SPARC: **S**chedule **P**ower **A**nd **R**educe **C**arbon

March 9, 2022
CS 329s | Stanford University

Nina Prakash , Kun Guo , Griffin Schillinger Tarpenning

App link:
https://share.streamlit.io/ninaprakash1/sparc-forecasting/main/frontend.py
Github link: https://github.com/ninaprakash1/sparc-forecasting

Problem Definition

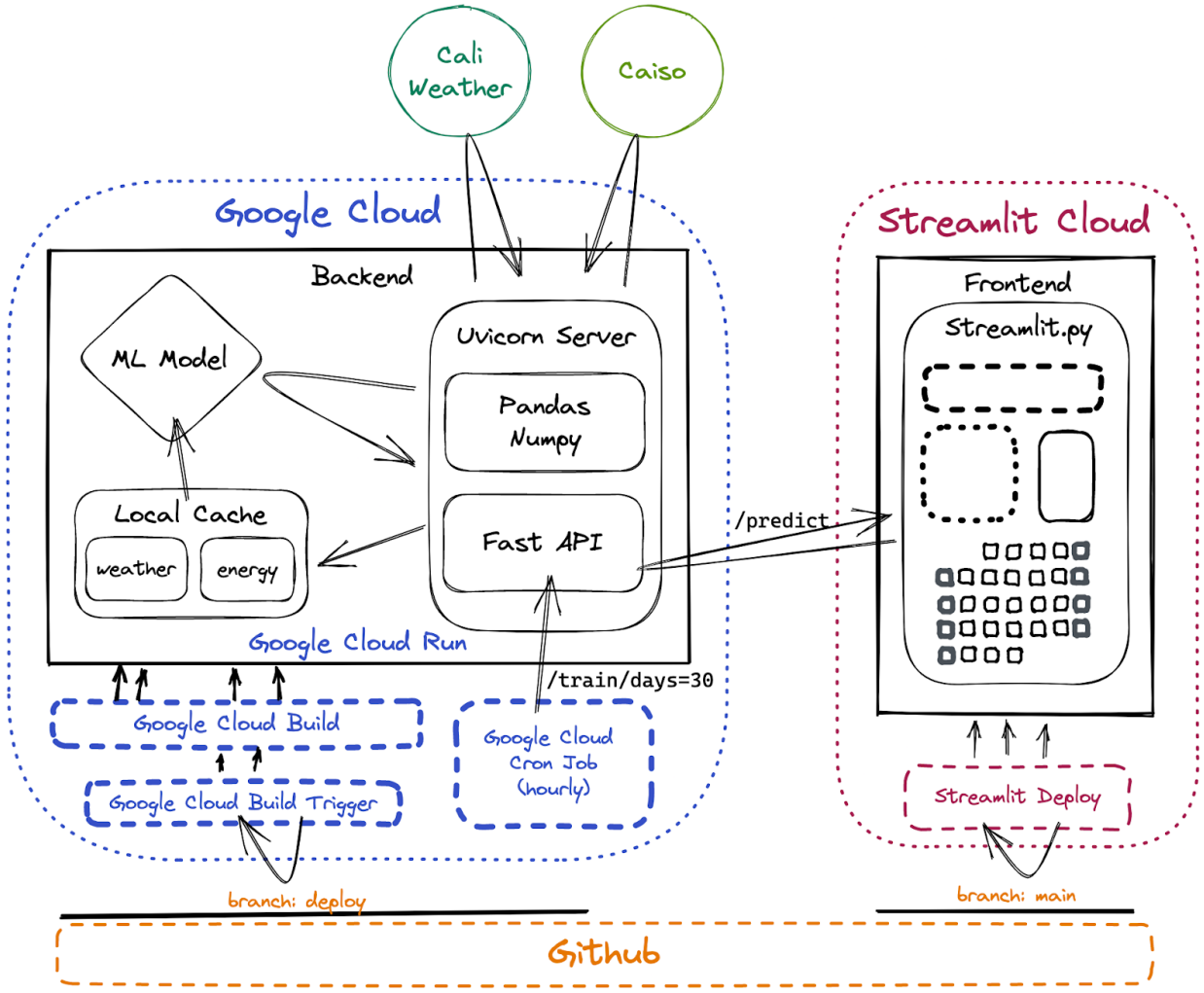
There is overwhelming evidence to suggest that anthropogenic sources of greenhouse gasses (GHGs) are the primary cause for climate change which results in extreme weather events and, in the extreme case, the mass extinction of species on earth. The reduction of GHG emissions, in particular of (CO_2), is essential to climate change mitigation.

