Optimisation of LEO satellites Doppler shift compensation by reducing TLE errors and improving trajectory prediction algorithms

Kushagra Shrivastava[1] (A0281477U)
Rajvrat Thapliyal[1] (A0272677N)

Staff Supervisor: Assistant Professor Abhishek Rai[2]
Lab Mentors: Ma Yuling[2], Yuan Fangxing[2], Fu Naifeng[3]
SPS Mentors: Lim Yi Zhen[1], Hillson Hung[1]

Semester 2 AY24/25

[1] Special Program in Science (SPS), National University of Singapore, Singapore

[2] Satellite Technology and Research (STAR) Centre, National University of Singapore, Singapore

[3] School of Marine Science and Technology, Tianjin University, China

Contents

1 Introduction **2**

2 Background **3**

2.1 LEO satellites and Doppler shift 3

2.2 Current methods to calculate Doppler shift and their challenges 5

2.3 Algorithms analyzed by this work 5

2.3.1 MCS Algorithm 5

2.3.2 ML-∂SGP4 6

3 Methods **6**

3.1 MCS Algorithm Framework 7

3.2 ML-∂SGP4 Architecutre and Training Outlines 9

3.2.1 ML-∂SGP4 Architecture 9

3.2.2 Data Pre-processing 11

3.2.3 Model Training Configuration 13

4 Results and Discussion **14**

4.1 MCS Algorithm 14

4.1.1 Analysis of Loss functions, number of TLEs, and the reference TLE 15

4.1.2 Why did MCS perform worse than SGP4? 17

4.2 ML-∂SGP4 Algorithm 18

4.2.1 Challenges in Training Full ML-∂SGP4 18

4.2.2 Medium ML-∂SGP4 20

4.2.3 Mini ML-∂SGP4 21

5 Limitations and Future Works **23**

6 Conclusion **24**

7 Acknowledgments **25**

8 References **25**

9 Supplementary Materials **27**

9.1 Structure of a TLE 27

9.2 Differentiability example 28

9.3 Errors in SGP4 prediction 28

9.4 Choice of Monte-Carlo Simplex Number 29

Abstract

Low Earth Orbit (LEO) satellites are currently of great importance as their constellations, such as SpaceX's Starlink, are becoming prevalent for communication and scientific research. Their low altitude orbit allows for low latency and less power consumption, but their high speeds result in high Doppler shift that must be accounted for. Current methods employ Two-Line Element (TLE) data with SGP4 algorithm to make satellite trajectory predictions, and thus calculate the Doppler shift. However, these methods suffer from inherent error accumulation. This study analysed two algorithms, the Monte-Carlo Simplex (MCS) and the Machine Learning differential SGP4 (ML-∂SGP4) to test their efficacy in trajectory prediction as compared to SGP4. We generalised the MCS algorithm to work with any satellite and found its optimising ability depended on the magnitude of the loss function's output. We also robustly showed that it performed worse than the standard SGP4 in all possible scenarios using TLEs. We accredit this bad performance to independent perturbations in parameters which do not evolve independently, and an unevenly spaced dataset. In addition, we analyze time complexity issues in the ML-∂SGP4 source code along with its original training method which implicitly assumes that error accumulation of all satellites is the same. We proposed a new training paradigm for the ML-∂SGP4 architectural template, using mini-ML-∂SGP4 models that can be trained on a single satellite dataset. This approach was applied to three Starlink satellites with varying orbital periods. Over a propagation period of eight days with a temporal resolution of 60 seconds, the model trained on STARLINK-5350 data achieved a significant reduction in error across all time stamps compared to the standard SGP4 propagator. Moreover, 81% of its datapoints showed a marked improvement over a pretrained ML-∂SGP4 model in literature. Since this training paradigm requires only a relatively small dataset—approximately 40,000 rows, the associated propagation error reduction is achieved with substantially lower data requirements. This efficiency decreases the lengthy training times associated with the original ML-∂SGP4 implementation. In the future, individual parameter perturbation analysis on MCS and further testing of the mini-ML-∂SGP4 models could improve trajectory predictions, creating a reliable method for calculating Doppler shifts and communicating with LEO satellites.

1 Introduction

Low-Earth Orbit (LEO) satellites are becoming the default for communication and scientific research, especially with SpaceX's Starlink constellations populating the night sky, replacing ground-based methods. The number of satellites have exploded, with European Space Agency estimating that more LEO satellites were launched in 2023 than any year prior (European Space Agency, Space Debris Office, 2024).

However, stable ground and LEO satellite link is still a challenge due to Doppler shift (DS), especially for LEO satellites without any Global Positioning System (GPS) on board. DS changes the transmission frequency of the satellite. If the

ground station is unable to calculate or predict the changed frequency, it cannot establish a transmission link with the satellite.

To combat this problem, current methods use predictions of the satellite's position and velocity to calculate DS. They use Two-Line Element (TLE) data and the SGP4 algorithm because it only requires one data point for prediction. However the predictions are only valid for a few days. TLE data is released infrequently and has inherent errors, and SGP4 algorithm accumulates error over time, leading to poor position and velocity predictions. Singapore's LUMELITE-4 and Galassias-2 both suffered from incorrect DS calculations, leading to difficulty in communicating with the satellites (National University of Singapore, 2024a, 2024b).

In this work, we implement and measure the efficacy of the Monte-Carlo Simplex (MCS) algorithm (inspired by Liu et al., 2023) to optimise TLE, and the Machine Learning differentiable SGP4 (ML-∂SGP4) algorithm (inspired by Acciarini et al., 2025) to reduce error accumulation present in standard SGP4. A better trajectory prediction model allows for improved DS prediction to maintain transmission links with the satellite.

2 Background

2.1 LEO satellites and Doppler shift

This work focuses on LEO satellites as they are the best option for communication. Satellites link with ground stations using Radio Frequencies (RF). As the speed of RF is a constant regardless of frequency in vacuum (Einstein, 1905), it dictates there will be a greater propagation delay with increasing satellite altitude. LEO satellites are defined as satellites with orbital period less than 225 minutes (Hoots and Roehrich, 1988). This corresponds to a semi-major axis of less than 12000 km which is much smaller than other orbits such as Geostationary with semi-major axis of 42000 km. As LEO is closest to the Earth, it provides low latency link and also requires less power, making it optimal for communication purposes.

However, the small orbital period corresponds to a high tangential speed, and consequently a high relative speed between the Earth's surface and the satellite. This induces a Doppler shift — a change in frequency observed by a stationary observer (ground station), as compared to the frequency emitted by a moving emitter (the satellite). Furthermore as the satellite moves above the ground station, the Doppler shift changes. If the ground station is unable to match the Doppler shifted frequency, it cannot establish links with the satellite. Therefore, the ground station must calculate or predict the new frequency.

Mathematically, the normalised Doppler shift is represented by,

$$\frac{f_o - f_e}{f_e} = -\frac{1}{c}\frac{\mathrm{d}|\mathbf{R}|}{\mathrm{d}t} \tag{1}$$

where f_o, f_e, c, and $\mathbf{R}$ corresponds to observed frequency, emitted frequency, speed of light, and the displacement vector connecting the ground station to the satellite (commonly known as the trajectory of the satellite) respectively. Measurements are performed in a non-inertial coordinate system which rotates at the angular speed of the Earth.

$\mathbf{R}$ will generally be a function of position of

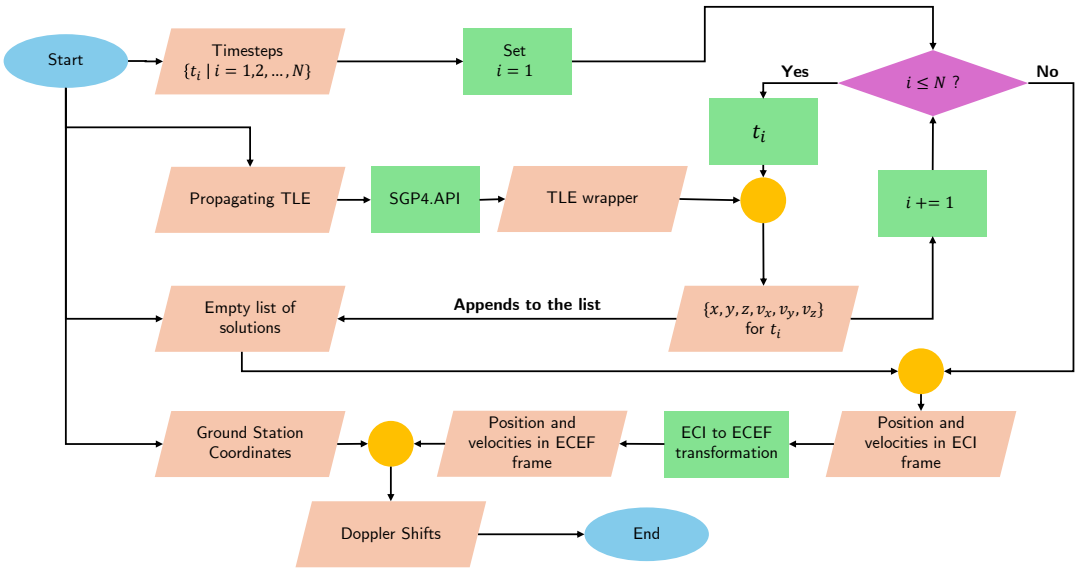

Figure 1: Flowchart of how to calculate Doppler shifts using SGP4. We input the time steps at which we want to calculate the position and velocity of the satellite and a TLE. Using the `sgp4.api` available for `Python`, we can loop the TLE over all the inputted time steps to obtain the positions and velocities in an Earth Centred Inertial (ECI) frame. In order to calculate the Doppler shifts with respect to a ground station, we need to convert the ECI frame into an Earth Centred Fixed (ECEF) frame and account for the coordinates of the Ground Station.

satellite, velocity of satellite, and time. It is possible to obtain an analytical solution for equation (1) if we assume no drag and circular orbits (Ali et al., 1998). However, a LEO satellite faces significant atmospheric drag, gravitational perturbation from the Moon, pressure from Solar radiation, and differential gravitational force from Earth's non-spherical shape (Yll, 2018). Overall, the evolution of $\mathbf{R}$ is determined by non-linear differential equations and thus, there is no general analytical solution to find the Doppler shift of a satellite (Acciarini et al., 2025).

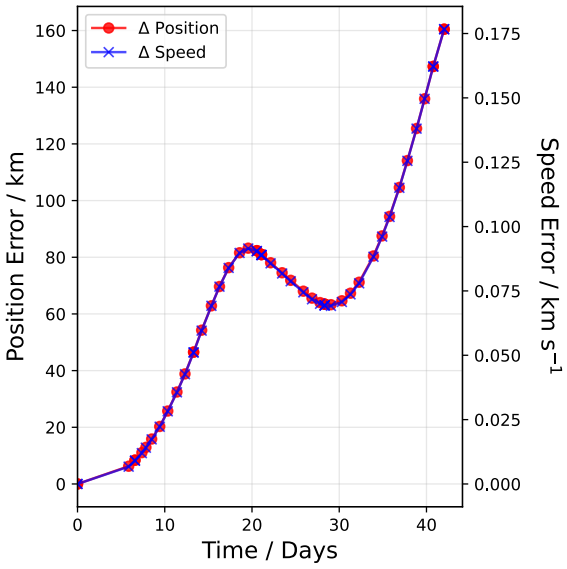

Figure 2: Increasing Position and Speed Error from SGP4 for KENT RIDGE 1 satellite. The position and speed error scale together and are thus shown on the same graph. Day 0 is defined as the day when the TLE was released, and it corresponds to 2460374.79645876 Julian Day. Refer to section 3 for the definition of error.

2.2 Current methods to calculate Doppler shift and their challenges

The most optimal method to calculate Doppler shift is to make use of Global Positioning System (GPS) technology (Asraf et al., 2021; Jason et al., 2004), however not all satellites can or already have a GPS system on board.

