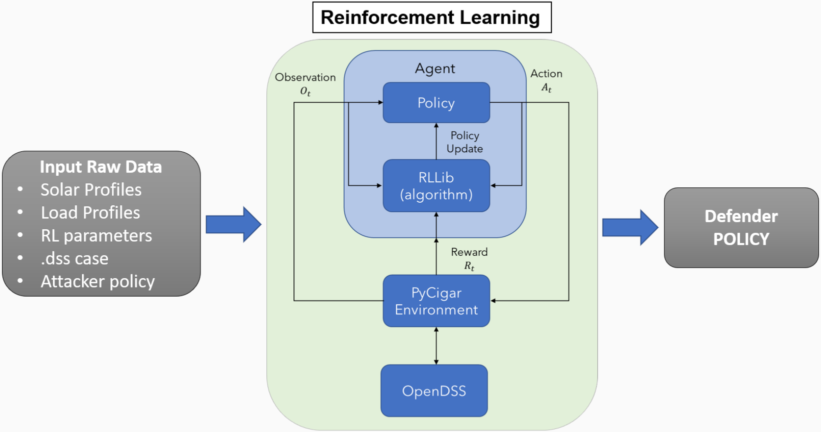
# Synthetic Solar Data Generation and Linear Power Flow Solver for RL training

**Ignacio Losada Carreño, Anna Scaglione**

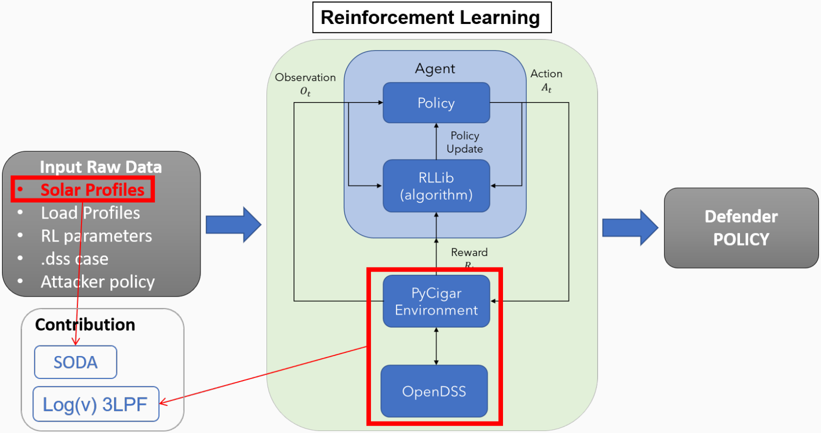
CIGAR Workshop. March 17, 2021

Arizona State University

# Reinforcement learning for distribution systems



# Improving RL training



# SODA: An Irradiance-Based Synthetic Solar Data Generation Tool

# Motivation and background

- RL agents trained on different solar and load conditions
- Low resolution datasets do not capture solar variability
- High resolution data not available (sparse locations)
- Combine a physics-based method (30-min resolution) with a stochastic model trained on PMU data to generate 1-second solar data

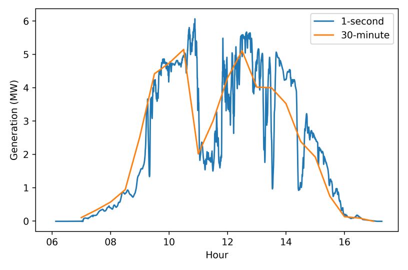



**Figure 1:** 1-second PMU data vs 30-min NSRDB data

[0] The National Solar Radiation Database (NSRDB) has a spatial resolution of 4km and a temporal resolution of 30 minutes

**Working hypothesis:** The conditional distribution of solar power given the cloud density is the same for different locations

