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

Design Document

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

Revised: 11/15/2020 v3

Executive Summary

Development Standards & Practices Used

- Agile work methodology
- Standard Software Development Lifecycle
- Code should be fully tested
- Push code to Git repository often
- Utilizing VM testbeds set up for us
- Using meaningful naming conventions.
- 1012-2016 - IEEE Standard for System, Software, and Hardware Verification and Validation
- 29119-2-2013 - IEEE Standard for software testing - Test Processes

Summary of Requirements

- Develop machine/deep learning algorithms to apply to power grid data
- Create a frontend dashboard to display the power grid data after having those algorithm(s) applied to it
- Test and validate the application with power grid simulators like OPAL-RT.
- The project will run on the Google Cloud Platform
- The system will allow for real-time power grid data as input
- The frontend will be implemented as a web application

Applicable Courses from Iowa State University Curriculum

- COM S 228
- COM S 309
- COM S 311
- COM S/S E 319
- DS 201
- DS 202
- MATH 495

New Skills/Knowledge acquired that was not taught in courses

- Tensorflow
- Docker
- Flask
- Redis
- Neo4j

Table of Contents

List of figures/tables/symbols/definitions

Figures:

Tables:

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 will be able to 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 Machine Learning algorithm that can learn to analyze the power grid data and display these analytics in the form of graphs and plots. The Machine Learning algorithm will be deployed on the Google Cloud 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 client side operation for this project will be run in a docker container so that all of the libraries and dependencies are included. The machine learning and backend side of the project will

be run on the GCP (Google Cloud Platform) which will handle the processing of power grid data and servicing the client docker containers. There will be no extreme or hazardous conditions that will affect this project.

1.4 Requirements

- User Interface to view analyzed data
    - The UI should be able to parse json data from the backend.
    - UI should be able to display meaningful insights about the power grid data in forms of graphs and statistical data.
    - UI should be in the form of a web application
    - The UI should show real-time data using a moving window
- The power grid data will be streamed into the backend through json format.
- The backend server will use a trained ML model to detect anomalies in the power grid data.
- The ML model will be created using tensor flow and trained on the OPAL-RT power grid simulator.
- The backend server will send information to the frontend about potential anomalies and metrics through json
- The backend server will maintain a database of previous data in an ORM.
- The backend will be migrated from the cyber-testbed VMs to the Google Cloud Platform

1.5 Intended Users and Uses

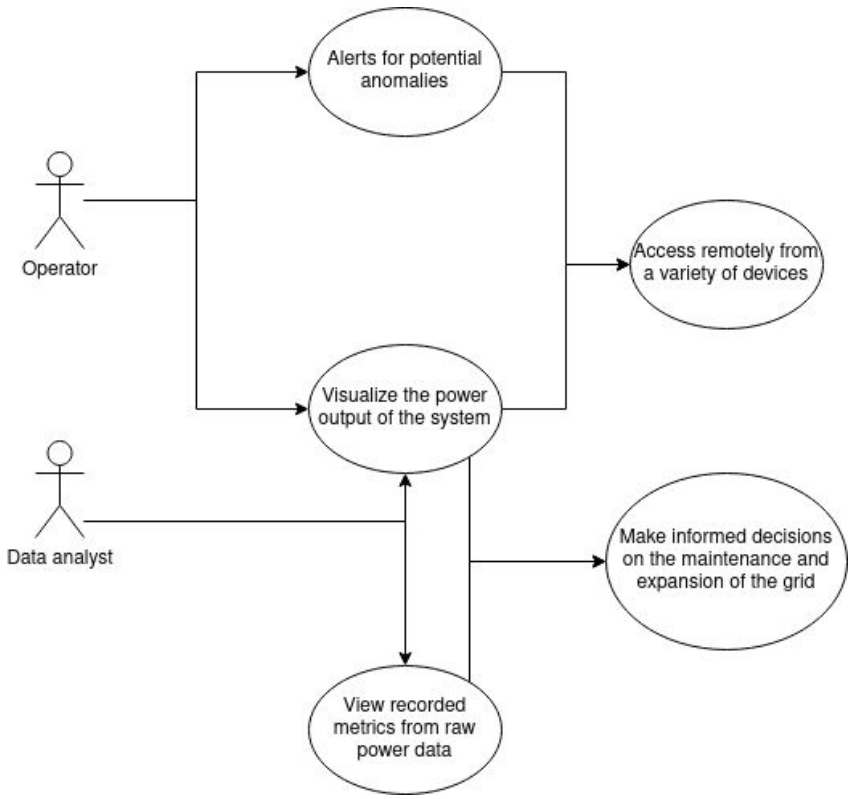

Figure 1: Use Case Diagram

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 access the data remotely from a variety of devices. Additionally, data analysts will want to view power grid data over long periods of time to make informed decisions on the power grid.

