

# OpenFIDO

## Testing and Validation Technical Report

David P. Chassin and Duncan Ragsdale  
*SLAC National Accelerator Laboratory*  
*Menlo Park, California (USA)*

EPC-17-047 Deliverable 4.1.1

5 December 2022

### Introduction

In California, utilities, customers, consulting engineers and regulators need to exchange power system data and models to verify system resource adequacy, evaluate resilience strategies, validate distributed energy resource plans, process permits for distributed energy resource interconnection, and demonstrate compliance when integrating resources in distribution systems. Data and model exchange between organizations using different software tools can be cumbersome, slow and error prone, which raises barriers to fast and effective renewable energy and storage resource integration, curbs the growth of these important climate change mitigation technologies, and limits on how quickly California can reach its climate change response goals.

OpenFIDO provides a framework for inter-organizational data and model interchange, data and model synthesis, and system performance analysis between different power systems tools such as CYME, GridLAB-D, and GLOW. OpenFIDO can be used to (1) collect data from a wide variety of open sources, private databases, and commercial products; (2) transfer model and telemetry data between various tools that are part of the suite of tools widely used by utilities, distributed energy resource (DER) engineers and regulators in California; and (3) provide the ability to create permanently available reproducible results.

OpenFIDO is designed for utility planners and grid researchers that need a tool and data integration framework to quickly move data from one application to another as part of their engineering, planning, and review activities. OpenFIDO also supports emerging user groups such as DER system integrators and aggregators that use multiple tools to analyze the grid impacts of DERs, as well as governments and agencies that use these models in both their oversight role and identifying opportunities for clean energy deployments.

This report documents the methodology and implementation of the deployed OpenFIDO platform (as reported at [OpenFIDO Application Status](#)). Note that this report does not cover the validation of the pipelines themselves. OpenFIDO pipelines are validated using idiosyncratic

methods, and are outside the scope of the OpenFIDO platform itself. Information on pipeline testing and validation is available from the GitHub repositories of the individual pipelines (see [OpenFIDO Pipeline Status](#) for details). A summary of the pipeline testing and validation status is provided in Appendix B.

## Methodology

In the course of developing OpenFIDO, SLAC developed and implemented three testing protocols to ensure consistent and full-function application usability.

The primary tests provide for code is coverage testing, which is integrated into our CI/CD pipelines for each microservice used to build the OpenFIDO application. The objective of coverage testing is to verify and validate the functionality of the codebase itself. Coverage testing not only ensures that the code is programmatically sound for deployments, but also that it meets rigorous style requirements and coding conventions, providing better maintainability and consistent standards throughout the project.

For the platform backend, *pylint* and *black* are used to verify programming correctness and style, respectively. Code coverage is provided by *pytest*. For the platform front-end, *eslint* and *jest* are used for correctness/style and code coverage, respectively, while *jest-puppeteer* is used to perform integration testing (discussed below).

A secondary set of tests are used for integration testing. The purpose of these tests is to ensure code changes during the development process do not break the user functionality of the application. These tests run an automated series of actions, which simulate actions that can be taken by users of the application.

The final set of tests is a full deployment of the application onto a set of staging servers on AWS, which exactly replicates the production deployment and ensures that all the required microservices are fully functional and available.

## Implementation

The testing process is integrated automatically throughout the entire development cycle of OpenFIDO. The code coverage, integration, and linting tests occur in two distinct phases.

The first phase occurs while developers actively write their code, with *eslint* and *black* providing real-time feedback on programming style and conventions, providing automatic corrections wherever possible, and ensuring consistent implementation of code and features. For the frontend, this also provides an additional layer of continuity for the code by enforcing type-checking of any data being passed through its functions, thus ensuring that functions are properly structured and maintained.

During development, contributors can also manually run the test suites to confirm that their changes do not break the functional flow of the application, for example by calling `npm test` or `pytest` to run their respective testing and validation protocols.

Prior to submitting code changes, contributors use *jest-puppeteer* to conduct integration testing and validation. Integration requires running all the respective microservices used within OpenFIDO and emulating their deployed environments locally. This includes verifying the functionality of and access to the database, the frontend client service, the authentication service, the workflow service, and the application service.

The integration tests covers the following features from OpenFIDO:

- User authentication (login and logout)
- Updating and editing user account information (add, edit, and delete)
- Creating a user organization (for administrators only)
- Creating and running pipelines (for users only)
- Removing a user organizations and pipelines

The second phase of testing and validation occurs when code changes are pushed to a branch of their respective repositories.

Code pushed to a branch on one of the repositories triggers an automatic run of github actions workflows, which conduct a final validation run of the code by running the code coverage and linting tests again. Code that fails these automatic tests is prevented from being merged with the main branch until it passes the required tests.

Once all prior stages of testing are cleared and the code is merged into the main branch, an automatic deployment to the development servers is initiated. Should the code successfully deploy to the development servers and be verified to run without breaking their operation, then the final production release can be manually approved by the project administrators.

A real-time summary of the application status is present at the OpenFIDO GitHub repository for the open-source distribution, as illustrated in Figure 1. The App Service provides the API for the main application services, including the user account management, pipeline management, pipeline execution, and database operations. The Auth Service provides the user authentication services including login, logout, and access control. It also handles Cyber Security protections, including validation of an internal API token to prevent unauthorized connection to OpenFIDO containers. The Workflow Services provide pipeline workflow management services, such as job sequencing and parallelization support. The Client Services provide the user interface and user experience (UI/UX) services including web page composition and delivery.

Each of the application stages listed in Figure 1 are designed to add additional layers of validation before making a deployment to the production environment (The official, published website). As each service is maintained in its own repository, it becomes possible to make changes and update only the relevant portion of the application, without redeploying or changing unaffected microservices.

### Test-build

This stage is solely designed to validate any updates to the codebase, and to provide reasonable assurance that any code changes should be able to successfully deploy. Each stage runs in a clean github runner (a clean virtual machine), and by running the initial testing, linting, and build steps, it informs the developer if they need to make further changes before attempting to deploy their updates. Running in a clean environment also allows the developer to catch if any additional dependencies were missed and need to be included in the build process.













### Deploy Staging

Once the build has been tested, this stage deploys the changes to a developer version of the application environment. (e.g. <https://app-staging.openfido.org>) The purpose of this step is to ensure that NO breaking changes make it to the production environment. The developer version of the application is a live deployment using all the same resources as the production version. Successful deployment to the staging environment gives a developer the highest degree of confidence that their code changes will be able to successfully deploy in production.

### Deploy Production

This is the final stage that runs when all prior stages have been completed, validated, and the new release of the application has been published to github. This stage will update the deployed OpenFIDO application at <https://app.openfido.org>.

## Application Status

App Service	Auth Service	Workflow Service	Client Service
			
			
			

**Figure 1 - Example real-time testing and validation status page (see [OpenFIDO WebSite](#) for current status) with a failure indicated.**

# Results

Each of the respective services' test results are listed below. Python *lint* quality scores are calculated using the formula<sup>1</sup>:

$$100.0 - 100 \times \frac{5 \times \text{Frequency of convention errors}}{\text{Number of statements}}$$

Code coverage is calculated by dividing the total number of tested statements by the total number of overall statements.

## Application Service

- Unit tests currently run with a 100% success rate and greater than 97% code coverage
- Submitted code quality does not fall below a 90% rating on *pylint*

## Authentication Service

- Unit tests currently un with a 100% success rate and greater than 92% code coverage
- Submitted code quality does not fall below a 90% rating on *pylint*

## Workflow Service

- Unit currently un with a 100% success rate and greater than 99% code coverage
- Submitted code quality does not fall below a 90% rating on *pylint*

## Client Service

- There are no style/linting/type-checking issues with the Client Service
- 100% success rate using integration testing
- Development time constraints greatly limited unit-testing development

Successful automated deployments and updates to both development and production servers are maintained continuously through this testing and validation system.

---

<sup>1</sup> Source: [https://pylint.pycqa.org/en/latest/user\\_guide/checkers/features.html](https://pylint.pycqa.org/en/latest/user_guide/checkers/features.html). Note that we are converting from "over 10" to percent score.

