Optimisation of Switching States
using Reinforcement Learning

AJIT GUPTA
AKANKSHA DHANCHOLIA
KUN LIN TSAI
NAGARJUNA GOTTIPATI
SARPER SAHIN
SUMEDH THANEKAR
TEJAS CHOUDEKAR
YUN YUN YANG

Agenda

01 Introduction of Problem

02 Reinforcement Learning

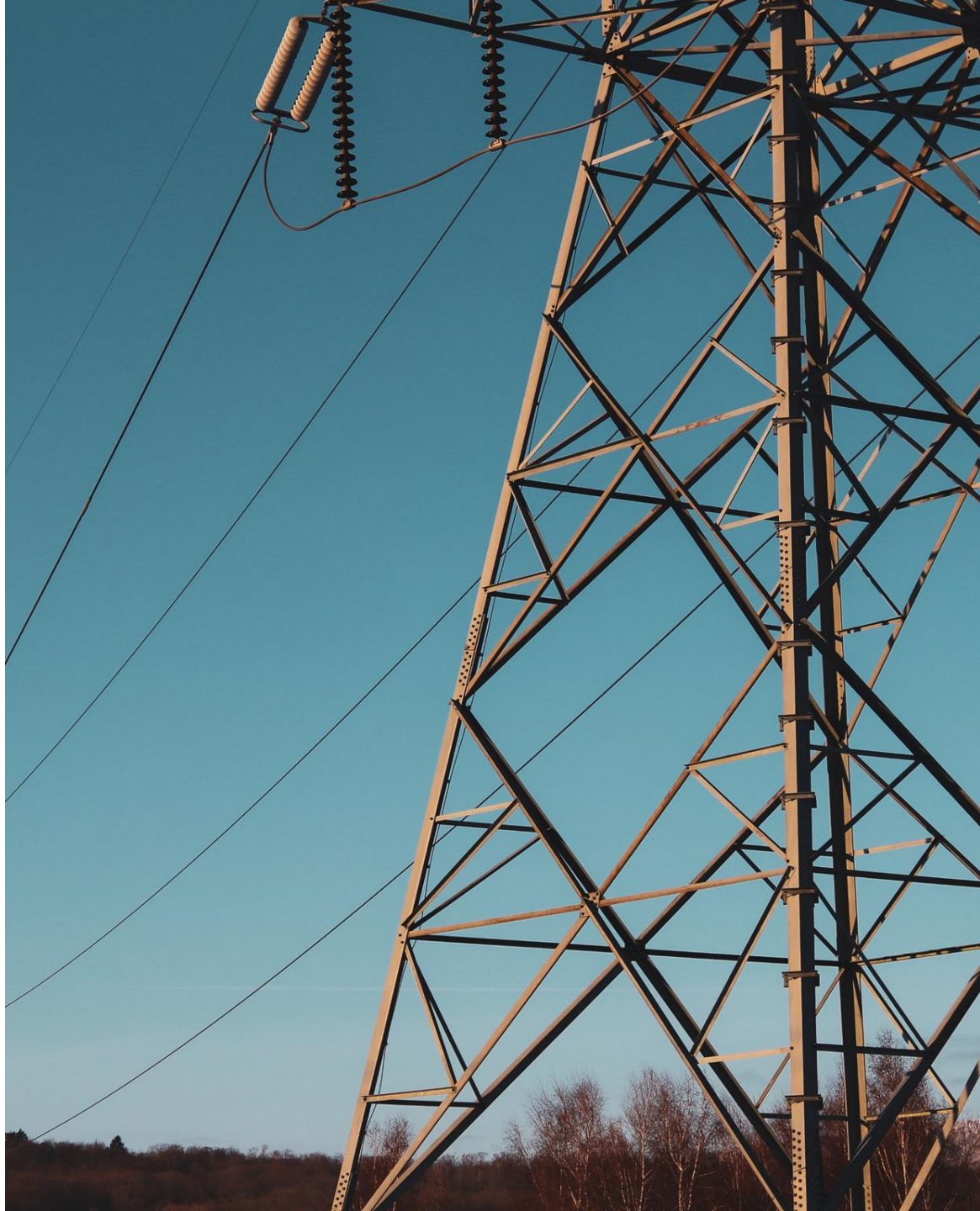
03 Grid2Op Framework

04 Agent Creation

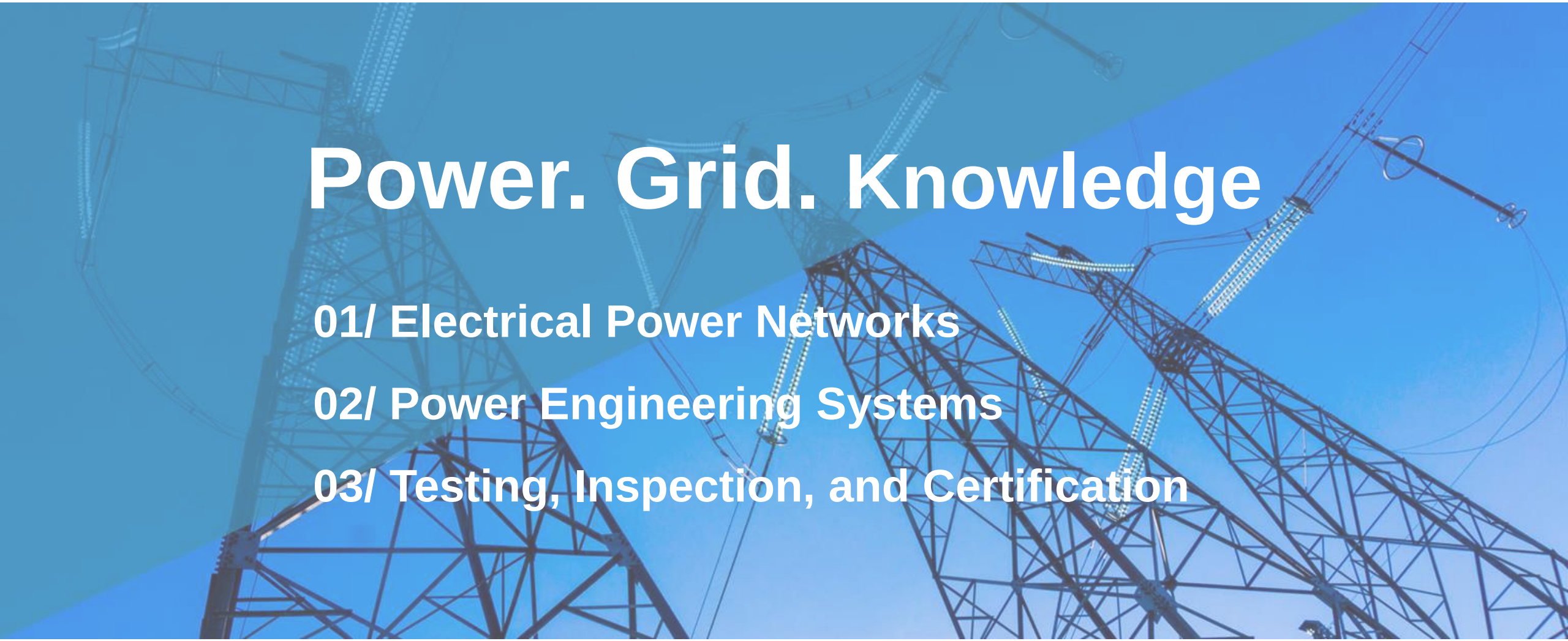
05 Baseline Agents

06 Visualization using Grid2Viz

07 Conclusion



About FGH


Power. Grid. Knowledge
01/ Electrical Power Networks
02/ Power Engineering Systems
03/ Testing, Inspection, and Certification

Power Grid: How does it work?

Power grid consists of below elements:

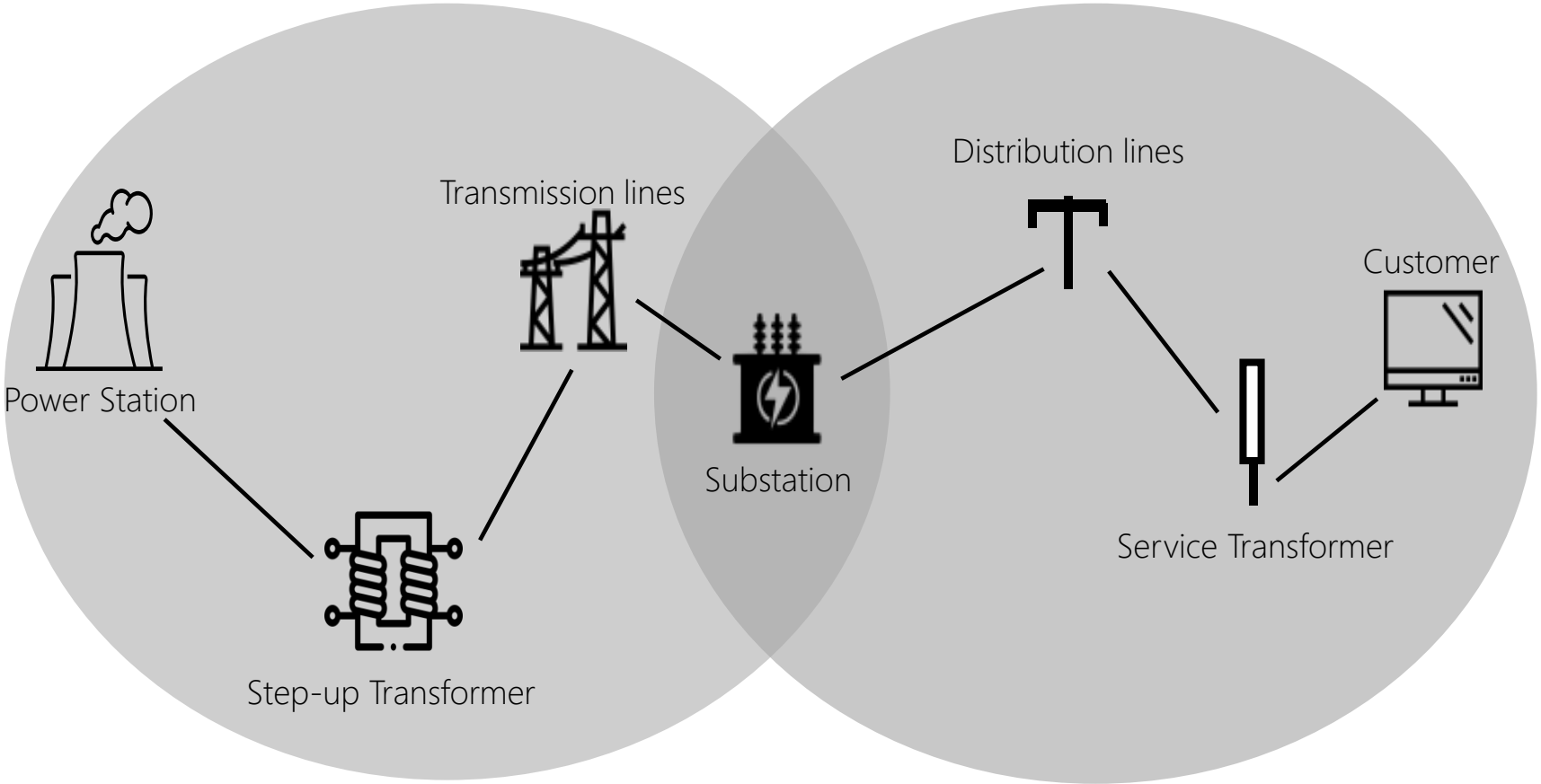


Fig1: Elements of a Power Grid and its working

Underlying Problem

Loading of electrical transmission grid increases

Sudden demands or excesses of energy cause brief changes in voltage that can cause an overload.

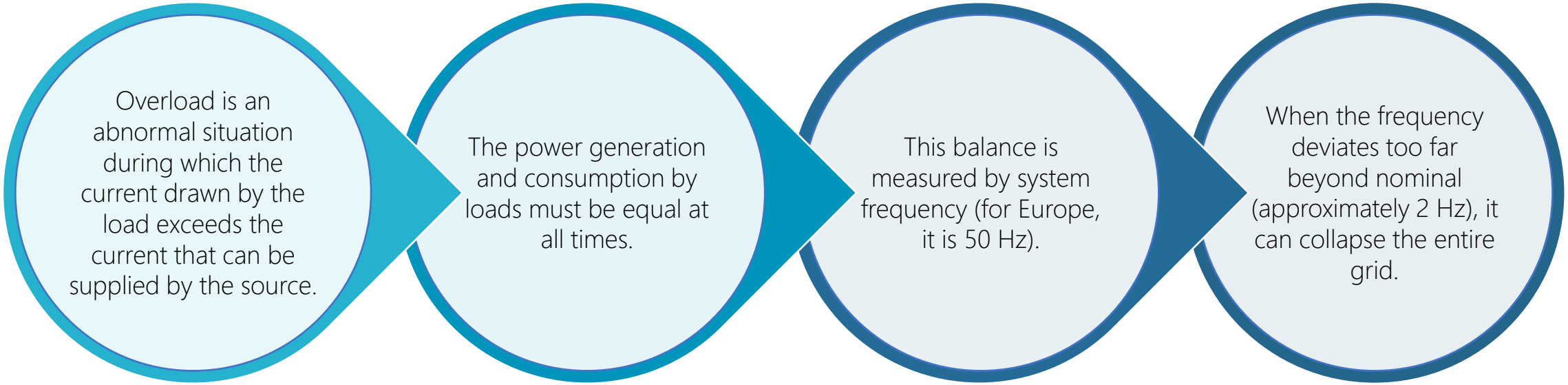
Higher share of renewable based decentral power plants

Renewable generation is a lot more fluctuating than the conventional, this causes new bottlenecks which leads to strains in grid.

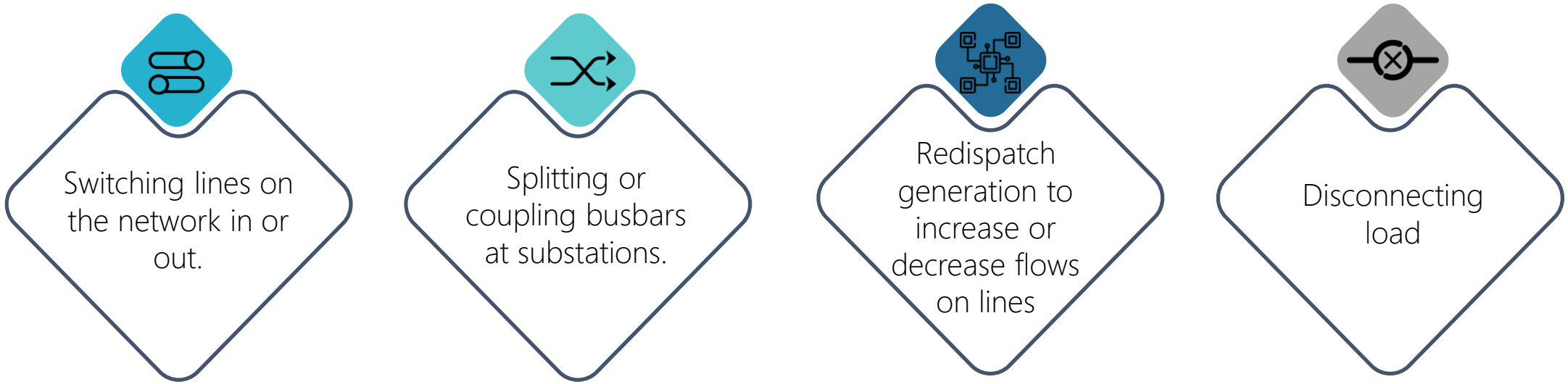
Coupling of the European electricity markets

Cross-border balancing of fluctuating generation from renewable energies, if the grid capacity is insufficient, the congestion resulting from the market outcome must be eliminated through congestion management measures.

Overloading: Why to avoid it?



Overloading: How to avoid it?



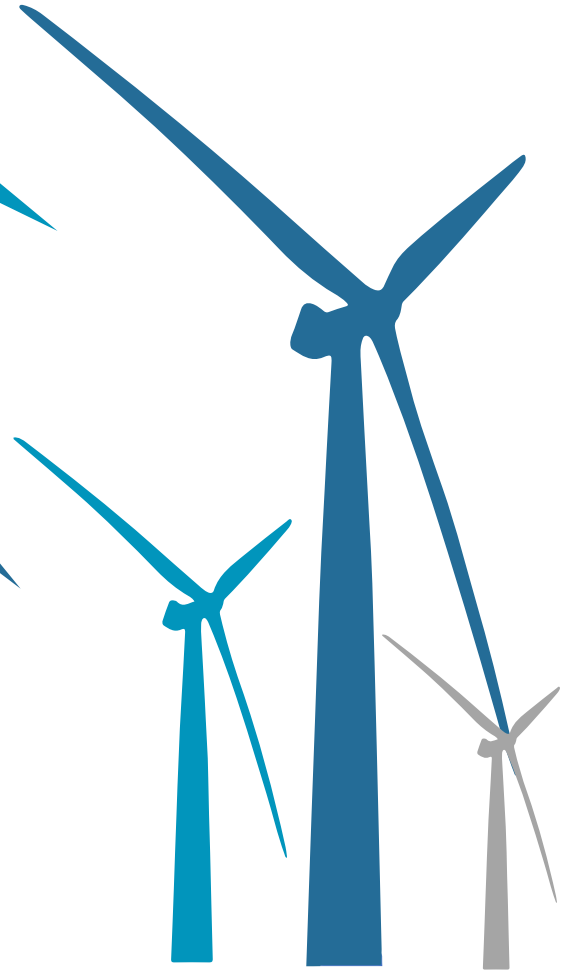
Overloading: Why it's a difficult challenge?

1 Constantly Monitoring

For a safe and reliable transmission of electricity, it is constantly monitored and managed by human experts in the control room.

2 Potential Attacks

Several attacks could happen, which leads to voltage and current overload.



3 Limitation of Human Experts

Human experts can take limited actions in case of an attack.

4 Enormous possible Actions

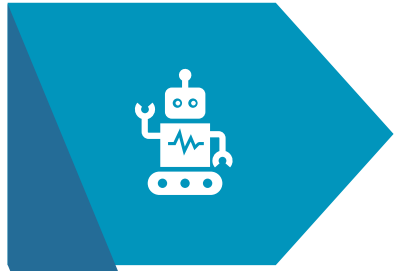
Number of possible actions to consider is enormous.

Methodologies

Possible Approaches

Characteristics of the problems:

- High number of possible switching states
- Discrete characters of various decision variables
- Combinatorial problem
- Complex sequential decision making task



RL Approach

Self learning
Instant specific

Hybrid Approach

Optimality
Efficiency

Heuristic Approach

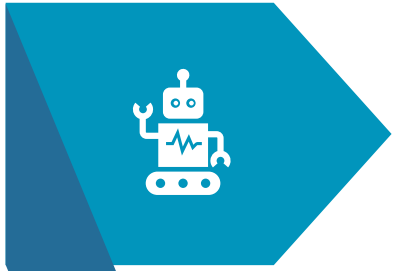
Generality
Efficiency

Exact Approach

Generality
Optimality

Exact Approach

- In theory, it tries to find the optimum solution if it is possible.
- In reality, it becomes computationally expensive for large datasets.



