GridAI: Cloud-Based Machine/Deep Learning for Power Grid Data Analytics

FINAL REPORT

sdmay21-23
Client/Advisor: Dr. Gelli Ravikumar

Team Members:
Karthik Prakash
Abir Mojumder
Abhilash Tripathy
Justin Merkel
Patrick Wenzel

sdmay21-23@iastate.edu
sdmay21-23.sd.ece.iastate.edu

Table of Contents

List of figures/tables/symbols/definitions

Figures:

Tables:

Acronyms/Terms

Acronym/Term	Meaning
ML	Machine Learning
DNN	Deep Neural Network
kWh	Kilowatt hour
MAE	Mean Absolute Error
MSE	Mean Square Error

Table 1: Abbreviations

1 Introduction

1.1 Acknowledgement

Special thanks to Dr. Gelli Ravikumar for providing us with the opportunity to work with him on this project, providing us with resources and ideas to complete this project, and for always being available to answer questions when needed.

We would also like to thank the research students for providing us the PowerCyber Testbed so that we could smoothly develop and test our application.

1.2 Problem and Project Statement

General Problem Statement

Power Grids serve several homes and businesses and are complex in architecture which can make them vulnerable to instabilities and unusual behavior. Our project aims to use the collected power grid data to provide analytics and insights into anomalies and provide more meaningful information about power usage.

General Solution Approach

To resolve the issue, our team will develop a web application that implements a Machine Learning algorithm to analyze the power grid data and display these analytics in the form of graphs and plots. The application will be deployed on a remote platform so that it can run continuously and provide the visuals to the Dashboard that a power grid operator can access. The goal of our application is to predict anomalies within power grid nodes at different locations to enable prompt response and to minimize potential outages.

1.3 Operational Environment

The entire application is designed to run on a remote server. Each functional component (i.e. Front-end, Back-end, and Databases) is contained within its own Docker container . Users will interact with the project by accessing the client-side application through a web browser. There will be no extreme or hazardous conditions that will affect this project.

1.4 Intended Users and Uses

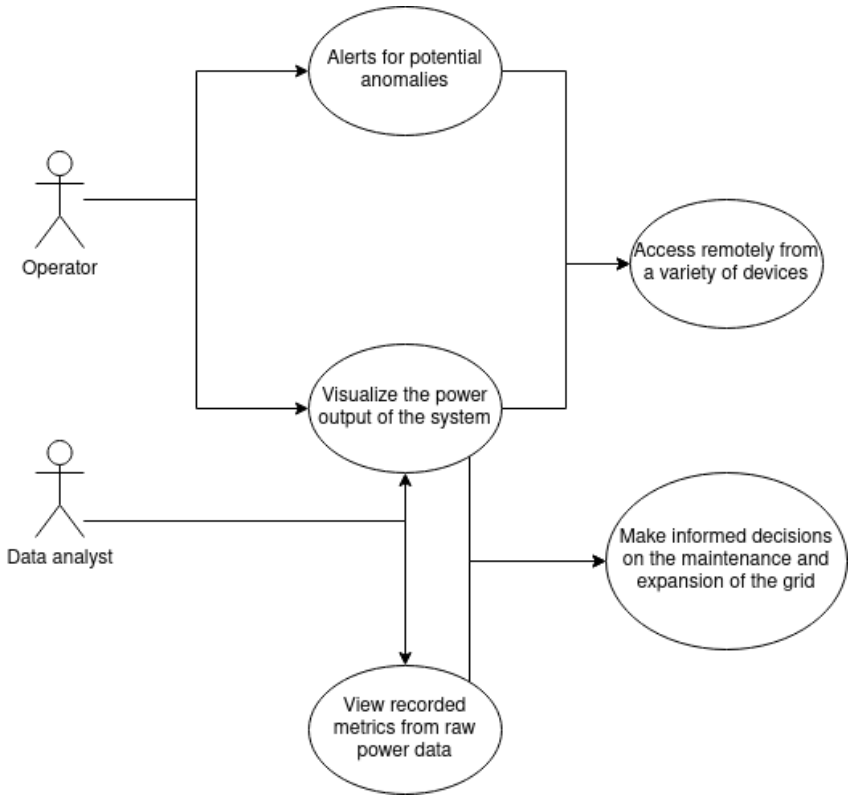

Figure 1: Use Case Diagram

From the figure above, the intended audience for this project is power grid managers/operators who will need to know real time analytics for the power grid. The application will provide them with visualizations of the power grid nodes and alerts if anomalies are present. The Dockerization of our project will also allow power grid operators to deploy our application regardless of the system specifications. Additionally, data analysts will want to view power grid data over long periods of time to make informed decisions on the power grid.

1.5 Assumptions and Limitations

Assumptions:

- Power grid data supplied to us for training is free of anomalies.
- Users will be able to provide clean, formatted data.

