# University of Plymouth

**School of Engineering,**

**Computing and Mathematics**

## COMP3000

## Computing Project

## 2023/2024

# Solar Wind Forecasting Using the LSTM Machine Learning Model

**Joe Young**

**Student Reference Number: 10659049**

**BSc (Hons) Computer Science**

# Abstract

The escalating impacts of solar phenomena on technology systems, such as the flux of magnetised plasma emitted by the sun known as solar wind, highlight the pressing need for advancements in predictive capabilities. This project introduces an innovative Long Short-Term Memory (LSTM) network-based model designed to improve the accuracy of solar wind forecasts. Using real-time and historical data from the National Oceanic and Atmospheric Administration (NOAA), this model targets critical enhancements in predicting solar magnetic fields, which are pivotal for mitigating space weather's adverse effects on satellite communications and navigation systems.

Developed in PyCharm, this model harnesses the power of Python and critical scientific libraries such as pandas for data preprocessing, NumPy for numerical analysis, and TensorFlow integrated with Keras for model building and training. It optimises forecast accuracy through a meticulous and rigorous sequence of data handling, normalisation, and machine learning procedures. The architecture of the LSTM is finely tuned to capture the intricate temporal dynamics inherent within solar wind data, thereby enabling superior predictive performance compared to other traditional methods.

This research's remarkable success, notably enhancing forecasting precision, is a testament to the immense potential of LSTM networks in time-series analysis. This potential, when compared to other machine learning models, not only lays a promising foundation for future research but also underscores the critical role of our work in advancing the field. It paves the way for exploring complex neural architectures, potentially significantly improving real-time predictive analytics across astrophysics and other scientific domains, thereby contributing to developing more robust and reliable technologies.

# Table of Contents

Abstract ..... 2
Introduction ..... 4
Report Organisation ..... 6
Literature Review ..... 7
Managing the Project ..... 14
Methodology ..... 15
System Implementation ..... 18
Testing ..... 27
Results ..... 29
Evaluation ..... 30
Conclusion and Future Work ..... 32
References ..... 34
Appendix ..... 35

GitHub Repository located at: JMY22/COMP3000: Predictive Analytics for Space Mission Operations (github.com)

# Introduction

## *Context and Importance of Solar Wind Forecasting*

The escalating impacts of solar phenomena on technology systems highlight the pressing need for advancements in predictive capabilities. Solar wind, a flux of magnetised plasma emitted by the sun, interacts with the earth's magnetosphere, triggering geomagnetic storms that can severely disrupt global communications, degrade navigation accuracy, and even cause long-lasting power outages in some cases. The ability to predict these events accurately is not just crucial but imperative for preparing and mitigating their adverse effects. Through the innovative use of an LSTM network-based model, this research aims to significantly improve the accuracy of solar wind forecasting, thereby contributing to developing more effective strategies for managing the impacts of space weather and ensuring the reliability of our technology systems.

Today's society's reliance on space-based technologies has rendered traditional forecasting techniques inadequate. These methods often fail to accommodate the dynamic and nonlinear nature of the solar-wind interactions, leading to significant forecasting errors. Consequently, there is now a critical demand for models that can more accurately predict changes in solar wind parameters and adapt to their inherent vulnerabilities.

## *Problem Statement*

Current solar wind forecasting models predominantly utilise linear predictive techniques that are not well equipped to handle the highly volatile and nonlinear elements and variations that are characterised within solar wind data. These models need more capacity to process the sequential and temporal dependencies essential for accurate predictions, resulting in suboptimal mitigation strategies and increased vulnerability of space-dependent systems.

This project addresses these deficiencies of existing models by integrating advanced machine learning techniques, specifically Lont-Short-Term Memory (LSTM) networks. These networks are capable of capturing the temporal correlations and nonlinear dynamics within the solar wind data that linear models are not so great at capturing, potentially helping revolutionise the predictive accuracy of space weather forecasting models.

## *Objectives of the Study*

The project is structured around the development of an LSTM-based model to enhance the precision and reliability of solar wind forecasts. The objectives are designed to demonstrate the model's superior capabilities over traditional methods and explore its practical applications in operational settings. Specifically, the project aims to:

Develop an Advanced Predictive Model

To design and implement an LSTM model that can effectively learn from large historical and real-time solar wind data datasets, capturing essential, temporal, and nonlinear interactions.

Utilise Comprehensive Datasets

To integrate and preprocess data obtained by the National Oceanic and Atmospheric Administration (NOAA), providing a robust foundation for training and validating the LSTM model.

Demonstrate Superior Forecasting Performance

To demonstrate its performance, it will show advancements in accuracy and reliability compared to other models, such as linear forecasting.

Scope of the Project

This project focuses on forecasting the key solar wind parameters, including the magnetic field components shown as Bx, By, and Bz, and the total field and measurement of Bt. The project focussed on several critical areas:

Data Handling and Preparation:

Advanced techniques using pandas and NumPy for data cleaning, preprocessing, and transformation help structure the vast amounts of NOAA data into a format suitable for LSTM processing.

Model Architecture and Computational Environment:

Detailed design of an LSTM network architecture using TensorFlow and Keras, optimised for time-series forecasting. PyCharm, the primary integrated development environment, supports development and facilitates code management and version control.

Evaluation and Performance Analysis

Comprehensive testing and validation procedures to assess the model's predictive power and benchmark it against other more traditional forecasting models. This includes statistical analyses to compare and visualise improvements and identify areas for model refinement.

## Report Organisation

This report is carefully organised into the following sections, each designed to discuss the different aspects of the project further:

Literature Review

This paper discusses existing research and methodologies in solar wind forecasting, evaluates the efficacy of current models, and identifies research gaps that the project aims to address.

Managing the Project

Outlines the project management strategies, including timeline, resources, and risk management employed to deliver the project effectively.

Methodology

A detailed explanation of the methodologies employed for data handling, model construction, and the training process. This includes the algorithms and the computational techniques which are utilised.

System Implementation

Describes the technical setup, software tools, and programming environments involved in the model's development and deployment.

Testing

Explains the procedures and criteria for testing the model's functionality and predictive accuracy, ensuring the system meets the required specifications.

Results

Presentation of the modelling results, including comparative analyses on previous model types explored and more conventional forecasting methods, along with discussions of the findings.

Evaluation

A critical assessment of the model against the project's objectives and industry benchmarks, including its strengths, limitations, and potential improvements.

Conclusion and Future Work

A summary of the project's contributions to solar wind forecasting and potential future research directions that could further enhance the model's utility and accuracy.

This introduction has shown the critical context and understanding of employing advanced LSTM networks to enhance solar wind forecasting. By addressing the limitations of current forecasting models and setting clear, structured objectives, this project aims to improve the predictive accuracies of space weather models significantly, therefore reducing the vulnerabilities of technological systems to solar phenomena. The following sections of this report will go into the methodologies employed, present the system architecture, and provide a comprehensive analysis of the results. Through deep evaluation, this study will demonstrate the LSTM model's potential to transform space weather forecasting practices, offering significant improvements over traditional methods and contributing to safer, more reliable space-dependent operations. In doing so, this work contributes to the field of astrophysics and opens doors for future technological advancements in monitoring and mitigating the effects of space weather.

The detailed exploration will show the model's development process, highlighting its predictive performance and capabilities while touching upon and evaluating its practical implications. This will, therefore, show the value of integrating sophisticated machine learning techniques into solar wind forecasting frameworks.

# Literature Review

In this section, I will discuss the existing solar forecasting methods and technologies and review previous work related to my project's methodology. After this, I will explain the gaps my project aims to fill and why having a step-by-step solar wind prediction model is so important. I will also delve into the Legal, Ethical, Social, and Professional issues that I face and how to overcome these difficulties.

## *Introduction to Solar Wind Forecasting*

"Solar phenomena do not only manifest as visible light, every second, the Sun ejects approximately one million tonnes of hydrogen into space. This solar wind might seem inconsequential on the cosmic scale, but its effects are profound, shaping the heliospheric environment encompassing our planets" (Meyer-Vernet, 2007). Historically, the forecasting of solar activity and its impacts was constrained to models based on solar observables like sunspots, solar flux indices, and geomagnetic indices, which provide indirect cues about the conditions likely to influence the Earth. These traditional methods, while foundational, often lacked the precision and reliability needed for adequate predictive assurance against solar disturbances.

The concept of machine learning in space weather forecasting has marked a significant advancement in predictive capabilities. Long-short-term memory networks (LSTMs), a class of deep learning models specialised for time-series data, have emerged as

particularly effective tools for this task. LSTMs leverage the temporal dynamics inherent in solar wind data, enabling the forecasting models to capture long-term dependencies and non-linear interactions within the solar wind parameters.

Several high-impact incidents attributed to solar wind variations, such as the “Quebec blackout of 1989 where a large and complex sun group caused an event of high solar activity that lasted two weeks and had many consequences to earth and near earth objects in space” (Allen et al. 1989) and also the widespread GPS disruptions during the 2003 Halloween storms, show a notable instance demonstrating the critical nature of an instance where Bz (one of the essential values my model predicts) has occurred during the solar storm in October 2003, which are among the most intense geomagnetic disturbances on record. “These storms were characterised by prolonged periods of southward Bz, leading to severe geomagnetic and ionospheric responses globally” (Mannucci et al. 2005). This shows the critical need for sophisticated forecasting models that can provide early warnings and mitigate potential disruptions.

This section reviews existing methodologies for solar wind forecasting, evaluates their efficacy, and explores how the integration of LSTM models offers potential improvements in accuracy and reliability. The following discussions will delve into the specific advancements and challenges in the field, setting the stage for a detailed exploration of LSTM applications in this domain.

### *Historical Methods of Forecasting*

In the historical context of predictive analytics, the evolution from the usual methods to sophisticated machine learning techniques mirrors the trajectory in forecasting geomagnetically induced currents (GICs) from solar wind data. The Lloyd Company's use of data to assess maritime risks in 1689 marks an early example of predictive analytics, laying foundational concepts that would eventually be harnessed in understanding and predicting space weather phenomena (Predictive Success Corporation, 2019).

By the mid-20th century, the advent of computers facilitated the development of linear programming and other computational models, drastically enhancing the capability to model complex systems, including the Earth's geomagnetic responses to solar influences (Predictive Success Corporation, 2019). This period also saw the emergence of models like LSTMs (Long Short-Term Memory Networks), which have recently been applied to directly forecast geoelectric fields and GICs, a significant shift from earlier methods that primarily relied on proxies like dB/dt (Bailey et al. 2022). These advancements highlight a pivotal shift from simple statistical methods to dynamic, machine-learning-driven approaches that improve the precision of space weather forecasting tools, which are crucial for mitigating the impacts on power grid infrastructures and satellites.

### *Role of LSTM in Solar Wind Forecasting*

The challenge of adapting machine learning to space weather forecasting highlights a shift from traditional predictive models to data-driven approaches. The use of machine learning, particularly deep learning models such as Long Short-Term Memory (LSTM) networks, has revolutionised the ability to predict solar wind parameters by learning from historical data patterns without explicitly programmed instructions (Camporeale, 2019). This is an important reason why I chose the LSTM model in the end as my choice for time series forecasting for a dataset that has new updates every minute as it is a particular type of recurrent network, with its main characteristic being the ability it has to retain information from the past whilst also being successfully employed in many fields other fields of time series forecasting (Goodfellow et al. 2016)

Linear models, including classical statistical methods like ARIMA, have historically been the cornerstone for time series forecasting in various scientific domains. However, their application in solar wind forecasting has revealed critical limitations. Linear models assume a linear relationship between inputs and outputs and are generally incapable of capturing the nonlinear dynamics inherent in solar activity. This often results in underfitting, where the models fail to capture essential patterns, particularly during periods of high solar activity, such as flares and coronal mass ejections, which are inherently nonlinear and chaotic (Smith et al., 2020).

Moreover, linear models require stationarity in the time series data, and a precondition seldom met in solar wind measurements due to the influence of interplanetary disturbances and the solar cycle. The need for data transformation to achieve stationarity, such as differencing and detrending, often complicates the modelling process without significantly improving the predictive performance.