One of the largest sources of CO_2 emissions in the US is electricity generation (32%), which is still dominated by fossil-fuel burning. Although carbon-zero energy, like hydro, solar and wind, are increasingly being used, they are very dependent on the time of day, week, year and the weather. In California, at times the grid produces nearly 100% carbon-zero energy, while at other times it is dominated by natural gas. Thus, there is a need for predictive energy mix statistics on the consumer side, which could enable a reduction in electricity-related carbon consumption. Using predictions of the CO_2 intensity of energy production, users might be encouraged to engage in energy intensive tasks only during times of cleaner energy production.

We propose a web-based application that allows a user to see the CO_2 emissions of daily activities such as charging an electric vehicle or running a load of laundry, in the future. Using high-fidelity time series data of power generation

sources plus weather data, an ensemble of time-series forecasting ML models predicts the energy supply mix 24 hours in advance for California.

System Design



In order to deliver accurate predictions for the energy generation mix, there are the following key components required, discussed below: data pipeline, ML model, deployment infrastructure, and UX. The way these components fit together are visualized above.

Data Pipeline

The data pipeline is a critical component of our project, ingesting, manipulation, and delivering data to the machine learning model for prediction. There are two major sources of data, the amount of energy produced by each different source of energy in California, and weather data, which are described in detail in the ML component section.

The data pipeline can be initiated by two different methods, train and inference. To train, the model requires 30 previous days of data. For the days that

have been previously requested from their respective API's, that data can be found in the local cache to improve latency, otherwise the external API's are hit and data cached. Generation mix and weather data are merged in a Pandas dataframe and then used to fit the SKForecast model. After training, the model is saved and re-loaded as the primary model used at inference time. A similar process is initiated to get a prediction, where data is ingested from the external sources if necessary, processed, fed into the ML model, and the output is returned.

Modeling

As mentioned, the model is loaded upon server startup from a saved file. However, due to the volatile nature of the prediction task, this model must be automatically retrained every hour in the cloud to ensure the most accurate predictions. Further discussion and justification of the model component is discussed in the ML component section.

Deployment

The deployment consists of two cloud providers, with Streamlit handling the frontend deployment and Google Cloud for the rest. The primary deployment mechanism is handled in cloudbuild.yaml which facilitates the building of the Python/Uvicorn/FastAPI server image. A separately defined Google Cloud Build Trigger sits in the cloud waiting for any changes to the remote deploy branch in github, upon detecting a change, and activates the cloudbuild.yaml steps. After pushing changes to deploy, the server is built from the Dockerfile in Google Cloud Build, pushed to Google Cloud Run, and deployed on our specified compute cluster. This process results in a zero-to-100 serverless deployment, resulting in two clear benefits and one drawback. During periods of inactivity, the compute cost is zero, while high traffic periods cause the compute cluster to scale up to handle more concurrent requests. However, this does come with the drawback of a slow response time for the very first request in a while (after 5 minutes). Lastly, the deployment also includes a Cloud Scheduler component, which makes an HTTPS POST request that triggers a retrain of the model every hour. For the frontend, pushing to main is automatically detected and redeployed using the wicked-fast optimized process Streamlit Cloud offers.

