AI fundamental blocks - final release

Deliverable D2.4

Distribution level: Public

Version: 1.0

Date of delivery: 28/03/2026

AI4REALNET has received funding from European Union's Horizon Europe Research and Innovation programme under the Grant Agreement No 101119527, and from the Swiss State Secretariat for Education, Research and Innovation.

Graph Neural Solver: Power Grid

IRT SystemX

Outline

- Context
- Methodology
- Original Contribution
- Overview of code structure
- Experiments

Context

Definition

Graph Neural Solver is a machine learning algorithm assisted (informed) by physics knowledge for compliance to physical constraints imposed in a Power Grid

Motivation

Each industrial domain has its set of constraints that should be respected in addition to classical machine learning evaluation criteria, and this implementation allows to integrate these constraints in the loss function of the Neural Networks

Use cases

This implementation is specific to Power Grid domain and enables the prediction of active powers from the injections in the substations (nodes of the graph). It could motivate the future implementations for decision-making in the context of RL and specifically the following power grid usecase: “Power Grid Assistant”

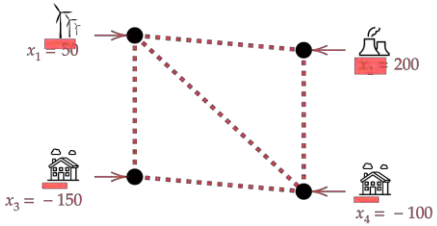
Context

- Currently used physical simulators
  - Inputs / outputs

X = injections
(productions + loads)
T = topology

Numeric Solver
(physics model)

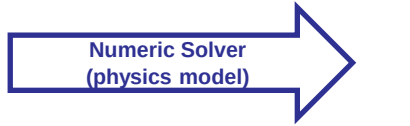
Y = power flows



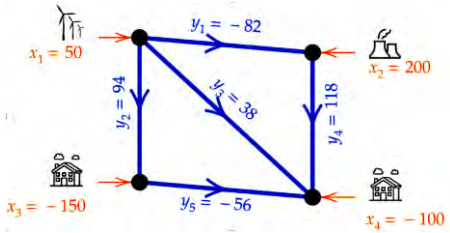
Context

- Currently used physical simulators
  - Inputs / outputs

X = injections
(productions + loads)
T = topology



Y = power flows



  - Characteristics
    - Relies on **physics equations (Kirchhoff law),** resolved by iterative optimization (Newton-Raphson)
    - Able to predict in a normal condition or different grid conditions

Power Grid equations

$$\begin{cases} 0 = -p_k + \sum_{m=1}^{K} |v_k||v_m|(g_{k,m}\cdot\cos(\theta_k-\theta_m) + b_{k,m}\sin(\theta_k-\theta_m)) & \text{Active power;} \\ 0 = -q_k + \sum_{m=1}^{K} |v_k||v_m|(g_{k,s}\cdot\sin(\theta_k-\theta_m) - b_{k,m}\cos(\theta_k-\theta_m)) & \text{Reactive power} \end{cases}$$

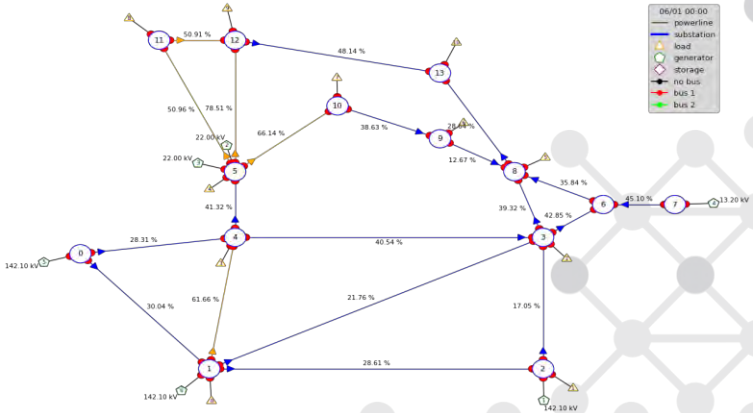
Context

- Need → Simulations allowing to predict the grid state in real time with respect to
  - Action (topological changes)
  - Exogenous factors (wind, temperature, anomalies, etc.)
- Problem → Physics simulation are computationally intensive
  - Need to accelerate the simulation [x100 à x1000]
- Solutions → Approximation of physical simulations using machine learning
  - The use of Physics Informed Neural Networks to ensure physics compliance
- Objective → Trade-off between computation time and model's quality

Problem setup

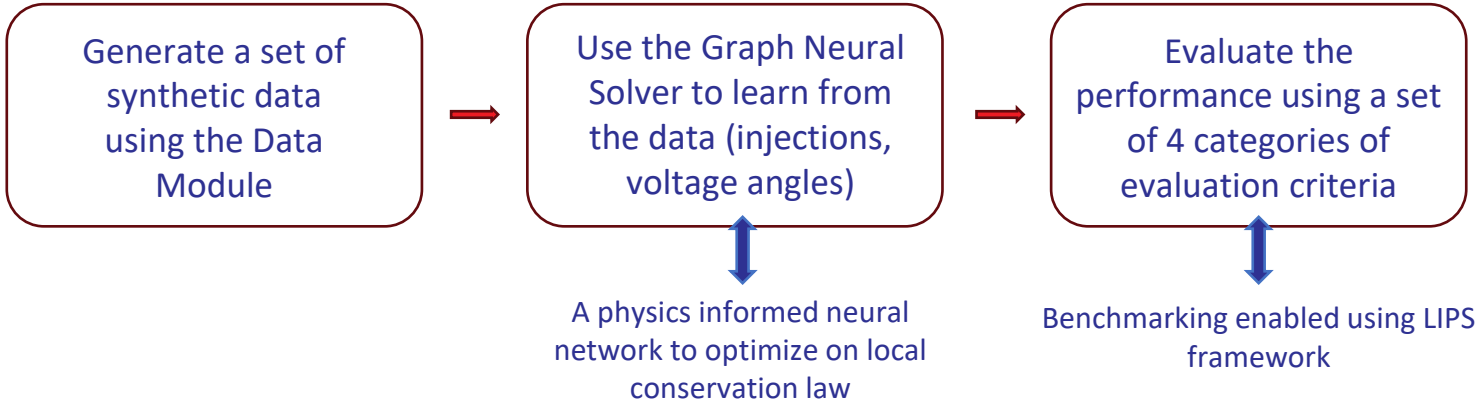
- Physical simulator → DC approximation
- Scenario → Risk identification (allowing the disconnection of power lines and topology reconfiguration)
- Contribution → Predict the active powers at power lines using a Hybrid Machine-learning based model exploiting physical constraints (local conservation law)
- Evaluation pipeline → Learning Industrial Physical Simulation (LIPS) framework with four categories of evaluation criteria

IEEE14 Power Grid



Methodology

- **Main idea** = Consider a physical constraint as the objective function of Neural Networks (GNN) that should be optimized during the training and used for the estimation of the parameters

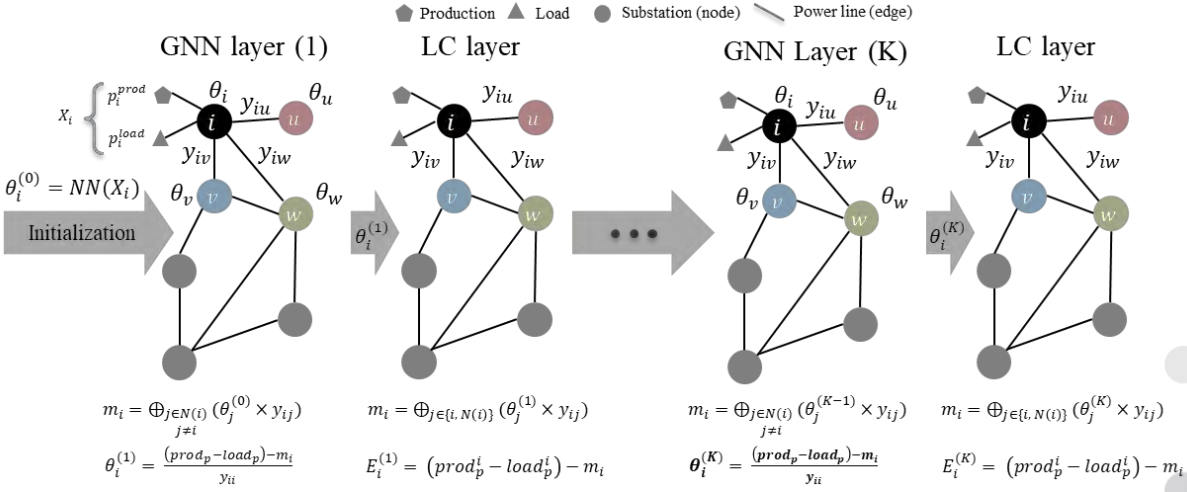


- **Preliminary experiments** on a toy usecase which could inspire the future works for AI4REALNET usecases on how include the physics knowledge in the learning algorithms

Methodology

- Hybrid models: Physics Informed Graph Neural Networks (PIGNNs)

$$p_i^{prod} - p_i^{load} = (\theta_i \times y_{ii}) + \underbrace{(\theta_u \times y_{iu})}_{\text{message from node } u} + \underbrace{(\theta_v \times y_{iv})}_{\text{message from node } v} + \underbrace{(\theta_w \times y_{iw})}_{\text{message from node } w}$$



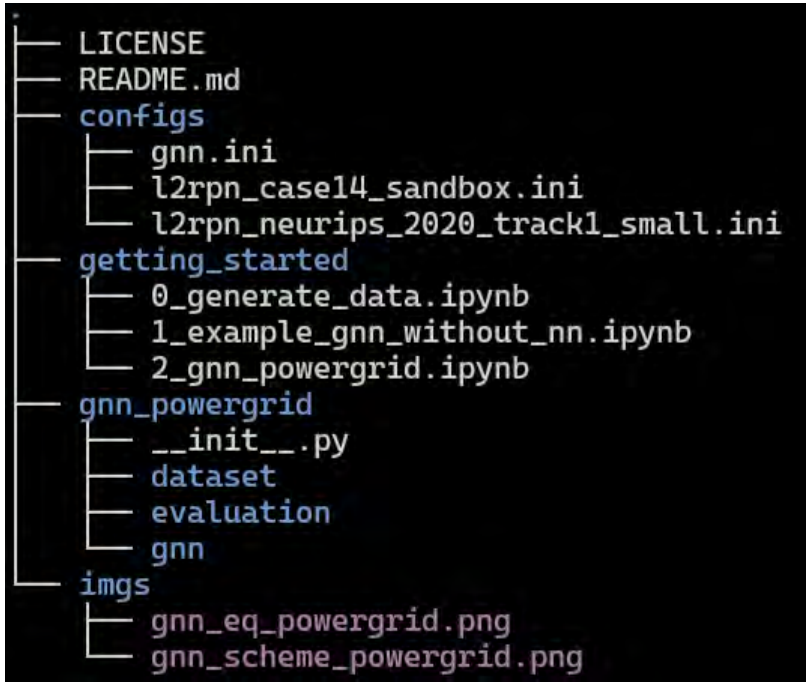
Original contribution

- Compliance to physics criteria / laws by integrating the local conservation as the optimization criteria (non-supervised learning)

ID	Type	Measure	Description
		Basic	
P1	Current positivity	$\frac{1}{L}\sum_{\ell}^{L}\mathbb{1}_{(a^{\ell}_{or,ex}<0.)}$	Proportion of negative current
P2	Voltage positivity	$\frac{1}{L}\sum_{\ell}^{L}\mathbb{1}_{(\hat{v}^{\ell}_{or,ex}<0.)}$	Proportion of negative voltages
P3	Losses positivity	$\frac{1}{L}\sum_{\ell}^{L}\mathbb{1}_{(\hat{p}^{\ell}_{ex}+\hat{p}^{\ell}_{or}<0.)}$	Proportion of negative energy losses
P4	Disconnected Line	$\frac{1}{L_{disc}}\sum_{\ell_{disc}}^{L_{disc}}\mathbb{1}_{(\lvert\hat{s}^{\ell}_{ex}\rvert+\lvert\hat{s}^{\ell}_{or}\rvert>0.)}$	Proportion of non-null a, p or q values
P5	Energy Losses	$\frac{\sum_{\ell=1}^{L}(\hat{p}^{(\ell)}_{ex}+\hat{p}^{(\ell)}_{or})}{Gen} \in [0.005, 0.04]$	energy losses range consistency
		Uni-dimension law	
P6	Global Conservation	$\mathrm{MAPE}((Prod-Load)-(\sum_{\ell=1}^{L}(\hat{p}^{\ell}_{ex}+\hat{p}^{\ell}_{or})))$	Mean energy losses residual
P7	Local Conservation	$\mathrm{MAPE}((p^{prod}_{k}-p^{load}_{k})-(\sum_{l\in neig(k)}\hat{p}^{\ell}_{k}))$	Mean active power residual at nodes
P8	Voltage equality	$\sum_{\substack{i,j\\ i,j\in k\\ i\neq j}}\mathbb{1}_{(\lvert v_i-v_j\rvert>0)}$	Proportion of not equal voltages at nodes

Overview of code structure

https://github.com/AI4REALNET/T2.1_graph_neural_solver



License (Mozilla Public License)

General overview and instructions to install dependencies

Configurations files used for initializing scenarios (two different l2rpn environments) and also the models (gnn.ini)

Getting started jupyter notebooks providing examples to generate data and reproduce the results

The Physics Informed GNN package, including dataset, evaluation and gnn modules

Input data

The data generated using a configuration file associated with a specific environment (2 envs):

- `L2RPN_case14_sandbox`: A toy environment including 14 nodes and 20 power lines
- `l2rpn_neurips_2020_track1_small`: A more complex environment including 38 nodes

The environment to use and paths to the configuration and dataset directory

```
env_name = "l2rpn_case14_sandbox"

path = pathlib.Path().resolve()
BENCH_CONFIG_PATH = path / "configs" / (env_name + ".ini")
DATA_PATH = path / "Datasets" / env_name / "DC"
LOG_PATH = path / "logs.log"
```

Specify the number of required data

```
NB_SAMPLE_TRAIN = 1e2
NB_SAMPLE_VAL = 1e2
NB_SAMPLE_TEST = 1e2
NB_SAMPLE_OOD = 1e2
```

The script to generate the dataset

```
benchmark = PowerGridBenchmark(benchmark_path=DATA_PATH,
                               benchmark_name="Benchmark4",
                               load_data_set=False,
                               config_path=BENCH_CONFIG_PATH,
                               log_path=LOG_PATH)


benchmark.generate(nb_sample_train=int(NB_SAMPLE_TRAIN),
                   nb_sample_val=int(NB_SAMPLE_VAL),
                   nb_sample_test=int(NB_SAMPLE_TEST),
                   nb_sample_test_ood_topo=int(NB_SAMPLE_OOD),
                   do_store_physics=True,
                   is_dc=True
                   )
```

Input data

The hyperparameters of the graph neural solver to be adjusted using two methods:

Using an existing configuration file

```
env_name="l2rpn_case14_sandbox"
name = "torch_gnn"
ref_node = 0
num_gnn_layers = 10
latent_dimension = 20
hidden_layers = 3
input_dim=2
output_dim=1
train_batch_size = 128
eval_batch_size = 128
device="cpu"
optimizer = {"name": "adam",
             "params": {"lr": 3e-4}}
epochs = 10
train_with_discount=False
save_freq = False
ckpt_freq = 50
```

Set them directly as arguments at the instantiation step which will override the ones imported by configuration file

```
SIM_CONFIG_PATH = path / "configs" / "gnn.ini"
gnn_simulator = GnnSimulator(model=GPGmodel,
                             name="gnn_torch",
                             sim_config_path=SIM_CONFIG_PATH,
                             input_size=2,
                             output_size=1,
                             epochs=EPOCHS)
```

All these configs could be set as arguments in class constructor

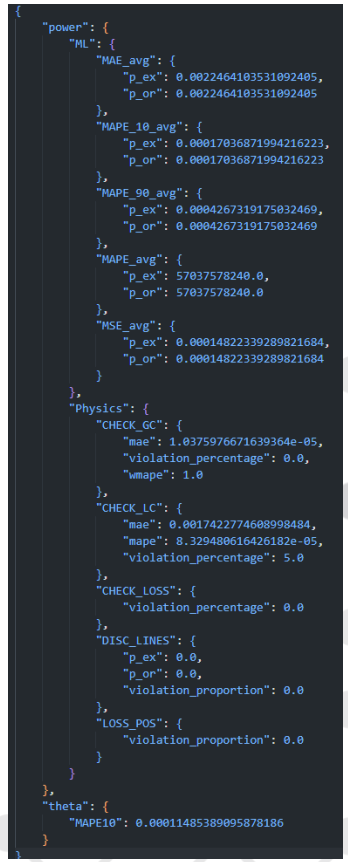
Output data

Obtained using main_wo_nn.py

- The outputs are evaluated criteria (ML & Physics) on the considered datasets
  - *Validation dataset*: dataset used for validation with the same distribution as training
  - *Test dataset*: dataset used for test presenting the same distribution as training data
  - *Test OOD dataset*: dataset used to test the out-of-distribution generalization capacity of the model

Machine Learning related criteria (accuracy measures)

Physics compliance criteria (physics constraints)



Quality of voltage angle estimation

Experiments

- Results (The values are the violation percentage of the corresponding metric)

Test dataset	Loss target	Output	Disc lines	Loss pos	Energy loss consistency	Global conservation	Local conservation
FC	P	P	0.0	43	0.0	88	91
GNN	θ	P	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**

Test OOD dataset	Loss target	Output	Disc lines	Loss pos	Energy loss consistency	Global conservation	Local conservation
FC	P	P	0.0	43	0.0	95	93
GNN	θ	P	**0.0**	**0.0**	**0.0**	**0.0**	**3.08**

Perspectives

- Improve the implementation of neural network based GNN which requires more training and layers to achieve the same performance as the GNN without NN
- Generalize the proposed approach for AC power flow simulation
- The physics compliance integration could inspire the algorithm design for batch 2 focusing on a control problem using Deep Reinforcement Learning algorithms
- Integrating the simplified physical equation and expert knowledge into RL agents which should suggest remedial actions with respect to observed context (environment)

Authors

Authors	Institution
Milad Leyli-abadi	IRTSX (IRT SystemX)

Expert Agent

Task 2.1 – Knowledge-assisted AI

IRT SystemX

AI4REALNET has received funding from European Union's Horizon Europe Research and Innovation programme under the Grant Agreement No 101119527

Outline

- Context
- Methodology
- Original Contribution
- Overview of code structure
- Algorithm #1
- Algorithm #2

Context

Definition

Expert Agent is an RL agent where the exploration capability may be guided by expert knowledge.

Motivation

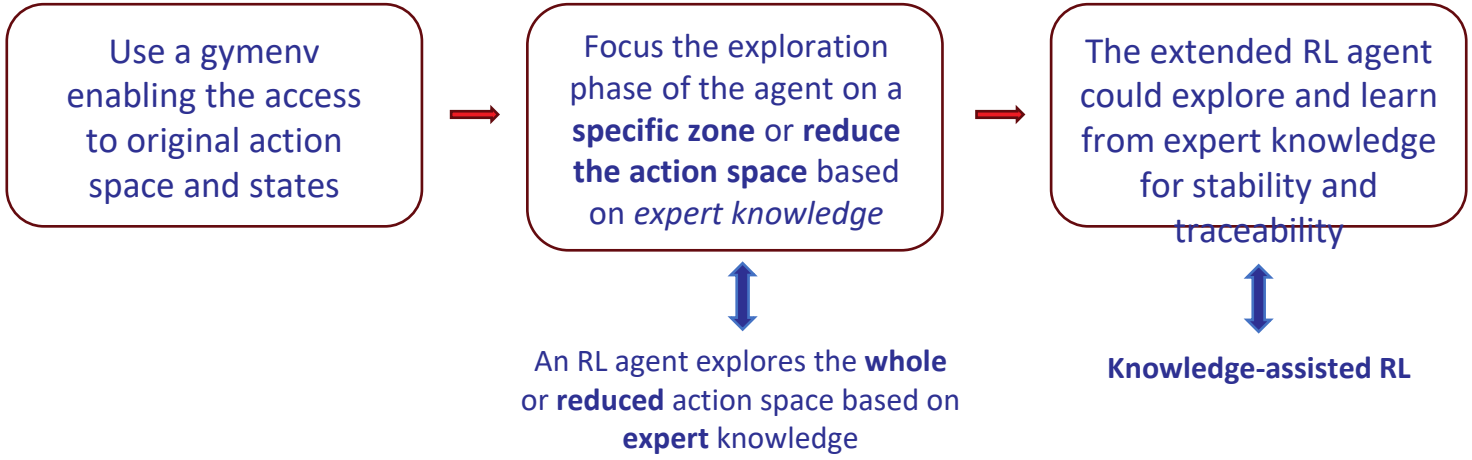
The huge action space size in some critical infrastructure may cause some problems such as generalization, overfitting, instable and delayed convergence and exploration becoming intractable. The expert knowledge help the agent to be trained on effective and reduced action space while not avoiding the whole actions space exploration.

Use cases

This implementation is specific to Power Grid domain and power grid assistant use case. In this use case, AI assistant supports human operators in decision-making and managing power grid congestion by suggesting remedial actions.

Methodology

- **Main idea** = Exploit the Expert Knowledge using *ExpertOp4Grid* package [1] to
  - (1) focus the exploration phase of an RL agent on specific zones of a power grid
  - (2) reduce the action space to most relevant ones and improve the scalability or RL agents



- **Preliminary experiments** on Power Grids, but could easily be extended to other domains if the expert knowledge is available

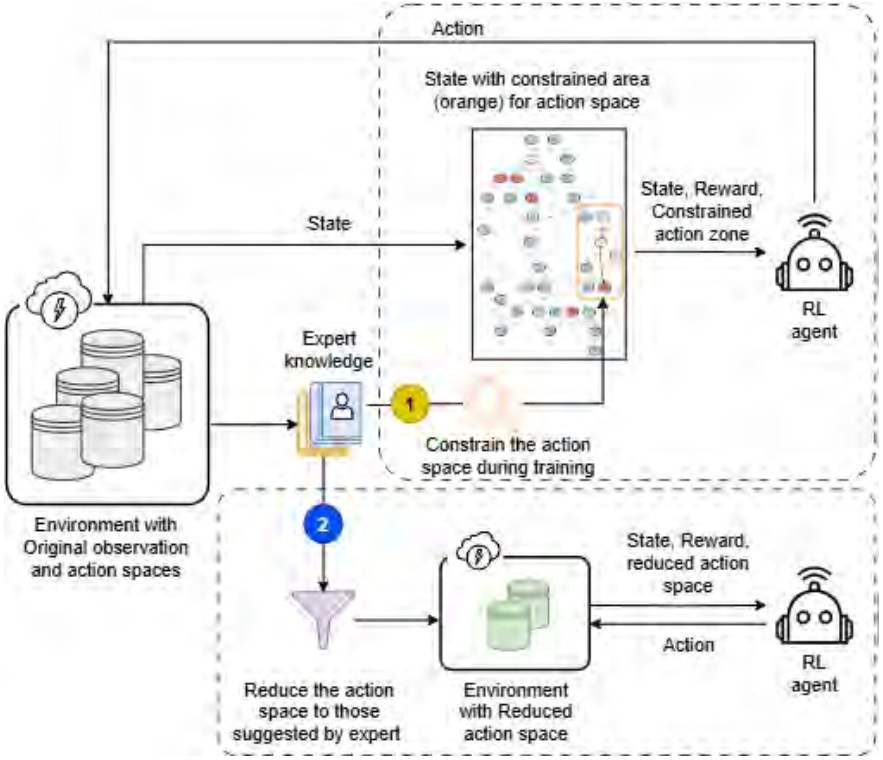
[1] https://github.com/marota/ExpertOp4Grid

Schematic representation of the methods

- **Main idea** = Exploit the Expert Knowledge to

1 focus the exploration phase of an RL agent on specific zones of a power grid

2 reduce the action space to most relevant ones and improve the scalability or RL agents