2 Design

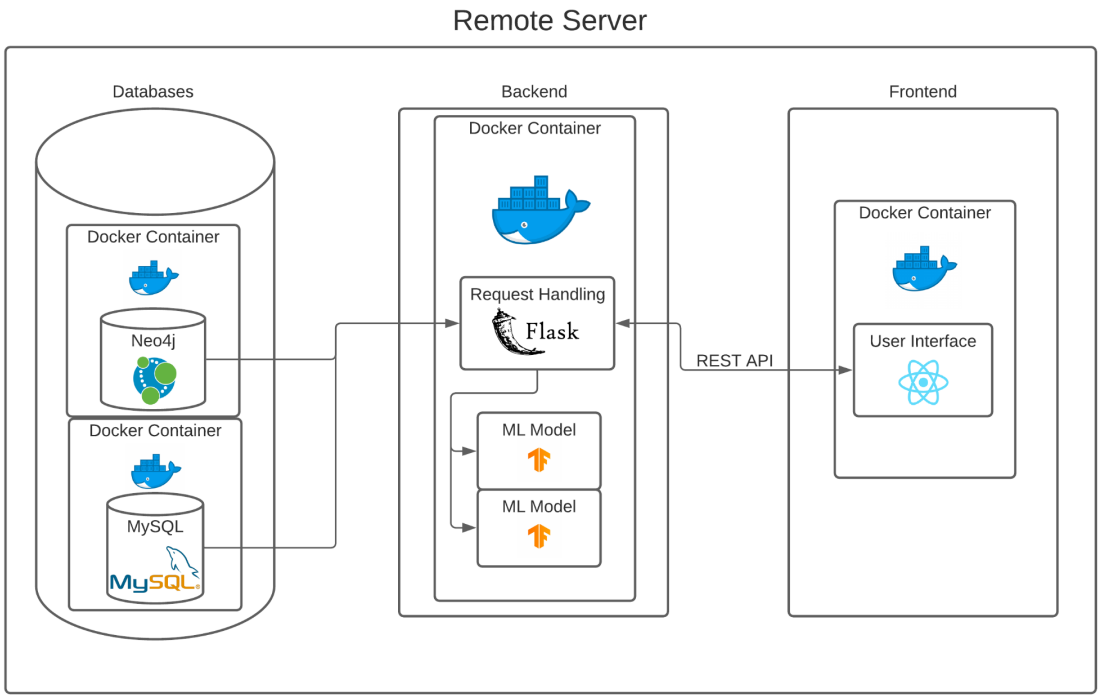

Figure 2: Design Architecture Diagram

2.1 Functional Requirements

- Machine Learning Requirements:
  - The ML models must use the most recent kWh value from the grid in the predictions.
  - The ML models must predict the next kWh output for each node in the grid.
  - The ML models must predict the probability of each anomaly class.
  - The ML models must use convolutional layers for deep learning.
- Front-end Requirements:
  - The front-end should receive data from the back-end
  - The front-end should visualize data
    - The front-end should interface directly with the back-end
    - The front-end should have:
      - Graph-based visualization
      - Geographical representation of the power grid
      - Charts for each node's history and predictions
      - Tabular data showing anomaly status for every node
- Back-end Requirements:
  - The server-side application will handle all data communication with the databases
  - All data processing, including ML analysis, will occur on the back-end
  - Provide real-time data to front-end

2.2 Non-Functional Requirements

- Clear documentation
    - Allows future teams to improve on the baseline
- Maintainability
    - Modular coding and Docker containers
- Scalability
    - ML models are generalized predictors for nodes.
- Response time
    - Lightweight front-end to accommodate response rate of work heavy back-end

2.3 Constraints

- Only have 1 year's worth of real data and the rest has to be simulated

2.4 Engineering Standards

- IEEE/ISO/IEC 12207-2017: Software life cycle processes

Our team aimed to be compliant with IEEE Standard 12207-2017. We chose to focus on the Technical Processes as defined in IEEE Standard 12207-2017. Specifically, the customer and ourselves conducted the Stakeholder Needs and Requirements Definition process, System/Software Requirements Definition process, Architecture Definition process, Design Definition process, Implementation process, and Integration process. From the beginning of our project definition, we have conducted weekly meetings with our customer. The needs and requirements as desired by our customer were outlined and slowly evolved into the system requirements and design described in this document. Since the requirements and functionality were not explicitly clear to us from the beginning, we chose to structure our software life cycle with an incremental development model. This way our team could employ an Agile development methodology to tweak the requirements throughout the development cycle. The foundational architecture of our Software System of Interest stayed relatively the same, but this incremental development model proved to be beneficial since our design and technology requirements changed slightly as stated below in Section 2.6. During the Implementation Process, we utilized project tracking and team communication applications to assign functional tasks. By continuing our weekly meetings with our customer, we ensured that the system was developing inline with his needs. The Integration Process was completed in concurrence with the Implementation Process. As functional modules were developed and deemed to be in an acceptable state they would be connected with the necessary systems.

2.5 Security Concerns and Countermeasures

Our biggest concern with this project would be some type of external data manipulation. That is why we identified our databases as the most likely target for an attack. However, with this in mind, direct database manipulation would be difficult if users deploy the application with simple security measures. To directly access the database, attackers would need to gain access to the host system and have the authentication necessary to access the databases within their respective Docker containers.

If, instead of a direct attack, the malicious entity attempted to backdoor the system with a type of SQL injection, our application should show few vulnerabilities. Majority of our endpoints do not support any user-defined parameters. The few that do would be difficult to manipulate into SQL attacks as erroneous user input would most likely result in an invalid query. We acknowledge that the security of the application would benefit greatly by looking into remote hosting security measures and implementing an authentication interface.

