Expansion of the Machine Learning Capabilities of the Ground Loop System for the Magnetosphere Multiscale Mission

E. Davis*
University of New Hampshire, Durham, NH, 03824, USA
(Dated: May 20, 2021)

This paper covers work done for the purpose of expanding the Ground Loop burst data management system introduced by Argall, Small, et al. 2020 [1] for the Magnetospheric Multiscale Mission. A brief overview of the mission and reasoning behind the GLS is provided. The completion and optimization of a manual labelling program is discussed. Specifically, lengthy primary component analysis calculations using Scikit-Learn are made significantly faster using two methods: one consists of scrubbing the code of unneeded calculations, the other uses NVIDIA's CUDA and NVIDIA GPUs to perform the calculations directly. It is shown that graphics cards are well-suited to these tasks. Unsuccessful efforts to construct event detection models are reviewed, and future work on the GLS is suggested.

INTRODUCTION

The Magnetospheric Multiscale Mission (MMS) was launched in 2015 to study magnetic reconnection in boundary regions of the Earth's magnetosphere [2]. Reconnection occurs when two regions of plasma with distinct field line sources meet and form a boundary. At this boundary, the field lines are metaphorically torn from their sources and reconnect to one another. This converts magnetic energy to kinetic energy and as a result the plasma is accelerated. An in-depth study of magnetic reconnection is beyond the scope of this paper, but the current understanding is well-established in literature, especially Burch & Drake 2009 [3]. Reconnection is the driving force of Earth's auroras as well as coronal mass ejections and solar storms from the sun. The geomagnetic storms present a hazard to global electronic infrastructure that cannot be discounted. The Carrington Event was a powerful geomagnetic storm that struck Earth on September 1-3, 1859, causing widespread damage to telegraph lines and electrical grids and auroras to be seen as far south as 18 degrees latitude [4]. Carrington-magnitude events cannot be considered simple statistical improbabilities as a 2012 storm of similar magnitude missed Earth by only 9 days [5]. A 2013 study conducted by insurance company Lloyd's of London and the Atmosphere and Environmental Research risk analytics company estimated the damage of a Carrington-magnitude event to be \$0.6-\$2.6 trillion USD to the United States alone [6]. Thus, studying magnetic reconnection is not only of purely physical interest but also has direct application to the security and stability of human society.

MMS Mission Overview and Project Motivation - MMS consists of four spacecraft orbiting the Earth in a tetrahedral formation (see Fig. 1). The orbit passes through the bow shock (on the sun side of Earth) and the magnetotail (in the Earth shadow) as there is significant reconnection activity in these regions [8]. The instrumentation on-board each spacecraft has four modes of data collection: slow-survey, fast-survey, and burst.

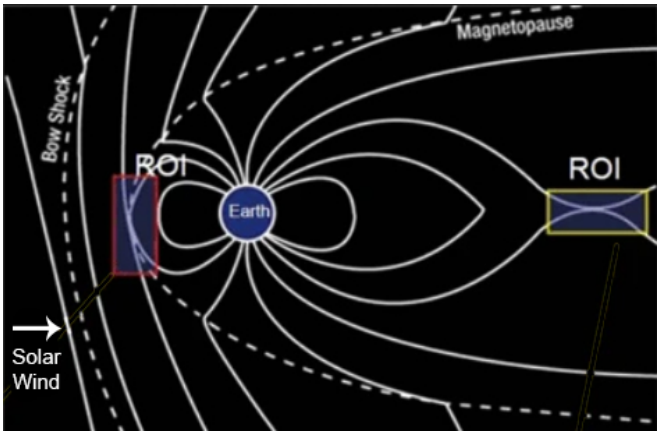

Figure 1. Figure showing regions of interest in MMS's orbit around the Earth. The orbit targets two areas of heavy reconnection, the bow shock (on the sun side, red-boxed ROI) and the magentotail (in the Earth's shadow, yellow-boxed ROI). Image credit: Tooley et al. 2015 [7].

Burst mode data has the highest time resolution, but the mission's use of the S-band of NASA's Deep Space Network means only 4% of burst data can be downlinked [2]. To ensure maximum scientific return on downlinked data, there are two systems in place. One is the automated burst system (ABS), located on-board each spacecraft. The ABS takes in burst-mode readings from each instrument that are averaged over 10 second intervals and applies prescribed tables of weights and offsets to the averaged values [1]. This creates four Cycle Data Quality factors, which are downlinked and weighted once more to create a Mission Data Quality (MDQ) factor [8]. This MDQ is used to prioritize burst data still on-board for downlinking.

The other system in place in the Scientist-in-the-Loop (SITL). Each orbit has divisions into sub-regions of interest (SROIs) which are likely to contain points of scientific interest. SITLs are presented with data from within SROIs by EVA, a graphical software tool suite developed

for the MMS-SITL pipeline. EVA provides the SITL with down-sampled data, assistance in selecting events of interest, and allows selections to be returned to the Science Data Center (SDC) and the selected burst data downlinked [1].