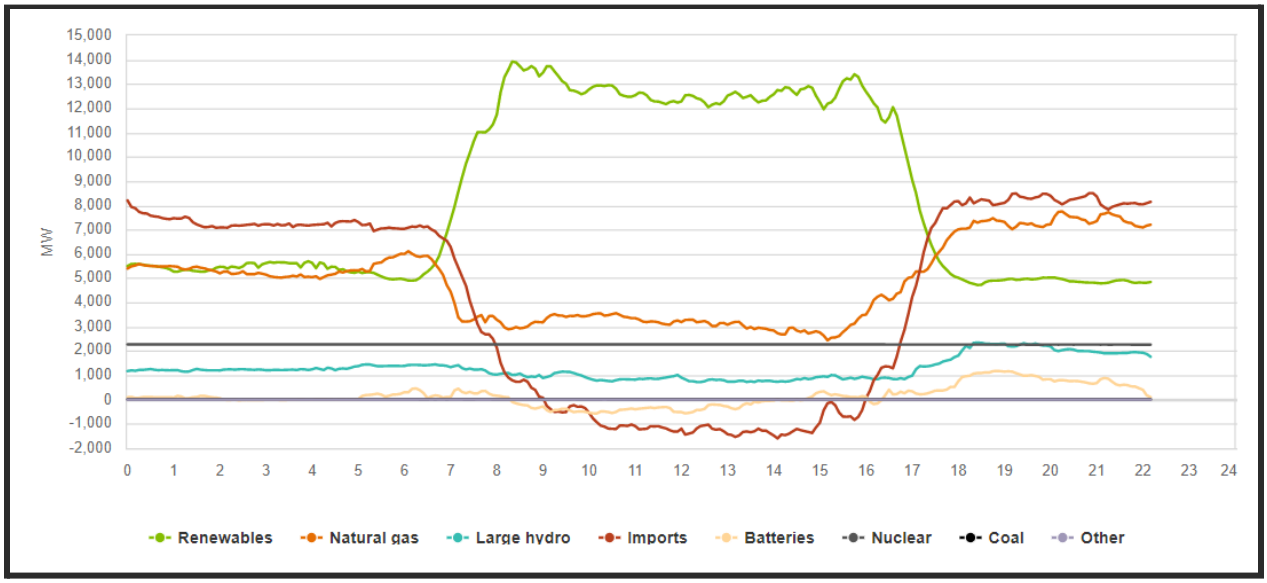
UX

The SPARC frontend is built entirely in Python, using a combination of streamlit and plotly libraries. Streamlit is an opinionated frontend library for Python that excels in displaying data, which results in a fast development cycle and stylish interface. However, the Plotly library is leveraged for custom graphs and charts. The UI/UX design aims to provide users an intuitive and interactive user experience.

Machine Learning Component

Dataset

We used historical energy mix data from the California Independent System Operator (CAISO) which manages California's grid, as well as weather data from World Weather Online (WWO) for both model training and inference. CAISO's energy mix data constitutes 12 energy sources: Solar, Wind, Geothermal, Biomass, Biogas, Small Hydro, Large Hydro, Coal, Nuclear, Batteries, Imports, and Natural Gas, sampled at 5 minute intervals. The weather data has features including temperature, UV Index, wind speed, cloud cover index, humidity and precipitation sampled at every hour. We resample the weather data to match the 5-minute frequency of the energy mix data. Both datasets are open source and can be continuously retrieved using the CAISO and WWO APIs.



Example energy mix daily profile from CAISO.

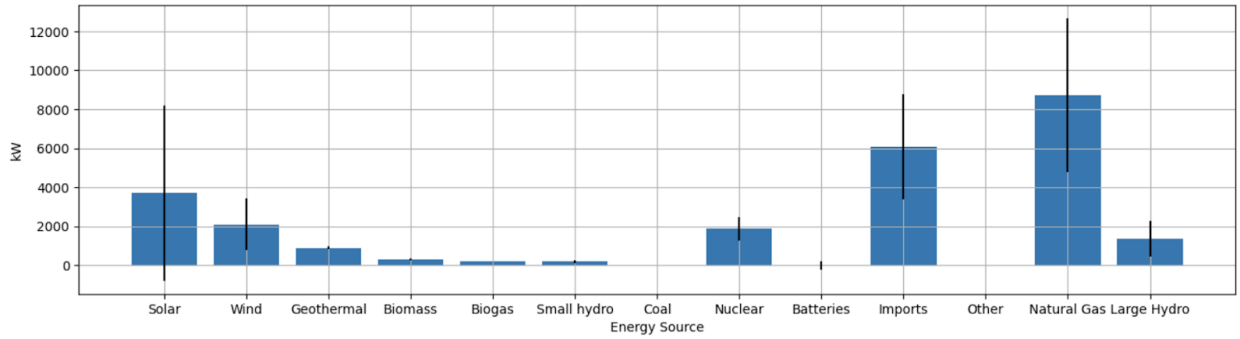
	date_time_hourly	date_time_5min	Solar	Wind	Geothermal	Biomass	Biogas	Small hydro	Coal	Nuclear	...	Imports	Other	Natural Gas	Large Hydro
0	2022-03-06 23:00:00	2022-03-06 23:35:00	-40	2887	816	267	203	160	9	2266	...	7431	0	6605	1680
1	2022-03-06 23:00:00	2022-03-06 23:40:00	-40	2838	815	265	203	160	9	2265	...	7391	0	6605	1677
2	2022-03-06 23:00:00	2022-03-06 23:45:00	-40	2788	816	268	203	161	8	2266	...	7327	0	6612	1677
3	2022-03-06 23:00:00	2022-03-06 23:50:00	-40	2739	815	269	203	162	9	2266	...	7233	0	6586	1675
9	2022-03-07 00:00:00	2022-03-07 00:25:00	-41	2383	816	268	203	164	9	2266	...	7655	0	6059	1542

Example energy mix data in MW at a 5-minute frequency from CAISO.

	tempC	uvIndex	WindGustKmph	cloudcover	humidity	precipMM
0	6	3	17	0	58	0.0
1	6	3	17	0	58	0.0
2	6	3	17	0	58	0.0
3	6	3	17	0	58	0.0
9	6	4	13	0	60	0.0

Example weather data returned from WWO API.