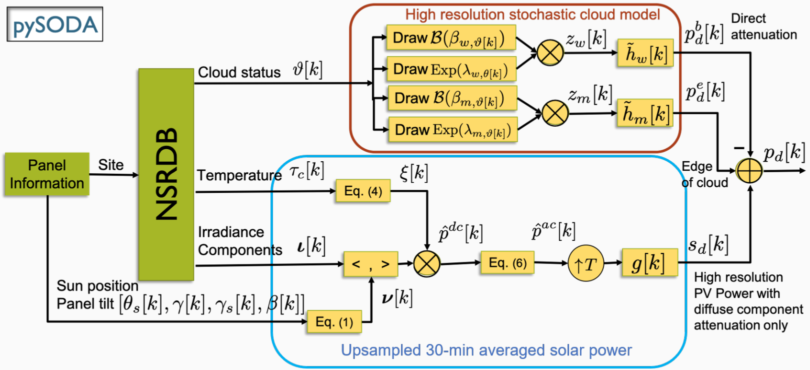



**Figure 2:** SODA block diagram

[0] https://github.com/Ignacio-Losada/SoDa

# Showcasing SODA. Synthetic vs PMU data

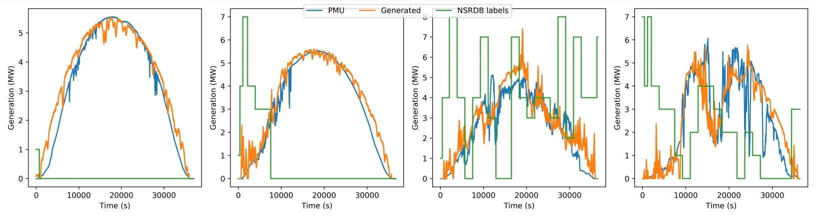



**Figure 3:** SODA vs 1-second PMU data from Riverside, CA

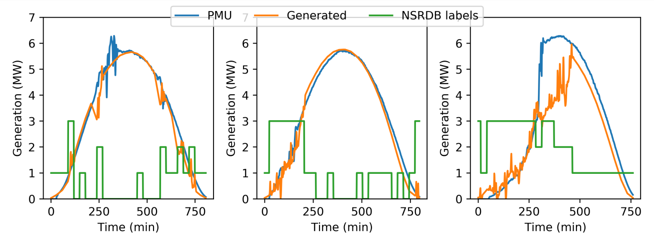



**Figure 4:** SODA vs 1-minute PMU data from Berkeley, CA

# Footprint of SODA

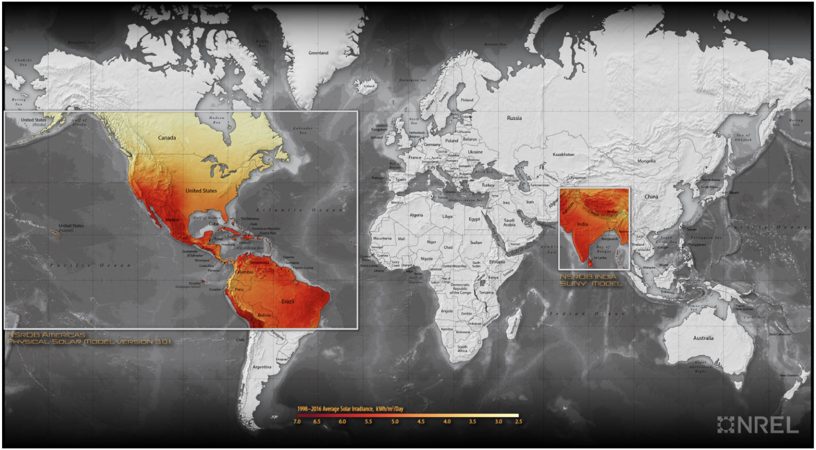



**Figure 5:** Area covered by the NSRDB

# Limitations and summary

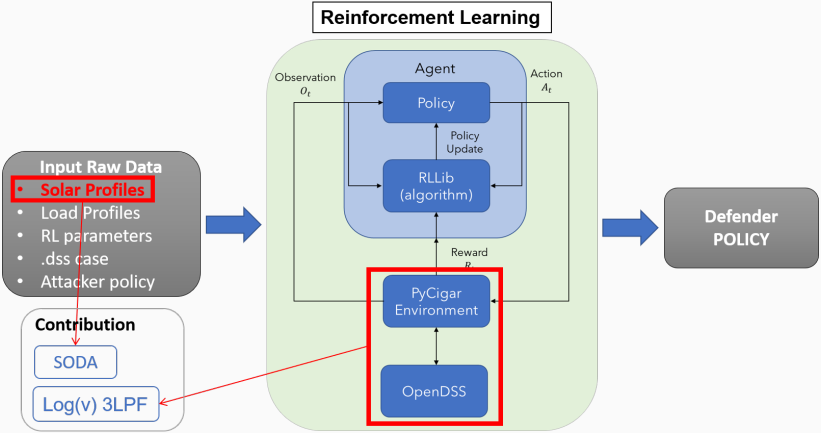
**Limitations:**

- Model does not account for time-of-day variability
- Mismatch NSRDB cloud labels
- Training was done with MW-scale data

**Summary:**

- We propose a stochastic model trained with PMU data that generates statistically representative 1-second resolution solar data
- This method, unlike NWP, scales for high resolutions
- https://github.com/Ignacio-Losada/SoDa

# Improving RL training



# Log(v) 3LPF: A linearized solution to train reinforcement learning algorithms for distribution systems

# Reinforcement learning on distribution systems, PPO-Clip

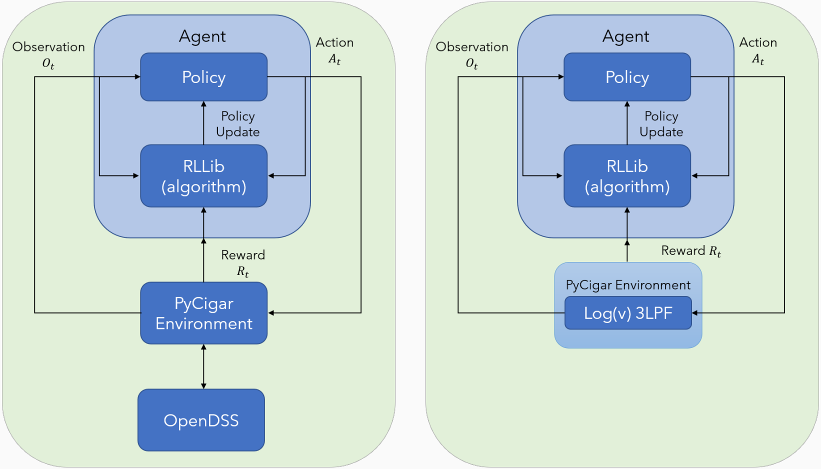