The current burst-selection systems have certainly proven their worth in terms of data output. The Geophysical Research Letters published "First Results from NASA's Magnetospheric Multiscale Mission", a collection of 58 papers published in the first eighteen months of mission operations [9]. A cursory search of Google Scholar returns over 3,000 articles for "Magnetospheric Multiscale Mission" published since mission launch in 2015. There clearly has been no shortage of usable data and new results from mission observations. However, both methods of burst data management have shortcomings that limit scientific potential. For example, the ABS is limited in flexibility by its nature and the computational power aboard the craft, necessitating the averaging of burst data which loses fidelity. The limitations mean the ABS is oriented towards detecting specific subsets of events. For example, Argall et al. 2020 found that when searching for magnetopause crossings, the ABS selected only 34% of all crossings and 28% of all SITL selections in the same frame [1]. The work found that this was due to the ABS being focused on crossings that also contained electron diffusion regions, thus under-selecting crossings that did not feature EDRs. Thus the ABS alone, while effective in certain situations, poses a risk of missing valuable events. The SITL presents its own set of problems. Although the system produces excellent selections, the necessity of a skilled operator spending substantial time making selections can be difficult to fulfill. As the mission ages, that difficulty grows. People at all levels–from students to late-career scientists–move on. Students finish their degrees and pursue other opportunities. Early and mid-career scientists move on to other projects and higher-up positions, while late-career scientists retire. Simultaneously, funding decreases over mission life. For fiscal year 2021, NASA requested $18.7 million USD [10], less than half of FY2015's request for $39.5 million USD [11]. The budget only shrinks beyond 2021, as the proposal projects $16.8 million USD for FY2022 and $15.8 million USD for FY2023-25. The lowered funding makes replacing everyone that leaves impossible, and puts the SITL system at risk of being unsustainable long term. This is where the proposed Ground Loop System comes in.

The Ground Loop System - The Ground Loop System (GLS) is a burst-selection system initially proposed in Argall, Small, et al. 2020 [1]. The system consists of ensembles of machine learning models in a three-tier hierarchy of region, event, and campaign classifiers (see Fig. 2 for a visual overview of the models). An ensemble of machine learning (also referred to as deep learning) models make sense in this application for a variety of reasons. First, classification of time series data is a well-studied and common application of deep learning; Google Scholar returns over 2 million results for a search of the topic. Secondly, there is a large amount of archived data with labelled events at a variety of resolutions, and as of 2020 SITL participants have made selections on over 1,090 orbits' worth of data [1]. Often the most time consuming part of creating a machine learning solution is getting sufficient amounts of labelled data to train and evaluate the model, years of SITL efforts have done this for us. In addition, the nature of deep learning means that future SITL-selected data can be used to improve the models as they age. The ensemble system allows each model to be trained for a specific task. An alternative system using one large model would suffer in performance for a myriad of reasons. First, it would need to learn to recognize all desired patterns in the data corresponding to events, necessitating a large model. Training such a model would take substantial data and be computationally expensive, and the system would be unusable until the model was complete. Finally, if it became necessary to expand the model, retraining would require not only data of new events but also intermittent data of recognized events so as to not bias the system towards the new events. An ensemble of models solves all of these problems. Since each model is focused on a region or subset of events, they can be smaller and thus trained and used much more quickly. Given SITL data is specifically labelled, it is simple to collect training and validation datasets for models with a narrow focus. Further, the system can be implemented in stages, rather than all at once. Indeed, Argall, Small, et al. showed a neural network in their 2020 paper that selected 78% of SITL-selected magnetopause crossings and generally showed much higher accuracy than the ABS alone [1]. As models are trained and become operational, they can be slotted into their place in the hierarchy. This modularity also allows future science to be extracted and integrated into the system. For example, if a phenomenon is believed to be related to other events or contained in a region, a sample of data believed to contain the phenomenon could be fed into the GLS, and the output used to study correlations or new theories. In addition, new models based on future data could be re-run through archived data, adding scientific value to data that may have otherwise been underused.

The work this paper is based upon consisted of two sub-projects, each contributing to the Ground Loop System. The first was the completion and optimization of a non-EVA GUI for manual event labelling. The second sub-project consisted of creating and evaluating machine learning networks for identification of solar wind and foreshock crossings.

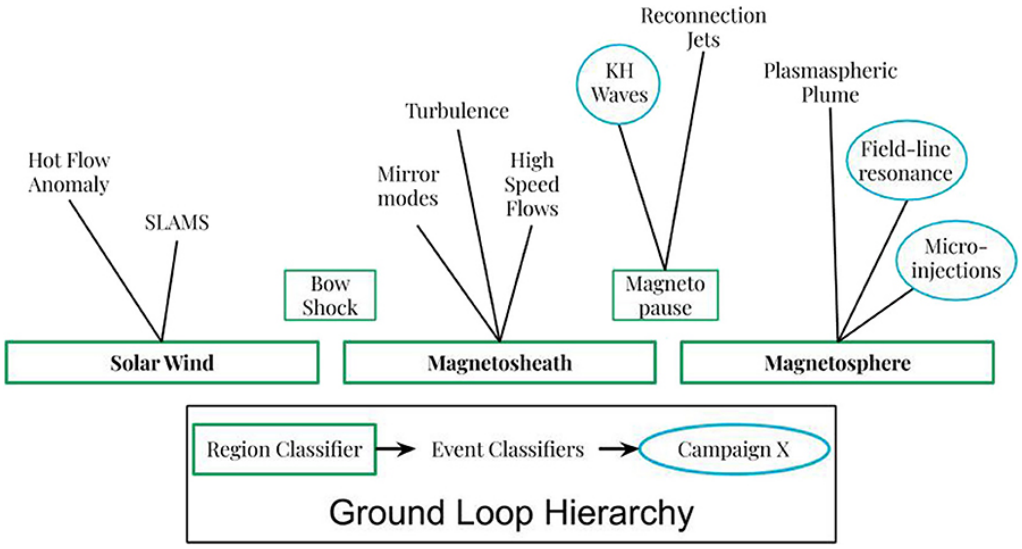

Figure 2. Figure showing a broad overview of the GLS hierarchy. The system enables models trained for specific events or scientific campaigns to make predictions on burst data in regions containing those events, as opposed to all downlinked burst data. Machine learning models will improve as more data is processed, and the modularity allows the Ground Loop to both evolve over time and use future models to find new science in archived data. Figure credit: Argall et al. [1].