2.6 Design Changes Since 491

In our original design, we pictured separate Docker containers for the REST API and the machine learning models. Our initial thinking was to configure an endpoint that would pass the necessary grid data to the Tensorflow Docker container running a machine learning application waiting for a feature matrix. However, after gaining more knowledge on how the two modules would interface, we chose to integrate both within one Docker container. This specific Docker container contains both the required Flask dependencies and Tensorflow dependencies. Therefore, the ML models, which are trained and saved externally, are stored in dedicated folders for the API to use with stock Keras function calls. This design decision does not change the modularity of our project since models can easily be replaced or added with minimal change to code.

Additionally, when we were first narrowing the technologies that were to be implemented, we identified Redis as a database instance to be used for storing some type of short-term data, such as prediction data from our ML models. Through the development phase, the team chose to replace the Redis instance with a MySQL database instance. MySQL provides much more functionality to our project. We store all of the grid time-series data in a single table within the MySQL instance. While this would also be possible with a Redis instance, the cache-based infrastructure of Redis conflicts with our desire to keep the project scalable. If future users want to store a longer period's worth of data or utilize the application for a larger grid network, Redis could pose a potential bottleneck.

As outlined in Appendix B.1, the ML algorithms were just conceptually what they should accomplish. As the understanding of ML progressed the original design was scrapped for two types of ML models that meet our requirements. One type is the DNN regression model that uses linear regression with convolutional layers to predict more precisely. The other is used to meet the requirements of anomaly classification. Logistic regression using softmax final layer allows us to use convolution to classify any transformer output to be of a given class (normal, power spike, power failure) with an associated probability.

2.7 Front-end Design

The front-end was designed to have a clean-looking and easy to use interface. In order to achieve this, we built the front-end on the open source "Material Dashboard React" by Creative Tim. This allowed us to save time in both the design and implementation of the frontend. We then just had to learn how the files were used for rendering, how they were routed, and what files and code we could do without and we were able to scrap everything we didn't need to make room for our data visualizations. For our data visualizations, we included a geographical representation of the power grid where you can click on the nodes to get its most recent output as well as seeing the node's kWh over time on a graph. You can also check a box and click on another node to compare the two nodes' output on the same graph. We also have charts for the kWh history and prediction for each node as well as a table showing the anomaly status for each node. These are all on their own separate pages to keep our interface clean-looking.

2.8 Back-end Design

The back-end was designed with simplicity in mind in order to keep the server-side application relatively lightweight so that it could support quick response times. Therefore, we chose to implement the REST API with the Flask framework because of its minimal resource requirements and relatively simple learning curve. For our databases, we implemented Neo4j and MySQL. Neo4j is a NoSQL graph database. Despite it being a NoSQL database, Neo4j still supplies exceptional support for relationship functionality. These properties make it an ideal choice for our power

grid-based project since it enables fast query times and practical representation of the data. To assist the Neo4j instance, the team also chose to use a MySQL database instance to assist with our data handling. MySQL is a well-known relational database service that is extremely scalable. The main reason we chose to implement a MySQL instance is because of its scalability. While we could potentially store the grid time-series data in a Neo4j database, that solution may not be scalable if future users desired to use the application for larger networks or a larger dataset.

2.9 Machine Learning Design

As stated in Section 3.1, the ML models must predict and classify which requires two types of models. One is used in the anomaly classification and the other is used in predicting a continuous value representing the future kWh output. This implementation will be split into two subsections covering each model.

2.9.1 DNN Regression

From Appendix C.1, there are 3 repeatable steps involved in ML. The first being representation. The feature vector that represents the information at each node will include the static node information of the transformer type, the previous and most recent kWh output, and the kWh from the previous node in the chain. These attributes were chosen because they represent a lot of the information needed to understand the context of each node temporally, locality, and statically.

For the second step we must choose a loss function to evaluate the model. For this we decided upon using MAE vs MSE. Both are popular loss functions with their own set of pros and cons. Since we wanted to focus more on the scalability and overall generalizability of our models the MAE was chosen. This is because MAE is more robust in dealing with outliers and prevents the overfitting of the DNN model.

The final step is optimization which is essentially the training parameters that determine how we traverse the gradient of the loss function. For our purposes these are not specifically designed for as during training time the hyperparameters can be optimized using a validation set.

2.9.2 Logistic Regression

The difference between DNN of 3.7.1 and Logistic Regression is the minor representation difference and loss function. We have identified 3 different classes of data. One is the normal data, the second is a power spike where the output exceeds twice what we expect, and a power failure where the power output drops dramatically. To accomplish this we need to use a softmax function.

$$\sigma(\mathbf{z})_i = \frac{e^{-\beta z_i}}{\sum_{j=1}^{K} e^{-\beta z_j}} \text{ for } i = 1, \ldots, K.$$

Figure 3: Softmax equation

This function converts K inputs into K outputs that sum to 1 which will be used for probability. For our purpose, K will be 3 (one for each class). This softmax function is the final layer of the DNN Logistic model.