**Figure 6:** Current PyCigar modeling diagram (left) vs proposed architecture (right)

# Reinforcement learning on distribution systems, PPO-Clip

- Power Flow (PF) equations are used to compute the rewards
- Rewards computed every training iteration, every time step and every action sampled by the RL algorithm
- Linear 3-phase unbalanced PF solver to speed up the training process

Formulation:

$$\boxed{\begin{array}{rl} \textbf{OpenDSS} \rightarrow & \boldsymbol{i} = \boldsymbol{Y}_{\text{bus}}\boldsymbol{v} \\ \textbf{Log(v) 3LPF:} \rightarrow & \boldsymbol{s} = D(\boldsymbol{v}\boldsymbol{v}^H \boldsymbol{Y}_{\text{bus}}^H) \end{array}}$$

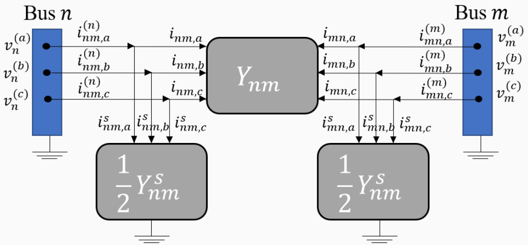



**Figure 7:** Pi-Model representation

# Our contribution

- **Modeling capabilities**
  - Embedded linear power flow solver
  - ZIP load models
  - Transformers, 3-phase and 240/120V center-tapped
  - Voltage regulators and controls
  - Capacitor banks and controls
- **Modeling accuracy**
  - PF solution validated against OpenDSS
  - Modeling of devices matches that of OpenDSS