1.6 Assumptions and Limitations

Assumptions:

- Clean power grid data will be supplied to us.
- Machine/deep learning algorithms will work as expected.
- Frontend will provide a clean interface to display data from the backend.

Limitations:

- Google Cloud Platform only allows us $300 worth of trial credits per account.
- Currently, the team has a very limited knowledge on machine/deep learning algorithms.
- Testing will not be performed with real power grid data, or power grid operators, therefore the validity of testing is only from the power grid simulators (OPAL-RT).

1.7 Expected End Product and Deliverables

- A trained Machine Learning algorithm for analyzing power grid data will be implemented (Deadline: April 2021)
    - The algorithm will be created using Python and Google's Tensorflow machine learning libraries. This allows for relatively easy development of ML or DL algorithms. The algorithm will first be trained using simulated data.
- The application will be connected to a live stream of power grid data (Deadline: February 2021)
    - One of the expected requirements is to provide the user a live stream of analyzed data. Therefore it is necessary to receive a livestream of power grid data for the ML algorithm to analyze.
- A web server created with Python to handle HTTP requests (Deadline: March 2021)
    - The backend of this project will be developed using the Flask framework. This component will handle all communication with the client-side of our application. It will also be connected to the database through which we can feed the live power grid data.
- The frontend will be implemented as a web application (Deadline: March 2021)
    - Our team has decided to design the frontend in Javascript using ReactJS libraries. React is a rich library for creating user interfaces. The application will then be built into a Docker container that we can deploy to our cloud platform.
- The frontend will provide some type of geographical visualization of the power grid (Deadline: April 2021)
    - The user will be able to view the various nodes generating power in the grid. Each node will provide analytics during its operation.

2 Project Plan

2.1 Task Decomposition

In order to solve the problem at hand, it helps to decompose it into multiple tasks and subtasks and to understand interdependence among tasks.