In the absence of GPS, there are two methods. First, compensate for Doppler shift through signal-based algorithms at the ground station or satellite (Yll, 2018). Second, improve the prediction of trajectory through numerically calculating **R** (Acciarini et al., 2025; Hoots and Roehrich, 1988; Liu et al., 2023). In this work, we focus on the latter because such predictions are agnostic of the equipments being used at the ground station or inside the satellite.

The Simplified General Perturbation (SGP) algorithms were created in 1980s to numerically obtain position and velocity of a satellite starting from some initial conditions. Out of the five, SGP4 was made for LEO satellites and has since been used to predict trajectories (Hoots and Roehrich, 1988). The initial conditions of all satellites are provided by North American Aerospace Defense Command (NORAD) in the form of a Two-Line Element (TLE) (Kelso, 2022). Refer to Supplementary Materials Section 9.1 for breakdown of TLE elements. Using TLE along with SGP4 allows us to calculate the position and velocity of a satellite at any given time. Figure 1 shows how to obtain the trajectory and thus the Doppler shift in `Python` using `sgp4.api`.

However TLEs are published infrequently without any error ranges, and SGP4 has inherent errors that accumulate over time. This is shown in Figure 2 using KENT RIDGE 1 satellite as an example. In SGP4 both position and speed errors scale together and are thus shown on the same graph. In only 10 days, the position error has increased by 20 km showcasing the instability of SGP4.

2.3 Algorithms analyzed by this work

Given the challenges in position and velocity prediction by current methods, this work analyses two algorithms proposed in literature to measure their efficacy in trajectory prediction.

2.3.1 MCS Algorithm

The Monte-Carlo Simplex algorithm is a minimization algorithm which attempts to find the global minima of a function. Liu et al., 2023 uses it to optimise one reference TLE in a set of TLEs, by minimizing the difference between actual and predicted values inside the set. It is a combination of two independent algorithms: the Nelder-Mead Simplex algorithm, which finds a minima, and the Monte-Carlo Algorithm, which ensures the minima is global. The algorithm inputs a set of TLEs with a chosen reference TLE, and outputs an optimised TLE. In their work, Liu et al. created a semi-analytical propagator and developed the MCS algorithm for one reference TLE and they observed a systematic shift in error. However it could not be deduced whether this improvement was due to MCS algorithm or their new propagator. Our work generalises the MCS Algorithm for any satellite and

uses the standard SGP4 as the propagator.

2.3.2 ML-∂SGP4

The ML-∂SGP4 model is a physics-aware machine learning architecture developed by Acciarini et al., 2025. It leverages the capabilities of the ∂SGP4 implementation in PyTorch — a Python package that re-implements the standard SGP4 orbit propagator within the PyTorch deep learning framework. One of PyTorch's key advantages is its support for automatic differentiation (AD), which enables efficient computation of partial derivatives essential for gradient-based optimization. As described by (Acciarini et al., 2025), this feature makes it possible to integrate orbit propagation directly into neural network training pipelines, allowing model parameters to be optimized using the backpropagation algorithm (Rojas, 1996). This is not possible with the standard SGP4, which is inherently non-differentiable (Acciarini et al., 2025). Refer to Supplementary Material 9.2 for an example showcasing the benefit of differentiability.

The use of neural networks is particularly well-suited for the task of correcting errors in satellite orbit propagation, as they excel at learning complex, non-linear mappings (Hussain et al., 2023) — a strength leveraged by the ML-∂SGP4 architecture. This enables ML-∂SGP4 to capture and correct non-linear error patterns that commonly arise in orbit prediction tasks as shown by the position and velocity error curves of the Kent Ridge 1 satellite in Figure 2.

The ability to propagate gradients through the orbital dynamics module allows ML-∂SGP4's neural networks to learn corrections to both the TLE input data and the propagated state vectors in a physics-informed manner, guided by the differentiable ∂SGP4 algorithm. This synergy is validated by the model's performance: ML-∂SGP4 achieves a 33.42% reduction in test loss on an unseen dataset compared to the standard SGP4 propagator (Acciarini et al., 2025).

Inspired by these results, we retrain ML-∂SGP4 from scratch using updated STARLINK ephemeris and TLE data. In addition to the full model, we introduce and evaluate several "lightweight" variants of ML-∂SGP4. These models retain the original architecture but are trained on smaller or more targeted subsets of the dataset. This approach enables us to benchmark performance under various data constraints and to explore more specialized applications, such as correcting errors for individual satellites with distinct orbital characteristics.

3 Methods

This section covers detailed description of the algorithms used. For both algorithms, the errors at a given time ($\Delta r(t_i)$ and $\Delta v(t_i)$) are defined as follows. Consider at time t_i, the real position and velocity of the satellite is $\mathbf{r}(t_i)$ and $\mathbf{v}(t_i)$ respectively in an Earth-Centered Inertial coordinate system. At the same time, let the predicted position and velocity be $\mathbf{r}'(t_i)$ and $\mathbf{v}'(t_i)$ respectively in the same coordinate system. Therefore,

$$\Delta r(t_i) = |\mathbf{r}(t_i) - \mathbf{r}'(t_i)| \tag{2}$$

$$\Delta v(t_i) = |\mathbf{v}(t_i) - \mathbf{v}'(t_i)| \tag{3}$$

The algorithm through which $\mathbf{r}'(t_i)$ and $\mathbf{v}'(t_i)$ were predicted is called the **propagator** hereon.

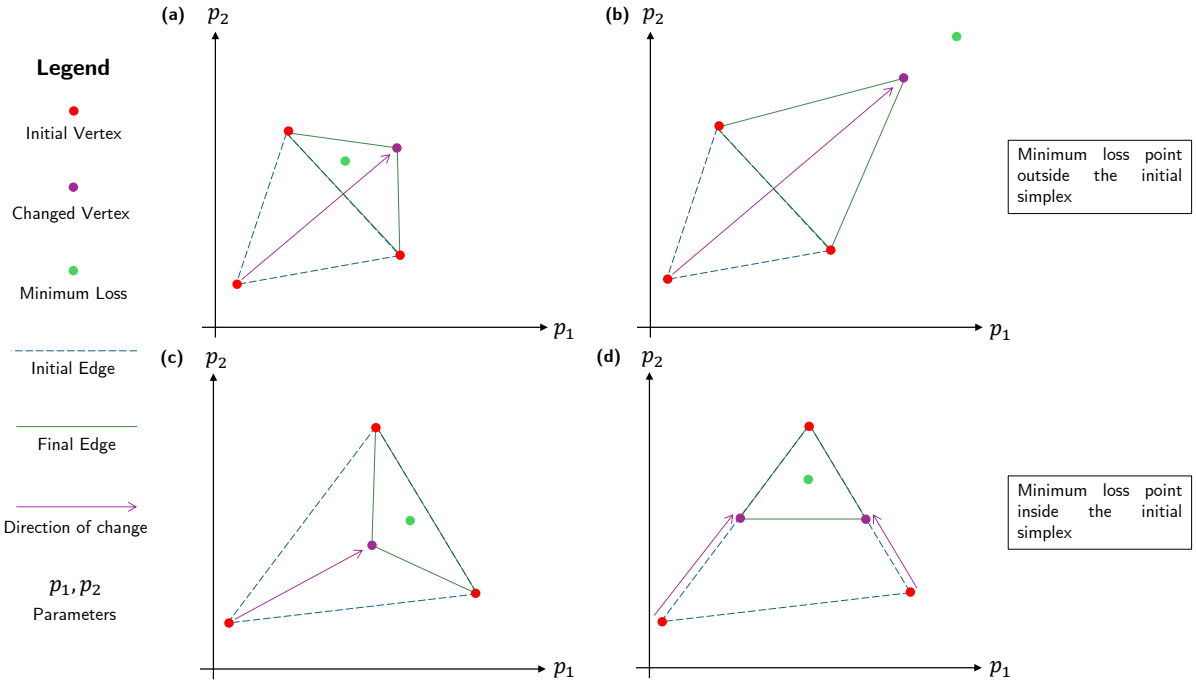

Figure 3: Simplified representation of a 2-dimensional simplex exploring the parameter space. p_1 and p_2 represents two parameters, for example eccentricity and inclination. Each point in the parameter space corresponds to a loss, and the simplex transforms itself to find the minimum loss in this space. The three vertex of the simplex each have a different error, and the highest error vertex moves first. (a) and (b) corresponds to the case where the minimum point is located outside the simplex. The simplex reflects the vertex with the highest error about its opposite side. If the minimum is closer to the edge, it changes as shown in (a). If the minimum is further away, it changes as shown in (b). If the minimum is located inside the simplex, it transforms like (c) or (d) depending on whether the minimum is closer to a vertex or an edge. In the MCS algorithm, we create multiple random simplex so that most of them converge towards the global minima of the parameter space.

The source code can be found here.

3.1 MCS Algorithm Framework

The first prerequisite of the algorithm is defining a state vector. As described in section 9.1, a TLE is made up of multiple elements of which not all can be optimised. We used the same state vector as Liu et al., 2023 $\mathbf{X}$ with the optimisable parameters as,

$$\mathbf{X} = \begin{pmatrix} B^* & i & \Omega & e & \omega & M & n \end{pmatrix}^{\mathrm{T}} \tag{4}$$

where B^* is Drag, i is inclination, Ω is Right Ascension of Ascending Node (RAAN), e is eccentricity, ω is Argument of Perigee, M is mean anomaly and n is mean motion. Refer to section 9.1 for definition of each term. These elements were chosen as they are used in SGP4 to predict position and velocity.

The second prerequisite is a loss function whose global minima is found by the algorithm. Suppose a set of N TLEs were used in the algorithm, labeled $\{T_1, T_2, ..., T_N\}$ with T_k being the reference TLE in the set. Each TLE has an Epoch Time at which it is supposed to be correct (refer to section 9.1). Let the epoch times associated with T_1 be t_1, T_2 be t_2, and so on. SGP4 is used to predict the position $\mathbf{r}_i$ and velocity $\mathbf{v}_i$ for each TLE T_i at its epoch time t_i. Therefore,

$$\mathrm{SGP4}(T_i, t_i) = (\mathbf{r}_i, \mathbf{v}_i), \quad i = 1, 2, ...N \tag{5}$$

Then, SGP4 is used to predict the position $\mathbf{r}'_i$ and velocity $\mathbf{v}'_i$ at all the epoch times using just the reference TLE, T_k,

$$\text{SGP4}(T_k, t_i) = (\mathbf{r}'_i, \mathbf{v}'_i), \quad i = 1, 2, ...N \tag{6}$$

The Loss function defined by Liu et al., 2023 was,

$$L_{\text{standard}} = \sum_{i=1}^{N} \left(|\mathbf{r}_i - \mathbf{r}'_i|^2 + |\mathbf{v}_i - \mathbf{v}'_i|^2 \right) \tag{7}$$

However, it was observed during preliminary runs that the position error is three orders of magnitude higher than the velocity error (also observable in Figure 2). Therefore this work further defined a weighted function to check whether it affects the final result,

$$L_{\text{weighted}} = \sum_{i=1}^{N} \left(|w(\mathbf{r}_i - \mathbf{r}'_i)|^2 + |\mathbf{v}_i - \mathbf{v}'_i|^2 \right) \tag{8}$$

where $w = 10^{-3}$. Two more loss functions were defined to check whether position or velocity independently affects the algorithm,

$$L_{\text{r}} = \sum_{i=1}^{N} |\mathbf{r}_i - \mathbf{r}'_i|^2 \tag{9}$$

$$L_{\text{v}} = \sum_{i=1}^{N} |\mathbf{v}_i - \mathbf{v}'_i|^2 \tag{10}$$

Using the loss functions and the state vector, the Nelder-Mead Simplex algorithm can be initiated. A simplex refers to a shape that can be made in a N-dimensional space with $N+1$ vertex. Thus a point in 1-dimension, a triangle in 2-dimensions, a tetrahedron in 3-dimensions and so one. Each vertex corresponds to a perturbed state vector. As previously defined, our state vector is 7-dimensional and hence is impossible to visualize the simplex.