# Log(v) 3LPF

**Kirchoff + Ohm + Losses**:

$$\boxed{\boldsymbol{S}_{nm}^{(n)} = \boldsymbol{v}_n \boldsymbol{v}_n^H \left( \mathrm{Y}_{nm}^{(n)} + \frac{1}{2} \mathrm{Y}_{nm}^{s} \right)^H + \boldsymbol{v}_n \boldsymbol{v}_m^H \left( \mathrm{Y}_{nm}^{(m)} \right)^H}$$

We want to remove the non-linearity $\boldsymbol{v}_n \boldsymbol{v}_n^H$ from the equation that relates power flows to voltage

$$\boldsymbol{v}_n := \left[ |v_n^a| \, e^{j\theta_a}, \left|v_n^b\right| e^{j\theta_b}, |v_n^c| \, e^{j\theta_c} \right]^T \tag{1}$$

$$|v_n^p| = e^{\log|v_n^p|}, \quad u_n^p := \log |v_n^p| \tag{2}$$

$$\boxed{v_n^p = e^{u_n^p} e^{j\theta_p}} \tag{3}$$

**First-order Taylor** expansion around 1 p.u. and 0 degree angles

# Log(v) 3LPF

$$\textbf{Non-linear ACPF:} \quad \boldsymbol{S}_{nm}^{(n)} = \boldsymbol{v}_n \boldsymbol{v}_n^H \left( \mathrm{Y}_{nm}^{(n)} + \frac{1}{2} \mathrm{Y}_{nm}^{s} \right)^H + \boldsymbol{v}_n \boldsymbol{v}_m^H \left( \mathrm{Y}_{nm}^{(m)} \right)^H$$

Expanding, dropping high-order terms, we may obtain

**Log(v) 3LPF (Linear):**

$$\tilde{\boldsymbol{s}}_{nm} \approx \tilde{\mathrm{Y}}_{\mathrm{bus}} \boldsymbol{x} \quad \text{where} \quad \boldsymbol{x} \triangleq \begin{bmatrix} \boldsymbol{u} \\ \tilde{\boldsymbol{\theta}} \end{bmatrix} \tag{4}$$

$$\boldsymbol{x} \approx \tilde{\mathrm{Y}}_{\mathrm{bus}}^{-1} \tilde{\boldsymbol{s}}_{nm}$$

and we may recover the voltage phasors as follows

$$\boldsymbol{v} \approx \boldsymbol{\Delta}_3 \mathrm{diag}\left(e^{\boldsymbol{u}}\right) e^{j\tilde{\boldsymbol{\theta}}_n}$$

## Outperforming OpenDSS when computing the PF solution

- Matrix Inversion, $\tilde{Y}_{\text{bus}}^{-1}$
    - Gauss-Jordan: $\mathcal{O}\left(n^3\right)$
    - LU decomposition (OpenDSS): $\mathcal{O}\left(\frac{2}{3}n^3\right)$
    - **Sherman-Morrison-Woodbury:** $\rightarrow \mathcal{O}\left(\left(2n+1\right)k^2 + 3k^3\right)$

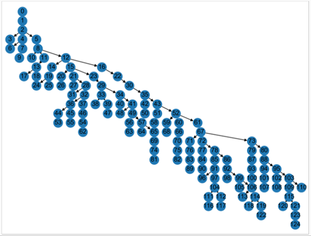
$$(\mathrm{A} + \mathrm{UBV})^{-1} = \mathrm{A}^{-1} - \mathrm{A}^{-1}\mathrm{U}\left(\mathrm{I} + \mathrm{BVA}^{-1}\mathrm{U}\right)^{-1}\mathrm{BVA}^{-1} \tag{5}$$

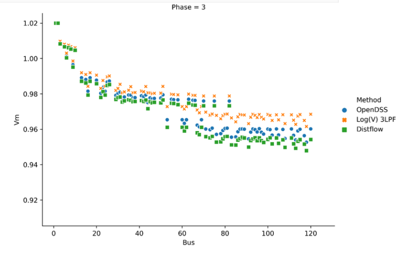
- Fixed-point iterations, $\mathcal{O}\left(n^2\right)$