# Appendix A - Test Results

**Table A.1 - GitHub Actions**

Component	Test	Result	Disposition
<a href="#">App</a>	211 unit tests	>97% coverage	100% tests passing. See Table A.2
	style	100/100 ok	
	lint	>90% score	See Table A.3
<a href="#">Auth</a>	136 unit tests	>92% coverage	100% tests passing. See Table A.4
	style	100% ok	
	lint	>90% score	See Table A.5
<a href="#">Workflow</a>	160 unit tests	>99% coverage	100% tests passing. See Table A.6
	style	100% ok	
	lint	>90% score	See Table A.7
<a href="#">Client</a>	jest-puppeteer	No issues	

**Table A.2 - App Missing Coverage**

File location	Missed Statements	Proposed Remedy
app/pipelines/services.py	18/254 uncovered	Add additional tests for uncovered statements.
app/workflows/routes.py	4/134 uncovered	Add additional tests for uncovered statements.
app/workflows/services.py	7/279 uncovered	Add additional tests for uncovered statements.

**Table A.3 - App Lint Issues**

Test Result	Issue Description	Proposed Remedy
92.6% lint score	Code Style and Formatting	Run the <i>lint</i> command, and review each potential style or formatting issue listed against the linter's recommended fix.

**Table A.4 - Auth Missing Coverage**

File location	Missed Statements	Proposed Remedy
app/auth.py	32/260 uncovered	Add additional tests for

		uncovered statements.
app/mail.py	13/52 uncovered	Add additional tests for uncovered statements.
app/org.py	14/221 uncovered	Add additional tests for uncovered statements.
app/services.py	3/211 uncovered	Add additional tests for uncovered statements.

**Table A.5 - Auth Lint Issues**

Test Result	Issue Description	Proposed Remedy
90.4% lint score	Code Style and Formatting	Run the <i>lint</i> command, and review each potential style or formatting issue listed against the linter's recommended fix.

**Table A.6 - Workflow Missing Coverage**

File location	Missed Statements	Proposed Remedy
app/workflows/queries.py	2/52 uncovered	Add additional tests for uncovered statements.

**Table A.7 - Workflow Lint Issues**

Test Result	Issue Description	Proposed Remedy
92.2% lint score	Code Style and Formatting	Run the <i>lint</i> command, and review each potential style or formatting issue listed against the linter's recommended fix.

## Appendix B - Pipeline Testing and Validation

OpenFIDO pipelines are validated using idiosyncratic methods, and are outside the scope of the OpenFIDO platform itself. Information on pipeline testing and validation is available from the GitHub repositories of the individual pipelines (see [OpenFIDO Pipeline Status](#) for details). Table B.1 provides a summary of the current status of the pipeline validation.

**Table B.1: Pipeline testing/validation summary**

Pipeline Name	Validation status	Cause/Action
Tariff design	Failed	HiPAS GridLAB-D template failed
Loadshape	Ok	
Weather	Ok	
HiPAS GridLAB-D	Ok	
Census	Ok	
Resilience	Failed	HiPAS GridLAB-D template failed
Hosting capacity	Ok	
Electrification	Ok	
Address	Ok	
Cyme Extract	Ok	

### Tariff Design

The tariff design use-case was evaluated on a number of California IOU tariffs as below:

**Table B.2: Tariff design rates evaluated**

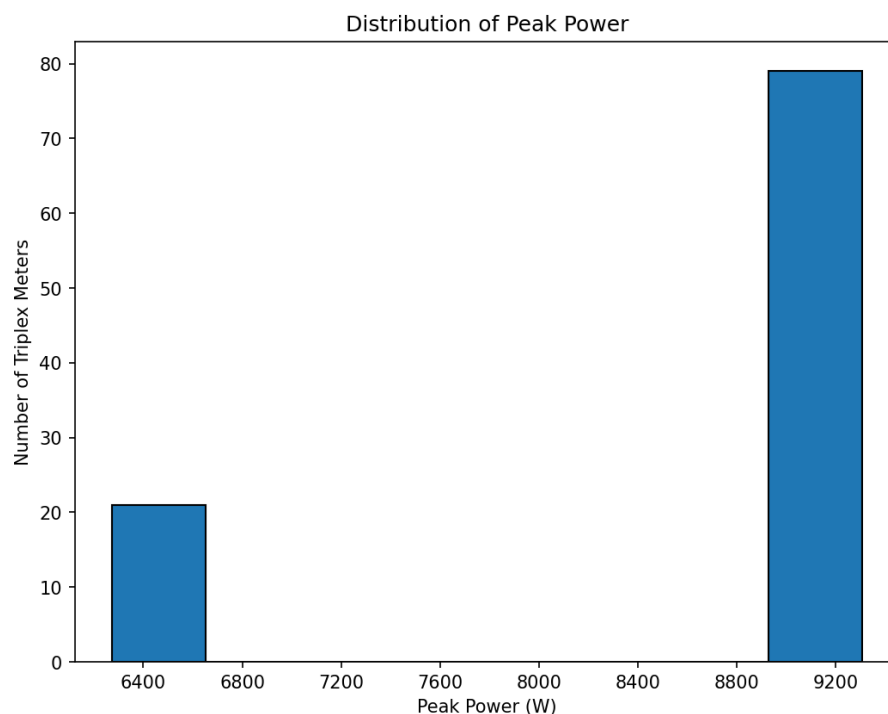
Utility	Sector	Name	Type	Region	Block Rate
PG&E	Residential	E-TOU-C3	TOU	Region R	Yes
PG&E	Residential	E7-RES-TOU	TOU	Region P	Yes
PG&E	Residential	E-1	TOU	Region T	Yes
PG&E	Residential	E-7	TOU	Region P	Yes
PG&E	Residential	E-6	TOU	Region Z	Yes



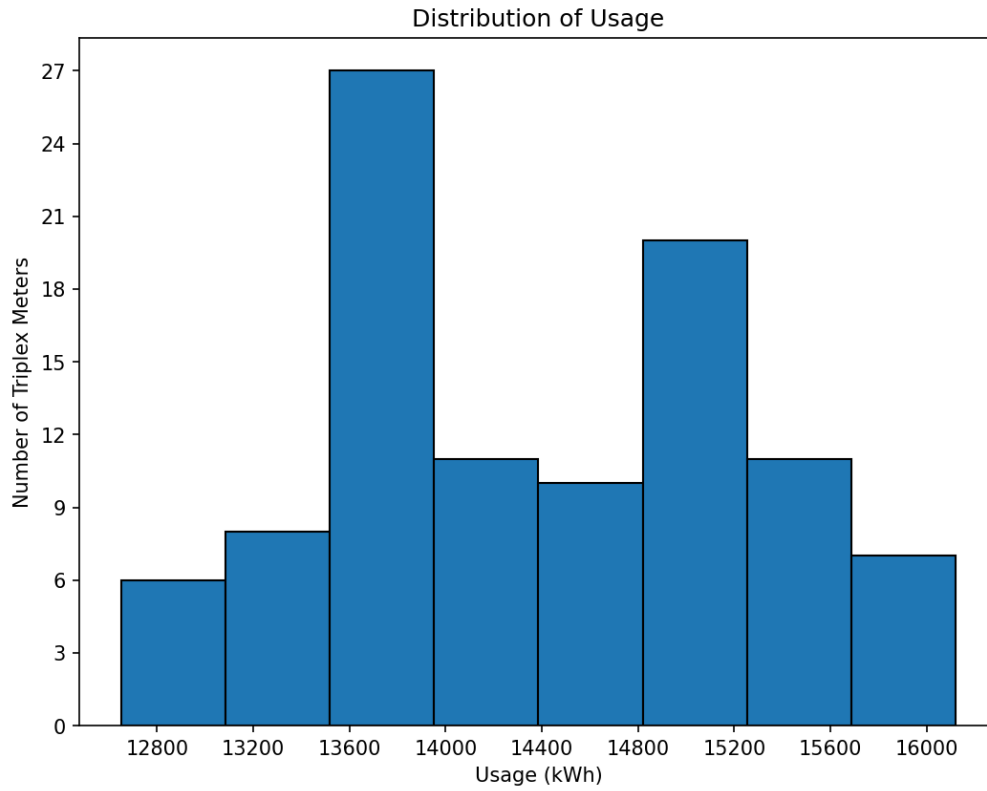
**Table B.2: Tariff design rates evaluated**

