SOLAR ENERGY TECHNOLOGIES OFFICE
U.S. Department Of Energy

EDI SI

Data Preprocessing Use Case

AMI Data to Load Profile and Data Imputation

November 17, 2023

Karthikeyan Balasubramaniam, Siby Plathottam, Kumar Jhala
Argonne National Laboratory

Overview

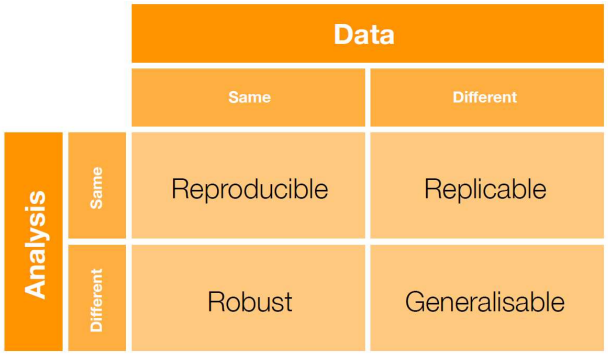
1. Goal
2. Dependencies
3. Use Case Overview
4. Data Anonymization: AMI Data to Load Profile
    - Overview
    - Demo
5. Data Imputation
    - Overview
    - Demo
6. Run your own algorithm
    - Overview
    - Demo

Goal

Goal of OEDISI Platform: Improved workflow that seamlessly connects data to algorithms

➢ **Challenges and Solution**

- ➢ System data is in multiple formats → Converters
- ➢ No system data → Use open datasets pulled from the cloud
- ➢ Measurement data is partially or completely unavailable → Augment with existing measurement data
- ➢ Available measurement data are in different formats and time resolution → Data conversion and synchronization
- ➢ Measurement data has missing data points → Data imputation
- ➢ Runtime environment → Docker/Podman



Whitaker (2018) https://doi.org/10.6084/m9.figshare.7140050.v2

Dependencies

- Why Docker?
  - Namespace isolation → all the runtime dependencies are part of the Docker image.
  - Prebuilt image can be downloaded to your machine using,
    - docker pull openenergydatainitiative/oedisi-singlecontainer-demo:0.0.1
  - **Reproducible results and errors!**
- Docker Installation
  - Linux distros → extremely simple process
    - Debian based -- sudo apt install -y docker.io
  - Windows,
    - Docker Desktop → Install instructions: https://openei.org/wiki/OEDI-SI/User_Guide

Dependencies

- Single container
  - https://github.com/openEDI/oedi-si-single-container/tree/0.3.0
  - Abstraction that allows runtime modifications,
    - User does not need to know about,
      - Containers
      - OEDISI framework
      - Co-simulation
    - Simple CLI that makes it easier for the user to quickly make changes and run the co-simulation.
    - Change config → change scenarios → analyze
- Requires Python to be installed on your machine.

