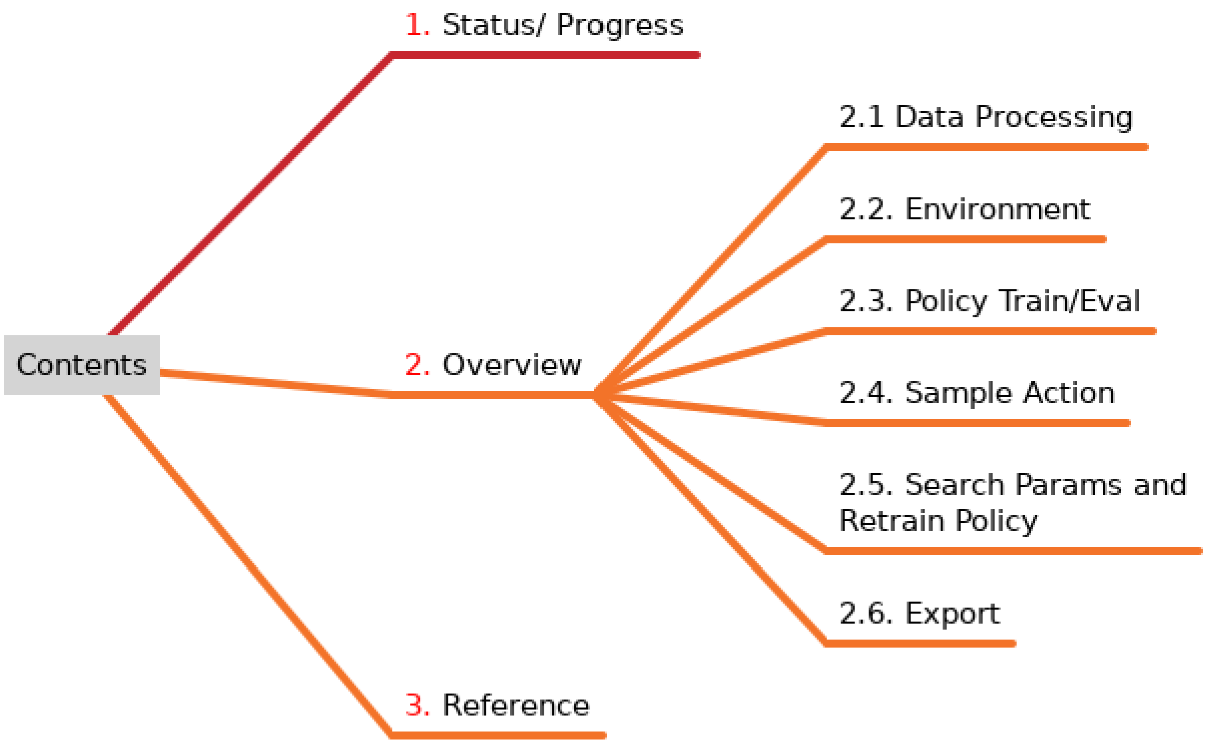
Training Prosumer Agents with Reinforcement Learning.

`>>>` **Biweekly Report 8.** ($27^{th} June - 10^{th} July : 2024$)



---

1. Status/ Progress

Current Iteration

- [x] Brief overview and status of the project

Next Iteration

- [ ] Full report contents and formats

2. Overview

development pipeline

> data --> custom env --> policy learn/eval --> test --> search/retrain --> export

2.1 Data Preprocessing

Description

Prosumer dataset containing the following columns were utilized:

- `Timestamp UTC :` An index column with frequency of `15mins` is used to temporally represent the dataset values from prosumer household.
- `Customer ID :` Identification number of the Prosumer. Two prosumers with ID of 6 and 7 are used.
- `Power Household :` Power consumed by a household. (Watt hour)
- `Power PV :` Power generated from the photovoltaics, represented with (-ve) power for net exchange to the grid (Watt hour)
- `Battery Initial SoC` : Initial state of charge of a battery, i.e. only the first entry of the dataset of SoC is used as part of the observation.

Energy day ahead auction price dataset containing the following columns were utilized:

- `Timestamp UTC :` An index column with frequency of `1hour` is used to temporally represent the day ahead auction price of the energy.
- `Auction Price :` A variable day ahead auction price of energy in Euro per `MWh` is used as input observation. This price is same for any prosumer within `DE_LU` region.