Utility	Sector	Name	Type	Region	Block Rate
SDG&E	Residential	DR	TOU	N/A	Yes
SDG&E	Residential	EV-TOU-2	TOU	N/A	Yes
SDG&E	Residential	DR	TOU	Mountain	Yes
SDG&E	Residential	TOU-D-B	TOU	N/A	Yes
SDG&E	Residential	TOU-D-TEV	TOU	Region 15	Yes

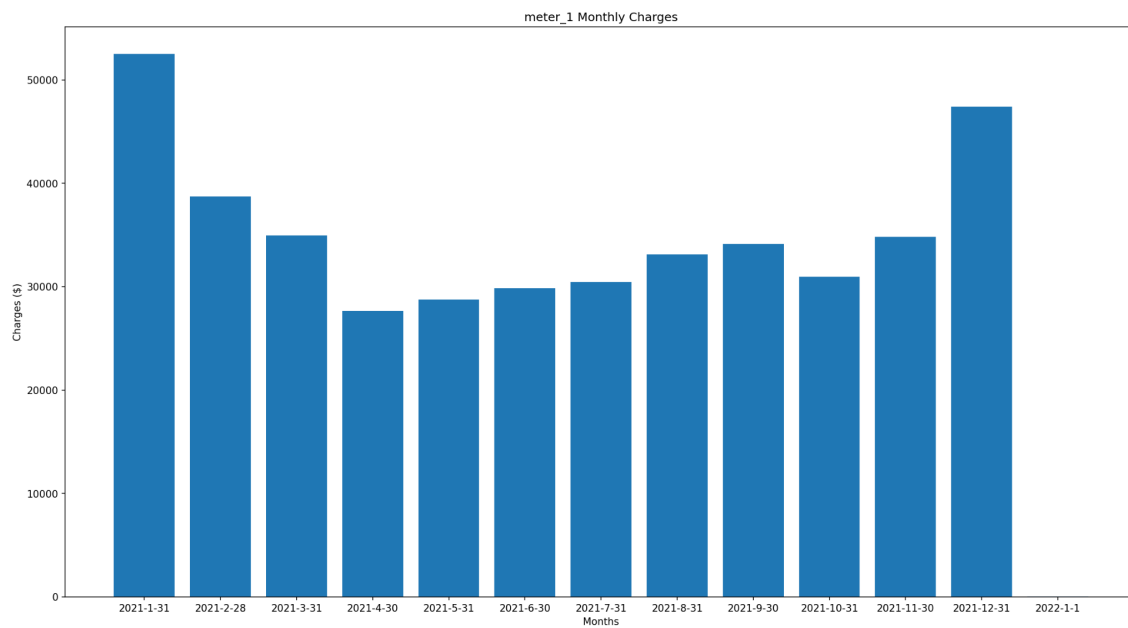
An annual tariff results on IEEE-13 model are shown in Figure B.1 to B.7, and include peak power, total energy, and cost distributions.



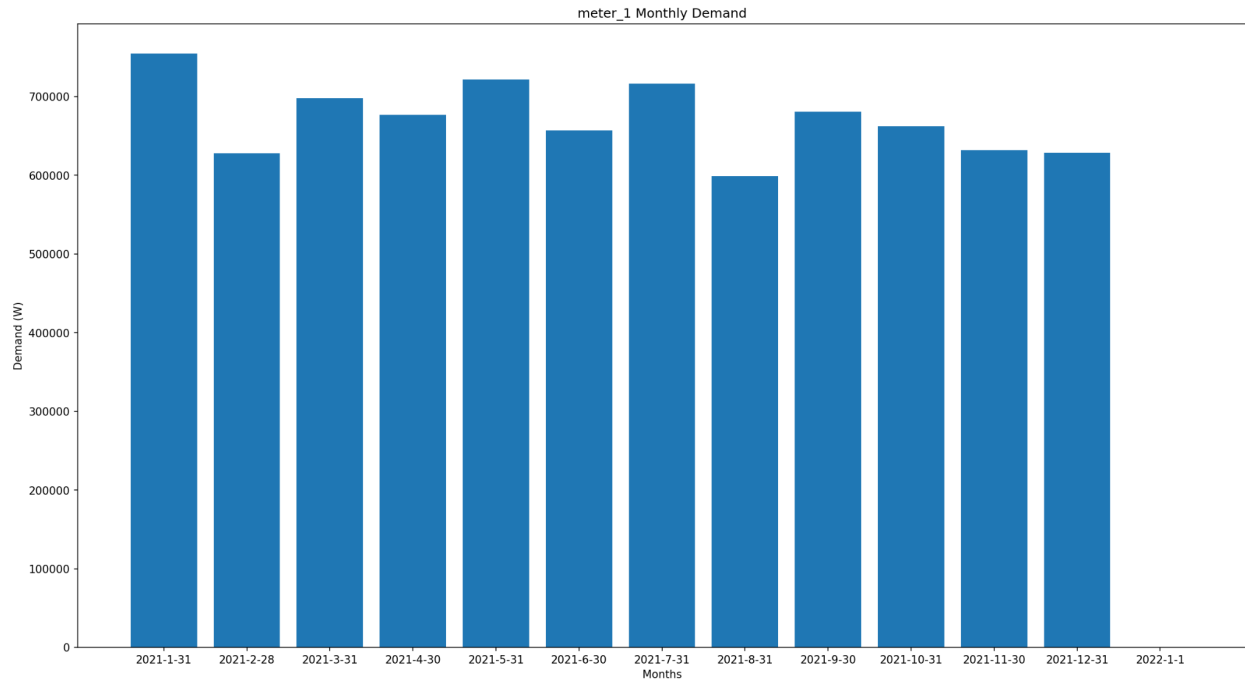
**Figure B.1: Distribution of peak power for tariff design validation**



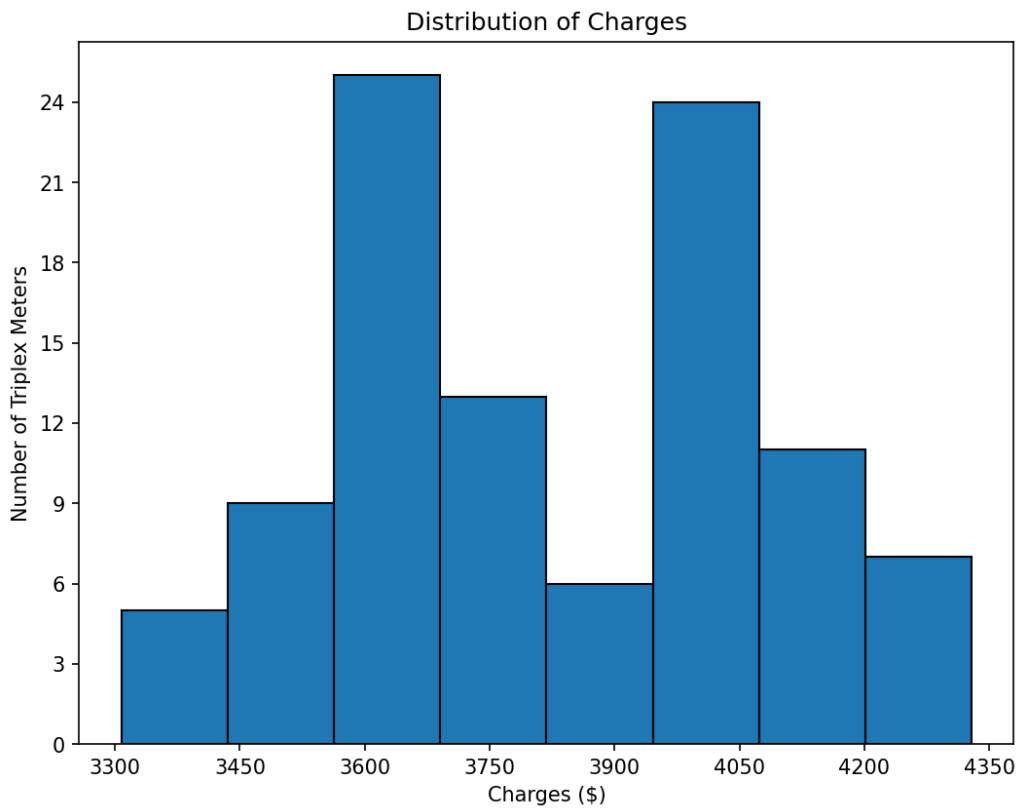
**Figure B.2: Distribution of energy consumption for IEEE-13 tariff design**