MANUAL LABELLER COMPLETION AND OPTIMIZATION

The labeller project was started by Slava Olshevsky while working on a joint project between KTH and IRFU, both of Sweden [12]. The program is written in Python and allows for informed evaluation and relabelling of local copies of spacecraft data. The data is stored in common data format (CDF) files, a format developed by the National Space Science Data Center specifically for space science missions [13]. While there are CDF tools available, there is limited capability for tasks such as large-scale editing of labels. The labeller uses CDFLib [14], a Python package written to interface with and edit CDF files. The program was created with an eye towards data extraction and machine learning projects, allowing for label adjustments during training without needing the full SITL GUI or to muddle through command-line CDF tools.

The labeller works in two parts. First is the classifier, which takes in spacecraft data in the form of CDFs and outputs a compressed file containing the data, assigned labels, and a primary component analysis (PCA). The labels are assigned through an included set of in-progress machine learning models created by Olshevsky; however, this can be avoided and the labels written to a default. The PCA requires additional discussion (see below). Once the PCA is complete, the file is constructed and compressed using Python's built in pickle function. The pickle file is loaded into the labeller GUI, which displays plots of a selection of attributes as well as a plot of assigned labels overlaid on a map of the orbit (see Fig. 3).

A primary component analysis is a linear transformation on a matrix of data $\hat{X}$ of size $m \times n$, where m is the number of samples (of time series data for the purposes of this work) and n is the number of features in the data. Conceptually, the PCA takes in $\hat{X}$ and returns a matrix of principal components $\hat{P}$ and of weights $\hat{W}$. The columns of $\hat{W}$ turn out to be the eigenvectors of the covariance matrix, $\hat{X^T}\hat{X}$. The principal components are a list of vectors, with each vector describing an axis through the n-dimensional data in $\hat{X}$ that maximizes the variance of the data, and has two specific properties: each principal component is orthogonal to all others, in other words, the variance between each component is zero. The other is that the total variance in the original data is concentrated in the first relatively few principal components. This makes PCA extremely useful as a dimensionality reduction technique to go from n original dimensions to p new dimensions. The labeller program's PCA takes in data with upwards of 10^6 dimensions, while the PCA output has $16,384$ dimensions which contain an average of 97% of the original variance. This makes later steps in the process more tractable as the computer has much less data to work with and plot.

When I received the code, it was approximately 95% written, but a significant portion was not working. The solutions were straightforward and very typical python fixes and none individually merit discussion in this paper. However, the PCA presented a challenge. It is extremely computationally expensive, and the program took upwards of one hour to calculate the PCA for one month of data. When timing methods were inserted into

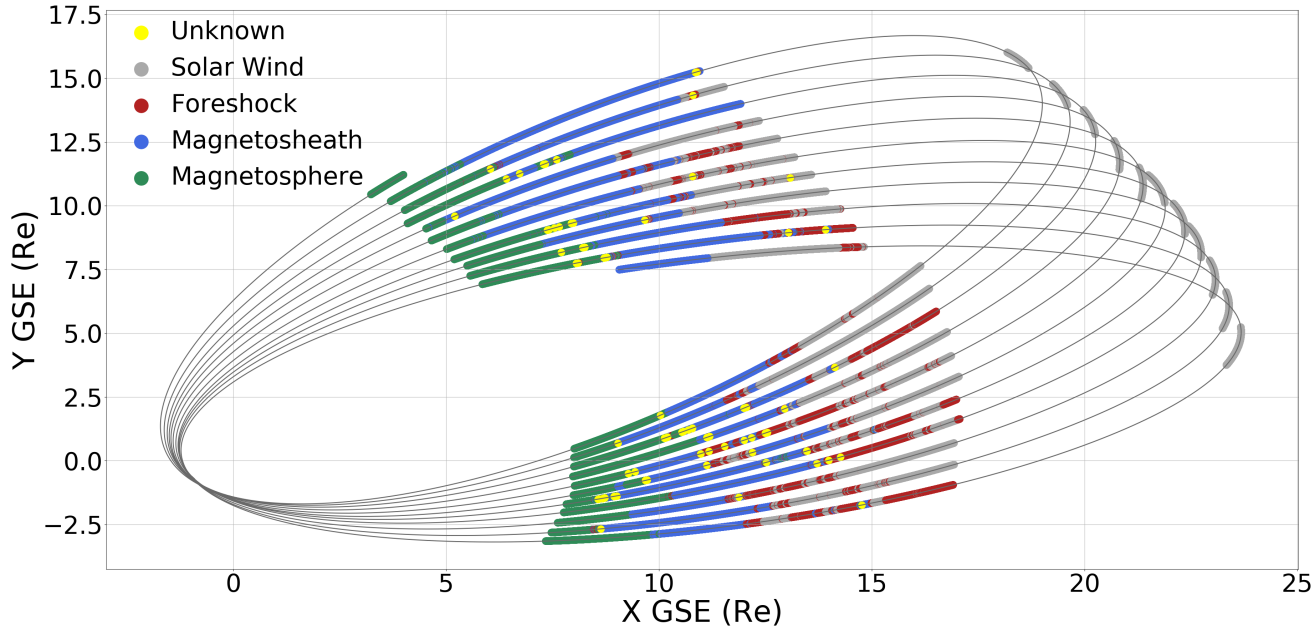

Figure 3. Figure showing output of the labeller program with model predictions overlaid on an orbit map. Figure recreated from Olshevsky [12].

the code, it was found that calculating the PCA and pickling its output took well over 99% of the program runtime. Thus my next goal was to improve the speed of the program by speeding up the PCA. Initially, the program used the PCA method of Scikit-Learn [15], a popular machine learning Python package. Scikit's PCA includes calculation products in its output that were being left unused in the labelling process, so the first attempt consisted of manually scrubbing the SciKit code of this extraneous data. Although this improved matters (see results section below), the calculation was still extremely slow, so I moved on to attempting to convert the PCA from running on the CPU to running on the GPU.

