Big Data Processing for Power Grid Event Detection

Bruno P. Leao
Siemens Technology
Princeton, NJ
bruno.leao@siemens.com

Dmitriy Fradkin
Siemens Technology
Princeton, NJ
dmitriy.fradkin@siemens.com

Yubo Wang
Siemens Technology
Princeton, NJ
yubo.wang@siemens.com

Sindhu Suresh
Siemens Technology
Princeton, NJ
sindhu-suresh@siemens.com

***Abstract*— In this paper we present the application of big data processing for the development of machine learning (ML) models to detect relevant events in power grid operations. This is based on almost 20TB of phasor measurement unit data corresponding to up to two years of operation of three grid interconnections which provide power to most of the United States. A significant aspect of the work consists in having all data processing performed on a single standard GPU server, from pre-processing to ML model training and testing. We describe the data and computational infrastructure, challenges faced and methods used in data processing, main findings and results. The ML approach employed for best utilization of the big data is also discussed, including sample results.**

Keywords—Big Data, Power Grid, GPU, Phasor Measurement Unit, Machine Learning, Event Detection

I. Introduction

In this paper we present the application of big data processing for the development of machine learning (ML) models to detect relevant events in power grid operations. Power grids are complex distributed systems which can extend over the area of a country or a continent. Modern life has become increasingly dependent on electric power. Detection and identification of events such as short circuits and oscillations have the potential of preventing power outages, reducing operation costs and increasing system reliability. Existing tools already provide grid operators with a certain level of information about relevant occurrences, but they have many limitations. Usually these tools can indicate the occurrence of some anomalous conditions, but information about event types and root causes, associated impacts and actions required to remediate the situation depend on manual analysis by domain experts. Machine learning models trained on big data from the power grid have the potential of automating this process, making it more efficient and effective. Therefore, this is an active field of research in academia and industry [1][2].

Big data used in this work is obtained from devices called Power Measurement Units (PMUs). Prior to the invention of PMU, power system operators have been using voltage magnitude and power measurements taken at RTUs (remote terminal units) to monitor the power system health. System operators had to assume quasi-steady state conditions and had to rely on state estimation techniques to recover system states (voltage phase and angle) from low frequency and poorly synchronized RTU data [3]. PMUs directly measure timestamped voltage and current magnitude and angle. Those measurements are synchronized among different PMUs by means of the Global Positioning Systems (GPS). Each PMU can provide measurements such as three-phase magnitudes and angles for voltage and current with sampling frequencies up to 60Hz. Compared to traditional RTU-based monitoring, PMUs provide better observabilities especially to systems with phenomenal dynamics [4]. Since the invention of the first PMU at Virginia Tech in 1988, PMUs have become a popular option for modernizing power systems – the number of PMUs deployed across North America has increased from about 200 in 2009 to almost 1700 in 2016 [5]. With the introduction of PMUs, new power system applications could be developed, such as power swing monitoring, damping ratio monitoring and island state detection, which have been important features in commercial wide area monitoring tools [6].

The dataset employed in this work comprises years of operation from a fleet of PMUs covering a large part of the continental United States. Therefore, the dataset corresponds to a huge amount of data containing precious information for power grid monitoring and operation which can only be extracted by proper big data processing tools. Costs or constraints associated to computational power for processing the data are relevant aspects of big data analysis. In this paper, available computation infrastructure is limited to a single GPU server. The setup can be considered typical for standard ML development, but it poses a relevant challenge for processing the amount of data available for this work.

The rest of the paper is organized as follows: section II presents details about the data and computational infrastructure employed for processing; data storage, pre-processing and visualization are described in section III; section IV comprises the description of the ML processing pipeline and section V is the conclusion.

II. Big Data and Computational Infrastructure

A. PMU Data

The PMU dataset employed in this work is described in detail below. It consists of almost 20TB of compressed time series data. To the best of the authors' knowledge, the collection of such amount of data in this domain in unprecedented and it was only made possible because of an initiative led by the US Department of Energy (DOE) which integrated data from multiple utility companies. Data was anonymized so that it could be distributed to selected third parties for analysis without revealing the identity of the data providers. In order to evaluate how large this dataset is in the context of ML, it is useful to compare it with typical big datasets employed for deep learning (DL) model development. The English Wikipedia, which is possibly the most usual

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000917.

source of big data for text analytics with over six million articles and three billion words, consists of around 17GB (compressed) [7]. Open Images Dataset [8], containing almost two million annotated images is potentially the largest public dataset for image analytics with total size of 561GB.

Folder structure: The data was provided in 17396 parquet files with snappy compression, each one ~1GB in size. Files were organized into three folders, one for each of three interconnections (ICs), referred to as A, B and C, where each one corresponds to a separate power grid with its own set of PMUs. Table 1 presents more information about each IC, including what actual interconnection it corresponds to, the number of PMUs, date range, data volume and number of files associated to the corresponding datasets. In terms of data points, dataset for IC B alone has over 93 billion records.