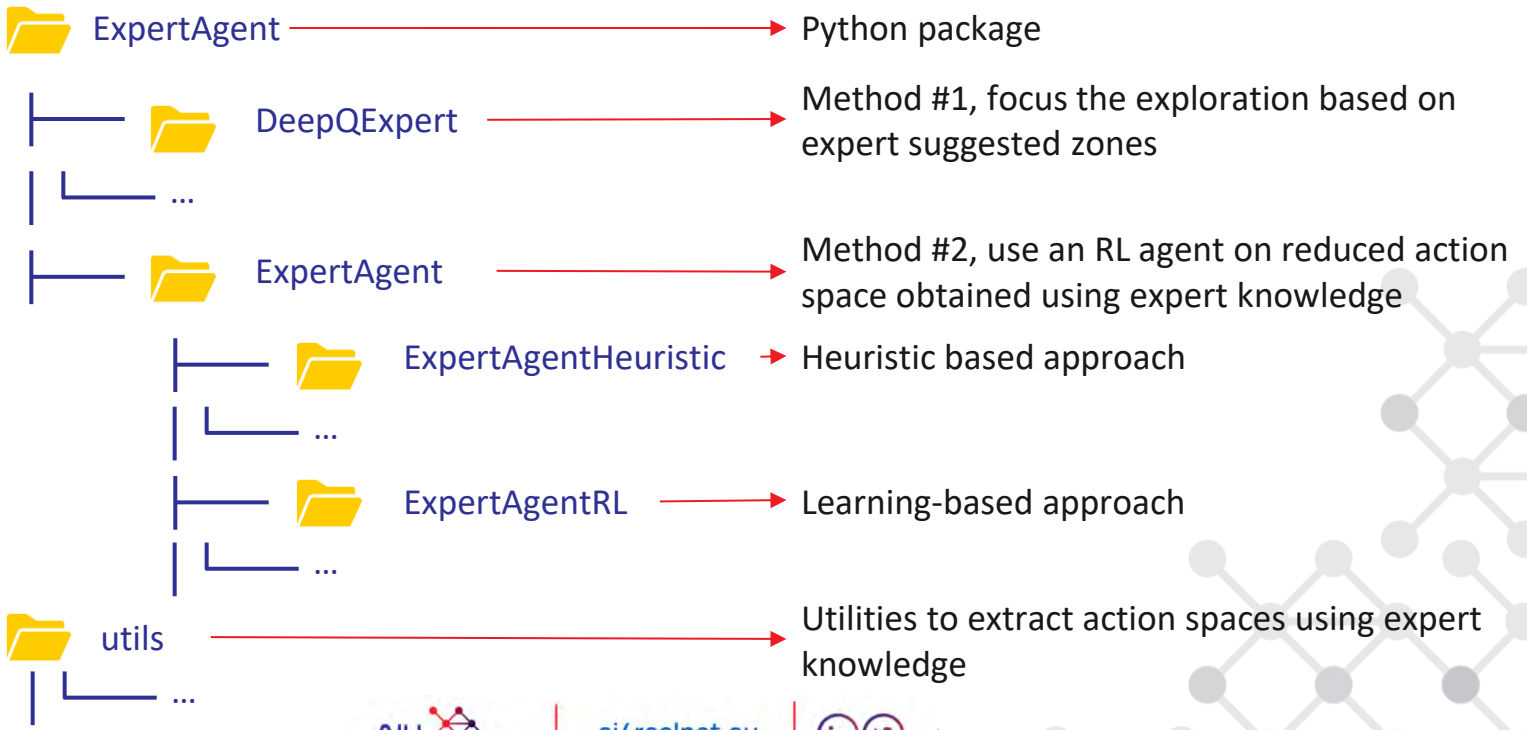
Original contribution

- **Algorithm #1 |** DeepQExpert
  - ➢ **Short description:** Focus the learning to the zones suggested by expert knowledge
  - ➢ **State-of-the-art:** Classical deep reinforcement learning algorithms[1] or expert system[2]
  - ➢ **Contribution:** Exploitation of expert knowledge during the training of an RL agent
  - ➢ **Implemented WP2 features:** Knowledge-assisted RL
- **Algorithm #2 |** ExpertAgent (better scalability)
  - ➢ **Short description:** Exploiting the reduced action-space obtained using expert knowledge as preprocessing
  - ➢ **State-of-the-art:** Classical deep reinforcement learning algorithms[1] or expert system[2]
  - ➢ **Contribution:** Train the classical RL algorithms on reduced action space obtained using expert knowledge
  - ➢ **Implemented WP2 features:** Knowledge-assisted RL

[1]Rocchetta, Roberto, et al., A reinforcement learning framework for optimal operation and maintenance of power grids (2019)
[2]Marot, Antoine, et al., Expert system for topological remedial action discovery in smart grids (2018)

Overview of repository structure

The provided repository includes a python package implementing two main expert agents



Algorithm #1

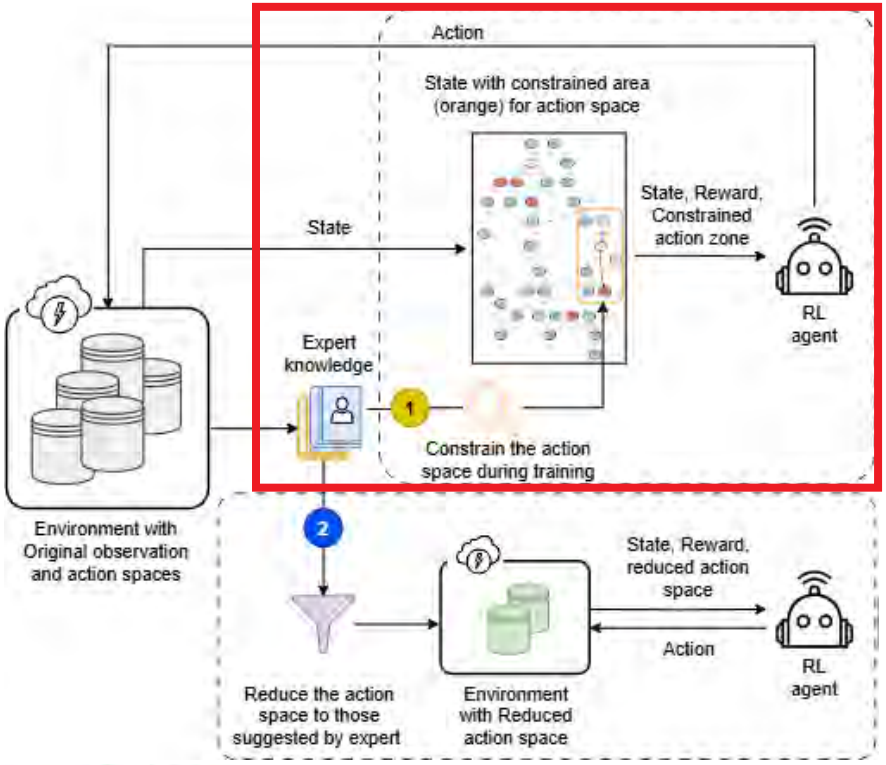
Schematic representation of Algorithm #1

- **Main idea** = Exploit the Expert Knowledge to

1 focus the exploration phase of an RL agent on specific zones of a power grid

2 reduce the action space to most relevant ones and improve the scalability or RL agents

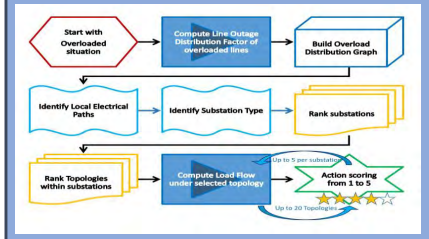
  - Heuristic-based
  - RL-based



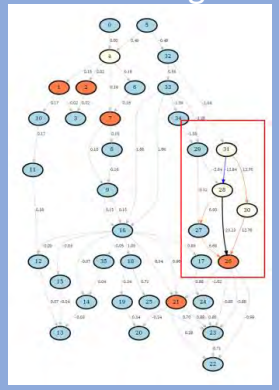
Procedure

Harness Expert knowledge for an RL agent

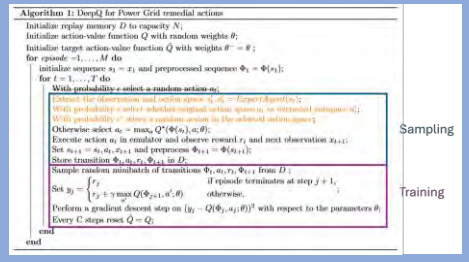
ExpertOp4Grid framework



Extract expert knowledge



Implementation of the adapted DeepQ to exploit the expert knowledge in power grid use case



Experimentation and benchmark

Analysis of the impact of the expert knowledge (transparency + Explainability)

Use an existing work to exploit expert knowledge

Allows to identify the high impact zones for exploration

Extended DeepQ algorithm harnessing expert-suggested zones

Explaination
See presentation
T2.3 - explainability

Algorithm

DeepQ Simple

Algorithm 1: DeepQ Simple for Power Grid remedial actions

Initialize replay memory D to capacity N;
Initialize action-value function Q with random weights θ;
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$;
for *episode* $=1, \ldots, M$ **do**
  initialize sequence $s_1 = x_1$ and preprocessed sequence $\Phi_1 = \Phi(s_1)$;
  for $t = 1, \ldots, T$ **do**
    With probability ϵ select a random action a_t;
    Otherwise select $a_t = \max_a Q^*(\Phi(s_t), a; \theta)$;
    Execute action a_t in emulator and observe reward r_t and next observation x_{t+1};
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\Phi_{t+1} = \Phi(s_{t+1})$;
    Store transition $\Phi_t, a_t, r_t, \Phi_{t+1}$ in D;
    Sample random minibatch of transitions $\Phi_t, a_t, r_t, \Phi_{t+1}$ from D ;
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1, \\ r_j + \gamma \max_{a'} Q(\Phi_{j+1}, a'; \theta) & \text{otherwise,} \end{cases}$;
    Perform a gradient descent step on $(y_j - Q(\Phi_j, a_j; \theta))^2$ with respect to the parameters θ;
    Every C steps reset $\hat{Q} = Q$;
  end
end

DeepQ Expert

Algorithm 1: DeepQ for Power Grid remedial actions

Initialize replay memory D to capacity N;
Initialize action-value function Q with random weights θ;
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$;
for *episode* $=1, \ldots, M$ **do**
  initialize sequence $s_1 = x_1$ and preprocessed sequence $\Phi_1 = \Phi(s_1)$;
  for $t = 1, \ldots, T$ **do**
    ~~With probability ϵ select a random action a_t;~~
    Extract the observation and action space $s'_t, a'_t = ExpertAgent(s_t)$;
    With probability ϵ select whether original action space a_t or extracted subspace a'_t;
    With probability ϵ' select a random action in the selected action space;
    Otherwise select $a_t = \max_a Q^*(\Phi(s_t), a; \theta)$;
    Execute action a_t in emulator and observe reward r_t and next observation x_{t+1};
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\Phi_{t+1} = \Phi(s_{t+1})$;
    Store transition $\Phi_t, a_t, r_t, \Phi_{t+1}$ in D;
    Sample random minibatch of transitions $\Phi_t, a_t, r_t, \Phi_{t+1}$ from D ;
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1, \\ r_j + \gamma \max_{a'} Q(\Phi_{j+1}, a'; \theta) & \text{otherwise,} \end{cases}$;
    Perform a gradient descent step on $(y_j - Q(\Phi_j, a_j; \theta))^2$ with respect to the parameters θ;
    Every C steps reset $\hat{Q} = Q$;
  end
end

Sampling

Training

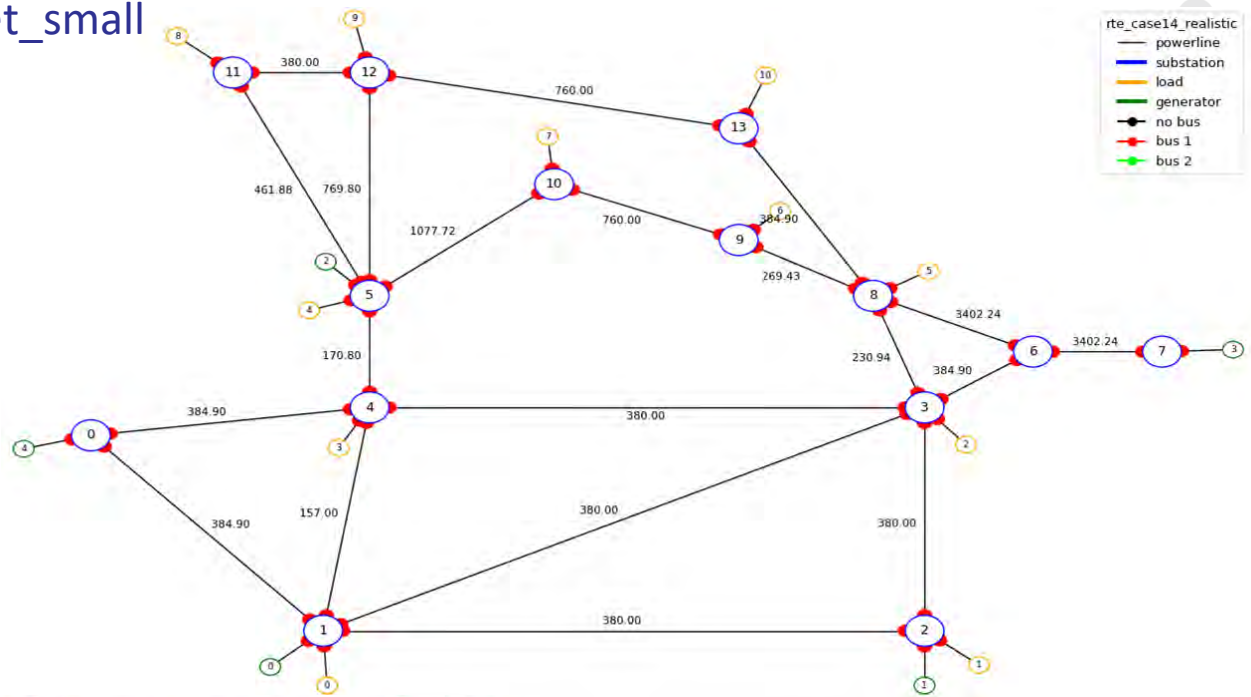
Overview of code structure

DeepQ harnessing the expert knowledge

- DeepQExpert → Python package
  - L2rpn_case14_sandbox → Trained DeepQ weights for 14 nodes environment
  - ...
  - DeepQAgent.py → Base class including main functionalities
  - DeepQNN.py → Neural Network (NN) implementation (tensorflow)
  - deepQExpert.py → Expert Agent class including training loop
  - config.ini → Configuration file used for expert knowledge (ExpertOp4Grid)
  - train.py → Script provided to train the agent
  - evaluate.py → Script provided to evaluate the agent

Grid representation

- Environment = ai4realnet_small
- # Lines = 20
- # Substations = 14
- # Generations = 6
- # Loads = 11



Inputs

- Observation space include the following variables with a total of 197 values

Variable	Length	Variables	Length
day_of_week	1	actual_dispatch	6
hour_of_day	1	target_dispatch	6
minute_of_hour	1	topo_vect	57
prod_p	6	line_status	20
prod_v	6	time_before_cooldown_line	20
load_p	11	time_before_cooldown_sub	14
load_q	11	rho	20
		timestep_overflow	20

Inputs (for 36 nodes)

- Observation space include the following variables with a total of 614 values

Variable	Length	Variables	Length
day_of_week	1	actual_dispatch	22
hour_of_day	1	target_dispatch	22
minute_of_hour	1	topo_vect	177
prod_p	22	line_status	59
prod_v	22	time_before_cooldown_line	59
load_p	37	time_before_cooldown_sub	36
load_q	37	rho	59
		timestep_overflow	59

Output data

- A folder with saved models, hyperparameters, cumulative rewards and alive times steps, and action space used during the training of the model
- A folder including the evaluation logs for each chronic selected during the evaluation and the following information:
  - Execution_time
  - Rewards
  - Other rewards (if there are any other optional rewards are computed in parallel)
  - Actions
  - Env_modifications
  - Observations
  - Opponent attack

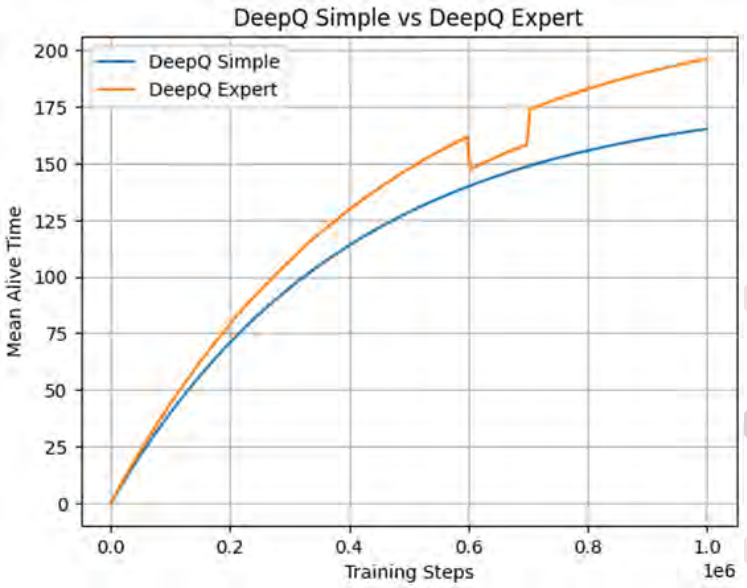
Experiments

DeepQ Simple

Training loss	0.0
Mean Alive	1 17 8

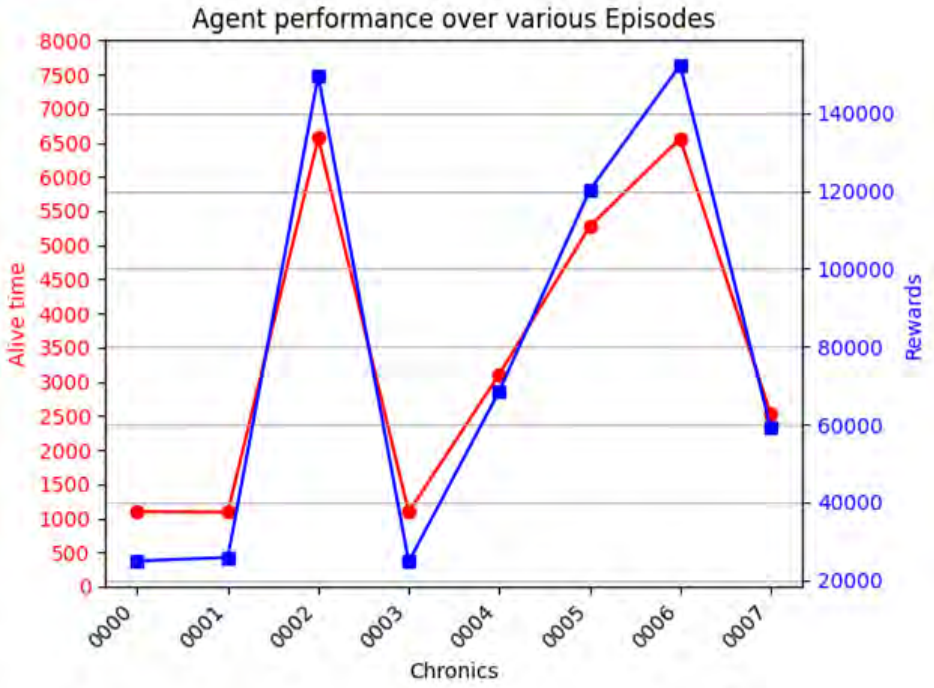
DeepQ Expert

Training loss	0.00
Mean Alive	8 23 0



Performance on test scenario

- DeepQExpert
- Number of episodes: 8
- Max duration: 8000
- Performance measured based on
  - Agent's alive time
  - Reward



Perspectives

- Improve the agent hyperparameters using a fine-tuning algorithm

Algorithm #2

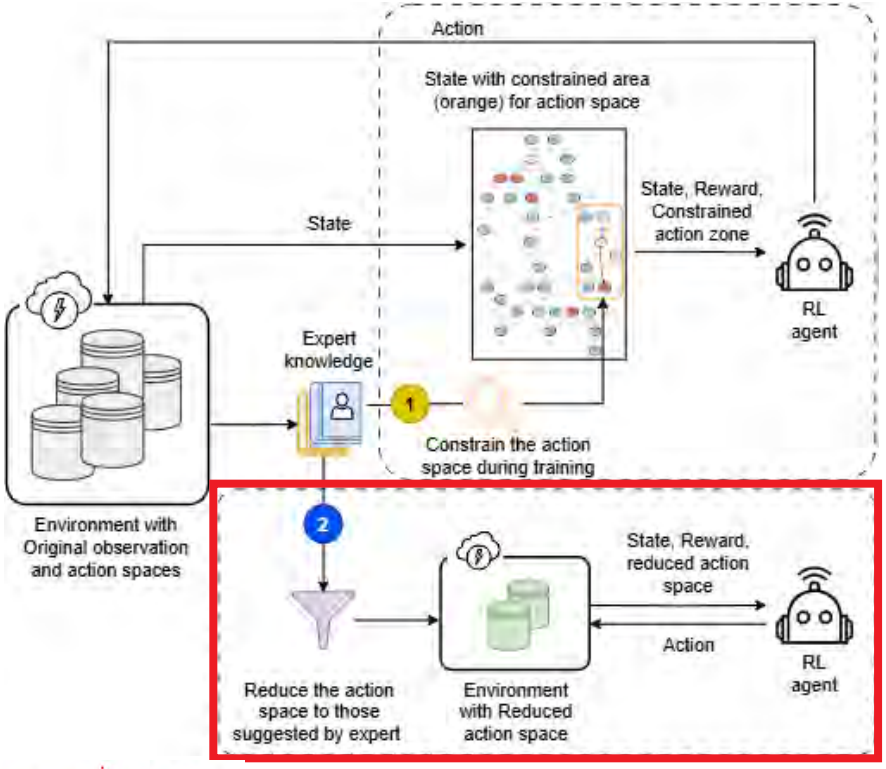
Schematic representation of the methods

- **Main idea** = Exploit the Expert Knowledge to

1 focus the exploration phase of an RL agent on specific zones of a power grid

2 reduce the action space to most relevant ones and improve the scalability or RL agents

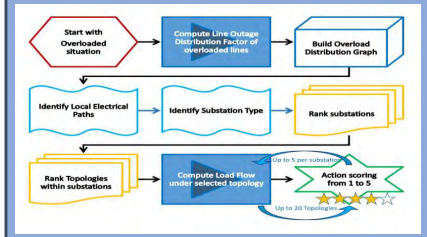
  - Heuristic-based
  - RL-based



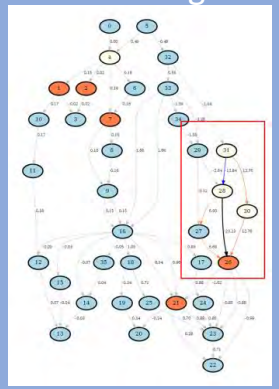
General Procedure

Harness Expert knowledge for an RL agent

Use ExpertOp4Grid framework

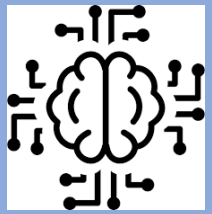


Extract expert knowledge



Using an approach to harness the expert knowledge

Heuristic-based Option 2.A



RL-based Option 2.B

Experimentation and benchmark

Analysis of the impact of the expert knowledge (transparency + Explainability)

STEP 1
Reduce the action space

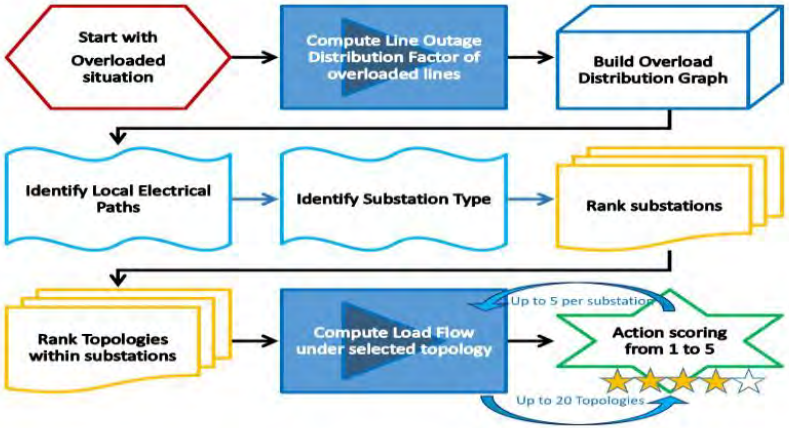
Extracting effective actions using expert

STEP 2
Selecting an action

Explanation
See presentation T2.3 - explainability

Step 1. Reduce action space

ExpertOp4Grid package



Simulate and assess the remedial actions and attribute a score as:

- 4 - it solves all overloads,
- 3 - it solves only the overload of interest
- 2 - it partially solves the overload of interest
- 1 - it solves the overload of interest but worsen other overloads
- 0 - it fails to relieve even partially the overload