GPUs for Parallel Calculations - Running the PCA calculation on a graphics card provides a speed increase through parallelization. Although the primary components can be found by calculating the eigenvectors of $\hat{X^T}\hat{X}$, that method is not computationally efficient nor is it well-disposed to being parallelized. Rather, most algorithims for PCA instead use singular value decomposition, where matrices are calculated such that:

$$\hat{X} = \hat{U}\hat{\Sigma}\hat{W^T} \tag{1}$$

where $\hat{U}$ is an $m \times m$ matrix made up of columns of orthogonal unit vectors (called the left singular vectors), $\hat{W^T}$ is a $p \times p$ matrix with columns of orthognal unit vectors (called the right singular vectors), and $\hat{\Sigma}$ is an $m \times p$ diagonal matrix of numbers called the singular values of $\hat{X}$. The details of the numerical algorithms used in to find these matrices and the parallelization thereof is far outside the scope of this work but are well-established in literature, such as Jessup and Sorensen 1994 [16].

There are two architectural factors in CPUs and GPUs that combine to make GPUs faster for certain parallelized work. First is their distributions and types of cores. Using my own desktop as an example, my CPU has 8 cores running at a maximum of 5.0GHz each. Because the CPU must handle nearly every task the computer does, the cores are generalized to be good at performing many tasks. Contrarily, the GPU contains 3584 cores running at a maximum of 1837MHz each. The GPU's primary task is rendering video and photo information, which mostly consists of distributing, performing, and recombining many matrix operations simultaneously. The count of GPU cores does not include specialized Tensor cores that are optimized for heavy workload environments. The second factor is in their memory distributions and how the cores interact with the memory. The CPU stores data in the RAM for quick access, however, the cores do not interact directly with the RAM, but rather with the CPU's cache, in this case 16MB. The memory lanes from the RAM to the CPU cache are latency-optimized, meaning they are designed to move small bits of information extremely quickly, so that as the data in the cache is used and disposed of there is more to reload. The GPU, on the other hand, is designed with VRAM (video RAM) on-board, in this case 12GB. The VRAM is designed so that it can be accessed by many cores at once, allowing for faster calculations in this application. In short, the memory is bandwidth optimized. Moving

data from storage to the GPU takes more time, but it can move more data at once and use more of the data once it has arrived, as compared to the CPU. A common metaphor is delivering physical packages of data, with the CPU's memory retrieval system represented by a fast, two-seater sports car and the GPU's by a tractor-trailer truck. The sports car can only take a few packages at once, but can move them much, much more rapidly. The tractor trailer is much slower to load, move, and unload, but can move thousands of packages at once. Expanding the metaphor to include the hardware, the CPU is a package processing facility with a few generalist workers who are extremely quick. They can intake, sort, and process packages all on their own in a short time. However, because each delivery only consists of a few packages, workers will end up waiting around for more packages in between arrivals. Preventing this downtime would require an entire fleet of sports cars constantly arriving and leaving, and even with that system in place each worker is only doing a few packages at a time. The GPU, on the other hand, is like a large processing facility with thousands of specialist workers. Some are in charge of distributing work to teams of workers, and teams only carry out specific tasks. Thus, each individual package takes a longer time to get through the facility, but between the organization of the workers and the thousands of packages delivered by the tractor trailer, the workers can spend more time working at maximum throughput and ultimately process more total packages in a period of time. Additionally, the delay in loading, moving, and unloading the tractor trailer is offset once the first truck arrives, as the workers are busy with the packages already delivered and there is no need for immediate followup as there is with the sports cars.

To use a graphics card for this application, an NVIDIA GPU is required. This is because NVIDIA created CUDA [17], a software toolkit written in C that contains a myriad of common mathematical and data science functions. The Scikit PCA was replaced with a combination of PyCUDA [18], which acts as an interface between Python code and the CUDA compiler, and Scikit-CUDA [19], which replaces much of Scikit-Learn with CUDA-enabled code. The Scikit-CUDA code required some modification to run properly, specifically a need to transpose input matrices when running on Windows and to filter extraneous products before they are packaged in the pickled file.

Results and Discussion - A full table of results of the PCA modifications can be found in Table . The same results are represented visually in Fig. 4. Unsurprisingly, the standard PCA package is very slow, even on a newer 8 core processor. Despite being an ad-hoc solution, the reduced PCA is more than twice as fast as the standard. However this is the result of cutting out what is not used as opposed to actually making the calculation faster. The process of cutting a significant portion of the calculation

PCA Optimization Results			
Hardware (Type)	Algorithm	Runtime (s)	Processing Rate (s/GB)
6.0GB of Data			
Intel X5690 CPU	Scikit-Learn	2957 s	493 s/GB
Intel X5690 CPU	Reduced Scikit	1084 s	180 s/GB
Intel i9-9900K CPU	Scikit-Learn	1629 s	272 s/GB
Intel i9-9900K CPU	Scikit-Learn	541 s	90 s/GB
GTX 1080Ti GPU	Scikit-CUDA	546 s	91 s/GB
RTX 3060 GPU	Scikit-CUDA	782 s	130 s/GB

Table I. Results of different hardware and software combinations on the time taken for PCA calculations. The newer Intel CPU is rivaled by the two-generation old 1080Ti. The 3060, by far the more modern card, is held up by its architecture necessitating CUDA 11.1+, while PyCUDA and Scikit-CUDA are written in CUDA 10. Until the packages are updated, subpar performance is to be expected.

while still returning the principal components was tedious and required a fair amount of trial and error while editing the code of the package itself. Thus while it is a solution, it is not one that could be easily or reliably implemented in a range of environments.