Using the reference TLE T_k, the initial state vector $\mathbf{X}_k$ is created. This vector acts as the first vertex of the simplex. The other six are created by randomly perturbing $\mathbf{X}_k$ with a vector $\hat{\delta}_i$. $\hat{\delta}_i$ is a vector where all elements are 0 except the i-th element, which is sampled from a normal distribution. Using a 2 element state vector as an example, the three vertex will be,

$$\mathbf{Vert}_1 = \mathbf{X}_k = \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \tag{11}$$

$$\mathbf{Vert}_2 = \mathbf{X}_k + \hat{\delta}_1 = \begin{pmatrix} a_1 + \delta_1 \\ b_1 \end{pmatrix} \tag{12}$$

$$\mathbf{Vert}_3 = \mathbf{X}_k + \hat{\delta}_2 = \begin{pmatrix} a_1 \\ b_1 + \delta_2 \end{pmatrix} \tag{13}$$

This simplex exists in a parameter space where each point corresponds to a loss value defined by equations 7 to 10. The simplex transforms itself until it converges on the minimum loss. We set a tolerance value of 10^{-2} for convergence, which means the algorithm stops when the difference in error between two consecutive runs fall below 10^{-2}. Figure 3 shows a toy example with a triangular simplex.

However, the Nelder-Mead algorithm alone is not effective as the simplex might converge on a local minima instead of the global minima. To achieve the global minima, the Monte-Carlo method is used, however this work's implementation is different from Liu et al., 2023. This work generalises the method for any reference TLE and any satellite, for robust testing of the algorithm.

Instead of just using one simplex, N independent initial state vectors are created from $\mathbf{X}_k$ and then used to create N independent simplex. $\mathbf{X}_k$ is

perturbed by a vector $\mathbf{s}_j$. The elements of $\mathbf{s}_j$ are random samples from a normal distribution, with mean 0 and standard deviation 1% of the initial element value. Mathematically, the j-th initial state is created by,

$$\mathbf{X}_{k,j} = \mathbf{X}_k + \mathbf{s}_j = \begin{pmatrix} x_1 + s_{j_1} \\ x_2 + s_{j_2} \\ \vdots \\ x_7 + s_{j_7} \end{pmatrix} \tag{14}$$

where,

$$s_{j_\alpha} \sim \mathcal{N}\left(0, \left(\frac{x_\alpha}{100}\right)^2\right) \tag{15}$$

The output from MCS algorithm is N final optimised state vectors. For each element, the final optimised value is the mean of all the final values, and the error associated with it is the standard deviation. The final optimised values are then converted into a TLE used for position and velocity prediction.

The algorithm was coded in `Python`. `scipy` library's `optimize.minimize()` function was used to create each simplex, with the `method='nelder-mead'` argument. Bounds were placed that restricted the simplex to only explore ±1% of each elements' neighborhood to prevent non-physical situations. We conducted preliminary testing using no bounds, physical bounds and ±10% bounds. In all the cases, SGP4 failed to calculate position and velocity due errors listed in Supplementary Material 9.3. To reduce the computation time drastically, `joblib` library was used to parallelize these calculations. Furthermore as `sgp4.api` requires TLE as a string, and float to string conversion may cause floating point errors, any incorrectly created state vector and the corresponding simplex detected by `sgp4.api` was associated an error of infinity to exclude it.

3.2 ML-∂SGP4 Architecutre and Training Outlines

3.2.1 ML-∂SGP4 Architecture

The ML-∂SGP4's architecture contains two neural network modules, each consisting of three fully connected layers. The first two layers of both networks use leaky ReLU ("LeakyReLU — PyTorch 1.11.0 documentation", n.d.) activation functions with a negative slope of 0.01, while the final layer applies a hyperbolic tangent (tanh) activation function to produce smooth, bounded outputs ("Tanh — PyTorch 1.13 documentation", n.d.). This design ensures that the correction factors applied to orbital elements and state vector components remain physically meaningful and do not introduce instability.

In the first neural network—positioned before the ∂SGP4 propagator, the input TLE parameters are passed through two hidden layers, and the output of the third layer is used to compute a correction factor. This correction is applied multiplicatively to the original input vector as shown in Figure 4. This corrected state vector is then used to update only the six TLE attributes corresponding to the satellite's position and velocity: eccentricity, argument of perigee, inclination, mean anomaly, mean motion, and right ascension of the ascending node.

The corrected TLE is propagated to the desired time using the differentiable ∂SGP4 function. The resulting propagated state vector is then normalized by dividing the position components by the position normalization constant $R_{\text{norm}} = 6958.137$, and the

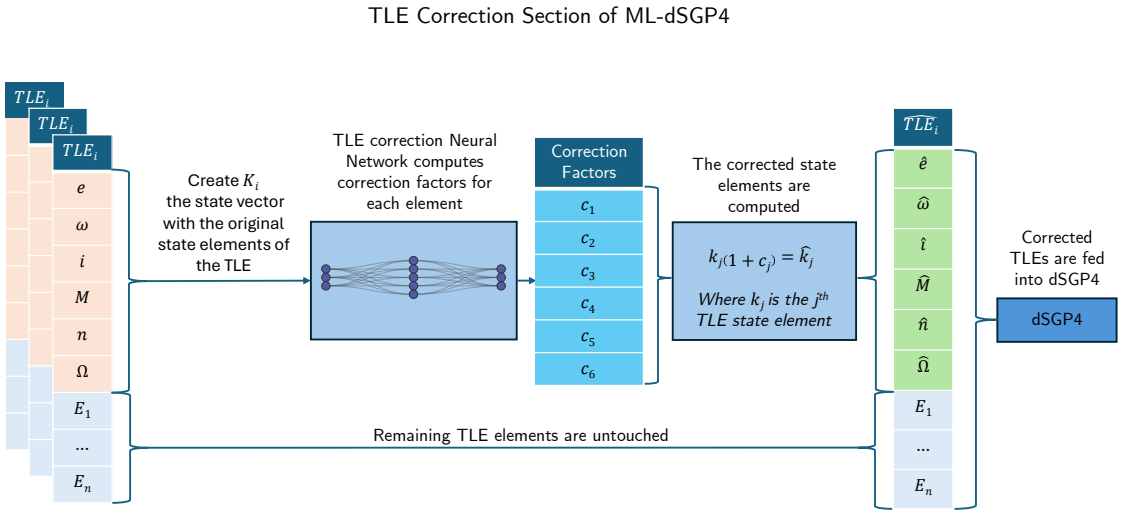

Figure 4: Flowchart of the ML-∂SGP4 TLE correction architecture. The model takes in Python TLE objects (TLE_i), which contain all elements of a TLE. The first neural network in the model attempts to correct the values of state vector elements—shown in red within the TLE_i vectors—which describe the satellite's position and velocity. Using its pre-trained parameters, the neural network produces correction factors c_j for each relevant TLE element. These are used to scale the original values into corrected values $\hat{k}_j$, highlighted in green. TLE elements not targeted for correction remain unchanged. ML-∂SGP4 does not correct the B^* value by design; hence, the state vector passed into the TLE correction neural network excludes it. A final corrected TLE, $\widehat{\text{TLE}}_i$, is then created and passed into the ∂SGP4 propagator.The ∂SGP4 algorithm outputs a propagated state vector in the ECI coordinate system. This output is subsequently refined by a second neural network, which aims to correct errors from the physics-based propagator. While the second stage of ML-∂SGP4's architecture is not shown in this figure, it operates similarly to the first: the neural network takes in the full propagated state vector and applies learned corrections in the same manner as the initial TLE correction network.

velocity components by the velocity normalization constant $V_{\text{norm}} = 7.947155867983262$. These normalization constants were obtained directly from the official ML-∂SGP4 source code (Acciarini et al., 2025).

The normalized state vector is subsequently passed into a second neural network, which mirrors the architecture of the first. This network applies a *second stage of correction* using a similar *multiplicative scaling mechanism*, where each component is adjusted by a learned correction factor constrained by a tanh activation ("Tanh — PyTorch 1.13 documentation", n.d.). The final output is a refined position and velocity vector at our desired time in the ECI coordinate frame, corrected using both *data-driven learning* and the *physical insights embedded in the orbital dynamics* of the ∂SGP4 propagator.

Figure 4 gives us intuition for ML-∂SGP4's TLE correction architecutre and the forward pass, (computations made by the neural network to make a prediction). However, to gain a clearer understanding of how a neural network learns, we can think of the neural network as a function f with a set of parameters θ. These parameters are adjusted during training to learn a mapping from input values to the desired optimal values. The learning process

involves repeated *forward passes*, where the network predicts outputs given the inputs and current weights, and *back propagation*, where it computes the error (or "loss") and calculates gradients of this loss with respect to each weight. These gradients are then used to update the weights in the direction that minimizes the loss.

3.2.2 Data Pre-processing

All TLE data used in this work was obtained using a custom Python script developed by us to make API requests to the Space-Track.org ("Space-Track.Org", n.d.), a satellite tracking website. The ephemeris data, described below, was collected in a similar manner. All data was stored and processed in a PostgreSQL database hosted locally on a personal machine.

We initially adopted a workflow similar to that of Acciarini et al., 2025, using SpaceX's STARLINK TLE and ephemeris data to construct the training, validation, and testing datasets. The ephemeris data (used as the ground truth in this study) was sourced from space-track.org ("Space-Track.Org", n.d.) and covers 6,613 unique satellites over the period from 22 March 2025, 20:50:42 UTC to 26 March 2025, 04:10:42 UTC, with a temporal resolution of 60 seconds. SpaceX generates this ephemeris data using a proprietary high-precision orbit propagator (Acciarini et al., 2025), and we treat these predictions as the best available proxy for true satellite positions and velocities as done by Acciarini et al., 2025.

Ideally, onboard GPS data would serve as the most accurate ground truth, as it directly reflects the satellite's true state vector. However, despite extensive efforts, we were unable to locate any publicly available GPS data.

For each satellite in the dataset, its TLEs are propagated to the various timestamps of its corresponding ephemeris observations. Additionally, the propagation time—defined as the time difference between the TLE observation and the corresponding ephemeris observation—is included in each datapoint. Multiplying the number of satellites by the number of TLEs observed per satellite and by the number of ephemeris timestamps yields a dataset comprising approximately 288 million rows. This dataset occupies roughly 90 GB within the PostgreSQL database. Moreover, our dataset is substantially larger, containing 6613 satellites compared to the 1519-satellite dataset used by Acciarini et al., 2025.