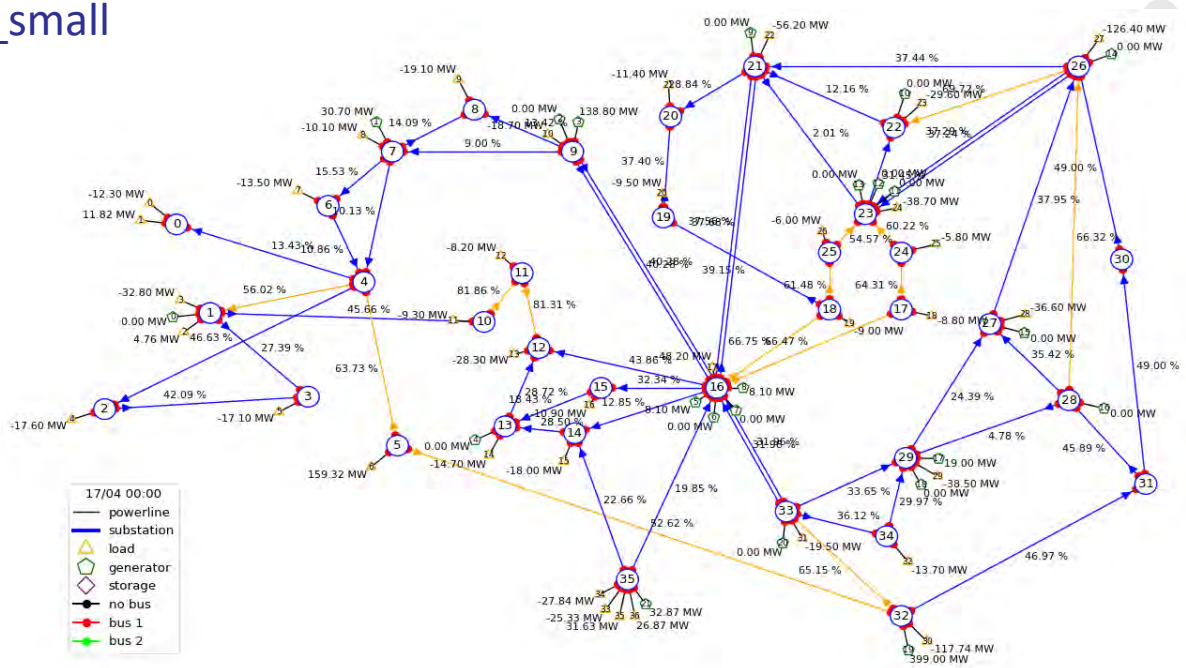
Action extraction pseudocode

```
# list to keep reduced action space
ExpertActions = []
# iterate over all the episodes
for episode in all_episodes:
   # iterate over all the time stamps
   for timestamp in len(episode):
      # call ExpertOp4grid Package
      topoActions, scores = call ExpertOp4Grid
      # filter actions with high scores (3 & 4)
      candidateActions = topoActions[scores>=3.]
      # verify to not implicate new overloads
      for action in candidateActions:
        obs, reward, done = env.simulate(action)
        # check no overload
        if obs.rho.max() < 1.:
           # append the action to the list
              topoActions.append(action)
# store all the extracted actions
store(ExpertActions)
```

https://github.com/marota/ExpertOp4Grid

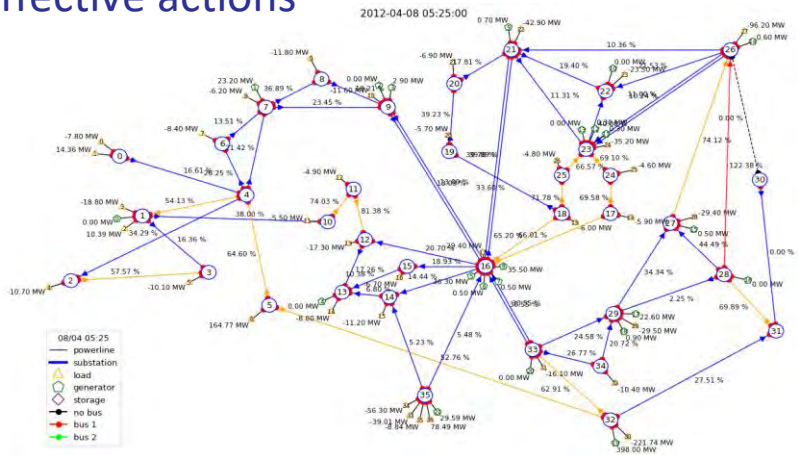
Grid representation

- Environment = ai4realnet_small
- # Lines = 59
- # Substations = 36
- # Generations = 22
- # Loads = 37

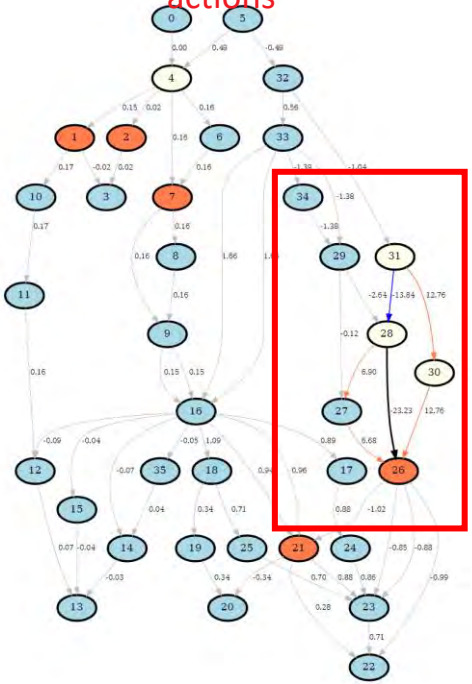


Reduced action space

- Extracting most effective actions using ExpertOp4Grid running on various scenarios
- Reducing the original action space including only *topological action set* actions from ~ 30 000 to only 224 effective actions



Overload Graph
Identifying most effective actions



Step 2. Option a) Use heuristic-based agent

Inspired by and extending the winning solution of L2RPN 2023 IDF challenge

Greedy search over reduced action space obtained using the expert knowledge

Use a set of well-known heuristics for power grid control

- Reconnecting the disconnected lines
- Return to the reference topology

Apply an existing convex optimization for continuous control (credit to LJN agent)

```
env, obs, reward # Inputs : Get the environment, current observation and reward
action = env.action_space({}) # Initialize the action by doing nothing
reco_action = try_to_reconnect(obs, action, reward) # Try to reconnect
if reco_action is not None: # verify if there exists a potential reconenction action
   _obs, _reward, _done = obs.simulate(action, time_step=1) # simulate the act
   if not(done) & obs.rho.max() < 2.:
      action += reco_action # add the reconnection to initial action if the overload less than 2
if obs.rho.max() > 0.99: # if an overload is observed
  # try to recover the reference topology if possible
  recover_act = try_to_recover_topo(obs, action, reward)
  if recover_act is not None:
    action += recover_act
  else:
    # retrieve the reduced action space obtained by the expert from step 1 (previous slide)
    topo_act = search_expert_actions(reduced_action_space)
    action += topo_act if topo_act is not None
    _obs, _reward, _done = obs.simulate(topo_act, time_step=1)
    if_obs.rho.max() > 0.9:
      # Peform a convex optimization for continuous actions if no effective actions suggested yet
      action = get_continuous_action_by_optimization(obs, action, reward)
elif obs.rho.max() < 0.9:
  # try to recover the reference topology if possible whenever possible, even if no overload observed
  recover_act = try_to_recover_topo(obs, action, reward)
  if recover_act is not None:
    action += recover_act
```

Input data

Heuristic-based agent

- Grid2op Environment (ai4realnet_small)
- The path to reduced action space (obtained using expert knowledge)
- The definition of overload thresholds
  - Danger >= 0.99
  - Safe = 0.9

Evaluation

- Number of episodes for evaluation
- Max iterations (time stamps) to evaluate per episode

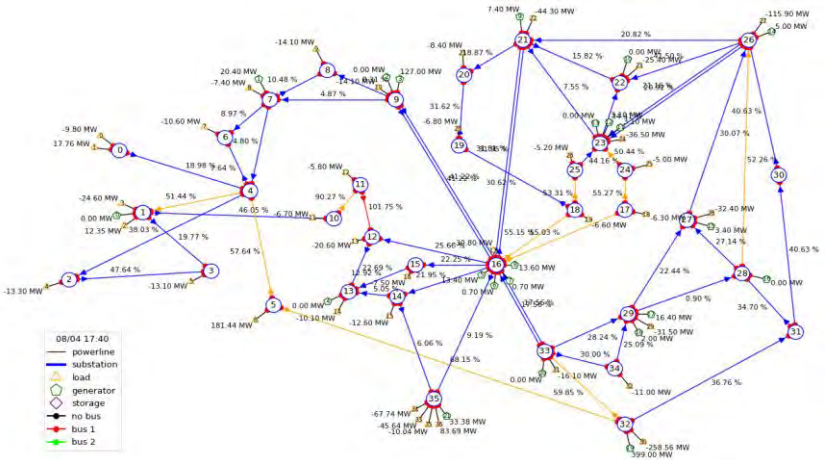
Output data

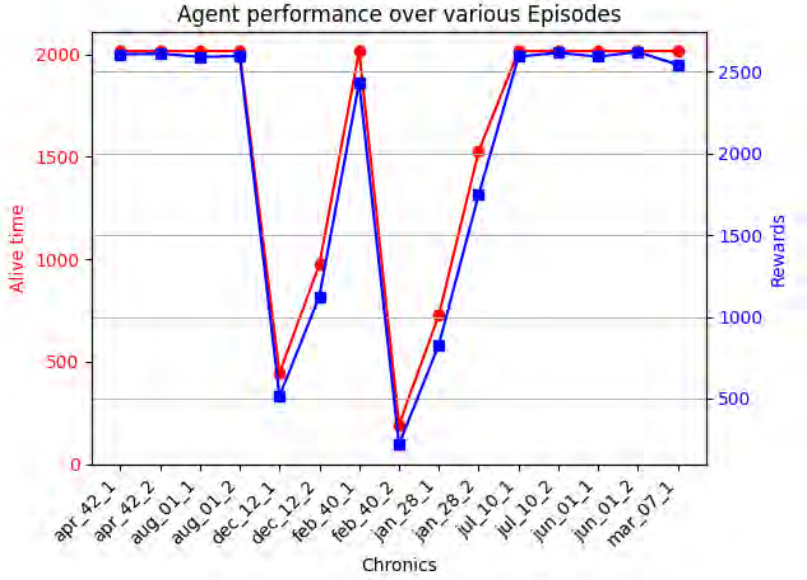
- Evaluation logs
- Proposed actions based on heuristic and expert knowledge
- Diagram showing the performance of the agent

Experiment

- Tested on 15 episodes of environment

`ai4realnet_small`





Step 2. Option b) Use RL-based agent

Train a Proximal Policy Optimization (PPO) algorithm on reduced action space obtained in step 1

Use a set of well-known heuristics for power grid control at inference time

- Reconnecting disconnected lines
- Return to the reference topology

Apply an existing convex optimization for continuous control (credit to LJN agent)

```
env, obs, reward # # Inputs : Get the environment, current observation and reward
action = env.action_space({}) # Initialize the action by doing nothing
reco_action = try_to_reconnect(obs, action, reward) # Try to reconnect
if reco_action is not None: # verify if there exists a potential reconenction action
   _obs, _reward, _done = obs.simulate(action, time_step=1) # simulate the act
   if not(done) & obs.rho.max() < 2.:
      action += reco_action # add the reconnection to initial action if the overload less than 2
if obs.rho.max() > 0.99: # if an overload is observed
  # try to recover the reference topology if possible
  recover_act = try_to_recover_topo(obs, action, reward)
  if recover_act is not None:
    action += recover_act
  else:
    # search the best action over top_k prediction of ExpertAgent RL
    topo_act = search(ExpertAgent_RL.top_k_predictions)
    action += topo_act if topo_act is not None
    _obs, _reward, _done = obs.simulate(topo_act, time_step=1)
    if_obs.rho.max() > 0.9:
      # Peform a convex optimization for continuous actions if no effective actions suggested yet
      action = get_continuous_action_by_optimization(obs, action, reward)
elif obs.rho.max() < 0.9:
  # try to recover the reference topology if possible whenever possible, even if no overload observed
  recover_act = try_to_recover_topo(obs, action, reward)
  if recover_act is not None:
    action += recover_act
```

Input data

RL-based agent

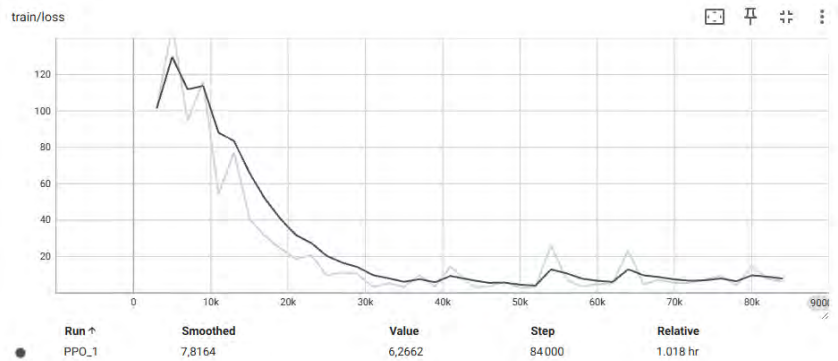
- Grid2op Environment (ai4realnet_small) and its Gym counterpart
- The path to the trained PPO model
- The hyper-parameters of the PPO policy (MLP Policy)

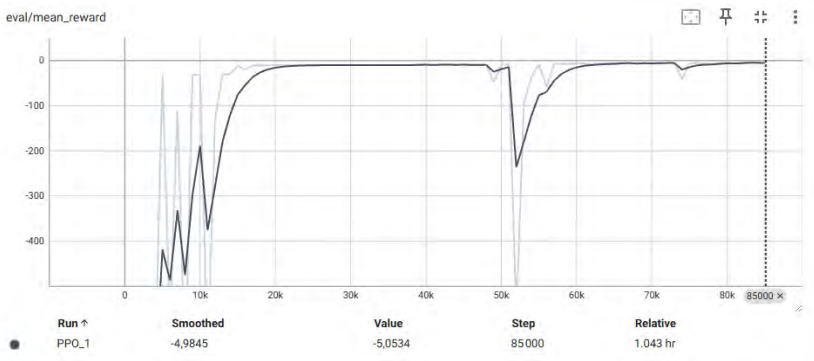
Evaluation

- Number of episodes for evaluation
- Max iterations (time stamps) to evaluate per episode
- Agent
- Environment
- Seed

Output data

- Evaluation logs
- Proposed actions based on heuristic and expert knowledge
- Diagram showing the performance of the agent
- Tensorboard logs showing convergence information and training losses

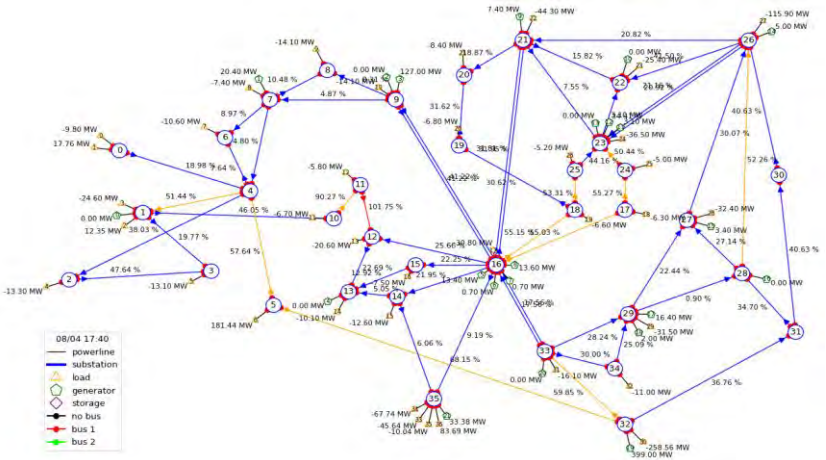


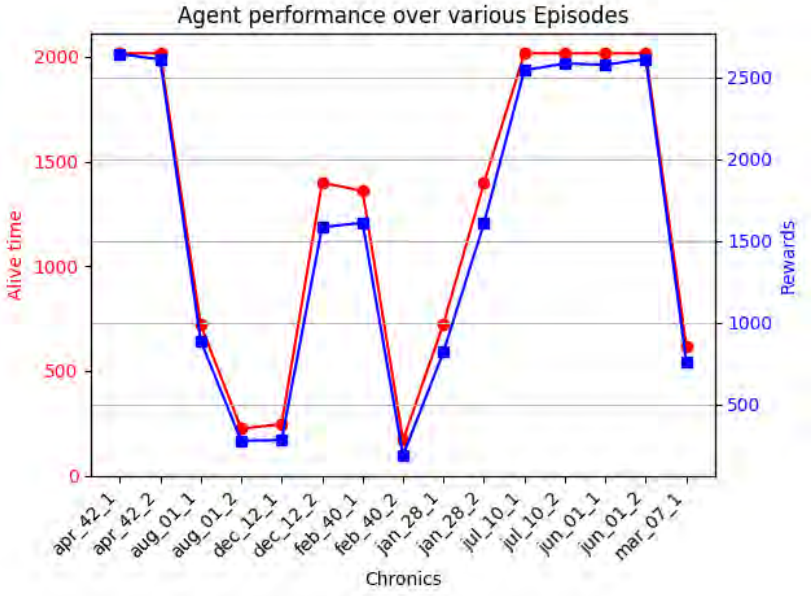


Experiment

- Tested on 15 episodes of environment

`ai4realnet_small`

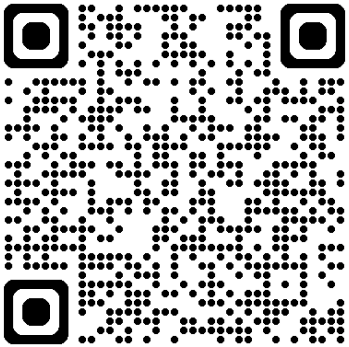




Perspectives

- Extensive hyperparameter tuning of the RL model (PPO)
- Integrate various variations of the environment (scenarios) during the training

Link to the repository



AI4REALNET/T2.1_deep_expert: Expert Agent exploiting the expert knowledge during the training

Authors

Authors	Institution
Milad Leyli-abadi	IRTSX (IRT SystemX)

Explainability dashboard

IRT SystemX

Outline

- Context
- Methodology
- Original Contribution
- Overview of code structure
- Experiments

Context

Definition

Explainability dashboard provides a set of statistical measures and visualizations to make the AI agent decisions explainable, while providing guidance to user to ensure the safety.

Motivation

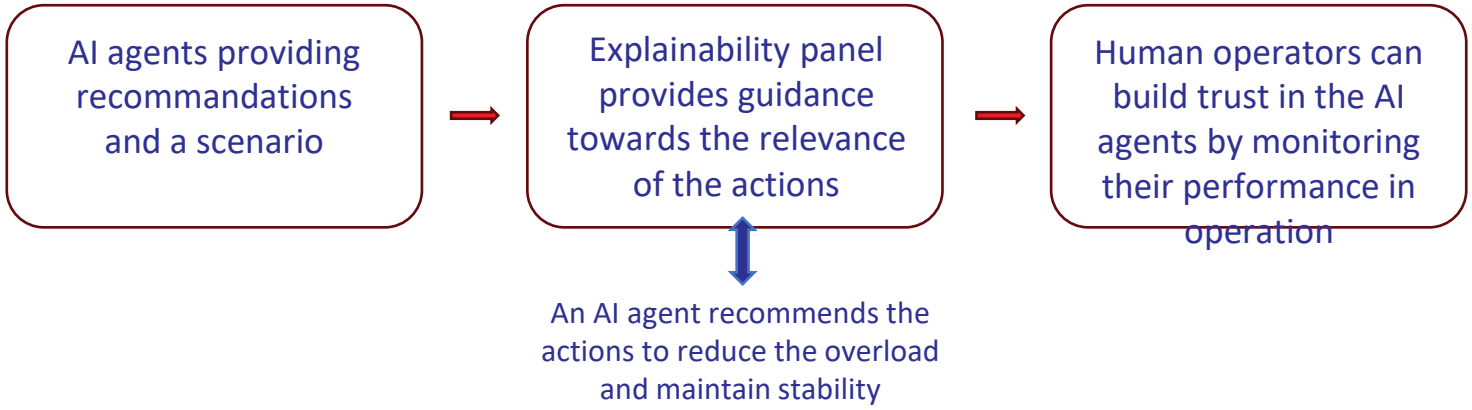
This dashboard supports understanding by connecting the recommendations to expert knowledge and complementing them with statistical analysis, visualizations, and brief explanations.

Use cases

This implementation is specific to Power Grid domain and power grid assistant use case. It serves as a companion tool for the AI agents developed in Task 2.1 (knowledge-assisted AI). In this use case, AI assistant supports human operators in decision-making and managing power grid congestion by suggesting remedial actions.

Methodology

- **Main idea** = leverage expert knowledge (using the ExpertOp4Grid package [1]) to explain and monitor the decisions made by AI agents and measure their relevance.



- **Preliminary experiments** on Power Grids, but could easily be extended to other domains if the expert knowledge is available

[1] https://github.com/marota/ExpertOp4Grid

Schematic representation of the panel

- **Main idea** = Monitor the AI recommendation

Task 2.1

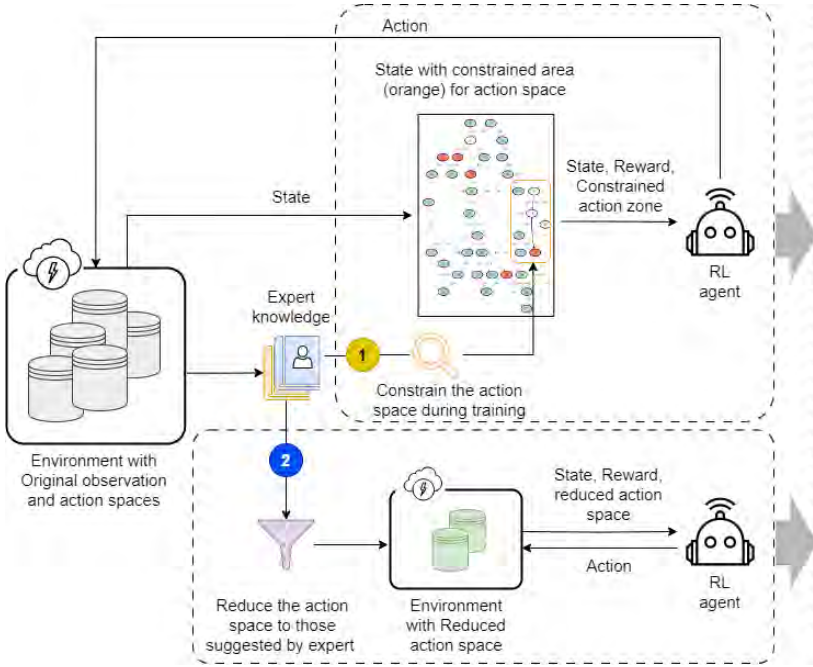
1 *AI Agent*: focus the exploration phase of an RL agent on specific zones of a power g

2 *AI Agent*: reduce the action space to most relevant ones and improve the s bility or RL agents

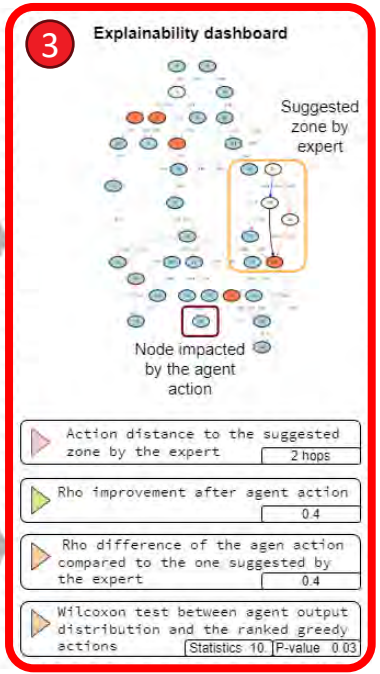
Task 2.3 (current)

3 Explainability Dashboard to analyze the AI agents (1 , 2) recommendations

Task 2.1
(See T2.1_KAI_ExpertAgent.pptx)



Task 2.3
(current)



Original contribution

- **Dashboard |** Explainability

➢ **Short description:** Explaining the AI agent recommendation by harnessing expert knowledge

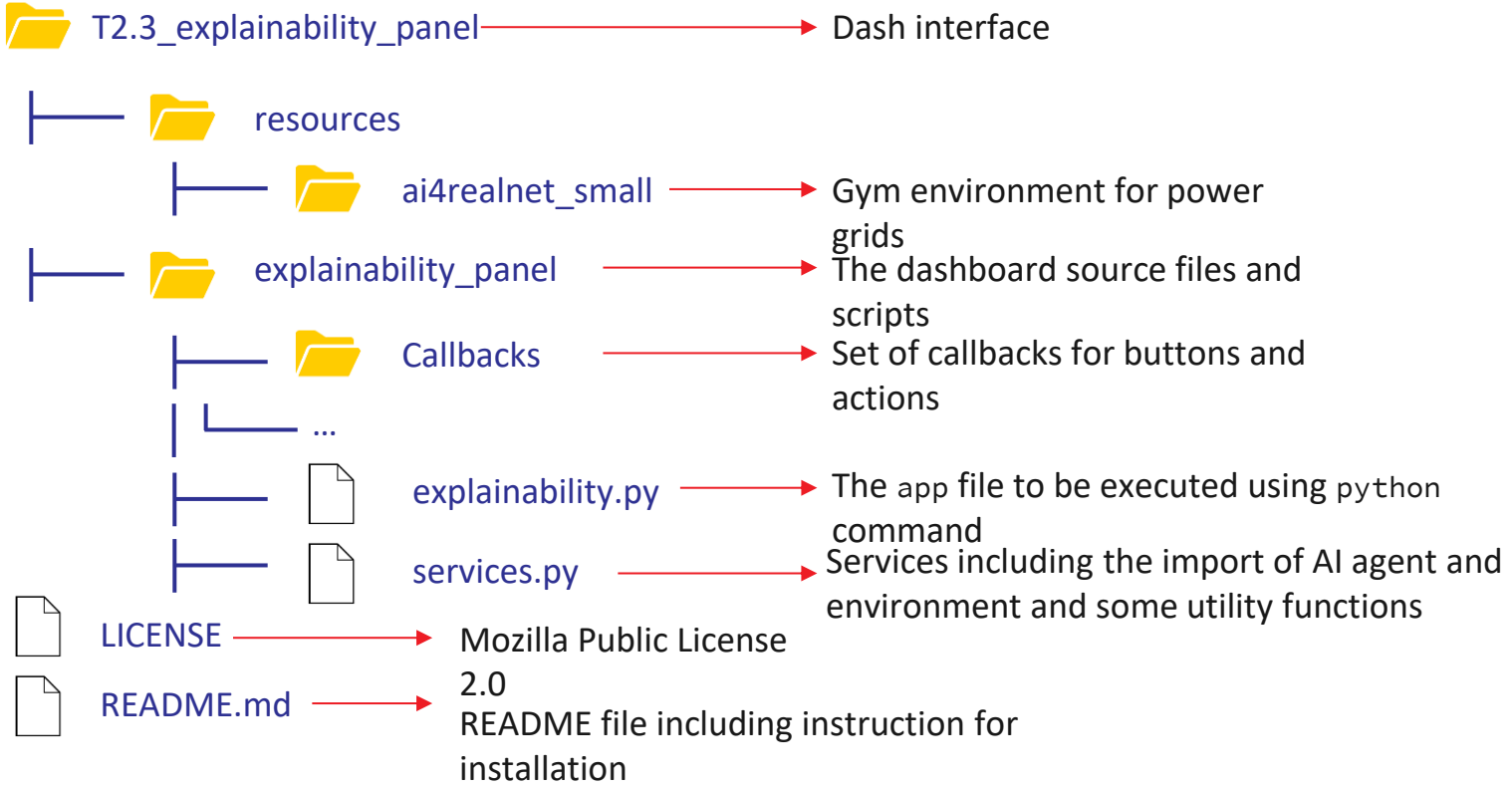
➢ **State-of-the-art:** Most of the explainability tools are generic and could not be used for specific use cases such as power grids