Table 1 - Data characteristics for each interconnect

Interconnection	**A**	**B**	**C**
Actual IC	Texas IC	Western IC	Eastern IC
Start Date	2018-07-21	2016-01-01	2016-01-01
End Date	2019-08-24	2017-12-31	2017-12-31
PMU Number	215	43	188
Data Volume	~3TB	~5TB	~11.5TB
File Count	2576	4365	10496

The data for each IC is further organized into folder hierarchy by year, then by month, then by day. The data covers six-week periods followed by omitted two-week periods. The omitted data will be provided by DOE at a future time for testing and validation. Therefore, it must be noted that the complete dataset will be considerably larger than what is described here. Each folder corresponding to a day contains multiple parquet files, each one containing data from all PMUs for a period of that day. The number of such files per folder differs across interconnections and dates.

File Structure: The structure of the parquet files is always the same and consists of 24 columns: *utc*, *vp_m*, *va_m*, *vb_m*, *vc_m*, *vp_a*, *va_a*, *vb_a*, *vc_a*, *ip_m*, *ia_m*, *ib_m*, *ic_m*, *ip_a*, *ia_a*, *ib_a*, *ic_a*, *f*, *df*, *status*, *id*, *interconnect*, *theHour*, *theMinute*. Here letters *v* and *i* indicate voltage and current respectively, while *m* and *a* correspond to magnitude and angle. Letter *p* means positive sequence and *a*, *b* and *c* are phases. Columns *f* and *df* contain frequency and time difference in frequency respectively. Original values for all columns except *status*, *theHour* and *theMinute* are strings, so conversion to numbers is required. Column *status* is discussed in more details below, and columns *theHour* and *theMinute* correspond to the hour and minute values of the timestamp associated to each sample. Current magnitudes were recorded in Amperes, while voltage magnitudes were recorded in Volts. Column *id* refers to PMU id, which is an artificial identifier created to preserve anonymity of data providers. Column *utc* presents timestamps in UTC. Sampling frequency is either 30Hz or 60Hz for each PMU. It must be noted that each row in the original dataset corresponds to measurements from a single PMU. Therefore, processing is also required for synchronizing the data across multiple PMUs.

Column *status* provides PMU status values as defined in IEEE Standard for Synchrophasor Data Transfer for Power Systems [9]. Python library PyMU [10] was adapted to process the available status codes in the data. From each status code, 9 fields were extracted as described in the referenced IEEE Standard: *STAT*, *PMUSYNC*, *SORTING*, *PMUTrigger*, *ConfigChange*, *DataModified*, *TimeQuality*, *UnlockTime* and *TriggerReason*. Based on discussions with domain experts, it was determined that valid measurements should have *STAT* value of *GOOD* and *PMUSYNC* value of *UTCSOURCE*. More details about how this information was employed to filter the data are presented in a later section.

Data Quality: Preliminary analysis of the data indicated various issues that had to be dealt with, including:

- Overlap in time between files
- Duplicate rows
- Unaligned timestamps for different PMUs
- Many missing values or, in some cases, columns which are completely missing for certain PMUs

Pre-processing and data preparation for ML model training must take all those characteristics into account.

Event Log Data: Besides the PMU data, some associated event log information has also been provided to help identify when events took place and what type of event they correspond to. This data was generated from the combination of records from different data providers. Each data provider has its own methods for collecting the logs, which in general involves some level of manual evaluation by domain experts. The combination of the data was also a manual process where the goal was to standardize as much as possible the contributions from all data providers. A complete event log record contains the following information:

- Start and end time of the event
- Category: type of event (e.g. frequency deviation, oscillation) or type of associated equipment (generator, line, transformer, bus)
- Cause: cause of the event, such as: planned service, trip, lightning, equipment
- Descriptors: more details about the event, such as which phases are affected

Information varies significantly among interconnections. For instance, there are only 29 event log records for IC A and only the date is provided, i.e. no time information is given, while IC B has 4854 records and minute resolution for start and end times of most records. IC C has 1884 records with minute resolution for start time but no information about end date/time.

Although the event log data presents invaluable information for identifying relevant power grid events, it must be noted that it is not adequate for direct use as labels for machine learning training. The main factors that justify this claim are:

1. the dataset is generated by multiple levels of manual processing, which is an error prone process.

2. Event categories and causes do not necessarily uniquely map to physical phenomena/patterns, i.e., multiple categories may correspond to the same underlying phenomena and certain categories may correspond to a variety of underlying phenomena. Similarly, reported event durations often cover the consequence of the event, e.g. the resulting power outage, and not the event itself which created the issue.

3. There is no information about which PMUs are affected. Since each interconnect covers a large area, even big events will usually affect only part of the PMUs. Also, the same type of event may happen in multiple locations with different consequences.