RL Approach

Self learning
Instant specific

Hybrid Approach

Optimality
Efficiency

Heuristic Approach

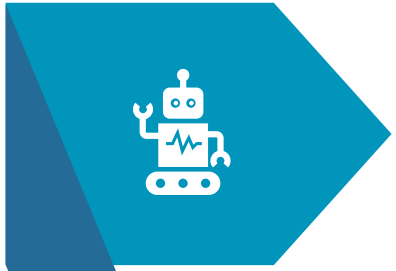
Generality
Efficiency

Exact Approach

Generality
Optimality

Heuristic Approach

- Try to solve a given optimization problem as good as possible within acceptable runtime.
- Likely to neglect optimality, precision or accuracy. Also, there is no lower bound.



RL Approach

Self learning
Instant specific

Hybrid Approach

Optimality
Efficiency

Heuristic Approach

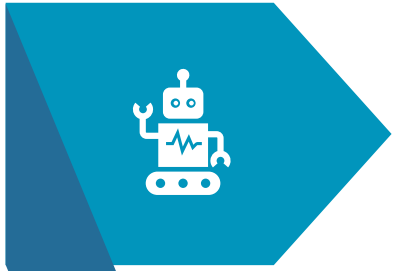
Generality
Efficiency

Exact Approach

Generality
Optimality

Hybrid Approach

- Find better solutions with multi-algorithms such as mathematical programming and greedy heuristics.
- Works well with particular instances, but computationally expensive



RL Approach

Self learning
Instant specific

Hybrid Approach

Optimality
Efficiency

Heuristic Approach

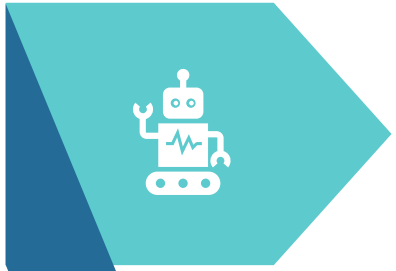
Generality
Efficiency

Exact Approach

Generality
Optimality

RL Approach

- Learning the optimal behavior in an environment to obtain maximum reward.
- The model progressively improves the performance with the training process.
- Works well with the complex sequential decision-making problems.



RL Approach

Self learning
Instant specific

Hybrid Approach

Optimality
Efficiency

Heuristic Approach

Generality
Efficiency

Exact Approach

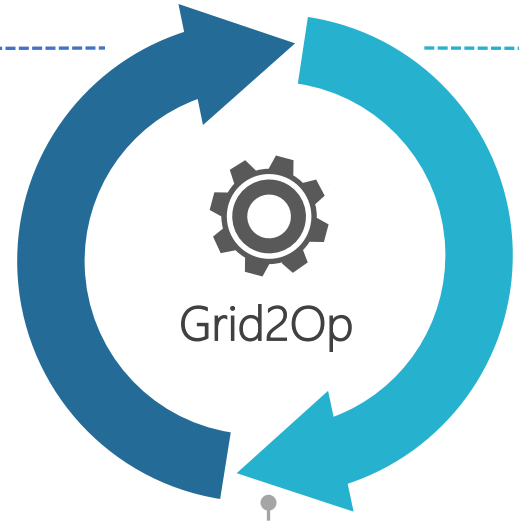
Generality
Optimality

Reinforcement Learning

Why Reinforcement Learning?

SEQUENTIAL DECISION MAKING

Grid2Op aims to model "sequential decision making" that could be made by human operator and applied to power systems



REINFORCEMENT LEARNING VIEWPOINT

Grid2Op adopts "reinforcement learning" point of view and is compatible with other reinforcement learning libraries

01/
Powergrid state switching is a dynamic environment.

02/
Consideration of time-coupling constraints for switching measures

03/
Solvability of the optimization problem in acceptable computing time

What is Reinforcement Learning?

- Reinforcement learning is about taking suitable action to maximize reward in a particular situation. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.

- Agent: an entity that performs actions in an environment in order to optimize a long-term reward
- Environment: the world in which the agent performs actions
- Action (A_t): A_t is the set of all possible moves the agent can make which cause a change in the environment
- Rewards (R_t): the reward R_t is a scalar feedback signal which indicates how well the agent is doing at step time t
- State (S_t): a state refers to the current situation returned by the environment

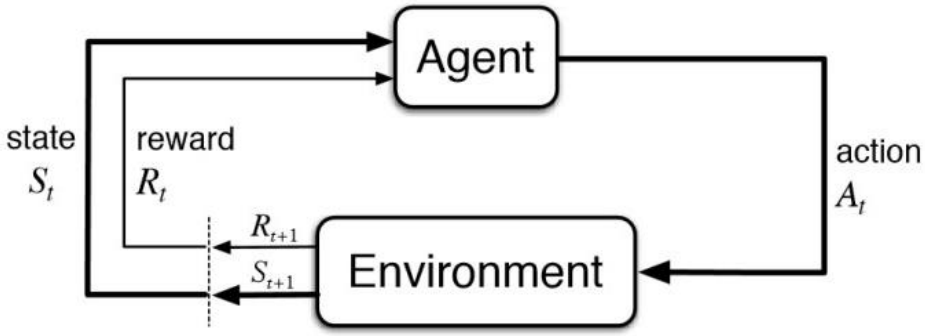


Fig2: Elements of a Reinforcement Learning Model

What is the Policy? How does it affect an Agent?

- The agent is composed of the policy and the learning algorithm
- The policy is the function that maps observations to actions
- The policy is independent of the agent's actions called off-policy.
- The learning algorithm is the optimization method used to find the optimal policy
- The final goal in a reinforcement learning problem is to learn a policy, that gives the optimal action

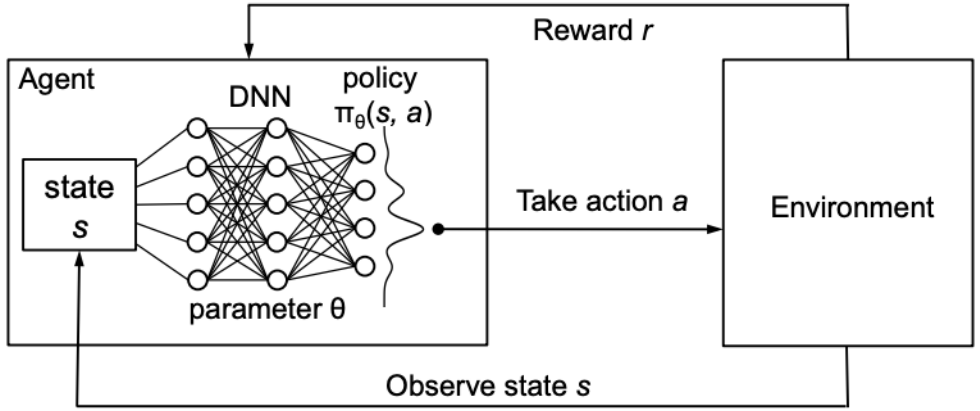


Fig3: Working of a Reinforcement Learning Model

Agent Overview

	Implemented?	On-Policy	Off-Policy
Deep SARSA	✕	✕	
DQN			✕
PPO		✕	
Switching Agent	✕		✕
Topology Agent	✕		

Table1: Overview of the different agents used

Grid2Op Framework

Grid2Op Framework

Grid2Op is a python based, user-friendly framework that models realistic concepts on real world power grid systems, allowing you to develop, train or evaluate performances of "agent" or "controller" that acts on an environment in different ways.

Environment
Defines the higher-level representation of the world with which an Agent will interact.

Agent
An entity that acts on the Environment.

Observations
One of the information that Agent receive before taking any action on environment.

Action
Define some actions on the underlying power grid. These actions are either made by an agent, or by the environment.

Grid2Op Environment - I

- Grid2Op provides a set of default environments that can be used to train and test the agents on.

env name	grid size	maintenance	opponent	redisp.	storage unit
l2rpn_case14_sandbox	14 sub.	✗	✗	✓	✗
l2rpn_wcci_2020	36 sub.	✓	✗	✓	✗
l2rpn_neurips_2020_track1	36 sub.	✓	✓	✓	✗
l2rpn_neurips_2020_track2	118 sub.	✓	✗	✓	✗
l2rpn_icaps_2021	36 sub.	✓	✓	✓	✗
l2rpn_wcci_2022	118 sub.	✓	✓	✓	✓
* educ_case14_redisp *	14 sub.	✗	✗	✓	✗
* educ_case14_storage *	14 sub.	✗	✗	✓	✓
* rte_case5_example *	5 sub.	✗	✗	✗	✗
* rte_case14_opponent *	14 sub.	✗	✓	✗	✗
* rte_case14_realistic *	14 sub.	✗	✗	✓	✗
* rte_case14_redisp *	14 sub.	✗	✗	✓	✗
* rte_case14_test *	14 sub.	✗	✗	✗	✗
* rte_case118_example *	118 sub.	✗	✗	✓	✗

For evaluation

For Testing

Fig4: Grid2Op available environments

Grid2Op Environment - II

For this presentation we will use a dedicated environment called "l2rpn_neurips_2020_track1_small", which has 36 substations.

The environment contains all essential information about the power grid such as:

- Substations where the loads, generators and transmission lines are connected
- Power lines and flows
- Chronics, it provides data to change the input parameter of a power flow between one time step and the other

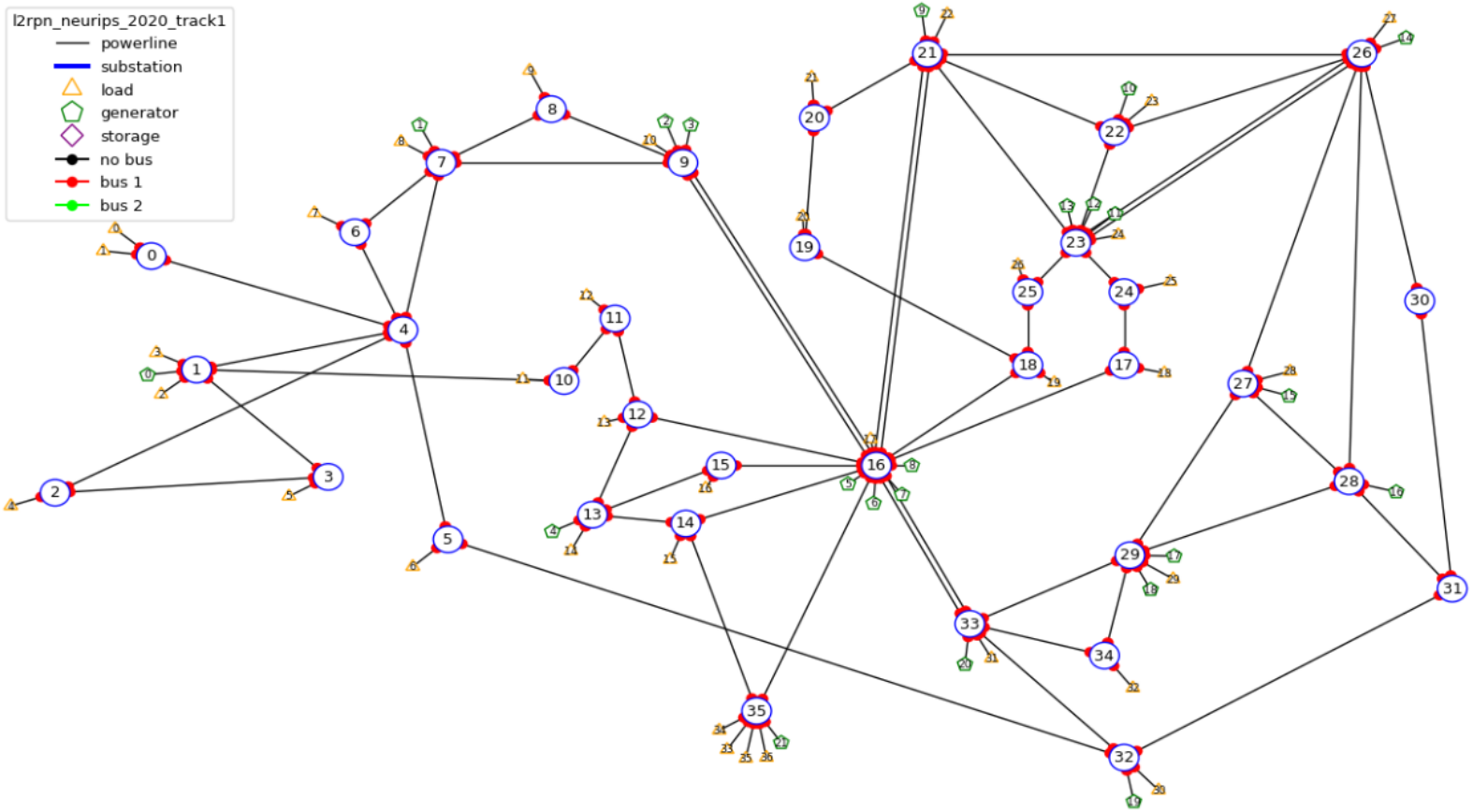


Fig5: l2rpn_neurips_2020_track1_small environment

Grid2Op Observation I

In a “reinforcement learning” framework, a grid2op. Agent receives two information before taking any action on the environment.

- Reward: It describes how well the past action performed
- Observation: This gives the agent partial, noisy, or complete information about the current state of the environment

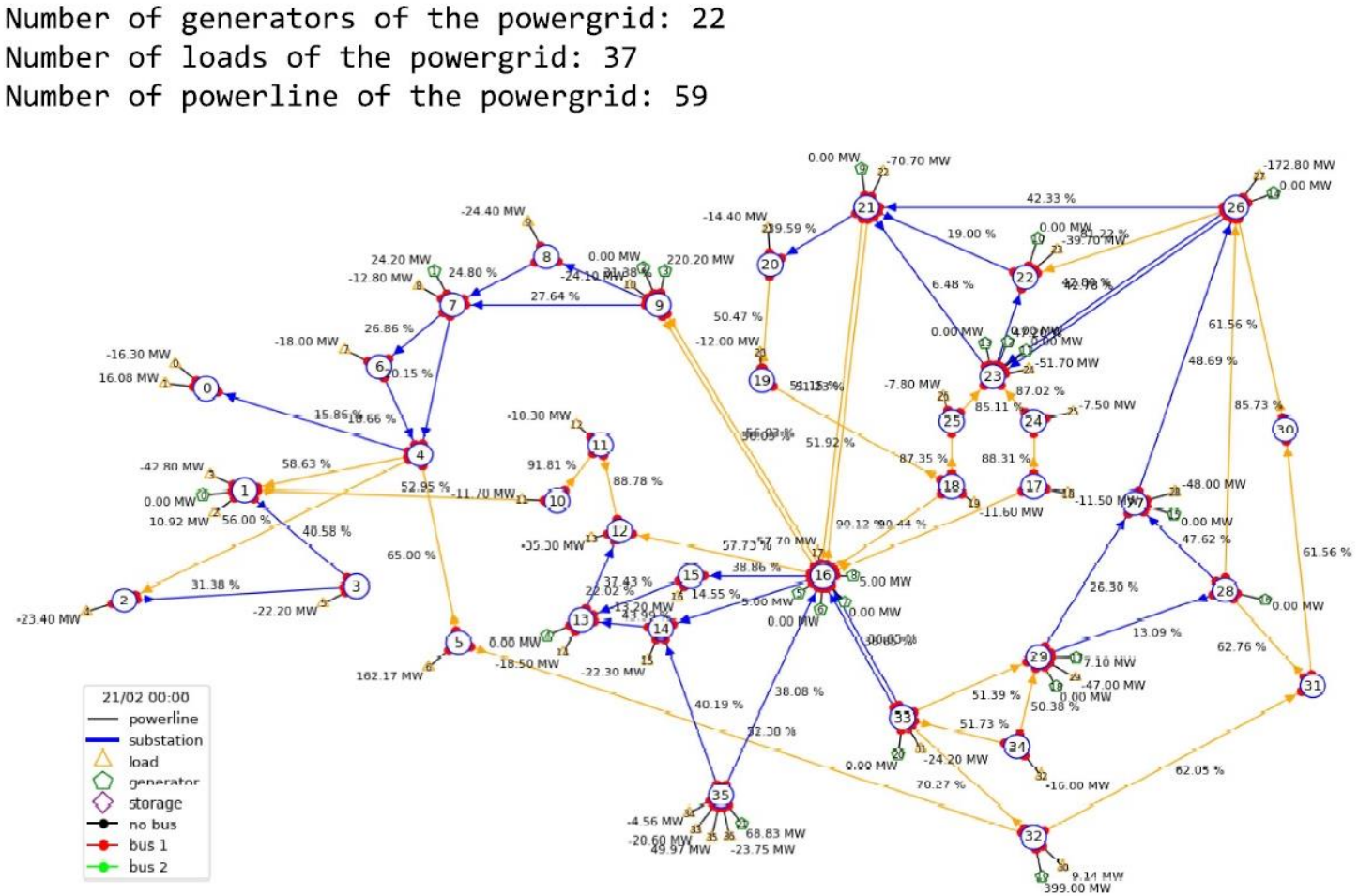


Fig6: “l2rpn_neurips_2020_track1_small” observations

Grid2Op Observation II

Name	Type	Description
Rho	Float	The capacity of each powerline. It is defined at the observed current flow divided by the thermal limit of each powerline.
line_status	Bool	Gives the status (connected / disconnected) for every powerline (True at position i means the powerline i is connected)
time_before_cool down_line	Int	How much cooldown time for each line is left. An agent cannot perform its action on these lines before the cooldown time ends.
n_gen	Int	Number of generator
n_load	Int	Number of load in environment.
n_line	Int	Number of lines connected to the substations
Simulate		This method is used to simulate the effect of an action on a forecast power grid state. This forecast state is built upon the current observation

Table2: Details of features in the state

Grid2Op Action

The environment we use contains action space of 494 actions. The agent can apply actions on environment to manage the power grid.