➢ **Contribution:** Exploitation of expert knowledge to enhance the trust towards the AI agent recommendations and develop domain-specific (Power Grids) explainability tool

➢ **Implemented WP2 features:** Knowledge-assisted RL**,** Explainability through the analysis of expert knowledge

Overview of code structure

The provided repository includes a python package implementing the interface using Dash library



Inputs and requirements

- RL agent → For the moment the explainability tool is adapted for the agents developed in Taks 2.1 of the project (knoweldge-assisted AI)
  - These agents could be installed as a package by cloning the corresponding Github repository

- RL environment → Using the Power Grid scenario designed for AI4REALNET project (`ai4realnet_small`)
  - This scenario could be obtained using the corresponding Github repository

- Dependencies → Installing the dependencies for this package using the requirements file provided in the Github repository

Dashboard presentation

- Suggesting a panel to follow up the agent decision with some statistics and visualizations

Statistics

Agent action distance to the expert knowledge : 2 hops

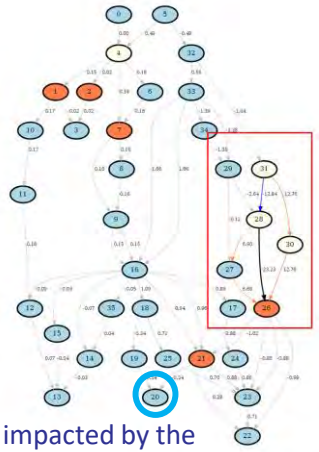
Rho improvement: 0.4

Rho difference wrt. the expert action: 0.2

Wilcoxon test between the agent output distribution and the ranked greedy actions:

Statistic 10. P-value 0.03

Visualization



Expert Interest Zone

Node impacted by the agent action

Multiple sources of information

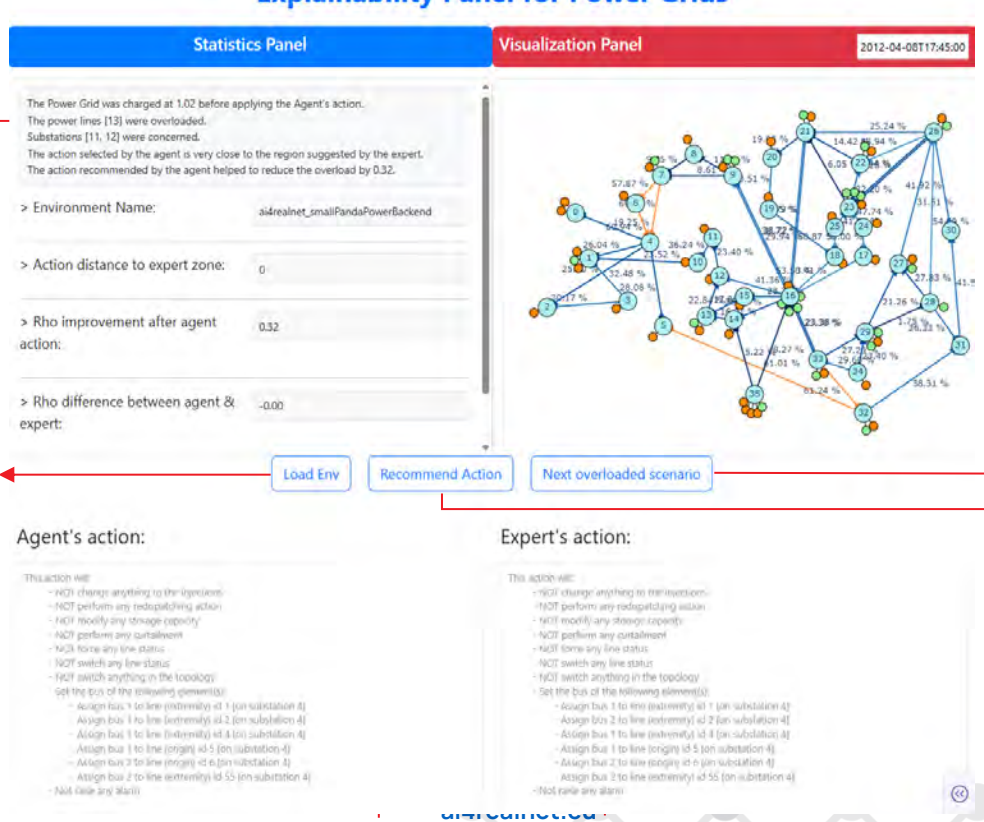
→ This information shows that the agent takes an action which is outside the zone that may be suggested by an expert method

→ The agent's action improves the Rho value (overload) by 0.4 points but is sub-optimal wrt. expert action.

→ Wilcoxon test shows that agent's output distribution is significantly (95% confidence) different from optimal greedy action.

Explainability dashboard – how to use

Textual explanation provided to user based on statistics or power grid current state and the state after the application of AI recommendation



Two functionalities
1. Load the environment for the first time
2. Return to the first time-stamp

Fast forward to the next overload scenario on the power grid

Two functionalities
1. Progress in scenario (next time stamp)
2. Take a recommendation from AI agent

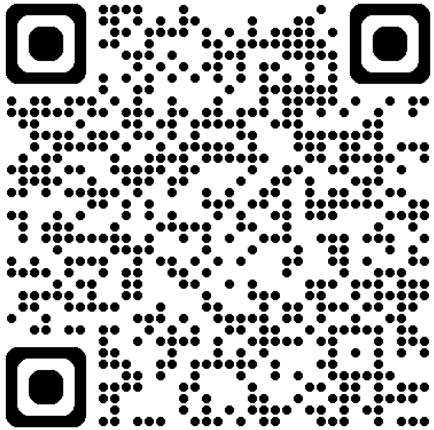
AI Agent's action

Expert action obtained using ExpertOp4Grid package

Perspectives

- Generate explanation using LLM
- Extend to other AI agents and ability to import AI agents via API
- Enabling the change of environment and scenarios during the analysis

Link to the repository



AI4REALNET/T2.3_explainability_dashboard: The explainability panel for RL agents and Power Grid use case

Authors

Authors	Institution
Milad Leyli-abadi	IRTSX (IRT SystemX)

GNPDT - Evolving Power System Operating Rules for Real-Time Congestion management

INESC TEC

Context

Definition

Novel hybrid AI framework for power grid topology control that integrates genetic network programming (GNP), reinforcement learning, and decision trees

Motivation

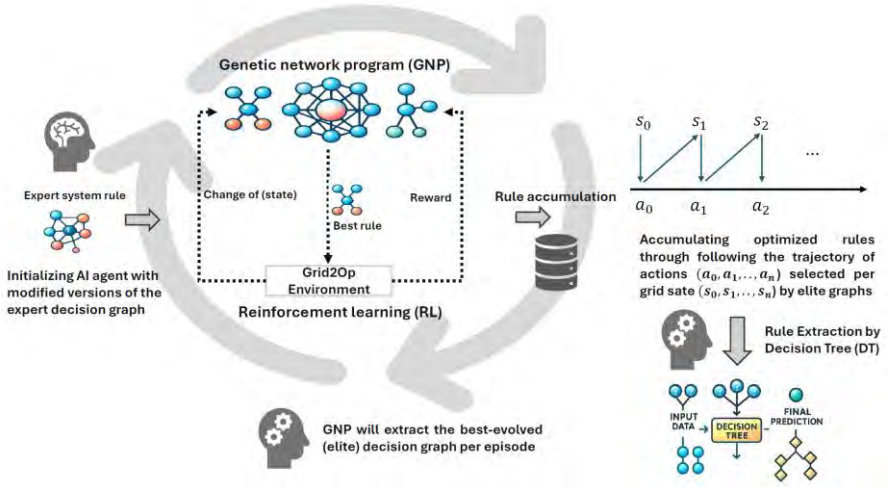
Existing expert systems are static and do not evolve from data, experience, and feedback from humans. The availability of data offers an opportunity to improve performance by learning

Use cases

Power grids use cases for congestion management, where an expert system is available but underperforms because it is not learning from data. Assist human operators in topology optimization and generation redispatch

General description

- Novel hybrid AI framework for power grid topology control that integrates genetic network programming (GNP), reinforcement learning, and decision trees
- A new variant of GNP is introduced that is capable of evolving the decision-making rules by learning from data in a reinforcement learning framework



A detailed description of the methodology can be found in deliverable D2.3

GitHub

General description

- The following contributions are produced
  - Genetic Network Programming (GNP) framework that incorporates dynamic node behavior, enabling context-aware decision-making in uncertain power system environments. Unlike conventional approaches that rely on fixed function nodes, this method employs functional nodes that dynamically adapt their behavior based on real-time system states
  - A hybrid methodology that combines Genetic Network Programming with Decision Tree (GNP-DT), and due to its graph-based structures, enhances interpretability through providing human-understandable reasoning for each control action
- The code targets real-time line congestion management in power grids. Its core concept involves evolving heuristic control policies, originally derived from operator expertise or pre-existing expert systems, encoded as a decision graph

Methodology

- The baseline knowledge for this work and also for the GNP rule evolution is the expert system (ES) developed by RTE, where the aim was to emulate the decision-making process of human operators. This ES was enhanced by considering improvements in the calculation method for determining the influence graph, using more flexibility options (e.g., line switching and redispatch, in addition to the bus splitting option), new ranking criteria, the possibility of decision revision, and the possibility of proposing multiple actions rather than a single action
- This code integrates a reinforcement learning GNP framework that supports dynamic and context-aware behavior. In this approach, each node is capable of altering its output depending on current grid conditions, such as line overloads, generation/load levels, or switching states
  - e.g., a judgment node that initially selects the reconfiguration action at 'Bus A in zone 1' under moderate congestion may instead recommend reconfiguration at 'Bus C in zone 3' if RES fluctuations create a localized congestion between the two zones
  - The node's behavior is therefore not statically defined, but instead learns a mapping from state features to decision criteria, enabling adaptive control that aligns with requirements for real-time management. Furthermore, the execution sequence of the decision graph is not fixed, unlike traditional GNP, which follows a hardcoded node traversal. Instead, the proposed method allows the policy graph to activate different substructures conditionally, based on current state inputs.

Methodology

- It has been recognized that a single decision graph may not generalize optimally across diverse grid conditions due to varying fault locations, network topologies, and intertemporal dependencies
- Therefore, the elite policies of the GNP algorithm are used to build a rule pool, consisting of the best-performing decision graph for each episode in the last generation. In particular, the elite graph represents an action selection trajectory that has the highest possible performance in an episodic evaluation
- The elite-derived actions per timestep will be used to generate labeled datasets (i.e., consisting of grid state features and the corresponding actions), in which these datasets are then used to train a multistage decision (DT) to capture different components of control logic
- Summary of the DTs and their corresponding state variables

Variable	DT1	DT2	DT3	DT3	DT3	DT3	DT4
Action type	any	any	bus reconfig.	line disconn.	bus recovery	line reconnection	bus reconfig.
Cong. Line IDs	✓	✓	✓	✓	✗	✗	✓
Max Overload	✗	✓	✓	✗	✗	✗	✗
Split Bus IDs	✓	✓	✓	✓	✓	✗	✓
Disconn. Line IDs	✓	✓	✓	✓	✓	✓	✓
Cong. Line Count	✓	✓	✓	✓	✗	✗	✓
Split Bus Count	✓	✓	✓	✓	✓	✗	✓
Disconn. Line Count	✓	✓	✓	✓	✓	✓	✓

Overview of repository structure

AIagent_GNP_modules

- This script is the GNP graph nodal (judgment and processing nodes) functions and the genetic main modules, including mutation, crossover, selection, and replacement. This script includes many functions that could not be used individually, but are essential to run other scripts
- This code is dependent to *lines_SI.py*, *reconfig_result.py*, *reconfigflx.py*, *Reconfiguration.py*, *VI_sens_l2rpn_wcci_2022.py* for addressing line switching and power flexibility actions

AIagent_GNPRL_train

- This is the main code used for training the adaptive GNP model in interaction with Reinforcement Learning and will extract the best-evolved (elite) decision graph per episode

top_rule_pool

- This script accumulates optimized rules through following the trajectory of actions (a_0,a_1,…, a_n) selected per grid state (s_0, s_1,…, s_n) by elite graphs

AIagent_test_predictor

- This script does the Rule Extraction by DT and will evaluate the AI agent over the test set

Usage of the repository

(1) The first step is to train the agent for a specific grid and parameters and call the train function, which will save the elite graphs in the training folder.

from GNPDT import AIagent_GNPRL_train from AIagent_GNPRL_train import train

The script contains an example execution in the if __name__ == "__main__": section at the end of the file, which demonstrates how to run the code with sample inputs.

To tune the GNP hyperparameters using Bayesian optimization, refer to Aiagent_GNPRL_hyperparam.py before setting the training parameters.

(2) Call the pool function from the *top_rule_track.py* to accumulate the action trajectory by elite decision graphs from training

from GNPDT import top_rule_track from AIagent_GNPRL_train import pool

The script contains an example execution in the *if __name__ == "__main__"*: section at the end of the file, which demonstrates how to run the code with sample inputs.

(3) Call the pool function from *AIagent_test_predictor.py* to run the decision tree for the selected test set and calculate the metrics

from GNPDT import AIagent_test_predictor from AIagent_test_predictor import validation, generate_random_numbers_in_range

The script contains an example execution in the if __name__ == "__main__": section at the end of the file, which demonstrates how to run the code with sample inputs

Input data

- String identifying Grid2Op environment
- State vector of each Grid2Op environment (line current, injected power, network topology, etc.) episode
- Topology and electrical characteristics of the power grid to calculate a sensitivity matrix
- **This repository works and interacts directly with the Grid2Op environment**

Output data

- Actions to be implemented via the Grid2Op interface
  - Topology change, generation redispatch, renewable energy curtailment
- Decision tree with interpretable controllable rules
- **This repository works and interacts directly with the Grid2Op environment**

Authors	Institution
Ferinar Moaidi	INESC TEC
Ricardo Bessa	INESC TEC

Planner enhanced AI

Task 2.1 – Knowledge-assisted AI

University of Amsterdam

Outline

- Context
- Methodology
- Original Contribution
- Overview of code structure
- Experiments

Context

Definition

Multi-Agent Path Finding (MAPF) involves finding collision-free paths for multiple agents in a shared environment.

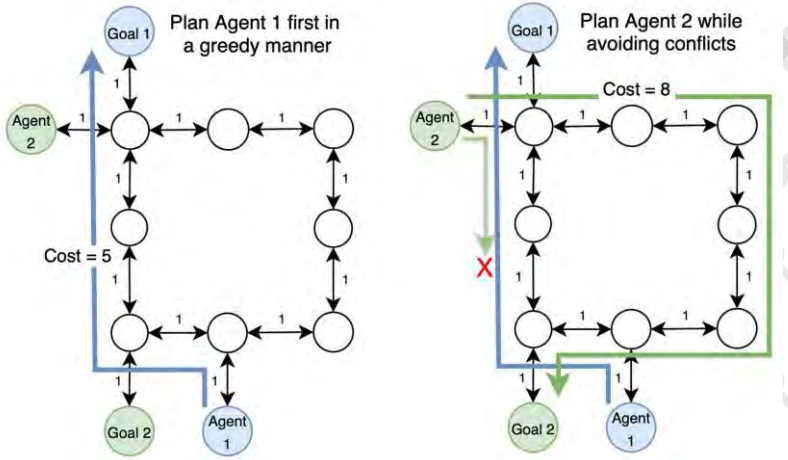
Motivation

Efficiently scheduling agents in congested networks (e.g., railway systems) is crucial for minimizing delays and optimizing flow. While optimal solvers do exist, they **fail to scale** to problems with more than a few dozen agents. More scalable solvers are

Railway networks: Scheduling and routing trains. Ability to quickly replan on case of train breakdowns.

Motivation: small scale example

- Plan Agent 1 **greedily**, by using shortest distance to goal
- Agent 2 has to **avoid collision** with Agent 1

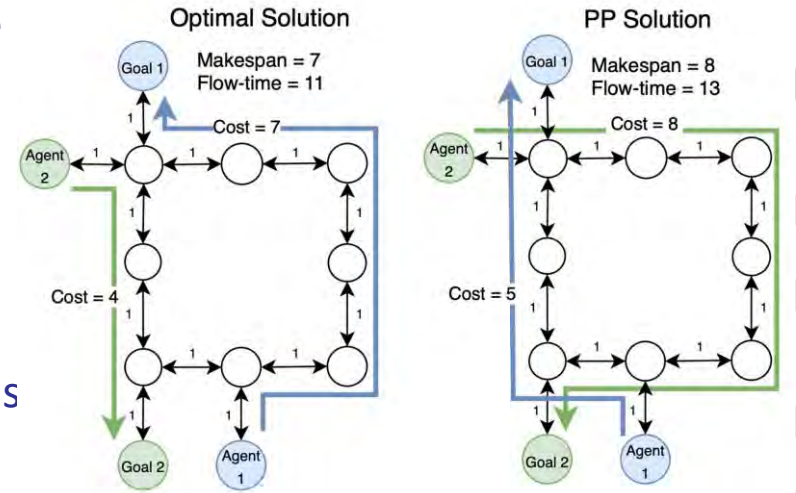


- **Prioritized Planning** (PP) is such an algorithm, that plans agents in sequence, based on the assigned priorities.
- Often times, PP it is **not** optimal.

Motivation: small scale example

- In this example the optimal path is for Agent 1 to take the slightly longer route, while Agent 2 takes the much shorter one.
- **Makespan** = max length of all paths
- **Flow-time** = sum of length of all paths

- However, computing such optimal solutions for large scale environments is **not scalable**.



Methodology

- **Main idea** = learn **graph edge weights** representation such that Prioritized Planning (PP) finds solutions that are closer to optimal, while still planning in a greedy way.

Flatland input instance. Graph weights set to **1** at the beginning.

Run PP w/ input graph weights

Proposed solution (P

Differentiable loss calculation between the 2 solutions

Expert (optimal)

Back-propagate loss to update the graph **edge weights**, making PP paths more similar to optimal.

Set **updated weights** for PP

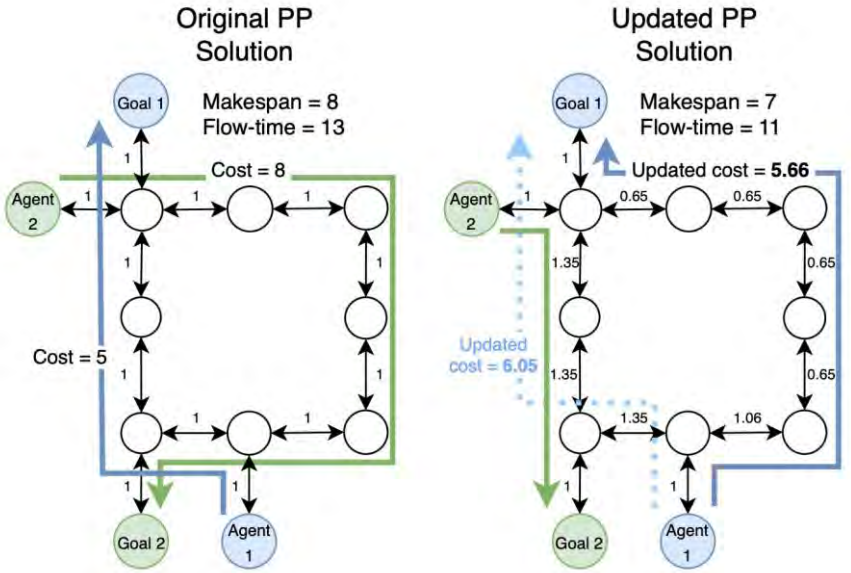
Use **Conflict Based Search** (CBS) to generate optimal paths.

Methodology

- **Conflict Based Search** (CBS) (Sharon, Guni, et al. 2015) is used to generate optimal paths.
- In order to get a meaningful gradient based on the computed loss, we use the method proposed in Vlastelica et al., 2019 which allows for the **Differentiation of Blackbox Combinatorial Solvers**.
- The main training loop is as follows:
    1. Generate optimal CBS paths.
    2. Compute initial PP paths on cost 1 graph.
    3. Calculate usage discrepancy using Hamming distance or similar metric.
    4. Update edge weights using the differentiable solver.
    5. Re-run PP with updated weights.
    6. Iterate until convergence or a fixed number of iterations.

Methodology: back to the small scale example

- Using the proposed methodology, we can learn to assign updated weights to edges in our first example.
- This way the **updated input graph** allows PP to find the optimal path, given this fixed set of priorities.
- PP still plans **greedily** in a fast way, since the input is the only change.



Overview of code structure

- **main.py** is the entry point for running experiments
- CBS and PP implementation can be found under **/solvers**
- **utils.py** provides utility functions related to graph and instance processing
- **/outputs** stores outputs:
  - For single experiments: as **images** of the agent's paths overlaid over the flatland environment
  - For multiple experiments: **3 csv files** with path length statistics for each seeded run
- **/tests** includes all tests which can all be run using:
  - python -m pytest

```
.
├── README.md
├── flatland_blackbox
│   ├── __init__.py
│   ├── compute_results.py
│   ├── models.py
│   ├── run_experiments.py
│   ├── solvers
│   │   ├── cbs.py
│   │   └── pp.py
│   ├── train.py
│   └── utils.py
├── main.py
├── outputs
├── pyproject.toml
├── requirements.txt
├── run_configs.json
└── tests
    ├── conftest.py
    ├── test_solver.py
    ├── test_train.py
    └── test_utils.py
```

Installation and running experiments

- The installation instructions can be found on the README.md
- Run single experiments using command line arguments:
  - python main.py —**mode** —**solver** [**pp** or **cbs**]
- To run a single training instance :
  - python main.py —**mode train**
- To run a set of experiments across multiple instances, over a set amount of seeds:
  - python main.py —**mode experiments**
  - The experiments parameters are stored in **run_configs.json**
- **python main.py —help** to get a list of all possible commands

```
.
├── README.md
├── flatland_blackbox
│   ├── __init__.py
│   ├── compute_results.py
│   ├── models.py
│   ├── run_experiments.py
│   ├── solvers
│   │   ├── cbs.py
│   │   └── pp.py
│   ├── train.py
│   └── utils.py
├── main.py
├── outputs
├── pyproject.toml
├── requirements.txt
├── run_configs.json
└── tests
    ├── conftest.py
    ├── test_solver.py
    ├── test_train.py
    └── test_utils.py
```

Experiments input and output

- **Input**: Flatland instances, defined by the underlying graph structure and the collection of various agent's start and goal positions.
- For the experiments the specific input is given by the parameters found inside **run_configs.json** on the right ->
- **Output**: A **Plan**. A plan is a dictionary of **Agent_ID: Path** entries. A path is a list of tuples ((row, col), timestep), representing the positions of an agent at each distinct timestep.

run_configs.json 229 B

```
{
  "iters": 300,
  "lr": 0.01,
  "lam": 3.0,
  "num_seeds": 100,
  "start_seed": 0,
  "num_agents_list": [
    3,
    7,
    11
  ],
  "max_cities_list": [
    2
  ],
  "width_list": [
    30
  ],
  "height_list": [
    30
  ]
}
```

Experiments on Flatland

- On 30x30 Flatland maps, overall **mean flow-times** over 100 seeds for each configuration:

	Mean Flow-time		
Number of agents	CBS	PP	Trained PP
3	**59.75**	60.74	60.38
7	**140.86**	144.39	143.03
11	**224.33**	230.14	227.67

Preliminary experiments on Flatland

- Filtered mean flow-times (**only** cases when flow-time of **PP > CBS**):

	Mean Flow-time		
Number of agents	CBS	PP	Trained PP
3	**56.65**	60.83	59.30
7	**137.90**	145.42	142.52
11	**214.81**	224.76	220.52

Example on 30x30 Flatland map with 4 agents

- End Stations are represented with a ▲, the numbers represents which agents has to finish there.
- Start Stations are represented with a ■, the numbers represents which agents start there.
- PP with the updated weights re-routes **Agent 0** through the middle rail, instead of the bottom one.
- **Agent 0 paths:**

Original PP → CBS → PP Trained (matches CBS Agent 0 path)

Vs

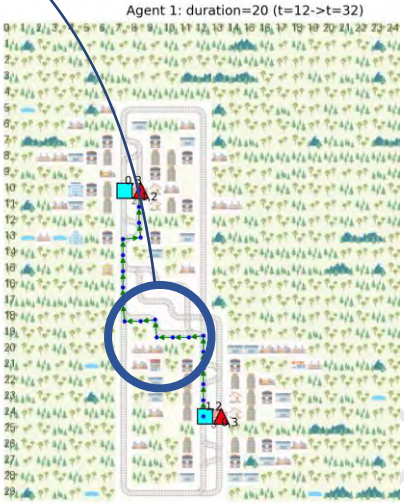
Preliminary experiments on Flatland

- So that Agent 1 can take the shorter route (**20 length**) using the **updated weights**, instead of the old PP route (**22 length**) to arrive at the goal, avoiding the conflict with Agent 0:

Original PP

V
s

PP Trained



Perspectives

- Learn based on **dynamic start/current** positions of agents. (This version of the algorithm is implemented and tested on the MAPF benchmark using a GNN as a function approximator to learn the weights).
- Extend to use flatland environment considering **breakdowns**.
- **Replan** from breakdowns using updated train position/weights (new call to function approximator)
- Use more scalable method to gather data instead of CBS
  (possible c++ implementations of algorithms as done in MAPF version of algorithm).