Therefore, additional analysis must be performed before this event log data can be employed in training of ML models. As part of the project, we have developed a web application which facilitates the evaluation and annotation of events by a domain expert thereby enabling the use of the log information for training ML models. More information about this topic is presented in section III.

B. Computational Infrastructure

A significant aspect of our work is the constraint of using a single GPU server for processing of the data. This was a project decision aiming to avoid the overhead associated with the creation of a dedicated cluster for the task and the costs of cloud-based computing. The employed server can be considered a typical configuration for a machine used for training of ML models which should be similar to various setups employed across industry and academia. Therefore, discussions about handling big data using such a server can be very useful for the research community.

The specifications for the GPU server employed in this work are described below:

- Processor: Intel® Xeon® Silver 4210, with 40 cores
- 196GB of RAM
- 4 NVIDIA Quadro RTX 6000 GPUs, each with 24GB of RAM
- 2TB NVMe drive for operating system
- 42TB of HDD space for data storage
- Ubuntu 18.04.2 LTS operating system

III. Data Storage, Pre-Processing and Visualization

A. Data storage

In terms of data storage, two options were considered: using a time series database or using the original files directly. Test of a time series database was performed based on influxDB [11] open source version installed as a single node in the server. The associated Python package was used for testing the data ingestion. However, the ingestion of each file took ~13 minutes and therefore more than 5 months would be required for ingestion of the complete dataset. Thus, our choice was to process parquet files directly.

B. Pre-processing

Given the characteristics of the problem, an effort was made to pre-process the files as efficiently as possible while producing as much useful information as possible. Besides adjustments to the data itself, such as type conversion, summary statistics of the data were also calculated during the pre-processing step in order to enable more efficient exploration of its contents to guide the ML model training. As Python was the language of choice for ML model development, it was also employed for the pre-processing steps. The pre-processing pipeline comprised the following operations for each original data file:

1) loading original parquet file
2) converting numerical fields type (originally stored as strings)
3) converting empty values to *null*
4) removing columns which are all *null*
5) converting timestamp type (originally stored as strings)
6) indexing the data based on the timestamp
7) calculating and saving statistics
8) synchronizing the data from all PMUs for each timestamp.
9) sorting by timestamp
10) saving the pre-processed data as a new parquet file

In order to implement such pre-processing, a comparison between the use of CPU and GPU for the task was performed. For the CPU option, parallel processing was employed, in which case the computer memory limited the maximum of the number of files that could be processed simultaneously. Based on testing, this maximum number was identified as 4 and the processing time for 4 files in parallel took ~10min, i.e. ~2.5min/file.

GPUs are commonly used for efficient computation in specialized tasks such as DL model training, but their application in general data processing was traditionally very limited as this required implementation of the processing routines based on CUDA[1] [12]. However, the recent release of RAPIDS cuDF provided adequate tools for this task. cuDF is a Python package that provides an interface that approximates that of the traditional Pandas library [14] but using GPU-based operations. Since such package is still experimental, required functionalities were double-checked against their Pandas counterpart before use. GPU-based pre-processing of the files using cuDF took around 2min/file on a single GPU. Therefore, it was chosen as the option for pre-processing. The main limitation of the GPU-based processing is memory. Files requiring more memory than that available in the GPU, e.g. those for interconnections with a large number of PMUs, were broken into smaller chunks for processing.

Statistics calculation comprised the production of summary statistics for each measurement of each PMU and for the occurrence of the various status codes for each PMU. The

[1] CUDA is specific to NVIDIA GPUs

choice was to aggregate the whole set of statistics for every minute, which was considered a good tradeoff between resolution and efficient handling of data in large time windows for visualization and exploration. The use of this resolution was also facilitated by the existence of *theHour* and *theMinute* fields in the dataset. The following statistics were calculated for each numerical field:

- count
- mean
- min, max
- quantiles: 10%, 25%, 50% (median), 75%, 90%

Concerning the status codes, statistics comprised the count of occurrence of each applicable status code for each PMU at each minute. Information about the time range covered by each file was also compiled in the process. All statistics information was stored in JSON files.

As a consequence of the definition of valid status codes described in section II and also the findings about clear outliers described below in sub-section C, additional data pre-processing was performed so that invalid data could be removed from calculations. Instead of removing invalid measurements directly from the data, the option chosen was to generate a binary mask for each data file, indicating what are valid measurements. This provides more flexibility for dealing with the invalid measurements. Besides, masks can be efficiently applied when preparing the datasets and results in a small cost in terms of disk space, as each mask file is only a few MB in size.

The criteria for defining valid measurements are then the valid status and valid ranges for the measurements defined with support from a domain expert. Values that don't fulfill such requirements are indicated in the corresponding binary masks as invalid. Using IC B dataset as a reference, generation of mask file takes ~11.5s per preprocessed data file using a single GPU.

