****

**Boston University**

**Electrical & Computer Engineering**

**EC464 Capstone Senior Design Project**

**Final Test Report**

**Machine Learning Powered Electrical Scheduling**



Submitted to

Alan Pisano

8 St. Mary’s Street

Boston, MA

(617) 353 - 6264

apisano@bu.edu

by

|  |
| --- |

Team 28

EcoStrip Solutions

Team Members

Avishai Lean alean@bu.edu

Benjamin Axline baxline@bu.edu

Mateusz Górczak mgorczak@bu.edu

Konstantin Agrachev kostya@bu.edu

Submitted: 4/13/2025

#### **Equipment and Setup**

Our physical equipment consisted of a laptop for demonstration purposes, a desktop computer used as our server, a Kasa Smart Outlet, and a router for the outlet to connect to. On the software side of things, there are several dependencies that had to be installed for this prototype to run, namely Python3, FastAPI, Node.JS, React, and the Kasa Devices Python library. However, our users will have no troubles using our website with minimal dependencies assuming an internet connection is present.

For the pre-demo setup, we connected the outlet and laptop to the router network. The laptop ran a FastAPI script listening for calls to that laptop’s IP address, calls which would be made from the website on a different network. On that same laptop, a tunnel bridging calls from outside its local network was open, so that the endpoints to the API could be hit (and so that the powerstrip can be operated through the website). Once a day our server pushes out the latest energy generation predictions to the github where our front end then displays this information on the website, for the testing we are displaying data from the auto-push last runs.

During the demo, we will start by showing how our lab computer is used as a server as well as reading through how the code works and going through the individual test error logs. While this is not necessary to the functioning of the website, it will provide insight into how our back-end functions. From there, we demonstrated how this data would be displayed via the website (in the form of graphs), and showed how one would log into the website to be able to see said data. Finally we demonstrated the scheduling feature which allows the user to determine when to turn individual outlets on or off. We will also showcase the current generation feature, which allows the user to instantly see what the latest generation in the area has been, and thus further informing them of the state of renewable generation.

#### 

**Measurements**

During the testing, we indirectly took various measurements including the test errors for each generation type's deep learning model. These are stored in text files that track test error over time for each model. We chose to represent our error as a test error so that we could compare how each model is performing against the other. Some energy sources have higher generation values, for example, the absolute error for wind was 21.4 MW and for nuclear was 34; however, the percent error for each was 3.64% and 1.02% respectively. This allows us to compare each model, evaluating each model’s performance against each other and determining if they are up to par or need more fine-tuning.

We also took a few binary measurements including successfully logging into the system and correctly displaying power forecasting. Logging in is exactly as it sounds; we used a sample email and password to log in, and we checked that the state got cached and that the proper tabs appeared upon login. We also checked to ensure that a new entry was added in our cloud database and that the person was recorded. For our backend functions we also wanted to monitor if the data was automatically pulled upon function call, and if the output csv was correctly formatted. Displaying power forecasting included checking for any anomalies involving the data being read in or how it is formatted. This ensures that we have a front end started and data is properly sent and displayed on the webpage. Lastly, we tested that the powerstrip could be successfully interacted with, sending it commands through the website to toggle power to its individual outlets and to schedule it to turn on at a certain time. The results for these tests are below:

| **ML Model** | **% Error** |
| --- | --- |
| Nuclear | 1.02 |
| Refuse | 1.42 |
| Landfill, Gas | 1.46 |
| Solar | 7.37 |
| Hydroelectric | 5.04 |
| Wood | 0.803 |
| Wind | 3.64 |

| **ML Model** | **Works Correctly?** |
| --- | --- |
| Automation of Data Pulling | Yes |
| Automation and Correct Format of Output CSV | Yes |

| **Website** | **Works Correctly?** |
| --- | --- |
| Globally Working | Yes |
| Log In System | Yes |
| Power Forecasting Display | Yes |
| Scheduling Works | Yes |
| Current Generation | Yes |

**Conclusions**

Based on the results of the machine learning test data, we have developed models that can accurately predict the electrical forecast. There is still small fine tuning to be achieved in solar, but generally the goal of 5% percent error may now be reachable. From the first prototype testing, we have reduced the percent error from around 66% to 7.37%. For the rest of the semester, we need to figure out the source of the error to achieve our goal of 5% and further develop models that have less error.