The CUDA results from the older GTX card appear to validate the appraoch. A two-generation and four-year-old GTX 1080Ti GPU is able to perform the calculation is less than one-third of the time of an 18-month-old, 8-core i9 CPU running the standard PCA. The graphics card is able to move the entire 6GB dataset into its on-board VRAM and distribute the calculation tasks and necessary data to its cores much more quickly than the CPU despite lower per-core performance and much higher memory latency. The currently-new RTX 3060 card, on the other hand, presents an unwelcome surprise. Despite having every advantage on paper, the 3060 takes almost four minutes longer than the 1080Ti on the same dataset. This is due to an unfortunate combination of factors, primarily the 3060 being a brand-new GPU. With the release of their "3000 series" of graphics cards, NVIDIA introduced a new card architecture called Ampere. The differences in the new architecture are so fundamental that NVIDIA has effectively drawn a line in the sand with the CUDA toolkit. All Ampere cards can run only CUDA 11.1+, whereas previous generations of cards extended backwards compatibility to at least one preceding CUDA version [17]. However, many software packages (including PyCUDA and Scikit-CUDA) have not been updated from the CUDA 10 standard. As a stopgap for these sorts of issues, CUDA 11 can compile CUDA 10

code, but this means the CUDA 10 code is first compiled it its native environment and then translated and recompiled in CUDA 11. This is not only hugely inefficient but also prevents the use of CUDA 11-specific methods, hugely undermining the architecture-based strengths of the new card.

In light of these factors, the question of which method should be used is highly machine-dependent. On a modern desktop or workstation with immense processing power, the standard Scikit-Learn PCA provides a reliable, plug and play method for performing the calculation–albeit slowly. The reduced Scikit PCA is much faster, but requires a deep dive in editing the Scikit package. Changing the package would impact any other code using it in the same environment, making the reduced option unappealing to those who use Scikit frequently. The CUDA PCA is at least equally as fast, and requires little to no editing. However, CUDA requires an NVIDIA GPU at minimum, and a fairly powerful (and therefore expensive) card would need to be used to see gains over modern processors. In addition, the current generation of NVIDIA GPUs are held back by software incongruencies and expected hiccoughs for new technology. These results thus stand as a broader lesson about the risks of adopting new technology; especially when working in scientific computing environments that frequently require flexibility in versioning and employ "out of date" tools.

CREATION AND EVALUATION OF DETECTION MODELS FOR THE GROUND LOOP SYSTEM

Methods - As part of this project, I was tasked with constructing additional models to detect additional events. Argall, Small, et al. [1] outlines the creation of a Bidirectional Long Short Term Memory (LSTM) neural network for detecting magnetopause crossings. The procedure of this work closely follows the steps outlined in the 2020 paper. All event detection networks were built in TensorFlow [20] for Python. Using TensorFlow allows all work to stay in Python, and given it is an academic and industry mainstay, it stands an excellent chance of providing the long-term use and modularity desired from the GLS.

MMS data was collected through the PyMMS [21] API. PyMMS downloads SITL and spacecraft data from the MMS Science Data Center then reads and re-formats the CDFs into a CSV file suitable for training and evaluating neural networks. It is able to filter data and output CSVs by event type based on SITL labels and has proven invaluable on this project. For all three events, work was focused around using Bidirectional LSTMs after the manner of Argall, Small, et al. Although the function of LSTMs is too broad for this work, it is helpful to have some intuition for the structure. A visual representation of an LSTM "spread out" across three timesteps can be found in Figure 5. Each LSTM "box" in the figure holds a series of gates and a hidden state, made up of information from previous states. The gates are activation functions, which map any input to a defined subset of outputs. By holding, passing, or forgetting data throughout training, the model can learn what data should be held, passed, or forgotten. Normally, an LSTM network only works forward in time, so that at a given step in training or use, the hidden state is based on earlier inputs and states [22]. A bidirectional LSTM effectively adds another LSTM network to the first, but the added network is facing backwards in time. Hidden states in the "backward" LSTMs are based on data from "future" timesteps, allowing recognition of correlations bidirectionally in time.

A particular challenge with MMS datasets is that the vast majority of timesteps in any timeframes of a month or longer are not selected. This creates a problem evaluating the model as it trains. Accuracy is not suitable; when 95% or more of points are not selected, the model would achieve 95% accuracy or better simply by predicting every point to be non-selected. There are recall and precision (also called sensitivity), which mathematically are:

$$R = \frac{TP}{TP + FN} \tag{2}$$

and

$$P = \frac{TP}{TP + FP} \tag{3}$$

where TP represents true positives, FN represents false negatives, and FP represents false positives. Viewing recall and precision as equations may not make their meaning leap out. Consider a model that identifies photos of Labrador retrievers. Intuitively, recall is answering "What portion of all Labrador retriever photos were correctly identified?". Likewise, precision could be characterized as answering "What portion of photos identified as a Labrador were actually Labradors?" Using these metrics would be better than accuracy alone in this data situation. They are nonetheless at risk of oversimplifying the problem for the model. If only recall were considered, a naive positive for every point will maximize the recall score. Leaning too heavily on precision creates a model that is too picky. In the MMS application, a picky model will miss events, while a model outputting naive selection is of no use in narrowing down burst downlink regions. However, as used by Argall, Small, et al., using a metric called the F1 score forces the neural network to balance precision and recall. The F1 score is defined as the harmonic mean of the precision and recall scores. We can write the harmonic mean of two numbers as:

