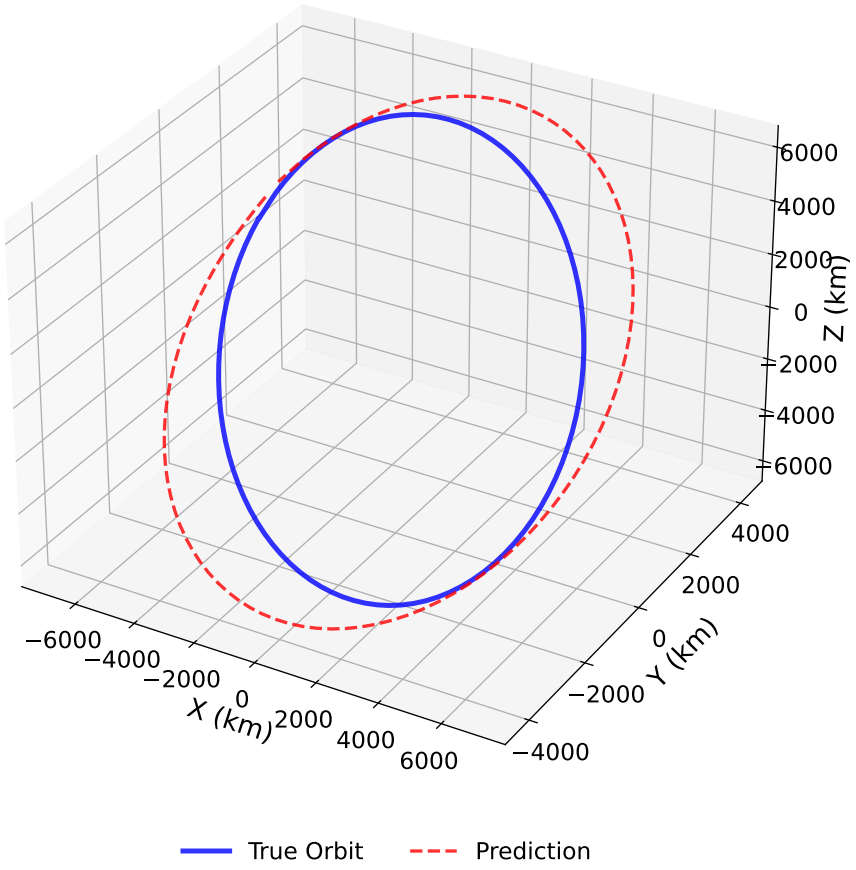


Figure B.3: Plot showing the predicted orbit in 3D (in red) versus the true orbit (in blue) for the worst performing test orbit using the MLP model. The position RMSE for this orbit is 1040.5 km and the velocity RMSE is 1.226 km/s. The evolution of the error for each component over time is shown is Figure B.8 for a better understanding of how the error accumulates over time.

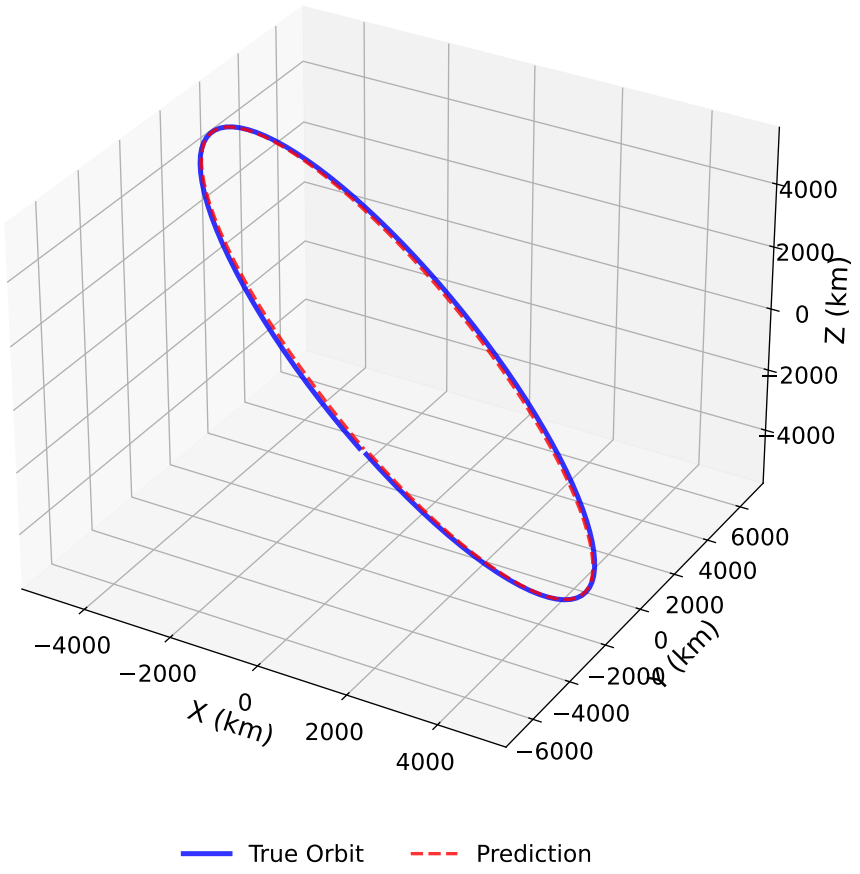


Figure B.4: Plot showing the predicted orbit in 3D (in red) versus the true orbit (in blue) for the best performing test orbit using the PINN model. The position RMSE for this orbit is 36.5km. The evolution of the error for each component over time is shown is Figure B.9 for a better understanding of how the error accumulates over time.

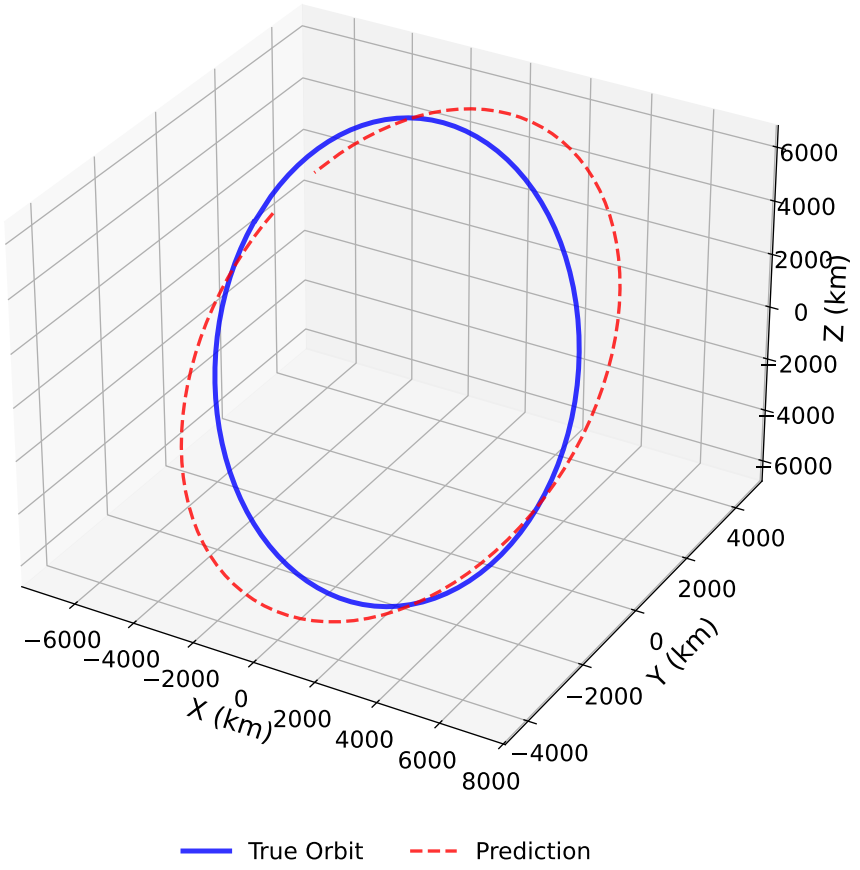


Figure B.5: Plot showing the predicted orbit in 3D (in red) versus the true orbit (in blue) for the worst performing test orbit using the PINN model. The position RMSE for this orbit is 1040.1 km and the velocity RMSE is 1.093 km/s. The evolution of the error for each component over time is shown is Figure 6.11 for a better understanding of how the error accumulates over time