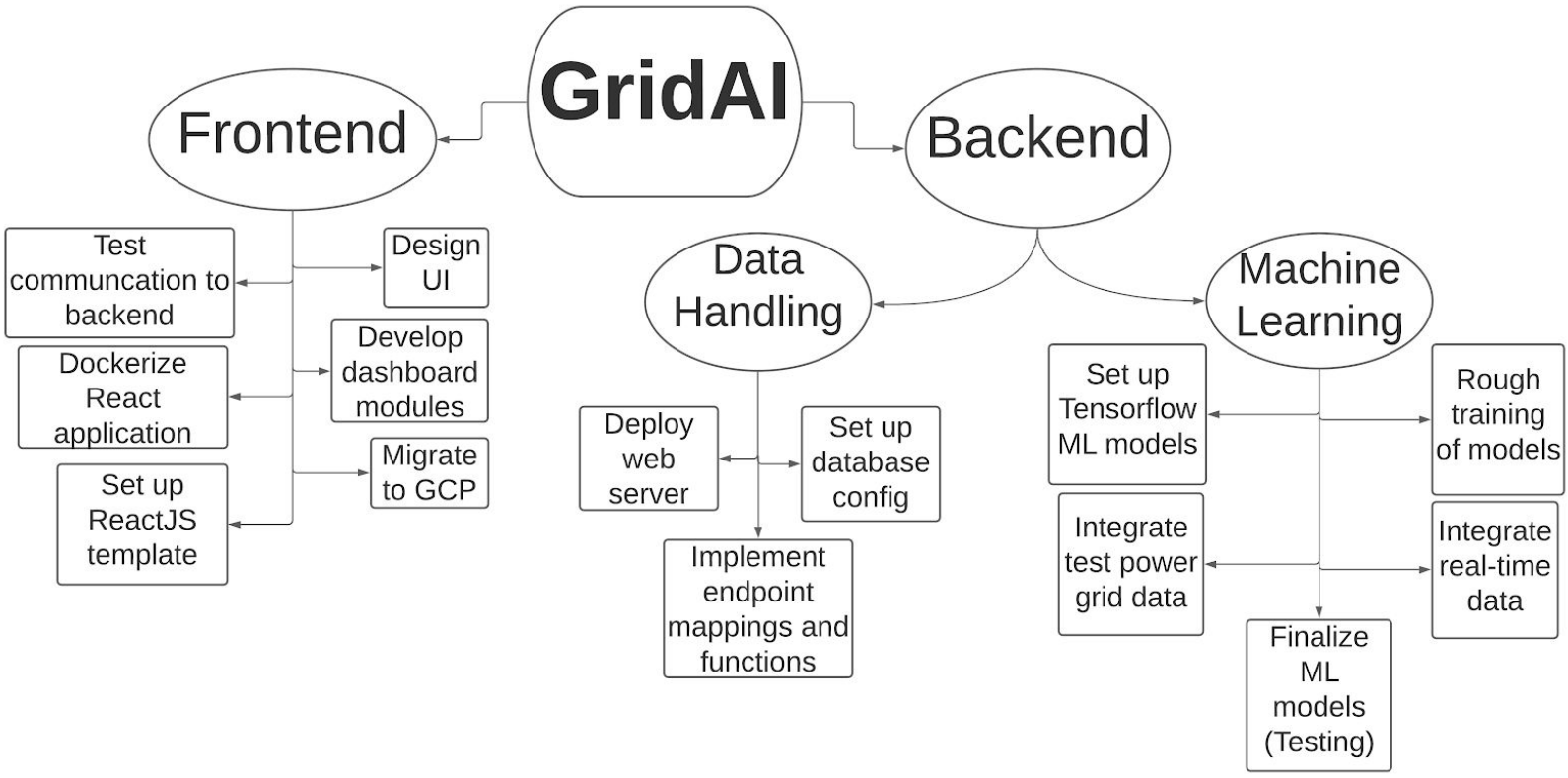

Figure 2: Task Tree Diagram

Frontend

- Set up ReactJS template for the Dashboard
- Dockerize ReactJS for PowerCyber testbed
- Migrate from PowerCyber testbed to GCP
- Design the UI for dashboard
- Start building the dashboard modules
- Test communication with backend to display data
- Prototype a working version of the dashboard

Backend

- Web server and Database
    - Deploy web server
    - Setup database config
    - Complete URL mapping to database
- Machine Learning Development
    - Setup Tensorflow
    - Integrate test power grid data
    - Configure ML real-time data
    - Rough training of ML model (Narrow algorithms)
    - Finalize ML model (Testing accuracy and functionality)

2.2 Risks And Risk Management/Mitigation

- The accuracy of the machine learning algorithm to detect anomalies will only be trained on OPAL-RT
    - This means that the machine learning model will only have simulated data to predict anomalies in real-world data
- There is a spending requirement of 300$ for using the Google Cloud Platform for the backend
    - This means that the migration will occur later into the project timeline so if there are any problems with the migration there will not be a lot of time to diagnose.

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

- The ML algorithm should be able to classify anomalies with 85% accuracy.
- The lag between the real-time data and analysis shown on the frontend should be less than 3 seconds.