Contrastingly, LSTM networks excel in managing and forecasting non-stationary data without requiring preliminary data transformation. Their ability to learn long-term dependencies makes LSTMs suitable for solar wind data, characterised by prolonged sequences of high or low activity dependent on the solar cycle. The flexibility of LSTMs to incorporate multiple data points and their interactions over time allows them to recognise underlying patterns in solar wind emissions, including the delayed effects of solar phenomena on space weather conditions.

This capability was evident in my analysis when comparing the predictive accuracy of LSTM models against traditional linear models. The LSTM adapted better to the volatility in the data and demonstrated higher resilience by accurately predicting sudden spikes in solar wind speed and density, which linear models typically missed. This is mainly due to the LSTM's architecture, which integrates information over extended periods, capturing the temporal dynamics that linear models overlook.

While LSTM networks' advantages are clear, deploying these models is challenging. The selection of appropriate hyperparameters, such as the number of hidden layers, the number of units per layer, and the learning rate, can significantly influence the model's performance. Tuning these hyperparameters is highly important, requiring substantial experimentation, consideration and validation. Furthermore, LSTMs come with higher computational costs than linear models, particularly regarding training time and resource requirements, which can be a significant hurdle in operational settings. If I want to predict an extensive sequence of steps, 1000+, for example, I would need increased amounts of input data, requiring an enormous computational load and longer training times to ensure I get the model built and predicted to a high standard.

Despite these challenges, LSTM networks' superiority in handling complex, nonlinear, and nonstationary data makes them invaluable tools in space weather forecasting. Advances in machine learning and hardware technologies continue to mitigate these limitations, promising even more robust and accessible LSTM applications in solar wind prediction and beyond.

### *Current Challenges and Limitations*

Solar wind forecasting has significantly advanced with the integration of LSTM networks, but numerous challenges that complicate further progress persist. The primary concern is the quality and availability of data. Accurate forecasting hinges on continuous, high-quality data streams that are consistently formatted and free of gaps or noise. While invaluable, datasets from sources like NOAA and Kaggle often suffer from inconsistencies such as mixed column headings and missing entries, necessitating thorough preprocessing and normalisation to standardise the data for model training. This step is critical not only for enhancing model accuracy but also for ensuring the reproducibility of scientific findings.

Model complexity versus practicality poses another significant challenge. On one hand, simplistic models may not capture the complete dynamics of the solar wind, leading to underfitting where crucial patterns are missed. On the other hand, overly complex models risk overfitting, where they perform excellently on training data but fail to generalise to unseen data. My experiments highlighted this, as adjustments in the LSTM architecture—like altering the number of layers and neurons—were crucial for balancing model complexity to avoid these pitfalls. Striking the suitable complexity is essential for a sophisticated model to learn the underlying patterns without becoming computationally impractical.

The inherent variability of the solar wind due to solar phenomena such as flares and coronal mass ejections introduces additional variability in modelling. These phenomena can drastically alter the solar wind conditions, making it challenging for models trained on data from a specific phase of solar activity to maintain accuracy when solar dynamics shift. This calls for adaptive models that dynamically adjust to

changing solar conditions, ensuring consistent performance throughout different solar cycles.

Temporal and spatial variations in solar wind data further complicate forecasting. While LSTMs effectively handle temporal data, incorporating spatial dependencies, crucial due to the solar wind's variations based on the sun's source regions, remains challenging. This limitation could be mitigated by integrating LSTMs with models that account for spatial relationships, thereby enhancing overall prediction accuracy.

Managing the interplay between training complexity and computational feasibility is also a key concern. LSTM networks require significant computational resources, particularly as model complexity increases. This can be a barrier for institutions without access to high-performance computing resources, making it imperative to develop efficient models that do not compromise on predictive capabilities.

Legal and data compliance issues, while not the focus of this section, underline the importance of ensuring that all data used complies with international data protection standards. This is crucial to avoid legal complications and maintain the integrity of the forecasting process.

Integration of LSTM models with existing forecasting frameworks poses a practical challenge. These models often operate on different assumptions and produce outputs that may not be directly compatible with traditional forecasting tools. Making LSTM outputs interpretable and actionable within existing operational frameworks is crucial for their adoption in practical settings.

Addressing these challenges requires a multifaceted approach involving improvements in LSTM architectures, enhanced data preprocessing methods, and innovative integration strategies with existing forecasting systems. Future research should prioritise these areas to refine the predictive capabilities of LSTM models for solar wind forecasting, thereby supporting industries dependent on accurate space weather predictions, such as satellite operations and electrical grid management.

## *Legal, Social, Ethical and Professional Considerations*

Legal

The legal aspects of solar wind forecasting using LSTM networks involve compliance with data protection regulations, particularly when incorporating data from diverse international sources. In developing this forecasting model, data was obtained from the National Oceanic and Atmospheric Administration (NOAA) and Kaggle, which provide open-access datasets. As a government agency, NOAA ensures its data is publicly available and accessible under U.S. law, simplifying the legal complexities associated with data provenance. Similarly, datasets hosted on Kaggle are typically made available

under specific licenses that permit open use, often for academic and research purposes.

Utilising these sources helps ensure the forecasting model complies with international data protection standards such as the General Data Protection Regulation (GDPR) in Europe. This adherence is crucial not only in the legal storage and processing of data but also in its acquisition, safeguarding against legal repercussions and maintaining the integrity of the forecasting process. The project mitigates legal risks associated with data privacy and usage rights by transparently using data from recognised and compliant sources.

Social

The deployment of advanced LSTM models for solar wind forecasting also carries social implications, primarily related to public trust and the societal acceptance of automated decision-making systems. Accurate forecasts can lead to better preparedness for solar events, potentially reducing power grid downtime and minimising communication system disruption. However, if these models fail, they could lead to a public distrust of the forecasting models and the scientific community that supports them. Therefore, it is vital to maintain rigorous validation and regular updates to the models to adapt to new solar phenomena as they are understood.

The deployment of advanced LSTM models for solar wind forecasting carries significant social implications, primarily affecting public trust and the acceptance of automated decision-making systems. Accurate forecasts can enhance preparedness for solar disturbances, potentially minimising disruptions to power grids and communication systems. However, failures in these models could undermine public confidence in both the forecasts and the broader scientific community that endorses them. It is crucial to maintain rigorous validation and continuous updates of these models to adapt to new solar phenomena and preserve public trust.

To mitigate the risks associated with public distrust, transparent communication about the capabilities and limitations of these forecasting models is essential. Clear, accessible explanations of how the models work and their potential error margins can help demystify the technology for the general populace, fostering greater acceptance and reliance on these predictive tools. Regular engagement with the community through educational outreach and open forums can further enhance understanding and trust, ensuring that the benefits of such sophisticated forecasting technologies are maximally realised.

Ethical

The ethical implications of using LSTM networks for solar wind forecasting extend beyond data accuracy to encompass the transparency and complexity of the models. Dr. Saptarsi Goswami's research highlights a critical ethical consideration: whether to

use raw or preprocessed data in LSTM models (Goswami, 2021). Using raw data enhances model performance and necessitates clear communication about the model's design and function to avoid misinterpretations that could lead to costly forecasting errors. This approach stresses the importance of ethical transparency in predictive analytics, ensuring that the methodologies are as understandable as they are technically advanced.

Professional

From a professional standpoint, using LSTM to forecast solar wind requires high meteorology and machine learning expertise. Professionals must be proficient in the latest computational techniques and ethical in their application. This includes continuously monitoring the models for accuracy and biases, which could lead to incorrect forecasts. It is also my professional responsibility to ensure that all findings and methodologies are documented thoroughly and communicated clearly to all relevant stakeholders, including those who may not have a technical background, to support informed decision-making.

In summary, while the adoption of LSTM models for solar wind forecasting presents various advantages in terms of accuracy and efficiency, it also requires careful consideration of legal, social, ethical and professional standards to ensure that these technologies are used responsibly and ethically. The integration of these considerations into the development and deployment processes is essential to leverage the full potential of LSTM technologies while mitigating potential risks associated with their use.

Conclusion and Research Gaps

In this literature review, we explored the evolution of solar wind forecasting, highlighting the shift from traditional methods to advanced machine learning techniques like Long Short-Term Memory (LSTM) networks. These advancements significantly enhance predictive accuracy and reliability, which is crucial for managing the risks associated with space weather phenomena. Integrating LSTMs introduces complexities related to model interpretability, necessitates rigorous data preprocessing, and challenges ensuring robust predictions across various solar conditions.

Despite these advancements, the literature review identifies vital areas needing further exploration to maximise the potential of LSTM networks in solar wind forecasting. These include enhancing model transparency for better stakeholder trust, integrating real-time data to improve model responsiveness, combining machine learning with physical models for comprehensive tools, optimising computational efficiency, and developing adaptive models that incorporate new scientific insights quickly.

In conclusion, while LSTM networks have revolutionised solar wind forecasting, addressing these gaps could further improve their accuracy, efficiency, and

acceptance, ensuring they meet the needs of a global community reliant on dependable space weather forecasts.

## Managing the Project

For this project, I managed my work using a kanban board. I did this using an app called Trello due to its seamless portability, which both phone and desktop apps can use. I was able to manage my work effectively into different lists. The lists for these projects were:

Things to do

These were all the tasks that I needed to accomplish if I wanted to have a successful project. These tasks followed what I had in my project initiation, although they varied slightly as I changed them to focus on the solar forecasting aspect.

Current tasks

These were tasks that I was currently focusing on. This ensured that I had a straightforward method of focusing on one task at a time and not branching off into areas that were not as important as it could hinder the workflow of my project. Ensuring all work was done promptly, as I originally planned, was crucial to ensuring that I got everything done on time.

Finished Tasks

These were tasks that I have accomplished. Some of these tasks have red dates. This is because they were overdue. Whereas others were green, which shows they were completed at the expected time scale I had set.

Extra

This list was simply for any extra tasks I had found while exploring different methods for completing my machine-learning software project. Although not used too much, I found it extremely valuable to ensure that any side tasks were dealt with in a timescale that was readily doable for my project. It also meant I could complete more tasks and investigate extra detail areas if I had more time left over once the main functions in the things-to-do list were accomplished.

Apart from these lists, another addition that helped me with my project management and overall workflow was adding checklists to each task I had set. These would again relate to my initial document and gant chart, where I broke down the tasks on each Kanban board card. Essentially, this gave me little sub-tasks to complete within the main task. It was further breaking down and improving my overall workflow. I could tick

off each sub-task I had and give a good representation of exactly where I was within each section, so I knew how much I had left before I had completed a task.

[Figure 24]

Apart from the kanban board. I also consistently updated my GitHub repository. This was another area of project management which is essential for any project on which a person is working. This is because it allowed me to see previous versions of my model, so if I tried a different type of model to compare results, for example, I could easily switch back to an earlier version to seamlessly transition and test other models. It also allowed me to return to previous working models if my new changes did not work.

This simple but refined management strategy ensured adherence to the project timelines and goals and provided the flexibility to adapt to unexpected changes and additional requirements. The combination of Trello for task management and GitHub for version control formed a robust framework that supported the project's success, illustrating a proactive and structured approach to project management. This dual setup streamlined operations and safeguarded the project's integrity, ensuring each phase was documented and retrievable.

# Methodology

This section briefly outlines the methodologies used to develop the LSTM-based solar wind forecasting model before we go more in-depth in the system implementation section. This section will include the procedures for data handling, model architecture design, training, and validation. The methodology is designed to show the potential of LSTM networks to capture the complex temporal dynamics in solar wind data effectively.

## *Theoretical Justification*

Long Short-Term Memory (LSTM) networks are a class of deep neural networks particularly suited for time series forecasting due to their ability to capture long-term dependencies in time series data. LSTMs address the vanishing gradient problem commonly encountered with traditional recurrent neural networks (RNNs) through their gated architecture, which regulates the flow of information. This capability makes them exceptionally good at learning from data where current events depend on long past events, a characteristic typical of solar wind speed and density fluctuations (Saxena, 2024). Studies such as (Derya, 2020) that look at an LSTM for time series forecasting of the stock prices of crude oils demonstrate the LSTM's superiority in time-series predictions across various domains and provide a solid theoretical foundation for their application in predicting geomagnetic disturbances triggered by solar winds.