As shown in the figure below, some energy sources have high variance relative to daily averages and thus are more challenging to predict with high accuracy.



Average daily energy consumption of each energy source +/- 1 standard deviation.

During the model training stage, we used 70% /20% / 10% split for training, testing and evaluation. Certain energy sources, such as Coal, only provide a very small percentage of the overall energy mix (<0.1%) and are challenging to predict. Such energy sources are aggregated with other sources of the same kind (e.g. Coal + Natural Gas = Fossil Fuel). Different aggregation combinations are tested to balance between information granularity presented to end users and model performance.

Model Iteration

We experimented with a variety of machine learning model architectures, including tree-based, neural network, recurrent neural network, NBEATS and transformers, as well as predicting at different time intervals from 5 min to every hour, all with the focus to reduce mean absolute percentage error (MAPE). There are several observations through initial model exploration and back testing. First of all, regardless of the models, we find MAPE increases with time as we increase the length of the forecast horizon from the end of training data. Secondly, more complex models, such as NBEATS and transformers, outperform models with simpler

architectures by 5% - 10% in MAPE. However, hyperparameter tuning is necessary to achieve such uplift, which takes significant time and compute resources. We also experimented with predicting fractions of energy sources, but found it less accurate than predicting absolute kW numbers. Last but not least, we find it difficult for the models to converge at 5 min intervals while predictions are much more robust at 1 hour intervals. We concluded that model re-training on the most recent data is more important than the model architecture and hyperparameter tuning. In addition, we decide to make hourly predictions instead of finer increments for more robust results. Therefore, we chose to use the Skforecast library with XGBoost regressors for its simplicity and fast training/inference time. A series of models are independently trained for aggregated energy sources as described in table below. A total of 7 models are independently trained and tested.

Summary of variables included in each model.

Model Number	**Features in Raw Generation Mix from CAISO**	**Corresponding Aggregate Variable**
1	Solar	“Solar”
2	Wind	“Wind”
3	Small Hydro, Large Hydro	“Hydro”
4	Geothermal, Biomass, Biogas, Nuclear	“Other Renewables”
5	Batteries	“Batteries”
6	Coal, Natural Gas	“Fossil Fuel”
7	Imports, Other	“Other”

System Evaluation:

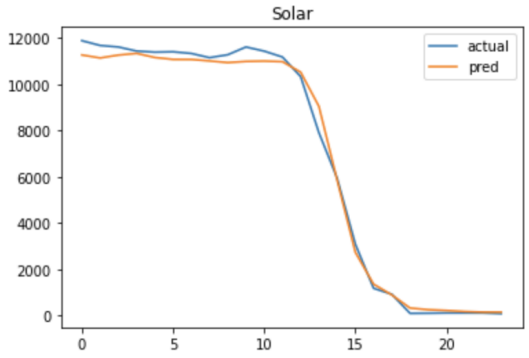
Prediction Accuracy

Our end goal is to make hourly CO_2 emission predictions and rank the emissions on a relative scale from low to high carbon intensity for a particular activity. We think it is intuitive to use MAPE (percentage error with respect to actual output) to evaluate the model performance in both training and testing stages, where 0 means perfect prediction. Table x, which summarizes the MAPE on the validation set, indicates that the model on renewable_other (geothermal, biomass, biogas, and nuclear aggregated) performs the best, followed by the model trained on fossil fuels (coal and natural gas). The model trained on batteries and other (imports and other) perform the worst, which is as expected because they are some of the most stochastic in the grid data. Top of the prediction error in energy mix, another source of error in final CO2 prediction is for the 'Other' category. California imports a significant amount of electricity from other states, in which the fraction of generation mix is not as transparent as the in-state and average fractions are assumed for time series CO2 emission calculations.