Use Case Overview

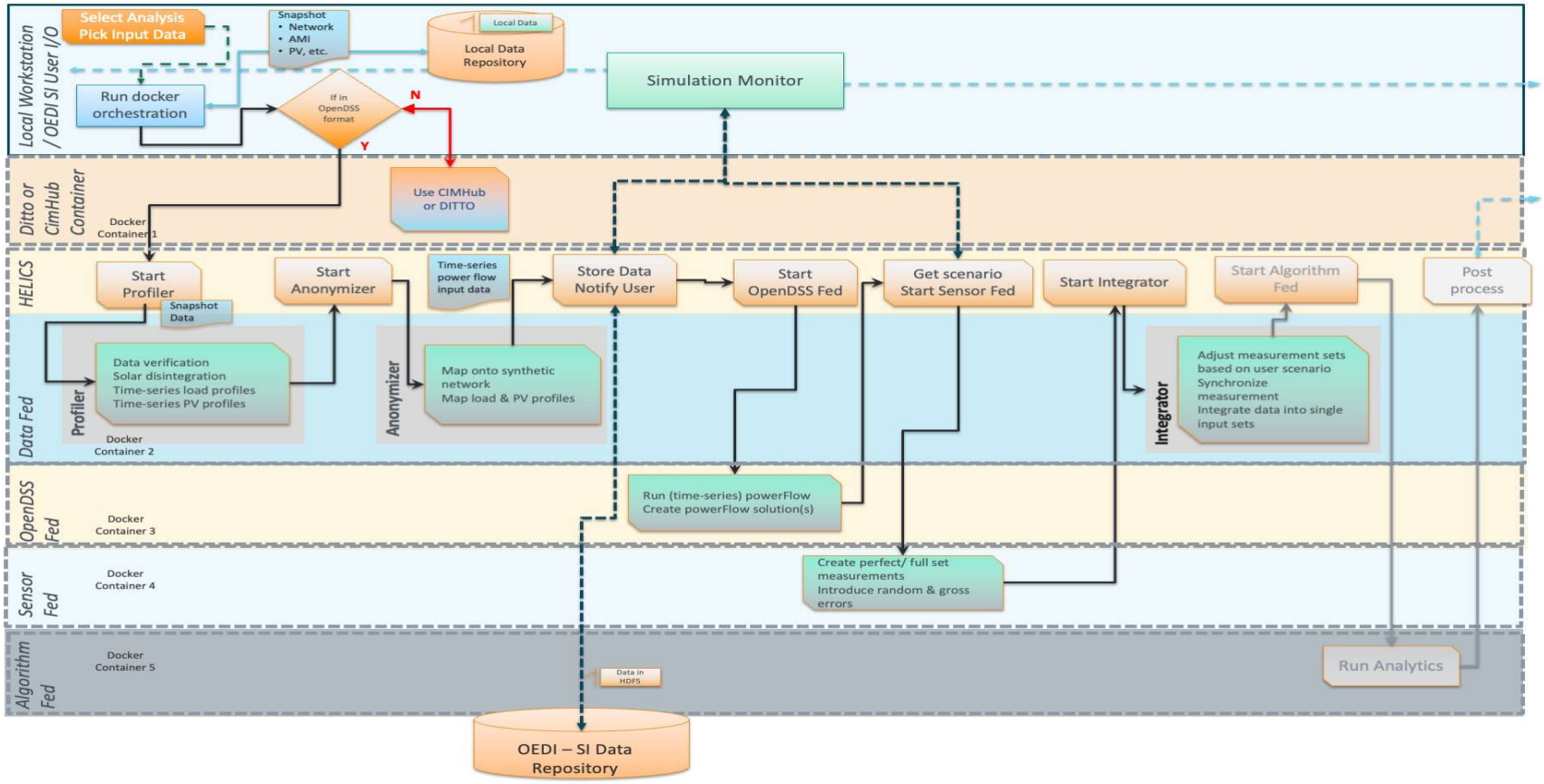
Data preprocessing -- transform inconsistent input dataset → consistent set of input and measurement data that can work with OEDI SI tools.

While the data preprocessing tools have wide range of uses, we will limit this use case to two scenarios,

1. Data Anonymization: AMI Data to Load Profile
2. Data Imputation

- For the data anonymization scenario, we will consider the case where the user wants to utilize AMI data to create load profiles for any test system.
    - **Map field measurement data → generic test system**
- In the second scenario, we will mitigate issues related to **partially missing streaming measurement data** i.e., imputing data when there are missing data points in a stream of data. This is useful for applications such as distribution system state estimation.