Invalid status codes correspond to ~4% of the data. Outliers as defined above occur on less than 0.01% of the frequency values and even less on magnitudes and angles. Phase currents are missing in almost 80% of measurements. Other measurements are missing less than 10% of the time.

C. *Data Visualization and Annotation*

Data visualization tools were created for exploring the data and aggregated statistics. A set of functionalities was developed to facilitate the creation of static plots for visualization of selected measurements from selected PMUs. Fig. 1 presents a sample heatmap plot of normalized mean values for positive sequence voltage magnitude for all PMUs in interconnection B during 1 year.

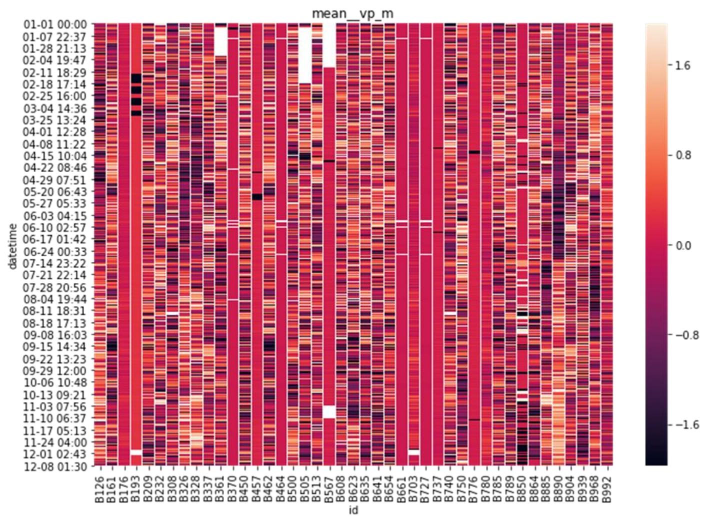
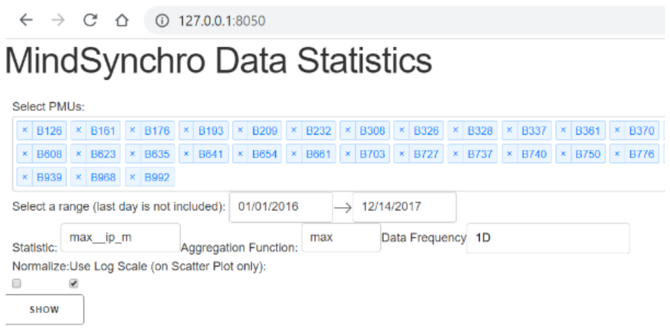


Fig. 1. Heatmap plot of normalized mean values for positive sequence voltage magnitude for all PMUs in interconnect B during one year

Interactive web-based plotting tools were also created since they are more convenient for some tasks, such as exploration and discussions with domain experts, as they enable users to perform operations such as adding/removing data and zooming in and out. We have developed and made use of interactive dashboards using the Dash framework for Python [15].

One of the developed tools enables the creation of aggregated statistics plots based on the following information specified by the user, as presented in Fig. 2:

- A set of PMUs
- A time range
- A statistic for a specific measurement
- Frequency of aggregation and aggregation function
- Options for normalizing the data or applying log transformations.


127.0.0.1:8050

MindSynchro Data Statistics

Select PMUs:

× B126 × B161 × B176 × B193 × B209 × B232 × B308 × B326 × B328 × B337 × B361 × B370
× B608 × B623 × B635 × B641 × B654 × B661 × B703 × B727 × B737 × B740 × B750 × B776
× B939 × B968 × B992

Select a range (last day is not included): 01/01/2016 → 12/14/2017

Statistic: max__ip_m Aggregation Function: max Data Frequency 1D

Normalize: ☐ Use Log Scale (on Scatter Plot only): ☑

Fig. 2. Sensor statistics exploration tool: configuration screen

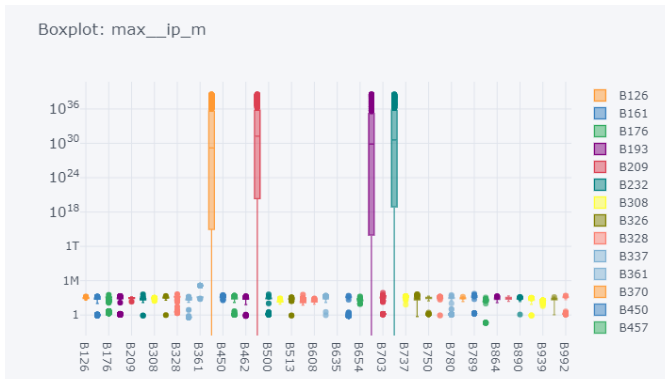


Fig. 3. Sensor statistics exploration tool: sample box plot of the maximum values of positive sequence current magnitudes for a subset of PMUs in interconnect B. Occurrence of outliers is clearly identified for some of the PMUs.