$$\mathrm{F1} = \frac{2(R \cdot P)}{R + P} \tag{4}$$

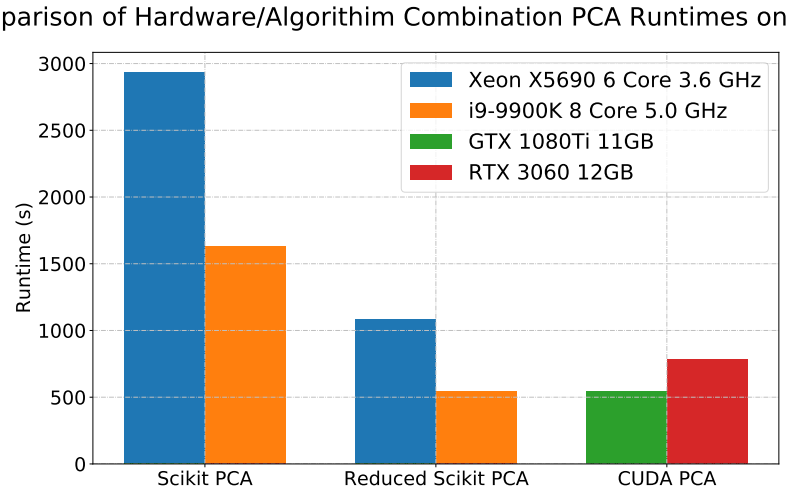

Figure 4. Bar chart showing comparison of PCA runtimes for different software/hardware combinations on 6GB of MMS data. Note the increase in time from the 1080Ti to the 3060; this is unexpected as the 3060 is a much more modern and faster card. The slowdown is due to incongruencies in the card's native CUDA versions of 11.1+ and the software running in CUDA 10.

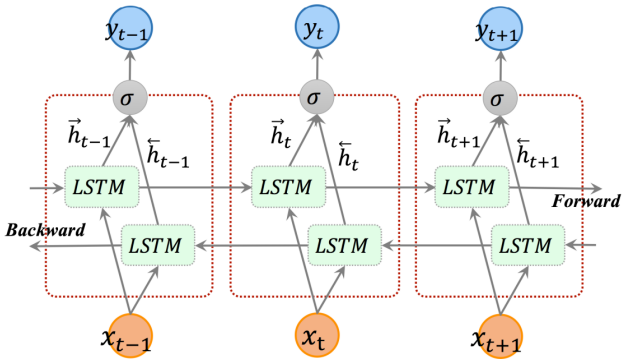

Figure 5. A graphic showing the basic structure of a bidirectional LSTM through three timesteps. Each input goes to forward and backward LSTMs. Depending on inputs and the state of their internal gates, LSTMs can hold, forget, or pass along information about the state for an arbitrary amount of time as training progresses. This allows for establishment of long-term dependencies in time-series data [22]. Figure source: Zhiyong et al. 2019 [23].

This can be expanded into a TP/FP equation, but that form doesn't do anything for the intuition. Consider the nature of equation 4 given $1 \geq R, P \geq 0$. Unless $R = P = 1$, $R \cdot P < R + P$. Even with the factor of two, the nature of the harmonic mean requires both scores to be high to get a good F1 score. For example, a model that exhibits a recall score of 0.9 and a precision of 0.2 will receive F1 $= \frac{2 \cdot 0.18}{1.1} = 0.33$, while a model with 0.7 recall and 0.4 precision would be scored F1 $= \frac{2 \cdot 0.28}{1.1} = 0.51$. Prioritizing F1 score creates more balanced, useful models.

Results and Discussion - A selection of results for the two event detection models can be seen in Table II. Both models exhibit obvious shortcomings. The magnetosphere crossing model only selected 21% of SITL-selected crossings (169 timesteps), yet managed to select 36,100 false positives. The solar wind model seems to show some promise, as it selected 94% of SITL-selected solar wind crossings. However, it also selected over 1.5 million false positives, completely defeating its own purpose of selecting 4% of burst data or less. Additionally, Figures 6 and 7 show a comparison made on data the models had never encountered before. The top plot shows the selections made by the SITL. The middle plot shows the unfiltered output of the model prediction function. The continuity of the function (as opposed to discreet points) is very visible in the middle plots as well. The bottom plot in each figure consists of the model selections after being run through a selection threshold of 0.5. This clips the data at y = 0.5, any points with y ¿ 0.5 are set to 1, any points with y ¡ 0.5 go to zero. Although applying the selection threshold cleans up the model's predictions plot, the disparities in selection are plainly visible.

Event	TP	FN	FP	Recall	Precision	F1
Magnetosphere	169	649	36100	0.21	0.0046	0.009
Solar Wind	25257	1658	$1.5 \cdot 10^6$	0.94	0.016	0.031

Table II. Table of summarized results for validation of event detection models on never before seen data. Both models exhibit a plethora of issues, especially the solar wind model's over-selection issue.

The primary challenges in developing the models are: first, that neural networks have tens of adjustable hyperparameters that allow for fine tuning but also make building a suitable network a challenge, especially as small changes can have disproportionate effects. There are few