**Figure B.3: Monthly charges for IEEE-13 Meter 1 tariff design**



**Figure B.4: Monthly charges for IEEE-13 Meter 1 tariff design**



**Figure B.5: Distribution of annual charges on IEEE-13 tariff design**

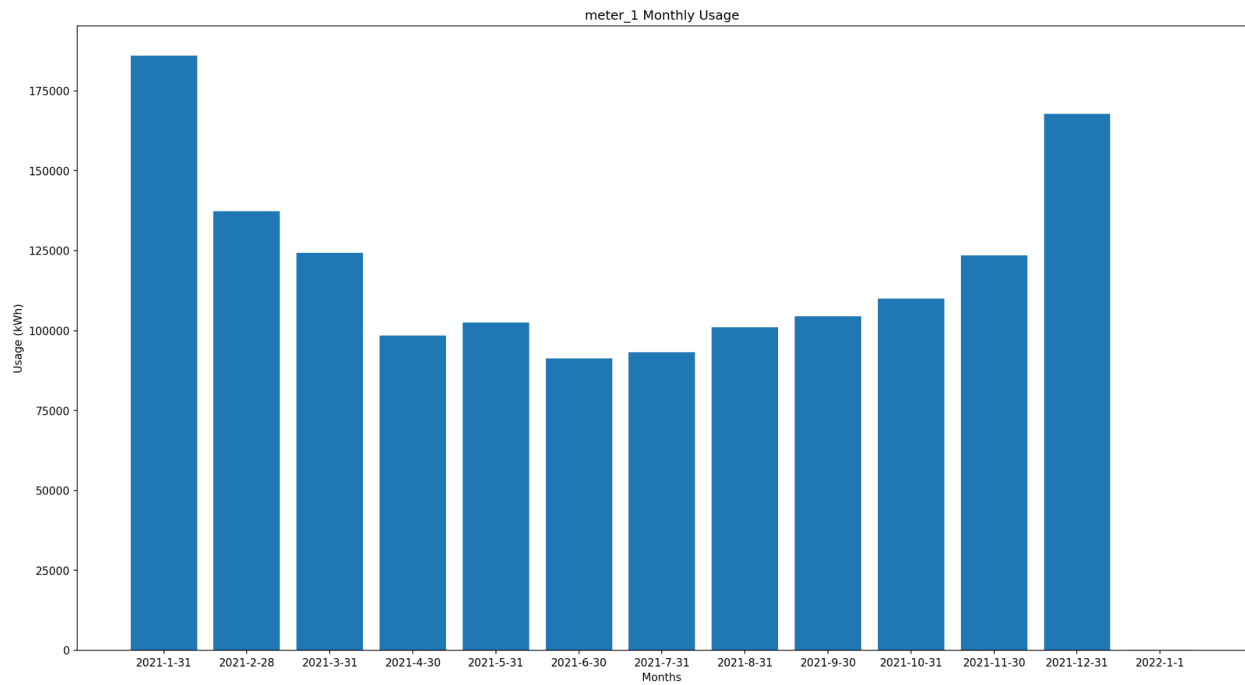


Figure B.6: IEEE-13 Meter 1 monthly energy use for tariff design

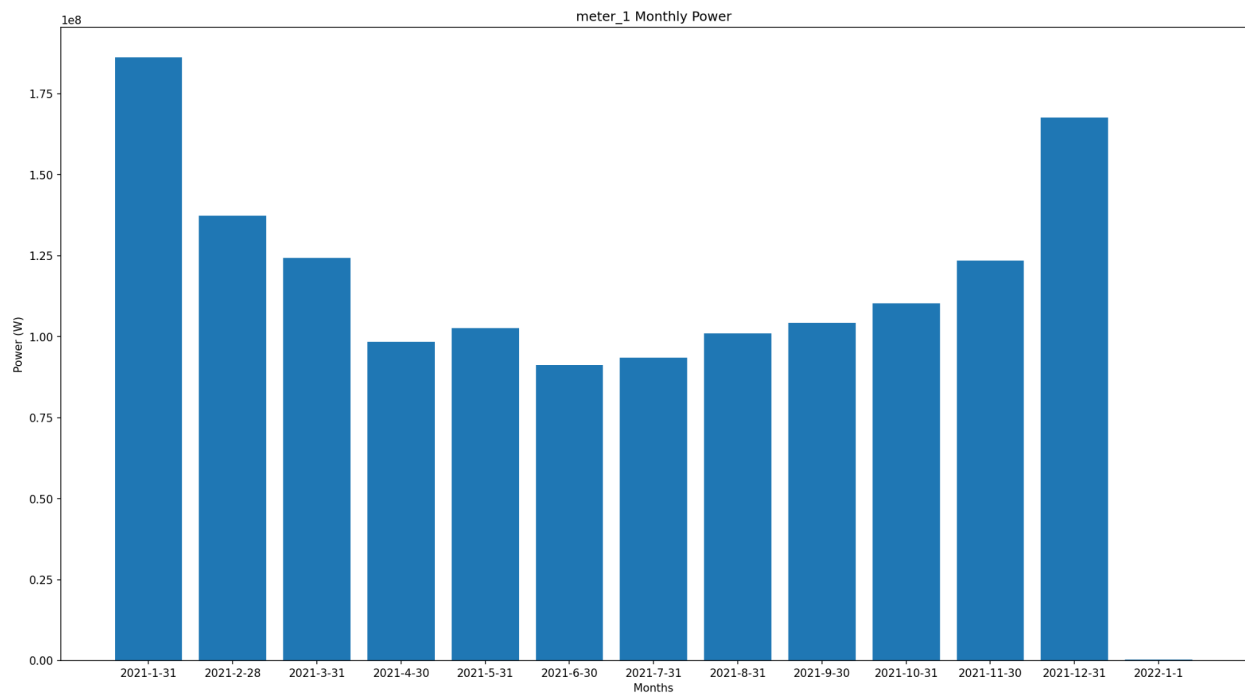
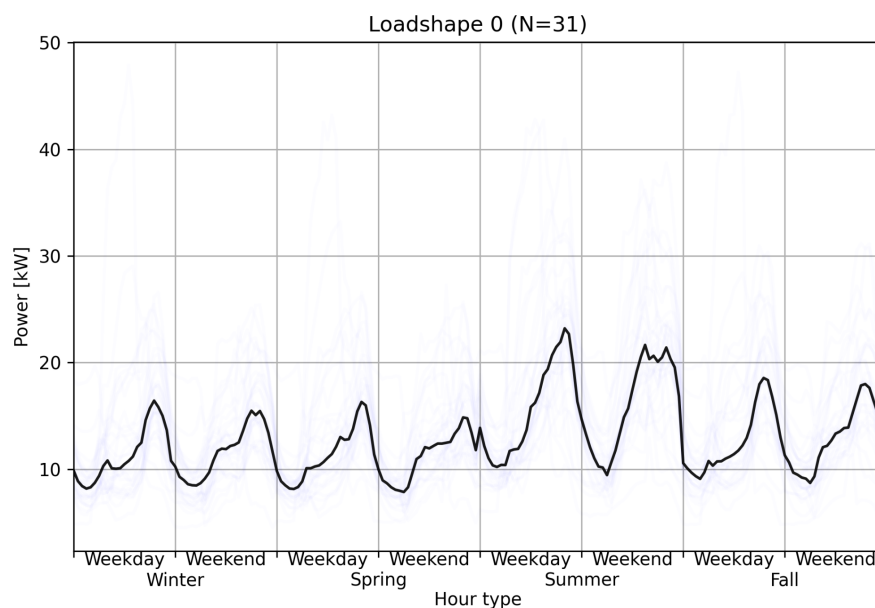


Figure B.7: IEEE-13 Meter 1 monthly peak power for tariff design

## Loadshape

The loadshape pipeline computes load shape clusters for a sample of 100 individual AMI recordings from a utility covering 1 year for a total of 878,160 hourly metering records. The data is processed in two tests. The first test generates load shape clusters such as the one shown in Figure A1, along with the GridLAB-D seasonal load schedules for use in generating load models. The load models are driven by the load schedules with the individual magnitudes necessary to ensure the same seasonal daily energy use as observed in the AMI data. The second test generates a simple load mapping based on the AMI load shape clustering.