Discrete actions (Topology modification)

- Change the status of powerlines
- Set the status "connected"/"disconnected" for powerline

Continuous actions

- Re-dispatching
- Curtailment
- Manipulation of storage units

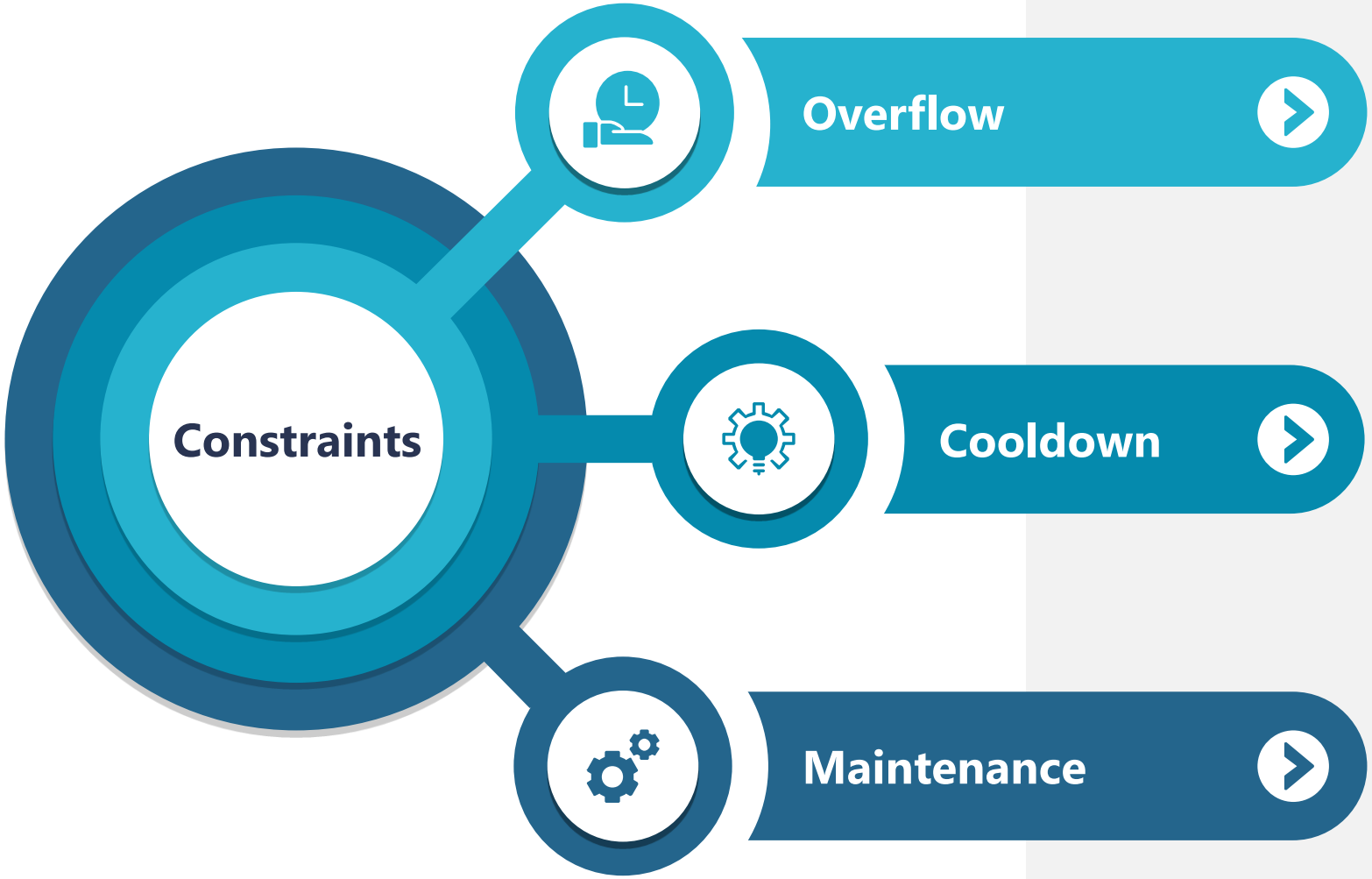
Grid2Op Agent

In this RL framework, an Agent is an entity that acts on the Environment (modelled in Grid2Op as an object of class Environment). In Grid2Op such entity is modelled by the BaseAgent class.

Agents	Description
DoNothingAgent(action_space)	This is the most basic BaseAgent.
PowerLineSwitch(action_space)	This is a GreedyAgent example, which will attempt to disconnect powerlines.
RandomAgent(action_space[, ...])	This agent acts randomly on the power grid.
TopologyGreedy(action_space)	This is a GreedyAgent example, which will attempt to reconfigure the substations connectivity.
MLAgent(action_space[, action_space_converter])	This agent is based on Machine Learning (ML) algorithms.

Table3: Details of agents available in grid2op framework

Grid2Op Constraints



Lines can hold for 3 time steps while overflow then a line is automatically disconnected.

Cooldown time (3 time steps later) for each component to reconnect.

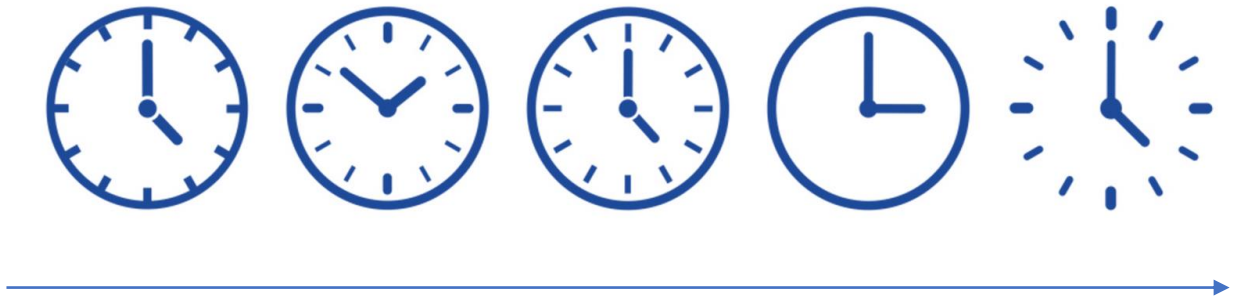
Stochastic events called maintenance that happens intermittently. During maintenance, a line is disconnected by force, and it cannot be reconnected.

Grid2Op Evaluation

Power grid systems have one primary objective: to allow electricity transmission from the producers to the consumers as efficiently as possible. In this case, we want to reduce the total cost for each episode and evaluate our Agent in the following ways:

- Rewards from runner

  The highest number of scenarios, rewards and time step (1 timestep = 5 mins) survival without game over would indicate better performance



More the timesteps, better the agent survival

Agent Creation

Agent Creation - SwitchingAgent

- SwitchingAgent would perform greedy actions on the grid, which will choose among:
  - Setting the **lines_or_id** and **lines_ex_id** to bus 1
  - Changing the **bus of origin** end of powerline
  - Changing the **bus of extremity** end of powerline

- Performance of SwitchingAgent:
  - Limited actions
  - poor performance on complex environment.

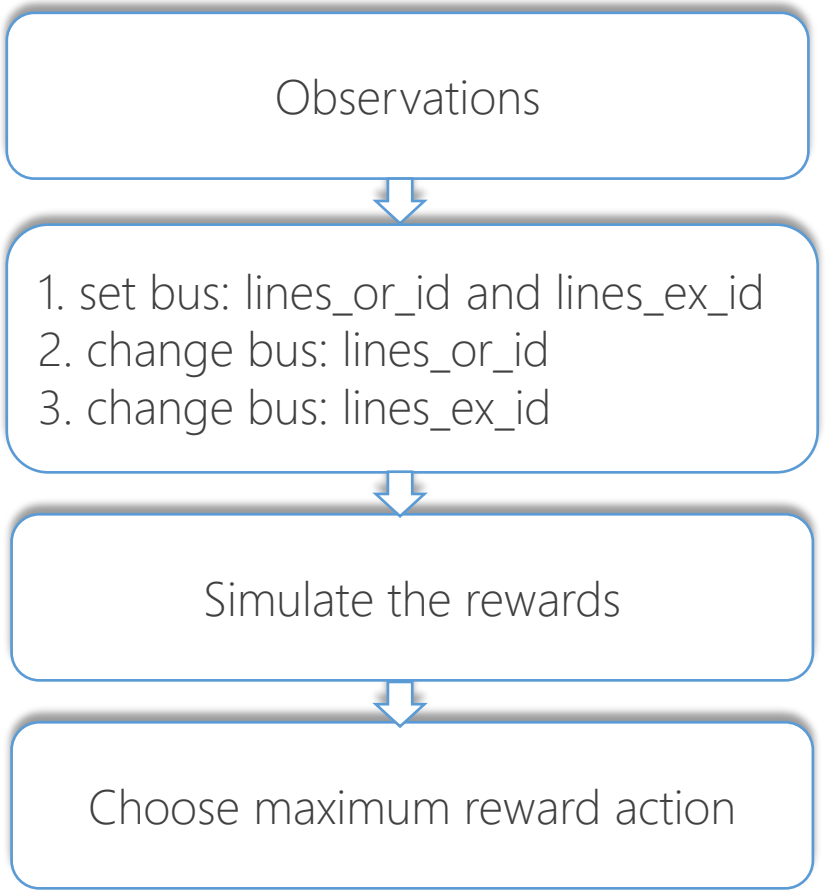


Fig7: Flow diagram for SwitchingAgent

Agent Creation - TopologyAgent

- Just considering topology change actions, it's exponential. It is possible to change the status of the powerline, meaning connecting or disconnecting them. If there are N powerlines, the number of such possible actions is $2^{N.}$ In our case, we have ~66K unitary actions.
- Since not all actions will generate satisfactory results, we first feed multiple scenarios to our function and then, based on overload result, we filter out the most frequent actions. This results to 177 actions.
- Performance of Topology Agent:
  - This reduction in action space will enable us to make appropriate decision fast in case of an attack on power grid.
  - Good performance even on complex environments
  - Filtering actions is time consuming

Agent Creation - TopologyAgent

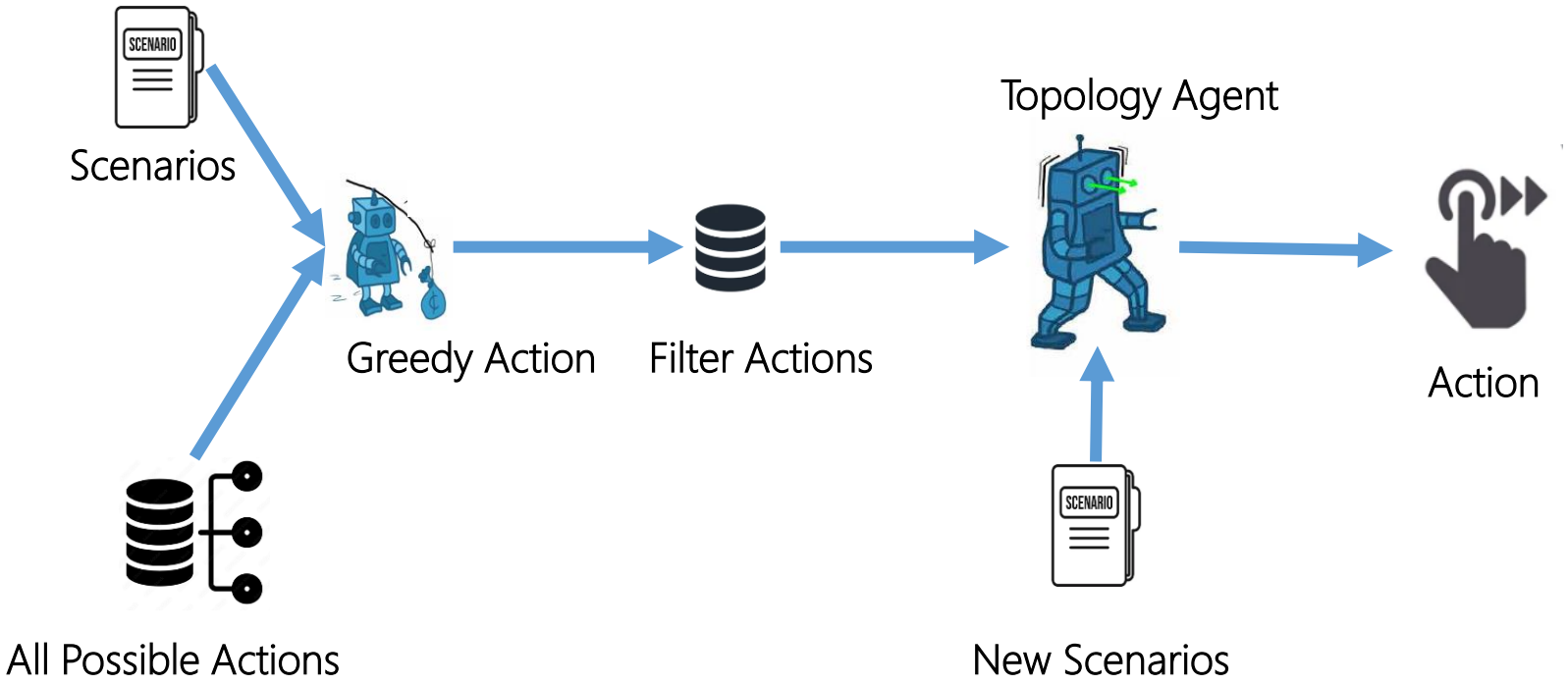


Fig8: General Overview of the TopologyAgent

Agent Creation - TopologyAgent

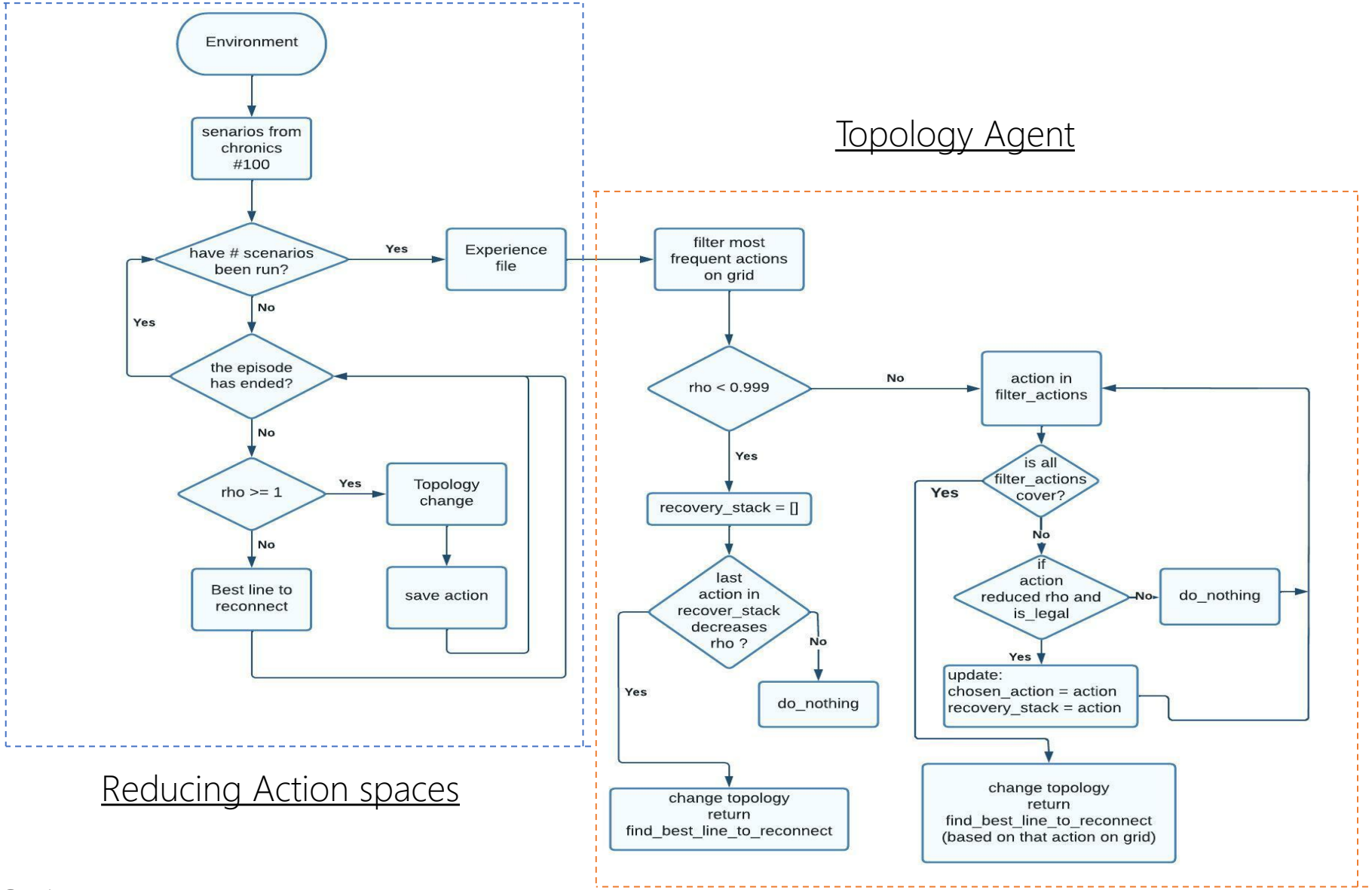


Fig9: Flow diagram for TopologyAgent

Agent Creation - TopologyAgent

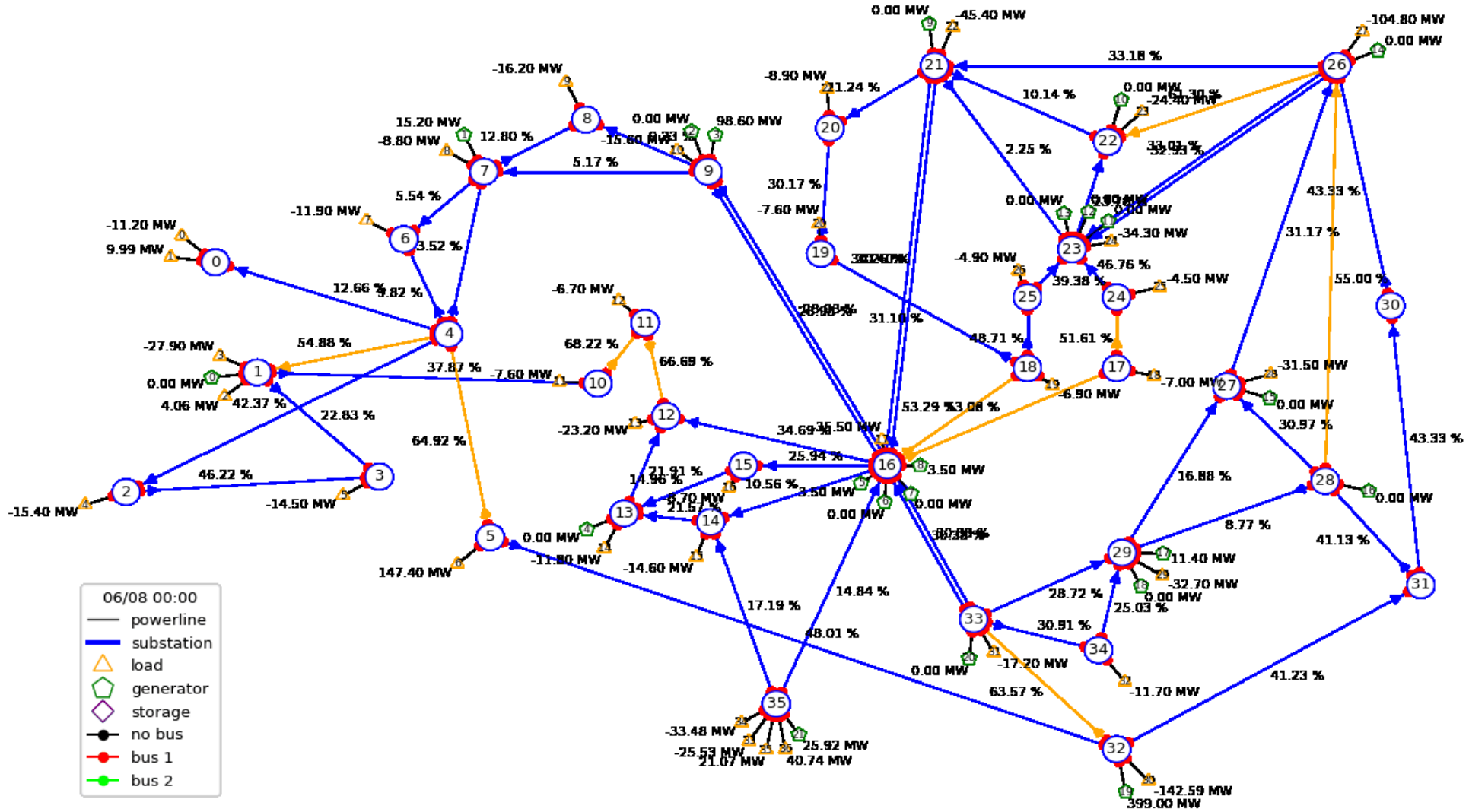
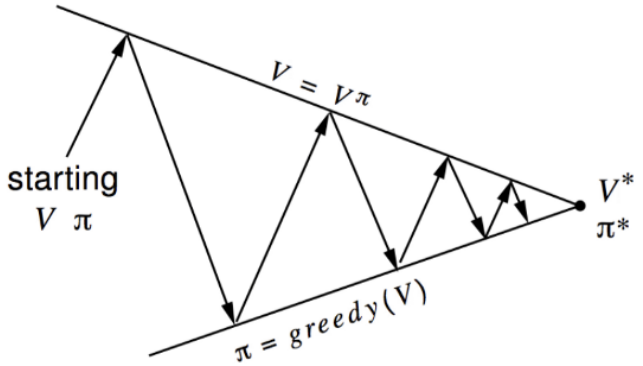