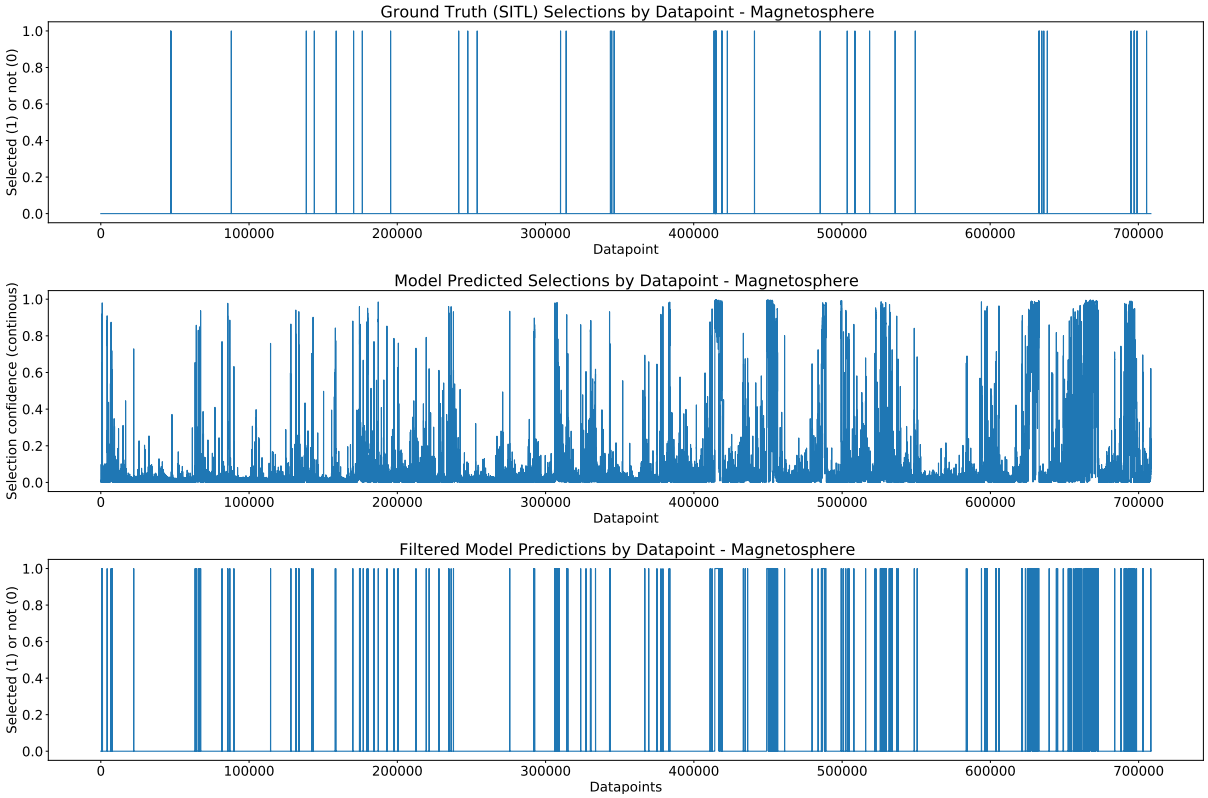

Figure 6. Collection of plots showing timesteps selected by the SITL (top), the unfiltered magnetosphere model output (middle), and the MS model output with a threshold of 0.5 applied.

if any guidelines for how to set hyperparameters initially, and it can be difficult to correlate changes in parameters with changes in model performance. The sheer number of attributes to change can make the process feel out of control and sometimes overwhelming. The other hurdle is that the process of training a network is computationally expensive and generally slow, especially as datasets grow in size. Even though there are uncountable adjustments that can be made, the reality is more and more time is spent waiting for training to finish so previous changes can be evaluated. Unfortunately, I wasn't able to overcome those challenges by the time of writing and the models at present would not be suitable for use in the GLS. Some challenges were encountered and overcome in the process. The aforementioned disparity in selected versus non-selected points wreaked havoc on early network designs. I learned how to dial in the weights given to each set of points and used techniques such as oversampling and undersampling. Undersampling removes data points in the majority class until a set proportion is reached. Conversely, oversampling duplicates data in the minority class to reach the target proportion. Undersampling tended to work better in this data, oversampling increased the rate of overfit as the model learned the heavily-weighted selected points too well to generalize. Trying different optimizers changed how quickly the models learned but didn't often change broad outcomes. Though I certainly still have much to learn, the hours spent working with models on this project have become valuable experience and knowledge moving forward.

CONCLUSIONS AND FUTURE WORK

Despite not succeeding at producing more event detection models, I feel this project was a strong success. The experience of finding and crafting solutions to unique problems is always valuable. Prior to this project I had very limited, self-taught experience in machine learning and being able to work with and learn from people who have much more knowledge and experience has grown my own knowledge but also my interest in the field, and I have started looking out more for career paths involving data science and machine learning. There was also the previously mentioned lesson on being careful of being too cutting edge when working with software from a variety of sources and development environments. I have also become much more aware of the role data systems currently play and will play in the future in our day-to-day lives. I see problems now and wonder if they could be solved with the right lines of TensorFlow and a labelled data set.