Authors

Authors	Institution
Herke van Hoof	UvA
Marius Captari	UvA

Link to the repository

Link to repository:
https://github.com/AI4REALNET/flatland-blackbox

Maze-Flatland

EnliteAI

The Maze-Flatland Repo

Maze-Flatland transforms the Flatland-rl environment into a **powerful AI training ground**, optimized for
AI-driven distributed decision-making research and development.

Powered by **Maze-RL**, a robust library for applied Reinforcement Learning, Maze-Flatland goes beyond the concept of a wrapper with **built-in tailored functionalities**. Significantly **improving sample efficiency**, **reducing exploration cost**, leading to lower training time and boosting agent robustness.

MAZE RL

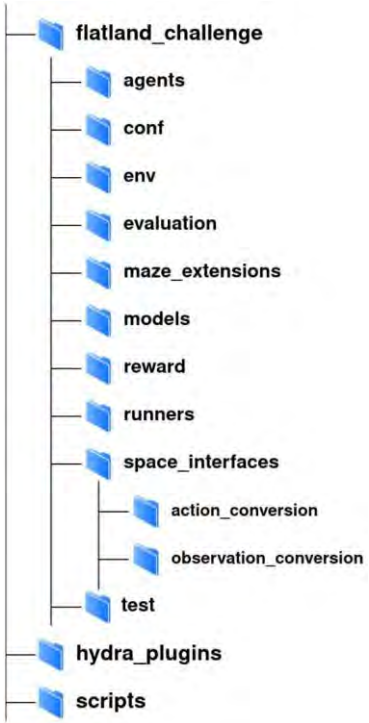
MAZE RL [GitHub - enlite-ai/maze: Maze Applied Reinforcement Learning Framework](https://github.com/flatland-association/flatland-rl)

https://github.com/flatland-association/flatland-rl

Outline

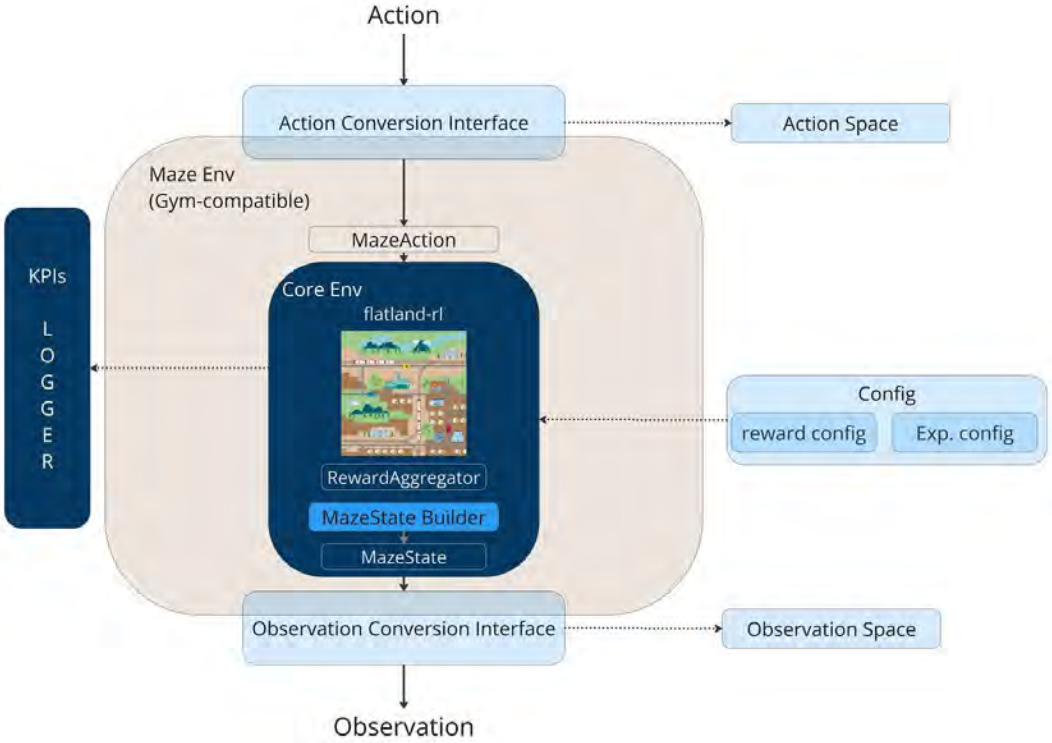
- Repository overview
- Environment architecture
- Multi-Agent formulation
- Wrappers
- Built-in masking logic
- Agent-Environment Interaction
- Maze-Flatland KPIs
- Offline Training
- Validation and Testing
- Replicate the results
- Summary

Maze-Flatland – Repository Overview



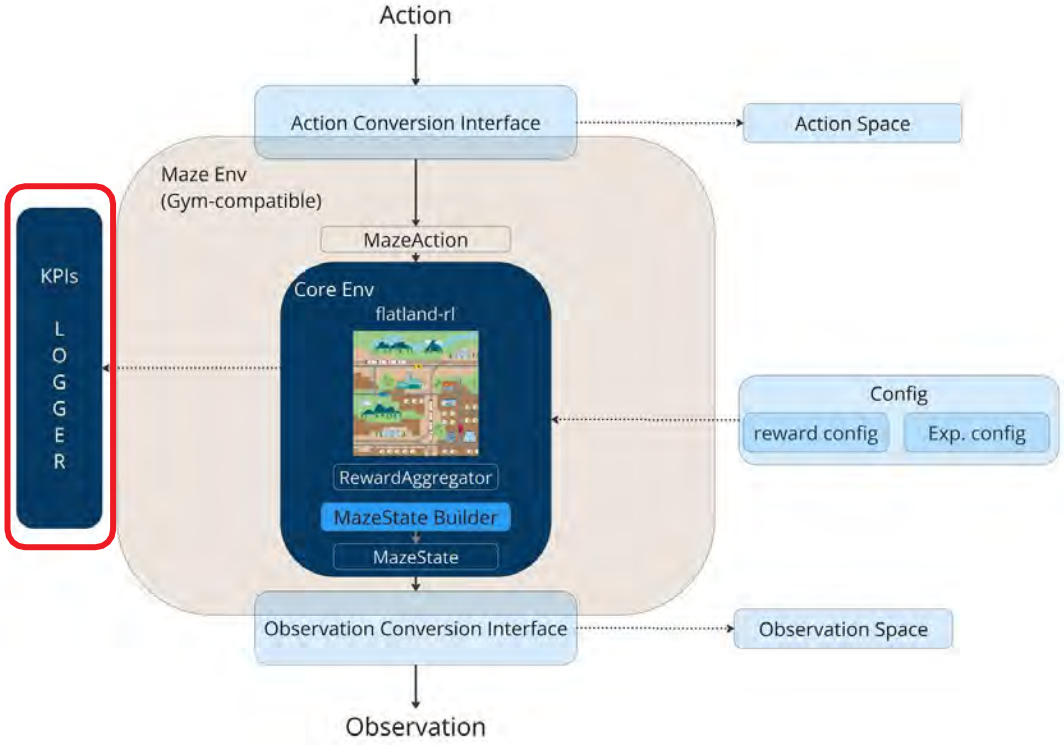
- YAML-based configuration
- Extensive logging support
- Automated testing
- Modular architecture
- Designed for scalability and future extensions

Maze-Flatland – Environment Architecture: Flow



Learn more at https://maze-rl.readthedocs.io/en/latest/environment_customization/customizing_environments.html

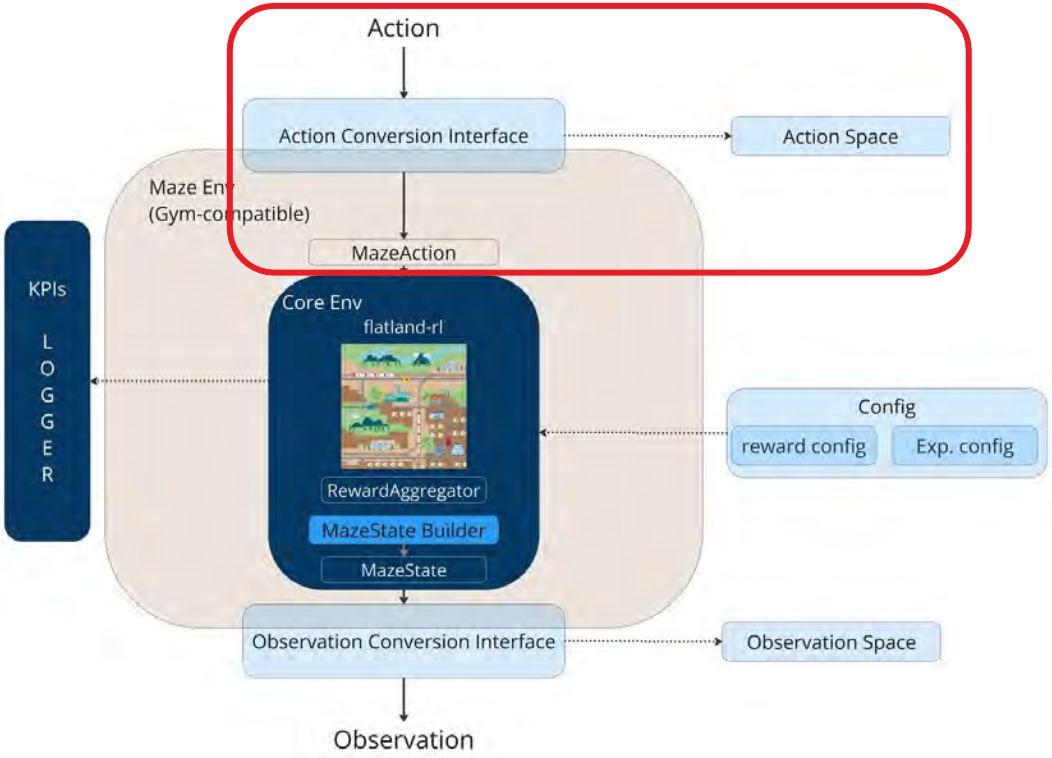
Maze-Flatland – Environment Architecture: Flow



Enables KPIs, events and statistics to be logged and aggregated per time-step, episode or epoch.

Learn more at https://maze-rl.readthedocs.io/en/latest/environment_customization/customizing_environments.html

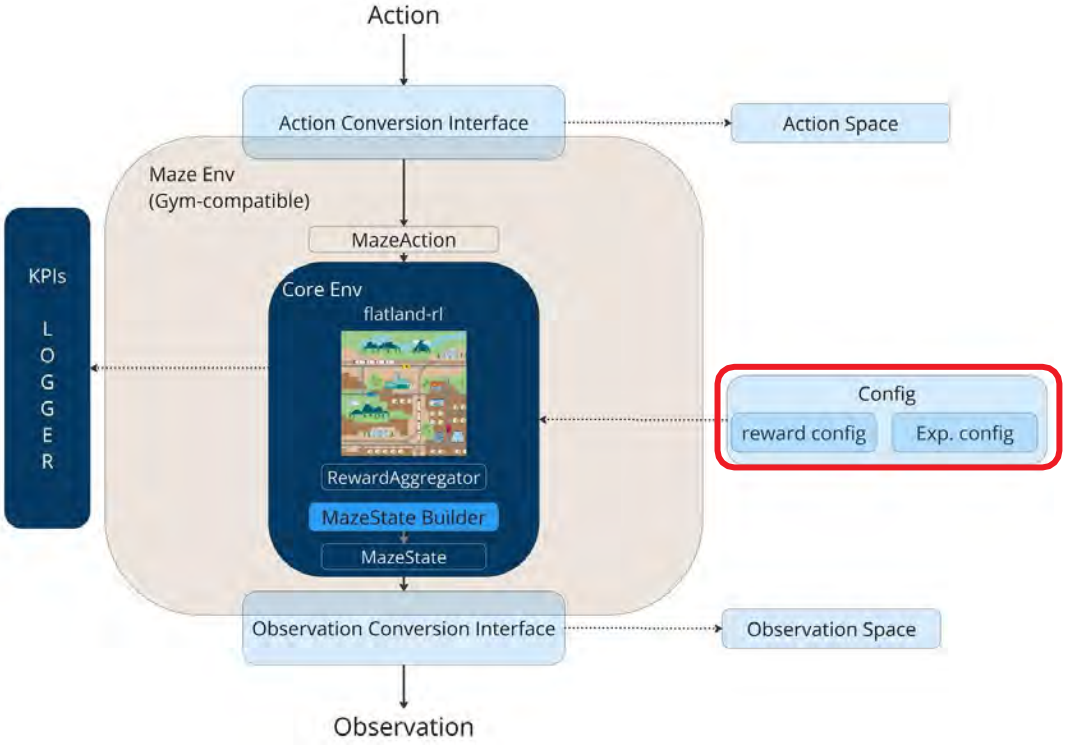
Maze-Flatland – Environment Architecture: Flow



Enables high-level action space's definition. MazeAction handles the conversion to flatland-like action.

Learn more at https://maze-rl.readthedocs.io/en/latest/environment_customization/customizing_environments.html

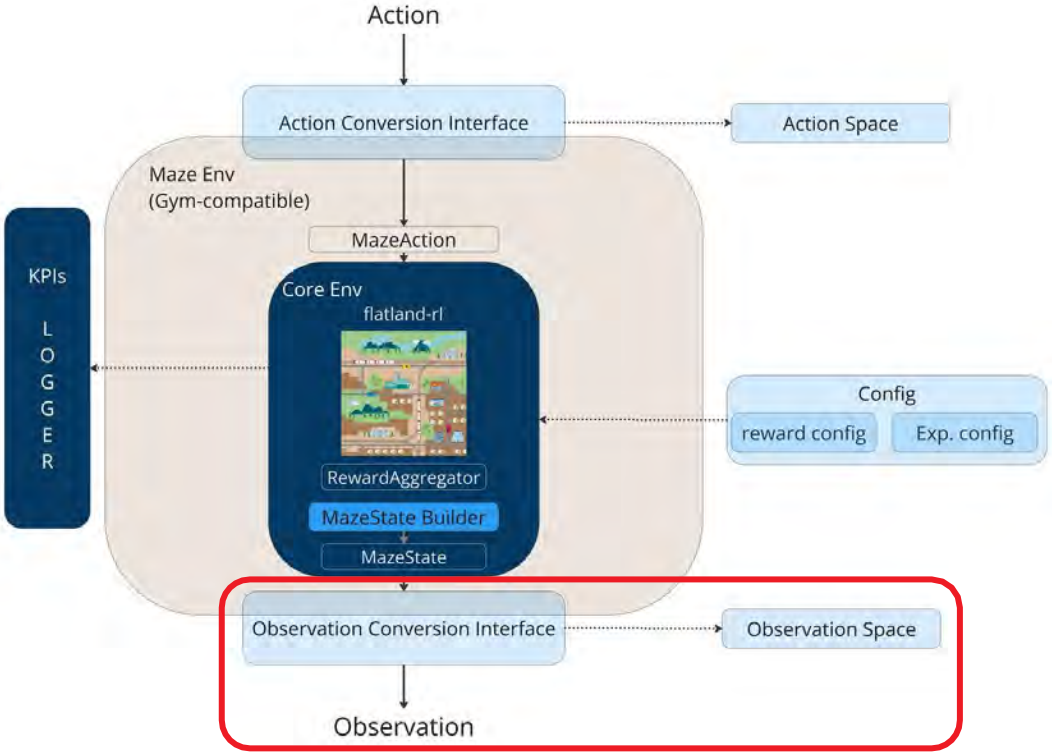
Maze-Flatland – Environment Architecture: Flow



Set rail config, #trains and reward used for experiment

Learn more at https://maze-rl.readthedocs.io/en/latest/environment_customization/customizing_environments.html

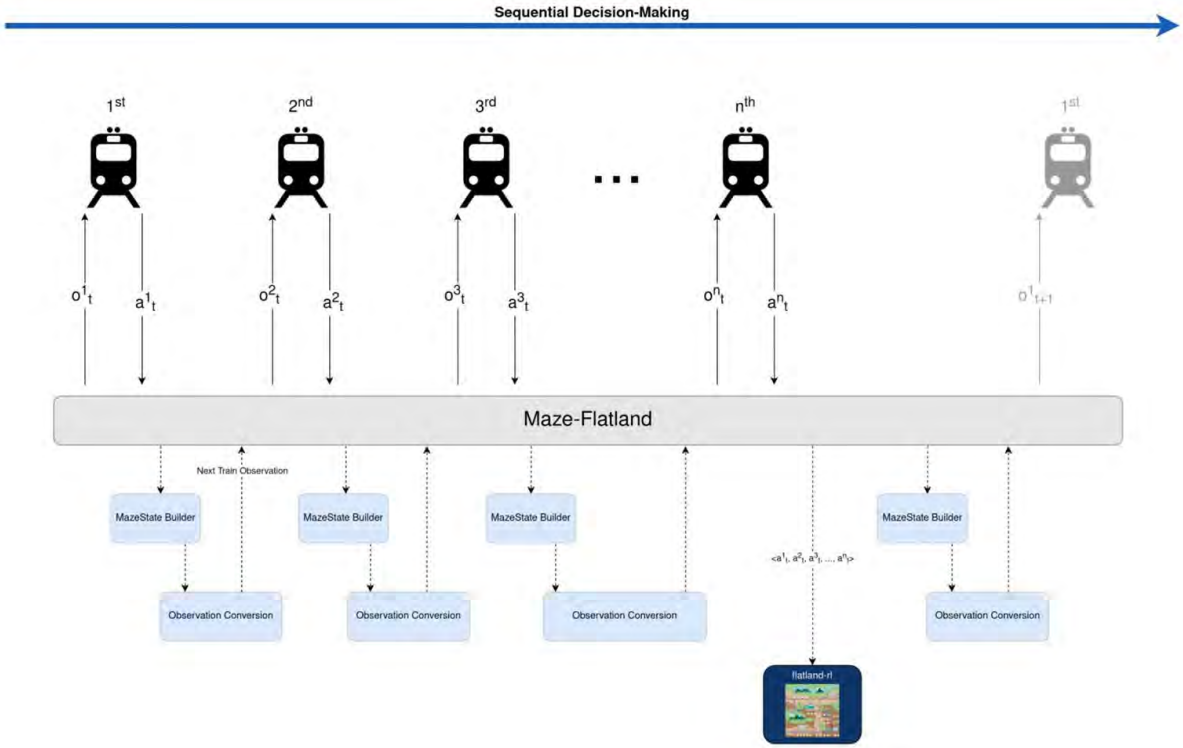
Maze-Flatland – Environment Architecture: Flow



Enables custom high-level observations

Learn more at https://maze-rl.readthedocs.io/en/latest/environment_customization/customizing_environments.html

Maze-Flatland – Multi-Agent Formulation



- MazeState is updated to point to the active train
- flatland-rl stepped, then rewards are computed
- Rewards are assigned to individual trains

Maze-Flatland Architecture – Wrappers

Masking Wrapper

- Improve sample efficiency – Agent avoids redundant actions
- Reduce exploration complexity – Lesser choice to choose from and each of these have

an unique outcome

Skipping Wrapper

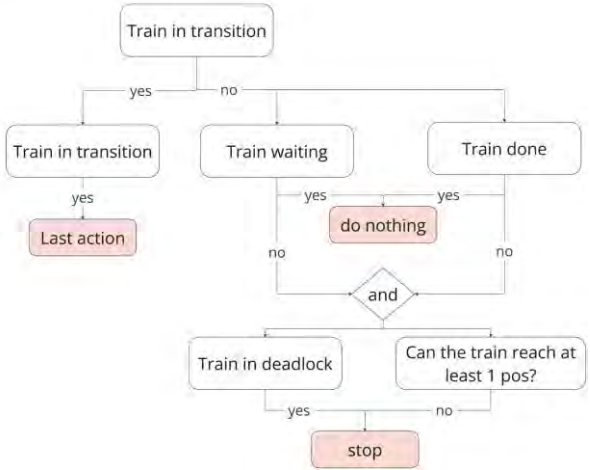
- Prevent unnecessary decision-making – when actions lead to the same outcome
- Improve sample efficiency – Agent is trained only on meaningful state transitions.
- Leading to faster learning and more robust policy

Rendering Wrapper

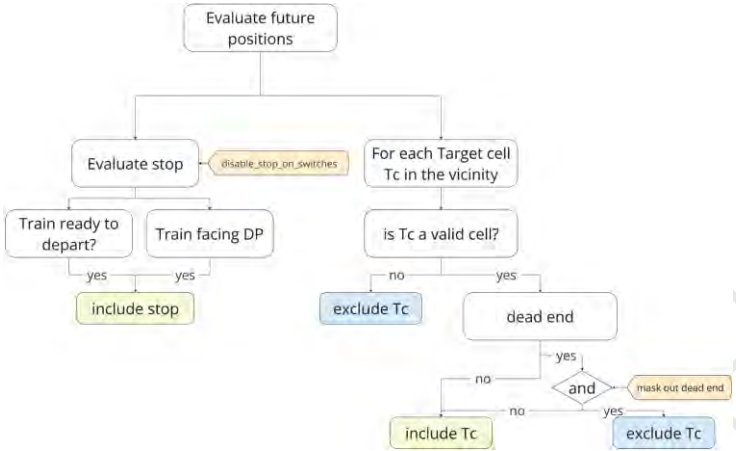
- Support custom visualization – allowing for different level of details

Masking Wrapper – Built-in Masking Logic

Phase 1 – single option



Phase 2 – multi-option decision



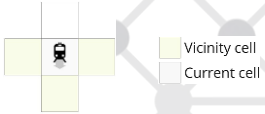
Decision Point (DP): A switch connected to a cell that cannot be reached from given the current train direction.