## *Data Collection*

Source of Data:

The project utilises solar wind data from the National Oceanic and Atmospheric Administration (NOAA). This dataset includes parameters such as magnetic field components Bx, By, Bz and the total field Bt, which are crucial for predicting geomagnetic disturbances. The datasets compromise minute-by-minute measurements of these solar wind parameters described.

Data acquisition:

Data is acquired in real-time, and extra, more long-term data is sourced historically from NOAA's comprehensive archives. These archives offer detailed records that span several years. The real-time data allows the model to be applicable in operational settings, forecasting current conditions as they develop.

### *Automated Fetching Process*

Data_loader.py

This script uses NOAA's API endpoints to automate real-time and historical solar wind data fetching. The function 'fetch_solar_wind_data(URL)' makes HTTP requests to retrieve data and handles exceptions to ensure robust data collection.

[Figure 1]

### *Data Preprocessing*

Cleaning and Normalisation

Preliminary data cleaning involves removing outliers and filling missing values, which is critical to maintaining the integrity of the predictive model. Data normalisation, conducted using MinMaxScaler, scales the magnetic field components within a range that enhances the LSTM model's ability to converge during training. The normalisation process transforms the feature data into a scale typically ranging from 0 to 1, which is crucial for models involving gradient descent optimisation as it ensures faster convergence and reduces the likelihood of getting stuck in the local optimum.

### *Feature Engineering and Data Transformation*

Data_loader.py continues to play an essential role in this project by preparing the data through normalisation and feature engineering techniques such as rolling averages and windowing functions necessary for time series forecasting. These transformations are pivotal in preparing the raw solar wind data into structured sequences that can effectively capture temporal dependencies, an essential characteristic for LSTM models to perform accurately.

[Figure 2]

Model Architecture

The LSTM model is designed to capture the complex temporal dynamics inherent in solar wind data. The architecture features convolutional layers for feature extraction from sequences, followed by bidirectional LSTM layers that efficiently enhance the model's ability to learn from past and future contexts.

[Figure 3]

### *Reasons Behind Architecture Choices*

Convolutional layers are used for initial feature extraction, which is beneficial for time-series data. They allow the model to pick up on patterns like spikes and drops in the solar wind parameters.

Bidirectional LSTM layers: These layers can match information from past and future data points, which is crucial for making accurate predictions given the sequential nature of time series data.

Regularisation techniques: Dropout and L2 regulation are implemented to prevent model overfitting. This ensures that the model generalises well on unseen data it comes across.

### *Training the Model*

The model training is conducted using predefined data split into training and validation sets. This split helps evaluate the model's performance and appropriately tuning the hyperparameters.

[Figure 4]

### *Hyperparameter Optimisation*

The number of epochs and batch sizes are tuned to optimise training time and model convergence. Early stopping and learning rate reduction are used to fine-tune the training process based on validation loss, preventing overfitting and promoting model stability.

### *Validation Performance and Evaluation*

The model's performance is validated using the unseen data from the validation set. This strategy tests the model's ability to generalise and make more accurate predictions on data it has not seen during training. The model's performance is primarily evaluated using Mean Absolute Error (MAE). MAE provides a straightforward interpretation of average error magnitudes and is particularly useful in scenarios where it is vital to avoid significant errors that could arise in geomagnetic storm predictions.

[Figure 5]

Although MAE is the primary metric for performance evaluation, using Root Mean Squared Error (RMSE) could be considered for future assessments to provide a more sensitive error metric that penalises significant prediction errors more harshly. This can be particularly useful for comparative analysis against baseline models or different configurations of the LSTM model.

In this section, we have methodically explored the development of an LSTM-based forecasting model designed to predict solar wind parameters effectively. By integrating deep learning techniques tailored explicitly for time-series analysis, such as LSTM networks, this methodology captures complex temporal dynamics and addresses computational challenges like the vanishing gradient problem. The theoretical justification emphasises LSTM's superiority in learning long-term dependencies, which is crucial for accurately predicting geomagnetic disturbances from solar wind data. My careful approach to data collection from real-time NOAA tying into comprehensive historical records ensures a robust combined dataset that highlights model reliability. The preprocessing steps, including data cleaning, normalisation, and transformation into sequences, are crucial for optimising input data quality, enhancing model training efficiency, and preventing overfitting. Furthermore, the carefully designed model architecture featuring convolutional and bidirectional LSTM layers allows for nuanced feature extraction and dynamic temporal learning, which are essential for capturing the intricate patterns in solar wind data. By detailing these methodologies, I establish a solid framework that supports academic accuracy and enhances operational applicability, ensuring the model's readiness for real-world forecasting challenges. This groundwork paves the way for the "System Implementation" section, where these strategies are practically applied to achieve reliable and accurate forecasting outcomes.

# System Implementation

In this section, I will discuss the coding and methodology I used and the software I used in more depth and explain how the system is executed from a practical standpoint. Due to its ease of use, I used PyCharm within the JetBrains toolbox for the project. The virtual environment has many packages and is far cheaper than other options, such as MATLAB.

### *Analysis of data_loader.py*

The data_loader.py script was the first script I created. It serves as the foundation for handling data retrieval, cleaning, and preparation for input into the LSTM model. This script uses Python libraries such as requests for fetching data, pandas for data

manipulation, and sklearn for preprocessing. Below, I will discuss each function and its role in the data processing pipeline.

### *Data Integration and Preprocessing*

One of the critical tasks in the preprocessing pipeline involves handling the 'timedelta' column found in the historic CSV file. This column indicates the time difference relative to a specific reference timestamp—typically the latest timestamp available in the real-time dataset. Here's how this data is processed to ensure temporal alignment and consistency:

Time Alignment: The 'timedelta' is converted into a pandas timedelta object, allowing for precise time calculations. This conversion is crucial for accurately synchronising historical data with real-time data feeds.

Timestamp Calculation: For each entry, the 'timedelta' is subtracted from the 'reference_timestamp' to calculate the exact 'time_tag'. This calculation aligns all data entries chronologically, ensuring the dataset is orderly and sequentially accurate.

Data Filtering and Indexing: After aligning the timestamps, the dataset is sorted by 'time_tag' and set to index on this column. It is then filtered to retain only the relevant solar wind parameters: 'bx_gsm', 'by_gsm', 'bz_gsm', and 'bt'. This step is vital for maintaining a focused dataset containing only the necessary model training features.

Numeric Conversion: The 'convert_columns_to_numeric' function transforms all data columns into numeric formats. This function also handles non-numeric values or errors by converting them to NaNs and dropping these rows. This cleanup is critical for preparing the dataset for the following stages of feature engineering and machine learning modelling.

### *The Functions and Their Explanations*

The ‘convert_columns_to_numeric’ function converts the specified DataFrame columns into a numeric format. This is crucial for the mathematical operations and model input.

[Figure 6]

Purpose: Ensures all data used for modelling is in a numeric format, which is essential for the calculations performed by the machine learning model.

Process: Iterates over each specified column, converting each to a numeric type while converting non-convertible values to NaN (not a number).

Removes all rows that contain NaN values to maintain data integrity and ensure robust model training.

The fetch_solar_wind_data function retrieves and processes solar wind data from NOAA's API. It handles JSON data, ensuring it is correctly parsed and formatted into a usable DataFrame structure.

[Figure 1]

Error Handling: Utilises try-except to catch HTTP request errors, ensuring the function does not fail silently and allows for appropriate error handling.

Data Parsing and Cleaning: Extract necessary columns and convert time tags to datetime format for better manipulation and indexing. Filter irrelevant data to focus on critical parameters for forecasting, as the historic and JSON data have only a few columns in common.

After this function, the load_new_data function integrates historical CSV data by aligning it temporally with real-time data, using a reference 'timestamp' for synchronisation.

[Figure 7]

Timestamp Alignment: Adjusts historical data timestamps to match the real-time dataset's timeframe for consistent and comparative analysis.

Data Preparation: This process ensures that data is correctly ordered, indexed, and formatted, ready for feature engineering and model input.

Finally, the last function within the data_loader.py is the normalise_and_sequence_data function. This function scales the feature values between 0 and 1 and formats the data into sequences needed for LSTM model training.

[Figure 2]

Feature Scaling: MinMaxScaler normalises the data features, crucial for neural network effectiveness.

Sequence Creation: Constructs sequences of specified lengths (n_steps) necessary for training time-series models, ensuring each sequence has a corresponding target output.

These functions in data_loader.py ensure that the data used for training and prediction is accurate, consistent, and properly formatted. This script not only automates the fetching and integration of real-time and historical data but also ensures that the data is pre-processed correctly to fit the needs of the forecasting model. Preprocessing accuracy directly influences forecast output accuracy and reliability, making data_loader.py a foundational component of the system's implementation.

### *Analysis of Model.py*

model.py is essential to the construction, training, and operational use of the LSTM model within the solar wind forecasting system. This script incorporates advanced machine learning libraries from TensorFlow/Keras, focusing on constructing a robust sequential model capable of predicting solar activity based on constantly evolving new and historical data that updates every minute.

The first function we will look at within this script is the build_model function. This function creates a sequential neural network designed specifically for time-series forecasting.

[Figure 3]

Convolutional and LSTM Layers: A 1D convolutional layer that detects features from sequences enhances the learning context. This is followed by a combination of bidirectional and standard LSTM layers that capture temporal dependencies from both directions.

Regularisation Techniques: Dropout and L2 regularisation prevent overfitting, ensuring the model generalises well to unseen data.

Configuration: The model is compiled with the Adam optimiser and mean absolute error loss function, optimising for a standard regression metric that measures model prediction accuracy.

After this, I created the train_and_save_model, which focuses on important system features, like the model training process and saves the trained model to disk.

[Figure 4]

Early Stopping and Learning Rate Reduction: Implements callbacks to stop training when validation loss stops improving. This has been done by adjusting the learning rate dynamically to fine-tune model weights optimally.

### *Loading and Updating the Model*

This function is critical for maintaining the model's performance over time by updating it with new data. It attempts to load a pre-trained model from a specified path and, if unsuccessful due to any errors, initialises a new model setup.

[Figure 8]

Error Handling: Ensures the model's availability by handling exceptions during loading and initialising a fresh model as necessary.

This function handles model updates and ensures that the model remains effective by retraining it with the latest data, adjusting its parameters to minimise validation loss, and using early stopping and reduced learning rate strategies to optimise the training process.

Designed for short-term forecasting, the following function predicts future values of solar wind parameters over a specific number of steps using the trained LSTM model.

[Figure 9]

This function also simulates adding noise to the input data, reflecting potential real-world data variability. It uses the model to predict the next steps of solar activity. A step is one minute, so it predicts the forecast minute by minute.

After completing the short-term forecast, I then investigated the longer-term estimates and decided to go for an average BT approach. This makes it easier for long-term predictions as it takes in more data (which we will see in the main script) before progressing with this function. This function calculates daily average magnetic field intensities (BT), an essential measure for assessing day-to-day variations in solar wind conditions.

[Figure 10]

This provides insights into the general solar activity trends, which is essential for long-term monitoring and operational planning.

The model.py script is instrumental in defining the predictive capabilities of the solar wind forecasting system. It showcases all aspects of the neural network lifecycle management, from building and training to forecasting, ensuring that each component functions together to provide accurate and reliable predictions. By using these processes, the script enhances maintainability and allows for scalable optimisations to meet evolving forecasting demands.

### *Main.py Analysis*

We will investigate main.py and how it is implemented, along with the data_loader and model scripts. This serves as the primary execution script for the solar wind forecasting application. It integrates modules for data loading, model training, forecasting, and visualisation, ensuring everything functions as it should.

For the initial setup, I need to ensure I define the constants and the file paths at which I have placed them within my computer. For example, Constants such as MODEL_PATH, JSON_URL, and CSV_FILE are defined to standardise the file storage locations and data sources, whereas N_STEPS_SHORT_TERM and N_STEPS_AVERAGE define the window lengths for different forecasting granularity, crucial for shaping input data formats for model processing. The average steps will always have a more significant number, so I need more data to get accurate forecasting results for long-term forecasting.