**Figure B.8: Sample load shape generated from test AMI data.**

## Weather

The weather pipeline tests the historical and forecasting weather queries. The historical test obtains one year of NREL NSRDB data for a weather site in the San Francisco area in 2020. The forecasting test obtains a weather forecast from NOAA for the same location. The data obtained is required to match the data previously obtained in cases where the result is not time-varying.

## HiPAS GridLAB-D

The HiPAS GridLAB-D powerflow solver has been tested extensively with 3 groups of models: (1) IEEE Test Models ( $N=5$ ), (2) PG&E Taxonomy Feeders ( $N=12$ ), and (2) DOE Modern Grid

Initiative Taxonomy Feeders ( $N=23$ ). To pass validation these models must produce nodal voltage results within 1% of the required result from the models' original author/publishers. Table B.3 provides the timing of the tests for the HiPAS GridLAB-D models.

**Table B.3: HiPAS GridLAB-D Powerflow Solver Speed on Test Models**

<i>IEEE</i>	<i>Time (s)</i>	<i>PG&amp;E</i>	<i>Time (s)</i>	<i>DOE</i>	<i>Time (s)</i>
123	0.63	AL0001	12.28	R1-12470-1	2.66
13	0.63	AT0001	8.95	R1-12470-2	0.72
342	2.18	BR0015	4.59	R1-12470-3	0.6
37	0.48	BU0001	2.37	R1-12470-4	0.78
8500	2.12	D0001	1.46	R1-25000-5	0.76
		HL0004	29.5	R2-12470-1	0.9
		MC0001	18.75	R2-12470-2	0.84
		MC0006	31.93	R2-12470-3	0.95
		MO0001	1.19	R2-25000-4	1.45
		OC0001	5.15	R2-35000-5	1.38
		PL0001	4.9	R3-12470-1	1.33
				R3-12470-2	0.91
				R3-12470-3	4.1
				R4-12470-1	0.93
				R4-12470-2	0.68
				R4-25000-3	0.64
				R5-12470-1	0.87
				R5-12470-2	1.25
				R5-12470-3	5.13
				R5-12470-4	1.99
				R5-12470-5	3.28
				R5-25000-6	2
				R5-35000-7	1.31
				TMP0009	4.72

## Census

The census geodata test requests the geographic "fence" for the census tract around an address in Menlo Park, California. The test is typically completed in under 1 ms.

## Resilience

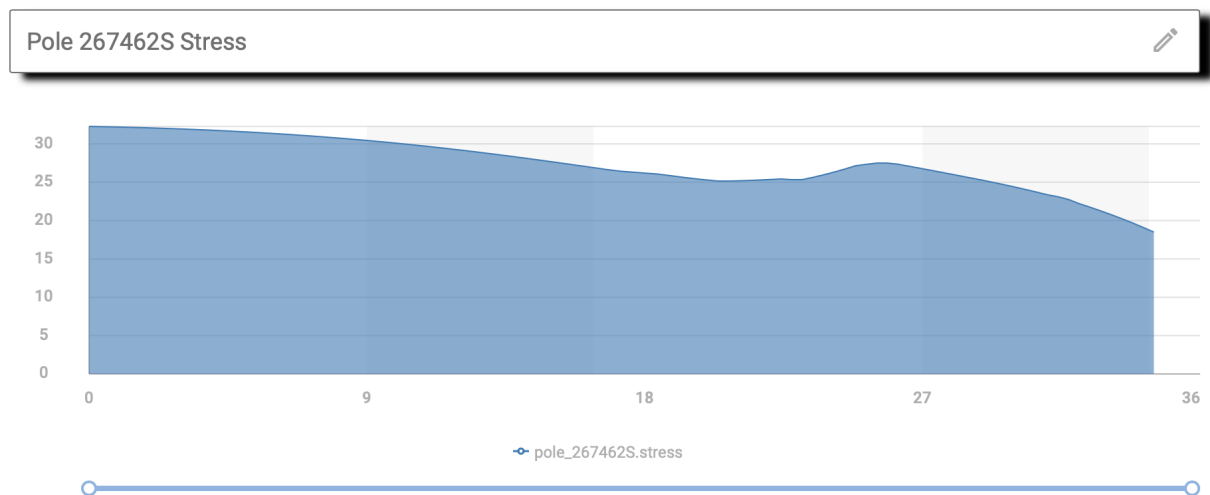
Resilience pipelines for pole analysis and vegetation analysis validation results are as follows.

The pole analysis is evaluated on a few typical SCE poles as below:

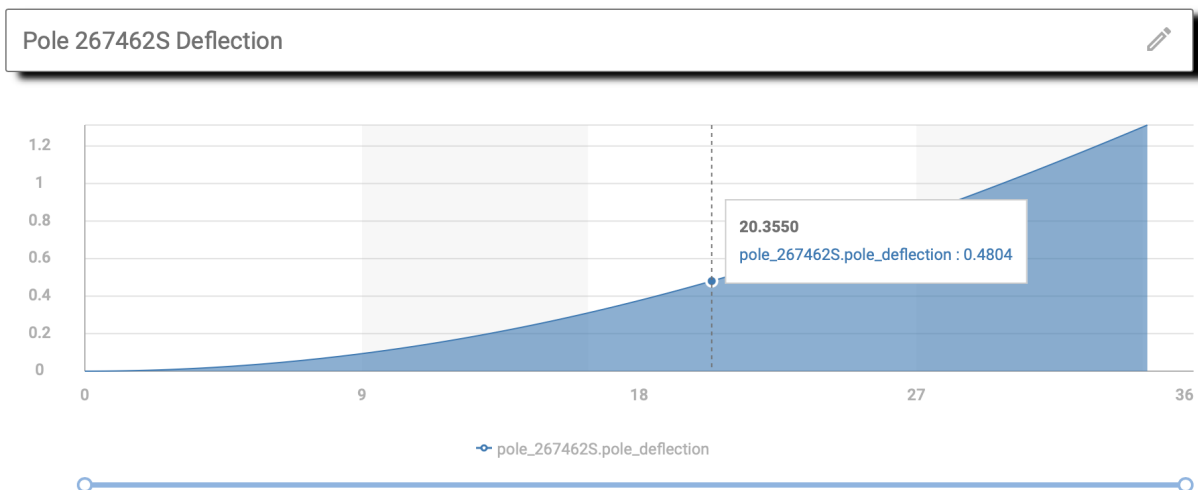
**Table B.4: Pole testing validation specifications**

Type	Length	Class	AGL	GLC	Effective Allowable Stress
Douglas Fir	40'	5	34' 6"	31"	8000 lb/in <sup>2</sup>
Douglas Fir	50'	4	42' 6"	38 1/4"	8000 lb/in <sup>2</sup>
Douglas Fir	45'	4	38' 6"	39"	8000 lb/in <sup>2</sup>
Douglas Fir	50'	1	41' 6"	41 1/4"	8000 lb/in <sup>2</sup>
Douglas Fir	45'	4	37' 4"	39"	8000 lb/in <sup>2</sup>