Terminal state

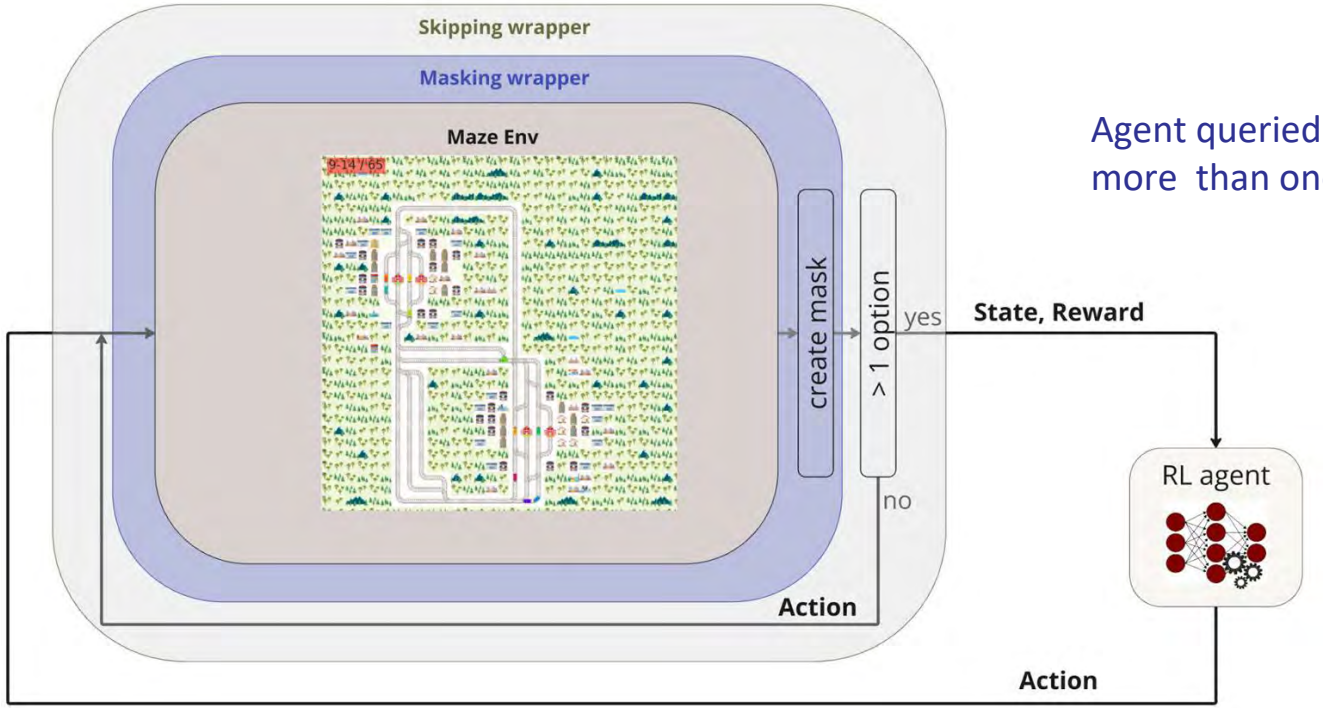
Flag used in masking to change the behavior

Non Terminal State – not included in the masking

Terminal State included in the masking



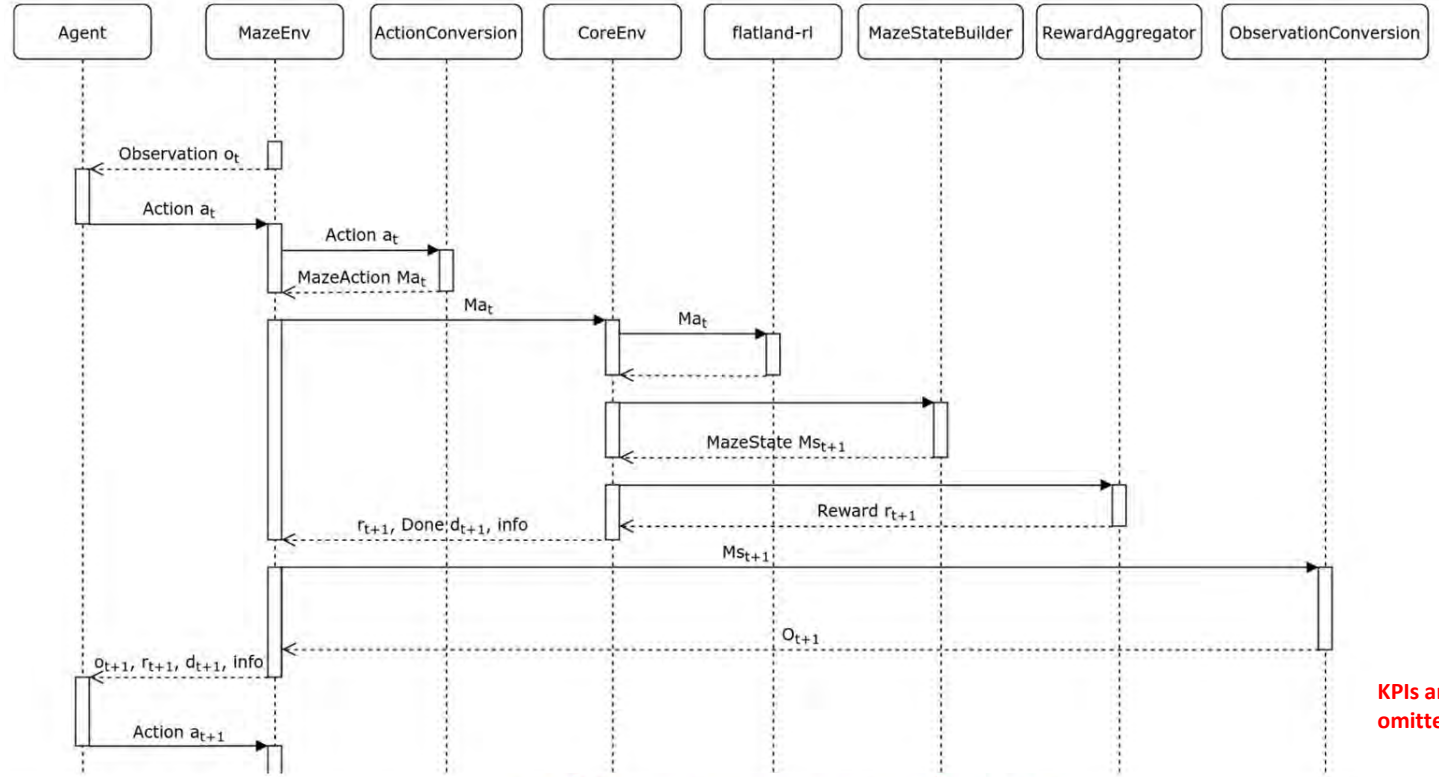
Maze-Flatland – Agent-Env Interaction



Agent queried only when there is more than one possible option

Learn more at https://maze-rl.readthedocs.io/en/latest/environment_customization/customizing_environments.html

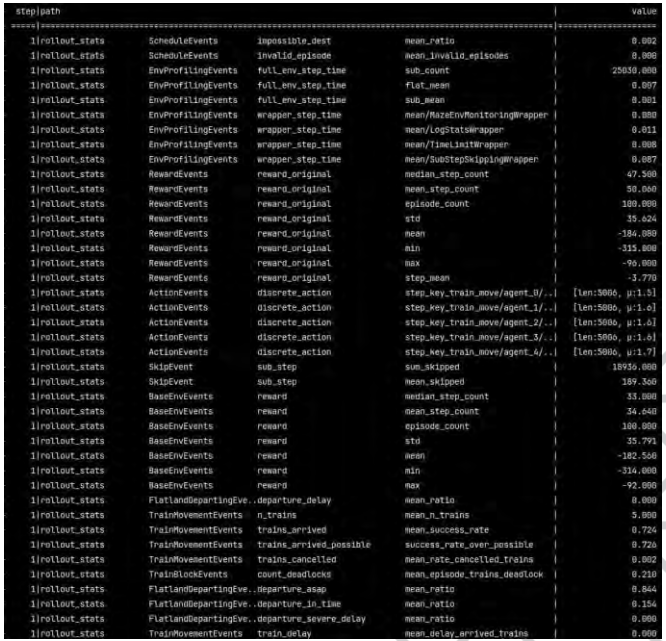
Agent-Env Interaction – Sequence Diagram



KPIs and wrappers are omitted

Maze-Flatland KPIs – Overview

- Printed at console
- Collected and stored to csv file
- Supports tensorboard



step	path				value
1	rollout_stats	ScheduleEvents	impossible_dest	mean_ratio	0.002
1	rollout_stats	ScheduleEvents	invalid_episode	mean_invalid_episodes	0.000
1	rollout_stats	EnvProfilingEvents	full_env_step_time	sub_count	25050.000
1	rollout_stats	EnvProfilingEvents	full_env_step_time	flat_mean	0.007
1	rollout_stats	EnvProfilingEvents	full_env_step_time	sub_mean	0.001
1	rollout_stats	EnvProfilingEvents	wrapper_step_time	mean/MazeEnvMonitoringWrapper	0.000
1	rollout_stats	EnvProfilingEvents	wrapper_step_time	mean/LogStatsWrapper	0.011
1	rollout_stats	EnvProfilingEvents	wrapper_step_time	mean/TimeLimitWrapper	0.008
1	rollout_stats	EnvProfilingEvents	wrapper_step_time	mean/SubStepSkippingWrapper	0.007
1	rollout_stats	RewardEvents	reward_original	median_step_count	47.500
1	rollout_stats	RewardEvents	reward_original	mean_step_count	50.060
1	rollout_stats	RewardEvents	reward_original	episode_count	100.000
1	rollout_stats	RewardEvents	reward_original	std	35.624
1	rollout_stats	RewardEvents	reward_original	mean	-184.080
1	rollout_stats	RewardEvents	reward_original	min	-315.000
1	rollout_stats	RewardEvents	reward_original	max	-96.000
1	rollout_stats	RewardEvents	reward_original	step_mean	-3.770
1	rollout_stats	ActionEvents	discrete_action	step_key_train_move/agent_0/..	[len:5000, μ:1.5]
1	rollout_stats	ActionEvents	discrete_action	step_key_train_move/agent_1/..	[len:5000, μ:1.6]
1	rollout_stats	ActionEvents	discrete_action	step_key_train_move/agent_2/..	[len:5000, μ:1.5]
1	rollout_stats	ActionEvents	discrete_action	step_key_train_move/agent_3/..	[len:5000, μ:1.6]
1	rollout_stats	ActionEvents	discrete_action	step_key_train_move/agent_4/..	[len:5000, μ:1.7]
1	rollout_stats	SkipEvent	sub_step	sum_skipped	18950.000
1	rollout_stats	SkipEvent	sub_step	mean_skipped	189.360
1	rollout_stats	BaseEnvEvents	reward	median_step_count	33.000
1	rollout_stats	BaseEnvEvents	reward	mean_step_count	34.640
1	rollout_stats	BaseEnvEvents	reward	episode_count	100.000
1	rollout_stats	BaseEnvEvents	reward	std	35.791
1	rollout_stats	BaseEnvEvents	reward	mean	-102.560
1	rollout_stats	BaseEnvEvents	reward	min	-314.000
1	rollout_stats	BaseEnvEvents	reward	max	-92.000
1	rollout_stats	FlatlandDepartingEve..	departure_delay	mean_ratio	0.000
1	rollout_stats	TrainMovementEvents	n_trains	mean_n_trains	5.000
1	rollout_stats	TrainMovementEvents	trains_arrived	mean_success_rate	0.724
1	rollout_stats	TrainMovementEvents	trains_arrived_possible	success_rate_over_possible	0.726
1	rollout_stats	TrainMovementEvents	trains_cancelled	mean_rate_cancelled_trains	0.002
1	rollout_stats	TrainBlockEvents	count_deadlocks	mean_episode_trains_deadlock	0.210
1	rollout_stats	FlatlandDepartingEve..	departure_asap	mean_ratio	0.844
1	rollout_stats	FlatlandDepartingEve..	departure_in_time	mean_ratio	0.154
1	rollout_stats	FlatlandDepartingEve..	departure_severe_delay	mean_ratio	0.000
1	rollout_stats	TrainMovementEvents	train_delay	mean_delay_arrived_trains	0.000

KPIs Categories

- Runtime profiling → Time spent in each part of the system
- Reward and action logging
- Domain-specific events

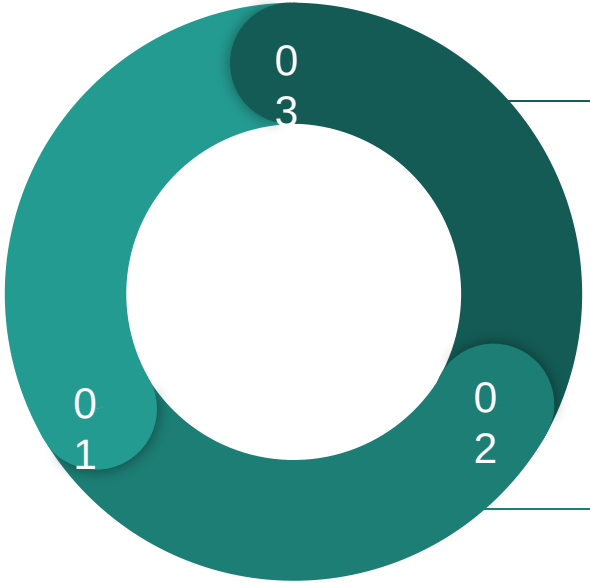
Can be extended!!!

Learn more at https://maze-rl.readthedocs.io/en/latest/getting_started/maze_env/event_system.html

Maze-Flatland KPIs – Domain-Specific Events

Event Name	Range	Description
Departure Delay	[0, ∞]	Average delay recorded for departure
Departure in Time	[0, 100%]	Trains departing on time to arrive within the scheduled time window
Arrival Ratio	[0, 100%]	Trains arrived at destination
Arrival Delay	[0, ∞]	Average delay for trains arrived at destination
Unsolvable Ratio	[0, 100%]	Trains without a valid path from departure to destination
Cancelled Trains	[0, 100%]	Trains that never departed
Deadlock Ratio	[0, 100%]	Trains on the rails unable to move due to blocked paths

Maze-Flatland Training – Offline RL



Create dataset

Use a heuristic or a policy to collect the trajectories

Offline training

Use the provided tool to learn from the collected experience.

Preprocessing

Apply a pre-processing filter when loading the stored trajectories

Maze-Flatland Validation and Testing

Simple rollout

- Focus on challenging episodes
- Customisable environment configuration

Competition validation

- Play fixed competition-like scenarios for round 1 and 2

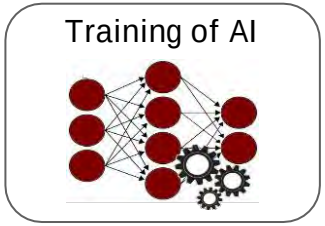
For more information visit:
https://flatland.aicrowd.com/challenges/neurips2020/envconfig.html

Maze-Flatland Training – Replicate the results

Offline dataset

Config file

Input



Output

Maze-Flatland Training – Replicate the results

1) Pre-step

Install maze-**flatland**

```
conda env create --file environment.yml
conda activate maze-flatland
pip install -e .
pip install git+https://github.com/enlite-ai/maze.git@dev
```

Download the dataset

Download and extract
https://drive.google.com/file/d/1FW6FnAKHqXXu_LDbdeWQR32jtTQeFc5o

2) Imitation Learning

Neural Network - run the following command

```
maze-run -cn conf_train +experiment=offline/train/bc_train
trajectories_data=</path/to/dataset>
```

XGBoost - run the following command

```
maze-run -cn conf_train +experiment=offline/train/xgboost_train
trajectories_data=</path/to/dataset/>
```

3) Validation

Neural Network - run the following command

```
maze-run +experiment=multi_train/rollout/validation_torch_policy
input_dir=<path/to/policy>
```

XGBoost - run the following command

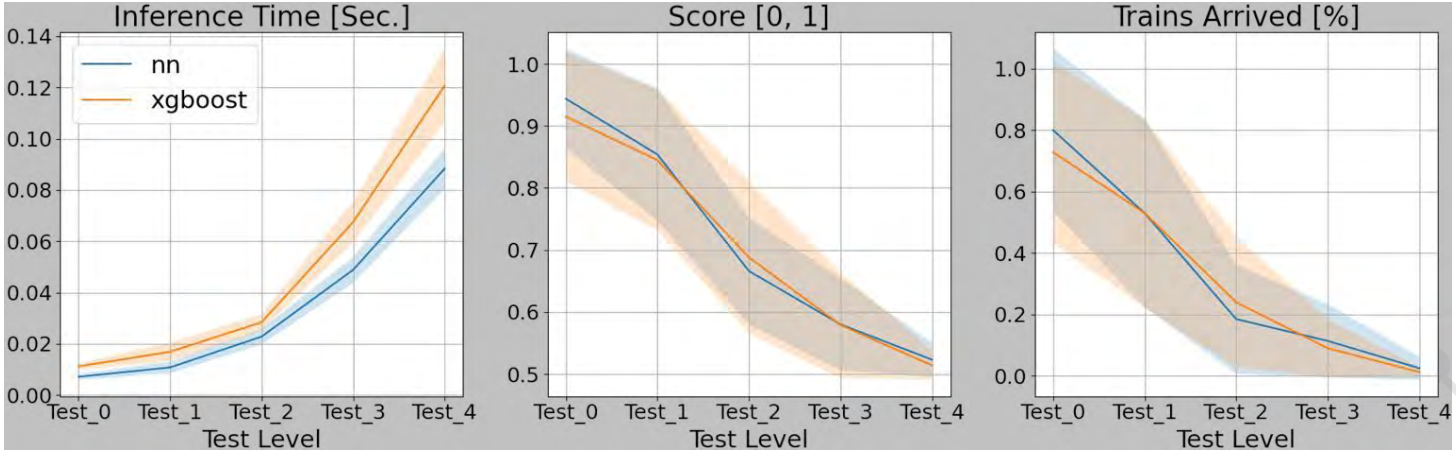
```
maze-run +experiment=multi_train/rollout/validation_xgboost
input_dir=<path/to/policy>
```

4) Compare

Analyse the *validation_stats.csv* files in the output directories

Repository link: https://gitlab.inesctec.pt/cpes/european-projects/ai4realnet/enliteai/beta_release/maze-flatland

Performance Comparison – NN vs XGBoost



Env setup for each level

	#agents	Map size	n_cities	Max Rail pairs in city	Max rails between cities	Malfunction rate	Malfunction interval
Test_0	7	30x30	2	2	2	1/540	[20, 50]
Test_1	10	30x30	2	2	2	1/900	[20, 50]
Test_2	20	30x30	3	2	2	1/1800	[20, 50]
Test_3	50	30x35	3	2	2	1/4500	[20, 50]
Test_4	80	35x30	5	2	2	1/7200	[20, 50]

Summary

- Improved environment engineering to **enhance AI-based agent development**;
- Provide built-in high-level customization: reward, observation and action;
- Integrated multiple KPIs for analysis
- Validation setup

	Multi-agent support	Hide illegal actions	Improved sample efficiency	Support custom observation	Support custom actions	KPIs & event support	Support custom reward formulations
maze-flatland	✔	✔	✔	✔	✔	✔	✔
flatland-rl	✗	✗	✗	✔	✗	✗	✗

Link to the repository

Link to repository:
https://github.com/AI4REALNET/maze-flatland

Authors

Authors	Institution
Anton Fuxjaeger	EnliteAI
Alberto Castagna	EnliteAI

State and action factorization

Politecnico di Milano (POLIMI)

Outline

- Context
- Methodology
- Original Contribution
- Overview of repo structure
- Algorithm
- Experiments

Context

Definition

Distributed Reinforcement Learning (DRL) is a distributed learning process to solve a sequential decision-making problem

Motivation

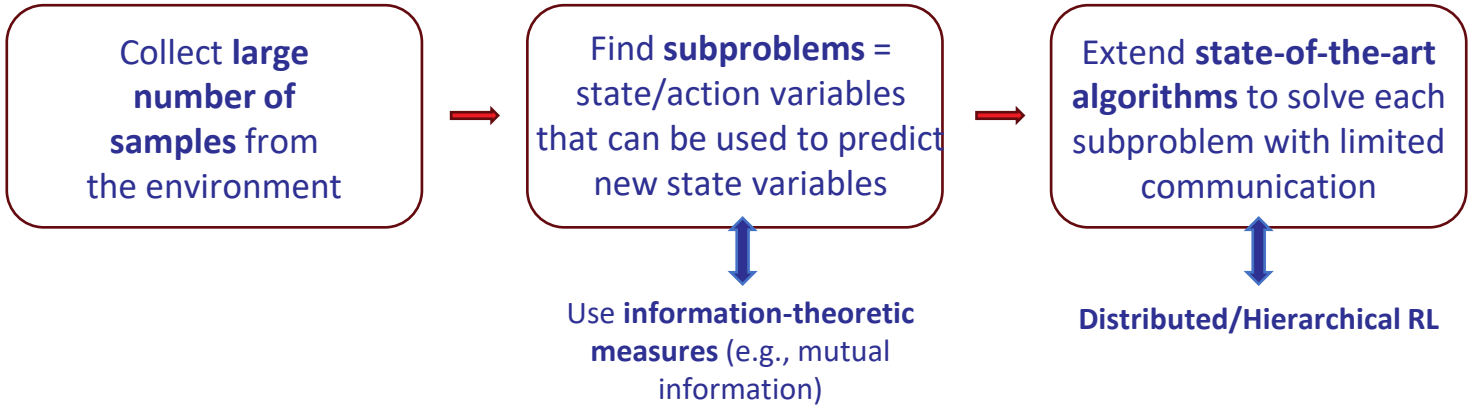
Large-scale networks are characterized by a large number of state and action variables that make the standard RL algorithms impractical/inefficient (***curse of dimensionality***)

Use cases

Power grids, railway networks and air traffic networks can be operated with **distributed agents** that observe and control small portions of the environment and cooperate to achieve a shared objective, e.g., prevent blackouts, avoid traffic congestions

Methodology

- **Main idea** = Identify in a *data-driven way the decentralized decomposition* of the problem that minimizes the introduced bias



- **Preliminary experiments** on Flatland

Original contribution

- **Dynamic State and Action factorization**
  - ➢ **Short description:** Creation of smaller Markov Decision Problems starting from a single MDP
  - ➢ **State-of-the-art:** Distributed control theory[1], power grid segmentation[2]
  - ➢ **Contribution:** First data-driven method for the factorization of control problems
  - ➢ **Implemented WP2 features:** Distributed RL

[1] Lunze, Feedback Control of Large-Scale Systems (1992)

[2] Marot et. al., Guided machine learning for power grid segmentation (2018)

State-action clustering

1. Collect a dataset of transitions from the original MDP $\longrightarrow$ $\mathcal{D} = \left\{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_t\right\}_{t=1}^{T}$

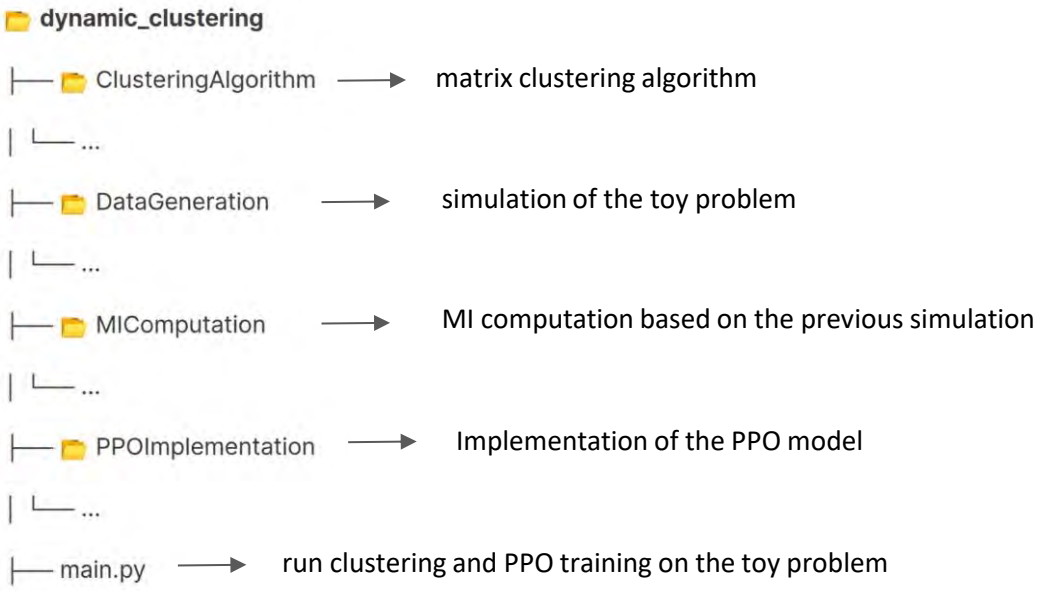
2. Compute the matrix of Mutual Information (MI) between pair of variables

3. Cluster rows and columns in a graph representation that allows clusters overlapping

4. Evaluate if it is beneficial to merge clusters using an evaluation function

$$S = \sum_{i,j} \left[\alpha(1 - M_{ij})\mathbf{1}(M_{ij} < t)\left(C_1(i,j) + C_2(i,j)\right) + (1-\alpha)M_{ij}\mathbf{1}(M_{ij} \geq t)\, O(i,j)\right]$$

Overview of code structure



Input data

- Desired data configuration switching between Flatland and custom toy problem
- Alpha parameter for evaluation of cluster merges
- Threshold for Mutual Information matrix

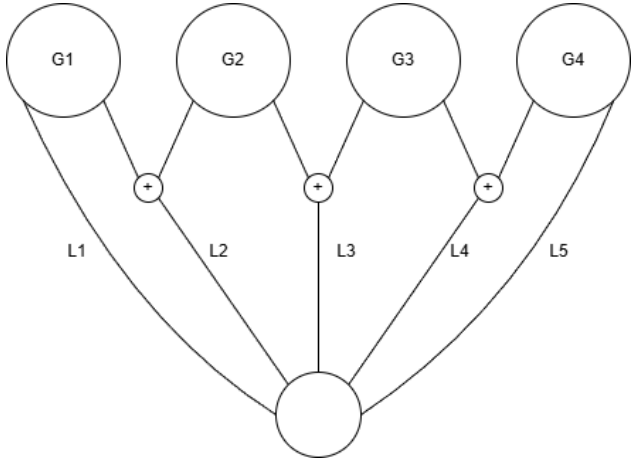
Output data

- Cluster sets for the Mutual Information matrix rows and columns

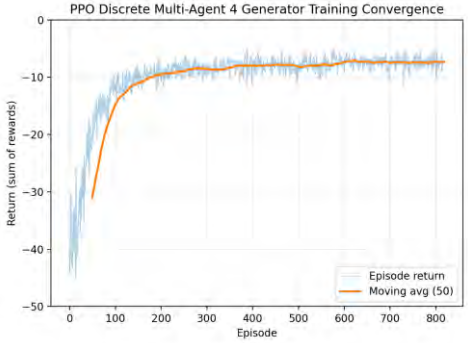
- Plot of the graph with the different sets

Experiment

- Tested on a custom toy problem with variable number of generators and power lines



- Noticed a decrease in training time when considering scenarios with 4 or more generators



Link to the repository

https://github.com/AI4REALNET/dynamic-state-action-factorization

Authors

Authors	Institution
Gianvito Losapio	POLIMI
Andrea Fondacaro	POLIMI
Marco Mussi	POLIMI
Alberto Maria Metelli	POLIMI
Marcello Restelli	POLIMI

Network-distributed Q-learning

Politecnico di Milano (POLIMI)

Outline

- Context
- Methodology
- Original Contribution
- Overview of repo structure
- Algorithm
- Experiments

Context

Definition

Distributed Reinforcement Learning (DRL) is a distributed learning process to solve a sequential decision-making problem

Motivation

Large-scale networks are characterized by a large number of state and action variables that make the standard RL algorithms impractical/inefficient (***curse of dimensionality***)

Use cases

Power grids, railway networks and air traffic networks can be operated with **distributed agents** that observe and control small portions of the environment and cooperate to achieve a shared objective, e.g., prevent blackouts, avoid traffic congestions

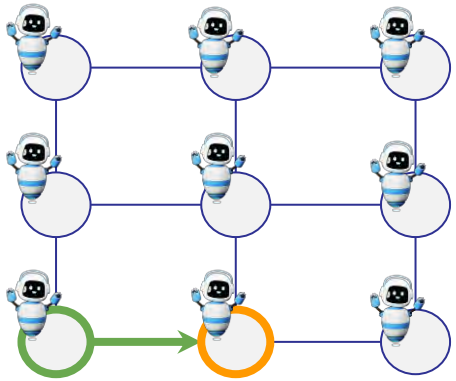
Original contribution

- **Network-distributed Q-learning**
  - **Short description:** Distributed version of the standard Q-learning algorithm
  - **State-of-the-art:** Other distributed RL algorithms[1]
  - **Contribution:** Proposed specific implementation for turn-based MDPs
  - **Implemented WP2 features:** Distributed RL