Data Preprocessing



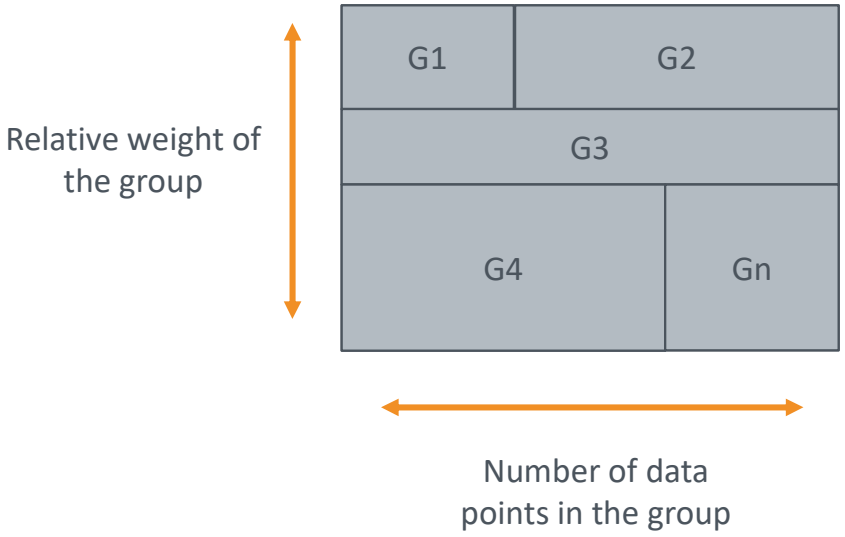
Data Anonymization: AMI Data to Load Profile

Data Anonymization

- Data anonymization -- The process of removing any personally identifiable data and ensuring that the data cannot be reverse engineered to reconstruct the field data.
- Headers – customer id, load type, zip code, and any such information that can be translated to an unidentifiable form will be translated.
- Within the scope of OEDI SI work, structural anonymization is implicitly employed by default. For instance, we take field data from different sources, do a linear transformation, and then map it to a system that is different from the system that generated the data.
- The combination of translating information that is of interest, along with structural anonymization, ensures that the transformed data cannot be used to reconstruct the original data.

AMI Data to Load Profile

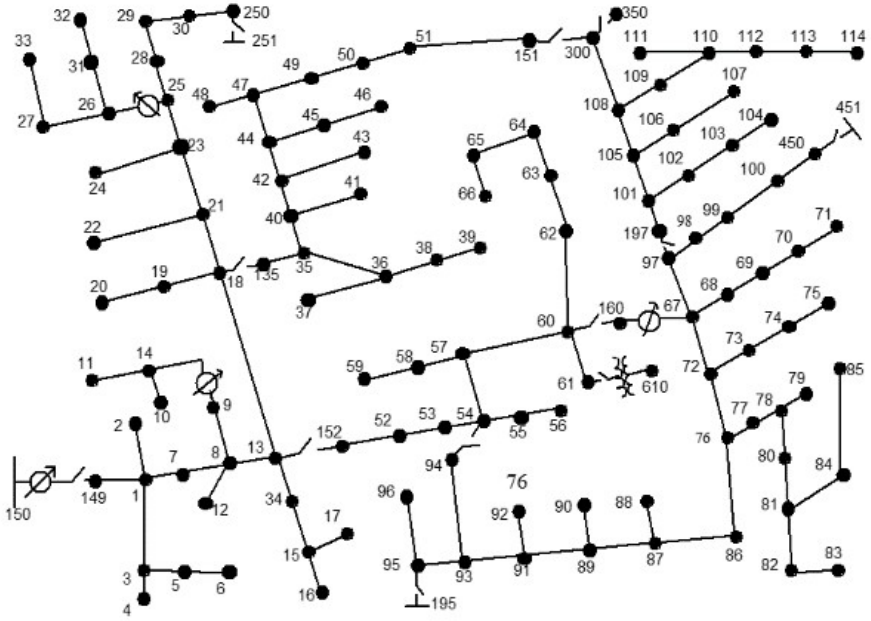
- A schema is provided to connect field AMI data with OEDI SI Framework. If the data is in the specified format, then the data can be used with OEDI SI tools. Data can be up/down sampled.
- An example adapter to show the user how to convert existing data to the required format is currently under development and will be made available at https://github.com/openEDI/oedi-si-single-container.
- The said data can be aggregated across user specified groups (such as different load types) or can be aggregated across random subset of data to form load profile.
- Weighted linear combination of the data group is then used to form load profile at each node.



$$P = \sum wG$$

AMI Data to Load Profile

- Works for any network.
- Tested with IEEE 13-bus, 123-bus, 8500-node, SmartDS test systems.
- Runtime -- Largest test system ~8600 nodes takes about 10 seconds.
- Approach allows generating new profile for each load bus during each run.
- Reproducible profiles via random number seed.



