AI-VVO: Cloud-Based Machine/Deep Learning for Volt-VAR Control and Optimization

FINAL REPORT

sdmay21-24

Client/Advisor: Gelli Ravikumar

Abdul-Salam Adedoja
Ian Kegley
Jacob Gleason
Rene Chavez
Tyler Norris

sdmay21-24@iastate.edu
https://sdmay21-24.sd.ece.iastate.edu

Table of Contents

List of Figures

1 Revised Project Design

1.1 Design Evolution

The core components of this project and their functionality over the course of the project's implementation. We implemented a machine learning algorithm using TensorFlow, a user interface using ReactJS, a web server using Django, and the entire project has been Dockerized and can be built using Docker containers. By Client request, bash scripts for building, starting, and stopping the Docker container were created and implemented. The PostgreSQL database was not used as much as originally planned but was left in the case that another team continues work on the project.

1.2 Functional Requirements

- Docker Containers
  - We implemented docker containers for ReactJS, PostgreSQL, and Django. We also created a Docker-Compose file that will build each Docker image for the project. These Docker containers allow for seamless deployment to the cloud.
- Communications/Web Server
  - The design's web server is implemented using Django. The Django web server handles HTTP requests from the dashboard and relayed data to and from our machine learning algorithm.
- Machine-Learning Algorithm
  - Our machine-learning algorithm was implemented using TensorFlow. The algorithm used a reinforcement learning approach and would take in information about the grid, such as nodal voltages and the status of VVC devices.
- Frontend Dashboard
  - We created our dashboard around a React skeleton program shared with us by our advisor. From this, we were able to develop it based on our user interface (UI) skeleton images from our 491 design document. We created windows for login, algorithm configuration, and energy grid visualization to house each of the different components of our project.

1.3 Non-Functional Requirements

- Application Portability
  - Through the use of Docker containers, we were able to make our project portable. Docker is a tool designed to make it easier to deploy and run our application by using containers. With Docker containers, we are able to wrap our application together with each library, dependency, etc. that is required to run our program for easy portability and deployment.
- Communication/Web Server Security
  - Each HTTP request sent through our Django web server was sent securely, protecting the energy grid information.
- Algorithm Accuracy and Efficiency
  - As with any algorithm, accuracy and efficiency are important and we wanted to ensure these were satisfied by ours as well, especially due to the importance of the energy grid in our daily lives.
- Dashboard Usability

    - The dashboard was made to be very user-friendly. It had easily identifiable components that were clearly labeled.
- Visualization Performance
    - Due to a large number of nodes within our simulated grid model, creating and displaying the visualization of the energy grid was programmed in an efficient way to maximize its performance.

1.4 Engineering Standards

- IEEE 1250-2018 - IEEE Guide for Identifying and Improving Voltage Quality in Power Systems
    - Although our project was based on simulations of voltage, we had to ensure that it conformed to the standard that made sure electrical equipment can withstand surges, faults, and distortions.
- IEEE 1854-2019 - IEEE Trial-Use Guide for Smart Distribution Applications
    - This standard was used as a guide to determine what type of application our project was in regards to its functions and how its components are defined.

1.5 Engineering Constraints

We faced a couple of engineering constraints when working on implementing our project. The first of these constraints was that our power grid data was not real-life data taken from actual Distributed Energy Resources (DERs). Instead, our advisor provided us with simulated data, saved and imported as a CSV file format. This simulated data was essentially the sole data we used to train our Machine Learning algorithm. We were able to continue using this simulated data, but this is a constraint in our minds. This data is not from a real-life power grid; this could prove some faults if ever tested on a real-life system.

Another engineering constraint that our group had to face was the limited number of time steps to reduce our state space.

1.6 Safety Concerns and Countermeasures

There were some safety concerns that we had to consider when designing and implementing our project. One substantial safety concern we dealt with is the current COVID-19 pandemic. We were all cautious about the virus and always kept this in mind for our semester plans. Because of this, we were never able to meet in person to work together on things. This was a challenge to adjust to but, we made use of our time effectively. The main countermeasure we used was coordinating our weekly meetings using Zoom and Discord platforms. We used these to communicate and collaborate when designing our project. Overall, our project was very secure and did not require any activity to place our group members at risk for any harm. Due to the structure of our project, we used simulated data from power grids and never tested our product in any real-life systems.