A training, validation, and test set was then constructed. The training set was used to perform forward propagation and compute the training loss, which was then used during backpropagation to update the model's parameters (Rojas, 1996) —this forms the core of the learning process. When the model completes a full forward and backward pass over the entire training set, one epoch is said to be completed. At the end of each epoch, the model is evaluated on the validation set by computing a validation loss. While this value is not used for weight updates, it serves to assess generalization performance (Barinov et al., 2023) and evaluate whether the model has overfit to the training data. If the model successfully learns an error-minimizing function, the training and validation losses are expected to converge and plateau at low value over time (Barinov et al., 2023). The test set, on the

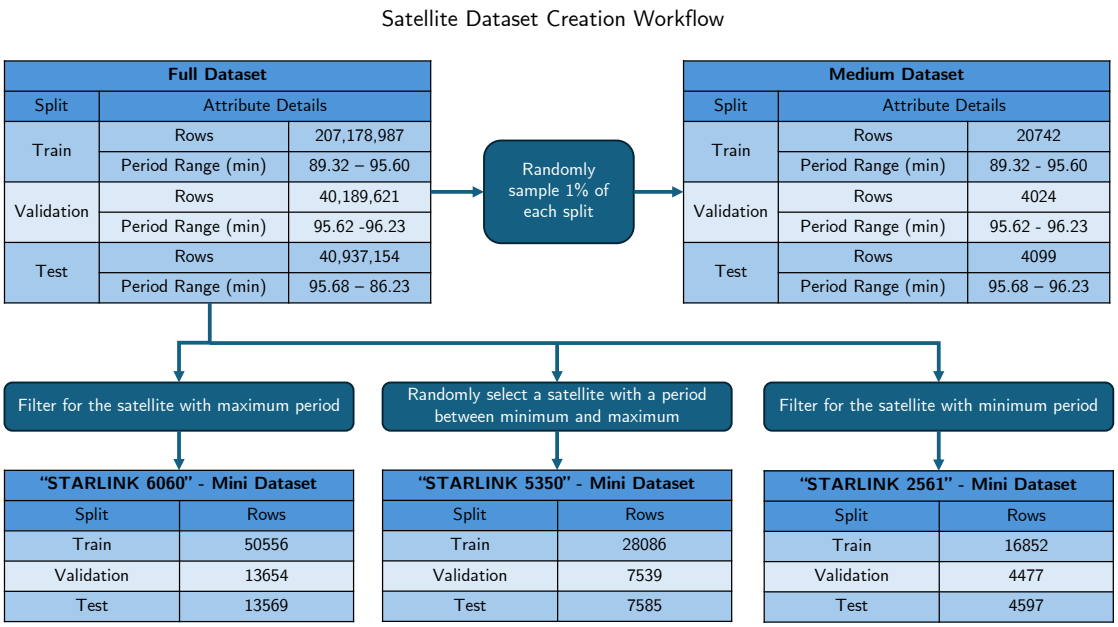

Figure 5: Flowchart of dataset generation used in this study. The full dataset was prepared using the data of all satellites present in both the ephemeris and TLE data. The medium dataset was created by randomly sampling 1% of rows from each split of the full dataset. Its satellite period range remains similar to the full dataset. For the mini datasets, we filtered the full dataset for satellites based on specific period criteria: the maximum, minimum, and a randomly chosen "middle" value. STARLINK 6060, STARLINK 5350, and STARLINK 48403 were selected respectively. The 65th percentile of propagation time was used as a threshold for splitting into training and validation/test sets for the mini datasets. The test and train sets were created by randomly sampling the the validation/test set to create largely equal splits

other hand, serves as a fully unseen dataset, used exclusively for final evaluation after training is complete.

To divide the full dataset, we first isolated all satellites with orbital periods roughly below 95.63 minutes and assigned them to the training set—accounting for roughly 70% of the total data. The remaining 30% was randomly sampled and evenly split into testing and validation sets (approximately 15% each). This partitioning was chosen to evaluate our version of ML-∂SGP4's ability to generalize across satellites with diverse orbital dynamics. The row counts and period ranges for each split are presented in the "Full Dataset" table in Figure 5.

In addition to the full dataset, we constructed a medium-scale dataset by randomly sampling 1% of each split (training, validation, and test). This allowed us to train a reduced model—referred to as Medium-ML-∂SGP4—within a more feasible time frame, while still preserving the period diversity of the full dataset.

To further investigate specialized versions of ML-∂SGP4 we created a set of satellite-specific mini datasets. These were constructed by selecting the satellites with the maximum and minimum orbital periods in the dataset, along with one randomly chosen satellite with period value between the maxium and minimum values. The goal of these mini datasets was to train lightweight versions of ML-

∂SGP4 tailored to individual satellites. These models are designed to specialize in correcting propagation errors unique to their respective orbital dynamics when passed through the ∂SGP4 model.

A summary of the dataset creation workflow, as well as row counts for each data split and dataset version, is shown in Figure 5.

3.2.3 Model Training Configuration

The hyperparameter configuration for ML-∂SGP4, used in Acciarini et al., 2025 was adopted. All models were trained for 30 epochs with a learning rate of 0.003. Mean squared error (MSE) was used as the loss function, and the Adam optimizer was employed for gradient-based optimization. Under this configuration, the total number of trainable parameters was 3,454.

As an additional hyperparameter, we introduced a batching mechanism. In an ideal setting, the entire training dataset can be passed through the neural network simultaneously as a single matrix. However, loading the entire dataset into GPU memory is often infeasible—especially for large datasets such as ours. Batching enables smaller subsets of the dataset (batches) to be passed into the network sequentially during training. After a forward pass is computed on each batch, the mean loss across the batch is used in backpropagation to update the model weights. Once all batches have been processed, one epoch is considered complete.

Using this configuration, five models were trained using the same architectural template. The first, referred to as *Full ML-∂SGP4*, was trained on the full dataset. The second model, *Medium ML-∂SGP4*, was trained on the medium-scale dataset and summarized in Figure 5. The source code of ML-∂SGP4 was adapted from Acciarini et al., 2025 and tweaked for our study.

Additionally, three mini models—each trained on one of the satellite-specific mini datasets—were developed. These datasets corresponded to STARLINK 6060 (maximum orbital period), STARLINK 5350 (minimum orbital period), and STARLINK 48403 (a randomly selected satellite with a middle-range period), as shown in Figure 5. These models were intended to evaluate how well the architecture performs when specialized for individual satellites with distinct orbital dynamics.

For models training on the medium and full datasets, we utilized personal computing resources and Graphic Processing Unit (GPU) nodes provided by the National Supercomputing Centre (NSCC) Singapore. The required environment was configured on the compute node using miniforge3. The larger models were trained on a supercomputer node equipped with a single NVIDIA A100 GPU (40GB VRAM), 16 CPU cores, and 512GB of RAM. In contrast, training for the mini models was conducted on a personal machine featuring an NVIDIA RTX 4070 Laptop GPU (16GB VRAM) and 8 CPU cores. The `num_workers` parameter in PyTorch's `DataLoader`—which controls parallel data loading from RAM to GPU—was set to 12 on the supercomputer and 4 on the personal machine. A wall time of 12 hours was allocated for each supercomputer training job, allowing models to train overnight and be evaluated the following day.

To further optimize training time, we pre-generated the Python TLE objects required for compatibility with the ∂SGP4 propagation algo-

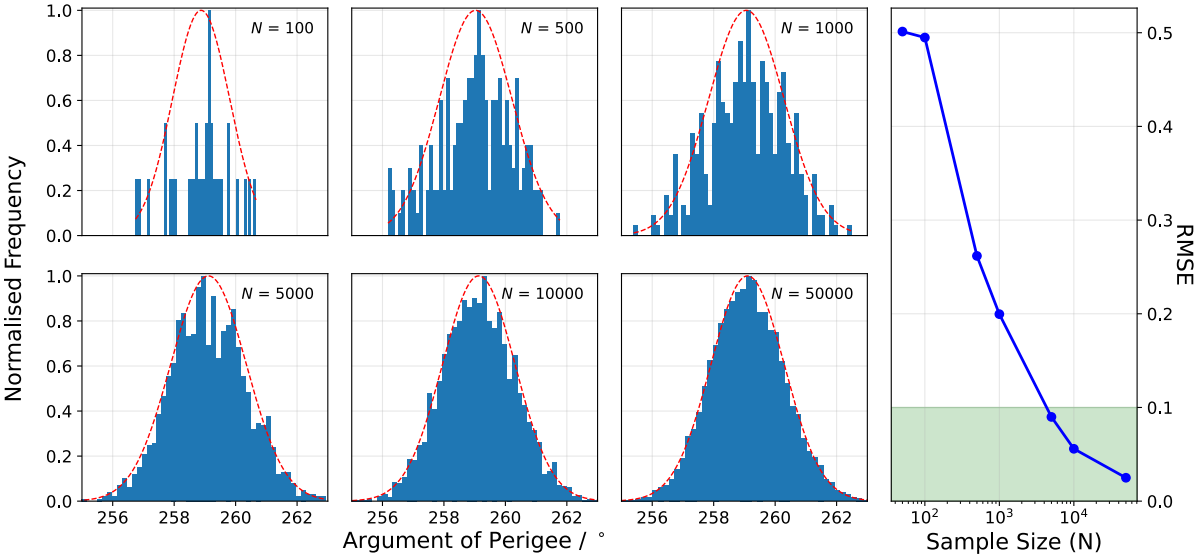

Figure 6: Histograms (left) showing the distribution of optimised Argument of Perigee of POEM-2 satellite for different numbers of MCS, with the RMSE values against Sample Size (right) for the gaussian fit. Better normal distribution is obtained with higher number of MCS. Accounting for computation time and a low RMSE below 0.1 for the gaussian fit, we decided to generate 5000 MCS for each run. Refer to Supplementary Material 9.4 for the distribution of all seven optimised elements (note that $N = 50$ histogram is not shown here).

rithm. This preprocessing was performed in a separate script prior to model training. Specifically, a loop iterated over each TLE in its text format and applied the `dsgp4.tle.TLE()` constructor to convert it into a Python TLE object. These objects were then serialized using `pickle` to avoid repeated conversions during training. This optimization was done to address a computational bottleneck found in the `dsgp4` source code, which is discussed in detail in the Results and Discussion section.

4 Results and Discussion

4.1 MCS Algorithm

Our implementation of MCS Algorithm had three variables for different satellites: (1) number of TLEs (T), (2) which reference TLE to use within the TLE set (f), and (3) the different loss functions. The number of TLEs is the initial amount of TLEs used inside the loss function. When the TLEs were placed in increasing order of epoch time, f is a number between 0 and 1 which refers to the reference TLE. $f = 0$ refers to the first TLE of the set, and $f = 1$ refers to the last. $f = 0.5$ refers to the TLE closest to an epoch time 0.5 times of the last TLE in the set. Furthermore, we used historical data for analysis. The satellite TLEs for MCS were provided by STAR, with the first TLE dating 1st January 2024. The TLEs were roughly spread across 13 months until February 2025. These TLEs were treated as true satellite parameters at their epoch time.

To ensure each simulation was statistically significant, the minimum number of simplex N had to be determined beforehand that provides a good normal distribution fit, without being computationally taxing. The Root Mean Square Error (RMSE) was used as a metric to judge the fit. The RMSE

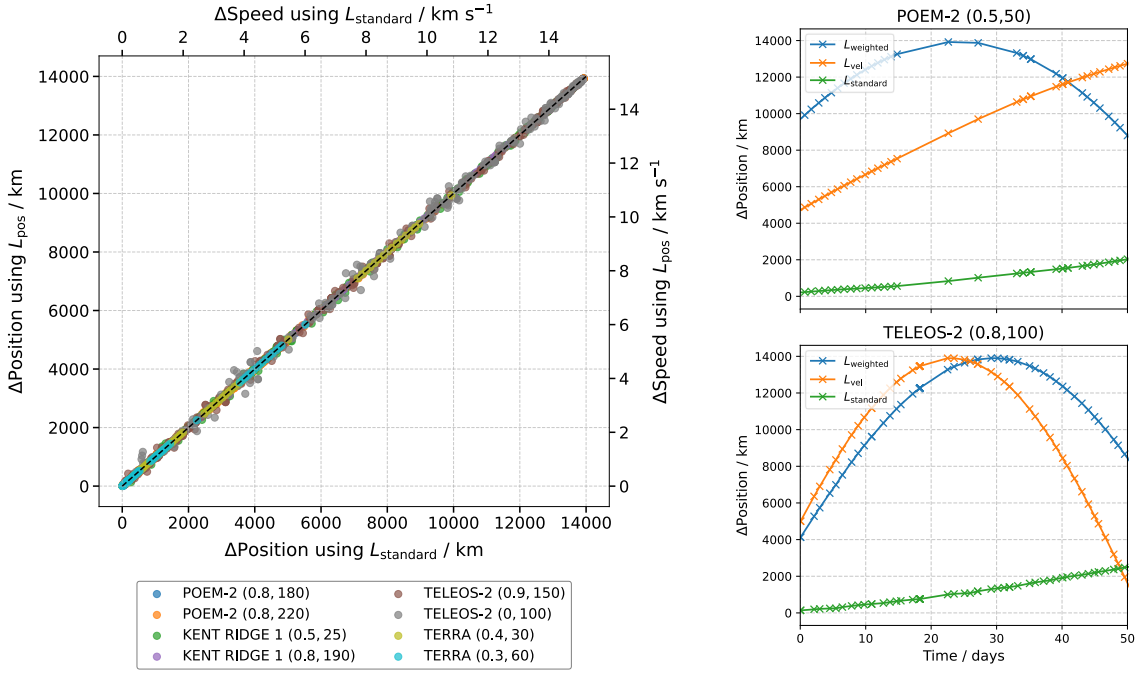

Figure 7: Error plot of L_{pos} against $L_{standard}$ (left) for multiple satellites, and the position error against time for different error functions (right). The legend in the figure is given in the format of "Satellite Name (f,T)". The figure on the left showcases that position and speed error scale together, as expected from SGP4 propagation, and also the L_{pos} loss functions equally as well as the $L_{standard}$ function. The figure on the right shows the error in position propagation for 50 days after the last TLE in the set was released. The markers represents the data points. The final error due to $L_{weighted}$ and L_{vel} was always worse off $L_{standard}$ and the example here shows the same (only position error is shown as the speed error scales with it).