IEEE 123-bus System

Acknowledgement: N. Samaan, et.al. "Combined transmission and distribution test system to study high penetration of distributed solar generation"

AMI Data to Load Profile Scenario Demo

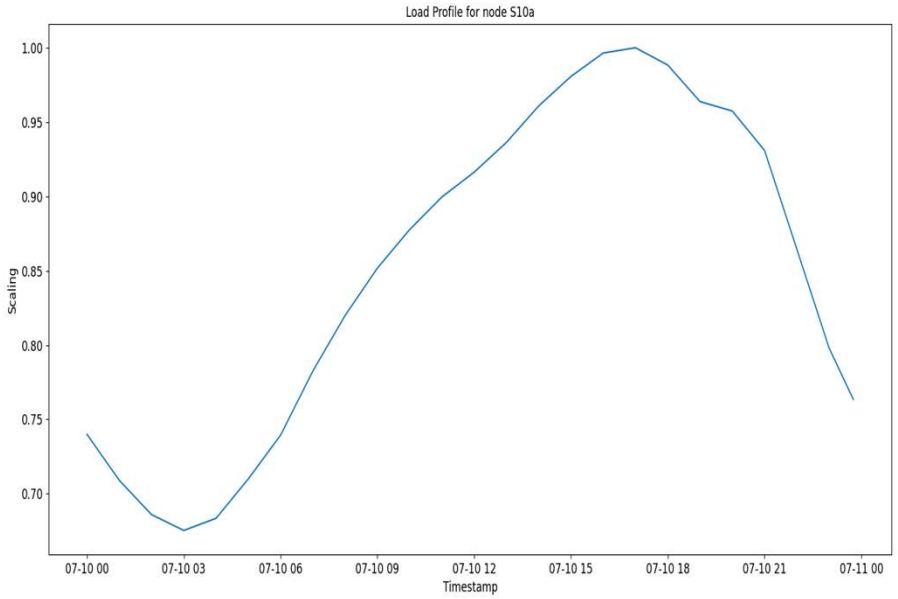
Input Data

Philosophy: Do not force the user to learn things that are not of interest to them.

Logical Information Flow

- Required data
  1. System data – information to run power flow
  2. Measurement data (used for generating load profile)
- **System Data:** https://github.com/openEDI/oedisi-ieee123/tree/main/qsts
- **Measurement Data:** https://github.com/openEDI/oedi-si-single-container/blob/main/build/datapreprocessor/datapreprocessor/datapreprocessor/data/solarhome/solarhome_customers-300_days-365.csv

Output Data



Load Profile for one of the nodes in the system

- https://github.com/openEDI/gadal-ieee123/tree/main/profiles/load_profiles
- Variations of profiles based on user input can also be generated during runtime.
- The container file system can be accessed by using cli, more information is given in the following link, https://github.com/openEDI/oedi-si-single-container/tree/main#using-the-cli
- Calling the run command through cli with –interactive option gives the user access to the container file system.