The difference is loss function is now sparse categorical cross entropy error. The reason we are using sparse is for simplicity so that the labels do not have to be one-hot encoded (there is no

functional difference) and the training time is faster. Cross entropy is the main loss function for logistic regression because it relates to the probability of predicting the correct classification.

3 Implementation

3.1 Front-end

The front-end for the web application uses the React library to visualize the User Interface. We have used the Material-UI framework's premade template as a basis for the design. We only needed to use the page routing, navigation bar and dynamic table that is provided by the library, as the various other tools were not necessary for our application. To visualize the 240-node power grid, we used the "react-d3-graph" library which allows us to create an interactive 240 node representative grid. The power grid schematic has relative coordinates for each of the nodes which is stored in a database as a static property of the node (Explained further in section 3.2). By fetching the coordinates, the D3 graph can position the nodes accordingly, along with labeling them with the appropriate transformer codes.

Now that we can see the nodes, a requirement is to show the current and previous kWh values for the chosen node. Through further REST API calls, the current value of the node is retrieved and displayed as a pop-up. The 24 hour history of values for this node is to be represented in a line chart. The React Google Charts library provides a simple looking line chart that we will use to display the past 24 hour time-series data. The final point of data within the chart is a Predicted value for the next hour. The predicted data is generated by the machine learning algorithm (Explained in section 3.3) which is then fetched from the backend through a REST API call and added to the line chart prior to rendering. There is also a feature to compare other nodes' history data by selecting the checkbox option that enables the user to do so. This overlaps the two line data in the chart that lets you see values for each hour and the overall pattern of changes in kWh.

The final requirement is to determine if any of the transformers have an anomaly in their power outputs. There are 3 possible anomaly types: Normal, Spike, Failure. A page showing anomaly statuses for every node in a tabular format is used for this purpose. The anomaly status is fetched from the backend and inserted into the Material-UI's provided table that has been modified to capture header and row information from the data array.

3.2 Back-end

The structure of our back-end consists of three main components: the Neo4j database, the MySQL database, and the REST API. The Neo4j database is intended to store any static information necessary for a given transformer in the grid network. These properties include the real and reactive power impedance ratios, voltage and current ratings, relative coordinates, and the previous bus in the network. Along with that, the nodes store the most recent kWh value reported from the node, the time that the node was updated, the kWh value reported previously, and the kWh output of the previous bus in the grid. Since Neo4j is a graph-based database, structuring each node as a transformer seems like the most practical and easy-to-understand method for grid representation.

Then, the MySQL database has one table to store the necessary time-series data from the power grid simulation. The table holds one year's worth of data from each of the busses in the grid since that was the dataset with which we were provided. The busses report values hourly. The date

and time of the reported values are used as the primary keys for the table with columns specific to each bus making it simple to query for a specific transformer or specific time interval.

Finally, our REST API provides the connection for the frontend to access the relevant data. The service is developed with the Flask framework. Our team developed with a basic RESTful methodology in mind. For that reason, we tried to keep the number of endpoints minimal. Endpoints were only created if there was a specific need for certain data. Each endpoint has some combination of Cypher and/or SQL queries to be executed by the respective driver connection for the Neo4j and MySQl database. In addition to basic data querying, URLs were instantiated for performing the ML predictions and anomaly detections. As mentioned earlier in Section 2, the saved ML models are stored in separate folders and loaded into their respective objects at startup. Aside from the URL mappings and functionality, the server-side application also configures a background scheduler to update the properties in the Neo4j nodes at the top of every hour. The task pulls the time series data from the MySQL table and inserts the necessary data into the respective nodes in Neo4j using a combination of Cyper and SQL queries. This gives our application its real-time functionality.

3.3 Machine Learning

The ML models are implemented using the Tensorflow Keras high-level API. This was originally chosen because it allows us to specify convolutional layers and model structures without fully understanding the underlying principles of the algorithms. An additional consideration that is not listed in the design section is that there has to be 3 separate models for each type of model. This is due to the fact that the transformers themselves have different static data depending on the type of transformer.

The DNN regression model uses multiple fully connected relu layers and a convolutional relu layer. Figure 3 is provided to help the visualization of these layers. The input feature vector is then convoluted to an arbitrary amount of nodes.

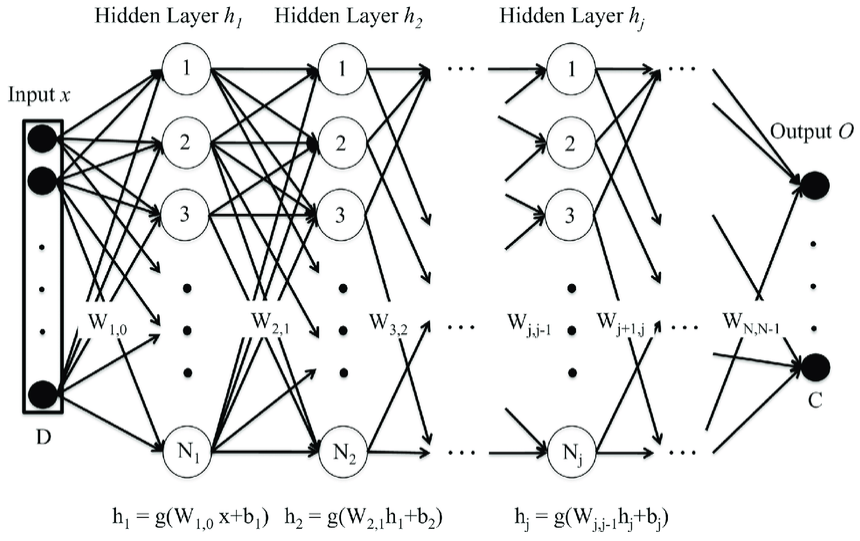

Figure 4: Visualization of DNN

These nodes are then further modified by the following layers until they are pooled into the output layer which in our case is the future kWh output. These inner layers are trained using gradient descent and backpropagation using the loss function.