2 Implementation

- **Django Web Server**

  The Django web server controlled HTTP requests made from the dashboard, most often requesting information from the machine learning algorithm. The web server consists of three main apps. The first handles routing for API calls to the server, storing information such as other installed apps, or apps that were created for the project, and connection

information for the database. The second app is used for authentication. This app handles logging a user in and out, as well as allowing a user to update their password. The final app handles requests to the machine learning model. This app will take in the current state of the VVC devices for our grid, and pass them to the model, it will then return the algorithm's output to the dashboard.

- **Dashboard/User-Interface**

  The dashboard for this project was implemented using ReactJS. Each page was implemented as a React component. The dashboard consisted of four main pages:

  - *Home page*

    The home page is the first page a user sees when they log in. The user is presented with 4 sliders. each slider corresponds to a different control device for the distribution grid. The first three sliders are for the three voltage regulators on our grid. The voltage regulators each have eleven different positions that correspond to an integer between -5 and 5, hence the ranges for the first three sliders are -5 to 5. The fourth slider is for the capacitor bank. The capacitor bank can only be on or off, so its slider only has options for 0 and 1. These sliders will indicate the starting positions for each control device in a simulation, so once the user has selected their desired starting positions, they can press the start button and will be brought to the ML output page.

Figure 1 - Home page user-interface

  - *Output page*

    The output page allows the user to run a twelve-time step simulation. Upon arriving at the page, the user is able to change their starting state for the grid's control devices. If the user previously selected starting positions on the homepage, those positions will already be reflected, otherwise all starting positions default to

zero. The user can then begin a simulation with the simulation button. In a simulation, the current timestep and device positions will be printed in the middle column, seen in figure 2. The new device conditions that are received from the machine learning algorithm will be displayed will be printed in the rightmost column. The new positions will then be sent back as the input for the next timestep.

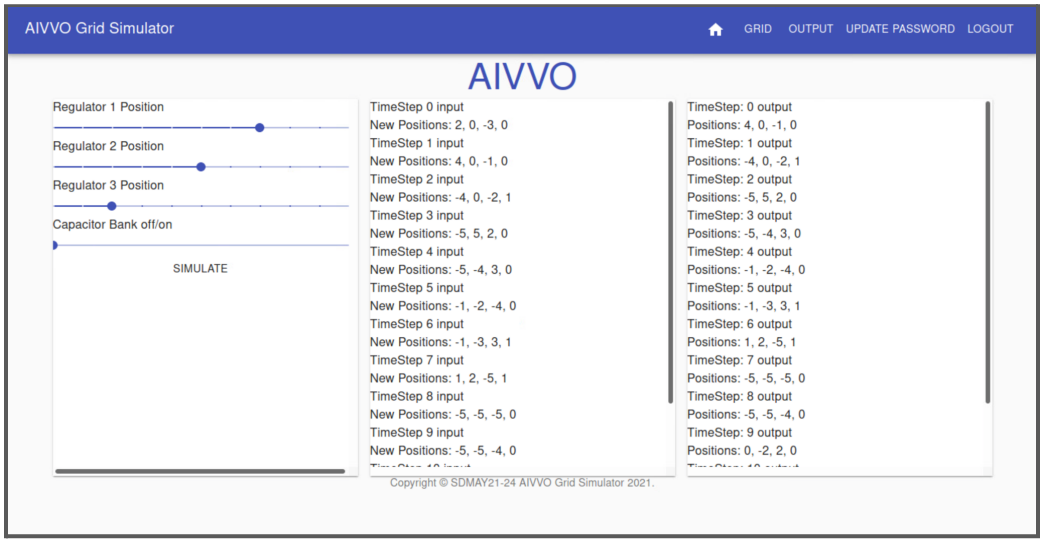

Figure 2 - Output page

- *Grid Display Page*

The grid display page provides the user with a visual representation of the Power Distribution Grid. The grid is visualized as a graph using the React-D3-Graph library. Each node is labeled with its name, and edges can be clicked on to see connection information. Unfortunately, planned features of this page, such as voltage information of each node, had to be foregone because of time constraints.