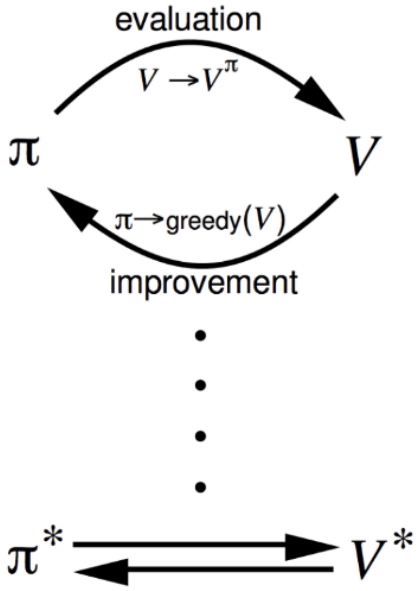
Fig10: Performance of TopologyAgent on scenario

Agent Creation - Deep SARSA

SARSA Learning

- SARSA → State-Action-Reward-Next State- Next Action
- SARSA learning is an on-policy method.
- That implies, the same policy is used for exploration and estimation of next action.





Policy evaluation Estimate v_π
Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$
Any policy improvement algorithm

π is policy to be trained

V are the values that we get from policy

π^* is optimal policy

V^* are optimal values obtained from optimal policy

Agent Creation - Deep SARSA

General Flow

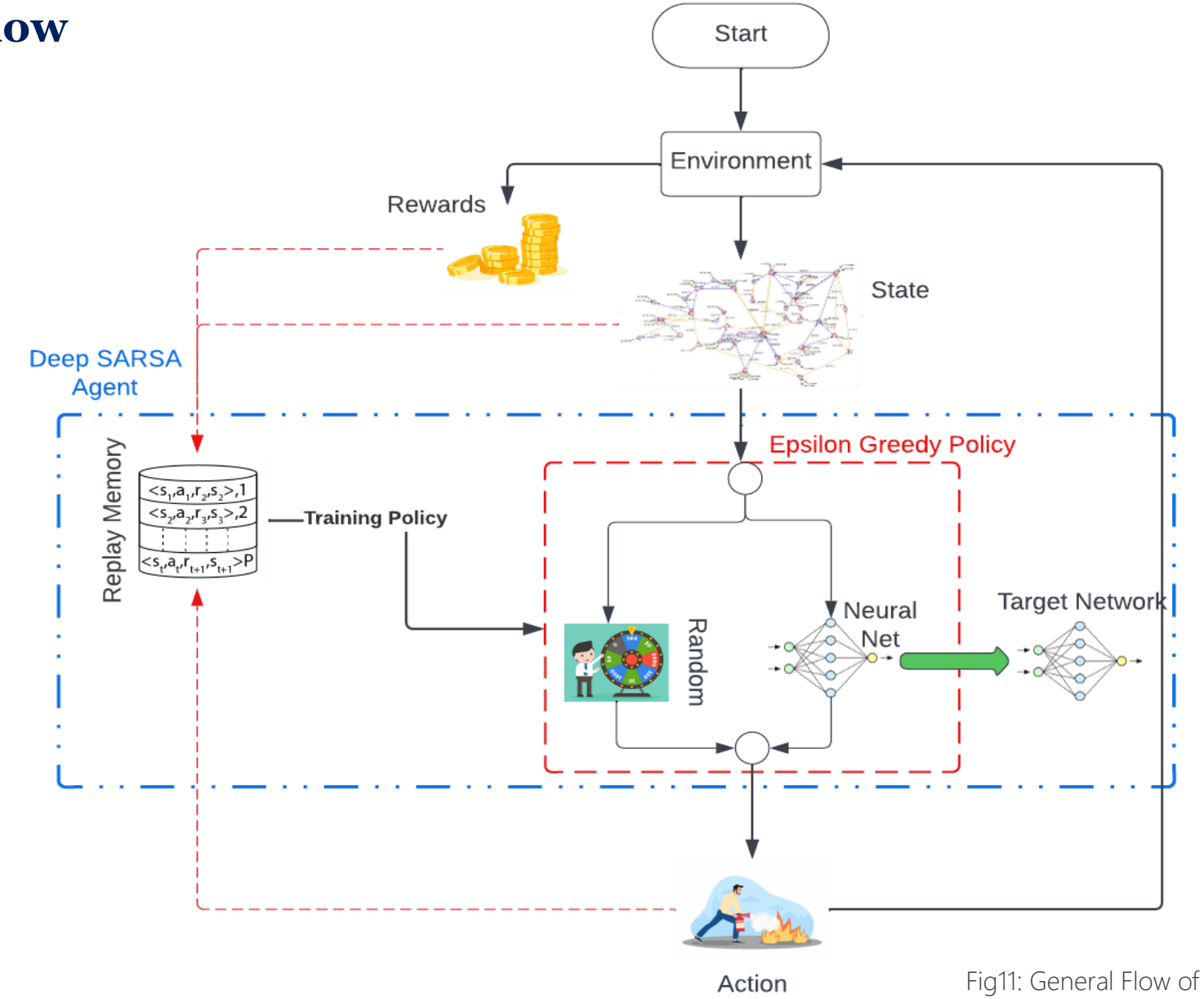


Fig11: General Flow of the Deep SARSA Agent

Agent Creation - Deep SARSA

ϵ – Greedy Policy

ϵ is a random number that states the percentage of random actions selected. In our implementation ϵ decays with increasing reward

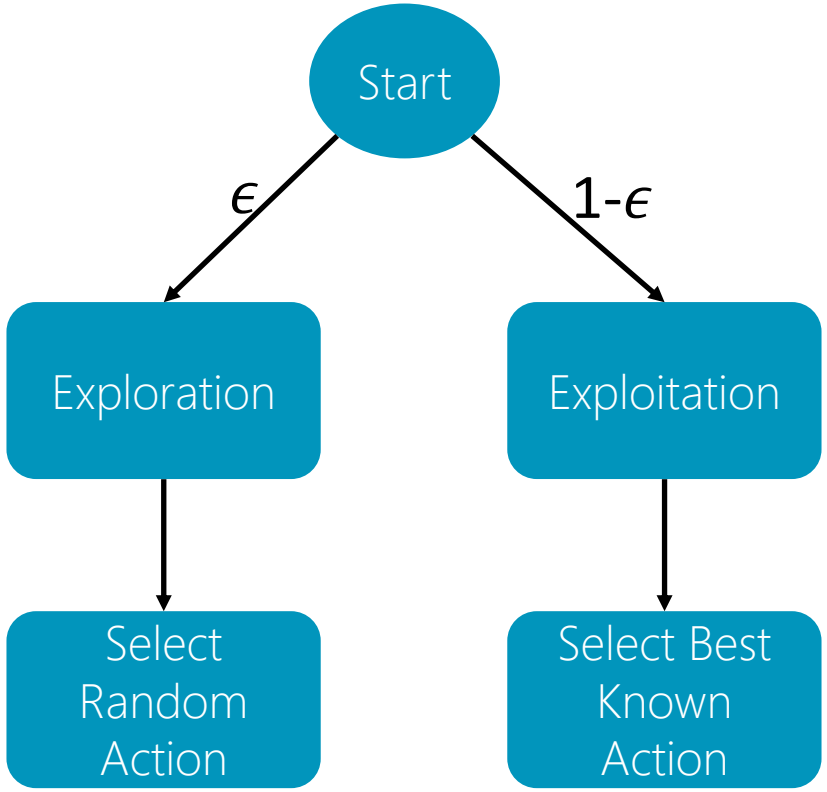


Fig12: ϵ – Greedy Policy

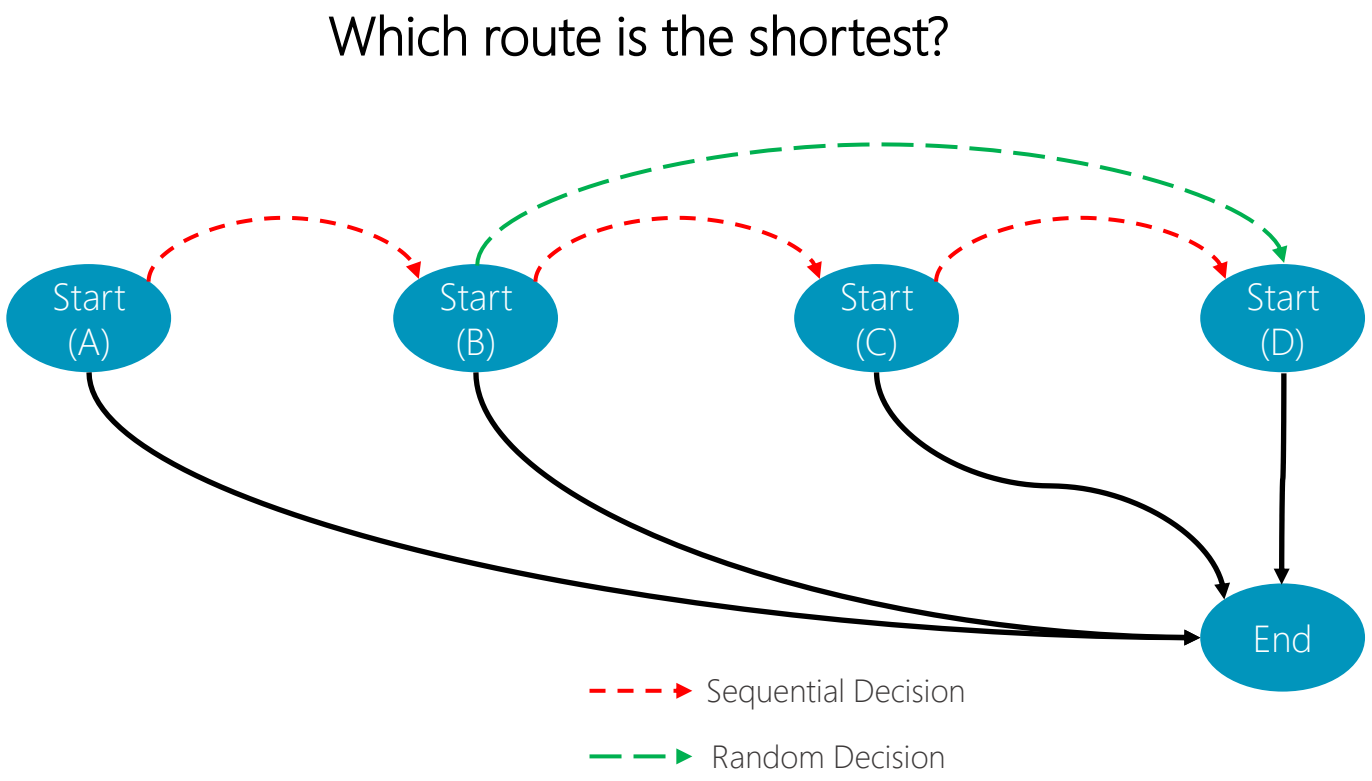


Fig13: Finding the shortest route

Agent Creation - Deep SARSA

Replay Memory

- Stores State-Action-Reward-Done-Next_State after every iteration
- Used to create batches of training data that is fed to the policy (i.e. neural network) for training
- In our implementation each batch has 64 data points

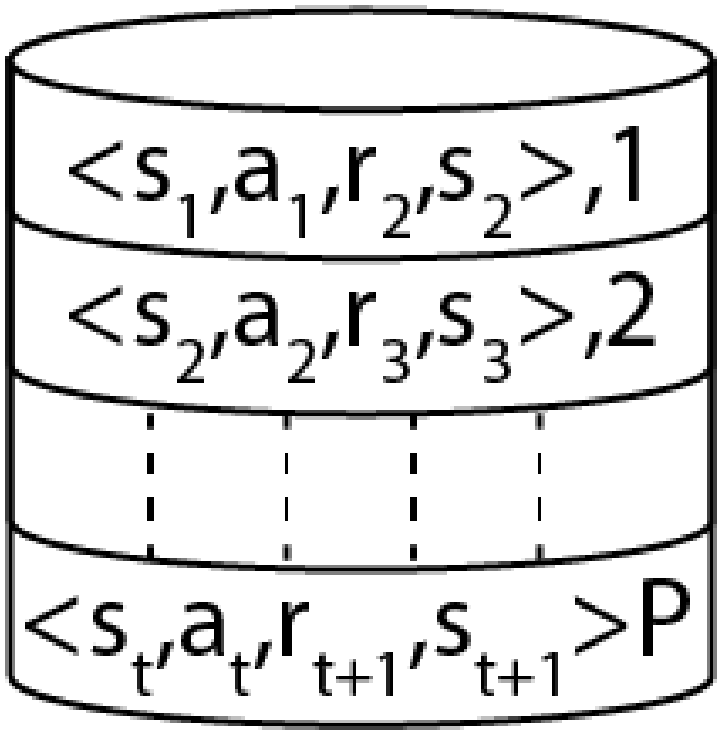


Fig14: Replay Memory

Agent Creation - Deep SARSA

Neural Network

- Goal → Maximise future rewards
- SARSA is smart way to explore the action space but not an intelligent way to improve policy
- Neural Networks provide specific functions to understand which is the next best action and how to improve policy

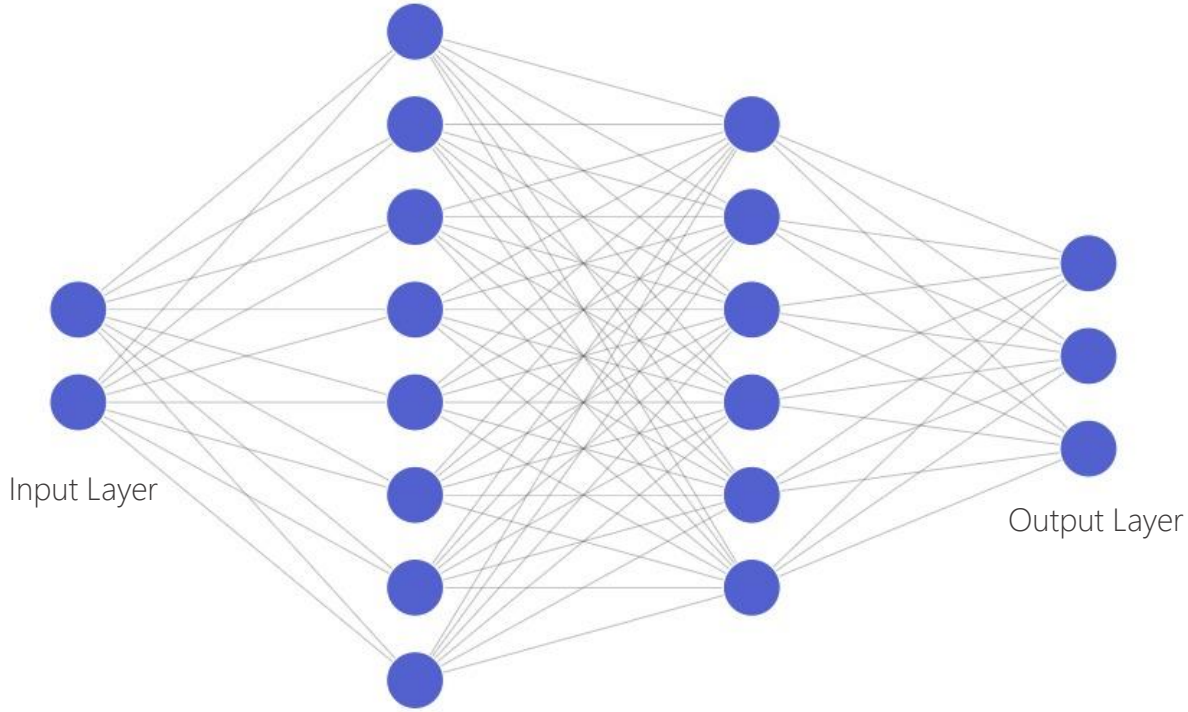


Fig15: Neural Network with 2 Hidden Layers

Agent Creation - Deep SARSA

Neural Networks – Architecture and Hyperparameters

Parameter	Approaches for initial values
Learning rate (α) speed at which your network learns	Must not be large must not be very small usually lower than 0.001
Training batch size	Problem specific
Activation functions	For multiclass classification ReLu or Softmax.
Number of layers (depth of network)	The deeper the better performing
Number of neurons (width of network)	$width = \frac{size\ of\ training\ set}{\gamma[input\ size + output\ size]}$ γ is integer between 1 and 10