The implementation of the Logistic Regression is similar to the DNN. The model uses the same kind of convolution process except for the final layer which is the softmax and will output 3 different probabilities corresponding to the chance the data is of the given class. Figure 4 is showing how this process would look like.

The ML model which is represented by the Neural Net layers and softmax take in an example from the testing dataset that is a power spike anomaly shown by the 1 in the anomaly field. The rest of the information is put into the ML model and the output is 3 values in a vector. the first value represents the chance this is normal data, the second represents the chance the data is a power spike, and the third represents the chance the data is a power failure. From the output we see that this datapoint is correctly classified as a power spike with 99.999% chance which is what we expect. The vector is typically interpreted by using the max value as the class. This means this would be classified as a power spike since the largest (MAX) value is the 99.999%.

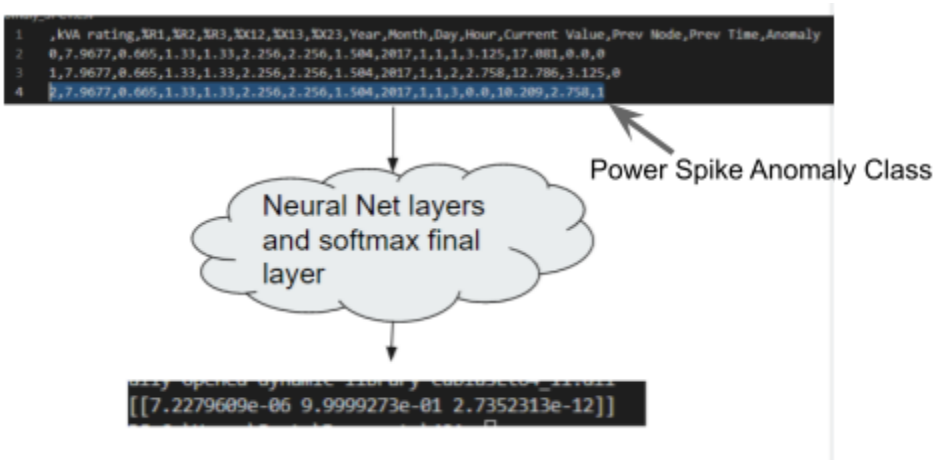

Figure 5: Logistic Regression Example

4 Testing

Since our application relies heavily on accurate data representation, we needed to make sure that our data was being handled correctly at each stage of the process from it going through the ML models to it being displayed on the front-end. Described below is how we did our testing.

4.1 Unit Testing

- Machine/Deep learning algorithms

The ML models are tested by first setting aside 20% of the total data to be used as testing. This data is not fed to the model prior to the testing process. This is because the purpose of the model is to try to generalize the predictions to new unseen data from the same population. This means that we can test the metrics of the models by using that same testing data which will help evaluate if the models are improving.

- Frontend functions

The React application can be easily tested within the demo mode. Changes made in the code can be compiled instantly to view the dashboard before sending it to production. In addition, our team can use hard-coded values to verify that data is represented as we expect. By doing this, we can narrow the source of other potential errors when completing the system integration.

- Endpoint functions

Our team used Postman to send HTTP requests to our backend application. By utilizing Postman and checking the Docker container logs, we were able to verify that the server-side application was running and capable of receiving requests. Through this method we could also see what data the application was sending to confirm it was as expected.

4.2 Interface Testing

- SQL queries -> REST API -> Machine/Deep learning algorithms -> Front-end functions -> Frontend design

To test the interconnection between the modules we tested each component individually then compared that output to the output obtained when the systems were integrated. For example, we tested queries directly in our databases and compared that output to that received from an API call configured to execute the same query.

In the same way, the Deep Learning models were tested separately and then with the integration into the REST API. Again, Postman was utilized to compare the raw output from the DL models and that received from the API function calls.

Finally, we compared the output from the API function calls to the data that was being received on the front-end. Using the web console, we were able to see the data being received by our React application, so we could confirm that the data was consistent even after being parsed for the dashboard display.

4.3 Acceptance Testing

We inspected our frontend application and verified that our graphs are displaying correct data, that there are no bugs present, and that the interface is acceptable. We involved our client by having him take a look at our application, our backend functionality, and had him look over both things to make sure they are functioning properly. Since we met with our client on a weekly basis, he was able to see how the application evolves incrementally.

Incremental acceptance testing by the client meets the IEEE Standard for Software Verification and Validation.

The verification process checks if standard programming practices (Agile/Scrum) and conventions (Testing code after implementing a module) are met. Life cycle activities are checked for consistency, accuracy, and correctness. If each life cycle activity is verified successfully, the next activity can be initiated.