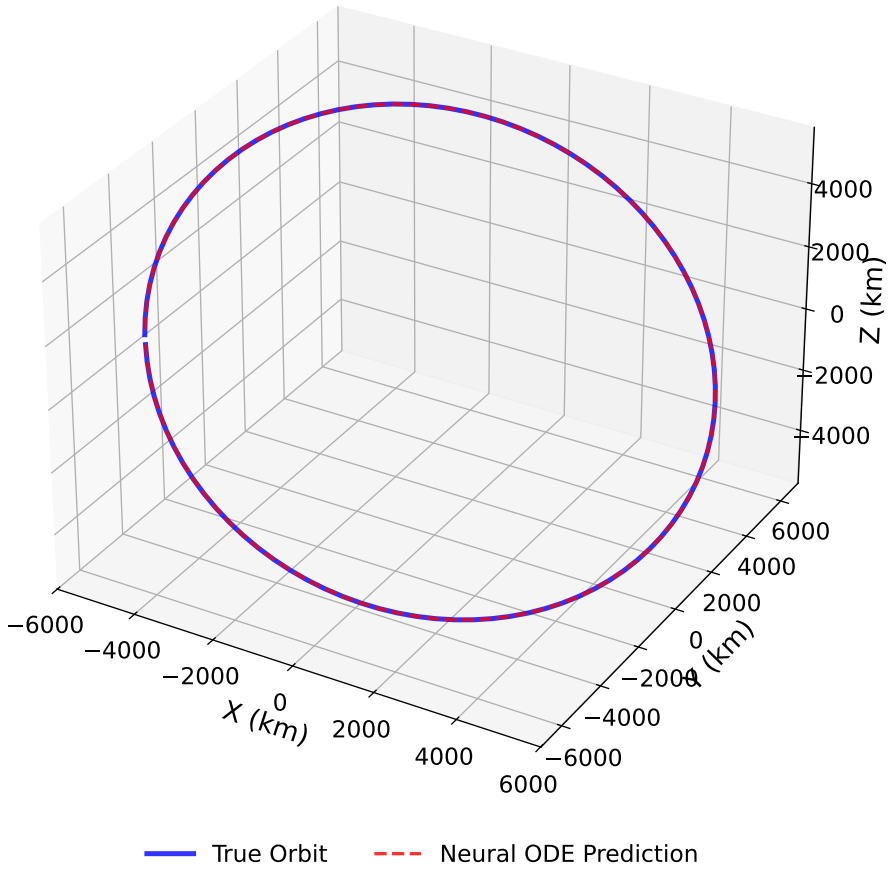


Figure B.6: Plot showing the predicted orbit in 3D (in red) versus the true orbit (in blue) for the best performing test orbit using the NODE model. The position RMSE for this orbit is 0.06km. The evolution of the error for each component over time is shown is Figure 6.13 for a better understanding of how the error accumulates over time.

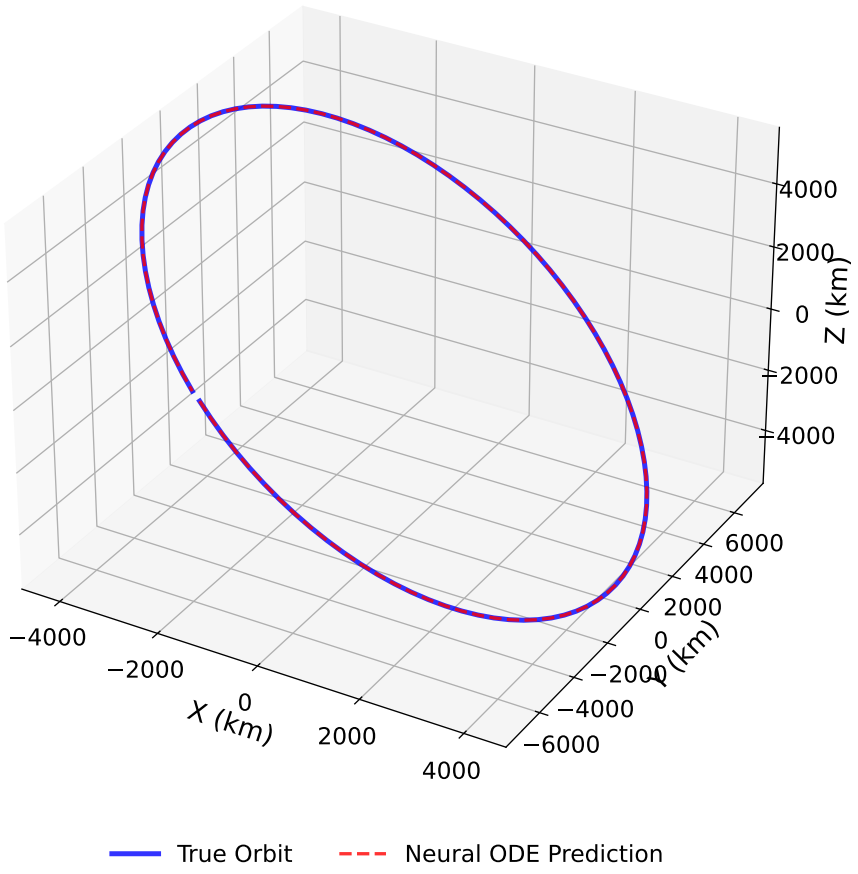


Figure B.7: Plot showing the predicted orbit in 3D (in red) versus the true orbit (in blue) for the worst performing test orbit using the NODE model. The position RMSE for this orbit is 0.6 km and the velocity RMSE is 0.00062 km/s.