Pre-Processing

- `prosumer` and `auction price` are merged to a single dataset frame, resampling the auction price to a frequency of `15mins`, with common index of `Timestamp UTC`. The range of the dataset timestamp is an year(`2021-08-15 00:00:00+00:00` - `2022-08-15 00:00:00+00:00`) for two prosumer, total of length 70082 timesteps.

- missing values are dealt by replacement with `0`, outliers are clipped to be in a range, units are converted to common unit for relevant columns:

  - Power PV, Power Household: `Wh` to `Kwh`
  - Auction price: `Eur/Mwh` to `Eur/Kwh`

Observation Inputs

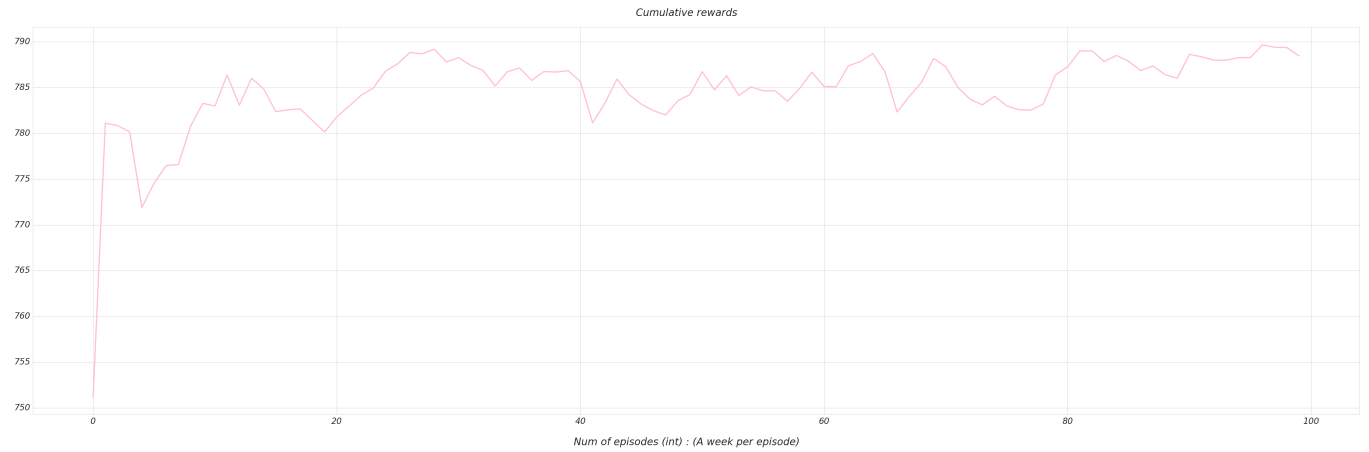
- A single timestep input contains [Power Household, Power PV, Auction price, and Current Battery soc]
- Training and Testing set are split in 90:10 ratio.

2.2 Environment

- Custom gymnasium environment with observation space and action space(-1, 1), with the range of max and min value they can take, for given observation inputs is initialized.
- Step function is used to iterate step transition of an agent in the environment.
  - soc constraint is applied to each step making sure the battery is not charging when full and discharging when empty.
  - action is rescaled to the range (-11, 11)
  - net grid exchange is calculated as a power balance equation. $\text{net exchange} = \text{power household} + \text{power pv} + \text{action}$
  - reward is calculated as combination of cost reward computed from net exchange and battery soc constraint that encourages the soc to stay within desired range.
    - cost reward = $\text{power sell cost} + \text{power household cost}$
    - soc reward = cost reward +/- 0.5 (+ encourage, - discourage) if soc is less than 50% or more than 50% of battery capacity respectively.
  - for each step, battery soc is updated with respective energy content resulting from charge and discharge action.
  - `next observation`, `current_reward`, `terminated/truncated` signal and `info` are returned per steps and is part of the agent's experience trajectory untill termination criteria is met.
- Before using the environment, it is reset to a initial state.