The validation process tries to prove that the product works by checking if user needs in a real-world environment are satisfied and that the product is ready to be deployed.

4.4 Testing Results

Through testing we were able to confirm that most of the functionalities work as intended. One issue is that the server-side application is not able to receive requests instantly after startup. This is because the API attempts to establish connections with the databases on startup, but the MySQL and Neo4j take a short time to actually initialize. This poses a small bottleneck at launch of the application, but such complications have not occurred past this phase.

The results of the testing data on the Logistic Regression ML models give a very small loss of 0.1167 for the cross-entropy error and an accuracy of 96.27% correct classification. This exceeds our expectations for the model to have a correct classification rate of around 80%. Additionally, the loss of the DNN prediction models is only 1.25 which means on average the predictions are only slightly off the actual kWh by 1.25 kWh. This is significantly better than the standard deviation of the data which is around 9.9 kWh.

Tests performed on the frontend UI produced errors that ranged from missing data due to React's proxy settings (A proxy to the Flask API port allows rest api calls) being misconfigured, typos in Rest API calls, application crashing due to undefined data from clicking nodes that don't have a current reading/output, re-rendering failures and null errors when trying to compare a node's data without initially selecting a node. The configuration issue was solved by simply setting the proxy port correctly. Program crashes from null and undefined values were fixed by checking the data's integrity before attempting to manipulate or display them and by also notifying the user through pop-ups to select a primary node before trying to compare with another node. React Render element issues were solved by correctly updating the program state as per the React lifecycle documentation.

5 Closing Material

5.1 Related products and Literature

One related work is Signal_Processing_ML_Power_Grids. This project used signal processing and machine learning analysis techniques to analyze signal data sent from power grid systems and detect if there are any anomalies like signals that generate partial discharges and lead to faults or power outages. Our product is similar in that we also have anomaly detection but ours is an actual application that can be continuously used while this was a project and only the notebooks are uploaded.

GridAPPS-D is an open-source software platform sponsored by the Department of Energy that has node distribution visualization with simulation of capacitors and regulators. It also produces graphs based on the power grid distribution system simulation. Users can force anomalies to be visualized in the graphs panel. This application provides the features our application has as well as more like model validation, Volt-Var optimization, traffic flow analysis, and more.

5.2 References

Lim, Ryan., Kin (2019) Singal_Processing_ML_Power_Grids

[Source Code] https://github.com/rlim1812/Signal_Processing_ML_Power_Grids

tonya1., poorva1209., craig8., blthayer., afisher1., (2020) gridappsd-python (Version 2020.08.0) [Source Code] https://github.com/GRIDAPPSD/gridappsd-python

Appendix A Operation Manual

The homepage of the GridAI application displays the network of power consuming nodes. This network also depicts the physical distance between the nodes and is placed on our display grid according to each node's respective physical coordinates. The text under each node signifies the name of that node. On the left of the homepage is a navigation bar for viewing data in formats other than the grid. This following section is an instruction manual/guide to our user-interface for the GridAI application:

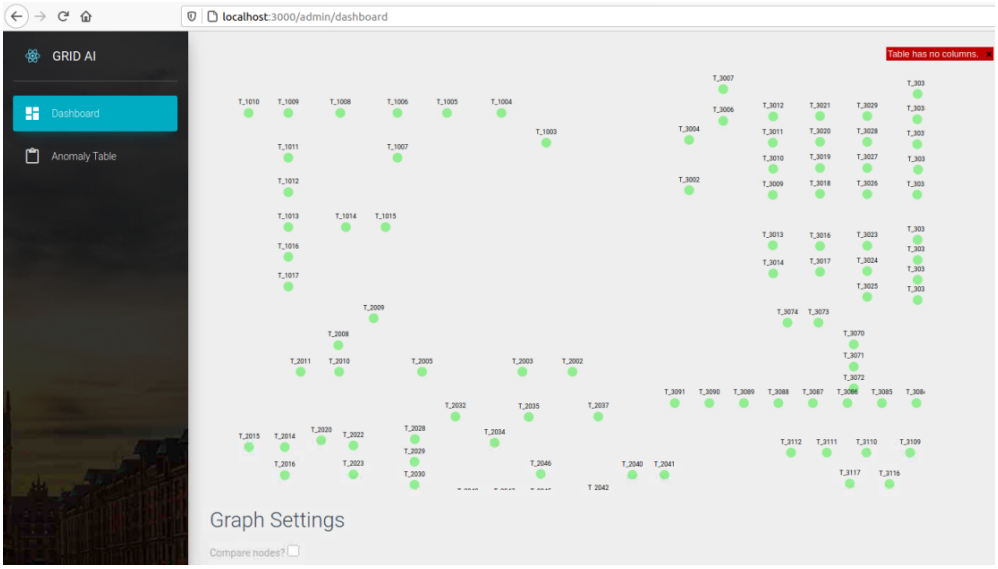

Figure 6: Geographical Representation of Power Grid

1. The grid (use and behavior):
    a. Zoom in/ zoom out into a cluster of nodes using the mouse scroller.
    b. Drag in the natural direction (when zoomed in) to see other nodes.
    c. Click on any node, a dialog box shows up which contains the current value of that node. To close the dialog box, click OK.