Table4: Hyper parameters and their initial values

- Tuning above is a separate area of research
- Methods like heuristics has shown good results
- Other methods like Auto-ML implemented by Auto-Pytorch library provides a black box approach for tuning these parameters, but requires high computational power

Agent Creation - Deep SARSA

Target Network

- Algorithm is updating 64 (batch size) states and action values at one time
- Causes catastrophic forgetting: Network works great for some timesteps then starts taking bad actions again and restarts the learning process
- Target Network is a deep copy of the defined neural network
- Target network saves the main network at every K time steps and provides a stable learning curve
- Allows main network to try out new actions while saving the previous weights as checkpoint until it finds out next best action

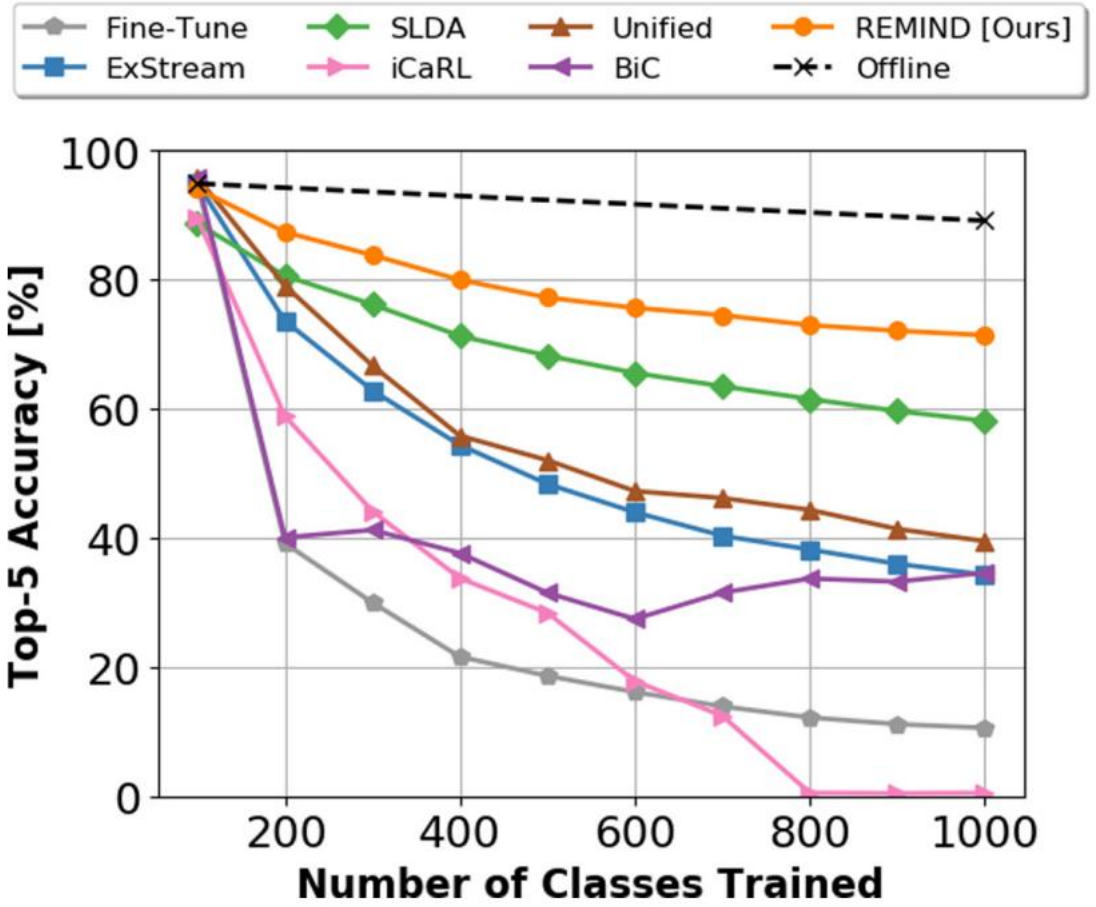
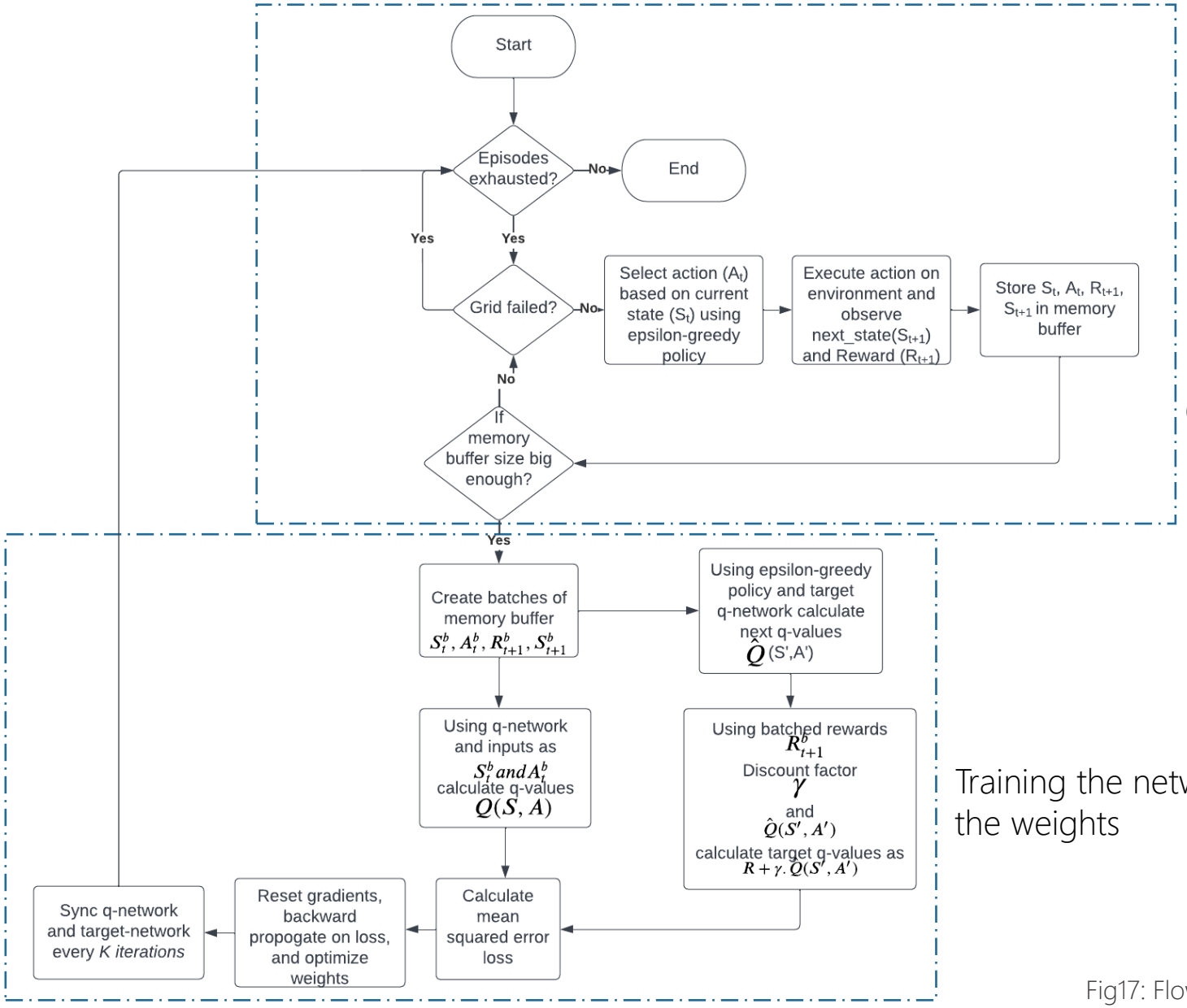


Fig16: Performance of streaming ImageNet models

Image Source (Computer Vision – ECCV 2020 pp 466–483)

Agent Creation - Deep SARSA



Creating the memory Buffer

Training the network and optimising the weights

Fig17: Flow diagram for Deep SARSA Agent

Agent Creation - Deep SARSA

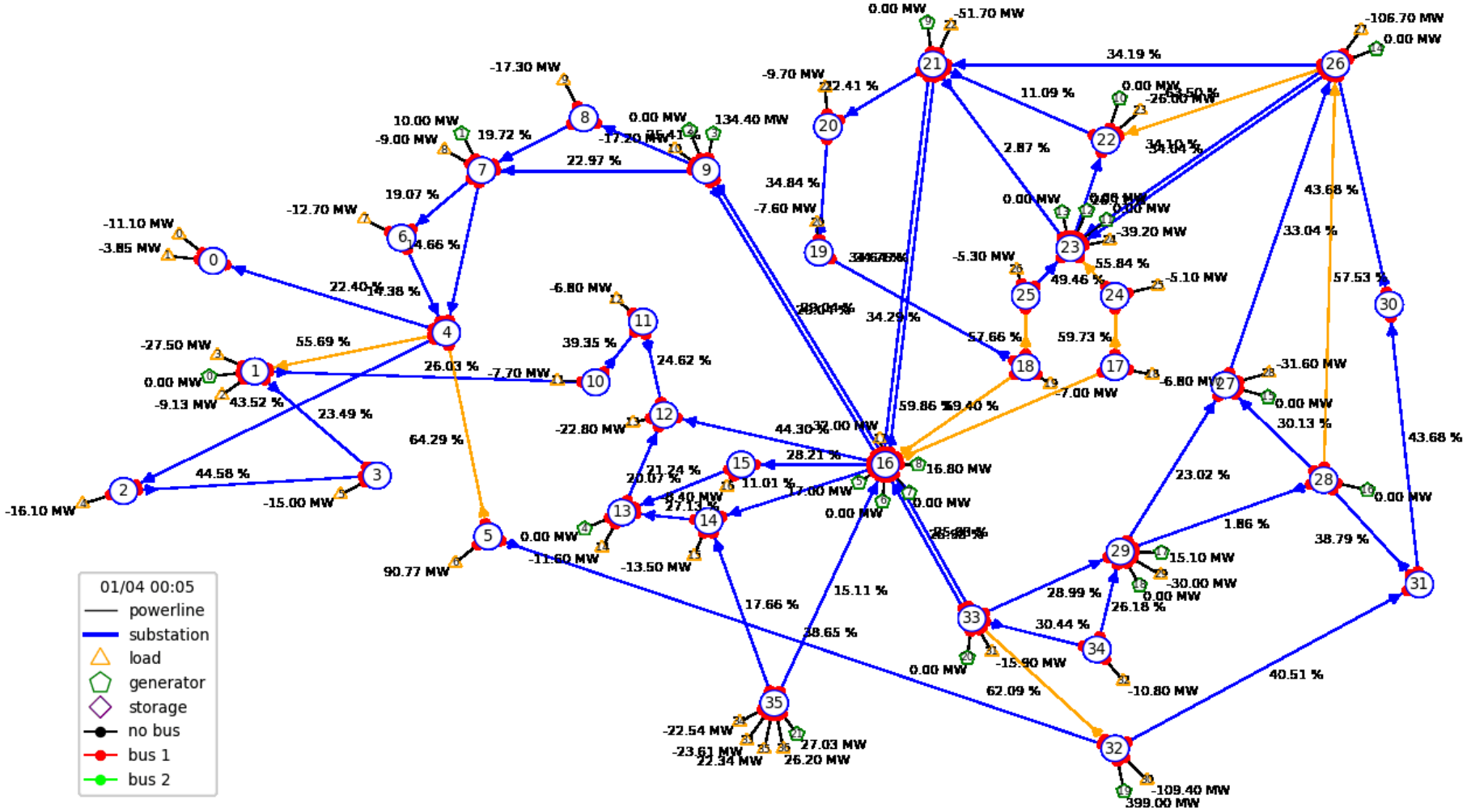


Fig18: Performance of Deep SARSA Agent on scenario

Agent Creation - Deep SARSA

Issues Faced

- Environment not compatible for working with tensors
- Last version of algorithm was giving out unacceptable outputs (not-a-number or NaNs)
- Last version was very slow to train (because of bad implementation)

Solutions

- Converted actions to numerical tensors
- Made code changes inspired by Deep-Q baseline implementation this improved training speed and fixed NaN issue

Baseline Agents

Deep Q Agent

- Implementation is almost same as Deep SARSA algorithm
- Also uses ϵ-greedy policy, memory buffer, and target network
- Only difference is that Q-learning is off policy method, that implies it uses different policy for exploration and exploitation

Advantages (off-policy methods in general):

- Off-policy methods are better at randomization
- Learns optimal policy regardless of performance of behavioral policy

Proximal Policy Optimization (PPO)

- PPO, the agent learns directly from the environment, and once it uses a batch of experiences, it discards that batch after doing a gradient update.
- The main idea of PPO is to avoid large policy update during training by computing the ratio between the old and the current policy.
- On-policy update method
- Works well in continuous action space

Advantages:

- PPO adds a soft constraint that can be optimized by a first-order optimizer.
- Good balance on the speed of the optimization.
- Experimental results prove that this kind of balance achieves the best performance with the most simplicity.

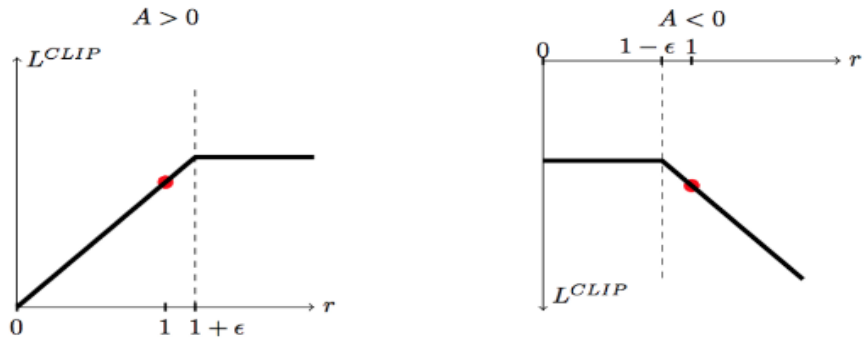


Fig19: Clipping as soft constraint

Results Summary

	Episode 1 Time steps survived	Episode 2 Time steps survived	Total Time Steps (per episode)
Switching Agent	7	7	8062
Deep SARSA Agent	432	481	8062
Topology Agent	555	1733	8062
Deep Q Agent	156	619	8062
PPO Agent	987	1918	8062

Table5: Agent Results Comparison

Visualization using Grid2Viz

Grid2Viz Introduction

Grid2Viz is a web application that offers several interactive views into the results of Reinforcement Learning agents that ran on the Grid2Op platform.

Analysis of different agents on the basis of:

- Scenarios Analysis
- Actions Taken
- Rewards (instant/cumulated reward)
- Attack Survive

Grid2Viz Scenarios

Our environment has multiple scenarios for different months, a few of them have been shown below:

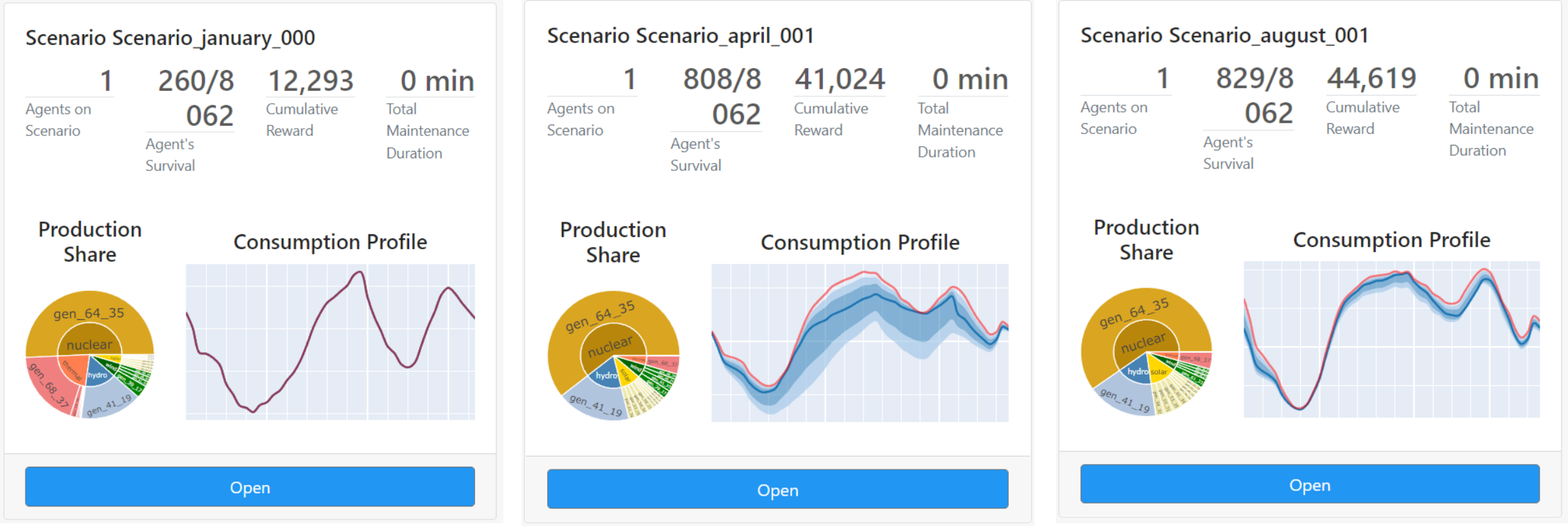


Fig20: Scenarios in our environment "l2rpn_neurips_2020_track1_small"

Grid2Viz - Analysis of SwitchingAgent

SwitchingAgent survived only 91 timesteps out of 8062, in this it faced 27 overflows and took 3 actions and ultimately lead to game over condition.

Fig21: SwitchingAgent Performance and Evaluation

Grid2Viz - Analysis of TopologyAgent

Topology agent survived all 5379 timesteps out of 8062 in one of the scenario and in this it faced 9 overflows and took 19 action.