was computed by finding the difference between the actual height of the bin and the height predicted by the fit. Figure 6 shows the histograms for different N using Argument of Perigee as an example. It also shows the RMSE values against different sample sizes. Refer to Supplementary Section 9.4 for histograms of all the parameters. As observed, $N = 5000$ has a good normal distribution fit along with a RMSE less than 0.1. While higher N decreases the RMSE, the decrease is not significant enough to justify the increased computation time. Thus, we decided to use $N = 5000$ for all the simulations hereafter.

4.1.1 Analysis of Loss functions, number of TLEs, and the reference TLE

Figure 7 shows errors obtained by different Loss functions. It was observed that the predictions by $L_{standard}$ and L_{pos} produced extremely similar error.

As shown in the figure, both position and speed errors lie on the $y = x$ line, with a R^2-value of 0.99975. This shows the two functions are effectively similar. Given that the position error is three magnitudes greater than the speed error, looking back at equation 8, $|(\mathbf{r}_i - \mathbf{r}'_i)|^2 \gg |(\mathbf{v}_i - \mathbf{v}'_i)|^2$. Therefore,

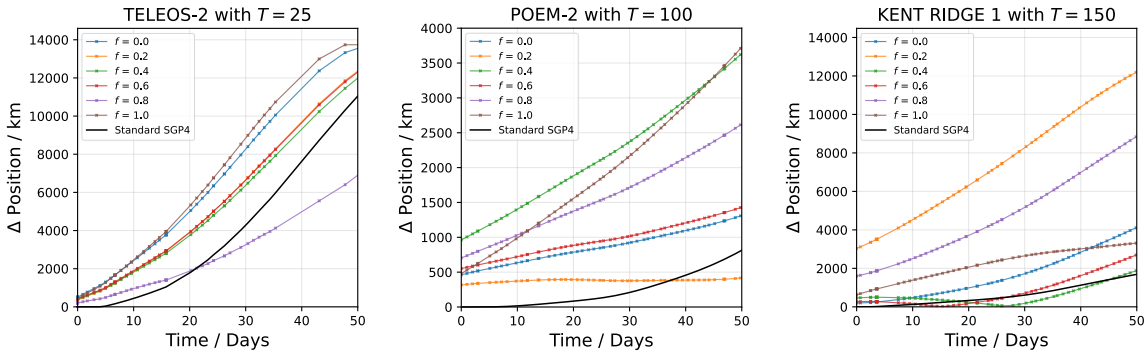

Figure 8: Position Error over 50 days for different reference TLE. The markers represents the data points. Day 0 is defined to be the epoch time of the final TLE in the TLE set. The speed error is not shown as it scales similarly to the position error. None of the examples shown here has any reference TLE which performed better than SGP4, suggesting a failure of the MCS algorithm.

$$|(\mathbf{r}_i - \mathbf{r}'_i)|^2 \gg |(\mathbf{v}_i - \mathbf{v}'_i)|^2 \tag{16}$$

$$\Rightarrow |(\mathbf{r}_i - \mathbf{r}'_i)|^2 + |(\mathbf{v}_i - \mathbf{v}'_i)|^2 \approx |(\mathbf{r}_i - \mathbf{r}'_i)|^2 \tag{17}$$

$$\Rightarrow L_{\text{standard}} \approx L_{\text{pos}} \tag{18}$$

Hence we decided to use L_{standard} in place of L_{pos}.

Furthermore, it was observed that the error using L_{weighted} and L_{vel} was consistently higher than L_{standard}. Figure 7 shows an example for POEM-2 with $f = 0.5$ and $T = 50$ and TELEOS-2 with $f = 0.8$ and $T = 100$. This implied the loss functions created by us were worse than the standard defined by Liu et al., 2023, and thus L_{standard} was the only loss functions used for the other simulations.

As the position error is much larger than the speed error, only minimising the speed error forces the prediction of the MCS algorithm to be worse off. We hypothesize that due to the smaller magnitude, the distance between a minima and a point slightly further away from the minima is very small. Therefore the algorithm might not be converging on the minima. This explain why both L_{vel} and L_{weighted} performed much worse due to their overall small magnitude, whereas the other two functions performed better due to their much higher magnitude of error. This shows that MCS algorithm is sensitive to the magnitude of output of the Loss functions.

However, regardless of different number of TLE (T) and different reference TLE used (f), the MCS algorithm failed to outperform SGP4 in all tested scenarios. Figure 8 shows examples of error increment when the reference TLE is changed but the number of TLEs is kept constant. We initially guessed that within each TLE set, choosing a reference TLE closer to the end of the set will result in a better optimised TLE because we are still using SGP4 to propagate and its inherent error increases with increasing number of days. Yet, this general trend was not observed. For example in Figure 8, POEM-2's error of $f = 1$ increased much faster than $f = 0.2$. Most importantly, no combination had a lower error than the standard SGP4. Similarly, keeping the reference TLE number constant and changing the total number of TLEs did not improve the algorithm prediction either. As shown

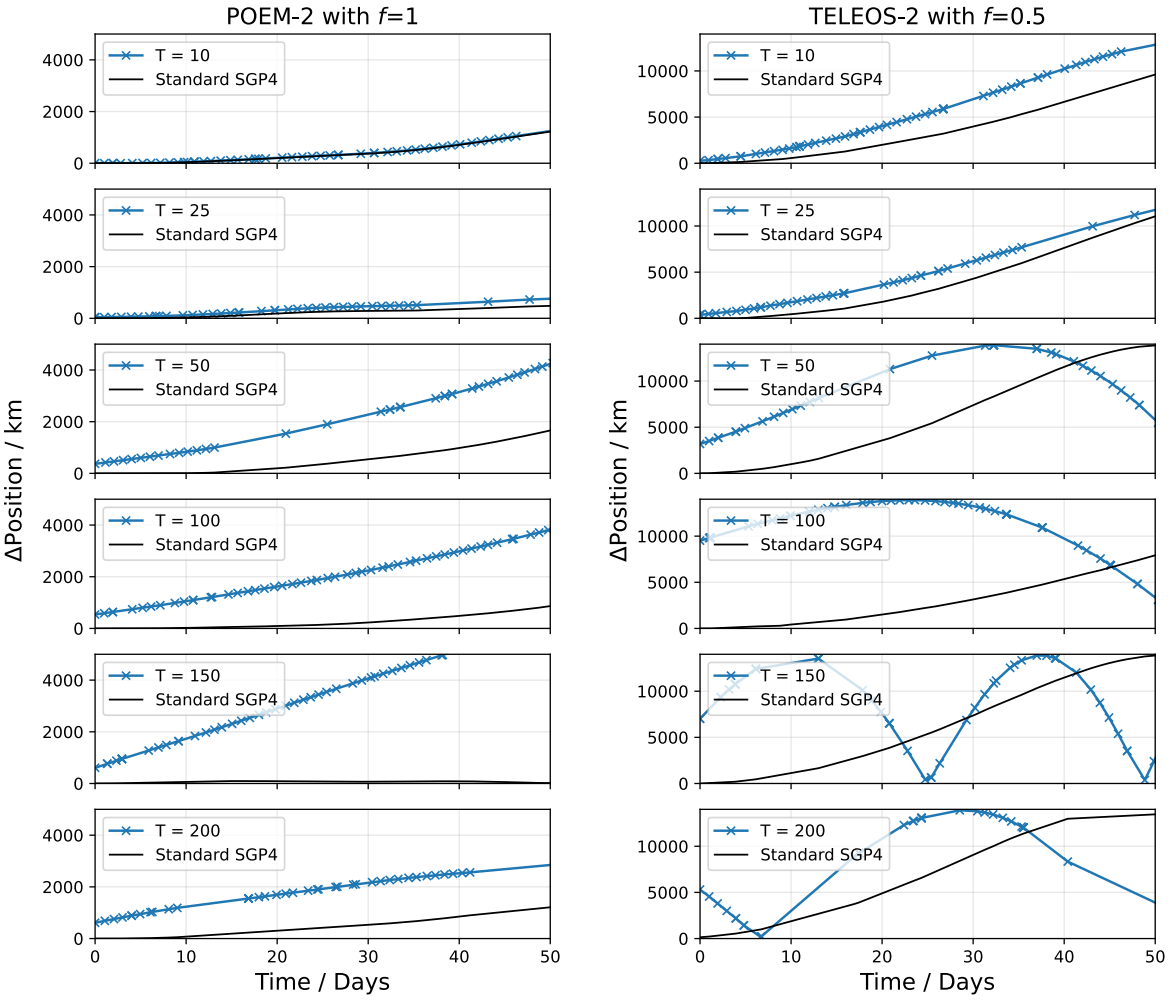

Figure 9: Position error over 50 days for different number of TLEs. The markers represent the data points. Day 0 is defined to be the epoch time of the final TLE in the TLE set, and therefore each graph is in a separate plot. The speed error is not shown as it scales similarly to the position error. Similar to the previous graph, none of the example shown here have a MCS optimised TLE which performs better than SGP4.

in Figure 9, the error due to MCS optimised TLE was always higher than the standard SGP4 prediction. In both scenarios, even if the error went below SGP4's error, it was only after a couple of days and still on the order of thousands of kilometers. If MCS algorithm was being actually used to predict position and velocity, the lower error after a few days will not help as we will not be know when the error becomes low. Overall, MCS algorithm did not optimise the TLE.

4.1.2 Why did MCS perform worse than SGP4?

We hypothesize MCS algorithm's poor performance is due to lack of physics awareness. Consider a sample initial state vector obtained from a reference TLE, `[19.531, 9.8876, 123.3254, 0.0021770, 83.6256, 276.6433,`

14.90420155], and the optimised state vector after using the MCS algorithm, [19.673, 9.8985, 122.4070, 0.0021865, 83.0047, 274.2398, 14.90306961]. The position error from the reference and the optimised TLE refers to the Standard SGP4 and $f = 1.0$ line respectively in Figure 8, POEM-2 with $T = 100$. As observed, a very small change in the state vector elements resulted in drastically different errors (800 km using standard SGP4 compared to 3750 km using the optimised TLE after 50 days).

This demonstrates orbits are very sensitive to initial conditions. In the MCS algorithm, even though we reduce the error between reference and other TLEs in the set, we end up with a completely new orbit with a different B^*. Therefore, the future prediction is inherently made for a completely new orbit. Given that TLEs are unevenly spaced and are released at least a day apart, MCS optimises at random points in the satellite's orbit which further affects the orbit and its propagation. This is further explored in Section 5.