Figure 7: Output from Clicking on Grid Node

d. After closing the dialog box, a graph depicting the history of the clicked is displayed on the right.

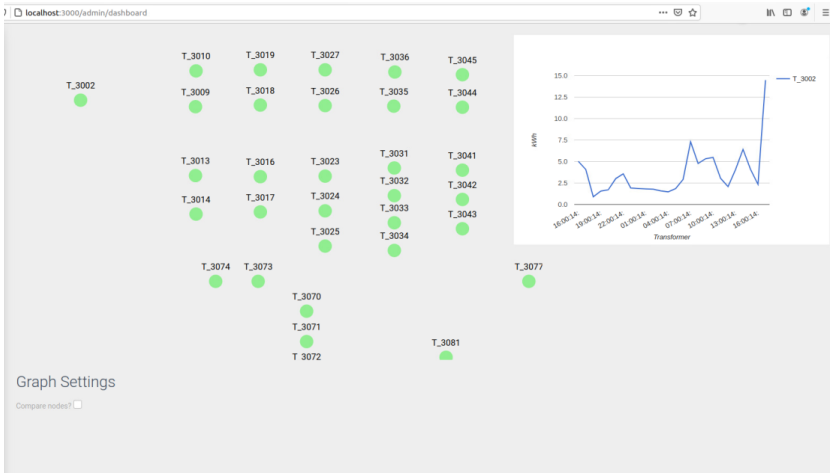

Figure 8: Graph of Clicked Node's Output History and Predictions

2. The Chart
   a. This graph shows the kWh consumption of the last 24 hours as well as the predicted data calculated using our ML models.
   b. To obtain the data of a single hour, simply hover over the line graph on a particular hour and the label will display the kWh value of the corresponding hour.

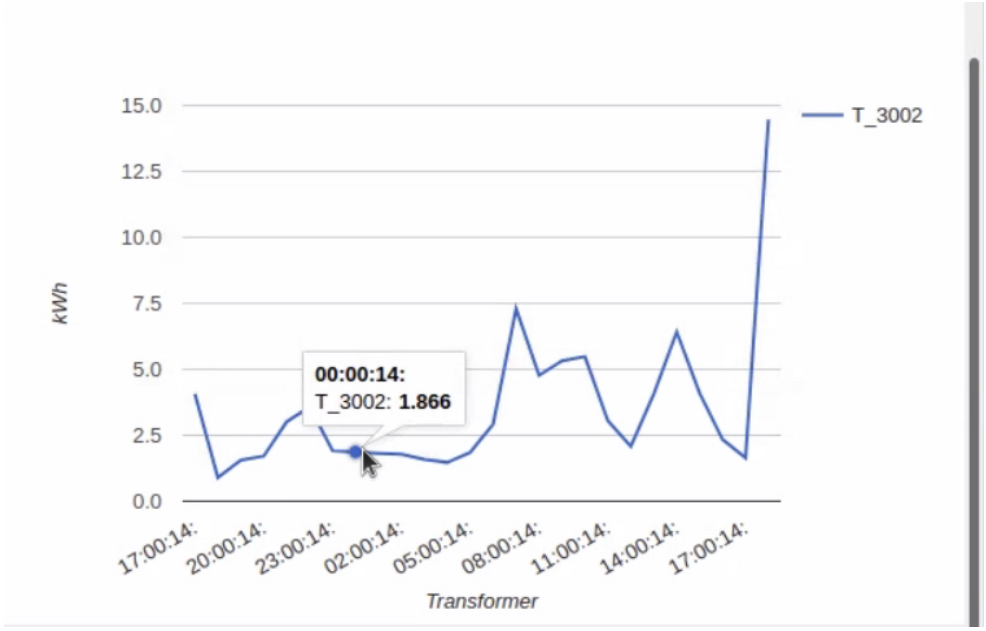

Figure 9: Toolbar Showing Node Output When Hovered Over

c. To obtain the predicted value of next hour, hover over the rightmost point in the line graph.