MAPE on validation set per model, predicting for Mar 8, 2022 (24 hours).

	total	renewable	renewable_other	fossil_fuel	Solar	Batteries	Hydro	other
count	24.000000	24.000000	24.000000	24.000000	24.000000	24.000000	24.000000	24.000000
mean	0.064140	0.447744	0.007154	0.536680	2.453702	6.874174	0.659558	4.784668
std	0.019498	0.192641	0.000757	0.061302	2.859073	3.027711	0.017543	0.945156

Sample 24 hours prediction for Solar on validation set.



Development Efficiency

One of the primary goals of the system was to set up solid infrastructure that required as little maintenance as possible. This goal was achieved. After completing

the Google stack setup process, which was of middling difficulty, any member of the team was able to trigger remote builds/deployments with a simple git push to the remote deploy. This, in addition to a trigger on the main branch for the Steamlit frontend, enabled the team to focus primarily on UX and model development, and not get bogged down in deployments and infrastructure. The automatic retraining cycle also prevented model drift over time, ensuring that no developer time was spent retraining and re-deploying models. One clear point of evidence this system was successful is that we had over 40 image builds on the backend side as a team, but never pushed an image manually the entire project!

Robustness

Robustness of system design was good but not excellent. Robustness was tracked by three indicators: max requests / scaling, uptime, and security. Because of the serverless infrastructure, scaling occurs seamlessly. Within 60 seconds of maxing out requests for one server instance, another server comes online and begins handling requests. This was tested up to 4 concurrent instances, but has a theoretical max of 100 instances for a max of 10,000 concurrent backend requests. The frontend, while opaque behind Streamlit's environment, also autoscales. Uptime was decent, as our specific Google Cloud Run setup ensured that new deployment servers were running and operational before hot-swapping the old server out. Thus, breaking changes never affected uptime on the backend. However, no staging environment meant there were times when the backend had runtime errors and hamstrung the frontend service. The Streamlit frontend was extremely quick to update (less than 5 seconds from push to fully deployed), allowing all changes to be immediately verified and fixed. Finally, the security of the application was a weak point. Using CORS middleware and exclusively POST requests on our FastAPI implementation, we ensured that all successful requests must originate from a whitelist of addresses, associated with the team and the frontend. However, these requests aren't verified past that initial check, with no authentication required to access prediction data beyond that Origin check.

Cost

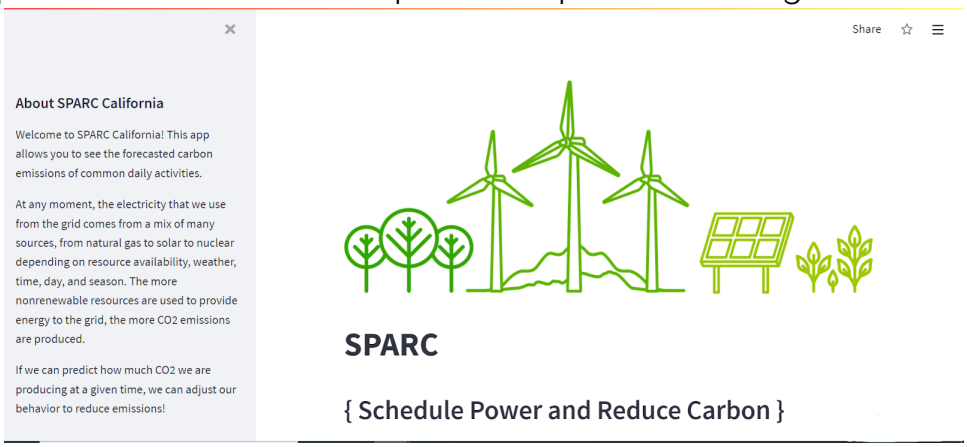
This is an area where our stack excels. Despite the hourly retraining, using four separate Google Cloud Services and an additional cloud provider in Streamlit, our total costs as of right now are expected to be less than 48 cents! Due to the generous free-tier of Google Cloud, and tolerance of the serverless environment, we remain well short of exhausting our $150 in cloud credits. Streamlit offers a free tier for education accounts, providing free service for the frontend.

Application Demonstration

We chose to develop a web application to interface with SPARC given the typical user of the application, who is expected to be any individual who cares about their carbon footprint and has access to an internet connection. A web application is well-suited to this type of user. Below is a set of screenshots that document the workflow for using SPARC.