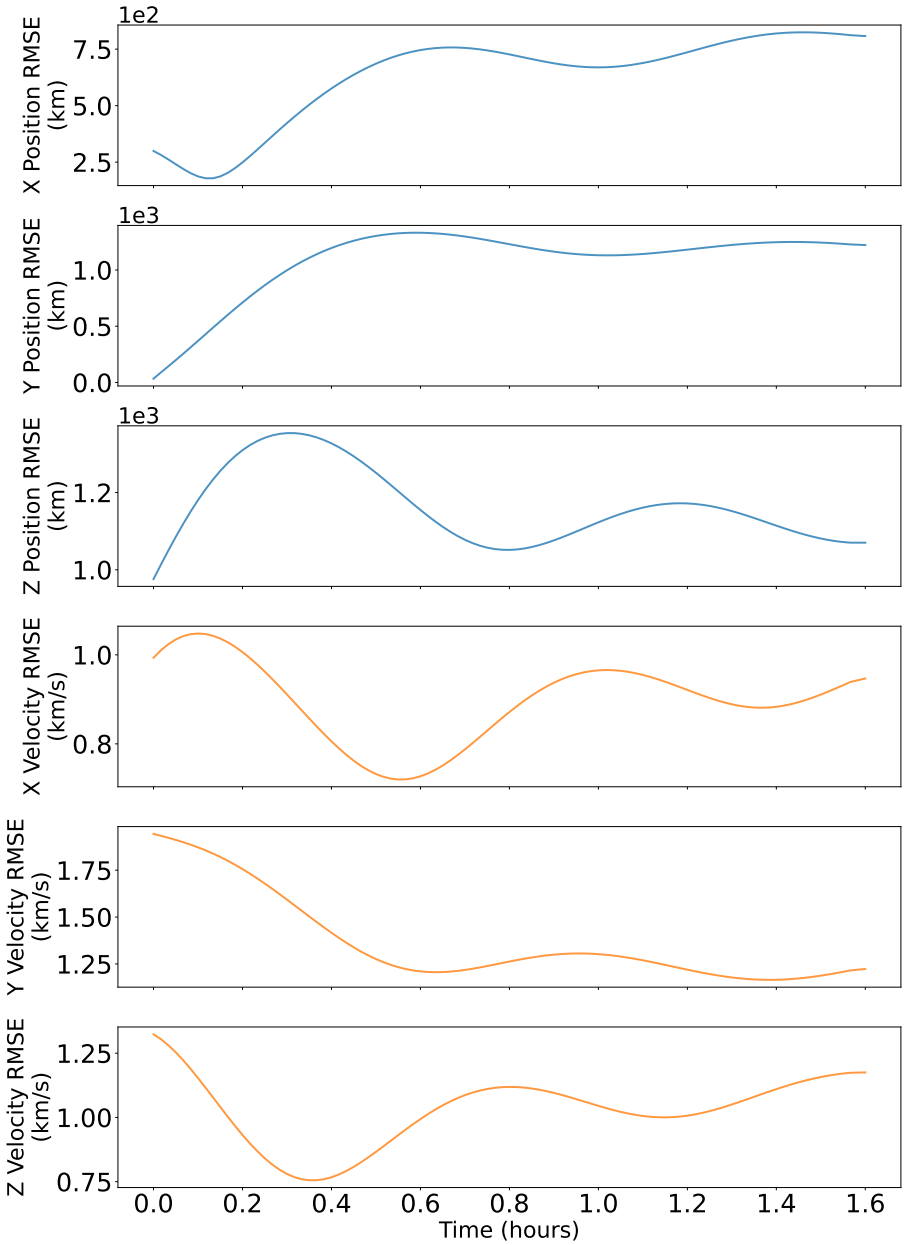


Figure B.8: Evolution of the RMSE for the worst performing test orbit using the MLP model.

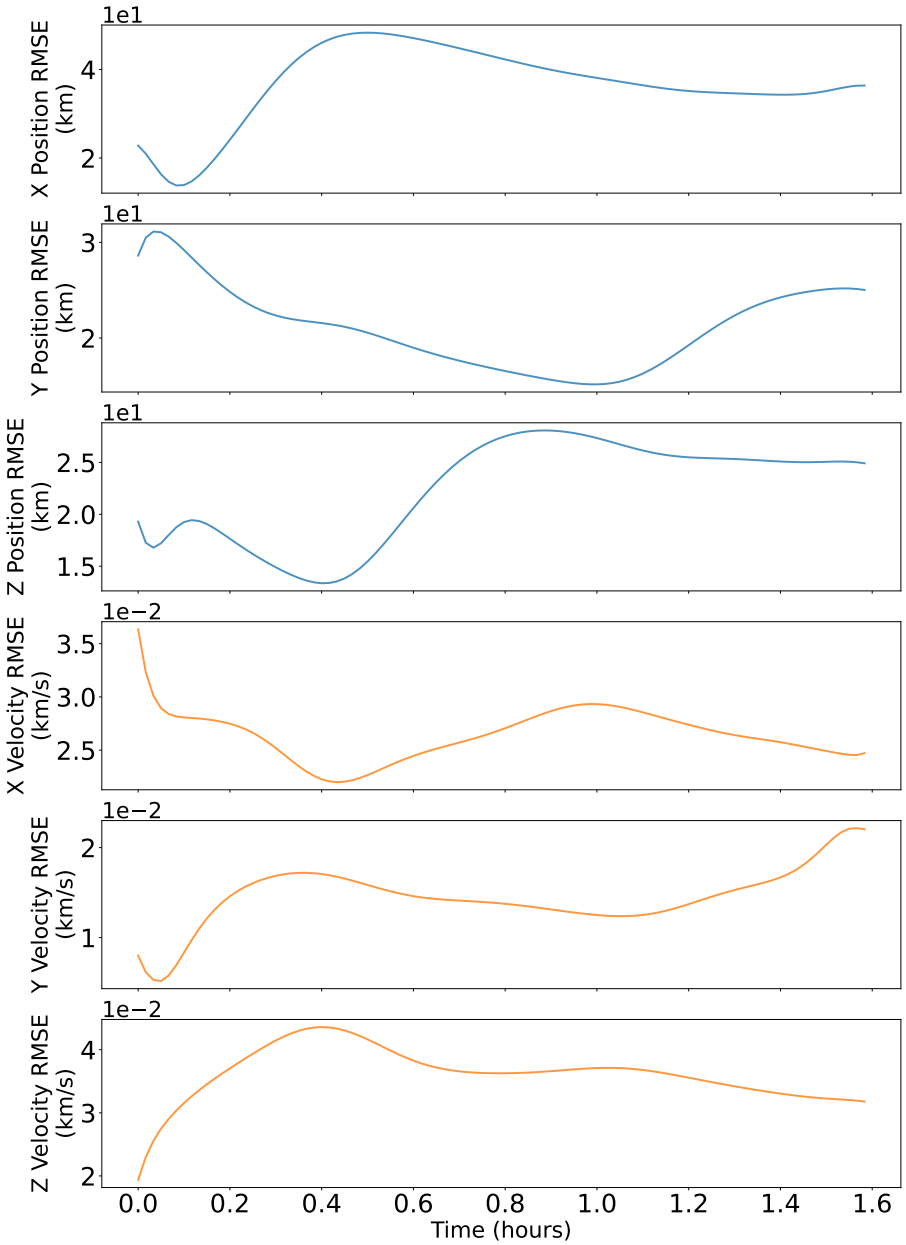


Figure B.9: Evolution of the RMSE for the best performing test orbit using the PINN model.