[1] Kar et. al., QD-Learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement Learning Through Consensus + Innovations (2012)

Distributed Q-learning

1. Each agent controlling a node



2. Information flowing from the next agent to the previous agent

from successor node

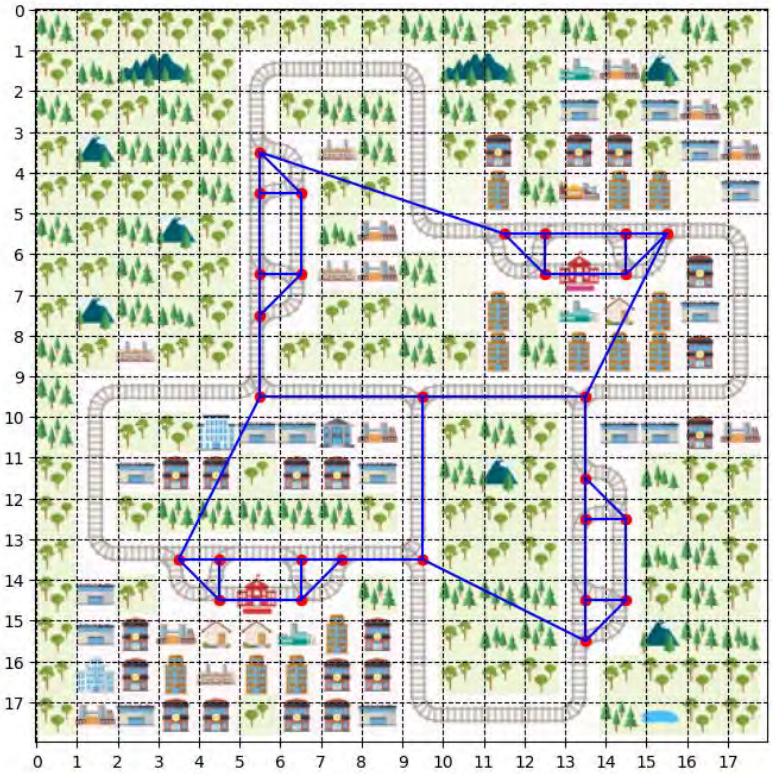
Q-learning update $$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

modified from the original Q-learning implementation [Watkins, 1992]

SwitchFL

One **agent** for each **switch** (junction node) instead of one agent for each train
(as in the original Flatland implementation)

In the figure switches are represented in red and their connections are represented in **blue**



Overview of code structure

- **network-distributed q-learning**
  - flatland patch → contains a patch to Flatland
    - ...
  - switchfl → contains the implementation of switchfl and the network-distributed Q-learning algorithm
    - ...
  - eval.py → script for evaluating a model
  - hyperparam_tuning.py → script for running multiple experiments in parallel
  - main.py → script for a single experiment from a config file
  - plot.ipynb → auxiliary plot functions
  - test_model.py → run a single experiment

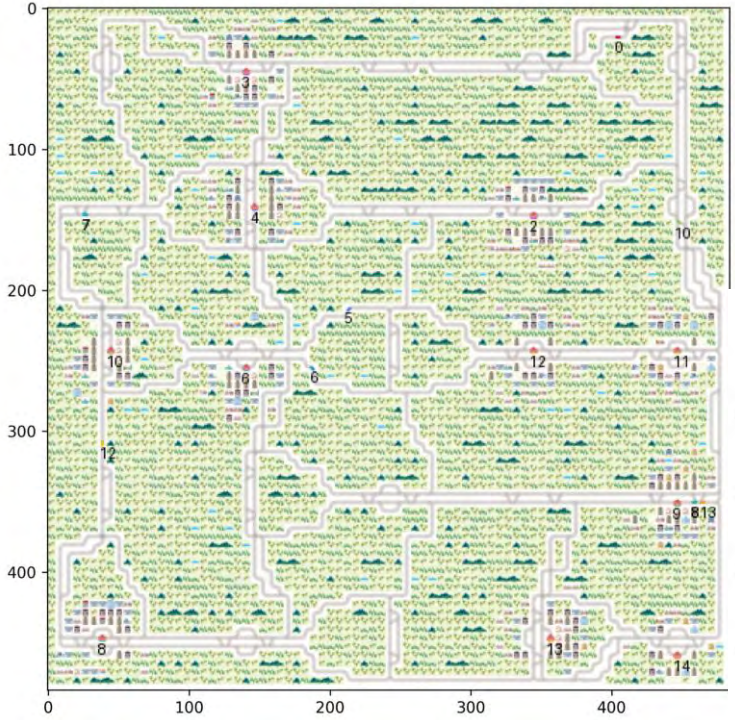
Input data

- Parameters of the Flatland environment (grid size, number of trains, …)
- Hyperparameters of the Network-distributed Q-learning algorithm (learning rate, epsilon decay, …)

Output data

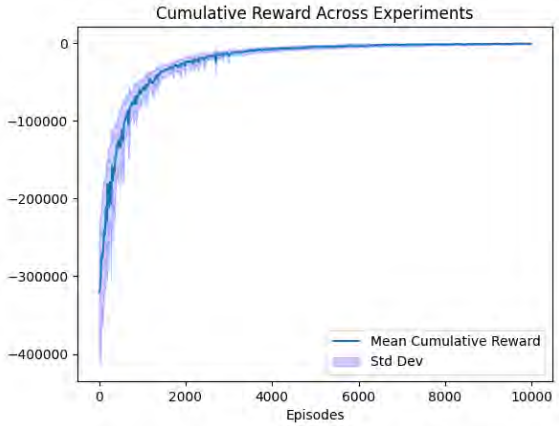
- one folder for each hyperparameter configuration, containing the saved models, the parameters of the experiments, the cumulative rewards obtained during the training phase, the number of trains arrived at destinations and the overall delays.
- a config file for each experiment (so that it can be reproduced) and the full hyperparam_tuning.py script to repeat the whole experiment

Experiments



286 switch agents

- Tested on Flatland grid 80x80 with **15 trains**, 40 stations
- Convergence to optimal solution, i.e., almost all the trains arrive at destination with no delay.



TRAINING

EVALUATION

Authors

Authors	Institution
Gianvito Losapio	POLIMI
Marco Mussi	POLIMI
Alberto Maria Metelli	POLIMI
Marcello Restelli	POLIMI

Link to the repository

https://github.com/AI4REALNET/network-distributed-q-learning

Integration of Communication in MARL

ZHAW

Context

Definition

Multi-agent reinforcement learning (MARL) focuses on studying the behavior of multiple learning agents that coexist in a shared environment. Each agent is motivated by its own rewards, and takes actions to advance its own interests.

Motivation

In many real-world applications, decision-making is inherently multi-agent, decentralized, and subject to strong interdependencies between agents. Introducing of communication enables effective and safe coordination among decentralized agents, while maintaining scalability, interpretability, and alignment with trustworthiness requirements.

Use cases

This implementation is specific to Railway network domain and simulation environment Flatland.

Methodology - baselines

- **Duelling Double Deep Q-Network (DDDQN)**

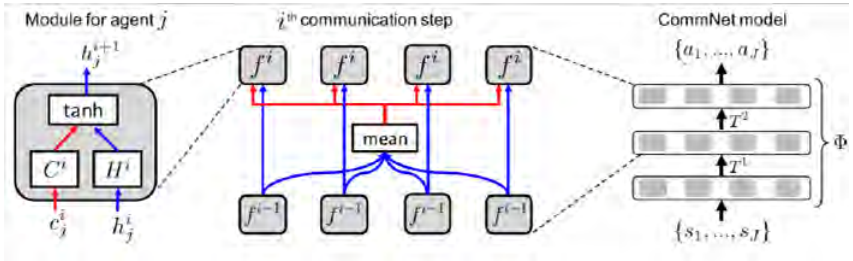
Combines ideas and benefits from two foundational deep reinforcement learning approaches: Double Q-Learning and Dueling Network Architectures. It uses the dueling architecture to learn better value estimations, and applies double Q-learning updates to reduce overestimation bias. The result is a more stable and accurate value-based RL algorithm.

- **Proximal Policy Optimization (PPO)**

Belongs to the class of actor-critic methods. PPO maintains two neural networks: the actor, which parameterizes the policy, and the critic, which estimates the value function. The agent interacts with the environment to collect trajectories, and then uses these trajectories to compute advantage estimates, which are used to update the policy in a direction that increases expected return. PPO alternates between sampling new trajectories from the environment and performing multiple epochs of mini-batch stochastic gradient ascent on the collected data.

Methodology: CommNet

The CommNet model consists of multiple stacked layers, each performing a communication step followed by local updates using shared parameters. This structure allows each agent to condition its action not only on its own observation, but also on a learned, global context synthesized from the collective agent states.



Benefits of CommNet integration for the baselines[1]:

- Facilitation of explicit message passing among agents.
- Improved coordination under partial observability.
- Minimal architectural footprint, requires only a small change in the network.
- Systematic evaluation of communication dynamics in Flatland3.

Limitations:

- Non-symbolic messages.

1. Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communicationwith backpropagation. Advances in Neural Information Processing Systems 29, pages 2244–2252, dec 2016

Original contribution

Algorithm | Inter-agent communication within a MARL framework

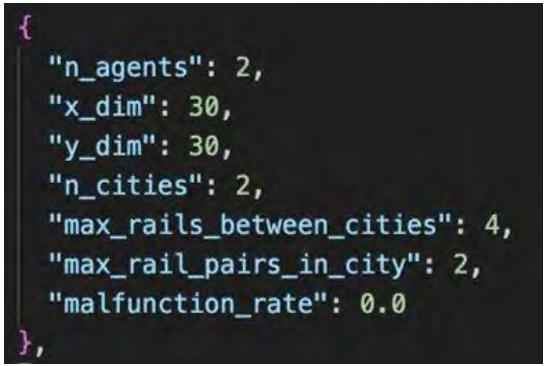
- **Short description:** Implementation of a communication network (CommNet) module for inter-agent communication within a MARL framework for two base policies, PPO and DDDQN.
- **State-of-the-art:** Multiple agents are sharing the environment without communication and alignment of their actions.
- **Implemented WP2 feature:** communication between the agents.
- **Contribution:** modular and extensible module for integration of communication among agents.

Simulation environment

Flatland models complex railway traffic as a multi-agent pathfinding and coordination problem, where hundreds of train agents must navigate to their destinations efficiently while handling bottlenecks and avoiding deadlocks.

Used environment properties:

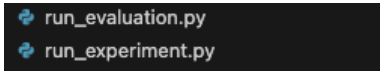
- grid size of 30 × 30 cells
- 2 agents
- 2 city nodes connected by 2 inter-city rail segments
- each city contains two intra-city rail segments.



AIcrowd. Flatland 3 Migration Guide. [Online]. 2024. url: https://flatland.aicrowd.com/challenges/flatland3/flatland-3-migration-guide.html (visited on 03/15/2025)

Overview of code structure

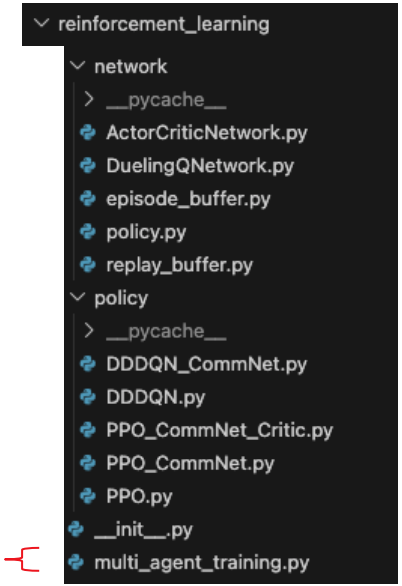
Configuration files to select environment, hyperparameters and training parameters.



A quickstart script to run tests, training and evaluation of selected models and seeds.

- python run_experiment.py
- python run_evaluation.py

Change config files to run your custom experiments.



Component / helper scripts for certain policies.

Policies with and without commnet integration.

Core script that handles initialization, loop, network update and logs.

Configuration files

These files are predefined and gets loaded before a training starts, to set the wanted parameters and environment settings.

- Parameters of the Flatland environment (grid size, number of trains, …) are provided by the **default_env_config.json** file.

- Hyperparameters of the policies (learning rate, epsilon decay, …) are provided by the **experiment_config.json file.**

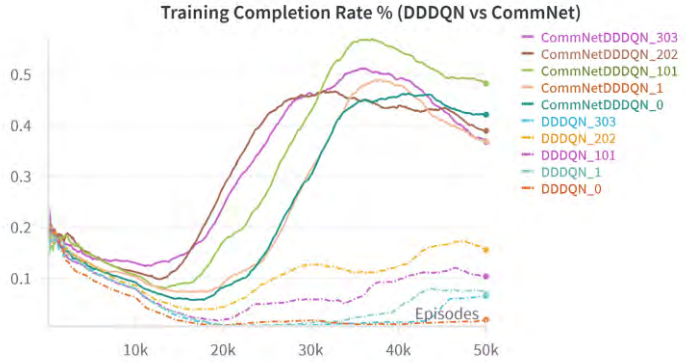
These files can be found in the **flatland_base/configurations** folder.

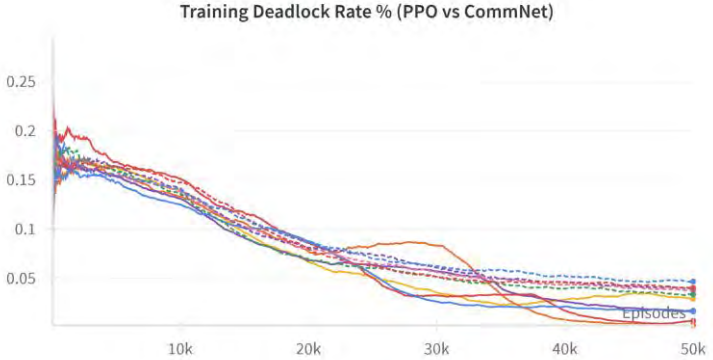

```
{
  "n_episodes": 50000,
  "n_evaluation_episodes": 10,
  "training_env_config": 1,
  "evaluation_env_config": 1,
  "checkpoint_interval": 10000,
  "eps_start": 1.0,
  "eps_end": 0.01,
  "eps_decay": 0.99975,
  "buffer_size": 32000,
  "buffer_min_size": 0,
  "restore_replay_buffer": "",
  "save_replay_buffer": false,
  "batch_size": 128,
  "gamma": 0.99,
  "tau": 0.0005,
  "learning_rate": 0.00005,
  "hidden_size": 128,
  "update_every": 200,
  "use_gpu": false,
  "num_threads": 4,
  "K_epoch": 10,
  "policy": "PPO",
  "load_policy": null,
  "action_size": "full",
  "eval_mode": false,
  "use_observation": "FastTreeObs",
  "max_depth": 2,
  "observation_radius": 10,
  "observation_max_path_depth": 30,
  "render": false,
  "eval_render": false,
  "render_deadlocked": false,
  "no_regeneration": false,
  "n_agent_fixed": true,
  "n_agent_iterate": false,
  "skip_unfinished_agent": 9999.0,
  "disable_wandb": true,
  "wandb_project_name": "commnet-fl",
  "seed": null
}
```

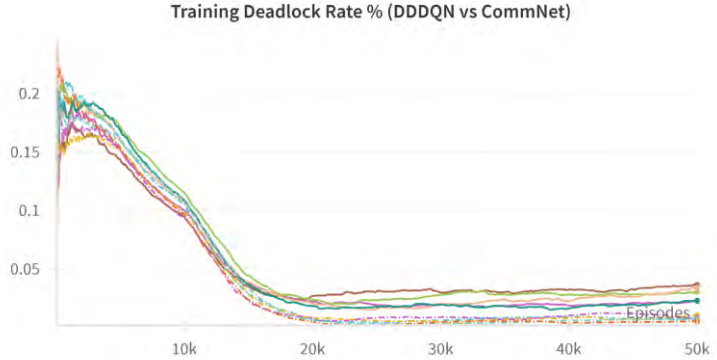
Output

- A detailed log of arrived trains, time arrival, deadlocked trains, cumulative rewards will be tracked during the evaluation phase in the training and are accessible.
- Hyperparameters, policy, checkpoints will be available after running the training.
- (Optionally) If enabled, in a form of wandb format, visualized in charts.

Performance Comparison



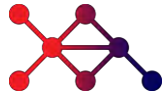




Perspectives

- Future work:
  - The scalibility of CommNet is a big bottleneck, as the fully connected message is computationaly expensive.
  - Other communication methods might be of interest like CACOM, as it learns a context based messaging system, resulting in sending only the necessary impactful messages.
  - Reward engineering, to improve the cooperation by communicating.

Authors



Authors	Institution
Ricardo Chavarriaga	ZHAW CAI
Anna Fedorova	ZHAW CAI
Minh-Khan Nguyen	ZHAW
Andres Mock	ZHAW

Link to the repository

https://github.com/AI4REALNET/flatland-commnet

Explainable and Transparent Failure Prediction of Agents

Fraunhofer/UKASSEL

Context

Definition

Failure prediction is proactive forecasting of imminent power grid disruptions based on the observed state of the grid and the behavior of the employed DRL agents.

Motivation

Despite the abundance of DRL agents proposed for power grid topology optimization, most studies concentrate on performance metrics like survival scores without investigating why these agents fail. Identifying failure types and foreceasting failures enable awareness for timely human interventions.

Use cases

Power grids, railway networks and air traffic networks agents can be anaylsed with regard to failure prediction as well as analysing their behavior and clustering common failure types.

Methodology

- **Main idea** = Understanding agents' failures by interpreting failure types and developing a multi-class forecasting framework that predicts imminent grid disruptions, providing actionable insights for enhancing grid stability.

Compile a dataset of failure events from 3 DRL agents on a IEEE118 grid → Forecasting of agent failure based on the current grid state using multiple ML models → Extracting feature importance values to highlight important input features and critical grid regions

- **Extensive experiments** for the power grid application using grid2op

Original contribution

- **Algorithm |** Failure Prediction

  - **Short description:** Forecasting of grid failures under DRL agent control up to 25 minutes in advance using Boosting algorithms as well deep learning. Analysis of feature importance, identifying crucial grid regions

  - **State-of-the-art:** Only cascading failure prediction models (without agent control) [1]

  - **Contribution:** Novel application of failure prediction for grids under control of DRL agents

  - **Implemented WP2 features:** Behavior Analysis, Explainability.

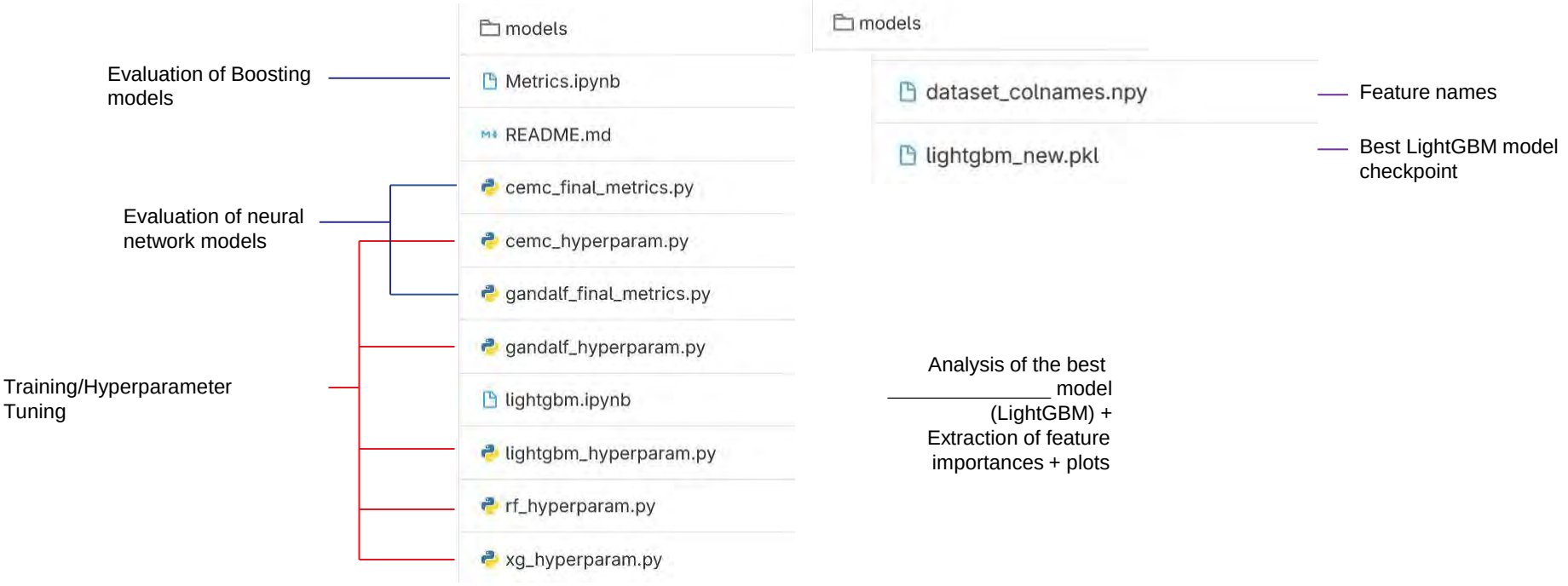
[1] Islam, Md Zahidul, et al. "Cyber-physical cascading failure and resilience of power grid: A comprehensive review." Frontiers in Energy Research 11 (2023)

Algorithm – Failure Prediction

Motivation & Approach

- Deep Reinforcement Learning (DRL) agents provide recommendations for actions in a black-box manner with no insights on confidence
- Alert operator sufficiently early with increasing certainty that AI agents are about to fail
- Generate data from following the recommendations of DRL agents and train models to predict cascading power grid failures based on the observations and the agent index/type
- Identify critical features and regions to enhance the global interpretability of grid failures.

Overview of code structure



Input data

- Dataset for training models: available at the following Zenodo link: https://zenodo.org/records/13948340
- Input for each model: Grid2op Observation (l2rpn_wcci_2022 environment) of a IEEE 118 transmission grid
  - E.g., generator injections, loads, line status, line loading (ρ) and cooldown, temporal data (weekday, hour), agent type (i.e. which agent was used to collect this data point)
- Format: Numpy / Torch Tensor

Output data

Multi-class prediction of failure

- A prediction on whether the power grid is about to fail, e.g. cascading failure is about to happen in 5, 3, 1 timesteps , i.e., 25min, 15min, 5min, or alternatively a no-failure (i.e.survival) prediction
- Multiple models: Boosting models vs. feed-forward neural networks
- Output type
  - Boolean (Failure vs. Survival)
  - In case of failure: Degree of imminency (i.e., failure in 25min, 15min or 5min)

Analysis of the global importance of features according the best model (LightGBM)

- Identify critical features that influence the stability of the power grid.
- Highlight critical regions of high importance in the power grid

Experiments

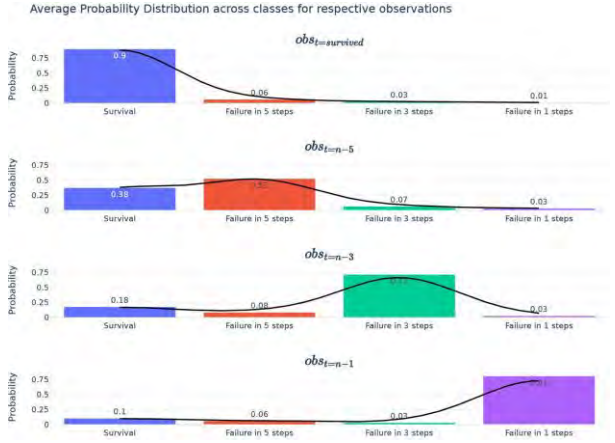
Comparison of 5 different models

- 2 Neural Network Architectures: Feed-forward NN, GANDALF [2]
- Neural Network models underperform, LightGBM performs best.
- □ Indicates a challenge for neural networks to capture feature representations

	accuracy	balanced accuracy	f1 micro	binary accuracy	OOD balanced accuracy	OOD binary accuracy
RF	0.73	0.62	0.73	0.82	0.61	0.82
XGBoost	0.80	0.73	0.80	0.83	0.73	0.83
LightGBM	**0.82**	**0.76**	**0.82**	**0.87**	**0.76**	**0.87**
FNN	0.79	0.73	0.79	0.84	0.73	0.84
GANDALF	0.79	0.72	0.79	0.84	0.72	0.83

Source: Lehna, M., Hassouna, M., et al.: Fault detection for agents on power grid topology optimization: A comprehensive analysis (2024)

Source: Lehna, M., Hassouna, M., et al.: Fault detection for agents on power grid topology optimization: A comprehensive analysis (2024)



Average probability distribution of the LightGBM model for the ground truth survival, failure in 5 steps, failure in 3 steps and failure in 1 step. The probability output is averaged for all observations in a validation set. The results indicate a higher uncertainty for the model in separating between the survival and failure in 5 steps class, indicating that long-term failures are harder to detect.