Furthermore, we are independently perturbing the seven elements in the state vector, however not all of them evolve independently from each other. For example consider the following relationships,

$$M = E - e\cos E \tag{19}$$

$$M = n(t - \tau) \tag{20}$$

where M is the mean anomaly, e is eccentricity, E is eccentric anomaly, n is mean motion, t is time and τ is pericenter time. Also since we are using SGP4 which accounts for the Earth's $J2$ perturbation (Hoots and Roehrich, 1988), the rate of change of Right Ascension of the Ascending Node Ω and the Argument of Perigee ω depends on other orbital elements as well,

$$\frac{d\Omega}{dt} \propto n\cos i \tag{21}$$

$$\frac{d\omega}{dt} \propto n\left(5\cos^2 i - 1\right) \tag{22}$$

where i is inclination. As MCS changes these independently, it does not respect the relationships between the parameters, especially when the data points used in the loss functions are spread apart on the order of days. This could further explain the continuous worse performance, because the "optimised" output TLE represents a completely new satellite.

4.2 ML-∂SGP4 Algorithm

4.2.1 Challenges in Training Full ML-∂SGP4

Despite numerous efforts, we were unable to successfully train the full ML-∂SGP4 model on the complete dataset due to the immense amount of time required to train the model. Initially, we hypothesized that the slow training speed was due to insufficient computational resources on our personal laptop. To address this, we gained access to the National Supercomputing Centre (NSCC) Singapore's high-performance computing (HPC) resources.

We successfully submitted the training job to a supercomputing node with specifications detailed in the methods section. However, even with these resources, the forward pass execution speed remained slow. To quantify this, we measured the average data throughput during training by observing the

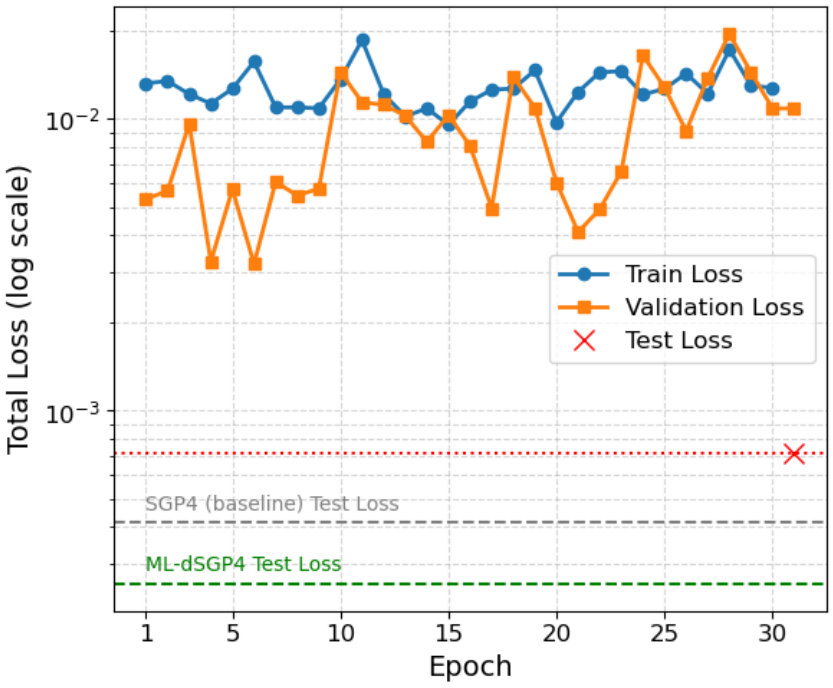

Figure 10: Training and validation loss curves with test loss plot. The training loss curve, shown in blue, exhibits an a largely flat learning curve with substantial noise, suggesting that the model is struggling to learn and optimize from the training set. The validation loss, shown in orange, displays even more erratic fluctuations, with spikes and dips of greater magnitude than those seen in the training loss. Its overall trend is increasing, and it gradually rises toward the higher values seen in the training loss. Note that the validation set is evaluated once more before evaluating on the test set, resulting in 31 points on the validation loss curve—one more than the number of training epochs. The test loss of the current Medium ML-∂SGP4 model is indicated by the red dot, with a value of 0.0007125, which is surprisingly lower than both the final training and validation losses. For comparison, the final test loss values reported in Accirani's paper are also plotted: 0.0004153 for standard SGP4 and 0.000256 for their ML-∂SGP4 model. All loss values (training, validation, and test) of the current Medium ML-∂SGP4 model are higher than the test losses reported in Accirani's work, indicating that the current implementation introduces additional error to the propagated states compared to standalone SGP4.

NSCC's node logs. Despite teh model training for 12 hours, it did not complete even one full epoch. According to NSCC's logs, the model was only able to execute the forward pass on 21% of the training data, corresponding to an average forward pass rate of approximately 1,007 data points per second. Extrapolating linearly, a single epoch would require approximately 2.38 days, implying a total training time of roughly 71.43 days for all 30 epochs.

This result was surprising, given the model's relatively small parameter count (3,454 parameters) and the efficiency of PyTorch's tensor computation engine (NVIDIA, n.d.). Post-training diagnostic logs showed extremely low GPU memory usage (approximately 40MB), accounting for just 0.1% of the total GPU capacity. This suggested that the bottleneck existed in our pipeline. Upon closer inspection of the ∂SGP4 source code, we identified the use of Python-based TLE objects as a potential problem. The ∂SGP4 implementation requires TLEs to be parsed into custom Python classes before propagation. These objects internally perform multiple pre-computations to ensure compatibility with the `propagate_batch()` function which propagates TLEs. However, the conversion from raw TLE text to TLE objects is handled via an explicit

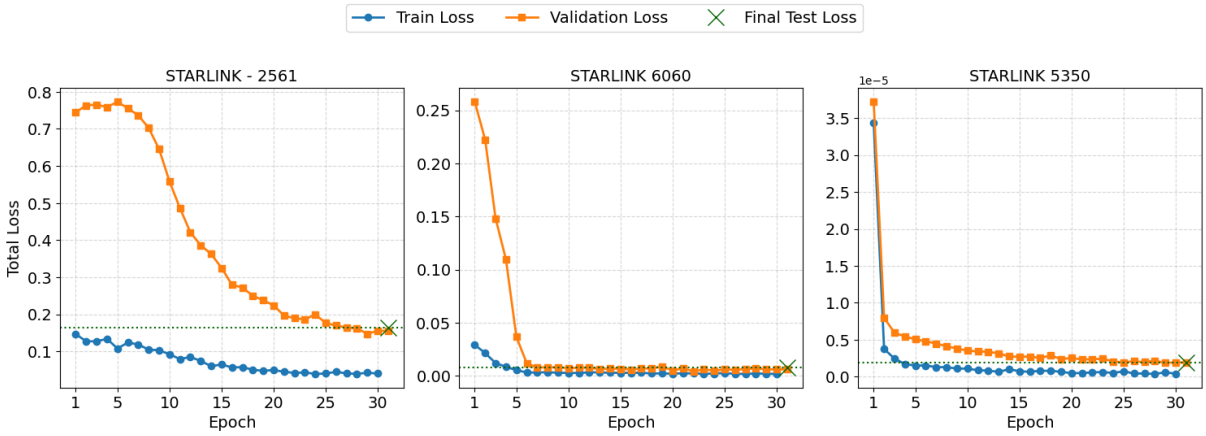

Figure 11: Training and validation loss curves with test loss plot for satellite-specific mini models. For all models, the training and validation losses exhibit an overall decreasing trend, indicating healthy optimization. In particular, the validation losses for the mini models of STARLINK-6060 and STARLINK-5350 closely converge with their training losses, suggesting good generalization. On the other hand, the training loss for the STARLINK-2561 mini model increases slightly during the initial epochs but eventually trends downward. However, its loss curves do not plateau, implying that this model may not have fully converged.

loop, resulting in an $O(n)$ time complexity—where n is the number of TLEs. We attempted to alleviate this bottleneck by pre-converting TLEs into python TLE objects and storing them as serialized `.pkl` files. Unfortunately, this had a negligible effect on the runtime of model training.

Another bottleneck was found within the `propagate_batch()` function of the ∂SGP4 source code. Specifically, this function calls an internal `initialize_tles()` routine, which again uses a loop to parse through each TLE in the batch and populate a tensor with initial conditions. This too, scales linearly with batch size and contributes to the overall inefficiency. Although we considered modifying the ∂SGP4 source code to support vectorized batch operations—allowing propagation in $O(1)$ time, such an overhaul was not feasible given this project's time constraints.

4.2.2 Medium ML-∂SGP4

The Medium ML-∂SGP4 model took approximately 8 hours to train on a compute node from NSCC's supercomputer. The training and validation loss curves across epochs are presented in Figure 10.

The generally flat and erratic trend observed in the training loss curve suggests that the model struggled to learn meaningful representations from the training data. As noted by Barinov et al., 2023, such behavior typically reflects the model's inability to update its parameters effectively to fit the training set. Although the dataset is small by deep learning standards, prior work has shown that even small datasets can enable neural networks to learn effectively while still generalizing well (Olson et al., 2018). Hence, we suspect that the Medium dataset may be under-representing the full problem space. In particular, only satellite period was explicitly controlled for diversity, while other important orbital parameters were not systematically varied. Thus the dataset might lack sufficient vari-

Loss Type	Starlink – 2561 Mini Model	Starlink – 5350 Mini Model	Starlink – 6060 Mini Model	Original ML-dSGP4
Training	0.0406	3.902×10^{-7}	0.00167	1.758×10^{-5}
Validation	0.156	1.915×10^{-6}	0.00597	0.000262
Test	0.1636	1.841×10^{-6}	0.00769	0.000256

Figure 12: Final training, validation and test losses of all mini models. This table contains the final values of the training, validation and test loss for the STARLINK - 2561, STARLINK -5350 and STARLINK - 6060 mini models. The same metrics are included for the original ML-∂SGP4 by Acciarini et al., 2025

ability, limiting the model's ability to learn effective corrections for TLE-derived state vectors and their propagation errors.

The upward trend observed in the validation loss further supports this hypothesis. While the model initially achieved a relatively low validation loss—likely due to favorable random initialization—its performance deteriorated as training progressed. This suggests that the model may have started closer to an optimal configuration and gradually diverged due to ineffective parameter updates and insufficient data diversity.

In summary, the Medium ML-∂SGP4 model failed to extract meaningful patterns from the underrepresented training data and ultimately performed worse than the standalone SGP4 baseline. Rather than mitigating propagation error, it introduced additional inaccuracies. Consequently, this model is excluded from further comparison in the remainder of our study.

4.2.3 Mini ML-∂SGP4

Each of the three Mini ML-∂SGP4 models took approximately 13 minutes to complete training on their respective datasets. The training and validation loss curves, along with the final test losses, are shown in Figure 11. All three models show signs of convergence, indicating successful optimization.