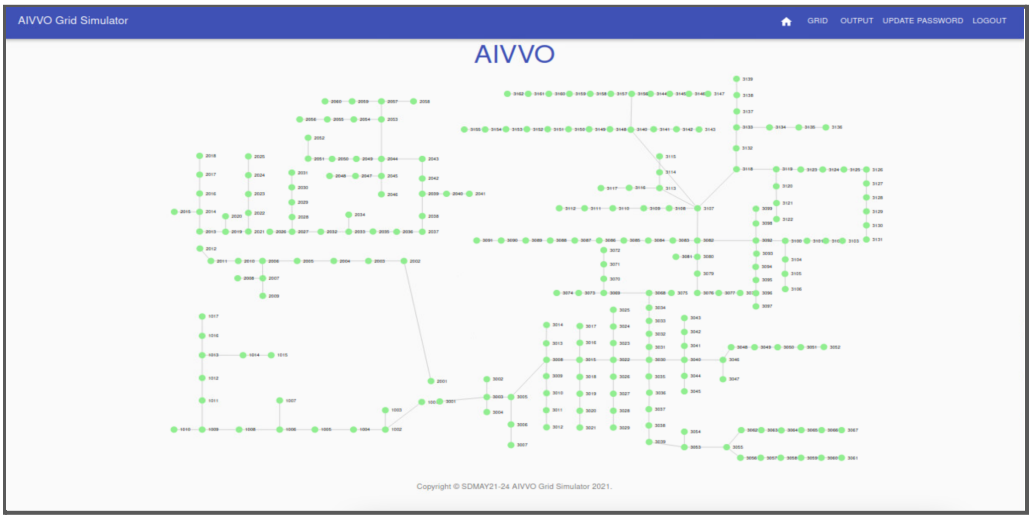

Figure 3 - Grid Visualization page

- *Login page*

  The login page is for authentication purposes. A user that does not have valid authentication credentials will be redirected to this page. Logging in will assign the user an authentication token. This token allows the user to access all other pages of the project.

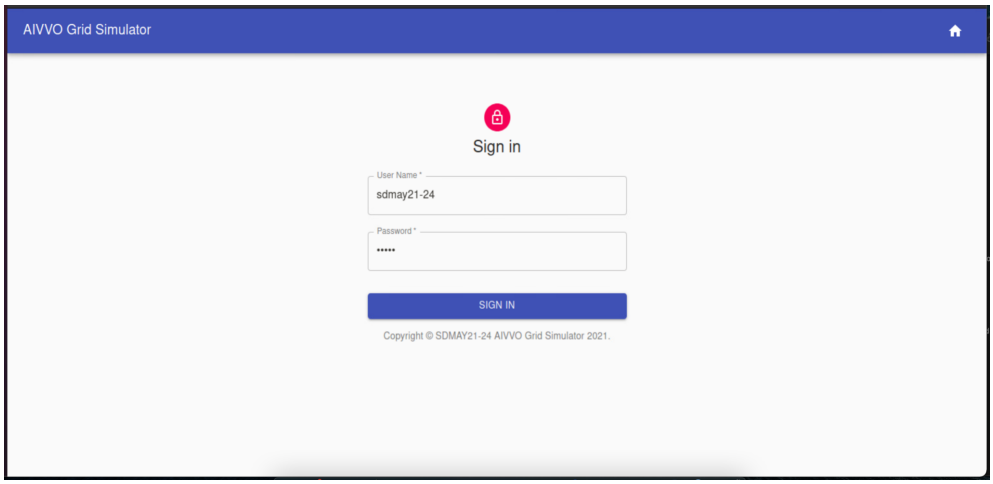

Figure 4 - Login page

- **Machine Learning Algorithm**

  For our machine learning algorithm, we elected to use reinforcement learning. Our agent was seen as a grid operator, and the environment was the tap positions for each of the control devices of the grid. Our algorithm used action value statements to determine what action should be taken. An action, in the case of our algorithm, was changing the position of each device. A state consisted of the current timestep, the nodal real and reactive voltages of the distribution grid, and the current positions of the control devices. Each state had a certain number of possible actions that could be taken from that state. The number of actions was determined by the number of possible combinations of device positions. Each voltage regulator has 11 possible positions. The capacitor bank can be on or off, each of which is considered a 'position'. This meant that there were 11 * 11 * 11 *2 = 2662 possible actions that could be taken in each state. The action taken was determined by the reward for that action in the current state. These reward values were stored in a table called a Q-table. The reward for a given action was calculated by first calculating the power loss in the distribution grid and multiplying it by a cost coefficient. Then the difference in device positions was calculated by a sum and also multiplied by a separate cost coefficient. Each of these values was then summed together to get the value of the reward. Q-table was filled during training by taking a random action a certain percentage of the time. Once our algorithm was trained, the reward table was exported as a joblib file, and placed in a directory where it could be accessed by the Django web server.

- **Docker**

  The Docker container was originally intended to be the last implemented component of the design. However, early on during implementation, we decided to create docker containers

for the design once the basic architecture of the design had been set up. This allowed us to verify that the changes and new features would not interfere with the deployment of the project. We used Docker images for React, Django, and PostgreSQL, and then created a docker-compose that could build each image.