The project team has used those visualization for analyzing the data and discussing it with domain experts.

One example of important findings enabled by the visualization was the existence of clear outlier values. While determining what constitutes *normal* and *abnormal* values for a specific measurement often requires domain expertise and/or statistical analysis, it was possible to immediately identify extreme values on multiple PMUs of interconnect B as presented in Fig. 3. Corresponding values clearly indicate errors in measurement.

Another example of relevant findings based on this analysis correspond to issues on PMU voltage levels. Voltages in the system are supposed to remain within 10% of a nominal value. Rare outliers could indicate anomalies/events. However, it was verified that for some PMUs the nominal voltage value changed over time, as illustrated in Fig. 4. One possible reason for this behavior could be that those PMUs may have been relocated to perform different measurements over time.

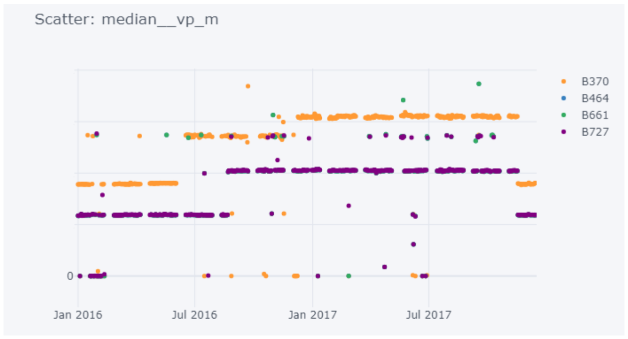


Fig. 4. Evidence of changes in voltage levels for PMUs over time. Plots present voltage magnitudes over the complete time range covered by the dataset. Voltage values are hidden to preserve anonymity of data providers.

The web-based visualization tool has also been converted into an annotation tool to capture information from domain experts about validation of event logs. This enables the proper use of this information as labels for ML training, given the issues discussed in section II. For each event log entry, a domain expert can perform one of the following actions:

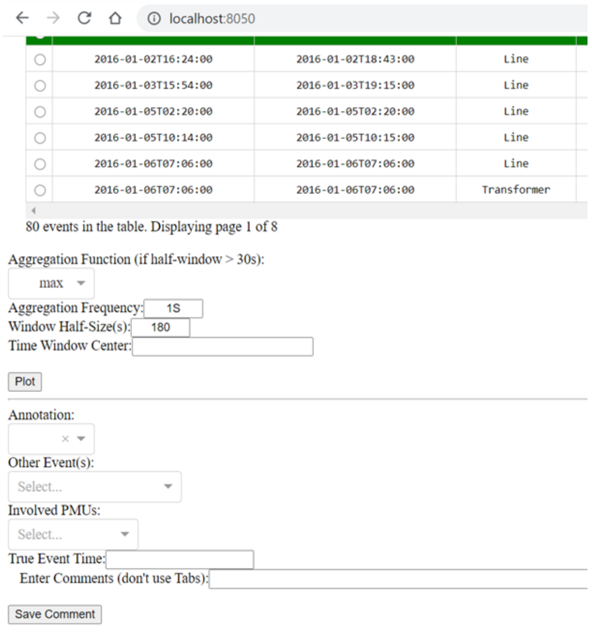
- Confirm presence of a relevant event in the data that matches the event description, map it to a specific physical phenomenon, and provide additional information such as PMUs affected and more precise times
- Confirm presence of an event which is different from what is described in the logs, also mapping it to the corresponding phenomenon and additional information.
- Mark the event as not observed in the data

The list of physical phenomena used for mapping the log events, also defined with support from the domain expert, is: *short circuit*, *trip (no short circuit)*, *line down*, *islanding*, *heavy load*, *low load*, *reactive power shortage*, *transmission corridor congestion*, *power plant controller issue*, *frequency event*, *oscillation event*.

Fig. 5, Fig. 6 and Fig. 7 show the annotation tool. It has the following features/capabilities:

- Presents a list of the event logs for selection and indications of whether or not each one has already been annotated (Fig. 5)
- Selection of various parameters (Fig. 5) for plotting data associated to the event under review. The tool provides plotting of multiple features from a single PMU (Fig. 6) or single feature and multiple PMUs (Fig. 7)
- Annotation fields are used to record the annotation (Fig. 5). They can be adapted to the specific scenario and set of events.
- Backend Efficiency: it is important for the tool to be fast and responsive, to avoid causing frustration to the user and to allow efficient annotation. This is a challenge given large amounts of data needed for labelling events. We use data aggregation and fast access storage of both original and aggregated data based on caching [16] to achieve adequate performance.

We have provided access to this tool to a domain expert from Siemens Digital Grid who annotated a subset of events on IC B for use in ML model training and validation.