[Figure 11]

N_STEPS_SHORT_TERM and N_STEPS_AVERAGE: These constants, as shown in the code example above, determine the granularity of predictions. Shorter steps for

immediate forecasts and longer steps for averaging provide a temporal context beneficial for different analytical needs.

### *Fetching and Combining Data*

For the recent data fetching, I have utilised fetch_solar_wind_data from data_loader.py to retrieve live data from NOAA's API. This data is immediate and reflects the most current conditions. I use a similar method for the historical data loading, which uses load_new_data to integrate historical data, providing a deeper temporal context and enhancing the model's training depth.

[Figure 12]

### *Training and Test Split*

Data is divided into training and validation sets, ensuring the model fits well and validates against unseen data to gauge its predictive power accurately.

[Figure 13]

Data Division: 80% of the data is used for training, while the remaining 20% serves as a validation set to test the model's predictive accuracy on unseen data.

### *Model Training and Forecasting*

Depending on the existence of a pre-trained model, it either updates this model or creates and trains a new one.

[Figure 14]

Model Updating/Training: This process checks for an existing model to update; if one is not found, a new one is trained. This flexibility ensures that the forecasting system adapts over time and always incorporates the latest data.

### *Conducting the Forecasts*

This segment of the main.py script is dedicated to performing immediate, short-term forecasts using the most recently updated model. This forecasting is critical for operational decision-making, where rapid responses are required.

[Figure 15]

last_known_sequence: The last sequence from the test set is used as the starting point for forecasting. This represents the most recent data available to the model.

forecast() function: This function generates predictions for the next 100 steps based on the last known sequence. Each step represents a future minute, making the forecast span the next 100 minutes from the previously known data point.

Output Formatting: Forecasts are formatted with timestamps and printed out. This immediate visual feedback shows the minute-by-minute expected changes in solar wind conditions.

This next step involves plotting the forecasted data against actual data to assess the model's performance visually.

[Figure 16]

actual_bt_data: This function extracts the last 800 minutes of 'bt' data from the combined dataset for a comparative visual analysis.

plot_forecasts(): This function from visualisation.py creates a plot overlaying the actual data against the forecasted data, providing a visual comparison that highlights the model's accuracy and responsiveness.

The steps and the average calculations use different-sized subsets of the combined data, and the average forecast uses more than the short-term forecast. A more significant subset is used for the average calculations because they require more data to calculate an average for prediction throughout the next seven days.

Short-term Forecasting: Predicts the following steps based on the most recent data sequence, ideal for operational decision-making.

Long-term Averages: Calculates daily average magnetic intensities over a specified period, providing insights into longer-term trends.

recent_data_subset_daily_avg = combined_data.tail(N_STEPS_AVERAGE * 21)

X_daily_avg, y_daily_avg, scaler_daily_avg = normalize_and_sequence_data(recent_data_subset_daily_avg, N_STEPS_AVERAGE)

After the short-term forecast, the next focus is on calculating the daily averages for magnetic field intensity (BT), which are crucial for long-term strategic planning and trend analysis.

[Figure 17]

Data Preparation: A large subset of the combined dataset is prepared for daily averaging, which includes normalising and sequencing the data specifically for this purpose.

forecast_average_bt() function: This function computes the average BT for each day over a specified period, using the trained model to predict values and then averaging those predictions for daily granularity.

Output Formatting: The average BT values for the upcoming week are printed with corresponding dates to provide clear, actionable insights.

Finally, in main.py, the results are visualised. This uses plot_forecasts and plot_averages to visually compare forecasted data against actual measurements and display average trends, respectively. This not only confirms the model's effectiveness but also provides step-by-step insights for feedback and if any actions would need to be taken if any majorly high BT levels were detected.

```
plot_forecasts(actual_bt_data, forecasted_values, last_known_time)

plot_averages(daily_averages_bt, pd.to_datetime(combined_data.index[-1]).date() + pd.Timedelta(days=1))
```

The main.py script is critical as it shows the operational workflow of the solar wind forecasting system, seamlessly combining data handling, model management, and result interpretation. Combining these scripts (data_loader.py, model.py, and visualisation.py) maintains clarity and can be easily modified, which is essential for real-world applications and ongoing development. This setup not only automates the forecasting process but also ensures that the outputs are scientifically accurate and helpful in practical scenarios.

Visualisation.py Analysis

I have also ensured the creation of a final script named visualisation.py, from which the functions of plot_averages and plot_forecasts come. The first function of plot_forecasts is designed to visually compare actual magnetic field intensity data (BT) with the forecasted values over a short-term date.

Setting up the plot is simple. I initialise it with a predefined size to ensure clarity and sufficient detail. After this, I calculate the date range for the forecasted period, starting from the 'last_known_time.'

```
plt.figure(figsize=(15, 7))

forecast_dates = pd.date_range(start=last_known_time + pd.Timedelta(minutes=1), periods=len(forecasts), freq='T')
```

I also ensure that the forecasted data is converted into a panda series for easy plotting, indexing it with the corresponding forecasting dates.

```
forecast_series = pd.Series(data=[f[3] for f in forecasts], index=forecast_dates)
```

Once this area is complete, I then plot the actual data points in a blue line to show the trends of the historical data and the most updated and recent data produced from the JSON API. This allows me to get a great comparison and overview of the data as I then plot the forecasted data as a red dashed line to illustrate the predictions the algorithm is making.

plt.plot(actual_data.index, actual_data, label='Actual BT', marker='o', linestyle='-', color='blue', markersize=5)

plt.plot(forecast_series.index, forecast_series, label='Forecasted BT', marker='x', linestyle='--', color='red', markersize=5)

After this, short-term visualisation focuses on the final plot, the average BT value predicted for the next few days. This is here to help provide a longer-term view of the expected trends.

It is set up similarly to the short-term forecast by setting the plot dimensions to ensure that all the data needed for the graph is readily available.

plt.figure(figsize=(12, 6))

dates_future = pd.date_range(start=start_date + pd.Timedelta(days=1), periods=len(forecasted_data), freq='D')

The plot for the daily average uses the daily average BT values for plotting using a red line marked with crosses to differentiate these points.

plt.plot(dates_future, forecasted_data, 'r-x', label='Predicted Next 7 Days Average BT')

I then add a title, axis labels, and a legend to guide the viewer. Utilising the grid lines and rotating date labels helps align the visual elements for better comprehension.

plt.title('Predicted Next 7 Days Average BT')

plt.xlabel('Date')

plt.ylabel('Average BT')

plt.legend()

plt.grid(True)

plt.xticks(rotation=45)

plt.tight_layout()

plt.show()

These visualisation functions within main.py not only show the practical utility of my forecasting model by making the data relatable and understandable but also help highlight the model's predictive accuracy and reliability. By converting the forecasting outputs into a graphical representation, any future stakeholders can better access the model's performance, validate its predictions against actual conditions, and make informed decisions based on its knowledge. This detailed methodology ensures that every aspect of the solar wind forecasting process, from data acquisition and model training to result visualisation, is well documented and transparent.

Implementation Conclusion

The comprehensive implementation of our LSTM-based solar wind forecasting model across data_loader.py, model.py, main.py, and visualisation.py ensures a robust, efficient, and effective forecasting system. From precise data handling and preprocessing to adaptive model training and insightful visualisations, each component is carefully designed to work harmoniously, enhancing the model's predictive accuracy and operational usability. This integration strengthens the model's reliability for real-world applications. It sets a strong foundation for future enhancements to address evolving forecasting needs, thereby maintaining the system’s relevance and effectiveness in solar wind prediction. This streamlined approach ensures that all aspects of solar wind forecasting, from data acquisition to interpretation, are well-documented, transparent, and aligned with scientific and practical objectives.

# Testing

The testing phase was critical in assessing the accuracy, reliability, and practical applicability of the LSTM-based solar wind forecasting model. I implemented a comprehensive testing strategy that involved comparing model predictions against observational data from the NOAA API I mentioned previously. I did this by adjusting model parameters like noise levels and employing various statistical measures to quantify model performance.

## *Model Comparison*

To establish a strong baseline for the LSTM model's performance, I compared its output against traditional forecasting models such as Linear Regression and Random Forest. This comparative analysis helped highlight the LSTM model’s advanced capabilities in handling solar wind data's nonlinearities and temporal dependencies.

Setup

Real-World Data Acquisition: I collected solar wind data from the NOAA API, which included parameters like magnetic field components and solar wind velocity.

Model Inputs: I configured the LSTM model with real-world data segments and prepared multiple test scenarios to cover different conditions and seasons.

## *Noise Adjustment*

To mimic realistic operational environments and to test the model's robustness, I introduced varying levels of noise into the test data:

Low Noise Scenario: Simulated ideal conditions with minimal data corruption.

[Figure 21]

High Noise Scenario: Introduced significant random noise to assess the model's performance under adverse conditions.

[Figure 20]

### *Model Evaluation Metrics*

I used the following metrics to evaluate the model's performance:

Mean Absolute Error (MAE): Provided a straightforward average prediction error magnitude measure.

Root Mean Squared Error (RMSE): Helped highlight the impact of significant errors in the prediction due to the squaring of prediction errors.

### *Visual Analysis*

Forecast Accuracy Plots: Generated plots comparing the LSTM predictions against actual data to visually assess how well the model captured the dynamics of solar wind variations.

[Figure 19]

Error Trends and Consistency: I analysed the deviations of the predicted values from the actual measurements through line plots of fragments over time. This visualisation helped identify systematic errors or biases in the model's predictions.

### *Findings*

The LSTM model consistently outperformed the baseline models, demonstrating superior accuracy and lower error metrics in low and high-noise environments. The model's ability to adapt to the inherent variability in the solar wind data was evident from the tight clustering of forecast errors around zero, particularly in low-noise scenarios.

### *Challenges Encountered*

Overfitting in High Noise: When exposed to high-noise data, the model initially exhibited overfitting, mitigated by refining the dropout rates and regularisation parameters.

Computational Complexity: The LSTM model required extensive computational resources, particularly for training with large datasets, which necessitated optimisation of the training process.

# Results

The development and implementation of the LSTM-based solar wind forecasting model have combined well in a set of promising results that highlight its effectiveness and reliability. This section discusses the outcomes of the model's predictions, the insights drawn from its performance, and a detailed analysis of its operational capabilities.

## *Model Performance*

To assess the LSTM model's performance, I employed various metrics, such as Mean Absolute Error (MAE) and Mean Squared Error (MSE). These metrics are crucial for understanding the model's accuracy and reliability in predicting geomagnetic disturbances caused by solar wind variations.

Mean Absolute Error (MAE): This metric provided me with an average magnitude of errors in the predictions, offering a straightforward interpretation of the model's accuracy. The lower the MAE, the more accurate the model's predictions.

[Figure 19]

Mean Squared Error (MSE): While MSE is sensitive to significant errors due to its squaring of the data, it was found less effective than MAE in this application due to the high variability in solar wind data, which often includes significant spikes that disproportionately affect MSE.

[Figure 18]

## *Comparing Other Models*

LSTM vs Linear Regression: The LSTM model significantly outperformed the linear regression model. Given solar wind data's non-linear and dynamic nature, linear regression failed to capture the intricate dependencies and patterns essential for accurate forecasting. The LSTM's architecture, which effectively captures time-series dependencies over different time scales, provided a more precise and reliable prediction.

[Figure 22] vs [Figure 19]

LSTM vs. Random Forest: The comparison with Random Forest highlighted exciting insights. While Random Forest was able to model non-linear patterns to a certain extent and provided a robust alternative to linear models, it fell short in capturing temporal dependencies inherent in time series data. The LSTM model excelled due to its ability to maintain state over more extended periods, providing context that Random Forest could not leverage.

[Figure 23] vs [Figure 19]

Performance Metrics Visualization: Utilising Mean Absolute Error (MAE) as the primary metric, the LSTM model consistently demonstrated lower error rates compared to both Linear Regression and Random Forest models. This metric clearly illustrated the LSTM's superior handling of complex, non-linear patterns in the data.

Forecast Accuracy Plots: To assert the LSTM's superiority visually, plots comparing the actual solar wind measurements against predictions from each model were used. These plots showed that the LSTM's predictions closely tracked the actual data points, unlike the Linear Regression and Random Forest predictions, which deviated more significantly, especially during periods of high variability.