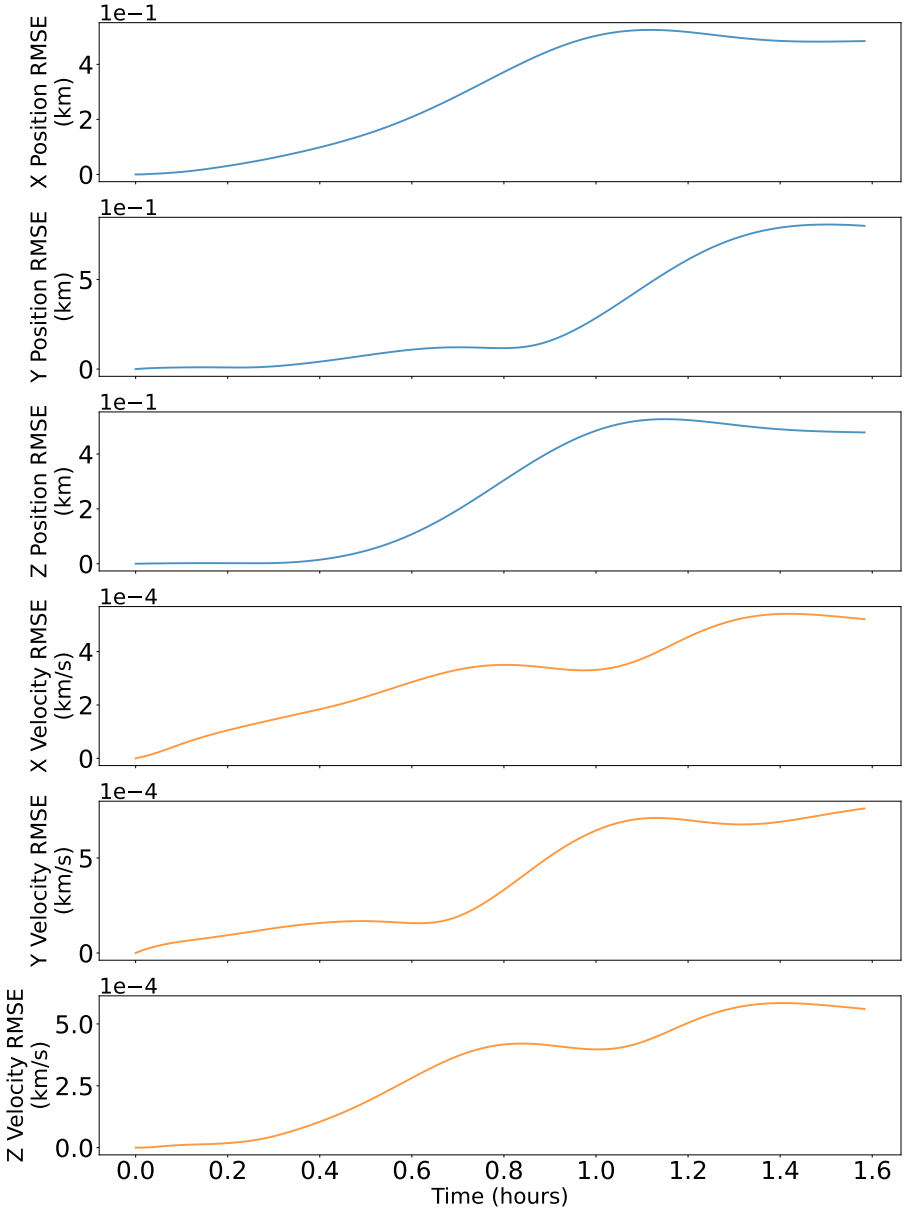


Figure B.10: Evolution of the RMSE for the worst performing test orbit using the NODE model.

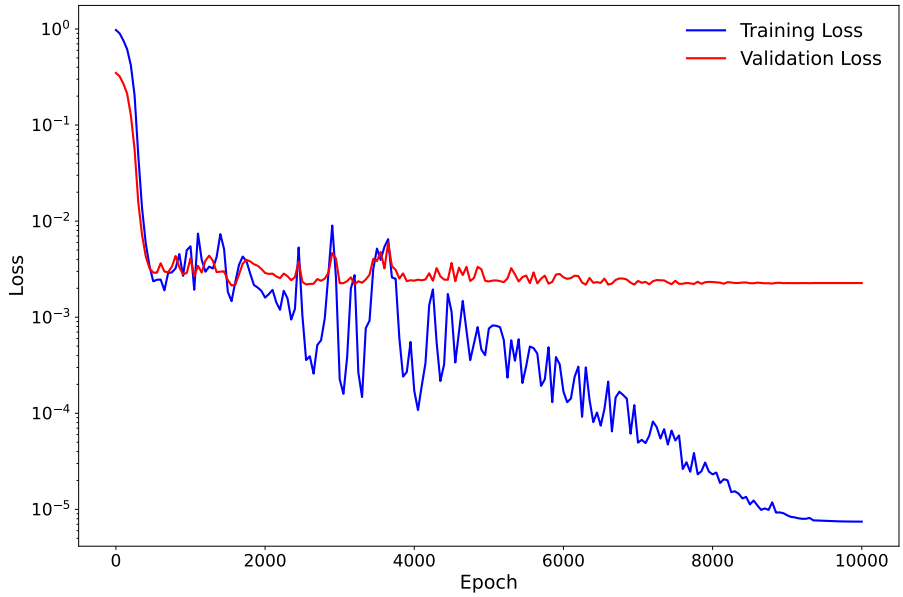


Figure B.11: Evolution of the training and validation loss for the MLP model. The training loss is shown in red, while the validation loss is shown in blue.

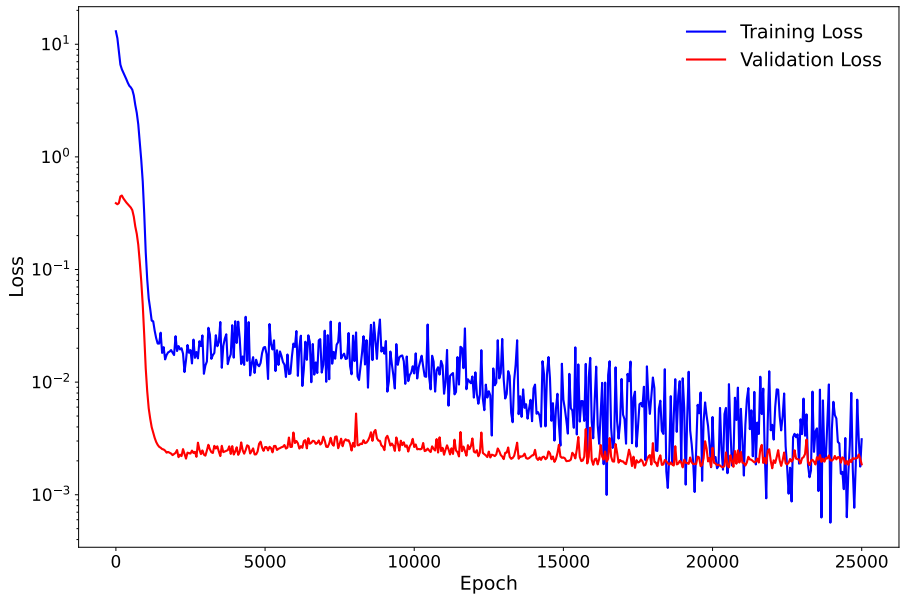


Figure B.12: Evolution of the training and validation loss for the PINN model. The training loss is shown in red, while the validation loss is shown in blue.

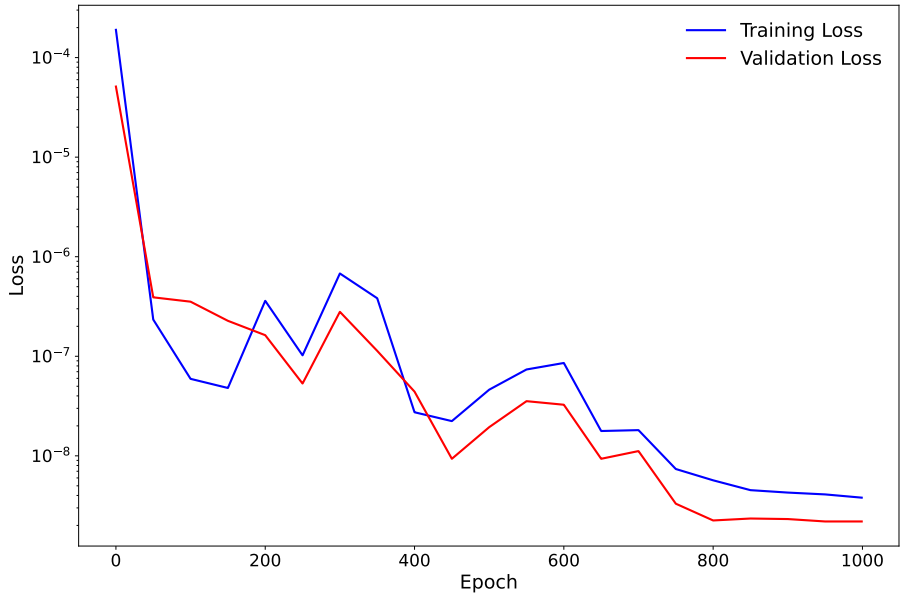


Figure B.13: Evolution of the training and validation loss for the NODE model. The training loss is shown in red, while the validation loss is shown in blue.