2.4 Project Timeline/Schedule

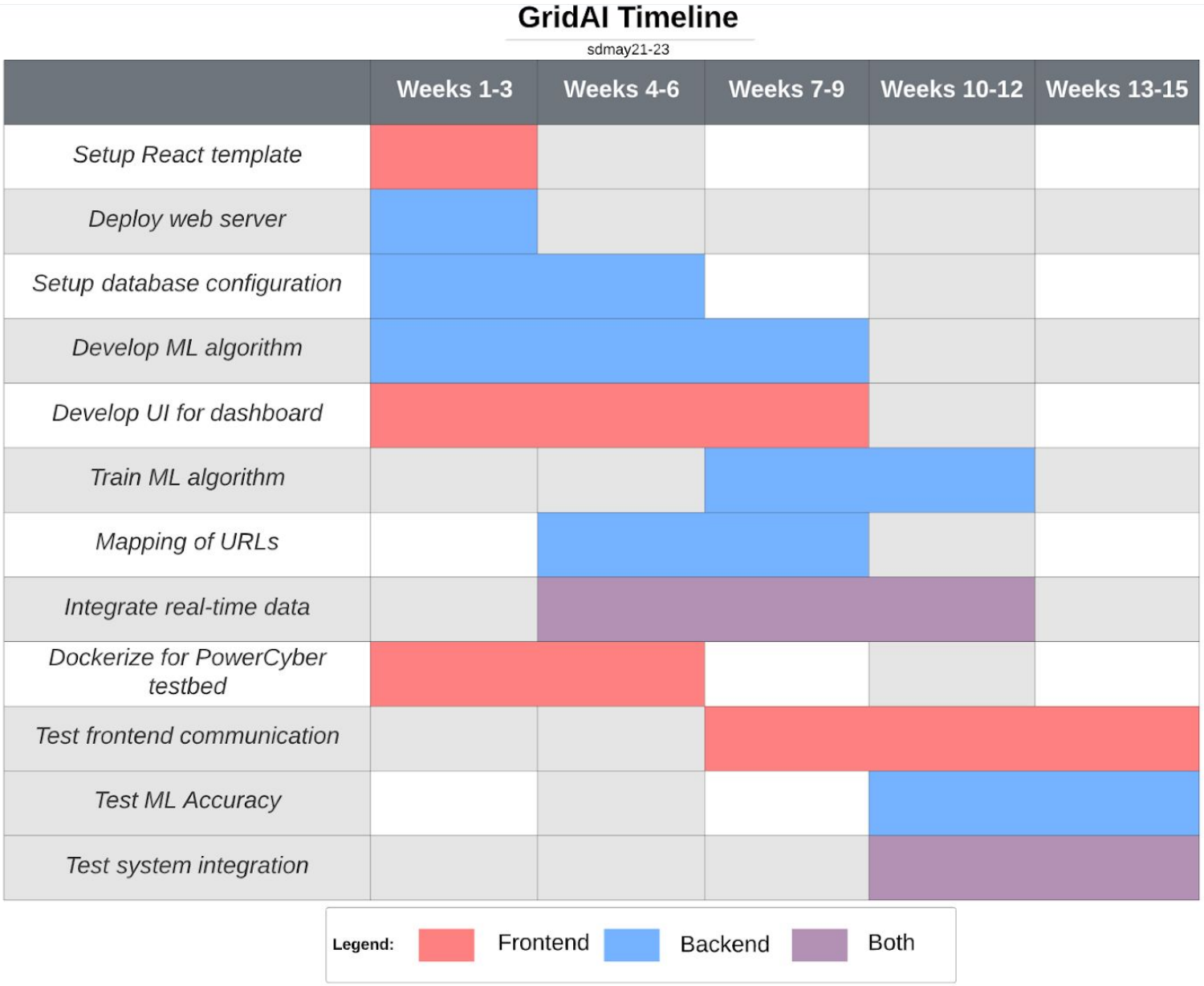

Figure 3: Grid AI Timeline

2.5 Project Tracking Procedures

- Git
- Discord
- Zoom
- Email
- Google Drive

- Jira

Git will be used for version control of our project, allowing team members to develop on the project at the same time, and to allow for future use of the project.

Discord will be used for all internal communication within the team, voice and text.

Zoom will be used for our weekly meetings with our client, Dr. Ravikumar.

Email will be used to communicate to our professor.

Google Drive will be used to store all group assignments like reports, reflections, lightning talks, design documents, and presentations made by us and the professor.

Jira will be used to allow us to follow the Scrum/agile methodology, keep track of what we are working on and who is working on it, what tasks have been completed and by who, and what tasks still need to be completed in the upcoming sprints.

2.6 Personnel Effort Requirements

Task	Description	Person-hours required
React template setup	The frontend dashboard uses ReactJS.	4-5
Deploy web server in Google Cloud Platform	Primary site for frontend and backend/ML will be setup via GCP	6-10
Dockerize ReactJS for PowerCyber testbed	ReactJS will be dockerized for use in PowerCyber and later in GCP server.	2-3
Setup database configuration	Setting up tables for storing input and output data for the ML algorithm	6-8
Design UI	Based on client requirements, a suitable UI needs to be designed	10+
Setup Tensorflow as the primary ML framework	Tensorflow initial setup to accommodate input/output data type(JSON) for training models.	8-12
Connect database repository to the backend framework	Sending and retrieving data from the ML framework will be done through a backend	3-5