2.3 Policy Training/Evaluation

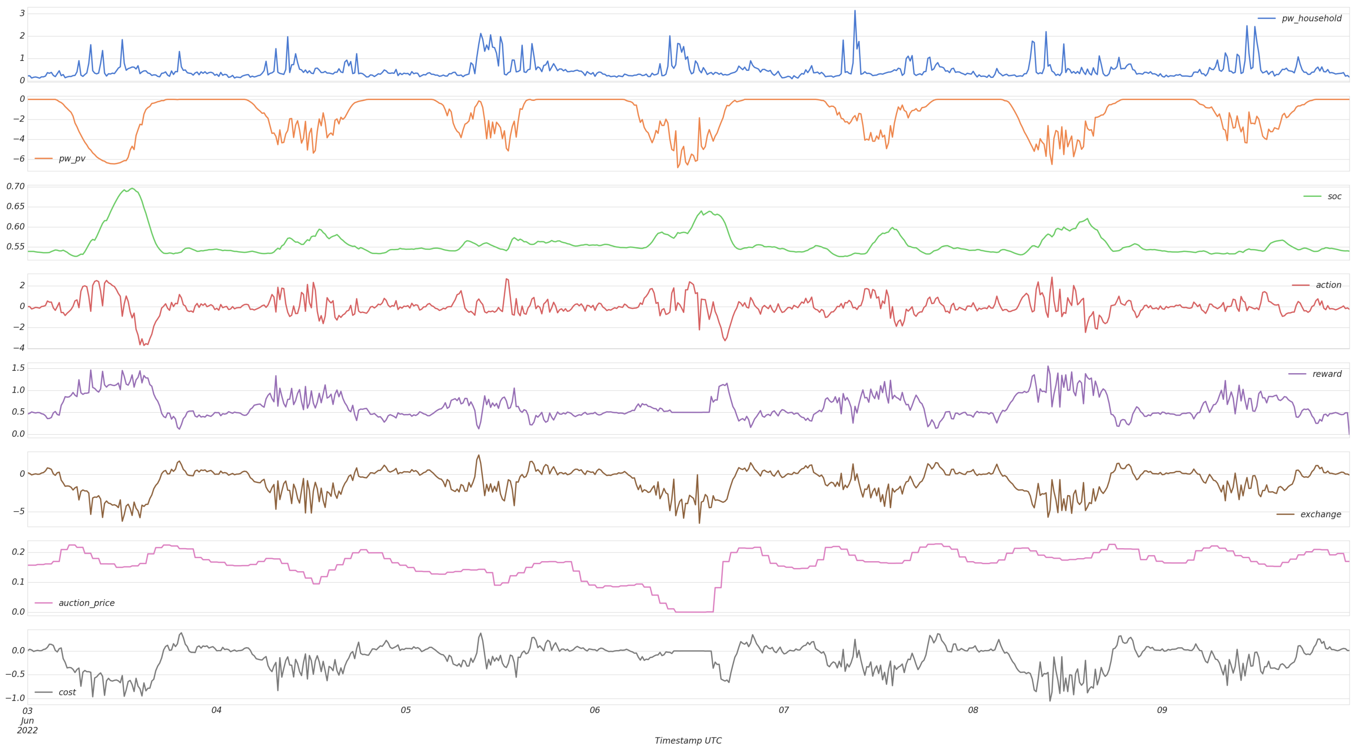
- Environments used for training and evaluation are initialized and reset.
- For specified number of episodes(a complete trajectory)
  - policy is trained and parameters of policy network are updated
  - policy is evaluated for n number of episodes and mean reward is computed for returned cumulative reward for those episodes.
- A sample figure for ppo with mean cumulative reward is shown below. Each episode here contains 2 weeks of timesteps(`24*4*7*2`).



- if needed the policy is retrained by loading from previous trained policy checkpoint and is updated and evaluated.

2.4 Sample Action (Using trained policy)

- A test environment is initialized and reset.
- For each observation step in the test set episode, action is predicted from the current policy.
    - The action is rescaled to the range (-11, 11).
    - The observation inputs and resulting action, reward, exchange, cost are saved as stats for graphical comparison as shown below.