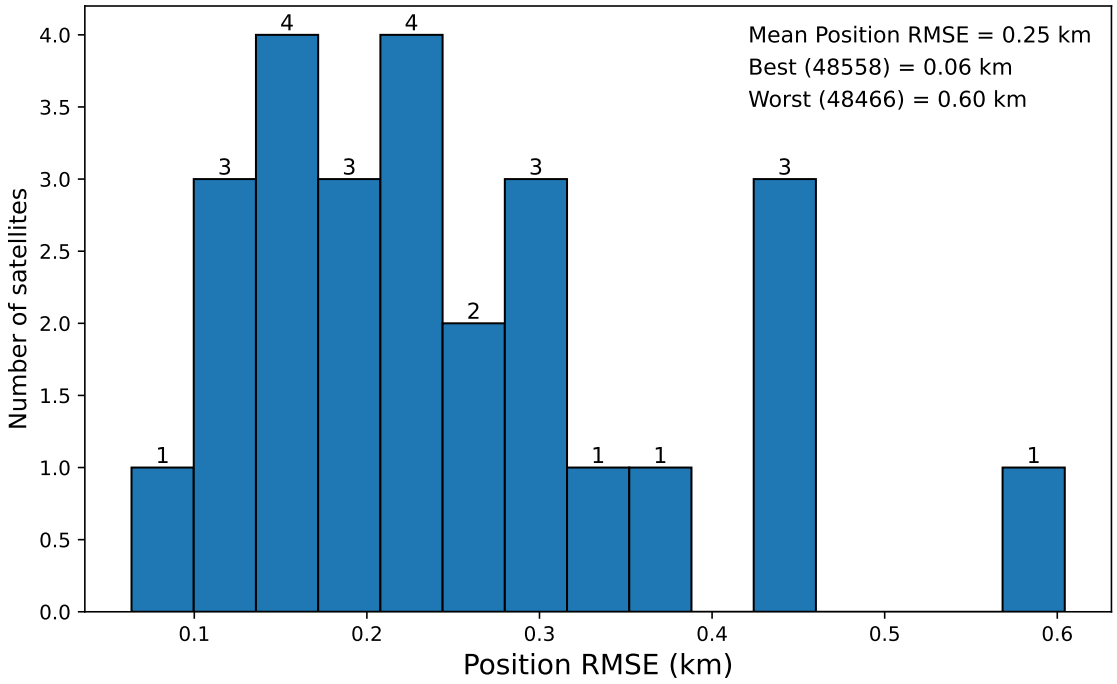


Figure B.14: Histogram of the RMSE values for the NODE model on the test set of Starlink stable orbits. The mean RMSE is 0.32 km.

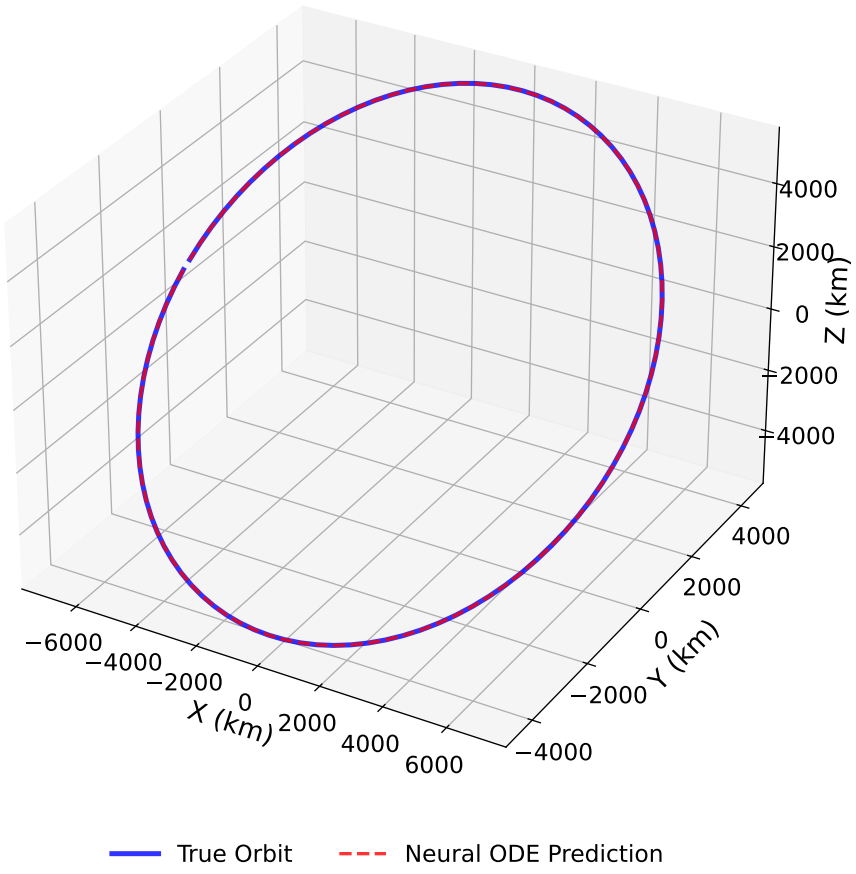


Figure B.15: Plot showing the predicted orbit in 3D (in red) versus the true orbit (in blue) for the best performing test orbit using the NODE model and Starlink stable orbits. The position RMSE for this orbit is 0.07 km. The evolution of the RMSE for this orbit is shown in Figure B.23.

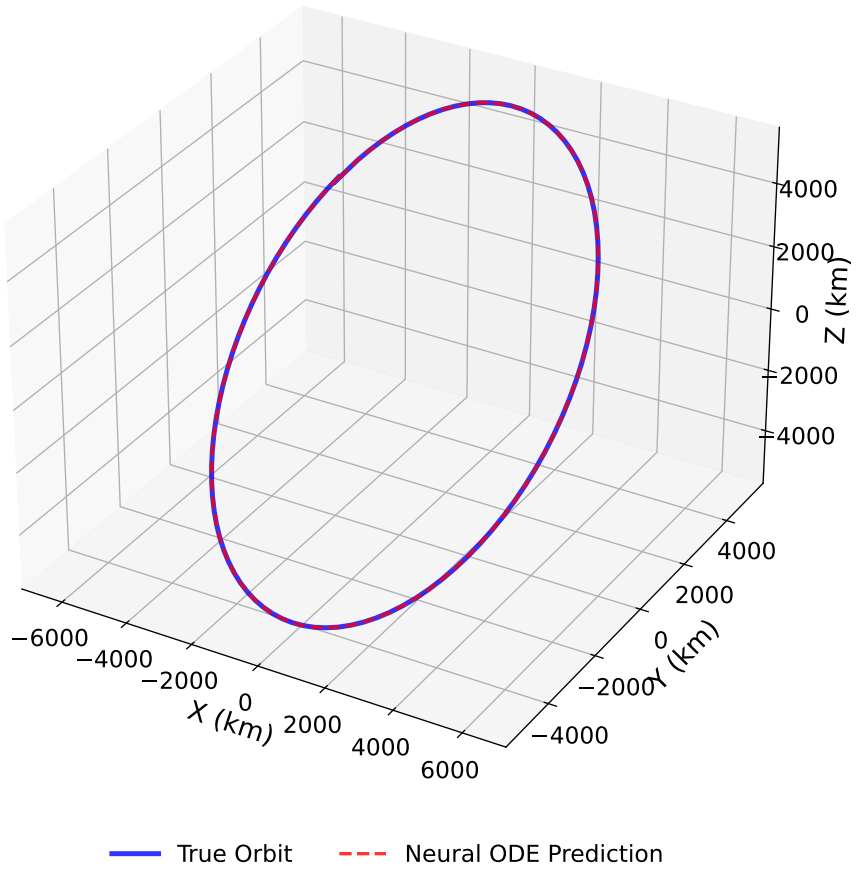


Figure B.16: Plot showing the predicted orbit in 3D (in red) versus the true orbit (in blue) for the worst performing test orbit using the NODE model and Starlink stable orbits. The position RMSE for this orbit is 0.88 km. The evolution of the RMSE for this orbit is shown in Figure B.22.