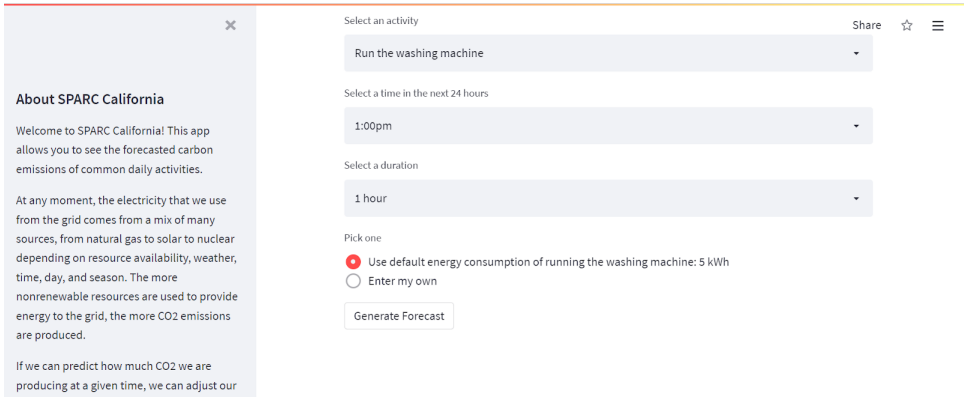
Step 1:

Go to the web application at https://share.streamlit.io/ninaprakash1/sparc-forecasting/main/frontend.py.



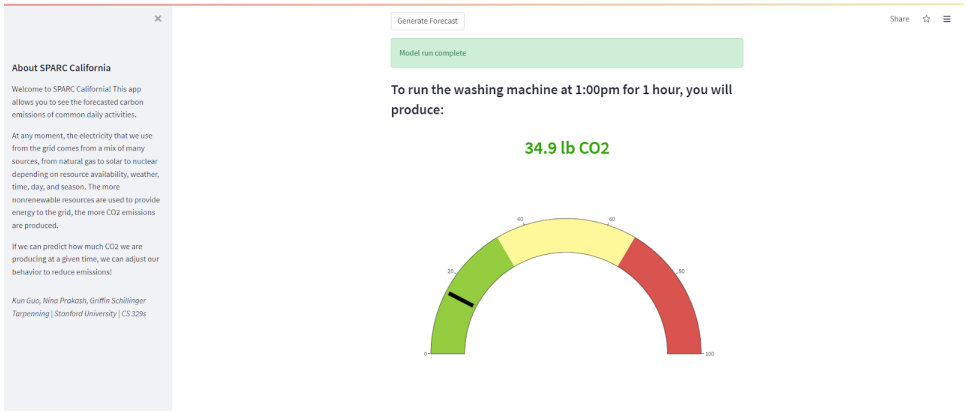
Step 2:

Select an activity, time, and duration from the dropdown menus and click "Generate Forecast".

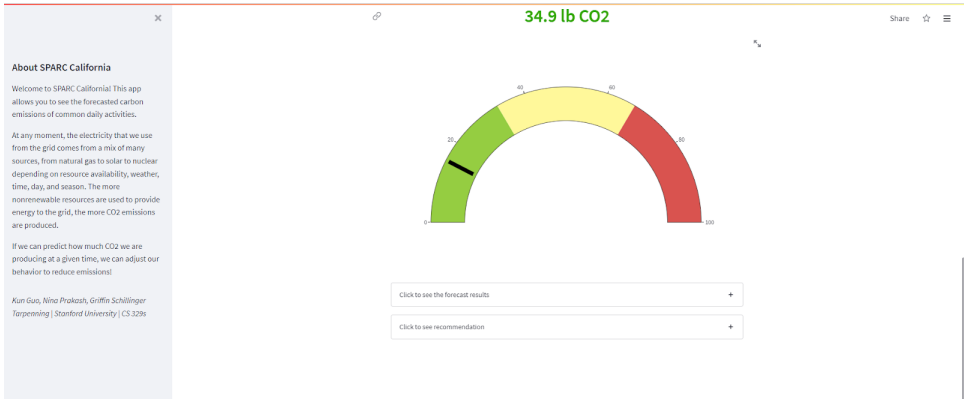


Step 3:

See the predicted carbon emissions.