However, the mini model for STARLINK-2561 exhibits a noticeable gap between its training and validation losses, which may suggest overfitting or limited generalization capacity. As evident from the loss plot, both curves are still decreasing but have not plateaued, indicating that further training could potentially enhance its performance. The final training, validation and test loss for all 3 models are shown in 12

To benchmark our models, we first applied the standard ∂SGP4 propagator to the TLEs in each of the three satellite datasets, propagating them to the time stamps corresponding to the ephemeris data. We then computed and plotted the resulting errors, using the Starlink ephemeris data as the ground truth. In parallel, we propagated the same data points using a pretrained version of ML-∂SGP4 provided in Acciarini et al., 2025's source code. Figure 13 displays the error plots from both methods alongside those produced by the satellite-

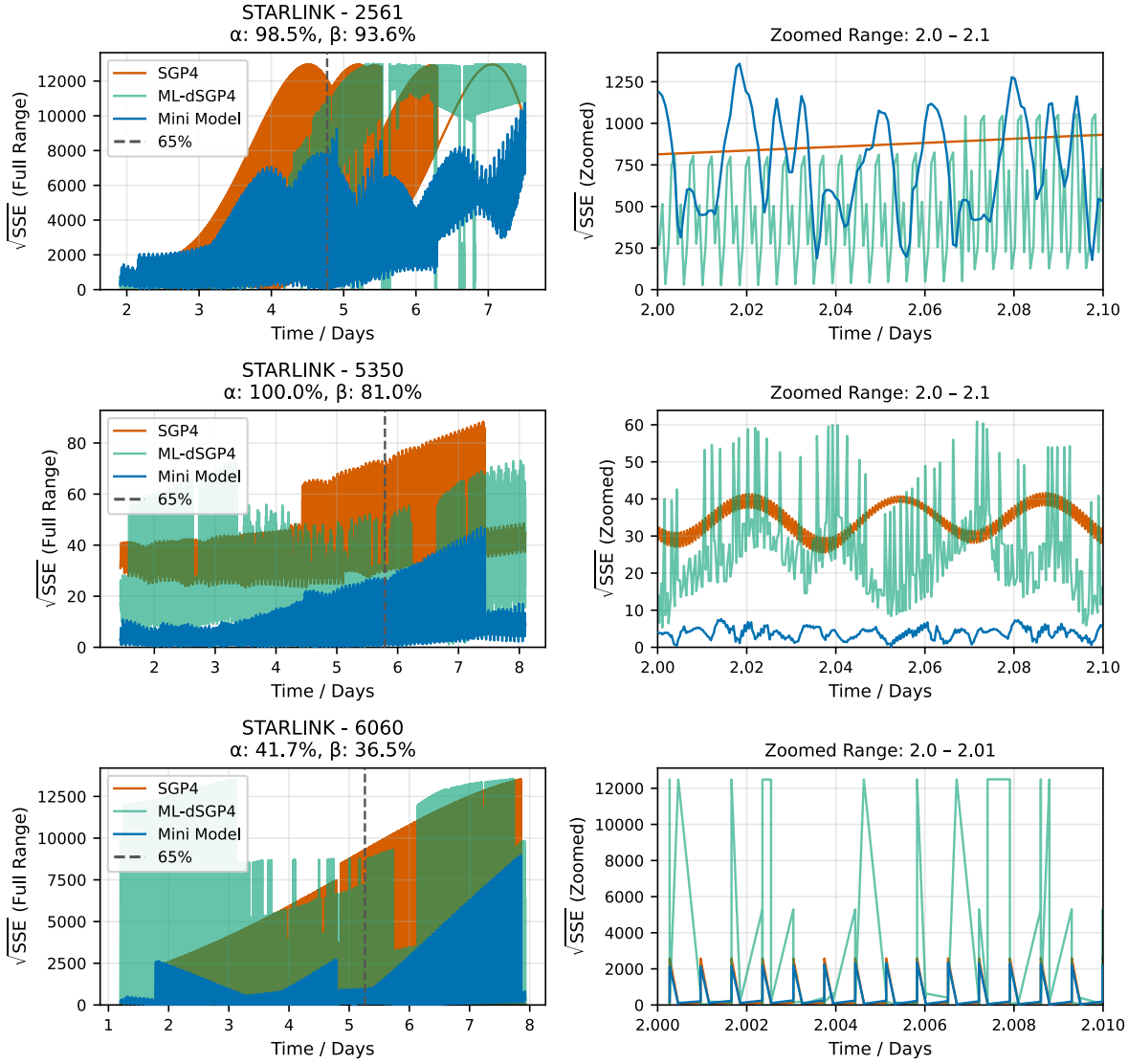

Figure 13: Plot of SSE of all mini models On the left column, the square root of the sum of squared errors between the propagated state from each model and the ground truth is plotted over the full range of propagation days represented in each satellite's mini-dataset. The right column zooms in on a specific interval to better capture the oscillations in error. The "65% time" line indicates the split between the training set (data points prior) and the unseen validation/test set (data points following). The alpha (α) value is computed as the sum of Starlink-propagated data points in the test set where the error is lower than that from SGP4, while beta (β) represents the percentage of data points where the Starlink mini-model error is lower than that from ML-∂SGP4. These metrics allow for a quantitative comparison of model performance.

	SGP4 vs Mini Model		ML-dSGP4 vs Mini Model	
Satellite	Mini Model Better	Mini Model Worse	Mini Model Better	Mini Model Worse
STARLINK - 2561	4200.88	842.65	4810.56	1122.51
STARLINK - 5350	34.32	N/A	16.86	8.26
STARLINK 6060	3803.40	218.39	4495.74	867.87

Figure 14: Comprehensive Evaluation of Mini-Model Error Metrics. The "Mini Model Better" columns report the mean reduction in error relative to the rival models, whereas the "Mini Model Worse" columns capture the average increase in error when the mini-model is less accurate. Notably, for the Starlink-5350 satellite, the mini-model never exceeded the SGP4 error, which is indicated by an N/A in the corresponding metric.

specific Mini-ML-∂SGP4 models.

The α and β metrics reveal that, for most satellites, the mini-models outperform both ML-∂SGP4 and the SGP4 propagator. However, the Starlink-6060 mini-model exhibits notably lower α and β values—both falling below 50%—indicating that for this higher period satellite the rival models deliver more accurate propagated states. For a comprehensive evaluation, we examined both the average magnitude of error improvement when the mini-model outperforms a rival model and its average increase in error when it underperforms, shown in Figure 14

These findings confirm that the mini-model approach not only delivers substantial error reductions when outperforming rival models but also limits the adverse effects when underperforming, thereby providing a tailored and effective solution for satellite state propagation.

In summary, the satellite-specific mini-models offer several advantages: they are fast to train using relatively small datasets, they frequently produce significantly lower errors than both ML-∂SGP4 and SGP4, and even when they underperform, the error discrepancy is minimal. These characteristics make a compelling case for adopting the mini-model approach in satellite propagation correction. Furthermore, with a training time of approximately 13 minutes on standard personal computing resources, these models are not only efficient but also practical for real-world ground station operations. This approach could potentially reduce ground station operators' dependency on the infrequent TLE updaets from NORAD and instead make use of a single TLE to accurately locate their satellite for a longer period of time.

5 Limitations and Future Works

As shown, the MCS algorithm is worse than SGP4 when used with TLEs. This is not to say Liu et al., 2023's model is completely flawed. It is more likely that the semi-analytical propagator that they used decreased errors, but they incorrectly credited it to the MCS algorithm. Also, MCS should still

be analysed by using continuous data. Our dataset had multiple gaps due to infrequent TLE releases. This meant each data point was not equally spaced and the satellite was not located at the same point on its orbit. Furthermore we were perturbing all elements independently. Thus for future testing, we must consider data points that place the satellite in the same position on its orbit day after another.

For example, we use multiple TLEs and then propagate it backwards or forwards until the satellite is at its perigee. We will obtain a dataset where the satellite is in the same point on its orbit on multiple days. After this, we can perturb each element independently and see the effect of the MCS algorithm. We can also put bounds on the MCS algorithms such that they obey the physical interdependence. Using this method, even if the MCS algorithm still performs worse than SGP4, it can be used as a perturbation test to find which orbital element causes the largest error in trajectory prediction.

On the other hand, the main issues with the full ML-∂SGP4 training arise from a bottleneck in the source code that makes training the complete model impractical due to its immense time complexity. This bottleneck severely limits the potential for iterative improvements to the ML-∂SGP4 architecture, as any change — whether it involves adjusting the learning rate, batch size, or exploring additional architectural hyperparameters —necessitates retraining the entire model. For example, increasing the number of layers or neurons to develop a deeper neural network would result in prohibitively long training times under the current implementation. To resolve this, the functions within the ∂SGP4 source code must be vectorized, which could reduce the time complexity for creating and initializing TLE objects to O(1).

Additionally, limitations in the original dataset used for ML-∂SGP4 further restrict improvements. The ephemeris data for STARLINK satellites, hosted on starttrack.org, is only available for a limited number of days. Consequently, the dataset referenced in Acciarini et al., 2025 and used to train the original ML-∂SGP4 is no longer accessible. This prevents us from training a comparable model and conducting ablation studies to pinpoint the primary sources of error reduction, thereby hindering the optimization of specific model components for improved performance.

6 Conclusion

In this study, we analysed two algorithms from the literature for improved position and velocity prediction of satellites, as compared to current method of using TLE with SGP4. The first was Monte-Carlo Simplex algorithm used to optimise the TLE. We generalised the algorithm to run with any satellite using SGP4 as the propagator and also tested new Loss functions. We robustly showed that the MCS algorithm was sensitive to the magnitude of the loss function's output, and it always performed worse than the standard SGP4. We accredit this bad performance to independent perturbations in parameters which do not evolve independently from each other and unevenly spaced dataset.

The second algorithm was ML-∂SGP4. Although we were unable to train a full ML-∂SGP4 model on full-sized or medium-sized datasets, we

devised a novel training technique. By leveraging single-satellite datasets to train the ML-∂SGP4 architectural template, we developed Mini-ML-∂SGP4 models that serve as effective alternatives to the original ML-∂SGP4 algorithm, catered to individual satellites. Specifically, for STARLINK-5350 and STARLINK-6060, the Mini models achieved a reduction in propagation error of at least 81% of the datapoints in their test sets compared to the ML-∂SGP4 model reported in Acciarini et al., 2025. Additionally, both models reduced propagation error for at least 98.5% of the datapoints when compared to the standard SGP4 propagator.

This approach not only increases the frequency of improved predictions but also more effectively reduces error magnitudes compared to ML-∂SGP4, MCS, and the standard SGP4. Moreover, this training paradigm circumvents the time complexity bottleneck in the SGP4 source code—specifically by requiring smaller datasets to match or perform better that the larger ML-∂SGP4.

Training satellite-specific Mini-ML-∂SGP4 models could serve as a viable alternative for small-scale ground station operators, enabling them to rely on the last known TLE for an extended period while still maintaining accurate satellite state estimates. With a more accurate state vector, it becomes feasible to calculate the necessary Doppler shift compensation, ensuring increased signal throughput without incurring the detrimental effects of excessive Doppler shift on transmission. Future work in this area could further improve trajectory prediction, leading to even more precise Doppler shift calculations over longer durations.

7 Acknowledgments

We would like to express our gratitude to STAR's director, A/Prof Abhishek Rai, and Lab Mentors, Ma Yuling, Yuan Fangxing, and Dr Fu Naifeng for their constant support, feedback and patience throughout our journey. We also thank our SPS Mentors, Lim Yi Zhen and Hillson Hung, for constant and even last minute feedback on the reports and presentations, and our SPS friends for the impromptu help in their specialized fields. Lastly, we would like to thank the National Supercomputing Centre (NSCC) Singapore for allowing us to use their resources.