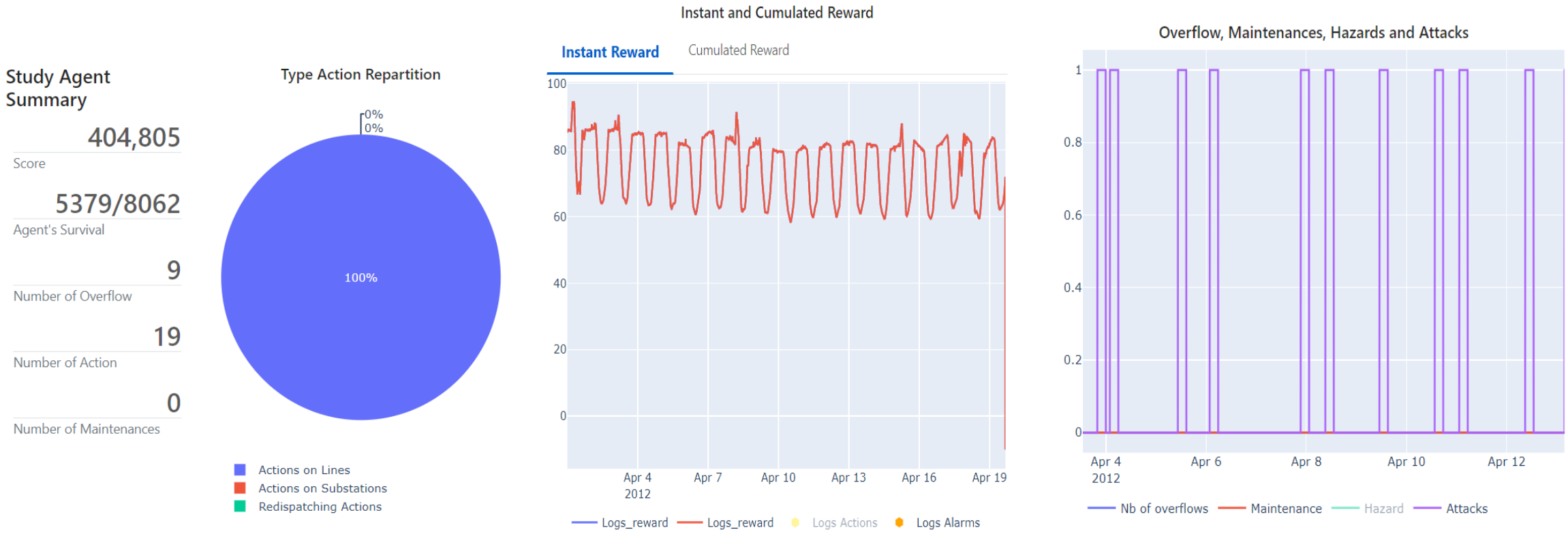


Fig22: TopologyAgent Performance and Evaluation

Grid2Viz - Analysis of Deep SARSA Agent

DeepSARSA agent survived only 511 timesteps out of 8062, in this it faced 16 overflows and took 511 actions and ultimately lead to game over condition.

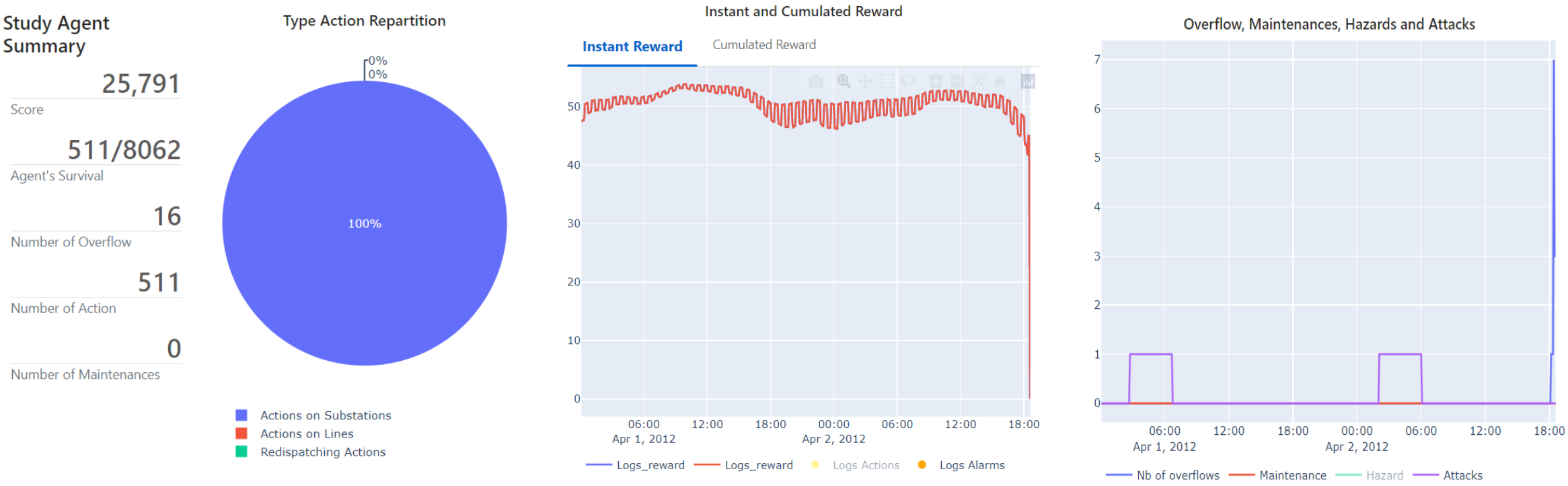


Fig23: Deep SARSA Agent Performance and Evaluation

Grid2Viz - Analysis of Deep-Q Agent

Deep-Q agent survived only 383 timesteps out of 8062, in this it faced 10 overflows and took 383 actions and ultimately lead to game over condition.

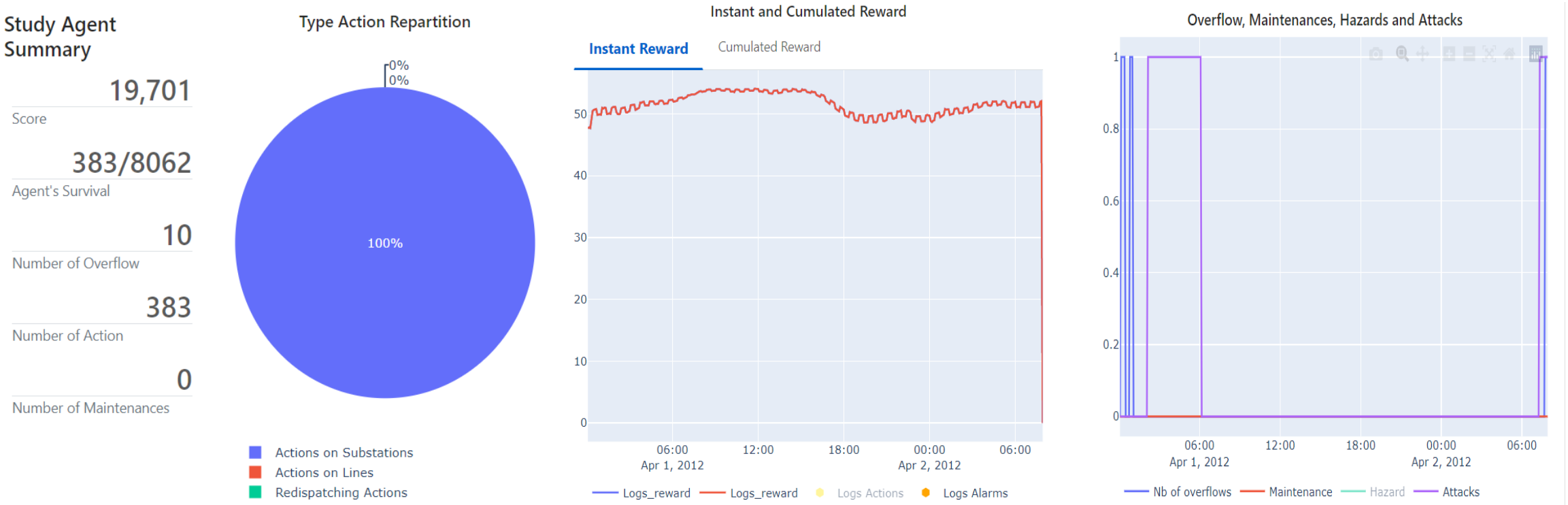
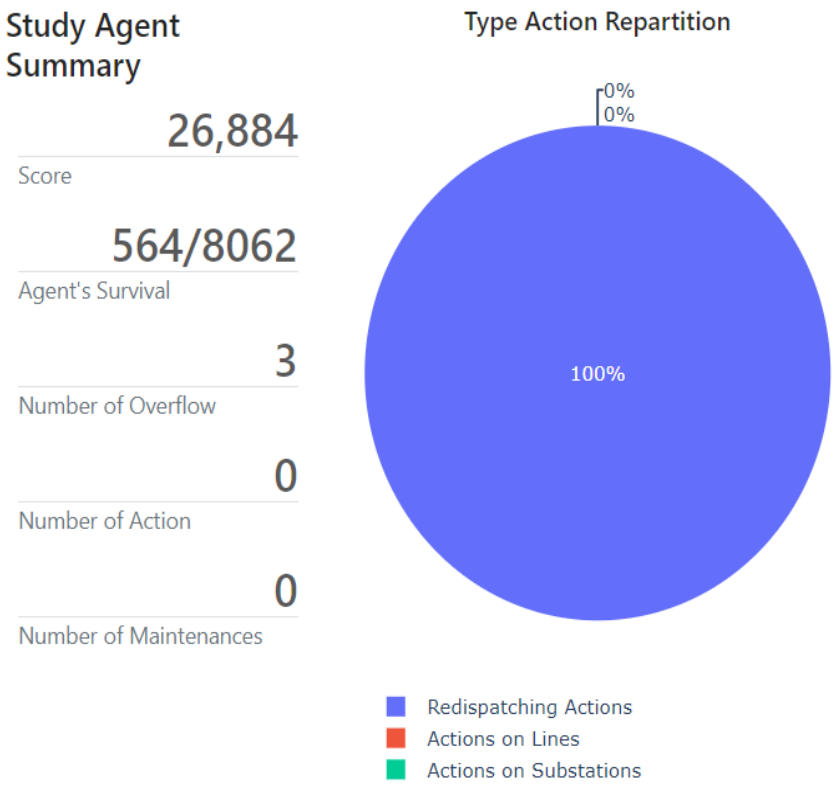


Fig24: Deep–Q Agent Performance and Evaluation

Grid2Viz - Analysis of PPO Agent

PPO agent survived only 564 timesteps out of 8062, in this it faced 3 overflows and took 0 actions and ultimately lead to game over condition.