	software. This software will communicate with the database	
Building UI modules	After initial design of the UI, the components will be put into development by the frontend team	10+
Integrate power grid data	Configure Tensorflow to take JSON data input	3-5
Connect backend with the frontend	Setting up communication link(s) between backend and frontend.	2-3
Training of ML model	After initial experimentation of the framework, training algorithms will be used on the provided data to detect anomaly and other analyses	30+
Configure frontend and backend for real-time data	The dashboard should show real-time data analyses for this project. Configuration of ML/backend with frontend to transmit data in real-time	6-8
Testing Dashboard (frontend) prototype	The prototype version of Dashboard UI should meet major client requirements	5
Testing/Finalize ML model	Assuming the ML has trained its model, test the accuracy and functionality of its anomaly detection capability	8

Table 1: Personnel Effort Table

2.7 Other Resource Requirements

Since this project is completely built through software, there are no additional resources required such as materials or parts. The project will be using Google Cloud Platform for backend and frontend. Resources will be provided by the client. We will be using open source software including Tensorflow, Flask, Docker, and MySQL.

2.8 Financial Requirements

Our one financial requirement is to ideally stay within $300 worth of Google Cloud Platform credits. Other than that, we will be utilizing open-source tools, so our team should not incur any monetary costs.

3 Design

3.1 Previous Work And Literature

Signal Processing ML Power Grids:

This project uses signal processing and machine learning to analyze signal data from power grid systems to determine whether the signals were "good" or "bad". Bad signals are classified as signals that generate partial discharges and cause faults and/or power outages.

Deep Machine Learning:

This project applies deep learning to power systems. Like our project, this project used TensorFlow as its machine learning engine. This project, however, uses InterPSS to simulate a power system while we will be using OPAL-RT.

GridAPPS-D:

Node Distribution visualization with simulation of capacitors and regulators. Produces graphs based on the power grid distribution system simulation. Users can force anomalies to be visualized in the graphs panel.

3.2 Design Thinking

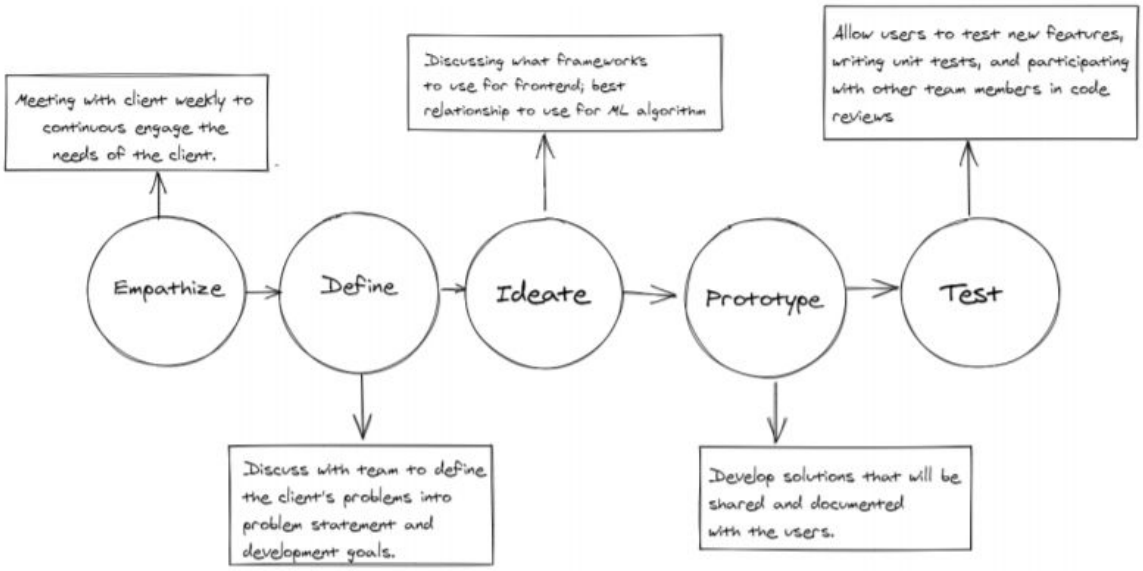

Figure 4: Design Thinking

The diagram in **figure 4**, shows the iterative process the team used to implement Design Thinking. The following table shows some of the problems the team defined and the resultant solutions that were devised during the process.