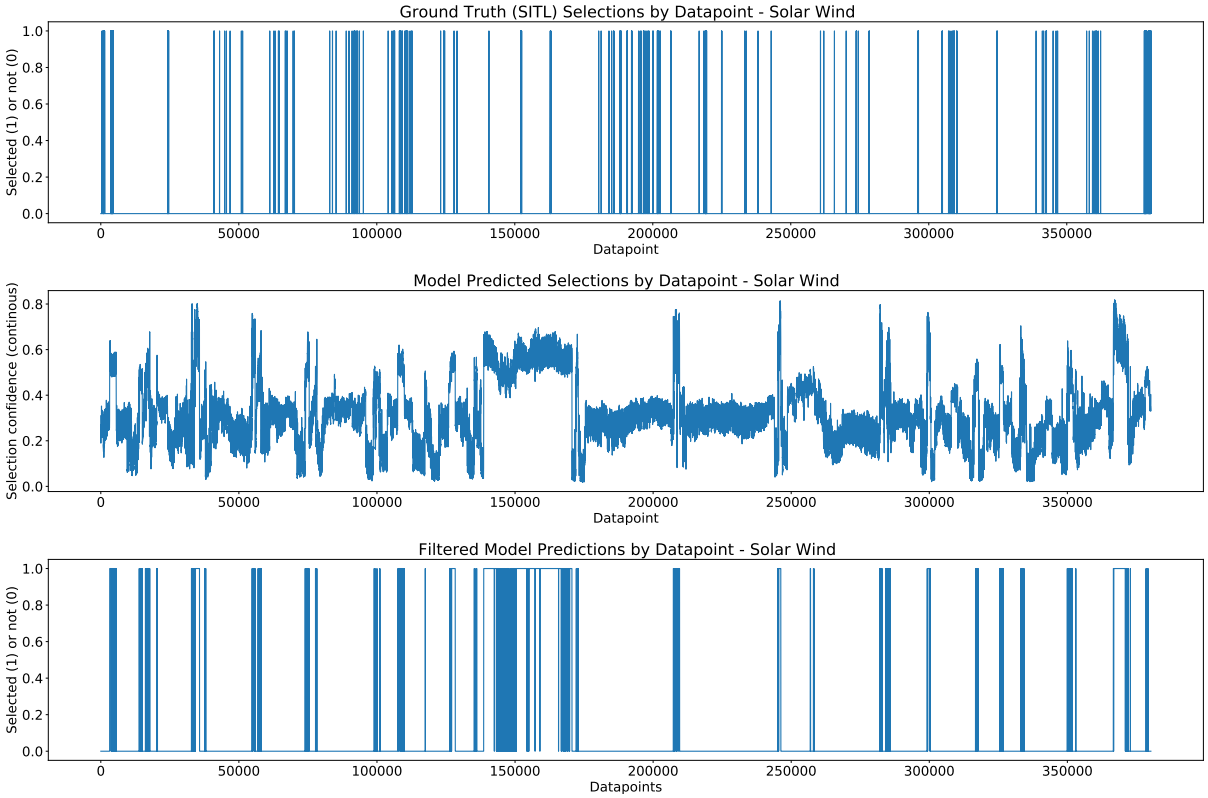

Figure 7. Collection of plots showing timesteps selected by the SITL (top), the unfiltered solar wind model output (middle), and the SW model output with a threshold of 0.5 applied.

There is an enormous amount of future work to be done on the Ground Loop System. In addition to the models that have been started and need to be improved, other events still need models constructed to identify them, and operational models can always be trained and tweaked more to get the best possible results. As more models are created and the system takes shape, science opportunities will grow as scientists spend less time being SITL and more time pursuing research and finding new physics. The GLS could also serve as a template for automating data extraction as well as tasks similar to burst downlink management. Alternatively, the GLS and similar systems could be specifically tailored for later-stage missions. After all, training the models would be impossible without years of data labelled by SITLs. Regardless, the Ground Loop System will continue to open significant scientific doors through MMS.

* Corresponding author: epd1007@wildcats.unh.edu

[1] M. R. Argall, C. R. Small, S. Piatt, L. Breen, M. Petrik, K. Kokkonen, J. Barnum, K. Larsen, F. D. Wilder, M. Oka, W. R. Paterson, R. B. Torbert, R. E. Ergun, T. Phan, B. L. Giles, and J. L. Burch, Frontiers in Astronomy and Space Sciences **7**, 54 (2020).
[2] J. L. Burch, T. E. Moore, R. B. Torbert, and B. L. Giles, **199**, 5 (2016).
[3] J. Burch and J. Drake, American Scientist **97**, 392 (2009).
[4] J. L. Green and S. Boardsen, Advances in Space Research **38**, 130–135 (2006).
[5] T. Phillips, Near miss: The solar superstorm of july 2012 (2014).
[6] L. of London and AER, Solar storm risk to the north american electric grid.
[7] C. R. Tooley, R. K. Black, B. P. Robertson, J. M. Stone, S. E. Pope, and G. T. Davis, Space Science Reviews **199**, 23–76 (2015).
[8] S. Fuselier *et al.*, Space Sci Rev **199**, 77 (2016).
[9] First results from nasa's magnetospheric multiscale (mms) mission: Geophysical research letters.
[10] B. Dunbar, Fy 2021 nasa budget (2021).
[11] B. Dunbar, Fy 2015 budget proposal (archive) (2015).
[12] V. Olshevsky, (2021).
[13] NASA, Cdf.
[14] D. Stansby, M. SDC, B. Harter, M. Hirsch, H. van Kemenade, A. Burrell, J. Smith, A. J. Weiss, J. Ireland, J. C. Terasa, P. L. Lim, and jibarnum, Mavensdc/cdflib: (2021).
[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Journal of Machine Learning Research **12**, 2825 (2011).
[16] E. R. Jessup and D. C. Sorensen, SIAM Journal on Matrix Analysis and Applications **15**, 530–548 (1994).

[17] J. Nickolls, I. Buck, M. Garland, and K. Skadron, Queue **6**, 40–53 (2008).

[18] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, Parallel Computing **38**, 157 (2012).

[19] L. E. Givon, T. Unterthiner, N. B. Erichson, D. W. Chiang, E. Larson, L. A. Pfister, S. Dieleman, G. R. Lee, S. van der Walt, B. Menn, T. M. Moldovan, F. Bastien, X. Shi, J. Schlüter, B. Thomas, C. Capdevila, A. Rubinsteyn, M. M. Forbes, J. Frelinger, T. Klein, B. Merry, N. Merill, L. Pastewka, L. Y. Liu, S. Clarkson, M. Rader, S. Taylor, A. Bergeron, N. H. Ukani, F. Wang, W.-K. Lee, and Y. Zhou, scikit-cuda 0.5.3 (2019).

[20] T. Developers, Tensorflow (2021), Specific TensorFlow versions can be found in the "Versions" list on the right side of this page.¡br¿See the full list of authors ¡a href="https://github.com/tensorflow/tensorflow/graphs/contributors"¿on GitHub¡/a¿.

[21] M. Argall, colinrsmall, and M. Petrik, argallmr/pymms: v0.4.0 (2020-11-16) (2020).

[22] S. Hochreiter and J. Schmidhuber, Neural computation **9**, 1735 (1997).

[23] Z. Cui, R. Ke, Z. Pu, and Y. Wang, Deep bidirectional and unidirectional lstm recurrent neural network for network-wide traffic speed prediction (2019), arXiv:1801.02143 [cs.LG].