○	2016-01-02T16:24:00	2016-01-02T18:43:00	Line
○	2016-01-03T15:54:00	2016-01-03T19:15:00	Line
○	2016-01-05T02:20:00	2016-01-05T02:20:00	Line
○	2016-01-05T10:14:00	2016-01-05T10:15:00	Line
○	2016-01-06T07:06:00	2016-01-06T07:06:00	Line
○	2016-01-06T07:06:00	2016-01-06T07:06:00	Transformer

80 events in the table. Displaying page 1 of 8

Aggregation Function (if half-window > 30s): max

Aggregation Frequency: 1S

Window Half-Size(s): 180

Time Window Center:

Plot

Annotation:

Other Event(s): Select...

Involved PMUs: Select...

True Event Time:

Enter Comments (don't use Tabs):

Save Comment

Fig. 5. Plotting parameters and annotation fields

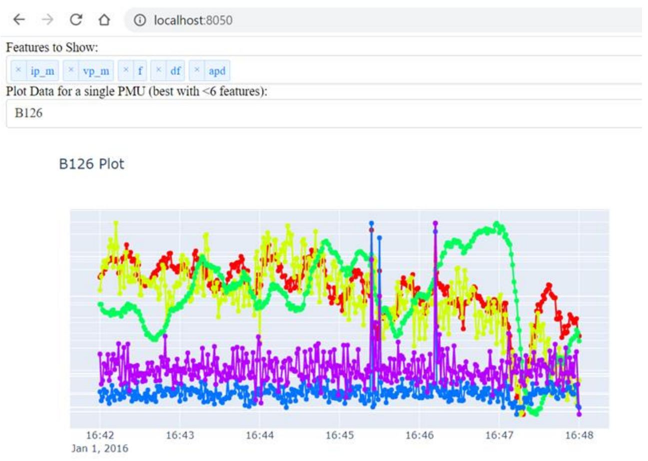


Fig. 6. Individual PMU plot with feature selection

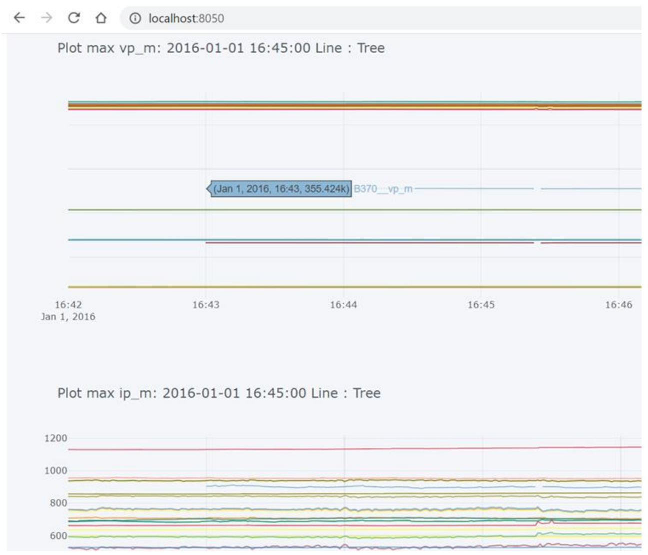

localhost:8050
Plot max vp_m: 2016-01-01 16:45:00 Line : Tree
(Jan 1, 2016, 16:43, 355.424k) B370__vp_m
16:42
Jan 1, 2016
16:43
16:44
16:45
16:46
Plot max ip_m: 2016-01-01 16:45:00 Line : Tree
1200
1000
800
600

Fig. 7. Feature plots for multiple PMUs. Voltage values are hidden to preserve anonymity of data providers.

IV. Machine Learning Development

A. Machine Learning Pipeline

In order to perform training and testing of a variety of ML model types and configurations for such a volume of data, a proper processing pipeline must be developed.

The framework of choice for ML model development was tensorflow 2 [17] which is one of the most popular tools for creation of DL models, leveraging all the processing power of GPUs for the task.

Although all the PMU data had already been pre-processed, it requires additional preparation before it can be used for ML model development. Such preparation is not incorporated into pre-processing as it may differ for different modeling tasks. In order to create an efficient pipeline, data preparation is performed in CPU in parallel to model training in GPU as illustrated in Fig. 8. Each block in the figure corresponds to a processing task employing either CPU or GPU and block widths correspond to processing duration.

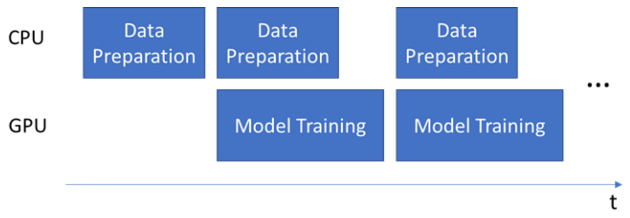


Fig. 8. Schematic of processing pipeline developed for model training where data preparation happens in CPU in parallel to model training in GPU. Block widths correspond to processing durations.