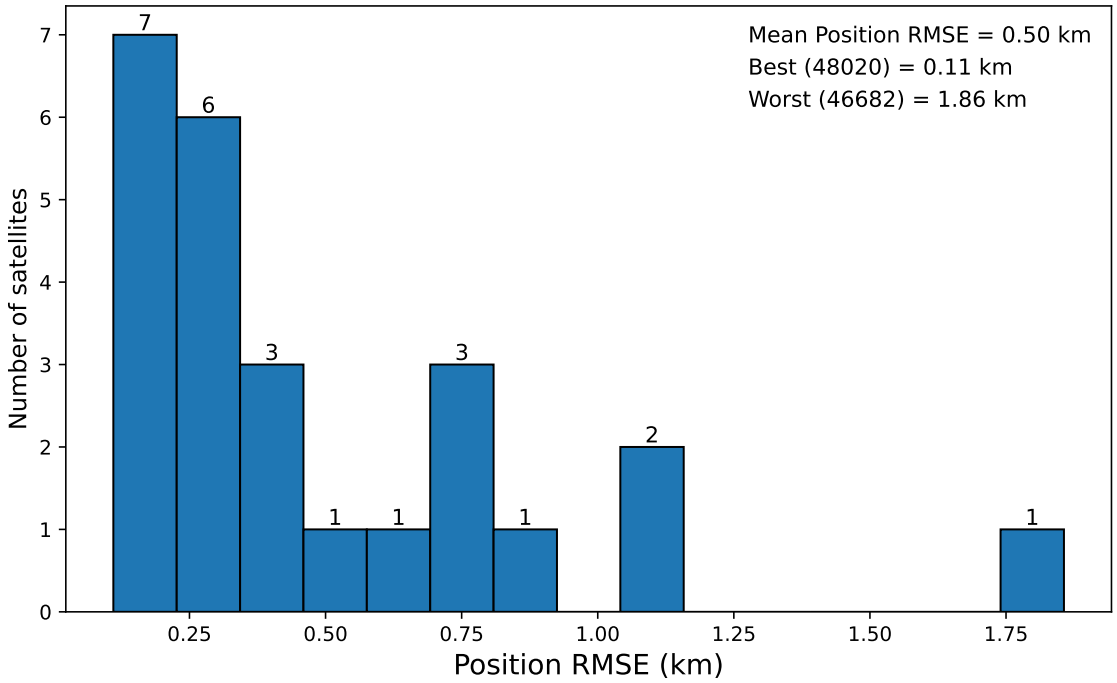


Figure B.17: Histogram of the RMSE values for the NODE model on the test set of Starlink deorbitting orbits. The mean RMSE is 0.50 km.

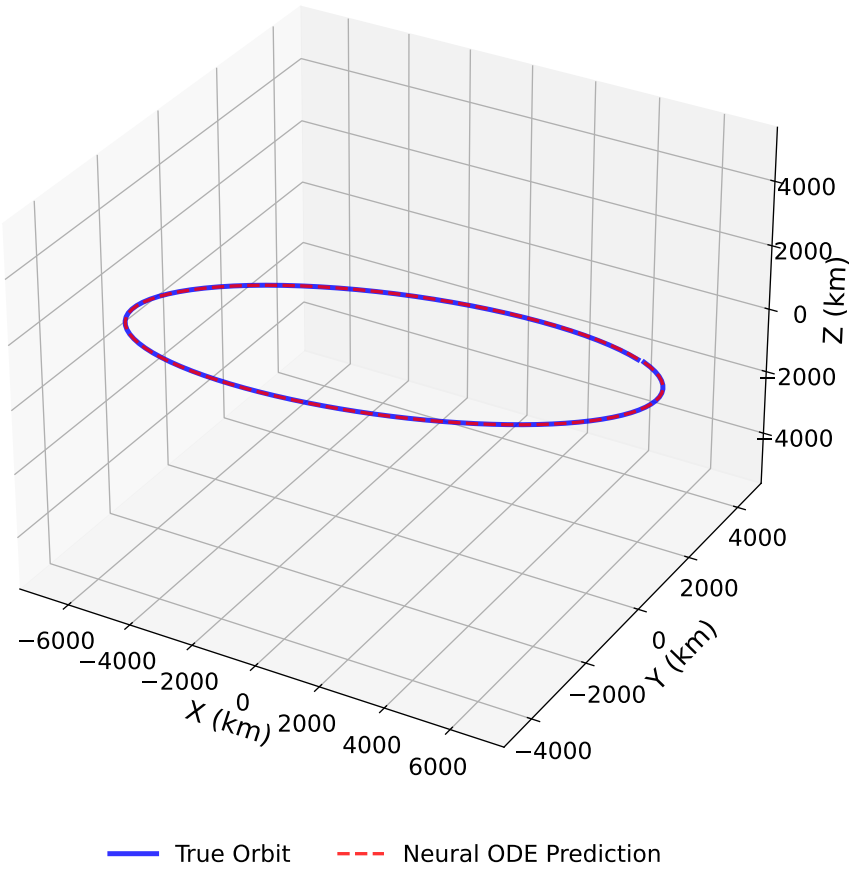


Figure B.18: Plot showing the predicted orbit in 3D (in red) versus the true orbit (in blue) for the best performing test orbit using the NODE model and Starlink deorbitting orbits. The position RMSE for this orbit is 0.11 km. The evolution of the RMSE for this orbit is shown in Figure B.21.

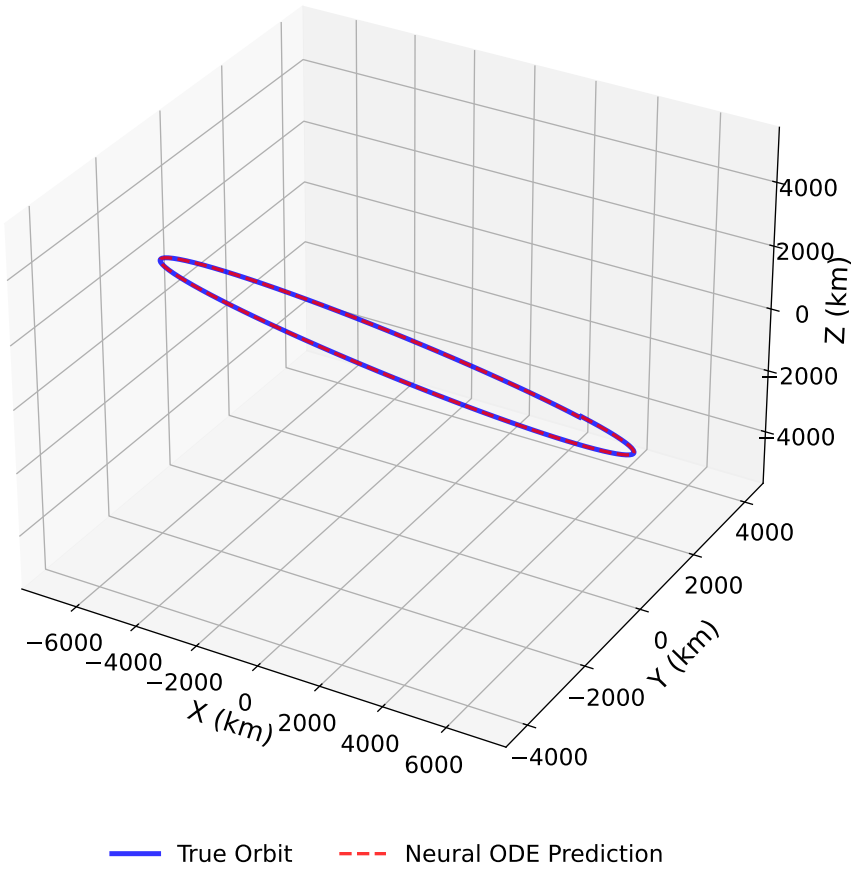


Figure B.19: Plot showing the predicted orbit in 3D (in red) versus the true orbit (in blue) for the worst performing test orbit using the NODE model and Starlink deorbitting orbits. The position RMSE for this orbit is 1.86 km. The evolution of the RMSE for this orbit is shown in Figure B.20.