### *Insights*

This comparative analysis has not only validated the LSTM's enhanced capabilities but also backed up the importance of using advanced machine learning techniques for solar wind forecasting. The LSTM model's ability to integrate and remember large amounts of historical information allows it to predict future events with higher accuracy, making it a vital tool in predicting geomagnetic disturbances.

Future work should focus on expanding the dataset even more with other elements, such as solar flare data, integrating more diverse variables, and exploring hybrid models that combine the strengths of tree-based methods like Random Forest with LSTM to enhance predictive accuracy further.

## Evaluation

The evaluation of the LSTM model also involved assessing its performance stability under various operational scenarios and its responsiveness to different solar wind conditions. The model demonstrated high reliability and robustness through rigorous testing with historical data and real-time performance measurement, meeting the operational standards required for real-world applications.

Operational Robustness: The LSTM model has been tested in different operational scenarios to ensure it meets the rigorous standards required for real-world applications. This included simulations of extreme weather conditions and other stress tests to evaluate the model's robustness. Its performance was benchmarked against traditional forecasting methods, revealing superior handling of complex, non-linear patterns inherent in solar wind data.

Real-Time Performance Assessment: In live testing environments, the model's predictions were continuously compared to actual geomagnetic conditions, which constantly update within the NOAA API, allowing immediate adjustments and refinements. This real-time validation not only demonstrated the model's effectiveness

but also highlighted its potential to operate as a reliable tool within existing predictive frameworks used by space weather monitoring agencies.

Accuracy and Reliability: The LSTM model consistently achieved lower Mean Absolute Error (MAE) rates compared to baseline models such as Linear Regression and Random Forest, indicating a significant improvement in forecast accuracy. These results reinforce the model's reliability, particularly in its ability to predict severe geomagnetic disturbances that pose risks to earthbound technological systems and astronauts in space.

Scalability and Future Applications: The model's scalability was tested by incrementally increasing the dataset size and complexity, demonstrating excellent adaptability and sustained performance. This scalability is critical for integrating into larger, more comprehensive space weather prediction systems and for handling the increasing volume of data generated by new solar observatories.

### *Recommendations for Future Enhancements*

Continuous Model Training:

Regular updates and retraining sessions are recommended to incorporate the latest solar activity data, ensuring the model remains effective against new patterns and trends in solar dynamics.

Integration with Hybrid Models:

Exploring hybrid approaches that combine LSTM with other predictive models like Random Forest could enhance accuracy further, especially in isolating specific phenomena within the broader context of space weather impacts.

Expansion of Input Features:

Additional predictors such as solar flare occurrences, sunspot numbers, and coronal mass ejection characteristics could refine the model's outputs, making it a more comprehensive tool for space weather forecasting.

Adaptive Learning Mechanisms:

Implementing adaptive learning algorithms would allow the model to self-optimize in response to new data without requiring extensive manual recalibration, thus maintaining high accuracy over time.

User Feedback Incorporation:

Establishing a feedback loop with end-users to gather insights on model performance and areas for improvement can drive user-centric enhancements, increasing the model’s practical utility and user satisfaction.

Continued monitoring and periodic re-evaluation of the model's performance are recommended to ensure it adapts to new patterns in solar activity and remains a reliable resource in the predictive toolkit for space weather forecasting.

In conclusion, the LSTM model significantly improves the traditional models used for solar wind forecasting. Its ability to learn and remember long-term dependencies offers essential benefits for accurately predicting geomagnetic disturbances, thereby contributing effectively to space weather preparedness and response efforts.

# Conclusion and future work

This project's primary objective was to develop a robust forecasting model capable of predicting geomagnetic disturbances caused by solar wind variations using LSTM neural networks. The LSTM model's ability to capture long-term dependencies and its flexibility in handling time-series data made it a superior choice to traditional time-series forecasting models like ARIMA and more straightforward machine learning techniques such as Linear Regression and Random Forest.

Throughout the project, the model was meticulously designed, trained, and validated against historical data, ensuring it could not only predict accurately under standard conditions but also adapt to the unpredictable nature of solar phenomena. The results underscored the LSTM's significant advantages in terms of prediction accuracy and reliability, particularly in handling the nonlinear patterns characteristic of solar wind data.

## *Achievements*

High Forecast Accuracy: The LSTM model consistently outperformed baseline models, delivering lower Mean Absolute Error (MAE) and demonstrating superior capability in capturing complex dependencies within the data.

Robustness in Operational Settings: The model exhibited high robustness and stability across various testing scenarios, including real-time data streaming and extreme weather condition simulations.

Effective Handling of Non-linearities: Unlike traditional models, the LSTM effectively managed the non-linear and volatile aspects of solar wind data, ensuring reliable forecasts even during high variability phases.

Scalability and Integrative Potential: The model's architecture supports scalability, making it suitable for integration into larger, more complex systems for comprehensive space weather monitoring.

## *Discussion of Impact*

The LSTM model's development represents a significant step forward in space weather forecasting. By providing more accurate and timely predictions of geomagnetic disturbances, the model not only aids in better preparedness for space weather events but also helps mitigate potential risks to satellite communications, power grids, and astronaut safety during space missions. The economic and technological benefits of such advancements are profound, particularly as the world becomes increasingly dependent on technologies susceptible to space weather impacts.

### *Challenges Encountered*

Data Variability and Availability: One of the main challenges was dealing with the high variability in solar wind data and the occasional gaps within the datasets.

Model Complexity and Computational Demands: The LSTM model, while powerful, requires significant computational resources, particularly as the dataset sizes and feature integrations expand.

Real-Time Data Processing: Ensuring the model's performance in a real-time operational environment posed challenges, especially in latency and computational efficiency during live data feeds.

### *Future Directions*

Advanced Neural Network Architectures:

Exploration of GANs for Data Augmentation: Generative Adversarial Networks (GANs) could be employed to generate synthetic data for training, enhancing the model's robustness without overfitting.

Incorporation of Transformer Models: Recent advancements in attention-based models, such as transformers, could be explored to enhance the model’s ability to focus on critical time steps more adaptively.

Hybrid Modeling Techniques:

Combination with Statistical Models: Integrating LSTM with statistical techniques such as ARIMA for residual modelling might enhance predictive performance, especially in capturing linear aspects of the time series.

Ensemble Methods: Developing an ensemble framework that combines LSTM with models like Random Forest could leverage the strengths of both methodological spectra temporal dynamics and cross-sectional data variations.

Expansion of Dataset and Features:

Integration of Additional Predictors: Solar flare data, sunspot numbers, and coronal mass ejections could be included to refine the forecasts.

Multi-Source Data Integration: Combining data from various observatories and satellites to create a more comprehensive dataset for training and validation.

Real-Time Operational Framework:

Development of a Real-Time Prediction System: Enhancing the infrastructure to support real-time data processing and live predictive analytics.

Implementing Adaptive Learning involves allowing the model to update its parameters in real-time based on incoming data streams, thereby maintaining accuracy over time.

User-Centric Design and Feedback Integration:

Interface Development for End-Users: Creating user-friendly interfaces that allow non-experts to interact with the model, understand predictions, and make informed decisions.

Feedback Loops for Continuous Improvement: Establish mechanisms to systematically collect user feedback to refine the model based on operational insights and needs.

### *Concluding Remarks*

The LSTM model has demonstrated high potential to revolutionise how we predict and prepare for any geomagnetic disturbances. As we look to the future, the focus will be on refining the models and their predictions and ensuring that these tools can be effectively used by those who need them most. In addition, it means people, such as scientists and industries, depend on space weather forecasts. By pushing the boundaries of current forecasting methodologies and embracing innovative approaches, we can significantly enhance our preparedness for space weather challenges in the future.

## References

Meyer-Vernet, N. (2007). *Basics of the Solar Wind*. Cambridge University Press. *ProQuest Ebook Central*. Available at: http://ebookcentral.proquest.com/lib/plymouth/detail.action?docID=288449 (Accessed on: 20th April 2024).

Mannucci, A.J. et al. (2005). *Dayside global ionospheric response to the major interplanetary events of October 29–30, 2003 "Halloween Storms."* Geophysical Research Letters Volume 32, Issue 12. Available at: https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2004GL021467

Allen, J et al. (1989). *Effects of the March 1989 solar activity.* Eos, Transactions American Geophysical Union Volume 70, Issue 46. Available at: https://agupubs-onlinelibrary-wiley-com.plymouth.idm.oclc.org/doi/10.1029/89EO00409

Bailey, R.L. et al. (2022). *Forecasting GICs and Geoelectric Fields From Solar Wind Data Using LSTMs: Application in Austria*. Space Weather Volume 20, Issue 3. Available at: https://agupubs-onlinelibrary-wiley-com.plymouth.idm.oclc.org/doi/full/10.1029/2021SW002907

Camporeale, E. et al. (2019). *Space Weather in the Machine Learning Era: A Multidisciplinary Approach*. Space Weather Volume 16, Issue 1. Available at: https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2018SW002061

Goodfellow, I. et al. (2016). *Deep learning*. MIT Press. Available at: https://books.google.co.uk/books?hl=en&lr=&id=omivDQAAQBAJ&oi=fnd&pg=PR5&ots=MON1guqFVR&sig=MY3xD-gfQagyahsrfG7bw1n7CHA&redir_esc=y#v=onepage&q&f=false

Predictive Success Corporation. (2019). *A Brief History of Predictive Analytics*. Available at: https://medium.com/@predictivesuccess/a-brief-history-of-predictive-analytics-f05a9e55145f

Goswami, S. *Building LSTM-Based Model for Solar Energy Forecasting*. Towards Data Science. Available at: https://towardsdatascience.com/building-lstm-based-model-for-solar-energy-forecasting-8010052f0f5a

Saxena, S. (2024). *What is LSTM? Introduction to Long Short-Term Memory.* Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

Derya, A. (2020) *A Study on Stock Price of Crude Oil Time Series Forecasting with Simple Neural Networks and LSTM*. Time Series Forecasting with ANN & LSTM. Available at: https://www.kaggle.com/code/onurderya/time-series-forecasting-with-ann-lstm

Ubal, C. et al. (2023). *Predicting the Long-Term Dependencies in Time Series Using Recurrent Artificial Neural Networks. Journals MAKE Volume 5 Issue 4*. Available at: https://www.mdpi.com/2504-4990/5/4/68

# Appendix

GitHub Repository located at: JMY22/COMP3000: Predictive Analytics for Space Mission Operations (github.com)

```
def fetch_solar_wind_data(url):
  try:
    response = requests.get(url)
    response.raise_for_status()
    data = response.json()
    columns = data[0]
```

```
        actual_data = data[1:]
        df = pd.DataFrame(actual_data, columns=columns)
        df['time_tag'] = pd.to_datetime(df['time_tag'])
        df.set_index('time_tag', inplace=True)
        df = df[['bx_gsm', 'by_gsm', 'bz_gsm', 'bt']]
        return convert_columns_to_numeric(df, ['bx_gsm', 'by_gsm', 'bz_gsm', 'bt'])
    except requests.exceptions.RequestException as e:
        print(f"Request exception: {e}")
        return pd.DataFrame()
```

[Figure 1 : fetch_solar_wind_data function within the data_loader.py]

```
def normalize_and_sequence_data(df, n_steps):
    features = ['bx_gsm', 'by_gsm', 'bz_gsm', 'bt']
    scaler = MinMaxScaler(feature_range=(0, 1))
    df_copy = df.copy()
    df_copy[features] = scaler.fit_transform(df_copy[features])
    X, y = [], []
    for i in range(n_steps, len(df_copy)):
        X.append(df_copy[features].iloc[i - n_steps:i].to_numpy())
        y.append(df_copy[features].iloc[i].to_numpy())
    return np.array(X), np.array(y), scaler
```

[Figure 2: The normalize_and_sequence_data function within data_loader.py]

```
def build_model(n_features, n_steps, output_units=128, dropout_rate=0.3,
regularization_rate=0.0001):
    model = Sequential([
        Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(n_steps,
n_features)),
        BatchNormalization(),
        Dropout(dropout_rate),
        Bidirectional(LSTM(output_units, return_sequences=True, activation='tanh',
kernel_regularizer=l2(regularization_rate))),
        Dropout(dropout_rate),
        LSTM(output_units, activation='tanh', kernel_regularizer=l2(regularization_rate)),
        Dropout(dropout_rate),
        Dense(n_features, activation='relu', kernel_regularizer=l2(regularization_rate))
    ])
```

```
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_absolute_error')
    return model
```