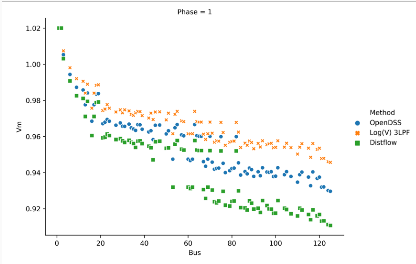
**Table 1:** LU vs SMW in MFLOPS

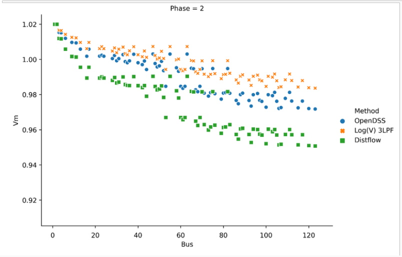
| | **IEEE 13** | **IEEE 37** | **IEEE 123** | **IEEE 8500** |
|---|---|---|---|---|
| **LU** | 0.4 | 8.5 | 114 | 3311300 |
| **SMW** | 0.1 | 0.5 | 9.6 | 19.7 |
| **Ratio** | 4 | 17 | 12 | 168000 |

# Solving one snapshot of the IEEE-123 test case





# OpenDSS vs Log(V) 3LPF for testing policies

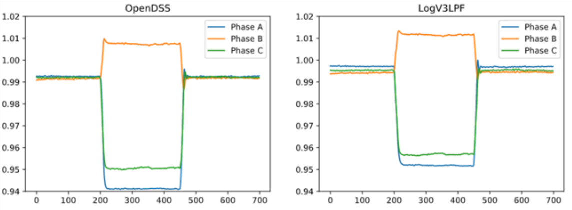



**Figure 8:** Voltage Imbalance attack

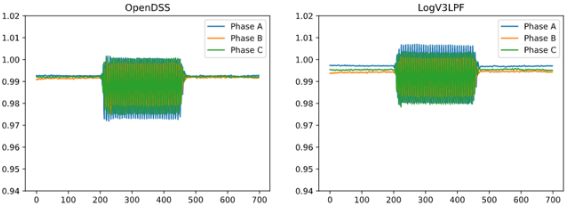
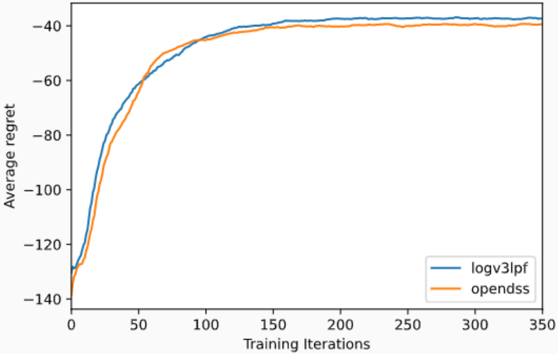
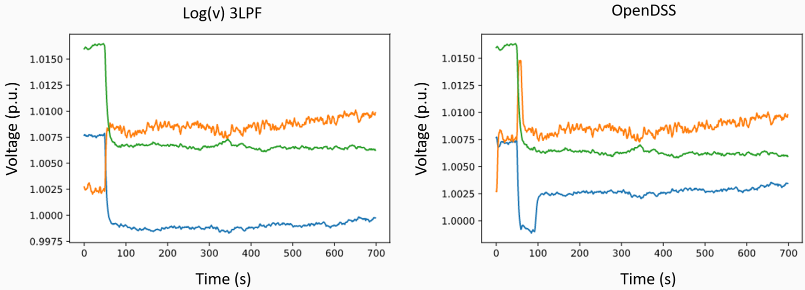



**Figure 9:** Voltage Oscillation attack

# Training with OpenDSS vs Log(V) 3LPF. Average regret



# Training with OpenDSS vs Log(V) 3LPF. Policy evaluation



# Limitations and summary

**Limitations:**

- OpenDSS rarely inverts the system matrix
- OpenDSS convergence achieved in 3-4 iterations

**Summary:**

- We provide a linear PF solver with all necessary modeling capabilities
- Fast inverse computation
- Convergence is guaranteed (single iteration, linear)
- Model is, unlike OpenDSS, not sensitive to fast changes in boundary conditions