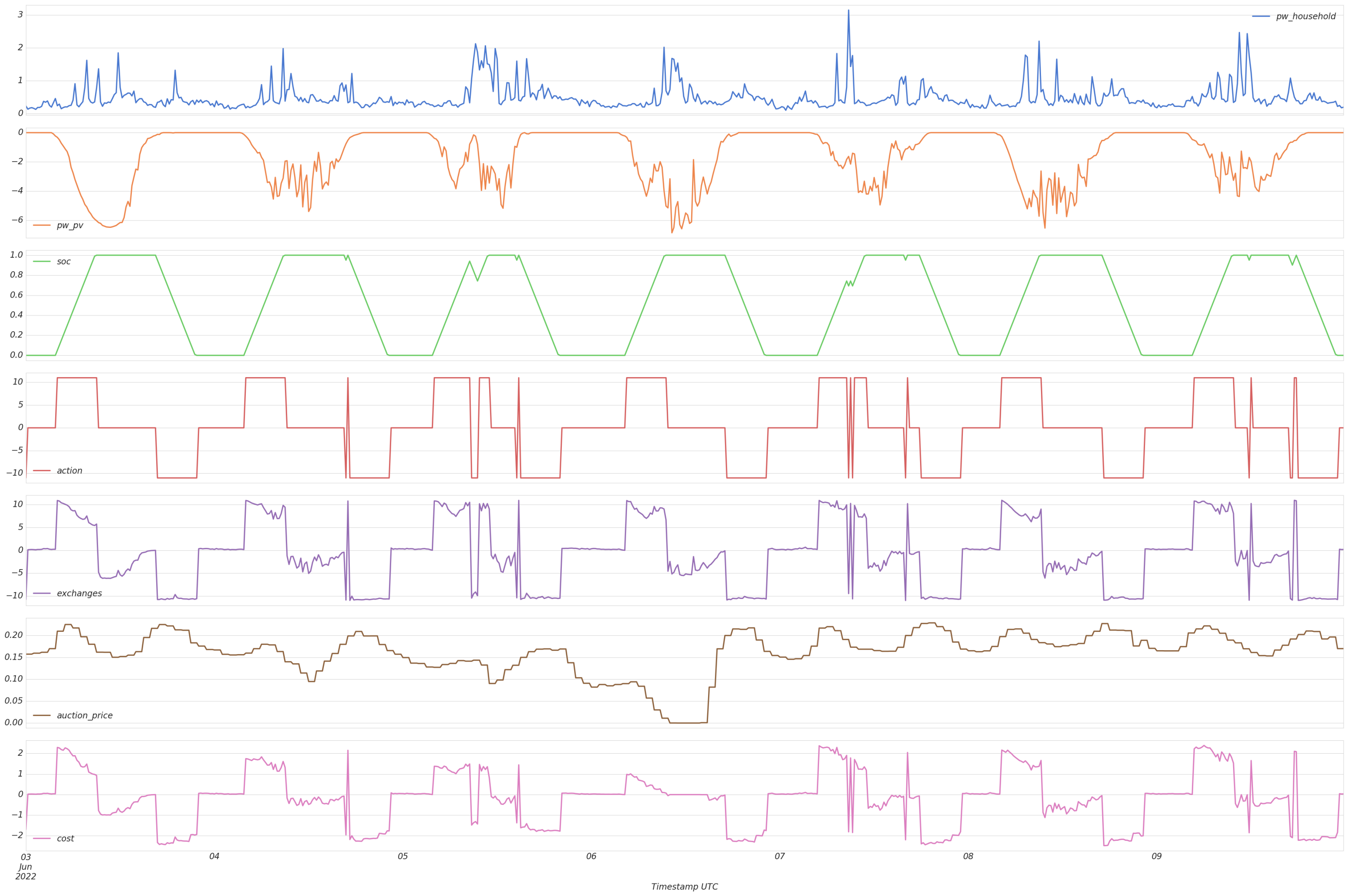
Rule based Policy

- A rule based policy is used to compare the resulting cost from the reinforcement learning policy to a conditional policy that is based on input observations.

Steps

- For each observation step in the test set episode, net exchange to the grid is calculated as a power balance equation. $\text{net exchange} = \text{power household} + \text{power pv} + \text{action}$
- If net exchange to the grid is negative(since pv is -ve during data collection), the household produced more pv that it consumed.
    - The excess pv is used to charge the battery `if it is not full` (i.e. soc != 1.0)
    - The excess pv is sold to the grid for current auction price `if the battery is full`.

- If net exchange to the grid is positive, the household consumed more pv that it produced.
    - The excess demand is mitigated by the `battery if it is not empty` (i.e. soc != 0.0)
    - The excess demand is met by buying from the grid for current auction price `if the battery is empty`.
- The resulting stats are accumulated to create a comparison figure as shown below.



Cost Comparison

A cumulative cost of an test episode is computed for rl and rule based policies. Following table shows the comparison between the two.

Models ->	PPO	SAC	TD3	RBC
Total Cost	-95.70052	-91.419365	-51.15069	-142.58888

2.5 Search params and Retrain policy.

- For n number of trials, better hyper parameters for each policy gradient algorithms are searched. The policy network is retrained, in order to get stable policy producing consistent actions.
- New hyperparameter are progressively added for search and retraining, replacing the default values.

2.6 Export policy

- The trained policy is saved as a checkpoint and is converted to onnx format(an open format for model exports). This model is served for inference through api calls.