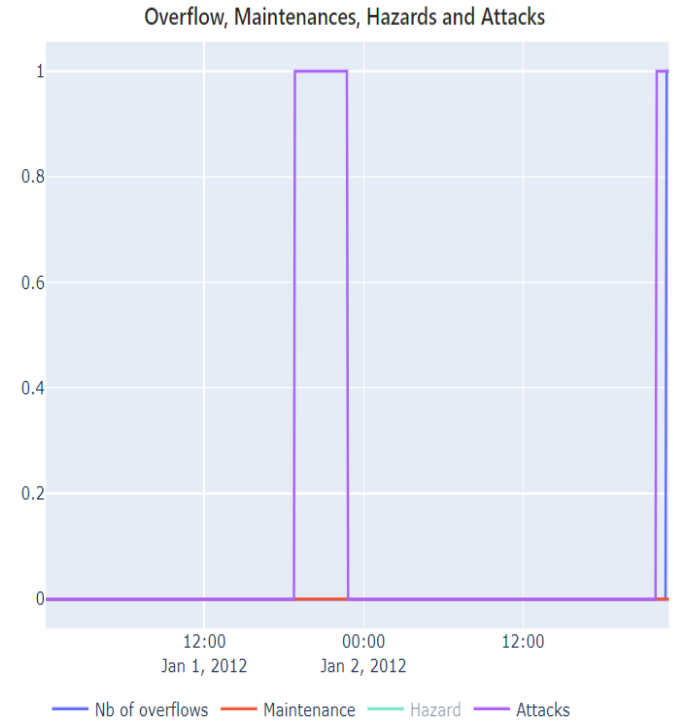


Fig25: PPO Agent Performance and Evaluation

Grid2Viz - Comparison of Agents

Fig26: Comparison of Deep SARSA vs PPO Agent

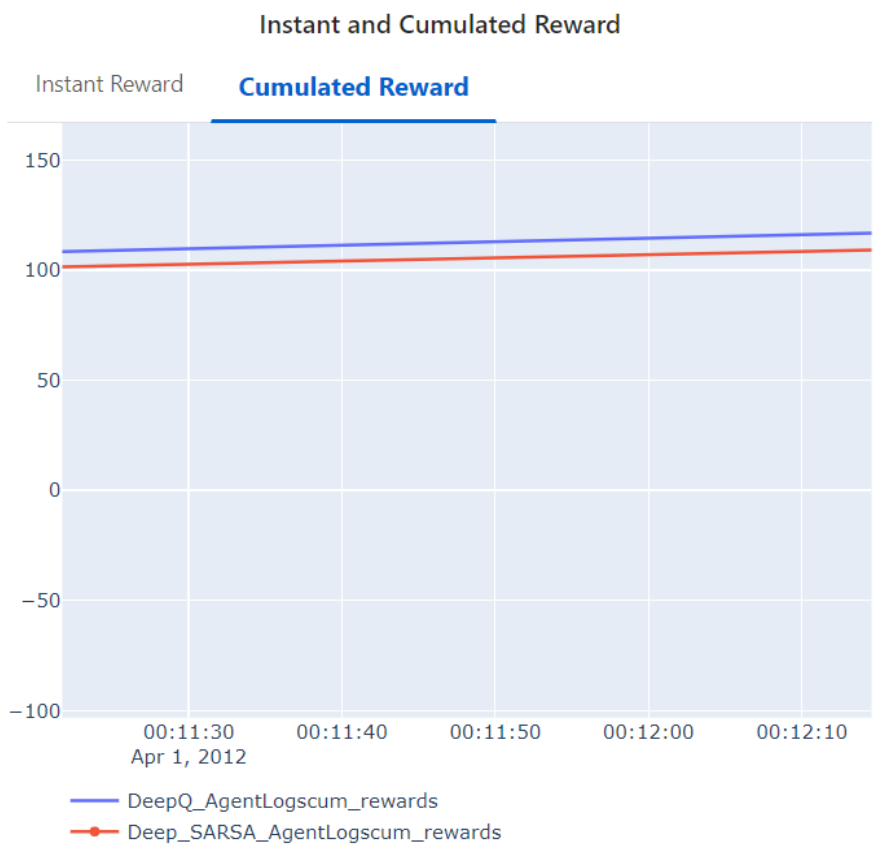


Fig27: Comparison of Deep-Q vs Deep SARSA Agent

Grid2Viz - Comparison of Agents

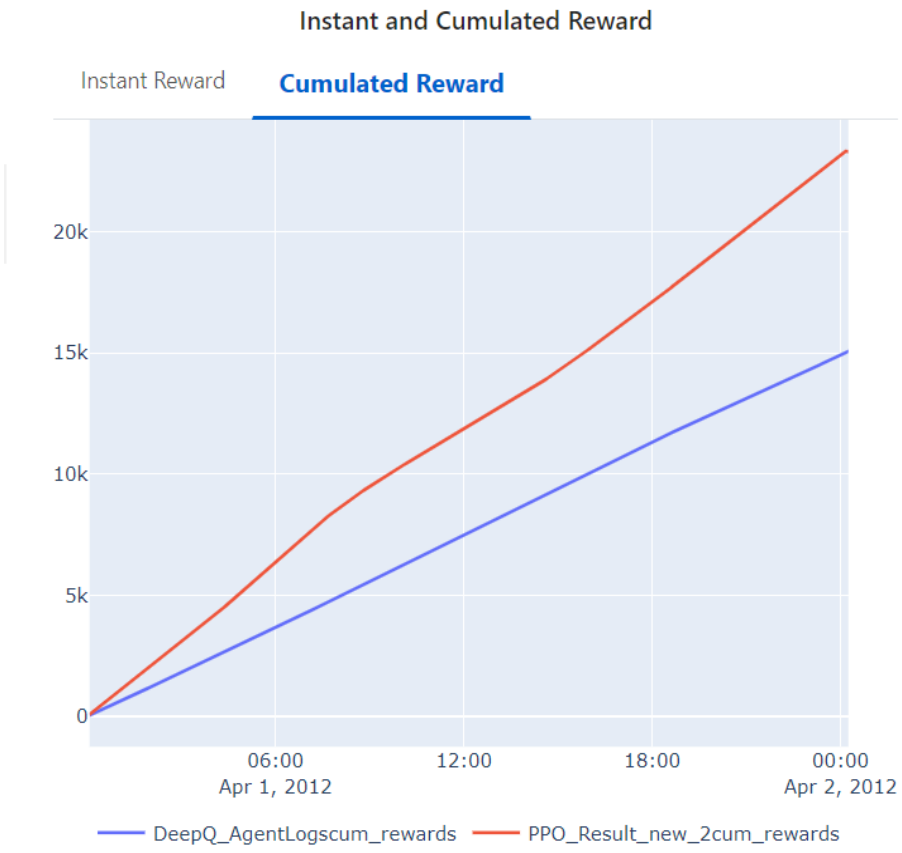


Fig28: Comparison of Deep-Q vs PPO Agent

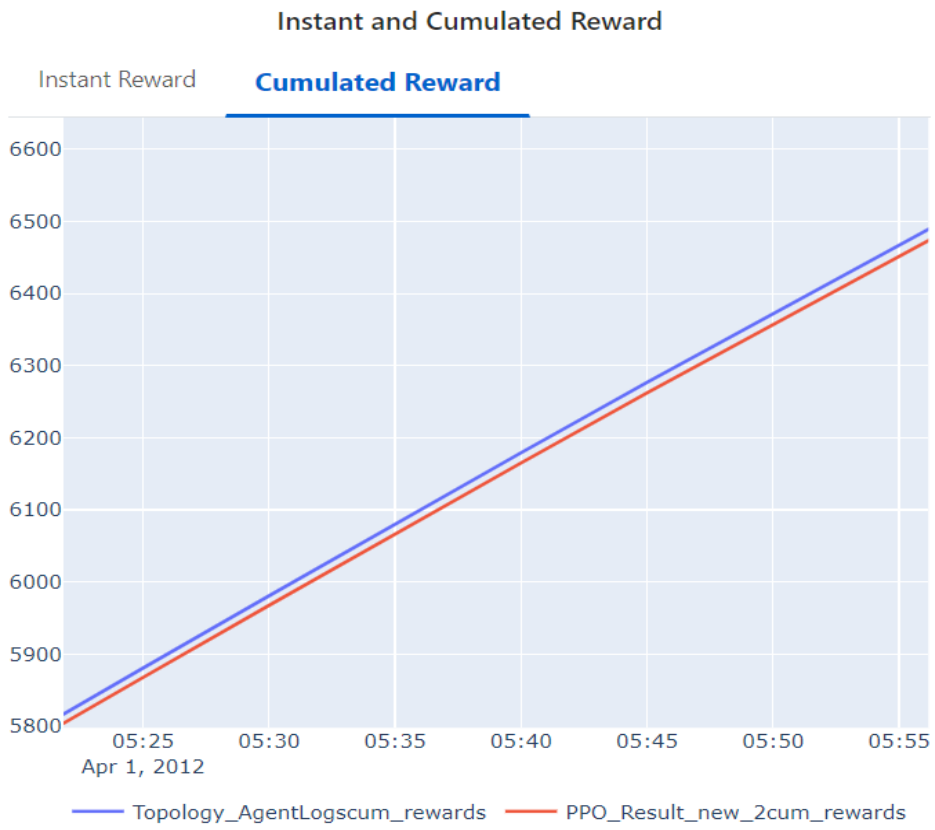


Fig29: Comparison of Topology vs PPO Agent

Grid2Viz - Comparison of Agents

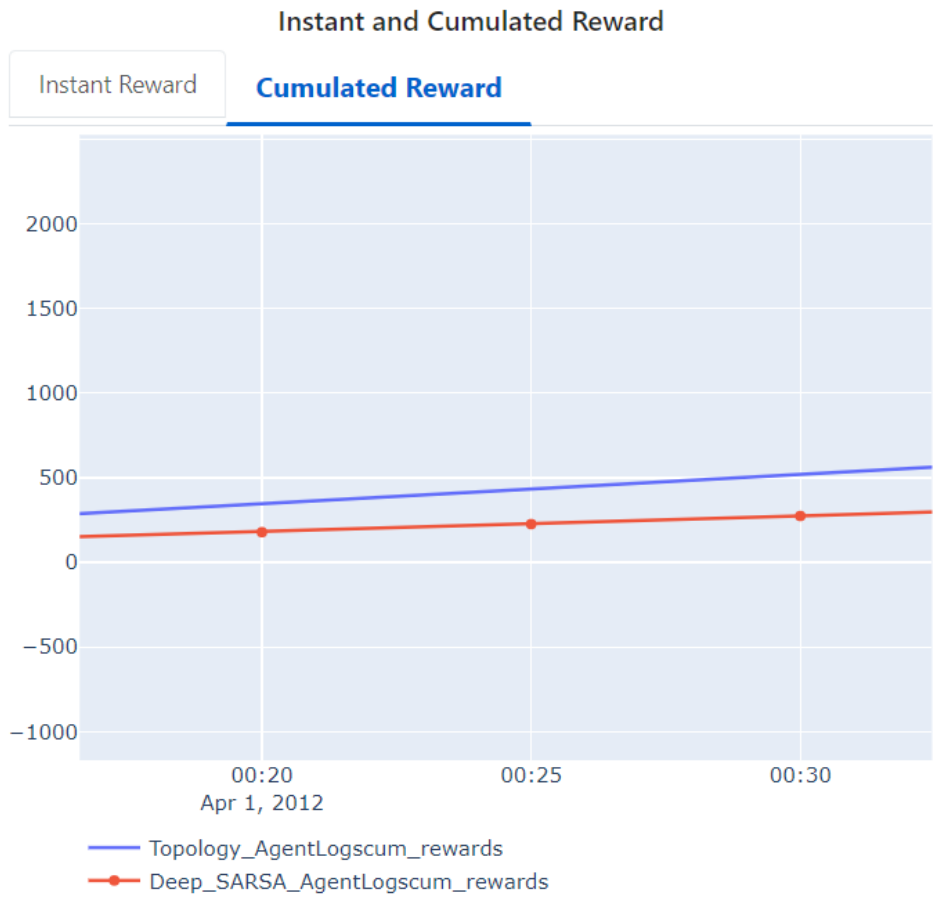


Fig30: Comparison of Topology vs Deep SARSA Agent

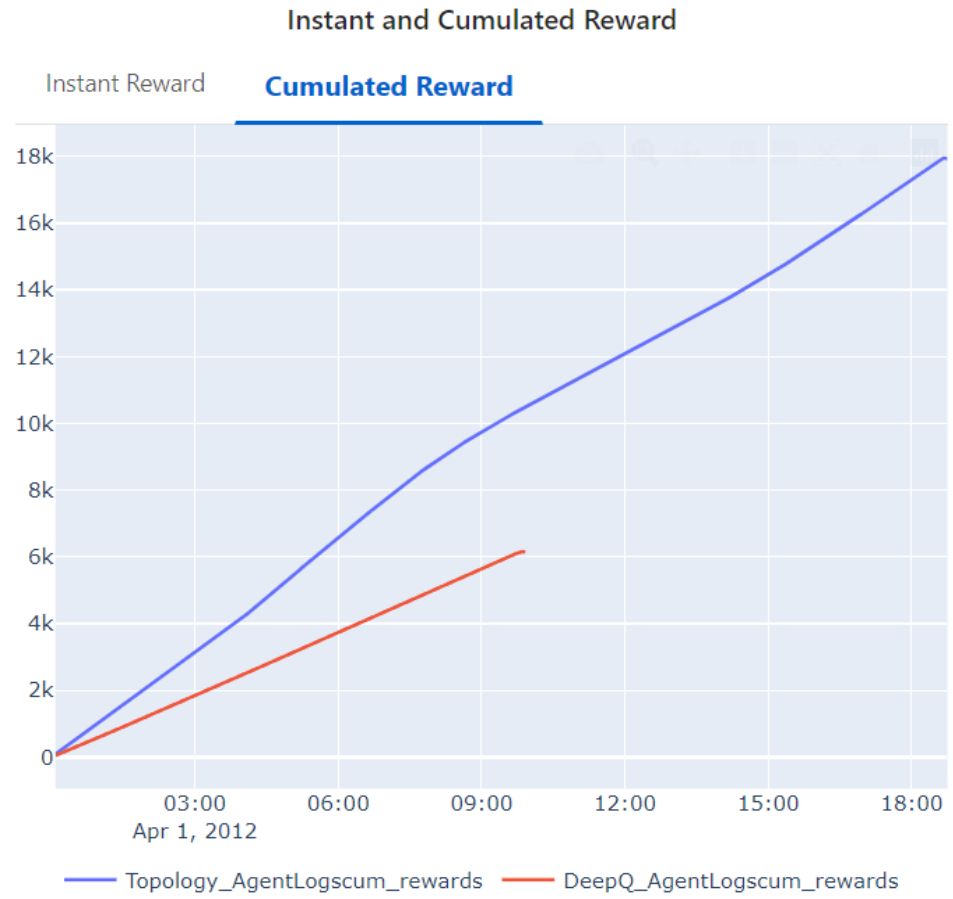


Fig31: Comparison of Topology vs Deep-Q Agent

Conclusion

- Reinforcement learning (RL) works well in complex sequential decision-making task, this shows promising results for **optimization of switching states**.
- In the **real-world application** emphasis is more how an agent can **survive scenarios** and at the same time it's **interpretability**.
- Topology agent used **filtered actions** from all possible actions to decide which action **reduced the overload** more.
- Deep SARSA showed potential to tackle the problem, it **requires more training** to learn from the scenarios to **make better decision** in case on an attack.
- PPO based agent work **better and fast** in comparison to other on-policy based agents. Further work on the implementation and using **discrete and continuous action** can improve the performance.

Important Links

- Documentation
- Custom Agents
- RL Environment
- Visualisation Tool

Reference

[1] Zhang, Zidong, Dongxia Zhang, and Robert C Qiu: Deep reinforcement learning for power system applications: An overview. CSEE Journal of Power and Energy Systems, 6(1):213–225, 2019.

[2] Zhao, Dongbin, Haitao Wang, Kun Shao, and Yuanheng Zhu: Deep reinforcement learning with experience replay based on sarsa. In 2016 IEEE symposium series on computational intelligence (SSCI), pages 1–6.IEEE, 2016.

[3] Kelly, Adrian, Aidan O'Sullivan, Patrick de Mars, and Antoine Marot: Reinforcement learning for electricity network operation. arXiv preprint arXiv:2003.07339, 2020.

[4] Marot, Antoine, Benjamin Donnot, Gabriel Dulac-Arnold, Adrian Kelly, Aidan O'Sullivan, Jan Viebahn, Mariette Awad, Isabelle Guyon, Patrick Panciatici, and Camilo Romero: Learning to run a power network challenge: a retrospective analysis. In NeurIPS 2020 Competition and Demonstration Track, pages 112–132. PMLR, 2021.

[5] Yoon, Deunsol, Sunghoon Hong, Byung Jun Lee, and Kee Eung Kim: Winning the l2rpn challenge: Power grid management via semi-markov afterstate actor-critic. In International Conference on Learning Representations, 2020.

[6] Huawei Technologies, 2020 EI Innovation Lab: https://github.com/AsprinChina/L2RPN_NIPS_2020_a_PPO_Solution.

[7] Nematzadeh, S., Kiani, F., Torkamanian-Afshar, M., & Aydin, N. (2022). Tuning hyperparameters of machine learning algorithms and deep neural networks using metaheuristics: A bioinformatics study on biomedical and biological cases. Computational Biology and Chemistry, 97, 107619. https://doi.org/10.1016/j.compbiolchem.2021.107619

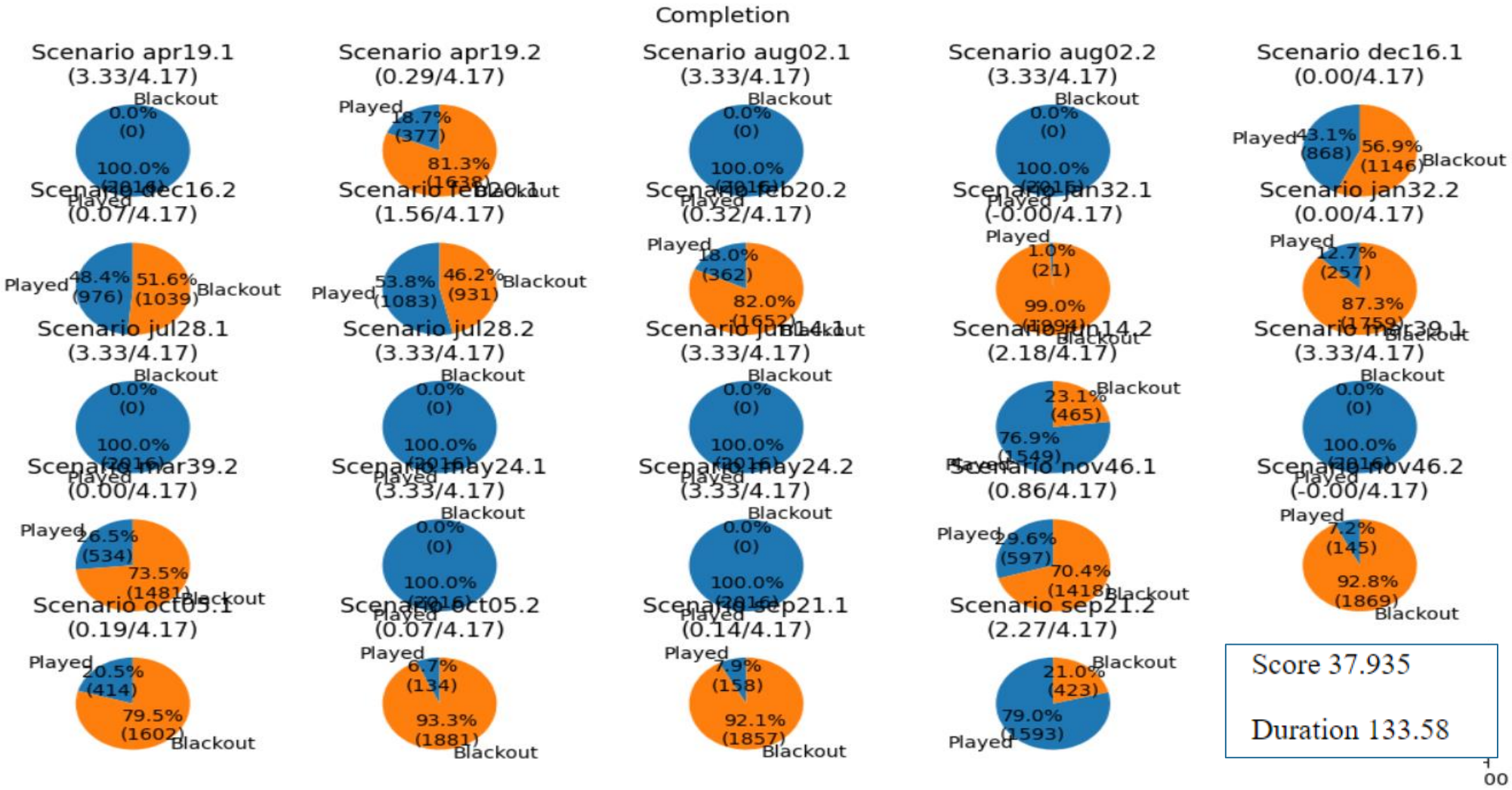
Reference

[8] Hayes, T.L., Kafle, K., Shrestha, R., Acharya, M., Kanan, C. (2020). REMIND Your Neural Network to Prevent Catastrophic Forgetting. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, JM. (eds) Computer Vision – ECCV 2020. ECCV 2020. Lecture Notes in Computer Science(), vol 12353. Springer, Cham. https://doi.org/10.1007/978-3-030-58598-3_28

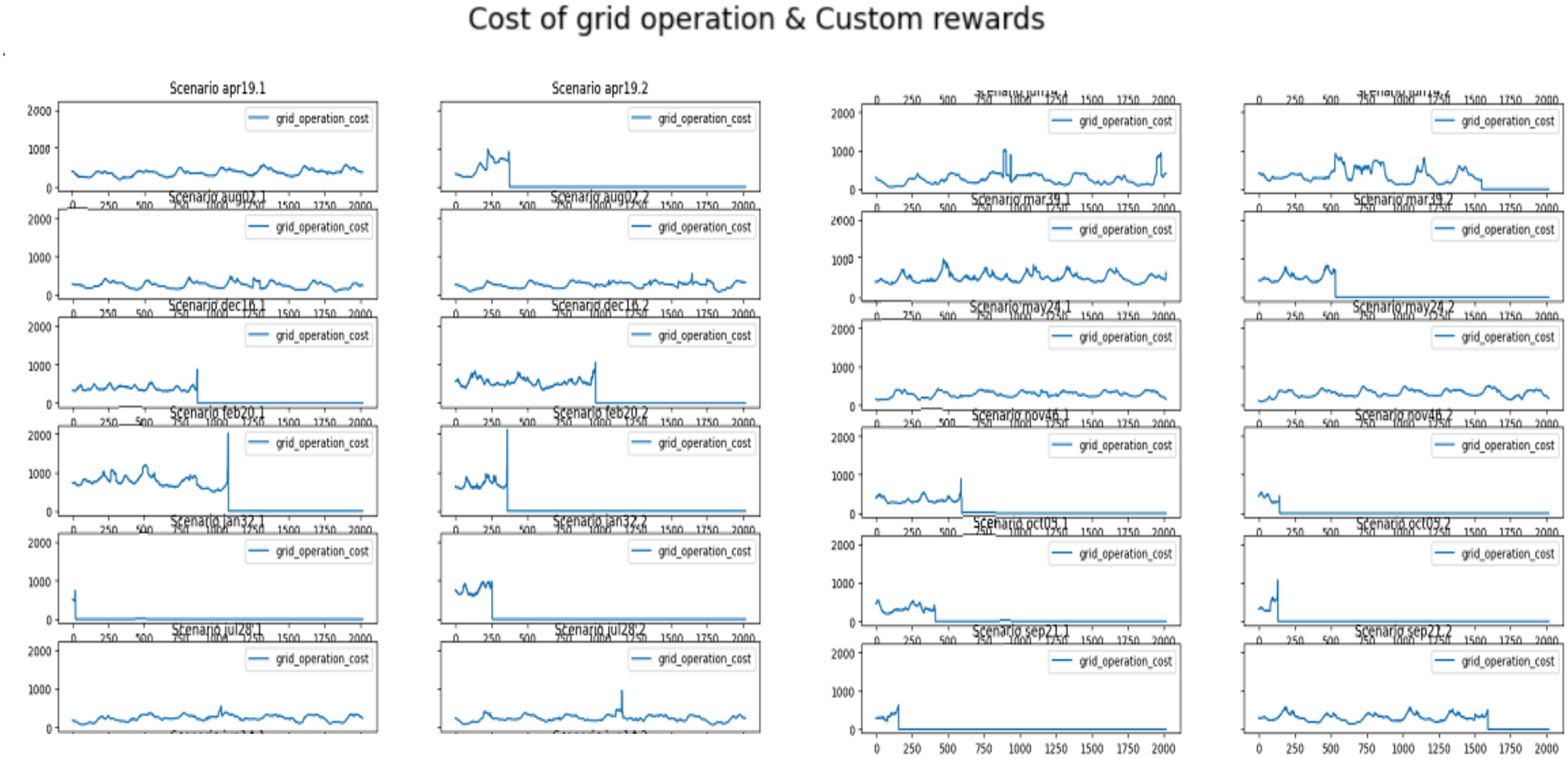
Optimisation of Switching States

using Reinforcement Learning

Appendix: A (Submission Score on robustness)



Appendix: A (Submission Score on robustness)



Agent Creation - TopologyAgent

Algorithm 1: topology agent

Input : environment, filter_actions
Output: chosen_action

```
Set threshold
recovery_stack = []
if if obs.rho.max ≤ threshold then
    if recovery_stack = [] or is_legal = False then
        a = do_nothing
    else
        if recovery_stack[0] decreases rho then
            a = action
        else
            a = do_nothing
        end
    end
    return action = best_line_to_reconnect
end

for action in filter_actions do
    update choose action which decrease rho maximum
    update recovery_stack
end
return action = best_line_to_reconnect
```

Agent Creation - Deep SARSA

Algorithm 1 Deep SARSA

Input: α learning rate, ϵ random action probability,
    γ discount factor,
Initialize q-value parameters θ and target parameters $\theta_{targ} \leftarrow \theta$
$\pi \leftarrow \epsilon$-greedy policy w.r.t $\hat{q}(s, a|\theta)$
Initialize replay buffer B
for episode $\in 1..N$ **do**
    Restart environment and observe the initial state S_0
    for $t \in 0..T-1$ **do**
        Select action $A_t \sim \pi(S_t)$
        Execute action A_t and observe S_{t+1}, R_{t+1}
        Insert transition $(S_t, A_t, R_{t+1}, S_{t+1})$ into the buffer B
        $K = (S, A, R, S') \sim B$
        Select actions $A' \sim \pi(S')$
        Compute loss function over the batch of experiences:

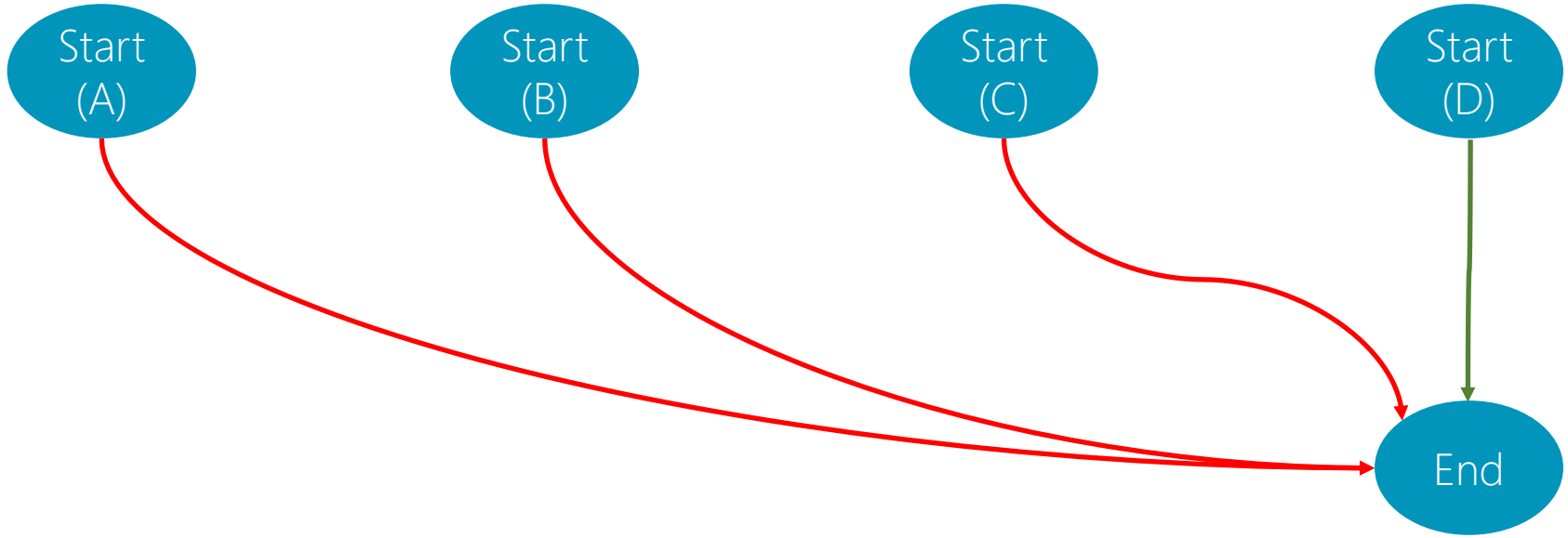
$$L(\theta) = \frac{1}{|K|} \sum_{i=1}^{|K|} \left[R_i + \gamma \hat{q}(S'_i, A'_i|\theta_{targ}) - \hat{q}(S_i, A_i|\theta)\right]^2$$

    end for
    Every k episodes synchronize $\theta_{targ} \leftarrow \theta$
end for
Output: Near optimal policy π and q-value approximations $\hat{q}(s, a|\theta)$

Agent Creation - Deep SARSA

ϵ – Greedy Policy: Why randomisation?