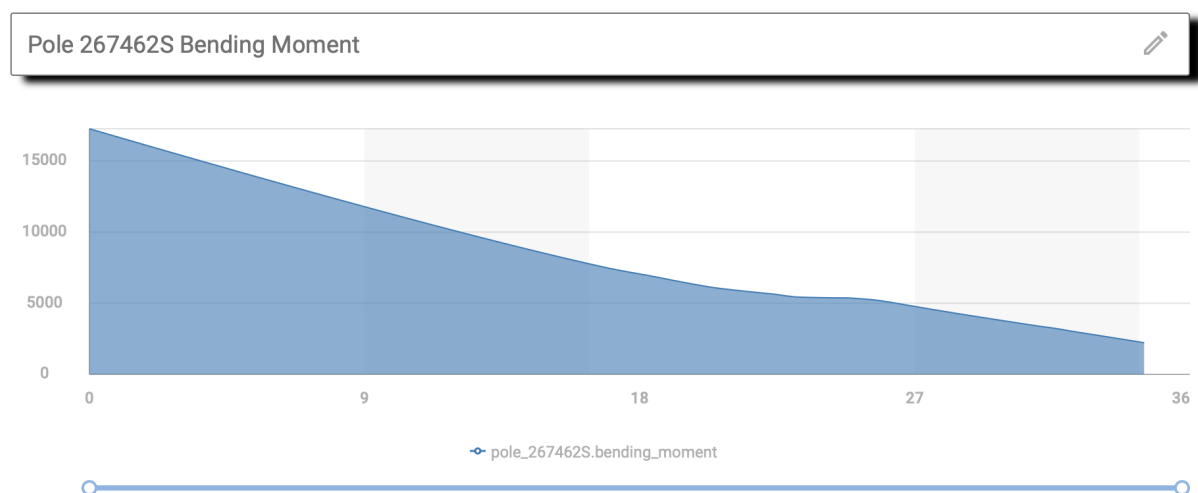
The resulting pole characteristics are shown below. The results were obtained within a few seconds.



**Figure B.9: Pole stress results for resilience template validation**

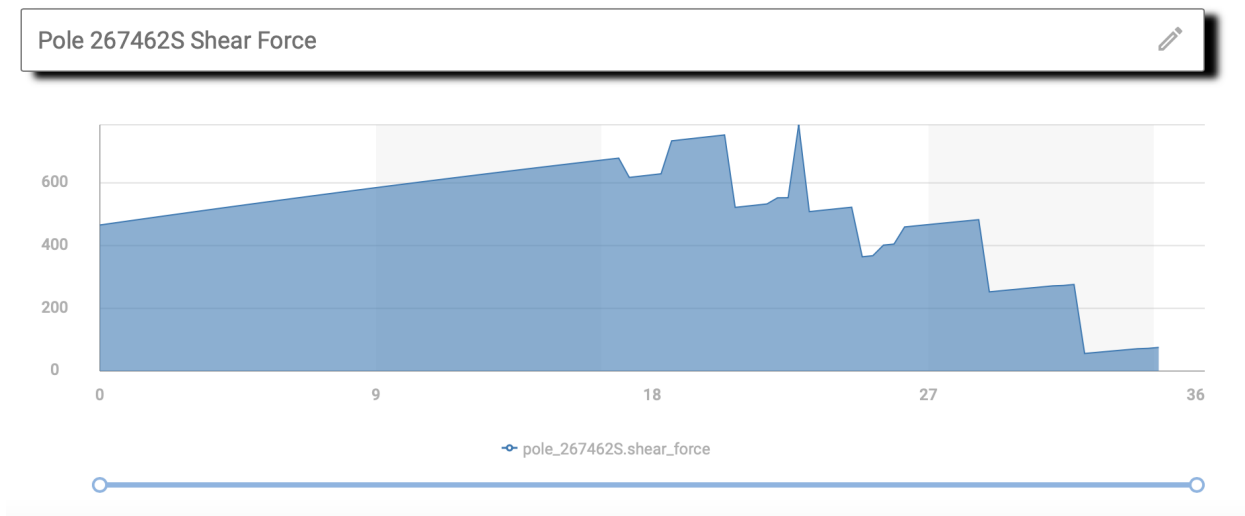


**Figure B.10: Pole deflection results for resilience template validation**



**Figure B.11: Pole bending moment results for resilience template validation**





**Figure B.9: Pole shear force results for resilience template validation**

Critical wind speed for the specified model is as follows:

**Table B.5: Wind loading for pole analysis in resilience template validation**

Pole Identifier	Critical wind speed	Wind direction
pole_267462S	45.0 m/s	0.0 deg
pole_280565E	60.0 m/s	0.0 deg
pole_306771E	60.0 m/s	0.0 deg
pole_311032S	65.0 m/s	0.0 deg
pole_312148E	60.0 m/s	0.0 deg

Vegetation analysis is evaluated on the 230kV PG&E lines using vegetation data from California Forest Observatory. The inputs for the poles are as follows:

**Table B.6: Pole configuration for pole analysis validation**

Latitude	Longitude	Pole configuration	Pole Height
37.4150451	-122.20565	flat3	18
37.414698	-122.20849	sideT	20
37.4144541	-122.21044	sideT	21
37.4142229	-122.21231	sideT	18

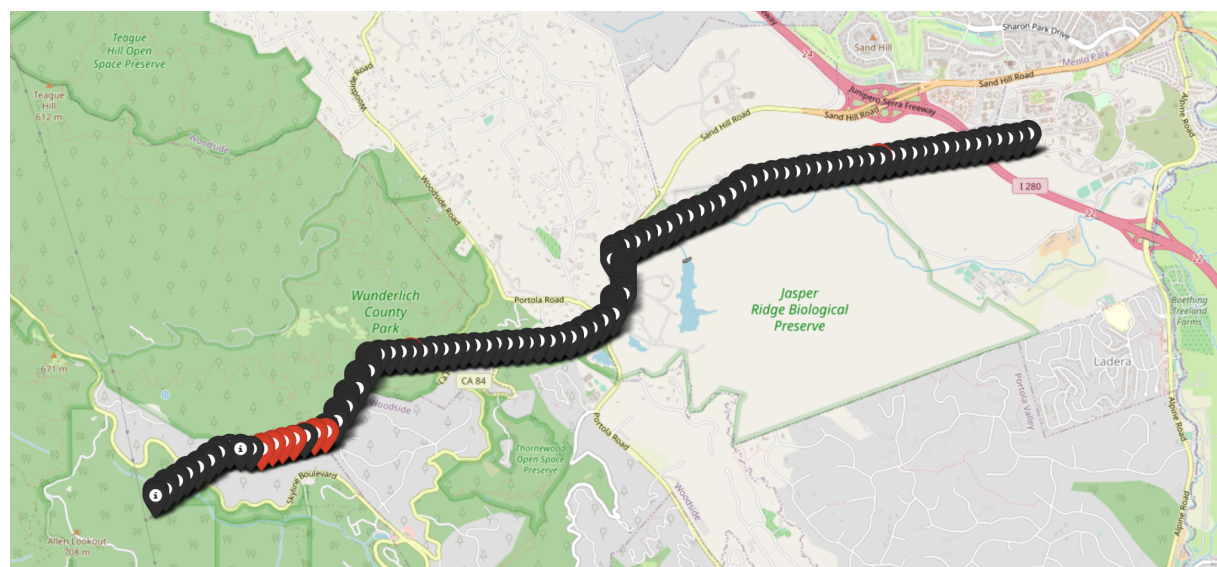
**Table B.6: Pole configuration for pole analysis validation**