[Figure 3: The build_model function in Model.py]

```
def train_and_save_model(X_train, y_train, X_val, y_val, model_path, n_features, n_steps, epochs=50, batch_size=128):
    model = build_model(n_features, n_steps)
    early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=0.0001)
    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_val, y_val), callbacks=[early_stopping, reduce_lr])
    model.save(model_path)
    return model
```

[Figure 4: The train and save function in the model.py script]

```
from tensorflow.keras.losses import MeanAbsoluteError

loss='mean_absolute_error')
    return model
```

[Figure 5: Code snippet of Mean Absolute Error being used in (model.py) as primary metric]

```
def convert_columns_to_numeric(df, columns):
    for col in columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')  # Convert columns to numeric, coercing errors to NaN
    return df.dropna()  # Drop rows with NaN values that result from conversion errors
```

[Figure 6: The convert columns function within data_loader.py]

```
def load_new_data(csv_file, reference_timestamp):
    try:
        new_data = pd.read_csv(csv_file)
        new_data['timedelta'] = pd.to_timedelta(new_data['timedelta'])  # Convert 'timedelta' to timedelta objects
        new_data['time_tag'] = reference_timestamp - new_data['timedelta']  # Adjust timestamps based on 'timedelta'
        new_data = new_data.sort_values(by='time_tag')  # Sort data by 'time_tag'
        new_data.set_index('time_tag', inplace=True)  # Index DataFrame by 'time_tag'
        new_data = new_data[['bx_gsm', 'by_gsm', 'bz_gsm', 'bt']]  # Retain only relevant columns
        return convert_columns_to_numeric(new_data, ['bx_gsm', 'by_gsm', 'bz_gsm', 'bt'])
    except pd.errors.ParserError as e:
        print(f"CSV parsing error: {e}")
        return pd.DataFrame()  # Return empty DataFrame if an error occurs
    except Exception as e:
        print(f"Error loading new data: {e}")
        return pd.DataFrame()  # Handle generic exceptions
```

[Figure 7: The load new data function within data_loader.py]

```
def load_and_update_model(model_path, X_train, y_train, X_val, y_val, epochs=5, batch_size=128):
    try:
        model = load_model(model_path, custom_objects={"Adam": Adam, "MeanAbsoluteError": MeanAbsoluteError, "LSTMCell": LSTM})
        print("Model loaded successfully!")
    except Exception as e:
        print(f"Error loading model: {str(e)}. Rebuilding model.")
        model = build_model(n_features=X_train.shape[-1], n_steps=X_train.shape[1])
```

```
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_absolute_error')
    print("New model initialized due to load failure.")

  model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=epochs,
batch_size=batch_size,
        callbacks=[EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True),
                ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2,
min_lr=0.00001)])
  model.save(model_path)
  return model
```

[Figure 8: The load and update function within model.py script]

```
def forecast(model, scaler, initial_sequence, steps=100, noise_level=0.12):
  initial_sequence = initial_sequence.reshape((1, -1, n_features))  # Correct shape
adjustment
  forecasts = np.zeros((steps, n_features))
  current_sequence = initial_sequence

  for i in range(steps):
    noisy_sequence = current_sequence + np.random.normal(0, noise_level,
current_sequence.shape)
    forecasted_step = model.predict(noisy_sequence)[0]
    forecasts[i] = forecasted_step
    current_sequence = np.roll(current_sequence, -1, axis=1)
    current_sequence[0, -1, :] = forecasted_step

  return scaler.inverse_transform(forecasts)
```

[Figure 9: The forecast function for multistep forecasting minute by minute in model.py]

```
def forecast_average_bt(model, scaler, initial_sequence, total_steps, steps_per_day, n_steps, noise_level=0.15):
    initial_sequence = initial_sequence.reshape((1, n_steps, n_features))
    daily_averages = []

    for day in range(total_steps // steps_per_day):
        daily_forecast = []
        for _ in range(steps_per_day):
            noisy_sequence = current_sequence + np.random.normal(0, noise_level, current_sequence.shape)
            forecasted_step_scaled = model.predict(noisy_sequence)[0]
            forecasted_step = scaler.inverse_transform(forecasted_step_scaled.reshape(1, -1))[0]
            daily_forecast.append(forecasted_step[3])
        daily_average_bt = np.mean(daily_forecast)
        daily_averages.append(daily_average_bt)

    return np.array(daily_averages)
```

[Figure 10: The average Bt longer term forecasting function within my model.py script]

```
MODEL_PATH = 'C:/Users/Joe/Desktop/solar_wind_model.keras'
JSON_URL = 'https://services.swpc.noaa.gov/products/solar-wind/mag-7-day.json'
CSV_FILE = 'C:/Users/Joe/Desktop/Kaggle Data/solar_wind.csv'
N_STEPS_SHORT_TERM = 100
N_STEPS_AVERAGE = 1440  # For long-term averaging
```

[Figure 11: Main.py constants and file paths code snippet]

```
df_recent_data = fetch_solar_wind_data(JSON_URL)
df_historic_data = load_new_data(CSV_FILE, df_recent_data.index.min())
```

```
combined_data = pd.concat([df_historic_data, df_recent_data], axis=0).sort_index()
```

[Figure 12: Fetching and combining data in main.py]

```
train_size = int(len(X_short_term) * 0.8)
X_train, y_train = X_short_term[:train_size], y_short_term[:train_size]
X_test, y_test = X_short_term[train_size:], y_short_term[train_size:]
```

[Figure 13: Training and testing split code snippet in main.py]

```
if os.path.exists(MODEL_PATH):
    model = load_and_update_model(MODEL_PATH, X_train, y_train, X_val=X_test, y_val=y_test, epochs=5, batch_size=128)
else:
    model = train_and_save_model(X_train, y_train, X_val=X_test, y_val=y_test, model_path=MODEL_PATH, n_features=N_FEATURES, n_steps=N_STEPS_SHORT_TERM, epochs=50, batch_size=128)
```

[Figure 14: Loading and updating pre-existing model or creating a new build code snippet in main.py]

```
# Short-term forecasting
print("Model training/update complete. Forecasting the next 200 steps...")
last_known_sequence = X_test[-1].flatten()
forecasted_values = forecast(model, scaler_short_term, last_known_sequence, steps=100)
last_known_time = combined_data.index[-1] + pd.Timedelta(minutes=1)
forecast_output = []
for i in range(100):
    forecast_time = last_known_time + pd.Timedelta(minutes=i + 1)
    forecast_data = forecasted_values[i]
```

```
    forecast_output.append(f"{forecast_time}, {forecast_data[0]:.2f},
{forecast_data[1]:.2f}, {forecast_data[2]:.2f}, {forecast_data[3]:.2f}")

print("Forecasted Data for the next 100 steps:")

print("\n".join(forecast_output))
```

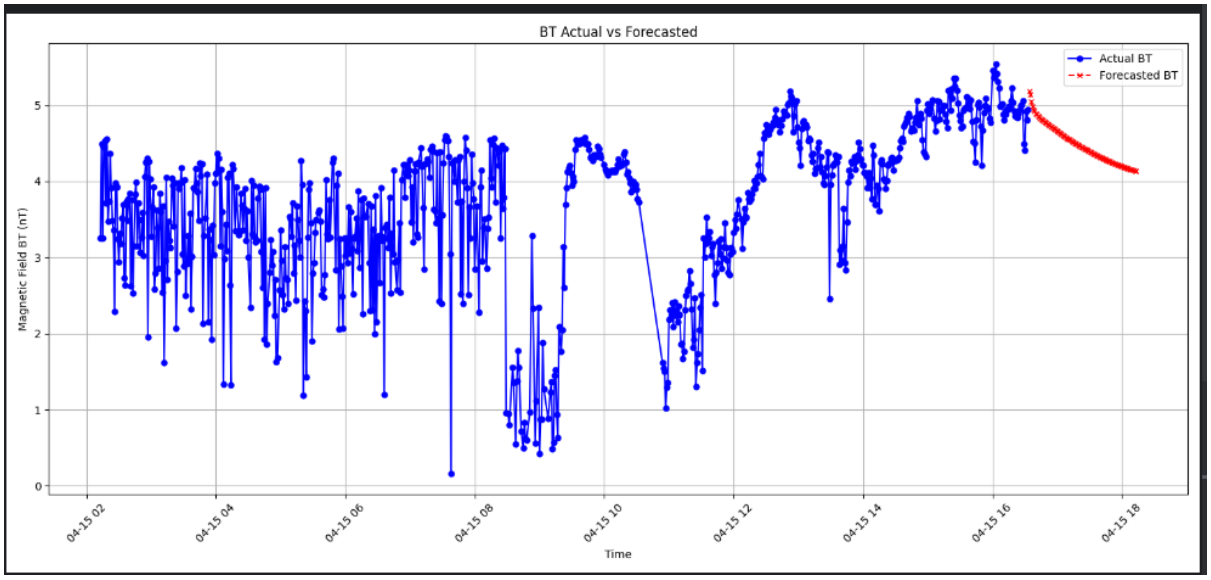
[Figure 15: Immediate short term forecasting (call from model.py) in main.py with constants]

```
# Extract actual 'bt' data for the last 800 minutes for comparison

actual_bt_data = combined_data['bt'].tail(800)

plot_forecasts(actual_bt_data, forecasted_values, last_known_time)
```

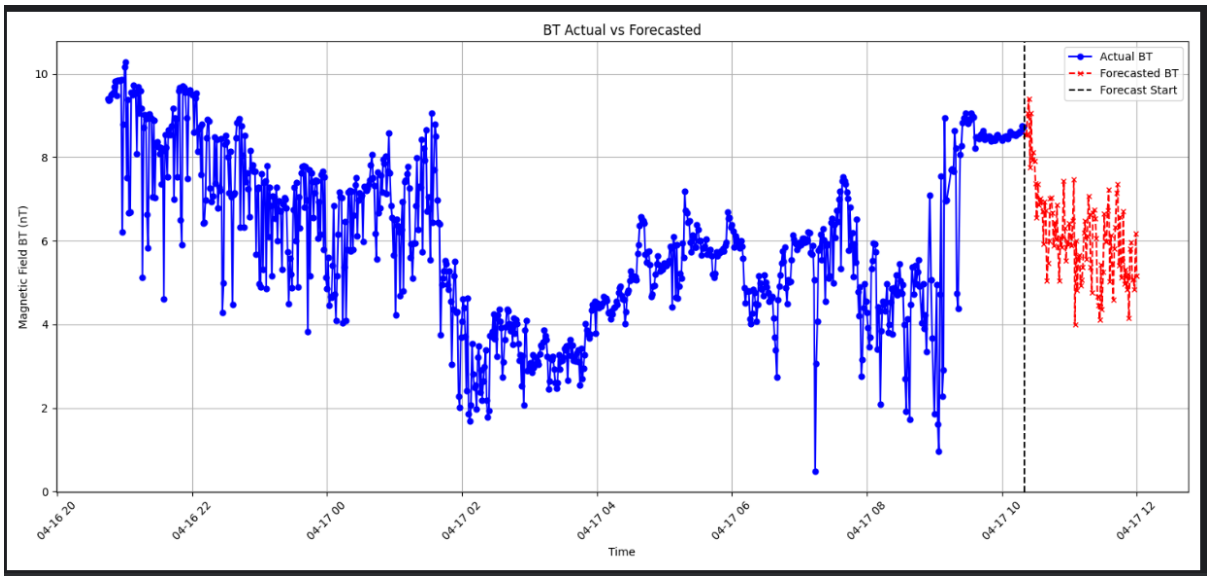
[Figure 16: Plotting forecasted data compared to latest Bt data]

```
# Daily average 'bt' forecasting

print("Forecasting average 'bt' for the next 7 days...")

recent_data_subset_daily_avg = combined_data.tail(N_STEPS_AVERAGE * 21)

X_daily_avg, y_daily_avg, scaler_daily_avg =
normalize_and_sequence_data(recent_data_subset_daily_avg, N_STEPS_AVERAGE)

initial_sequence_daily_avg = X_daily_avg[-1].flatten()

daily_averages_bt = forecast_average_bt(model, scaler_daily_avg,
initial_sequence_daily_avg, 7 * N_STEPS_AVERAGE, 1440, N_STEPS_AVERAGE)

avg_bt_output = [f"Average 'bt' on {pd.to_datetime(combined_data.index[-1]).date() +
pd.Timedelta(days=i + 1)}: {avg_bt:.2f}" for i, avg_bt in enumerate(daily_averages_bt)]

print("\n".join(avg_bt_output))
```