Define Problem	Ideate Solution
The real time prediction of anomalies.	This problem requires a pre-trained ML model to respond to a constant stream of data.
In order to implement a scalable and generic solution to power grids, a robust system agnostic ML model must be developed.	To achieve a scalable and generic solution, the ML portion of this project will need an algorithm to predict the expected power output for a given node and a discriminator algorithm to determine if the actual power output constitutes an anomaly.
The defined requirement to have a client that can access data from any device.	The client will be in a dockerized container to be accessed from any device.
The client will want to be able to see data over a larger period of time.	The backend will store all of the power grid json data in the ORM database. The frontend will visualize this data using a dynamic react template.

Table 2: Define & Ideate

3.3 Proposed Design

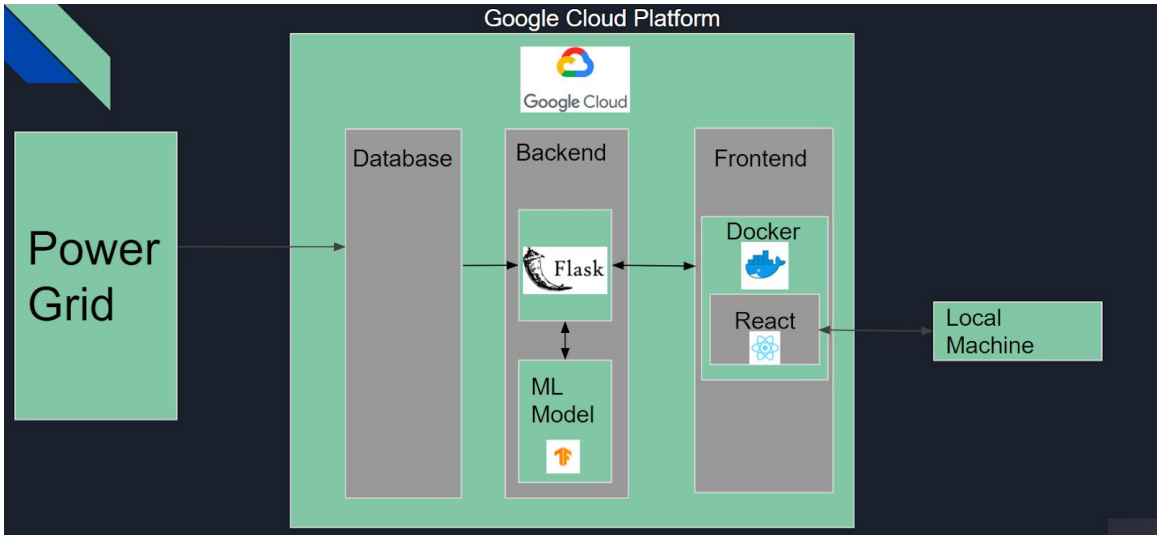

Figure 5: Proposed Design Diagram

As of right now, we are familiarizing ourselves with the basic technologies. We have set up basic applications for the frontend and backend to test communication. Additionally, we have created a simple machine learning algorithm, though it is not related to power grid data. Given that we are developing the project in a modular format as seen in the diagram above, the design satisfies our client's need for a scalable application. The core functionalities are abstracted into three categories: Database, Backend, and Frontend. The backend is further divided into the request handling application and the machine learning algorithm. This modularity also lends itself to the project being maintainable because problems can be easily repaired within a module without heavily affecting the system. Using Flask to implement the request handling, allows our team to optimize the performance of the application since Flask is lightweight.

Design elements progress so far:

1. Setting up Docker with Hello World Image.
2. Tensorflow (and Keras) libraries downloaded.
3. Anaconda (Python) setup for use with Tensorflow.

3.4 Technology Considerations

We have decided to use Flask as the framework for our backend as opposed to Django. Django allows access to a rich library for backend functionality, but it is more strict with structure requirements. Also, the rich library for Django comes at the expense of efficient performance. Since GridAI depends on real-time analysis, our team chose Flask for its lightweight implementation and flexible structure.