Switching gears to the web app, the first prototype initially passed the login test, scheduling test and the power forecasting display test. Our web app for the second prototype implemented scheduling, current generation display, and was hosted on a server instead of locally, all of which was tested and working perfectly. We demonstrated our complete control over the powerstrip, seeing as how all outlets could be toggled and scheduled through our web app. From the feedback we received on this end of our project, we will find more intelligent ways to display data from which the user could inform themselves and make decisions about energy consumption.

#### 

#### **Required Materials**

Hardware:

* Kasa Smart Outlet
* Server - (Lab PC)
* TP-Link AC1350 Wireless Router

Software:

* Python 3 Scripts
  + API querying and data formatting scripts
  + Individual Generation Model Scripts
  + Prototype Script to Unify Models
  + Automation Combinations of the models
  + Tunnel for Powerstrip connection
* Javascript Website Design
  + React
  + Node.JS
  + npm
  + FastAPI

#### **Set-Up**

Kasa Smart Plug: We will have the smart plug setup before testing occurs. This process is somewhat tedious and involves connection to a separate Wi-Fi router as well as interaction with the Kasa Smart Plug app.

Data Collection: To begin the data collection process we obtained usernames, passwords, and API keys to update the training CSV file that contains both weather data and power data for ~5-minute intervals. The weather API is <https://www.visualcrossing.com>, while the power data is from <https://www.iso-ne.com>. Similarly, we use these authentications to pull testing (future) data from the API’s.

ML: We will run our projections on this testing data. Once we have updated the data, we will retrain models with the latest information. We then pass these completed models to the prototype function. This function takes in the test data set generated previously and the completed models and calculates the next couple of days of predictions based on these models. The script then writes these predictions to a CSV file energy predictions.

Website: We read these new energy predictions and generate visualizations based on these outputs. When the CSV file energy prediction is pushed to GitHub, the website displays the predicted generation for hydro, solar, nuclear, and wind in the first line graph. The second line graph displays the predicted generation for refuse and wood. The technologies were divided into two line graphs because the need for refuse and wood is much smaller, therefore they would not be visible on one line graph. The user can also log into the website to control the powerstrip, where individual outlets could be toggled and scheduled.

#### **Pre-Testing Setup Procedure**

Smart Plug Side:

1. Connect the Kasa Smart Strip to the Kasa App and choose the MOTODEE2 network.
2. Make sure you have a computer at home connected to the same network.
3. Install ngrok on that same computer: <https://ngrok.com/downloads/>
4. Run ngrok with this command:

ngrok http http://localhost:8080 --url="fast-kid-sterling.ngrok-free.app"

1. Start the powerstrip server with this command:

uvicorn app:app --port 8080 --reload

Machine Learning Side:

1. A CSV File with Training Data from 2021 to the Present has already been created. This file is Auto\_Combine.csv

Website Side (Developer) :

1. Install all necessary dependencies: Node.js, npm, Python3, python3-venv, python-kasa, FastAPI
2. Change the directory to the paper-dashboard
   1. npm install
   2. npm start

Website Side (User):

1. Visit [www.ecostripsolutions.com](http://www.ecostripsolutions.com)
2. Login to the website or create an account.
3. Use website.

#### **Testing Procedure**

Start with ML:

1. Run the total backend function.

Website:

1. Login to the website using credentials.
2. Get the latest projections.
3. Display the latest data projections.
4. Turn the outlet on and off.
5. Demonstrate scheduling functionality.

#### **Measurable Criteria**

The criteria for successful running and output are as follows:

* Successful in gathering data.
* Calculating error for individual ML models to verify results with less than 10% error.
* Ensure that logging into the website successfully occurs.
* Verify data is successfully displayed.
* Verify the website can turn the smart outlet on and off.

#### 