<b>Latitude</b>	<b>Longitude</b>	<b>Pole configuration</b>	<b>Pole Height</b>
37.4139811	-122.21427	sideT	20
37.4136967	-122.21657	sideT	18
37.4134848	-122.21827	sideT	21
37.413211	-122.22048	sideT	18
37.4129756	-122.22232	sideT	16
37.412705	-122.22456	sideT	21
37.4123588	-122.22736	sideT	18
37.4121128	-122.2294	sideT	19
37.4118124	-122.23184	3pole	15
37.4099525	-122.23541	3pole	16
37.4091206	-122.23771	2pole	15
37.4078934	-122.24107	2pole	17
37.4071499	-122.24311	2pole	15
37.4064444	-122.24503	3pole	17
37.405623	-122.24443	vert3	17
37.4045576	-122.24418	vert3	20
37.4033885	-122.24429	vert3	20
37.4024888	-122.24468	vert3	20
37.4013606	-122.24566	vert3	22
37.3998673	-122.24927	3pole	16
37.399663	-122.25113	2pole	18
37.3993704	-122.25627	2pole	22
37.3992772	-122.25795	3pole	15
37.3988173	-122.26449	3pole	18
37.3987093	-122.2661	2pole	15

**Table B.6: Pole configuration for pole analysis validation**

Latitude	Longitude	Pole configuration	Pole Height
37.398606	-122.26757	3pole	18
37.3944838	-122.27063	3pole	21
37.3930648	-122.2718	3pole	21
37.3924848	-122.27446	2pole	17
37.3918978	-122.27717	2pole	21
37.391605	-122.2786	2pole	23
37.3917099	-122.28139	3pole	16
37.388064	-122.28844	tower	23

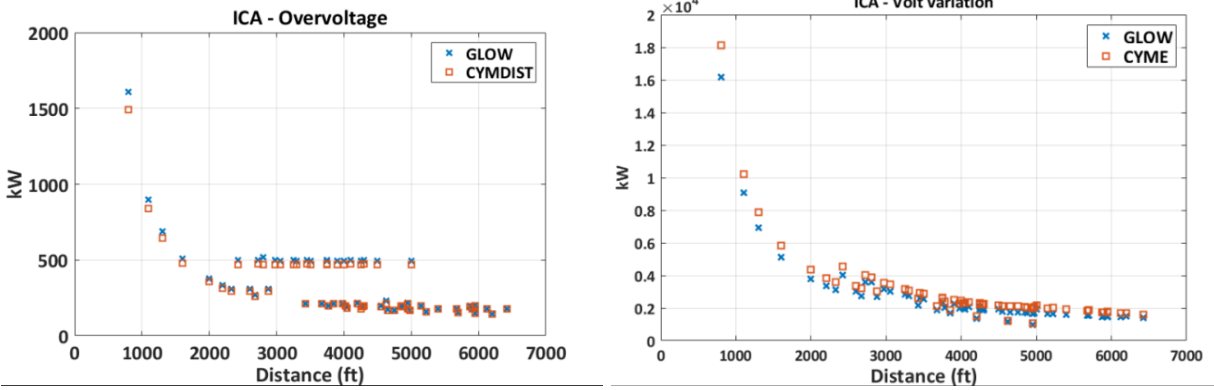
The strike risk and contact list is calculated and shown using the Folium interface where the Red section indicates the potential vegetation contact risk. The results obtained within a few seconds.



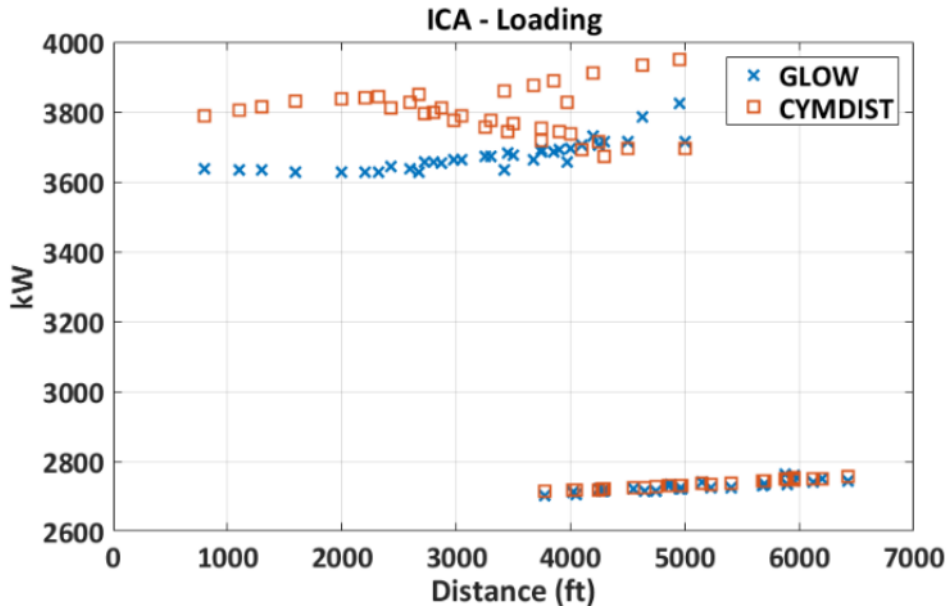
**Figure B.10: Pole analysis results for 230kV line at SLAC with vegetation contact risk**

## Hosting Capacity

The Hosting Capacity pipeline was evaluated in two ways. First the results were compared with Cyme results for the IEEE 123 network model. Over-voltage limits and voltage fluctuations were compared, as shown in Figures B.11 and B.12.

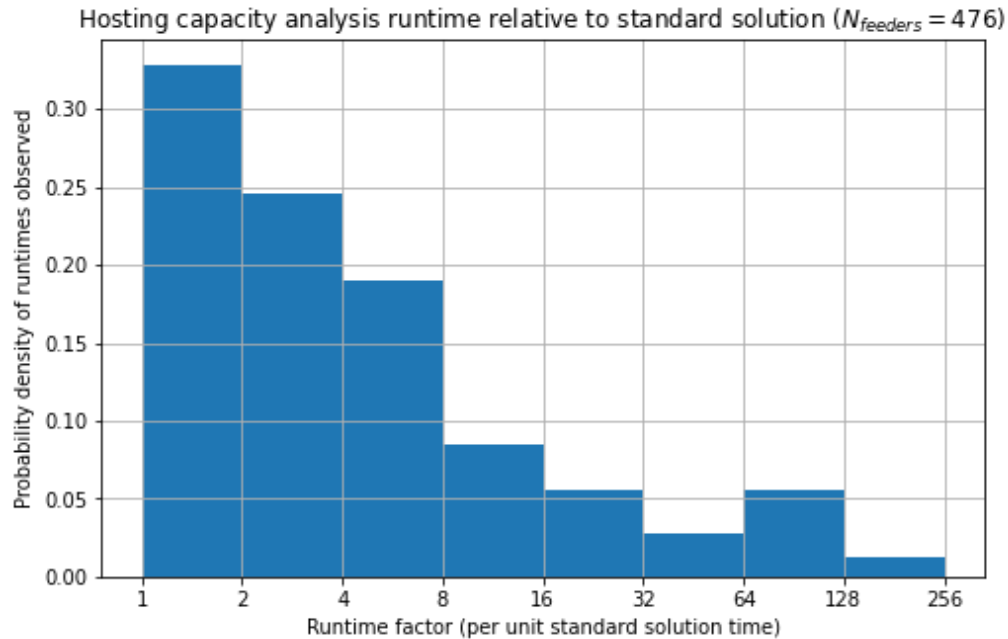


**Figure B.11: GLOW ICA IEEE-123 over-voltage (left) and voltage variation (right) results compared to IEEE-123**



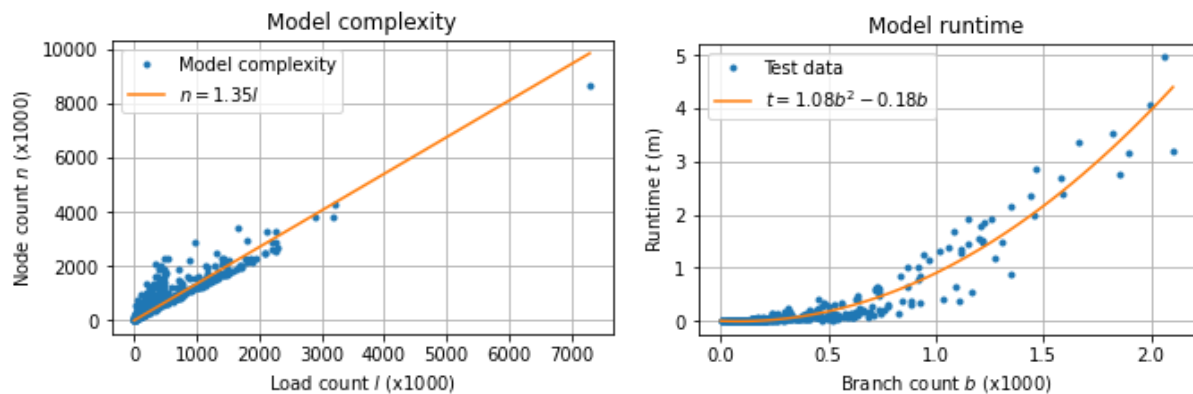
**Figure B.12: GLOW and CYMEDIST comparison of ICA capacity**

The computational speed of the ICA implementation was also evaluated using the National Grid feeders. Of the ~2000 feeders provided by National Grid, 476 were run for ICA analysis. The runtime relative to a non-ICA solution is shown in Figure B.13.