When deciding what framework/library to use for the frontend, our team narrowed the options to AngularJS and ReactJS. While Angular allows for more flexibility when designing, the learning curve is steeper than React. React offers more benefits to our project. For instance, React only requires one coding file in JavaScript while Angular requires both a TypeScript and HTML file. Our

team has some prior experience with React as well. Both of these reasons contribute to React being the better choice for our project.

PyTorch and Tensorflow were our choices for machine learning libraries. Both open-source libraries are plentiful and well-developed. Some say that PyTorch is more intuitive and easier to learn, but Tensorflow has a larger community. Thus, many tutorials can be found regarding implementations of Tensorflow. PyTorch may run into issues when trying to scale a project, and scalability was a highly desired characteristic from our client. With this information, Tensorflow seems like the correct choice.

3.5 Design Analysis

The proposed design from section 3.3 (see Figure 5) has google cloud integration of our backend infrastructure. Since we currently do not have the google cloud resources available to us right now, we progressed with constructing a naive "Hello World" project to get started on the backend and frontend communication.

We are still in the process of learning and experimenting with ML models which is why we have not integrated the test data into our current design.

In the next steps on the development sprint, we will work on integrating the test data into our ML model. As per our ongoing discussions with the project adviser, we will set up different ML models for specific metrics which will help us to provide different types of insights into the power grid data.

For the frontend, we plan to progress from the "Hello world" application to a project suited application having several endpoints to receive data from the backend.

3.6 Development Process

We plan towards working incrementally on different aspects of the project by distributing responsibilities in the group. We plan to follow the regular agile development process for the project i.e. Meet -> plan -> assign tasks -> develop -> test -> evaluate.

3.7 Design Plan

The project has 3 main modules, the frontend, backend, and database each with their own unique requirements and dependencies. The diagram in 3.3 proposed design shows the visual interconnection of the modules. The context for those connections is provided with respect to the requirements and use cases is as follows.

Frontend:

- Requirements
    - The frontend must display the data for the operator and data analyst to clearly understand the status of the power grid.
    - The frontend must be able to be accessed from a wide variety of devices to facilitate the operator in all situations. This requires the use of a docker container for the react application.
- Dependencies
    - The frontend needs to receive the ML processed data from the backend to display.

Backend:

- Requirements

    - The Flask application must be able to detect anomalies with ML so that the operator can have up to date information.
    - The backend must communicate in real time with both the database and frontend to facilitate the requirement of real time data.
- Dependencies
    - The backend needs to receive real-time power grid data from the database.

Database:

- Requirements
    - The database needs to be robust enough to access large amounts of data to send to the frontend so that the data analyst can make fully informed decisions on power grid maintenance..
- Dependencies
    - The database needs access to the uninterrupted real time stream of power grid data.

4 Testing

Test Process Standards:

The IEEE standardizes Software Testing processes. We will implement two of these processes: Test Management Process and Dynamic Testing Process.

Test Management Process : This process documents how the testing will be performed. The steps are to plan the test process, monitor and document the testing results. The test engineer will develop a plan to test the various modules in the backend, frontend and ML platform. The testing will be done phase by phase as the modules start being implemented so that bugs and other issues can be resolved quickly. Table 1 shows tasks that need to be completed; the test plan will follow according to the set of tasks scheduled.

Dynamic Test Process: This is where the actual Unit, Integration and Acceptance testing occurs for each of the modules. The overview of these test requirements can be found below.

4.1 Unit Testing

- Machine/Deep learning algorithms

The machine learning algorithms will need to be tested in isolation to ensure it functions as intended. We will use dummy data to feed into the algorithm to check that it is processed correctly. Then we will need to examine different statistics for each model to determine its accuracy. These metrics can be obtained through Tensorflow.

- 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 can use an application such as Postman to send REST requests to our backend application. Then we can see exactly what a response to the specific requests will contain.

4.2 Interface Testing

- SQL queries -> Machine/Deep learning algorithms -> Frontend functions -> Frontend design

We will first start out by testing our SQL queries to make sure that the correct data is being pulled in the correct form. We will also be testing our ML/DL algorithms with test data to make sure that they are working properly and outputting the correct data. We will then be passing the data from our SQL queries through the ML/DL algorithms.

Then, we will be testing from the frontend that we can make a call to the backend to get the data that got passed through the ML/DL algorithms correctly. Next, we will test any extra functions that we have written to further manipulate the data to make sure they are performing as expected. Finally, we will then be testing the user interface we create and the visualizations we've made for the data are well organized.