Data preparation consists of a series of processing tasks which transform pre-processed files into batches of data for model training. Developed data preparation consists of the following steps:

1) load parquet metadata from pre-processed file and corresponding valid data mask
2) filter columns to specific PMUs if needed
3) load corresponding columns for data and mask
4) apply valid data mask to the data
5) eliminate regions of overlap with other files
6) unwrap phase angles (optional)
7) fill missing columns with token value
8) resample to 30Hz
9) forward fill and then back fill missing values
10) eliminate extra rows that won't fit in batch
11) normalize values based on pre-defined ranges for each measurement
12) reshape data to batch format
13) adjust all angle values to the same reference in each batch (optional)
14) replace token value from missing columns with zeros

Unwrapping in step 6 is performed to eliminate the abrupt changes in phase angles as they are originally represented in the range between -180 and 180 degrees. Adjustment of phase angle values to the same reference as presented in step 13 is performed because the most important information from the phase is not in its absolute value but in the phase difference among different measurements.

Resulting batch consists of a tensor which is a collection of samples, each sample comprising a time window of measurements. Figure 9 presents a heatmap depicting one such sample which consists of a 2D array with dimensions 300x18, corresponding respectively to time (10s of data at 30Hz) and the 18 measurements. One of the main challenges corresponds to dealing with missing data. Whole measurements may be missing for each PMU in which case they are replaced with zeros. Values are normalized to pre-defined ranges. Data preparation is performed in CPU as described above. With no parallel processing, it takes from ~0.4s to ~2.5s to convert all data from one PMU in one file into batches, depending on how many PMUs are considered on the same file.

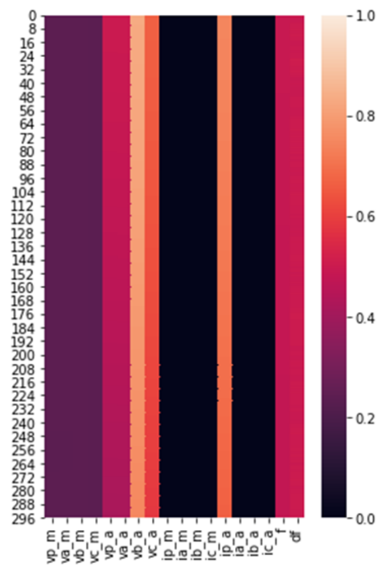


Fig. 9. Heatmap presenting a sample for model training resulting from data preparation. It corresponds to 300 seconds (y-axis) of normalized values for all measurements (x-axis) from one PMU.

The processing pipeline also comprises facilitated means for definition of multiple model architectures and configurations related to data preparation and training, so that multiple options can be easily tried for hyperparameter tuning. Each such option can be created as a folder containing two files: a Python script defining the model architecture based on tensorflow and a configuration file which defines parameters for data preparation, such as:

- files/data to use for training
- time window length
- options such as angle corrections and randomization of files
- values to use for normalization

And hyperparameters and options related to model training, such as:

- number of epochs
- use of early stopping or regularization
- validation split
- method specific hyperparameters such as dimensionality of the latent space in autoencoder

B. ML Models and Sample Results

In order to create models for relevant power grid event detection based on the available PMU data, a deep semi-supervised learning approach is employed [18]. Based on such approach, the large amount of unlabeled data can be combined with a reduced set of labeled data to potentially achieve performance in the classification tasks which is superior to that achieved using the labeled data alone. One approach employed in this work is based on an autoencoder architecture as depicted in Fig. 10. The standard autoencoder architecture comprises the encoder and decoder presented in the figure, where the deep neural network (DNN) is trained to produce an estimate of its own input *X*, which in this case corresponds to a multivariate time window of fixed length as presented in Fig. 9. In our work, various approaches have been tested where encoder and decoder are created using convolutional neural networks (CNN) or long short-term memory (LSTM) networks. The autoencoder presents a bottleneck which is a latent space, also referred to as code, which has reduced dimensionality compared to *X*. Once trained, the encoder can produce such code for new input samples. The difference between the standard autoencoder and the semi-supervised architecture employed in this work is in the classification head, as presented in Fig. 10. This classification head comprises additional DNN layers which receive as input the code and provide as output an estimate of the class to which the input sample belongs to. In this case, classes are associated to normal operating conditions and different types of anomalous events. Training of the standard autoencoder (encoder and decoder) is unsupervised, therefore it can be performed using all the big data from PMU measurements so that the DNN can learn to produce the best code to represent the information in the samples. The classifier, which includes the classification head as well as the encoder, is trained using the reduced set of labeled data containing ground truth information about sample classes.