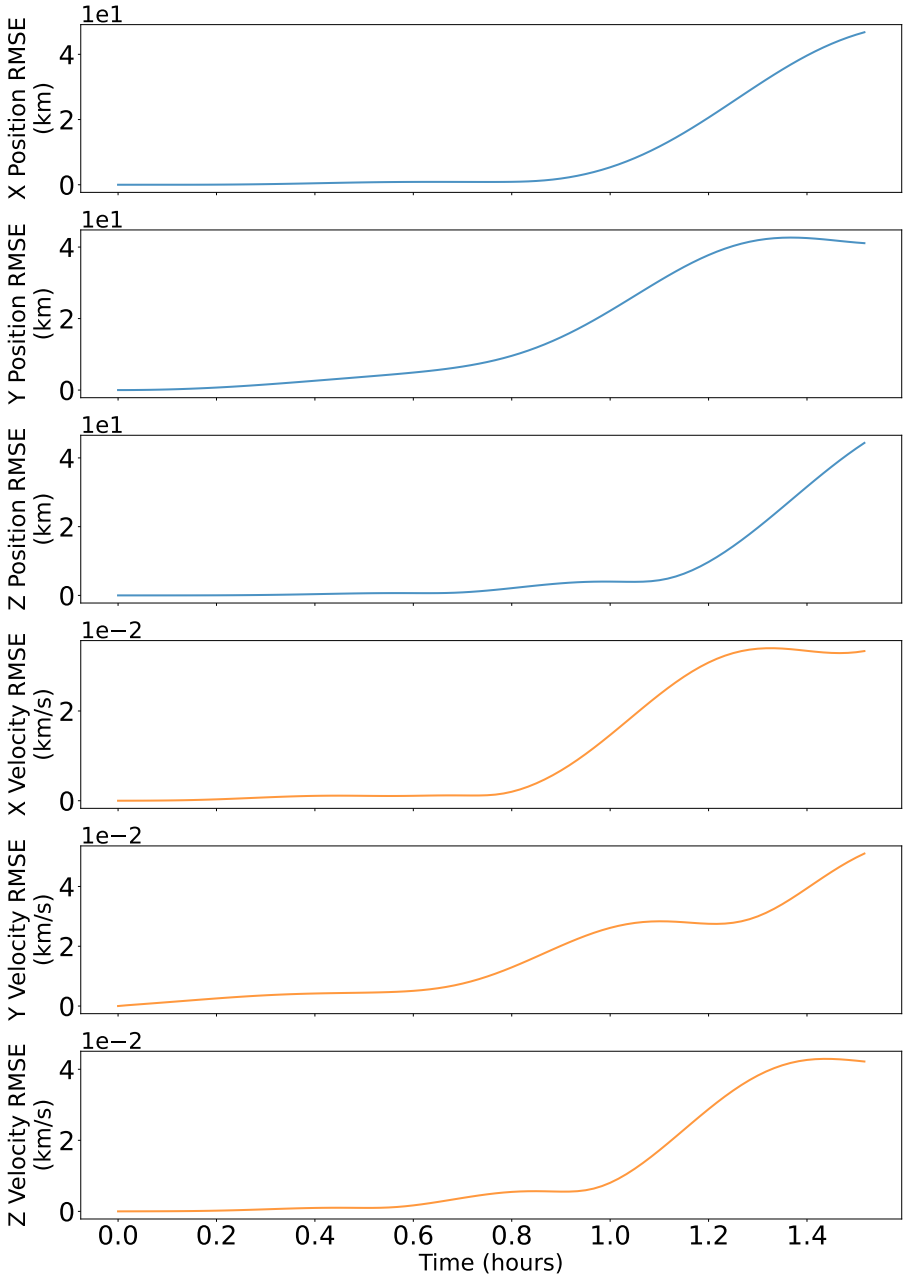


Figure B.20: Evolution of the RMSE for the worst performing test orbit using the NODE model with deorbiting Starlink data.

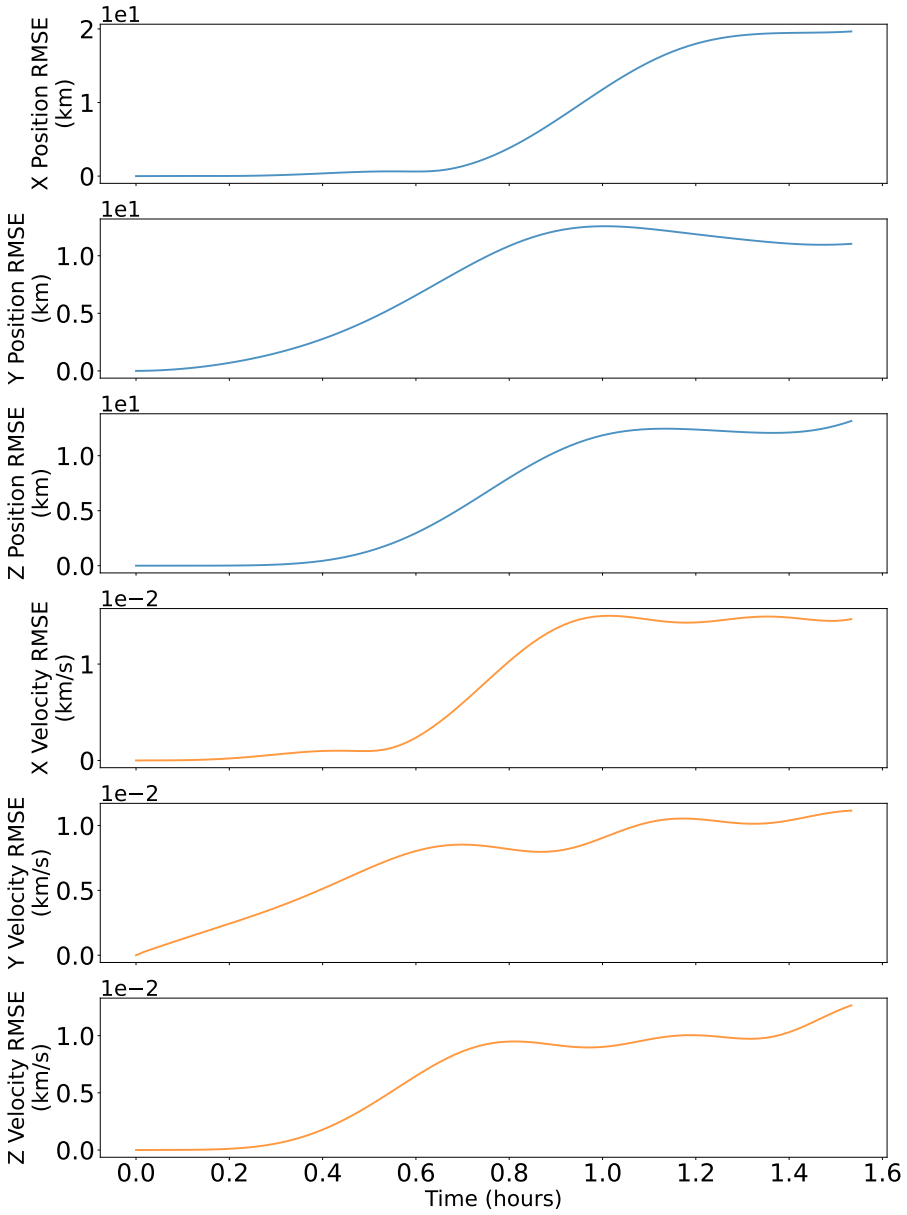


Figure B.21: Evolution of the RMSE for the best performing test orbit using the NODE model with deorbiting Starlink data.