4.3 Acceptance Testing

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

Incremental acceptance testing by the client will meet 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 Results

Through discussions with our client, we have determined that splitting the design into three separate applications is the best plan of action. These applications would be the user dashboard, the request handling backend, and the machine learning algorithm. This has allowed for easy testing of individual components since we can work independently. As we discover more complexities within our project, further abstraction of the applications may be necessary, such as running multiple ML algorithms for separate metrics; however, the independent nature of our modules will accommodate this flexibility.

So far, we have created a simple React application that we can iterate upon to work towards a fully functional metrics dashboard. We have initialized the connection between the frontend and backend with simple data retrieval endpoints. Our plan is to gradually increase the complexity of the data being passed so that we can work towards sending the large power grid datasets. Our team is still working to build our knowledge base of Tensorflow before attempting to

analyze the test grid data. We have successfully created an algorithm that takes stock price data to predict future prices. Since stock prices are a time-series dataset similar to the power grid data, the skills we learn through this experiment will translate to the GridAI project. Attempts to Dockerize our applications have varied in difficulty. The React and Flask applications are relatively simple to run in Docker containers, but the machine learning application requires more effort given the number of dependencies needed to run the application. As the design becomes more complex, creating Docker containers will prove more difficult. For this reason, configuring the Docker containers early in our project timeline will eliminate potential problems with integration, scalability, and maintainability in the future.

5 Implementation

Our plan for implementing GridAI is to follow the same workflow that we have been utilizing this semester. We will continue our Agile development process, so each module as described in Figure 5 can be iterated upon independently. The backend team will work to configure the database to fit the requirements and format of our simulated power grid data. Simultaneously, other members of the backend team will work to set up the request endpoints to receive and pass data. Also during the same period, backend members will be working on creating an initial machine learning model to predict the power grid values with some reasonable level of accuracy, which will be iterated upon. While backend members work on those components, the frontend team will aim to configure the dashboard with the appropriate modules to clearly display our data. The frontend team can use both dummy data as well as the sample data that we have now to ensure that the client-side application functions as intended.

As we can see, each of the categories listed in Section 3,3 can be individually progressed and tested. System integration will occur as the necessary functionality is completed and approved by all team members. For example, a member from both the frontend and backend team can work to display raw data from the database as soon as the database is configured, an endpoint is established for retrieving the data, and an appropriate module is present in our web application. This type of integration will help to confirm our components are operational incrementally as opposed to integrating the components only once each module is fully completed. Once components are integrated, members will continue to test the connection and to refine the functions until it is satisfactory to both our team and our client.

6 Closing Material

6.1 Conclusion

So far for the frontend, we have downloaded an example Material UI dashboard to build our application on. We are currently in the process of figuring out the structure of how the pages and their components get rendered so that we are able to start working on our own pages. On the backend, we have started integration of a database that will be used for logging, created test REST endpoints for passing of simple data, implemented a deep learning model to enhance our understanding of machine learning, and begun work on a deep learning model to analyze sample

power grid data. Our next steps are to integrate a database for storing the power grid data and to refine our ML model to accurately analyze the test data.

Our end goal for the frontend of this application is that it will be able to grab the power grid data from the backend and display it in graphs and charts. We also want to be able to display a map of the power grid. With these features, we want to provide a clean, easy to use, and good looking interface for the user when they are utilizing this application. We aim to provide quick and efficient data retrieval for the frontend so that we can ensure the data is visualized in real-time with the backend. Along with that, we want to train a machine learning model that can accurately predict and classify power grid data. By doing all of the above, our team can optimize the effectiveness of the GridAI project.

In order to achieve these goals, we will be using the Agile framework over the course of implementation and checking in with our advisor/client to make sure the work we are doing is acceptable. This is the best plan because the Agile framework will allow us to see what tasks we have completed during the sprint, what tasks are in progress, what tasks we still have left, and how much time those tasks should take. Frequent check ins with our advisor/client will also allow us to get constant feedback on the work we've done and if it is satisfactory or if there is something we should change.

6.2 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

Zhou, Mike., RL, CB Dev., Feng, Donghao., InterPSS (2020) DeepMachineLearning

[Source Code] https://github.com/interpss/DeepMachineLearning