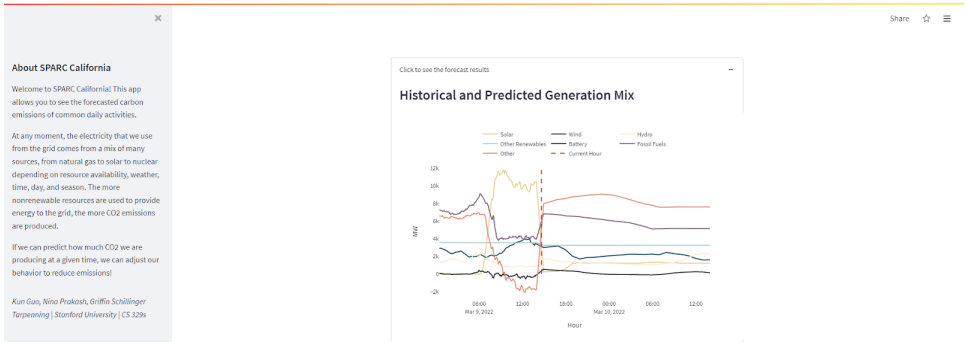


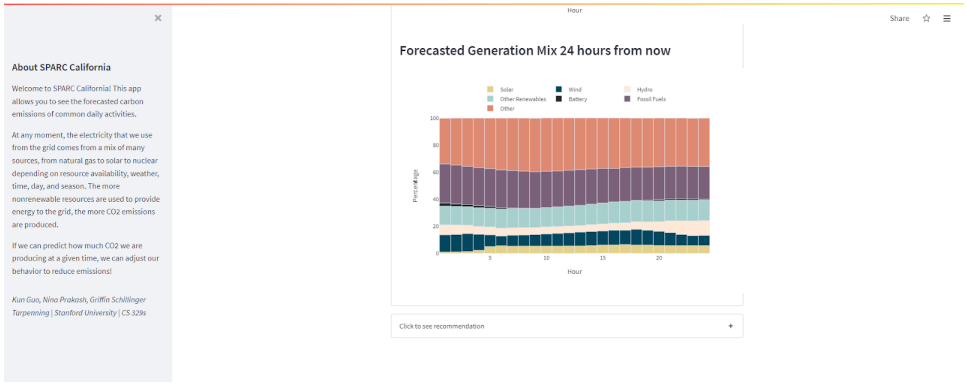
Below the gauge should also appear two dropdown menus with more details about the results.



Step 4:

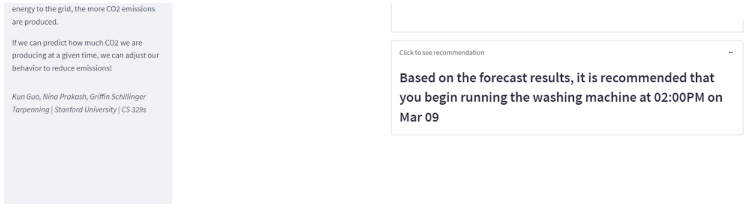
Open the first results dropdown to see more information about the results.





Step 5:

Open the second results dropdown to see a recommendation to reduce carbon emissions.



Our hope is that the user can use these dropdown menus to explore various options for timing and duration to see the estimated CO_2 emissions and adjust their behavior accordingly.

Reflection

Overall, our team had a very positive experience with this project from ideation, development to deployment. Streamlit is awesome! Google Cloud Run was surprisingly easy to work with once set up, and the Python tools were simple to get rolling. We were able to successfully set up a working pipeline relatively quickly including the frontend, model, and server, and iterate throughout the quarter as necessary. Each tool used including Streamlit, Google Cloud, FastAPI, Uvicorn and Skforecast served our purposes well and were able to integrate together smoothly. Between the three team members, we each took the lead separately on the backend, model, and frontend so were able to divide up the tasks and become point experts in each area.

Given more time and resources, we would have explored more established machine learning tool sets like MetaFlow for continuous iteration and development. This would have ensured continuous development of the ML model, rather than an ad hoc development style more akin to a data science task. In addition, we would have more rigorously checkpointed the iterative models and used more aggressive hyperparameter tuning, both to keep track of different aspects of progress as well as see a more visual representation of the different benefits of various models. Here perhaps the use of an external tool like Weights and Biases might have been useful. This could also have facilitated better collaboration between team members during model iteration and broken down the relatively large barrier to entry to developing the ML model itself. Finally, there is a pretty significant argument to be made that this tool is actually best suited for a mobile environment (discussed below), so with unlimited time that would have been implemented.