Steps to Run Use Case

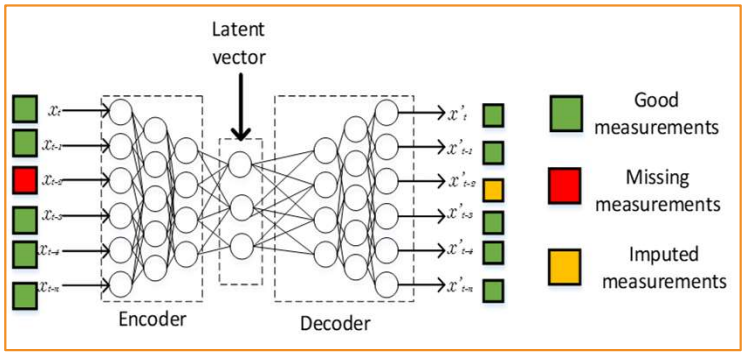
- The steps outlined below are given at https://openei.org/wiki/OEDI-SI/Scenarios/1_-_Data_Anonymization:_AMI_Data_to_Load_Profile_Scenario

1. docker pull openenergydatainitiative/oedisi-singlecontainer-demo:0.0.1
2. git clone https://github.com/openEDI/oedi-si-single-container.git (or download the zip file) and cd to the folder
3. pip install -e .
4. oedisisc init -p sampleProject
5. oedisisc run -p sampleProject -c sampleProject/config/user_config.json

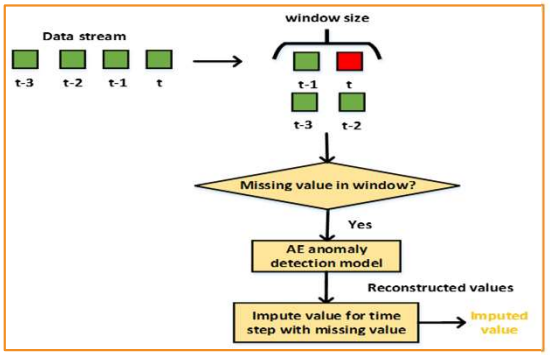
Data Imputation

Data Imputation

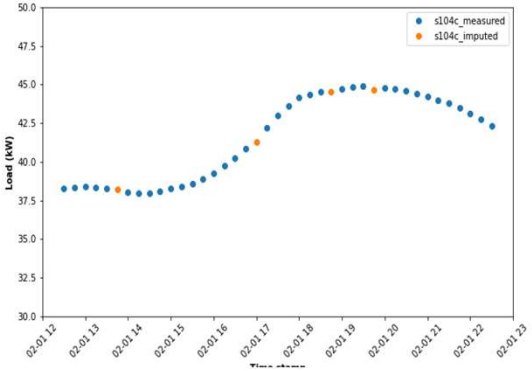
- ✓ Denoising Autoencoders (DAEs) are unsupervised Deep learning algorithms.
- ✓ Encoder encodes training into an encoding on a higher or lower dimensional hyperplane.
- ✓ Decoder decodes the encoding to reconstruct the original data.
- ✓ Impute on **streaming data** when missingness is detected.



Denoising Autoencoder



Streaming Data Imputation



Imputed Data

Data Imputation Scenario Demo

Data Imputation: Input Data

Required data

1. System data: https://github.com/openEDI/oedisi-ieee123
2. User input: https://github.com/openEDI/oedi-si-single-container/blob/main/runner/user_config.json

Data Imputation: Output Data

- Based on the user provided configuration, algorithms will be run, and the respective outputs will be generated in csv and feather format.
- Instructions on how to access individual run outputs -> https://github.com/openEDI/oedi-si-single-container/tree/main#output

Swap Public Algorithms With Your Algorithms

Swap Public Algorithms With Your Algorithms

- Reference Code -- https://github.com/openEDI/oedi-si-single-container/blob/main/user_federates/user_dsse/main.py
- Agnostic to co-simulation and OEDI-SI framework
- Adding your algorithm is simply a function call,
  - outputData=algorithm(inputData)

Logical Information Flow

- Currently only Python is supported. Support for MATLAB/Octave is under development.
- Even executables can be run this way. For instance, one can take the same example code above and write the input data to disk and make an os.system/subprocess call to start a DSSE executable that takes the input file as an argument and writes out the output as a file which will then be read and passed as output from Python program. Not the most efficient approach but works.