8 References

Acciarini, G., Baydin, A. G., & Izzo, D. (2025). Closing the gap between sgp4 and high-precision propagation via differentiable programming. *Acta Astronautica*, *226*, 694–701. https://doi.org/https://doi.org/10.1016/j.actaastro.2024.10.063

Ali, I., Al-Dhahir, N., & Hershey, J. E. (1998). Doppler characterization for leo satellites. *IEEE Trans. Commun.*, *46*, 309–313. https://api.semanticscholar.org/CorpusID:206409378

Asraf, A., Surayuda, R. H., Ribah, A. Z., Kamirul, & Mukhayadi, M. (2021). Determination of mean orbital elements using gps data for lapan satellite daily operation. *2021 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES)*, 1–6. https://doi.org/10.1109/ICARES53960.2021.9665177

Barinov, R., Gai, V., Kuznetsov, G., & Golubenko, V. (2023). Automatic evaluation of neural network training results. *Computers*, *12*, 26. https://doi.org/10.3390/computers12020026

Einstein, A. (1905). Zur elektrodynamik bewegter körper. *Annalen der Physik*, *322*(10), 891–921. https://doi.org/10.1002/andp.19053221004

European Space Agency, Space Debris Office. (2024). *ESA's Annual Space Environment Report* (tech. rep. No. ESA/GEN-DB-LOG-00288-OPS-SD) (Accessed: 2025-03-27). European Space Agency. Darmstadt, Germany. https://www.sdo.esoc.esa.int/environment_report/Space_Environment_Report_latest.pdf

Hoots, F. R., & Roehrich, R. L. (1988). *Models for propagation of norad element sets* (tech. rep. No. Spacetrack Report No. 3). CelesTrak. https://celestrak.org/NORAD/documentation/spacetrk.pdf

Hussain, W., Merigó, J. M., Gil-Lafuente, J., & Gao, H. (2023). Complex nonlinear neural network prediction with iowa layer. *Soft Computing*, *27*, 4853–4863. https://doi.org/10.1007/s00500-023-07899-2

Jason, Z., Zhang, K., & Grenfell, R. (2004). On the relativistic doppler effects and high accuracy velocity determination using gps.

Kelso, T. S. (2022). Norad two-line element set format [Accessed: 30 March 2025]. https://celestrak.org/NORAD/documentation/tle-fmt.php

Leakyrelu — pytorch 1.11.0 documentation. (n.d.). *pytorch.org*. https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html

Liu, J., Long, W., Wu, Y., Xu, J., Sang, J., & Lei, X. (2023). TLE orbit determination using simplex method. *Geodesy and Geodynamics*, *14*(5), 438–455. https://doi.org/10.1016/j.geog.2023.03.004

National University of Singapore. (2024a). Galassia-2 satellite built by nus students successfully launched [Accessed: 2025-03-31]. https://news.nus.edu.sg/galassia-2-satellite-built-by-nus-students-successfully-launched/

National University of Singapore. (2024b). Successful launch of lumelite-4 to enhance maritime communications [Accessed: 2025-03-31]. https://news.nus.edu.sg/successful-launch-of-lumelite-4-to-enhance-maritime-communications/

NVIDIA. (n.d.). What is pytorch? *NVIDIA Data Science Glossary*. https://www.nvidia.com/en-us/glossary/pytorch/

Olson, M., Wyner, A., & Berk, R. (2018). Modern neural networks generalize on small data sets. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 31). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/ffface8385abbf94b4593a0ed53a0c70f-Paper.pdf

Rojas, R. (1996). The backpropagation algorithm. *Neural Networks*, 149–182. https://doi.org/10.1007/978-3-642-61068-4_7

Space-track.org. (n.d.). *www.space-track.org*. https://www.space-track.org/auth/login

Tanh — pytorch 1.13 documentation. (n.d.). *pytorch.org*. https://pytorch.org/docs/stable/generated/torch.nn.Tanh.html

Yll, D. N. (2018). *Doppler shift compensation strategies for leo satellite communication systems* [Ph.D. Dissertation]. Universitat Politecnica de Catalunya. https://upcommons.upc.edu/bitstream/handle/2117/123510/DanielNietoYll_Doppler_compensation_for_LEO.pdf

9 Supplementary Materials

9.1 Structure of a TLE

A TLE is published by NORAD (Kelso, 2022) and contains orbital elements that can be used to recreate the orbit of a satellite. All the elements are labeled in Figure 15. These orbital elements are said to be correct at the Epoch time of the satellite.

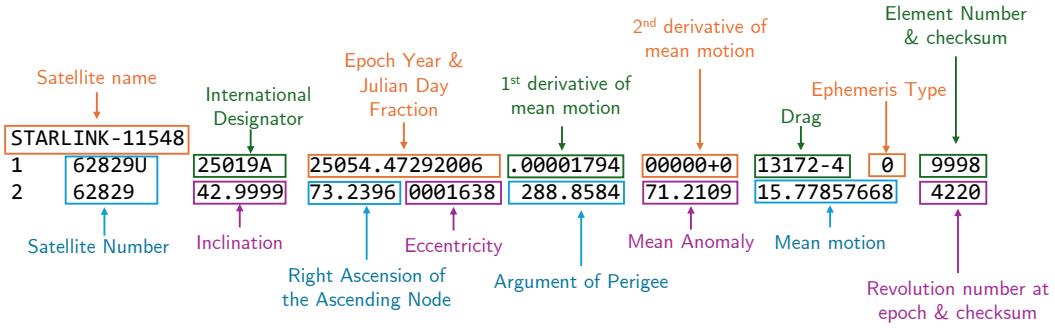

Figure 15: A labeled TLE of STARLINK-11548 valid at 25054.47292006 JD. Drag, inclination, Right Ascension of the Ascending Node, eccentricity, Argument of Perigee, mean anomaly, and mean motion are the seven elements changed in this work.

In this work, we focused on seven elements listed below and the others were left untouched.

1. **Drag**: This is the element that accounts for the friction faced by a satellite through all forces other than spherical Earth's gravitational force.

2. **Inclination**: This is the angle between the celestial equator and the plane of the orbit. It is the first Euler's angle to describe the orbit.

3. **Right Ascension of the Ascending Node (RAAN)**: This is the angle between the vernal equinox and ascending node of the orbit. It is the second Euler angle to describe the orbit.

4. **Argument of Perigee**: This is the angle between the RAAN and the perigee of the orbit, measured in the plane of the orbit. It is the third Euler angle to describe the orbit.

5. **Eccentricity**: This is a measure of how oblate the orbit is. 0 refers to a perfect circle.

6. **Mean Anomaly**: This is the angular distance from the pericenter which a fictitious body would have if it moved in a circular orbit, with constant speed, in the same orbital period as the actual body in its elliptical orbit.

7. **Mean Motion**: This is the number of revolutions the satellite makes in a day.

9.2 Differentiability example

To illustrate why differentiability is central to ML-∂SGP4's physics-aware design, consider the following abstracted computation chain:

$$x \xrightarrow{\text{NN}_1} a \xrightarrow{f} b \xrightarrow{\text{NN}_2} \hat{y} \xrightarrow{\mathcal{L}} \text{Loss}$$

In this setup:

- x is the input data (e.g., TLE features),
- NN_1 is a neural network that optimizes TLE parameters,
- f is a function representing an orbit propagator (either standard SGP4 or differentiable ∂SGP4),
- NN_2 is a neural network that corrects the propagated state vector,
- $\mathcal{L}$ is the loss function comparing $\hat{y}$ with ground truth.

Case 1: When f is the standard SGP4 propagator, which is non-differentiable, the gradient flow through f is blocked:

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial \mathcal{L}}{\partial b} \cdot \underbrace{\frac{\partial b}{\partial a}}_{\text{undefined or } 0} = 0$$

As a result, NN_1 receives no learning signal and cannot be trained via backpropagation.

Case 2: When f is the differentiable ∂SGP4 propagator, gradients can flow through f and reach NN_1:

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial \mathcal{L}}{\partial b} \cdot \frac{\partial b}{\partial a} \quad \Rightarrow \quad \text{NN}_1 \text{ receives gradients and can be trained}$$

9.3 Errors in SGP4 prediction

When the bounds on MCS were not strict enough to give a good prediction, any of the following SGP4 errors disrupted the code:

1. Mean eccentricity is outside the range 0.0 and 1.0.
2. Mean motion is less than 0.
3. Perturbed eccentricity is outside the range 0.0 to 1.0.
4. Semilatus rectum is less than zero.
5. The computed position has a displacement vector with a magnitude smaller than the radius of the Earth.

Hence, the bounds were placed to be ±1% of the initial elements as they prevented any of the above errors.

9.4 Choice of Monte-Carlo Simplex Number

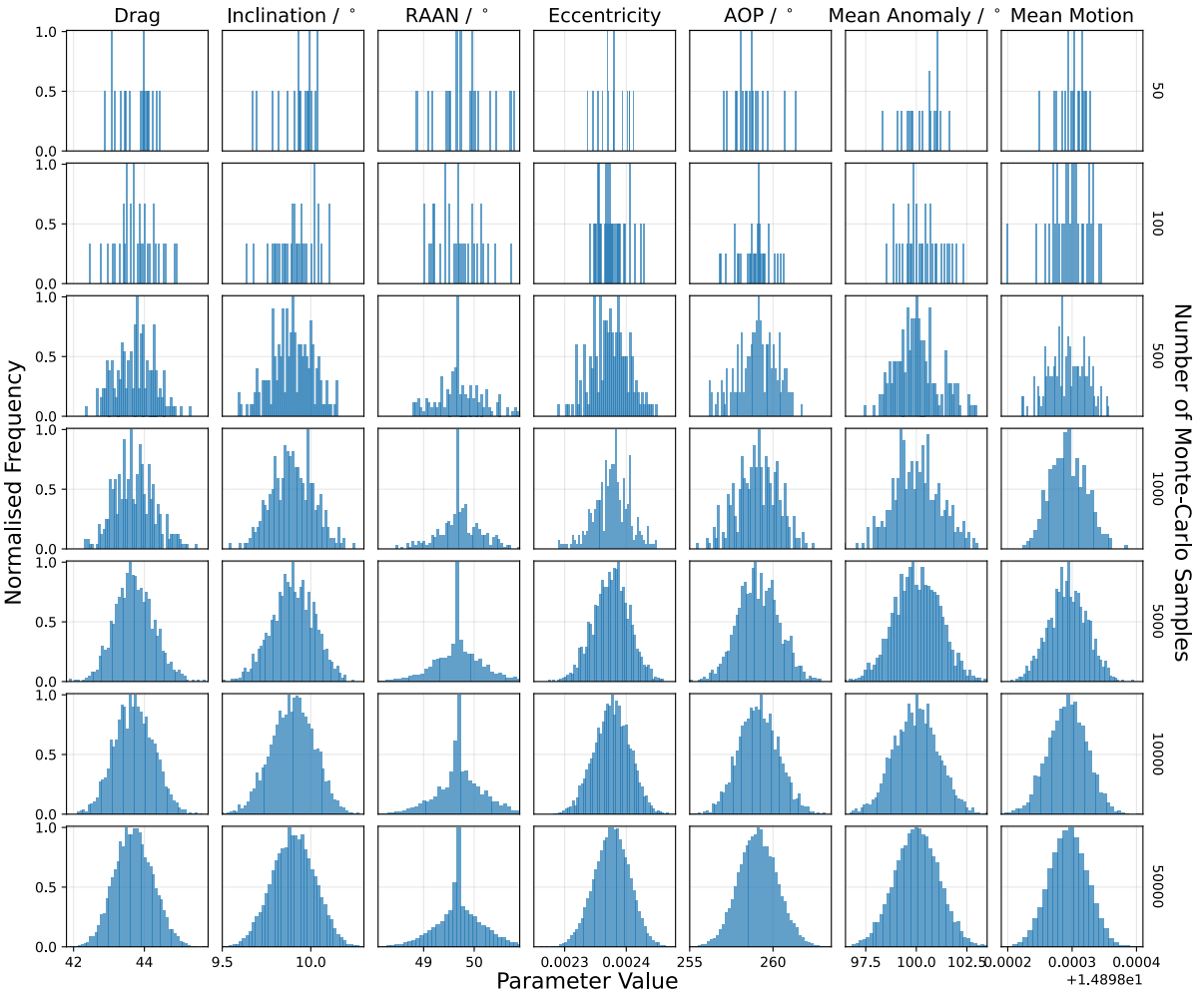

Figure 16: Distribution of optimised values obtained from MCS for different elements at different number of Monte Carlo Simplex. This graph is the extension of Figure 6 and shows the distribution for POEM-2 satellite's parameters. As observed, $N = 5000$ provides a good enough distribution.