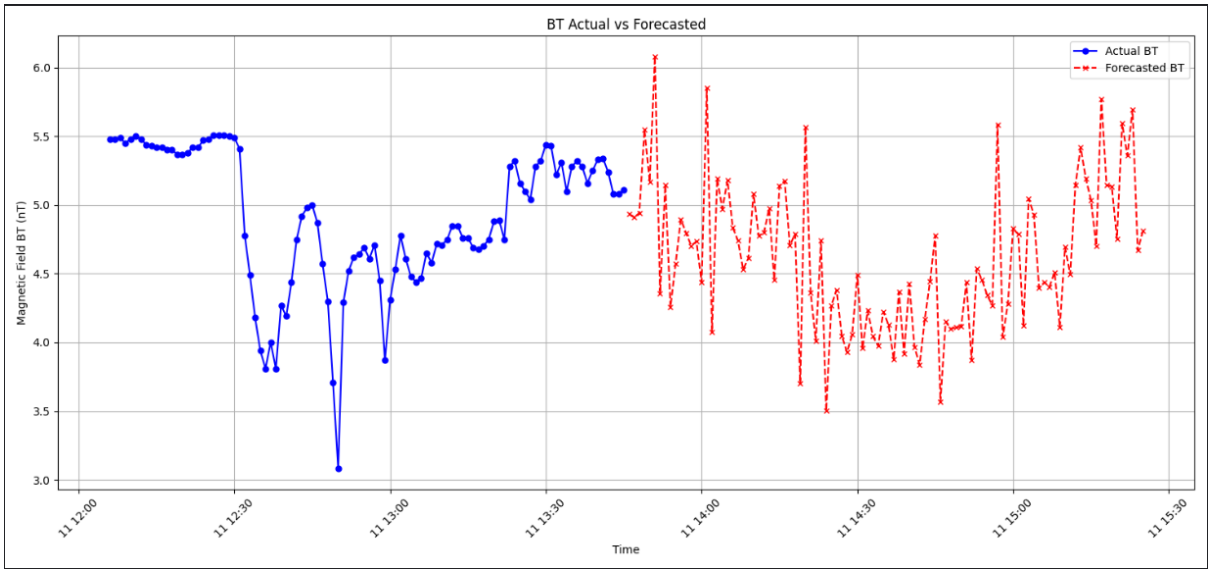
[Figure 17: Average Bt Forecasting method in main.py]



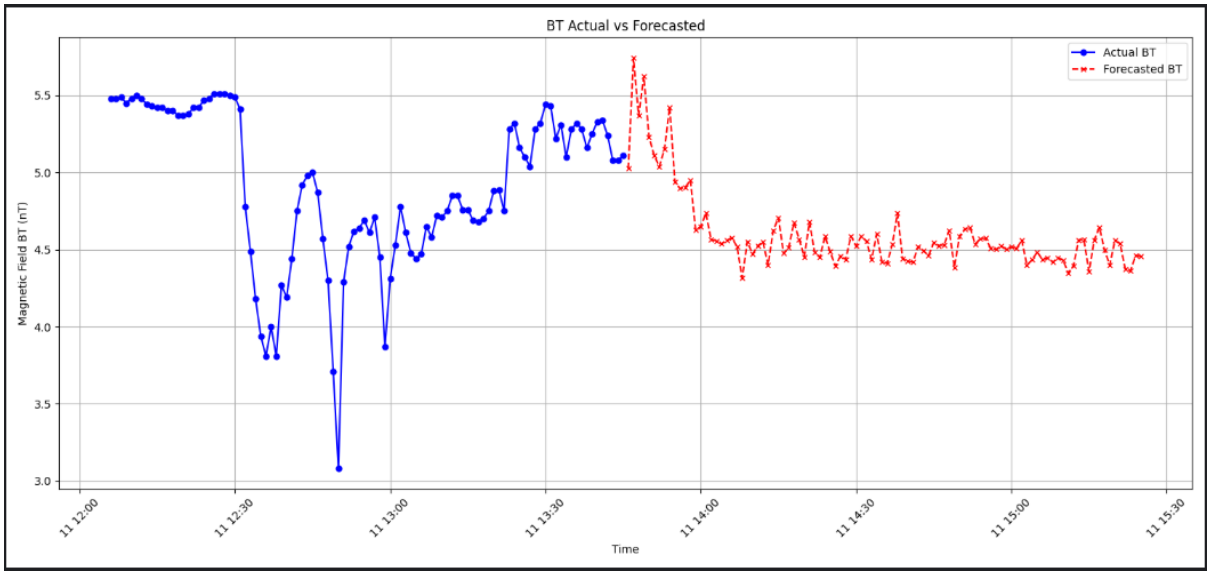

[Figure 18: MSE used in prediction not capturing the more significant errors or fluctuations (LSTM)]



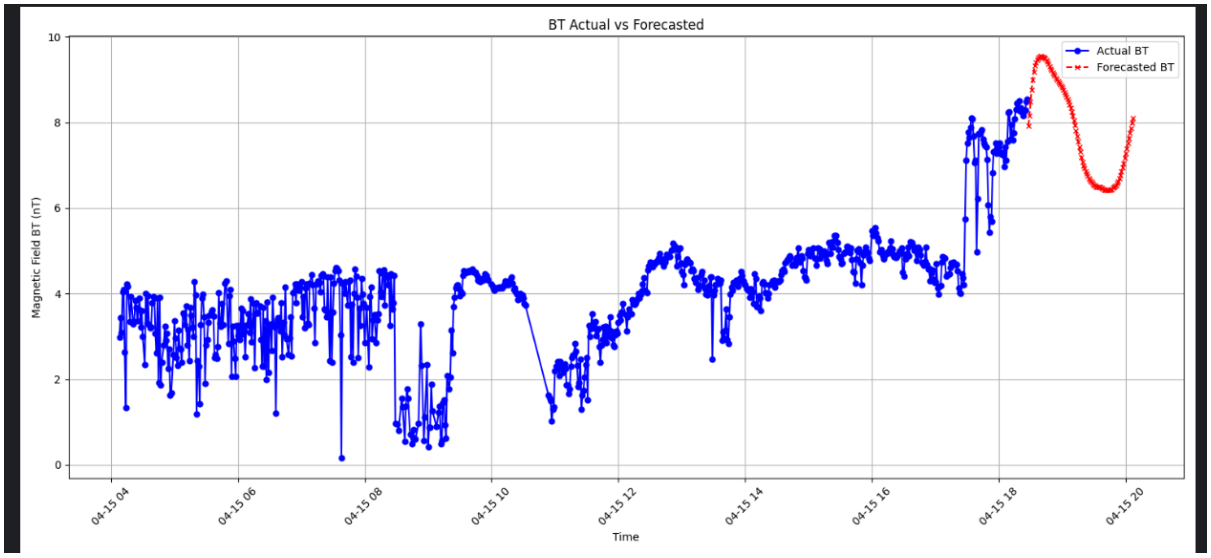

[Figure 19: MAE being used showing more fluctuations and allowing more room for variability (LSTM) Also showing the forecast (in red) compared to previous actual values of Bt (in blue)]



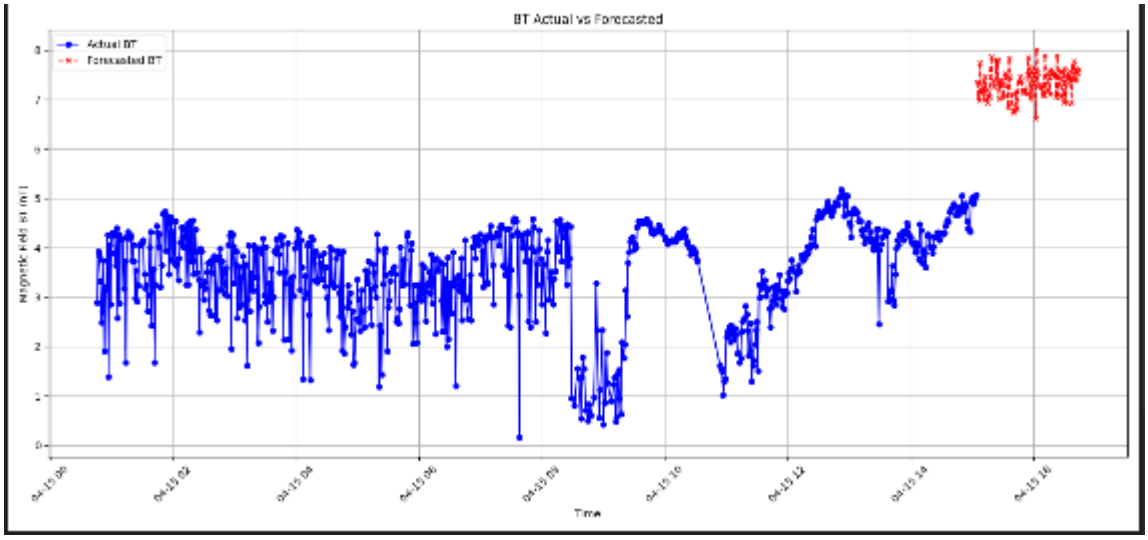

[Figure 20: High noise example]



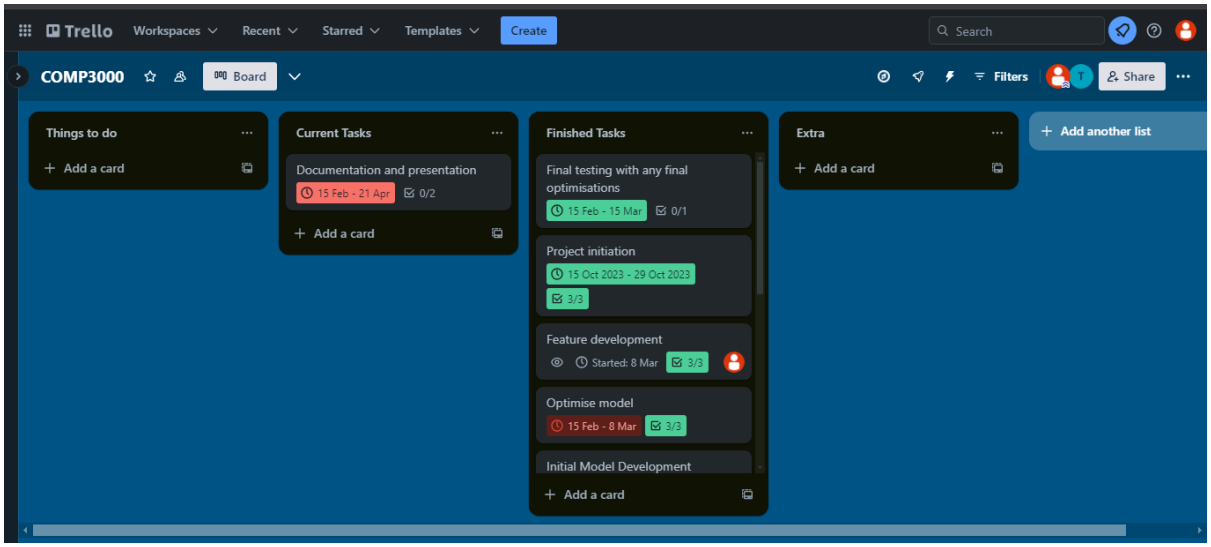

[Figure 21: Low noise example]




[Figure 22: Linear Regression example]




[Figure 23: Random Forest example; captures some consistency but can be off-scale]




[Figure 24: My Kanban board within Trello that I have used for my project, the checklists have recurring similarities to my original project initiation document]