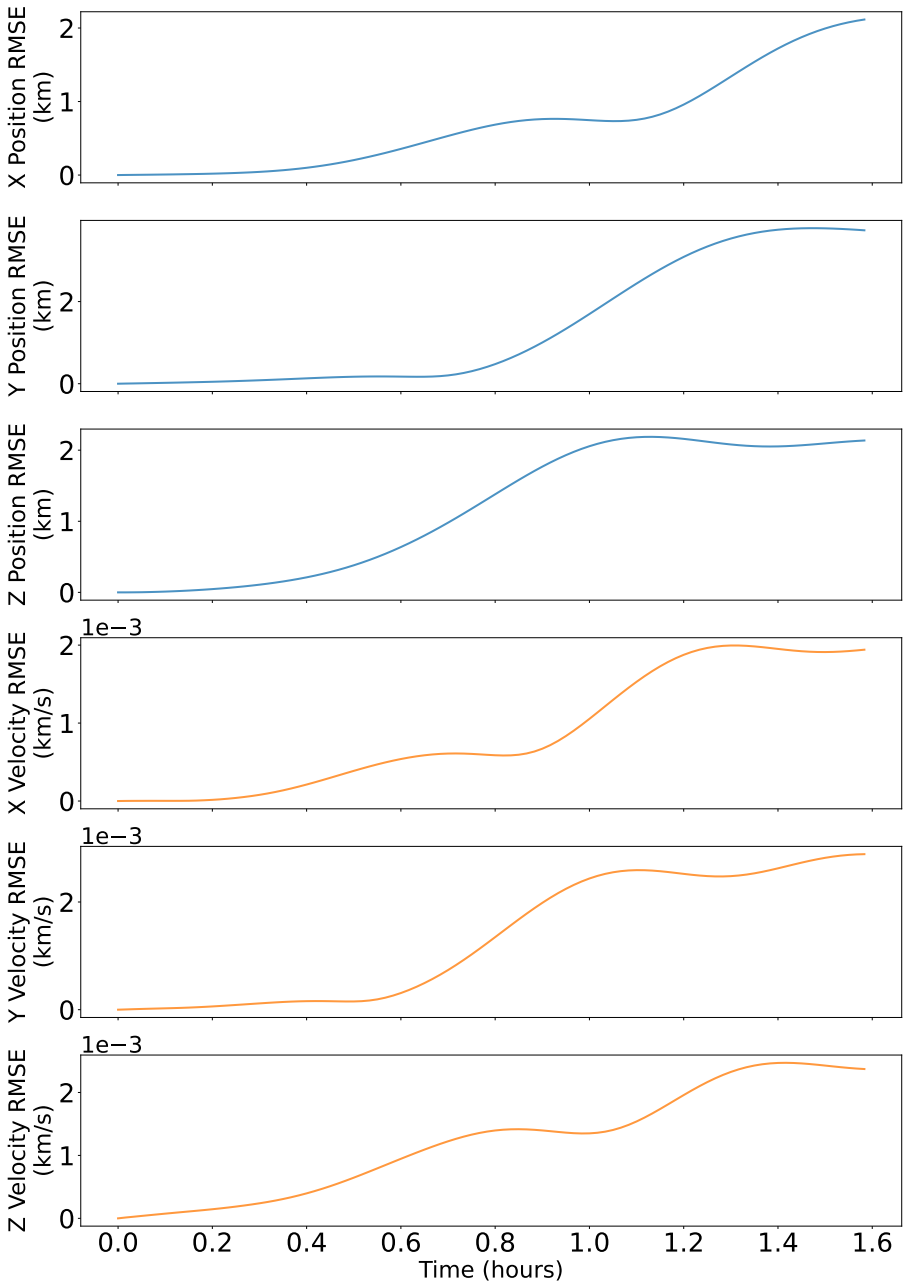


Figure B.22: Evolution of the RMSE for the worst performing test orbit using the NODE model with stable Starlink data.

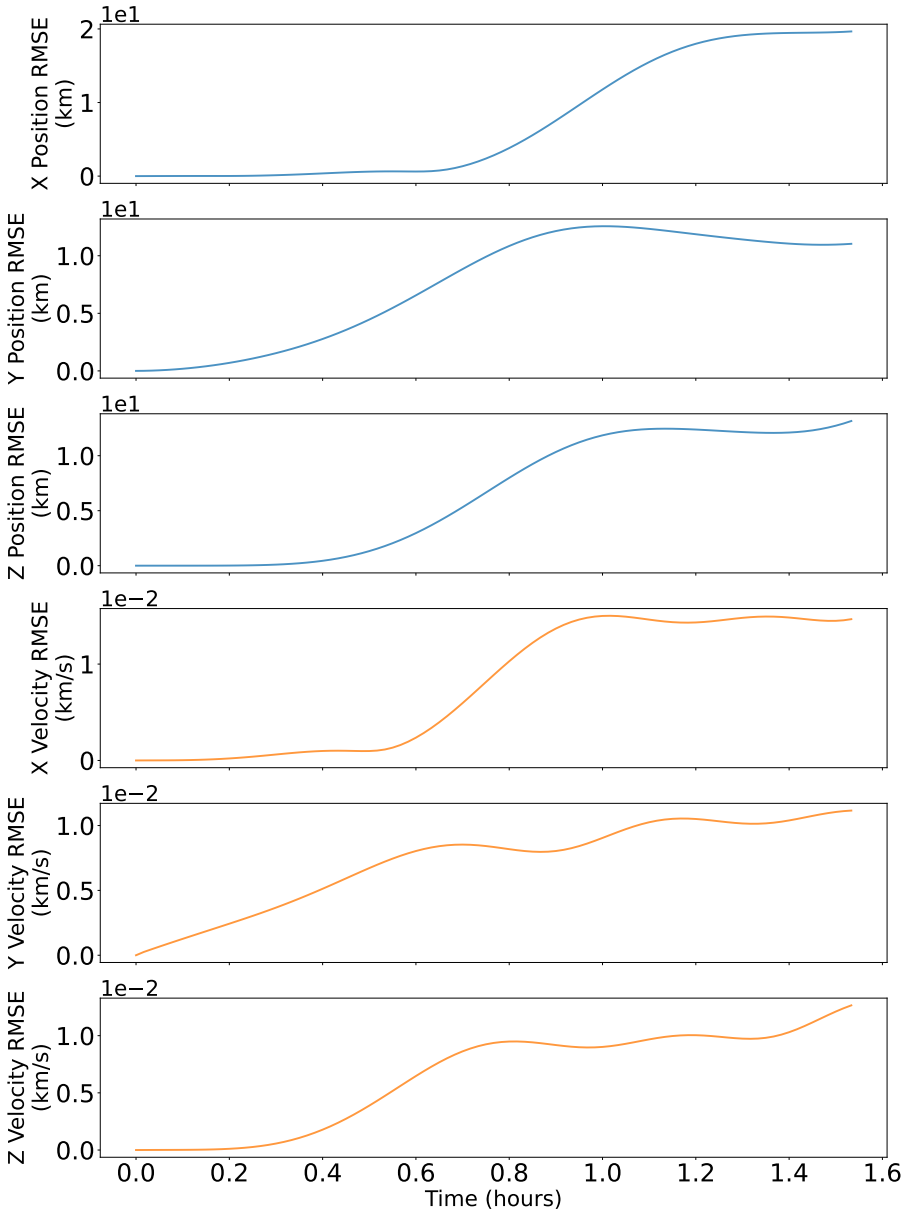


Figure B.23: Evolution of the RMSE for the best performing test orbit using the NODE model with stable Starlink data.