As of now, the plan is to leave the project within the scope of the course. However, we have a ton of features that would greatly expand the scope and value of the project. The most important of which is the granularity of our predictions. Currently, we are using hour or half-hour long chunks of time when making predictions, but the generation supply data has a 5-minute fidelity and thus it would be possible to make predictions on that scale as well. This, in combination with a more robust frontend UX, which might explore the use of extensive sliders or a calendar widget, would bring both increased confidence in prediction as well as better ease of use. Additionally, we are currently only looking at data from California, but many states (and countries) with grids that have fluctuating CO_2 content would benefit from this type of analysis. Moreover, integrations like a notification system to let users know the optimal time to charge devices, at that very moment, would increase the likelihood of users actually using the prediction to lower their carbon footprint. In a similar vein, a quick confirmation button asking if a user actually changed their behavior following a suggestion would naturally lead to a tracker or

scorecard, adding up all of the carbon saved by the user. Gamification or a “share how much you saved” button might then increase its virality and get our little program into the hands of more users.

Broader Impacts

The intended user of this application is any environmentally-minded individual making everyday decisions related to electricity usage in the household. However, it is not restricted to this use case. For example, understanding day-ahead carbon impact can be used at a much-larger scale by businesses with much larger workloads than the everyday individual. By using the API directly (which is possible from whitelisted Origins), businesses can use this information to schedule batched computation workloads to help reach corporate emissions goals that put pressure on businesses to decrease their carbon footprint. In addition, forecasting generation mix using machine learning is a very little-studied problem; typically analytics in the electricity sector focuses on forecasting demand. The ability to predict day-ahead grid mix has potential to automate decision making across the electricity sector from generators placing bids in the market to load-serving entities that purchase the electricity to the Independent System Operators that manage the grid.

However, it is possible that a bad actor could use this data to increase their carbon footprint rather than decrease it. It is also possible that a bad actor with access to the API could induce extensive computation through some form of DDOS attack. To mitigate this problem, we put CORS in place to prevent non-app-originated queries from being able to hit our endpoint with POST requests. However, we currently do not have any safeguards against users using this app to intentionally increase their carbon footprint. One possible mitigation is to implement the gamification mentioned above. Gamification incentivizes users to schedule their energy by showing off to their friends and family how “green” they are. Although not completely solving the problem, this would likely increase the proportion of users doing good, counteracting the impact of bad actors.

References

Documentation and Tutorials

1. Skforecast: https://www.cienciadedatos.net/documentos/py27-time-series-forecasting-python-scikitlearn.html
2. Google Cloud: https://cloud.google.com/docs
3. Streamlit: https://docs.streamlit.io/
4. Data scraping from CAISO: https://github.com/tamu-engineering-research/COVID-EMDA/tree/master/parser

Data Sources

5. CAISO (energy data, carbon intensity of natural gas): http://www.caiso.com/outlook.html, https://www.caiso.com/Documents/GreenhouseGasEmissionsTracking-Methodology.pdf
6. World Weather API (weather data): https://www.worldweatheronline.com/developer/api/
7. EPA (carbon intensity of imports): https://www.epa.gov/sites/default/files/2020-12/documents/power_plants_2017_industrial_profile_updated_2020.pdf
8. Energy usage data for common activities:
    a. Charging an electric vehicle: https://rmi.org/electric-vehicle-charging-for-dummies/#:~:text=If%20you%20just%20plug%20an,7%20kW%20and%2019%20kW
    b. Run a washing machine: https://blog.arcadia.com/electricity-costs-10-key-household-products/#:~:text=An%20average%20cycle%20for%20a,to%20run%20for%2030%20minutes
    c. Run the dryer: https://majorenergy.com/how-much-electricity-does-a-dryer-use/#:~:text=Dryers%20are%20typically%20somewhere%20in,and%206%20kilowatts%20an%20hour.
    d. Take a hot shower: https://greengeekblog.com/tools/shower-cost-calculator/#:~:text=An%20average%208.2%20minute%20shower,energy%20to%20heat%20our%20water.
    e. Run central AC: https://energyusecalculator.com/electricity_centralac.htm

f. Run a space heater: https://experthomereport.com/do-space-heaters-use-a-lot-of-electricity/#:~:text=Although%20amounts%20vary%20per%20space,but%20in%20hours%20of%20kilowatt.
g. Run a hot water heater: https://www.directenergy.com/learning-center/how-much-energy-water-heater-use#:~:text=Typically%2C%20a%20hot%20water%20heater,month%2C%20or%20%24438%20per%20year.
h. Run the dishwasher: https://www.inspirecleanenergy.com/blog/sustainable-living/cost-to-run-dishwasher#:~:text=Most%20dishwashers%20use%20an%20average,hours%20(kWh)%20of%20electricity.
i. Watch TV: https://letsavelectricity.com/how-much-power-does-a-tv-use-in-an-hour/