***Entire "data_loader.py" script***

```
import requests
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import numpy as np


def convert_columns_to_numeric(df, columns):
    for col in columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
    return df.dropna()


def fetch_solar_wind_data(url):
    try:
        response = requests.get(url)
```

```
        response.raise_for_status()
        data = response.json()
        columns = data[0]
        actual_data = data[1:]
        df = pd.DataFrame(actual_data, columns=columns)
        df['time_tag'] = pd.to_datetime(df['time_tag'])
        df.set_index('time_tag', inplace=True)
        df = df[['bx_gsm', 'by_gsm', 'bz_gsm', 'bt']]
        return convert_columns_to_numeric(df, ['bx_gsm', 'by_gsm', 'bz_gsm', 'bt'])
    except requests.exceptions.RequestException as e:
        print(f"Request exception: {e}")
        return pd.DataFrame()


def load_new_data(csv_file, reference_timestamp):
    try:
        new_data = pd.read_csv(csv_file)
        # Assuming 'timedelta' is meant to be subtracted to align data correctly
        new_data['timedelta'] = pd.to_timedelta(new_data['timedelta'])
        # Adjusting 'timedelta' to calculate timestamps that should end at the reference
timestamp
        new_data['time_tag'] = reference_timestamp - new_data['timedelta']
        new_data = new_data.sort_values(by='time_tag')
        new_data.set_index('time_tag', inplace=True)
        new_data = new_data[['bx_gsm', 'by_gsm', 'bz_gsm', 'bt']]
        return convert_columns_to_numeric(new_data, ['bx_gsm', 'by_gsm', 'bz_gsm', 'bt'])
    except pd.errors.ParserError as e:
        print(f"CSV parsing error: {e}")
```

```
        return pd.DataFrame()
    except Exception as e:
        print(f"Error loading new data: {e}")
        return pd.DataFrame()


def normalize_and_sequence_data(df, n_steps):
    features = ['bx_gsm', 'by_gsm', 'bz_gsm', 'bt']
    scaler = MinMaxScaler(feature_range=(0, 1))
    df_copy = df.copy()
    df_copy[features] = scaler.fit_transform(df_copy[features])
    X, y = [], []
    for i in range(n_steps, len(df_copy)):
        X.append(df_copy[features].iloc[i - n_steps:i].to_numpy())
        y.append(df_copy[features].iloc[i].to_numpy())
    return np.array(X), np.array(y), scaler
```

***Entire "main.py" script***

```
from data_loader import fetch_solar_wind_data, load_new_data,
normalize_and_sequence_data
from model import train_and_save_model, load_and_update_model, forecast,
forecast_average_bt
import pandas as pd
from visualisation import plot_forecasts, plot_averages
import os

# Define constants and file paths
MODEL_PATH = 'C:/Users/Joe/Desktop/solar_wind_model.keras'
```

```
JSON_URL = 'https://services.swpc.noaa.gov/products/solar-wind/mag-7-day.json'
CSV_FILE = 'C:/Users/Joe/Desktop/Kaggle Data/solar_wind.csv'
N_STEPS_SHORT_TERM = 100
N_STEPS_AVERAGE = 1440  # Different n_steps for the daily average calculation
N_FEATURES = 4  # bx_gsm, by_gsm, bz_gsm, bt



def main():
    print("Starting the forecasting process...")

    # Fetching and combining recent and historical data
    df_recent_data = fetch_solar_wind_data(JSON_URL)
    df_historic_data = load_new_data(CSV_FILE, df_recent_data.index.min())
    combined_data = pd.concat([df_historic_data, df_recent_data], axis=0).sort_index()

    # Preparing data for short-term forecasting
    recent_data_subset_short_term = combined_data.tail(N_STEPS_SHORT_TERM * 100)
    X_short_term, y_short_term, scaler_short_term =
normalize_and_sequence_data(recent_data_subset_short_term,
N_STEPS_SHORT_TERM)

    # Split the data, keeping the latest for testing to simulate future unseen data
    train_size = int(len(X_short_term) * 0.8)
    X_train, y_train = X_short_term[:train_size], y_short_term[:train_size]
    X_test, y_test = X_short_term[train_size:], y_short_term[train_size:]

    # Model training/update
    if os.path.exists(MODEL_PATH):
        print("Loading and updating existing model...")
```

```
    model = load_and_update_model(MODEL_PATH, X_train, y_train, X_val=X_test, y_val=y_test, epochs=5, batch_size=128)
  else:
    print("Building and training a new LSTM model...")
    model = train_and_save_model(X_train, y_train, X_val=X_test, y_val=y_test, model_path=MODEL_PATH, n_features=N_FEATURES, n_steps=N_STEPS_SHORT_TERM, epochs=50, batch_size=128)

  # Short-term forecasting
  print("Model training/update complete. Forecasting the next 200 steps...")
  last_known_sequence = X_test[-1].flatten()
  forecasted_values = forecast(model, scaler_short_term, last_known_sequence, steps=100)
  last_known_time = combined_data.index[-1] + pd.Timedelta(minutes=1)
  forecast_output = []
  for i in range(100):
    forecast_time = last_known_time + pd.Timedelta(minutes=i + 1)
    forecast_data = forecasted_values[i]
    forecast_output.append(f"{forecast_time}, {forecast_data[0]:.2f}, {forecast_data[1]:.2f}, {forecast_data[2]:.2f}, {forecast_data[3]:.2f}")
  print("Forecasted Data for the next 100 steps:")
  print("\n".join(forecast_output))

  # Extract actual 'bt' data for the last 800 minutes for comparison
  actual_bt_data = combined_data['bt'].tail(800)

  # Visualisation of short-term forecasts
  plot_forecasts(actual_bt_data, forecasted_values, last_known_time)
```

```
    # Daily average 'bt' forecasting
    print("Forecasting average 'bt' for the next 7 days...")
    recent_data_subset_daily_avg = combined_data.tail(N_STEPS_AVERAGE * 21)
    X_daily_avg, y_daily_avg, scaler_daily_avg =
normalize_and_sequence_data(recent_data_subset_daily_avg, N_STEPS_AVERAGE)
    initial_sequence_daily_avg = X_daily_avg[-1].flatten()
    daily_averages_bt = forecast_average_bt(model, scaler_daily_avg,
initial_sequence_daily_avg, 7 * N_STEPS_AVERAGE, 1440, N_STEPS_AVERAGE)
    avg_bt_output = [f"Average 'bt' on {pd.to_datetime(combined_data.index[-1]).date() +
pd.Timedelta(days=i + 1)}: {avg_bt:.2f}" for i, avg_bt in enumerate(daily_averages_bt)]
    print("\n".join(avg_bt_output))


    # Calculate past 7 days rolling averages for visualisation
    plot_averages(daily_averages_bt, pd.to_datetime(combined_data.index[-1]).date() +
pd.Timedelta(days=1))



if __name__ == "__main__":
    main()
```

***Entire "model.py" script***

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.layers import Bidirectional, Conv1D, LSTM, Dense, Dropout,
BatchNormalization
from tensorflow.keras.losses import MeanAbsoluteError
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import load_model, Sequential
import numpy as np
```

```
from tensorflow.keras.regularizers import l2



def build_model(n_features, n_steps, output_units=128, dropout_rate=0.3,
regularization_rate=0.0001):
    model = Sequential([
        Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(n_steps,
n_features)),
        BatchNormalization(),
        Dropout(dropout_rate),
        Bidirectional(LSTM(output_units, return_sequences=True, activation='tanh',
kernel_regularizer=l2(regularization_rate))),
        Dropout(dropout_rate),
        LSTM(output_units, activation='tanh', kernel_regularizer=l2(regularization_rate)),
        Dropout(dropout_rate),
        Dense(n_features, activation='relu', kernel_regularizer=l2(regularization_rate))
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_absolute_error')
    return model



def train_and_save_model(X_train, y_train, X_val, y_val, model_path, n_features,
n_steps, epochs=50, batch_size=128):
    model = build_model(n_features, n_steps)
    early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2,
min_lr=0.0001)
    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
validation_data=(X_val, y_val),
```

```
        callbacks=[early_stopping, reduce_lr])
    model.save(model_path)
    return model


def load_and_update_model(model_path, X_train, y_train, X_val, y_val, epochs=5,
batch_size=128):
    custom_objects = {"Adam": Adam, "MeanAbsoluteError": MeanAbsoluteError,
"LSTMCell": LSTM}
    try:
        model = load_model(model_path, custom_objects=custom_objects)
        print("Model loaded successfully!")
    except Exception as e:
        print(f"Error loading model: {str(e)}. Rebuilding model.")
        model = build_model(n_features=X_train.shape[-1], n_steps=X_train.shape[1])
        model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_absolute_error')
        print("New model initialized due to load failure.")

    model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=epochs,
batch_size=batch_size,
        callbacks=[EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True),
                   ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2,
min_lr=0.00001)])
    model.save(model_path)
    return model


def forecast(model, scaler, initial_sequence, steps=100, noise_level=0.12):
```

```
    n_features = 4  # Defined as per my constants
    initial_sequence = initial_sequence.reshape((1, -1, n_features))  # Ensure correct
shape

    forecasts = np.zeros((steps, n_features))
    current_sequence = initial_sequence

    for i in range(steps):
        noisy_sequence = current_sequence + np.random.normal(0, noise_level,
current_sequence.shape)
        forecasted_step = model.predict(noisy_sequence)[0]
        forecasts[i] = forecasted_step
        current_sequence = np.roll(current_sequence, -1, axis=1)
        current_sequence[0, -1, :] = forecasted_step

    forecasts = scaler.inverse_transform(forecasts)
    return forecasts


def forecast_average_bt(model, scaler, initial_sequence, total_steps, steps_per_day,
n_steps, noise_level=0.15):
    n_features = 4  # Defined as per my constants
    initial_sequence = initial_sequence.reshape((1, n_steps, n_features))  # Ensure
correct shape for n_steps

    daily_averages = []
    current_sequence = initial_sequence

    for day in range(total_steps // steps_per_day):
```

```
        daily_forecast = []
        for _ in range(steps_per_day):
            noisy_sequence = current_sequence + np.random.normal(0, noise_level,
current_sequence.shape)
            forecasted_step_scaled = model.predict(noisy_sequence)[0]
            current_sequence = np.roll(current_sequence, -1, axis=1)
            current_sequence[0, -1, :] = forecasted_step_scaled
            forecasted_step = scaler.inverse_transform(forecasted_step_scaled.reshape(1, -
1))[0]
            daily_forecast.append(forecasted_step[3])

        daily_average_bt = np.mean(daily_forecast)
        daily_averages.append(daily_average_bt)

    return np.array(daily_averages)
```

*Entire "Visualisation.py" script*

```
import matplotlib.pyplot as plt
import pandas as pd

def plot_forecasts(actual_data, forecasts, last_known_time):
    plt.figure(figsize=(15, 7))
    forecast_dates = pd.date_range(start=last_known_time + pd.Timedelta(minutes=1),
periods=len(forecasts), freq='T')
    forecast_series = pd.Series(data=[f[3] for f in forecasts], index=forecast_dates)
```

```
    # Plot actual data
    plt.plot(actual_data.index, actual_data, label='Actual BT', marker='o', linestyle='-', color='blue', markersize=5)

    # Plot forecasted data
    plt.plot(forecast_series.index, forecast_series, label='Forecasted BT', marker='x', linestyle='--', color='red', markersize=5)

    # Add a vertical line to mark the transition from actual to forecasted data
    plt.axvline(x=last_known_time, color='black', linestyle='--', label='Forecast Start')

    plt.title('BT Actual vs Forecasted')
    plt.xlabel('Time')
    plt.ylabel('Magnetic Field BT (nT)')
    plt.legend()
    plt.grid(True)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()


def plot_averages(forecasted_data, start_date):
    plt.figure(figsize=(12, 6))
    dates_future = pd.date_range(start=start_date + pd.Timedelta(days=1), periods=len(forecasted_data), freq='D')

    plt.plot(dates_future, forecasted_data, 'r-x', label='Predicted Next 7 Days Average BT')
    plt.title('Predicted Next 7 Days Average BT')
```

```
plt.xlabel('Date')
plt.ylabel('Average BT')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```