- An algorithm might start from A and one by one exploring every alternative will reach to D
- Randomisation will allow to jump directly from A to D in an optimistic scenario
- Randomisation allows to explore the action space faster and may be able to find better solutions faster
- Decaying randomness will allow to stick to trial and tested actions after some good experiences



Agent Creation - Deep SARSA

Neural Networks – Architecture Selection

- No specific rules for selecting architecture
- Hit and trial is the best approach (lot of libraries available for that)
- General rules of thumb to begin with:
  - Keeping number neurons constant increasing the layers usually improves performance
  - For feed forward neural network number of neurons in a layer can be

$$width = \frac{n}{\gamma[N_1+N_2]}$$

    - Where, n is size of training set
    - $N1$ is size of input layer
    - $N2$ is size of output layer
    - γ is an integer parameter between 1 and 10

Agent Creation - Deep SARSA

Q-Learning and SARSA Learning

Considering a Markov decision process (MDP), the goal of learning task is to maximize the future reward when the agent interacts with environment.

$$R(t) = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$$

Where $y \in \{0,1\}$ is the discount factor, r_t is the reward when an action is taken at time t and T is often regarded as the time when the process terminates.

$$Q^{\pi}(s,a) = E_{\pi}\{R(t) \mid s_t = s, a_t = a\}$$
$$= E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\}$$

Where $E_{\pi}\{ R(t) \mid st = s, a_t = a \}$ is the expected return, and π is the policy function over actions.

Agent Creation - Deep SARSA

Q-Learning and SARSA Learning

Now the learning task aims at obtaining the optimal state-action function $Q^*(s',a')$.

- Q-Learning is an off-policy method.
- The agent selects the action a.
- Receives reward r and the next state s' is derived.
- $Q(s,a)$ represents the current state action value.
- To update the current state action value function, next state-action value is used.
- Next state s' is available, but the next action a' is unknown.
- So, in Q-Learning it is necessary to take a greedy action to maximize the next $Q(s',a')$

The update equation is

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Agent Creation - Deep SARSA

SARSA Learning

- By contrast, SARSA learning is an on-policy method.
- It means when updating the current state-action value, the next action a' will be taken.
- But in Q-Learning, the action a' is completely greedy.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$$

- The results for the Deep SARSA agent:

```
The results for the Deep-SARSA agent are:
        For chronics with id Scenario_august_dummly
                - cumulative reward: 1845.074951
                - number of time steps completed: 73 / 200
        For chronics with id Scenario_february_dummy
                - cumulative reward: 632.909546
                - number of time steps completed: 36 / 200
```

Outlook on Project Progress

So far, we have scrutinized the literature and adopted appropriate methodologies to solve the problem.
In the final stage, we will attempt to achieve a relatively better result.

01

- Understand the underlying problems
- Literature review and models adaptation
- Proposed methodologies

02

- Implementation of selected methodologies
- Create the Agent
- Achieve preliminary result
- Test on different scenarios

03

- Explore deep RL algorithms with topology agent framework
- Experiment on multiple environments
- Tuning agent and comparing score for robustness

Agent Creation - Deep SARSA

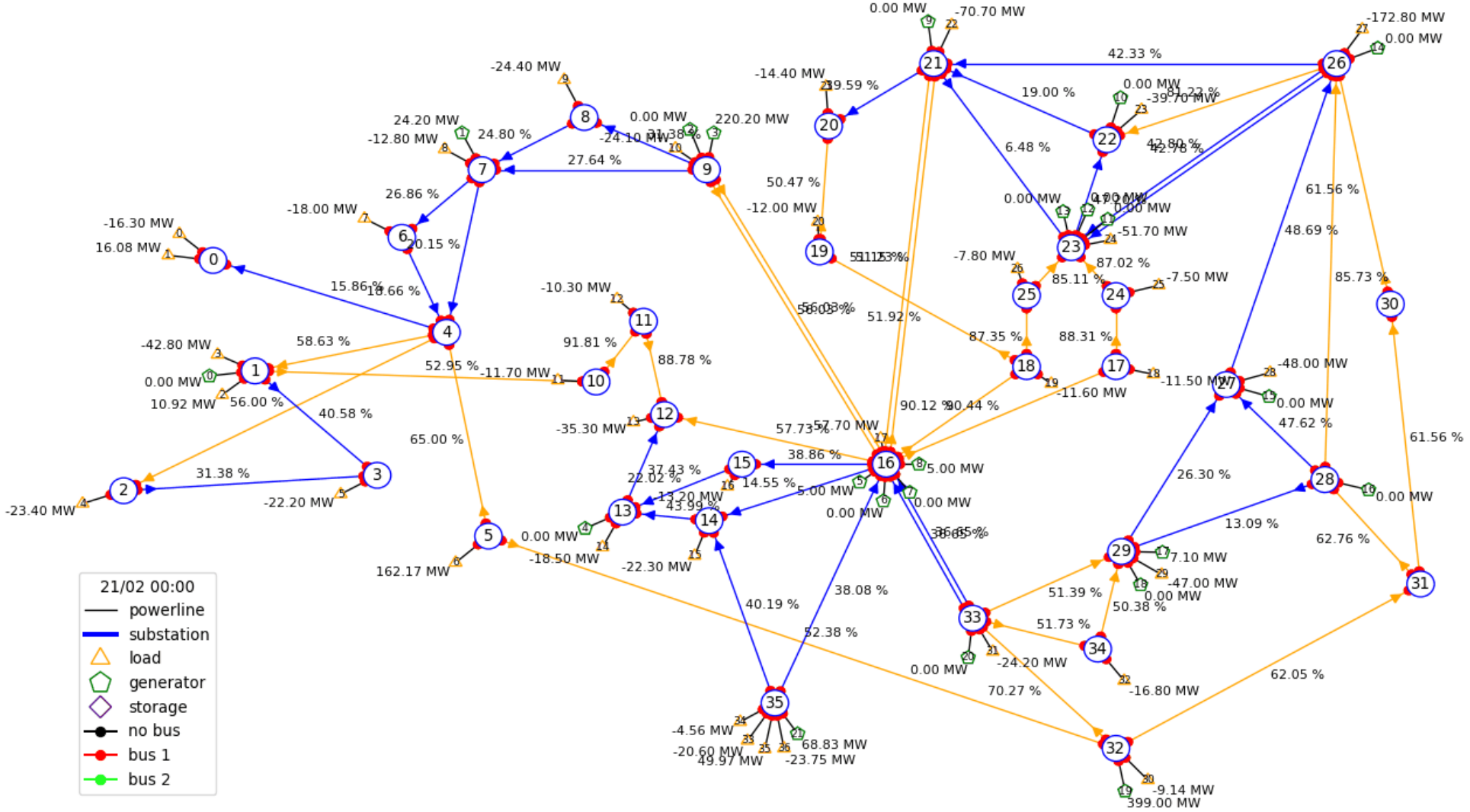


Fig10: Performance of Deep SARSA Agent on scenario 1 (August)

Agent Creation - Deep SARSA

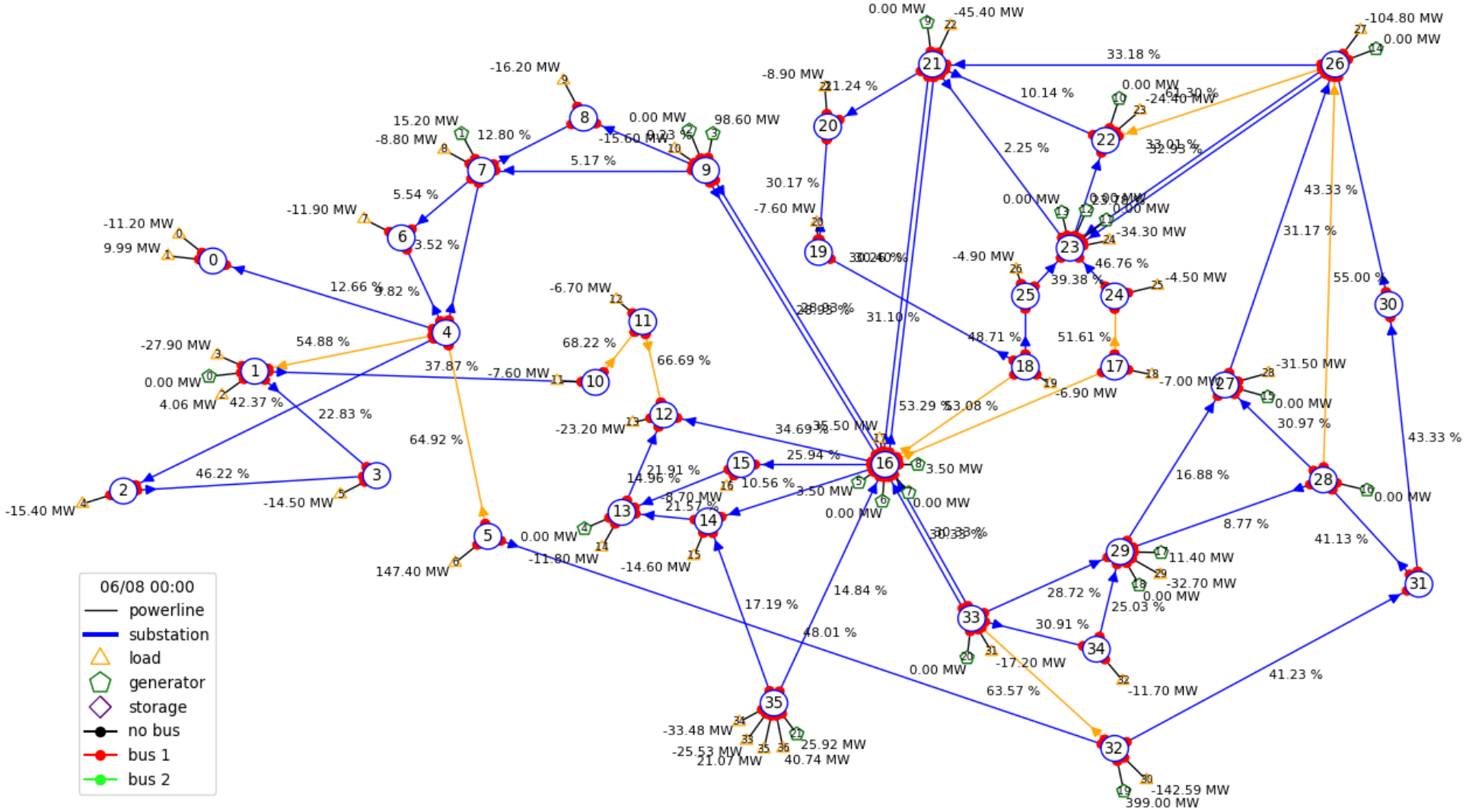


Fig11: Performance of Deep SARSA Agent on scenario 2 (February)

Agent Creation - TopologyAgent

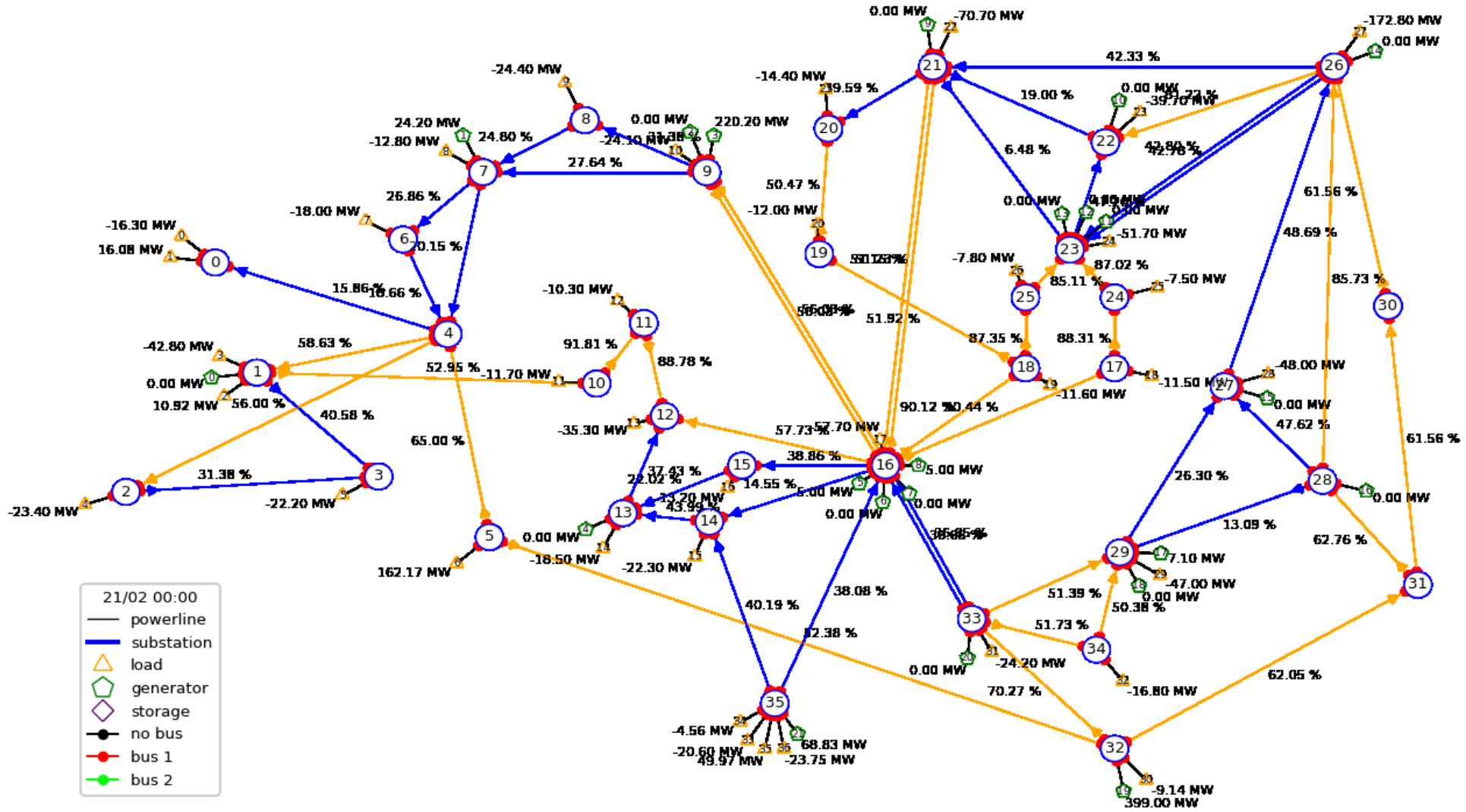


Fig7: Performance of TopologyAgent on scenario 2 (February)

Underlying Problem

Sudden demands or excesses of energy cause brief changes in voltage that can cause an overload.

Loading of electrical transmission grid increases

Renewable generation is a lot more fluctuating than the conventional, this causes new bottlenecks which leads to strains in grid.

Higher share of renewable based decentral power plants

European electricity trading enables cross-border balancing of fluctuating generation from renewable energies, if the grid capacity is insufficient, the congestion resulting from the market outcome must be eliminated through congestion management measures.

Coupling of the European electricity markets

Hurdles Faced

Hurdles faced while creating Deep SARSA Agent:

- Grid2op environment is not compatible with gym environment hence had to convert environment to gym compatible environment and hence conversion of observation space and action space was necessary
- The evaluation of agent created some problems since the Grid2viz needed data and logs in some particular format and for a particular size which is not documented properly hence little hit and trial approach was needed for the evaluation part and it consumed lot of time
- The algorithm needs creation of q-network and target network which was giving out "Not a number" (NaN) output on one of the environment, but on other environments its working fine (but need to deep dive on this issue)
- The network is working slow and has a very high run time (max_iterations * number_of_episodes) hence optimisation of the algorithm is necessary

Conclusion

Reinforcement learning (RL) works well in complex sequential decision-making task, this shows promising results for optimisation of switching states.

Grid2Op provides various environment with different scenarios to evaluate the performance of an agent. In the real-world application emphasis is more how an agent can survive scenarios and at the same time it's interpretability.

Huge amount of topology actions (2^N) makes this complex task more difficult. Our approach in topology agent focused on generating all possible actions which we can take and filter out those which reduced the overload. Then provide this to our agent which refer these previous actions on grid to make decisions.

Deep SARSA showed potential to tackle the problem, it requires more training to learn from the scenarios to make better decision in case on an attack.

PPO based agent work better and fast in comparison to other on-policy based agents. Further work on the implementation and using discrete and continuous action can improve the performance.