**Figure B.13: ICA runtime relative to a non-ICA powerflow solution**

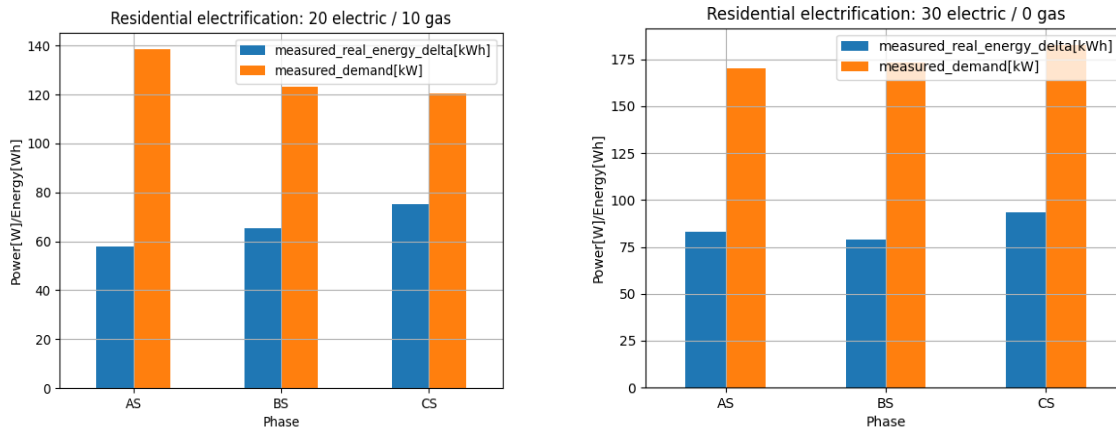
The National Grid feeder model ICA complexity and performance relative to model size for the 476 National Grid feeder models are shown in Figure B.14.



**Figure B.14: ICA model complexity (left) and runtime relative to model size (right)**

## Electrification

The electrification use-case was assessed on an evenly distributed feeder with 90 houses. The results were achieved within a few seconds, however the simulation does not include connection to a distribution feeder at this time. The results for converting 1/3 of the homes is shown in Figure B.15.



**Figure B.15: Impact of 1/3 electrification in San Jose, CA**

## Address

Address resolution geodata pipeline is tested in both directions (i.e., address to lat/lon, and lat/lon to address) using a "round robin" test to ensure the Nominatim platform works correctly. A test of 20 addresses in San Mateo County is used to complete a performance test, which is completed in about 10 seconds. A test of 100 addresses requires approximately 50 seconds to complete.

## Cyme Extract

The Cyme Extract pipelines was evaluated on ~2000 feeders provided by National Grid during the Load Growth Forecast (LGF) study conducted in Fall 2021. The purpose of the study is to develop an annual update to the 15 year LGF for New York state. The 2021 study was conducted using the DOE version of GridLAB-D (version 4.0). The results of the 2021 LGF study were used as the baseline for comparison with the results of the 2022 LGF study.

Two key performance metrics were used to evaluation the performance of HiPAS GridLAB-D's Cyme MDB to GridLAB-D GLM converter: (1) unsupervised conversion success, and (2) energy consumption relative to 2021 LGF results.

## Unsupervised Conversion success

The CYME to GLM converter was run on the ~2000 feeder models provided for the 2022 LGF study. Of these 97.5% converted successfully to solvable GLM models. Of the 2.5% (roughly 50 models) that did not convert successfully, the principal root causes of failure were identified as follows:

1. **Cyme model network errors:** These models contained topological and asset configurations that were invalid, such as invalid phase configurations, load bus

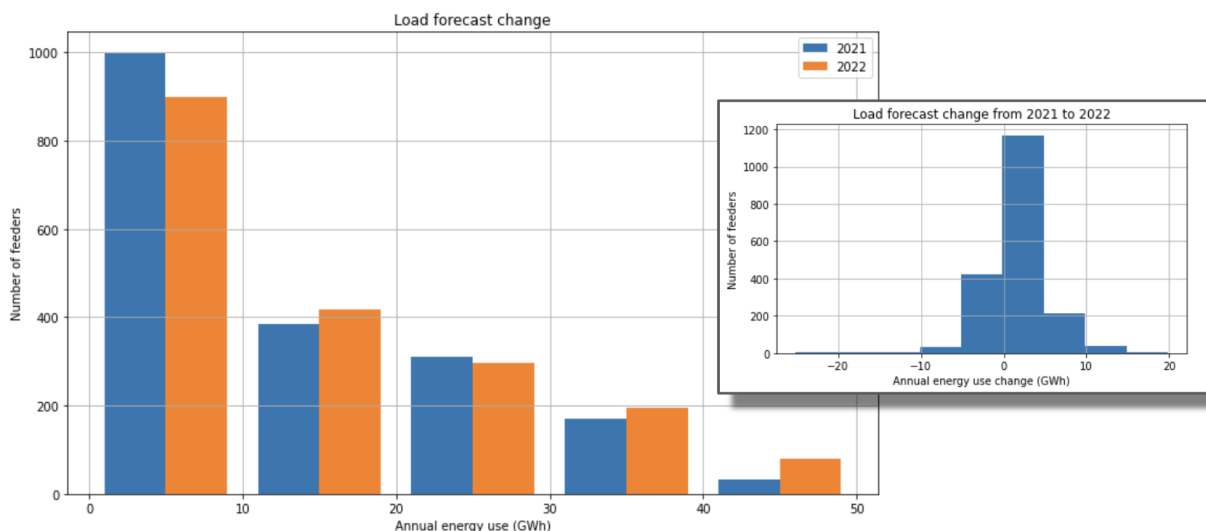
connections, or transformer configuration. These errors were typically corrected with manual interventions by adjusting the asset configuration in consultation with the National Grid engineers.

2. **Cyme load model errors:** These models contained loads that could not be supported by the network at the point of interconnection, such as loads for which the unit of load were incorrect, e.g., MW instead kW, or loads with impossible magnitudes. These were typically corrected by manual intervention by adjusting the load magnitude or limits in consultation with the National Grid engineers.
3. **Cyme-GridLAB-D model incompatibility:** These models contained asset or load configuration that cannot be modeled in GridLAB-D, such as distributed loads. These were typically corrected by creating an equivalent asset or load, the choice of which was made in consultation with the National Grid engineers.

Of the failed models, 0.5% (about 10 models) could not be converted successfully and were abandoned.

## Converted Model Performance

The CYME to GLM converter performance was evaluated on the ~2000 feeder models by comparing the 2021 LGF to 2022 LGF, with specific attention to the difference between the 2021 forecast for 2022 and the actual 2022 values, as shown in Figure B.16.



**Figure B.16: National Grid LGF change from 2021 to 2022 and distribution (inset).**

The results suggest that the load 2022 results from the HiPAS GridLAB-D LGF study are consistent with the expected 2022 load based on the 2021 DOE GridLAB-D LGF study.