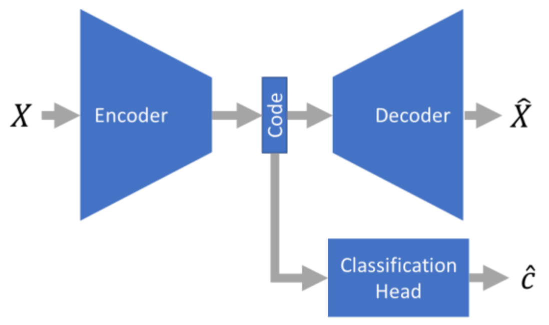


Fig. 10. Semi-supervised autoencoder architecture.

Here we describe a sample experiment employing a CNN-based semi-supervised autoencoder. A dataset consisting of 370143 samples produced by applying the data preparation process described above is used for the unsupervised learning task. These samples correspond to a randomly selected subset of the complete dataset from 24 IC B PMUs covering 2 years of operation. Data labeled by a domain expert corresponding to the occurrence of short circuit events are used to train the classifier. The labeled dataset contains 221 labels in total of which 29 correspond to the occurrence of short circuits. Twenty percent of the samples stratified by class (*short circuit* or *no short circuit*) are used as hold-out data for validation and calculation of performance metrics. All training was performed in a single GPU and each training epoch consisted of one step of autoencoder training, taking ~40s to complete, and multiple steps of classifier training taking ~0.3s each. This approach resulted in a perfect result with 100% accuracy.

V. Conclusion

In this paper we have presented the application of big data processing for the development of ML models for detection of relevant events in power grid operations. All data processing of the almost 20TB of phasor measurement unit data has been performed using a single GPU server, which is a typical configuration for ML development. Despite the many challenges faced due to the large data volume and issues normally associated to real world data, we have successfully employed GPUs to scale the processing power to enable the development of ML models, not only on the model training and testing tasks but throughout the whole data processing pipeline.

This is part of an ongoing project, and the main tasks planned for future work include: exploration of other ML approaches, introduction of physics-based features in ML model development, application of weak supervision approaches for a more scalable extraction of labels from event logs, analysis of complete dataset of all interconnects including additional test data which will be provided by DOE at a future time.

Acknowledgment and Disclaimer

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000917. This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

References

[1] B. P. Bhattarai, et al. "Big data analytics in smart grids: state-of-the-art, challenges, opportunities, and future directions." IET Smart Grid, vol. 2, no. 2, pp. 141-154, 2019

[2] A. Shahsavari, M. Farajollahi, E. M. Stewart, E. Cortez and H. Mohsenian-Rad, "Situational Awareness in Distribution Grid Using Micro-PMU Data: A Machine Learning Approach," in IEEE Transactions on Smart Grid, vol. 10, no. 6, pp. 6167-6177, Nov. 2019, doi: 10.1109/TSG.2019.2898676.

[3] Hug, Gabriela, and Joseph Andrew Giampapa. "Vulnerability assessment of AC state estimation with respect to false data injection cyber-attacks." IEEE Transactions on smart grid 3, no. 3 (2012): 1362-1370.

[4] Almasabi, Saleh, and Joydeep Mitra. "An overview of synchrophasors and their applications in smart grids." In 2016 International Conference on Intelligent Control Power and Instrumentation (ICICPI), pp. 179-183. IEEE, 2016.

[5] North America SynchroPhasor Initiative, "Synchrophasor technology fact sheet", available at: https://www.naspi.org/sites/default/files/reference_documents/33.pdf?fileID=1326

[6] Siemens Energy, SIGUARD® PDP, available at: https://new.siemens.com/global/en/products/energy/energy-automation-and-smart-grid/power-quality-measurement/grid-monitoring-using-synchrophasors-siguard-pdp.html

[7] Wikipedia: Size of Wikipedia, available at: https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia

[8] Open Images Dataset, available at: https://storage.googleapis.com/openimages/web/index.html

[9] IEEE Standard for Synchrophasor Data Transfer for Power Systems, in IEEE Std C37.118.2-2011 (Revision of IEEE Std C37.118-2005) , pp.1-53, 28 Dec. 2011, doi: 10.1109/IEEESTD.2011.6111222.

[10] PyMU documentation, available at: https://pythonhosted.org/PyMU/

[11] Influxdata. InfluxDB, available at: https://www.influxdata.com/products/influxdb-overview/

[12] NVIDIA. About CUDA, available at: https://developer.nvidia.com/about-cuda

[13] RAPIDS. cuDF documentation, available at: https://docs.rapids.ai/api/cudf/stable/

[14] The pandas development team. Pandas, available at: https://pandas.pydata.org/

[15] Dash framework, available at: https://dash.plotly.com/

[16] Flask-Caching, available at: https://pythonhosted.org/Flask-Caching/

[17] TensorFlow, available at: https://www.tensorflow.org/

[18] Y. Ouali et al. An overview of deep semi-supervised learning. 2020. Available at: https://arxiv.org/abs/2006.05278