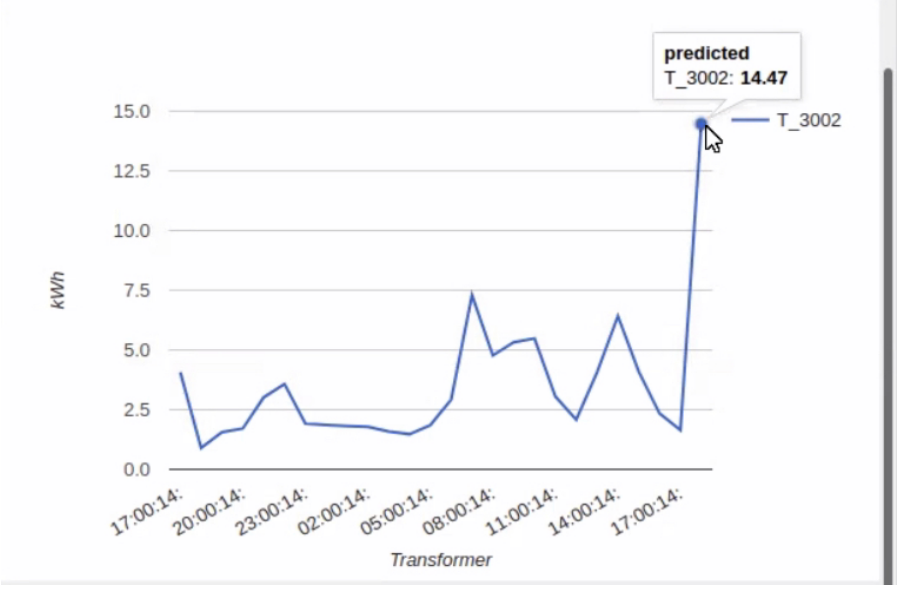

Figure 10: Toolbar Showing Node Predicted Value When Hovered Over

3. The navigation bar
   a. The navigation bar on the left is to assist the user with additional ways of viewing the information.
   b. To view the anomalies table, click on the 'Anomaly Table' under the Dashboard.

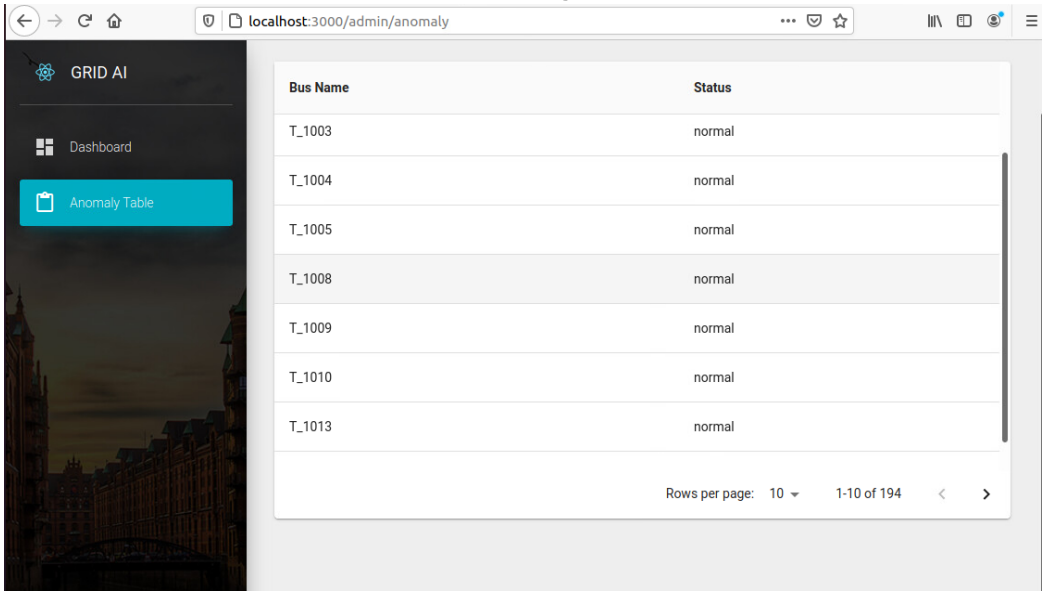

Figure 11: Table Showing All Nodes' Anomaly Statuses

Appendix B Alternate Design Versions

B.1 Original Machine Learning Design

The design and understanding of the ML algorithms and models has evolved significantly since the initial project plan and design document. At the time, not much was known about how the models would detect anomalies or what to predict. The first draft of the ML models was to have one model that outputs what the expected value should have been and then a separate model that discriminates based on the difference between the expected and actual kWh values to classify as an anomaly. Once ML in general was better understood, this design was discarded for a logistic model to classify anomalies and a linear regression model to predict the future.

The reason for the change is that the prediction model does not really fit with the purpose of this project as the expected vs actual value is better suited if the prediction is for the future hour. Additionally, the anomaly classification in the original design is more of a conjecture on how a model could theoretically work. The reality is that there already exists an algorithm that would not require the use of another model and is similar in usage to a linear regression type model. This is why we went with a Logistic model instead as that better fit our goals and purposes for anomaly classification.

Appendix C Other Considerations

C.1 Machine Learning Primer

In order to have a more in depth discussion on the ML that is being done for this project, an idea of ML on a conceptual level is important. This section is to provide readers with the knowledge needed to understand the design and implementation decisions for this project. ML at its core is a series of 3 repeatable steps, representation, evaluation, and optimization. The representation covers how the data is formatted into an input vector. The input vector covers all of the features or variables of what you are modeling. Then the model itself is represented as a series of weights and matrix multiplications that produce an output. This is all a ML model is on a fundamental level.

The evaluation stage is how you compare what you expect the output to be and what the actual output of a given datapoint was (known as the label). If you are trying to predict a continuous value you could use the distance, the square distance, or more complex ways to discriminate the expected and actual data. Truly the evaluation depends on the purpose of the model you are training and what the representation is meant to accomplish.

Finally, the optimization stage is to minimize the error of your evaluation. By taking the derivative, or gradient in multivariate models, the change of the error with the respect to the weights can be known. Therefore you can change the weights towards the direction that decreases this error given your representation of the model. This process occurs many times until you either reach a convergence or the amount of training cycles that you predetermine. There are more complexities to this process but to understand our design decisions this is all you need to know on a fundamental level.