3 Testing

3.1 Algorithm Training and Results

The extent of the testing for our project was on the training and testing of our machine-learning algorithm. Essentially, we unit tested and manually verified that the results were meeting our expectations. With the given data that was provided to us, we were able to simulate our Machine Learning Environment. Using our data, we were able to test and train our Machine Learning algorithm to ensure that output is what we need. As a result of implementing our algorithm, one primary issue that we ran into with our algorithm is that it fails to converge. This could be a result due to not training and testing our algorithm sufficiently. When working on implementing our algorithm, we were not fully successful in implementing our power loss formula for our core application. This could also be a result of why our machine learning algorithm failed to converge.

3.2 Application Testing and Results

The testing of our application involved ensuring each individual interface was functioning on its own. Once this was done, we needed to establish a connection. Ensuring the front-end was functioning consisted of making sure that each panel was operating in an expected manner. An example of this was guaranteeing that we could input the correct values in our algorithm configuration page. Certifying correct back-end functionality involved checking that data was properly being received and sent. Thus, confirming that our algorithm correctly received the simulated data from our database and that our algorithm output was being sent back to the database.

4 Appendices

4.1 Appendix I: Operation Manual

In order to run the project locally, this assumes that PostgreSQL is already installed, and is for an ubuntu system:

1. Clone the git repository
2. Create a local python environment called localPythonEnv using the command "virtualenv localPythonEnv"
3. Activate the environment using "source localPythonEnv/Scripts/activate"
4. Use "pip install -r requirements.txt" to ensure that Django and its dependencies are installed
5. Create a PostgreSQL database named 'predictions' that is owned by a user with the following credentials:
   username: postgres_user
   password: postgres_password
6. Once that has been created, in the backend/django_app directory use the command "python manage.py migrate" to migrate the database
7. Create a superuser, the name and credentials of this superuser will be stored in the local database. Use the command "python manage.py createsuperuser"
8. Navigate to the frontend/react_app directory and run "npm install" to install the required dependencies for the user interface
9. At this point the application can be built by navigating to the main project directory and using the command "bash aivvo-start.sh" to start the entire project
10. The project can be terminated using the command "bash aivvo-stop.sh"

Using the Application:

1. Once the application is running, navigate to the home page.
2. You can log in by entering the username and password of the superuser that was created when setting up the application.
3. Upon logging in, you will be taken to the home page. On this page, 4 sliders will be available. Each slider selects the starting tap position of a different control device, starting with voltage regulator 1 and ending with the capacitor bank.
4. Each regulator has 11 possible tap positions. The tap positions are each mapped to a number between -5 and 5, with 0 being the center reference.
5. The capacitor bank has 2 "positions" that correspond to the capacitor bank being on or off, 0 is off, 1 is on.

6. Once you have moved the sliders to their desired positions, you can press the start button, and will then be taken to the ML output page.

7. On this page, you can again change the starting tap position of each device, however, the sliders should already be set to the positions selected on the home page.

8. Once you are satisfied with the positions, you can select the simulate button.

9. When the simulate button is pressed, the text will start appearing in each of the columns to the right of the sliders; the middle column will display the tap positions input into the machine learning algorithm, the positions will be displayed from left to right in the following order: regulator 1, regulator 2, regulator 3, capacitor bank.

10. In total, the simulation should run for 12-time steps, with each time step being one hour. When the simulation has finished, you can readjust the sliders, and run another simulation without needing to return to the home page.

11. A visualization of the power distribution grid can be seen by selecting “Grid” in the navigation bar at the top of the screen.

12. Upon arriving at the page, you will need to initialize the map by pressing the “Initialize Grid” button. When pressed the grid should automatically appear.

13. The name of a node can be seen by clicking on it.

14. Clicking on a bus will display which nodes that section of the bus is connected to.

15. Once you have finished using the application, you can log out by selecting “Log Out” in the top right corner of the screen.

4.2 Appendix II: Citations

"IEEE Guide for Identifying and Improving Voltage Quality in Power Systems," in IEEE Std 1250-2018 (Revision of IEEE Std 1250-2011) , vol., no., pp.1-0, 16 Nov. 2018, doi: 10.1109/IEEESTD.2018.8532376.

"IEEE Trial-Use Guide for Smart Distribution Applications," in IEEE Std 1854-2019 , vol., no., pp.1-65, 30 Aug. 2019, doi: 10.1109/IEEESTD.2019.8820199.