[2] Joseph, M., Raj, H.: Gandalf: Gated adaptive network for deep automated learning of features (2024)

Experiments – Feature Importance Analysis (1/3)

Evaluation Method

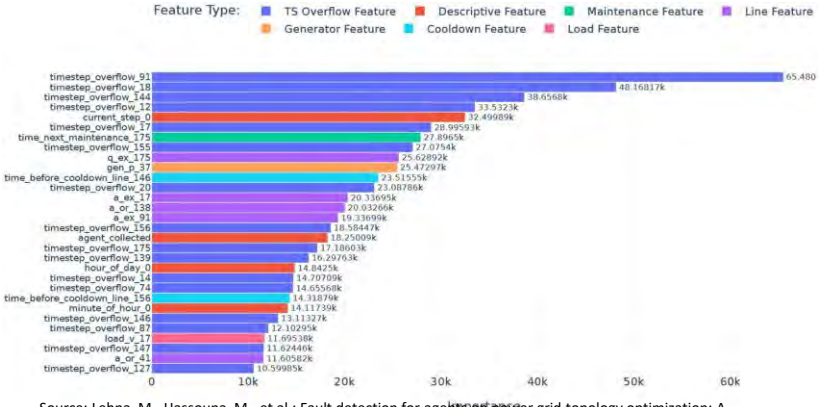
- Used the gain metric to measure how much each feature improves accuracy.
- Helps understand key patterns and relationships in the data.

Top Features

- 16 out of 30 top features are ts_overflow (time since line overload) → strong instability indicator.
- Descriptive Features (e.g., current step, minute of hour, hour of day) show that grid failures follow temporal patterns.
- Agent Type ranks 17th → Highlights different survival behaviors between agents
- Only 2 load/gen features in the top 30 → Grid stability may depend on singular or few generation/consumption fluctuations.

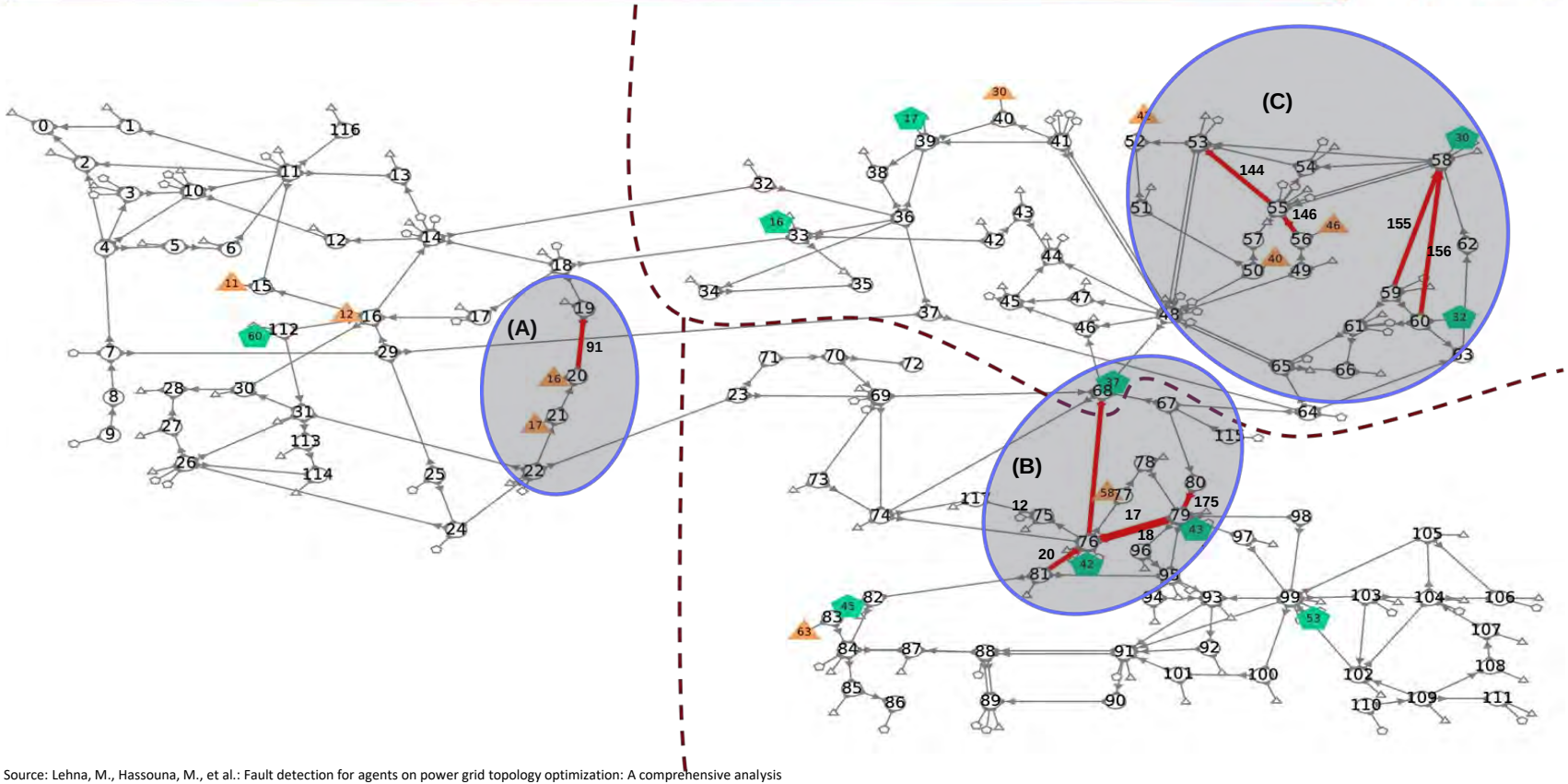
Dominance of line features

- Many features correspond to the same critical lines.



Source: Lehna, M., Hassouna, M., et al.: Fault detection for agents on power grid topology optimization: A comprehensive analysis (2024)

Experiments– Failure Prediction (2/3)



Source: Lehna, M., Hassouna, M., et al.: Fault detection for agents on power grid topology optimization: A comprehensive analysis (2024)

Experiments – Feature Importance Analysis (3/3)

Region A (High-Risk Line & Load Dependency)

- Line between substations 21-22 is frequently attacked, leading to cascading failures in 3 steps.
- Occurs in 4057 out of 39635 cases

Region B (Key Sub-Grid Connection)

- Line between substations 68-76 links two sub-grids, carrying high power flow.
- Generator 37 frequently spikes to 500% of its starting power.
- Lines 79-80 and 76-79 are critical for power routing

Region C (Frequent Adversarial Attacks)

- 10 out of 23 adversarial line attacks occur in this sub-grid.
- High-load lines (e.g., 58-62, 62-63) are frequently attacked, disrupting load supply.
- Three of the top 10 most important loads are in this region (substations 50, 52, 56).

Reproducibility

- Major dependencies: Grid2op, pytorch, pytorch lightning, pytorch tabular, sklearn, xgboost
- Download the dataset and place according to the readme.
- To train the models: Run {model}_hyperparam.py
- To evaluate the models: Run Metrics.ipynb for Boosting models, and cemc_final_metrics.py/gandalf_final_metrics.py for the neural network models.
- Run the Lightgbm.ipynb Notebook to generate Feature importance plots

Authors

Authors	Institution
Mohamed Hassouna	Fraunhofer/UKASSEL

Link to the repository

https://github.com/AI4REALNET/failure_prediction

Learning Topology Actions for Power Grid Control: A Graph-Based Soft-Label Imitation Learning Approach

Fraunhofer/UKASSEL

Context

Definition

Soft-Label Imitation Learning (IL): A method for controlling power grid topology that learns from multiple viable actions rather than a single "correct" expert action.

Motivation

Renewable Integration - Rising renewable energy shares increase grid complexity and congestion events. Limitations of Hard Labels - Traditional IL forces agents to mimic a single expert action, ignoring other effective solutions. This leads to rigid policies that fail to generalize.

Use cases

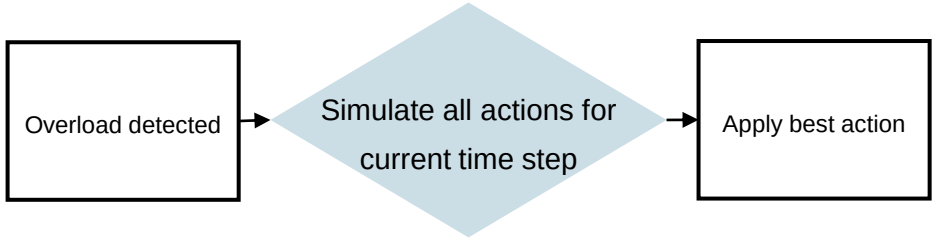
Congestion Management: Automating substation reconfiguration (topology changes) to resolve line overloads.

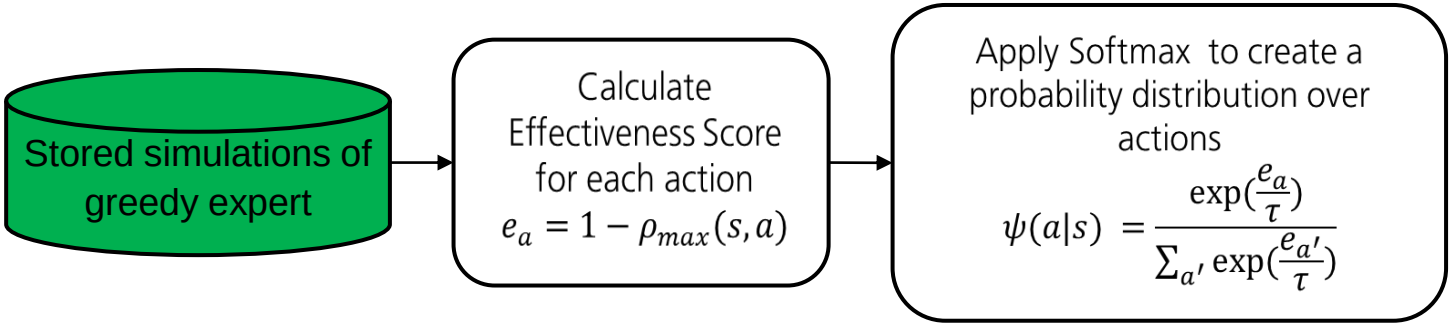
Decision Support: Providing operators with a ranked list of effective remedial actions.

Methodology

- **Main idea** = Combine Soft Labels (capturing the effectiveness of multiple actions) with Graph Neural Networks (GNNs) to create a robust grid control agent.

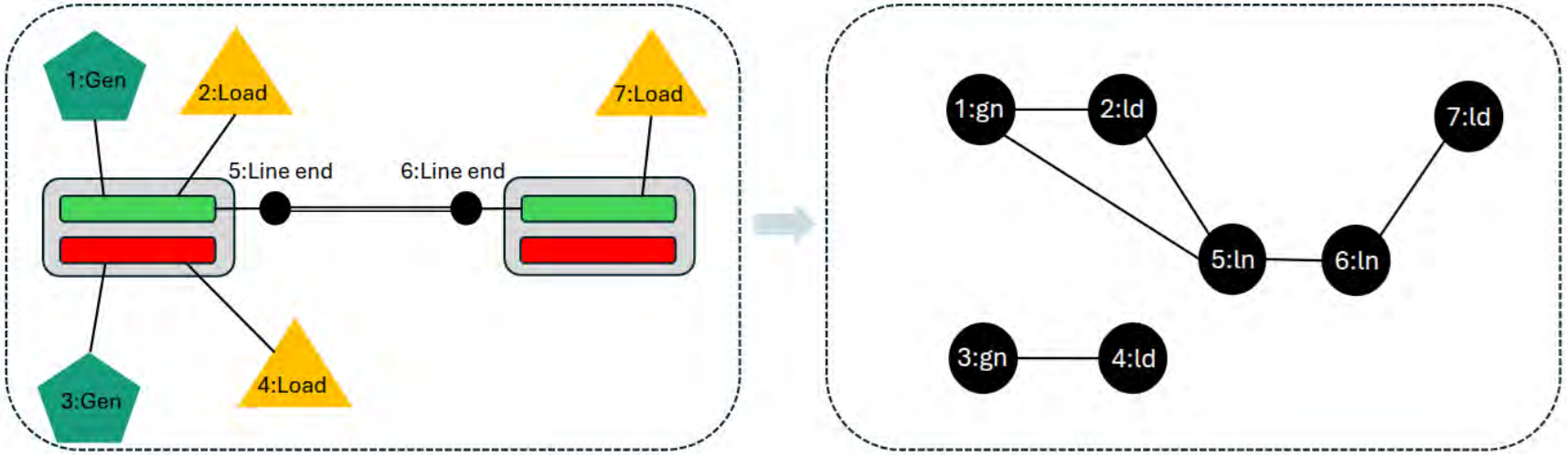
1. **Greedy Simulation („Expert“)**
2. **Soft Label Generation**





Methodology

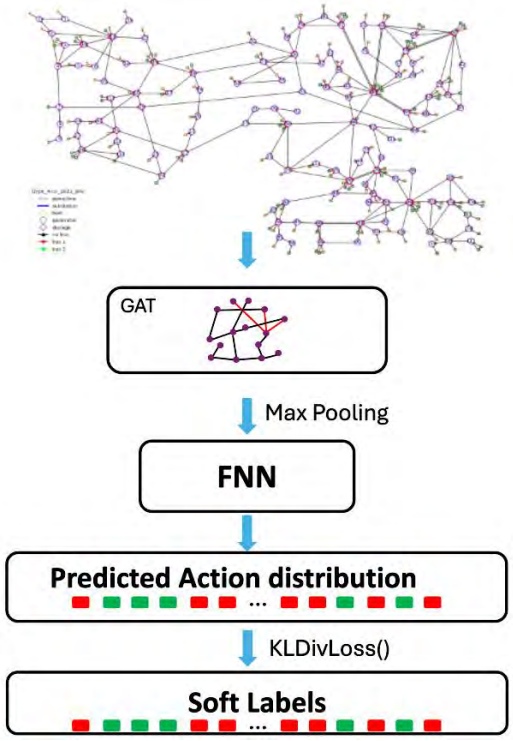
2. Graph Encoding: Convert grid observations into graph structures



Methodology

3. Training: Train a GNN to predict the "effectiveness score" distribution using KL-Divergence loss.

Architecture: Graph → GAT layers → pooled embedding (max) → action distribution ↔ KLDivLoss()



Original contribution

- **Algorithm |** Soft Label GNN Agent

  - **State-of-the-art :**
    - Deep Reinforcement Learning (DRL): Effective but hard to train and unstable. [1]
    - Hard-Label IL: Mimics a "Greedy" expert exactly, inheriting its biases and limitations [2,3].

  - **Contribution:** Novel soft-label approach that captures the full solution space, allowing the agent to learn "universally effective" actions rather than memorizing specific expert choices.
    - **GNN Integration**: Explicitly models the physical topology of the grid for better spatial reasoning.
    - **Superior Performance**: The SoftGNN agent significantly outperforms the expert (Greedy) agent it was trained on as well as SOTA DRL agents from literature.

[1] Lehna, M., Viebahn, J., et al.: Managing power grids through topology actions: A comparative study between advanced rule-based and reinforcement learning agents. Energy and AI (2023)
[2] Hassouna, M. et al. (2026). Learning Topology Actions for Power Grid Control: A Graph-Based Soft-Label Imitation Learning Approach. In: Dutra, I., et al. Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track and Demo Track. ECML PKDD 2025. Lecture Notes in Computer Science(), vol 16022. Springer, Cham. https://doi.org/10.1007/978-3-032-06129-4_8
[3] de Jong, M., Viebahn, J., Shapovalova, Y.: Generalizable graph neural networks for robust power grid topology control (2025), https://arxiv.org/abs/2501.07186

Algorithm –Soft Label GNN Agent

Motivation & Approach

- In power grids, congestion can often be solved by multiple different topology changes. "Hard" labels discard valid alternatives.

1. Soft Label Generation

Algorithm 1 Soft Label Generation

Require: Environment `env`, Set of actions $\mathcal{A}$, temperature parameter τ

Initialize dataset $\mathcal{D} \leftarrow \varnothing$ and environment `env`
for each observation $s \in S$ **do** ▷ Iterate through environment and receive s
  for each action $a \in \mathcal{A}$ **do** ▷ Simulate maximum Line Loads
    Run `env.simulate(`a`)` to get $\rho_{max}(s, a)$
    Compute effectiveness score: $e_a = 1 - \rho_{max}(s, a)$
  end for
  for each action $a \in \mathcal{A}$ **do** ▷ Compute soft labels
    $\Psi(a \mid s) = \frac{\exp(e_a/\tau)}{\sum_{a' \in \mathcal{A}} \exp(e_{a'}/\tau)}$
  end for
  Store $(s, \Psi(a \mid s))$ in dataset $\mathcal{D}$
  Apply greedy optimal action $a^* = \arg\min_{a' \in \mathcal{A}} \rho_{max}(s, a')$ using `env.step(`a^*`)`
end for
return $\mathcal{D}$

Algorithm –Soft Label GNN Agent

2. Conversion to Graph Data to use with GNNs

Node Features

- Nodes are generators, loads and line ends
- Voltage, active and reactive power of generators & loads, line loadings
- Current topology assignment of each node
- Operational constraints line cooldowns on lines & substations

Edges

- Electrical connection is represented in the graph connectivity, i.e. both line ends are connected as well as the busbar connectivity

Algorithm –Soft Label GNN Agent

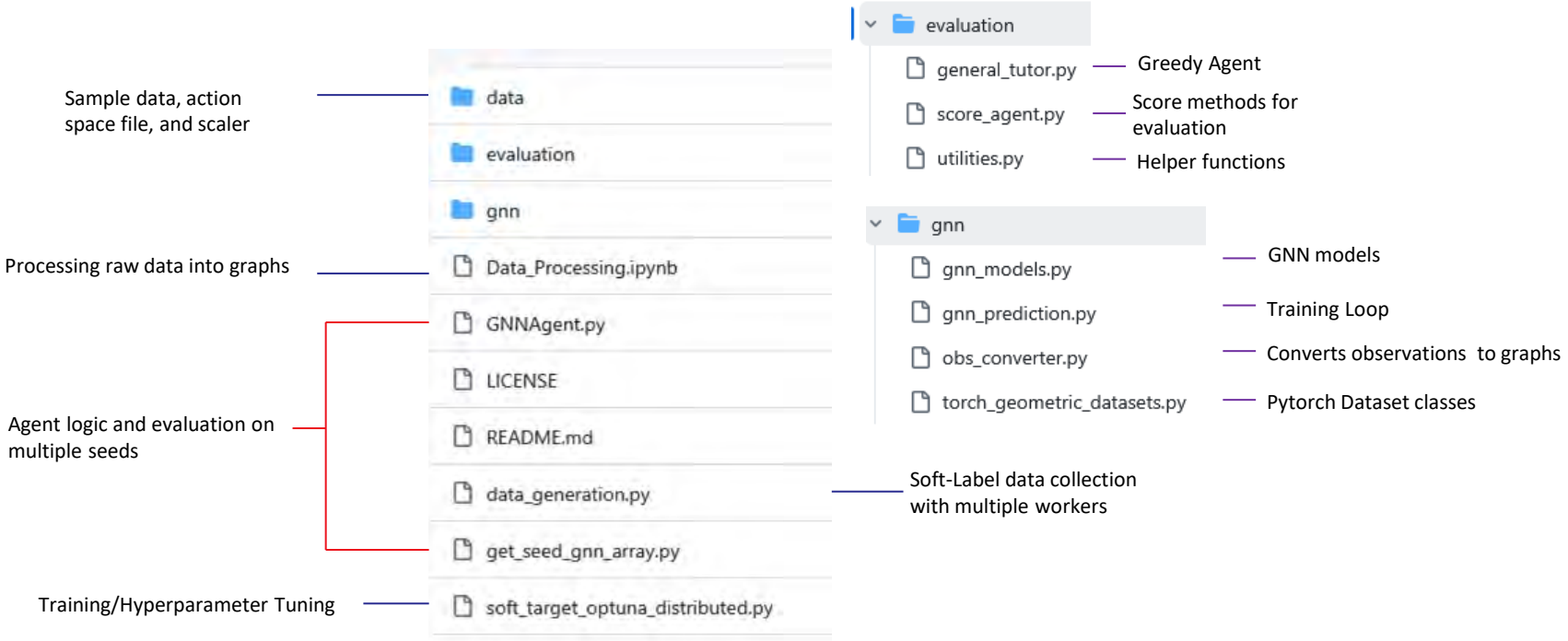
3. Training and Agent

Architecture: Graph → GAT layers → pooled embedding (max) → action distribution ↔ KLDivLoss()

SoftGNN Agent

- Activates if max line loading ρ_max≥90%
- GNN predicts action distribution → rank actions
- Validate with simulation → pick first feasible action reducing overload
- Enhancements: Topology Reversion & N-1 Security Check

Overview of code structure



Input Data

Grid Representation

- Environment: L2RPN WCCI 2022 (IEEE 118-bus system).

Graph Construction

- Nodes: Generators, Loads, Storage, Transmission Line ends.
- Features: Active/reactive power, voltage magnitude/angle, line loading, …
- Edges: Physical connections (busbars and power lines).

Format: Numpy/Torch Tensor

Output Data

Action Predictions

- Soft Scores: A probability distribution over the discrete action space (2030 possible topology actions).
- Interpretation: Higher scores indicate higher confidence that the action will resolve congestion. This serves as a confidence measure in the action.

Agent Decision

- Ranked List: The agent sorts actions by predicted score.
- Feasibility Check: The top actions are simulated to ensure they reduce ρ-max below 90% (and optionally pass N-1 checks).

Experiments

Comparison of Agents

- Ablation: DoNothing, Greedy Expert (Greedy90%), Hard-label IL (FNN & GNN), and Soft-label FNN.
- State-of-the-art: Senior95% and TopoAgent (CurriculumAgent)

Agent	L2RPN Score					Survival Time	
	$\bar{x}$	σ	$\tilde{x}$	Q_1	Q_3	$\tilde{x}$	MSTCM
DoNothing	00.00	0.00	00.00	00.00	00.00	229	383
$Greedy_{90\%}$	37.91	3.89	37.07	35.62	40.78	1014	1280
$Senior_{95\%}$ [10]	37.13	4.49	37.21	33.48	39.84	988	1160
$SeniorFix_{95\%}$	39.40	**2.98**	39.50	37.14	41.25	1026	1468
$TopoAgent_{85-95\%}$ [10]	41.26	3.01	40.41	39.41	43.69	1232	1436
$TopoAgentFix_{85-95\%}$	41.81	3.11	42.17	40.33	44.00	1263	1494
$HardFNN_{90\%}$	37.54	3.87	37.08	35.06	40.22	1020	1114
$SoftFNN_{90\%}$	40.73	4.16	40.54	36.60	43.64	1113	1316
$HardGNN_{90\%}$	38.28	3.58	37.95	35.85	40.08	1048	1255
$SoftGNN_{90\%}$	43.84	3.60	**43.96**	41.40	46.09	1293	1479
$SoftGNN_{90\%}$ $N-1$	**44.43**	3.27	43.49	**42.33**	**47.34**	**1299**	**1566**

- Demonstrates the value of soft labels (L2RPN Score: 43.84 vs. 38.28).
- SoftGNN beats the teacher (Greedy90%).
- Safety Aware: N-1 checks achieves the highest score and median survival time.

➢ SoftGNN ourperforms SOTA DRL (Senior/TopoAgent), sets new record score on IEEE 118 grid with rigorous 20-seed evaluation.

Source: Hassouna, M. et al. (2026). Learning Topology Actions for Power Grid Control: A Graph-Based Soft-Label Imitation Learning Approach. In: Dutra, I., et al. Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track and Demo Track. ECML PKDD 2025. Lecture Notes in Computer Science(), vol 16022. Springer, Cham. https://doi.org/10.1007/978-3-032-06129-4_8

Experiments – Feature Analysis & Discussion

Why Soft Labels Work?

- Generalization: Soft labels prevent overfitting to sub-optimal expert decisions.
- Label Noise: Reduces the impact of "noise" where the expert arbitrarily picks one of two equally good actions.
- Actions that are effective across many states naturally receive higher label weights, guiding the agent toward broadly useful strategies.
- Implicitly learns to prefer generalizable actions

Why GNNs Work?

- Topology Awareness: GNNs capture spatial dependencies, propagating congestion information across the grid structure better than FNNs.

Real-world Relevance

- Provides a "confidence score" akin to those needed in control rooms, allowing operators to see multiple valid options.
- Multiple Action Recommendations

Reproducibility

Major dependencies

- PyTorch, PyTorch Geometric (for GAT implementation), Optuna (hyperparam. tuning)

Training Details

- Hardware: Trained on NVIDIA A100 GPUs (~8 hours).

- Hyperparameters: Optimized learning rate, weight decay, and dropout using Tree-structured Parzen Estimator (TPE).

Authors

Authors	Institution
Mohamed Hassouna	Fraunhofer/UKASSEL

Link to the repository

https://github.com/AI4REALNET/soft_label_gnn

GridExplainer: Explaining Reinforcement Learning Agents for Power Grid Operations

Fraunhofer/UKASSEL

Context

Definition

GridExplainer is a comprehensive framework tailored for Explainable Artificial Intelligence (XAI) specifically designed for Deep Reinforcement Learning (DRL) agents in power grid management.

Motivation

Black-Box Nature: While DRL is effective for dynamic environments, its opacity raises concerns about transparency and reliability in critical infrastructure. **Trust**: Operators and developers need to understand decision-making processes to trust automated agents, as well as analyze and improve agent development.

Use cases

Topology Control: Explaining adjustments to power lines and bus assignments.
Decision Support: providing visual insights to human operators in control rooms.

Methodology (1/3)

- **Main idea** = Combine **Post-hoc Feature Attribution** methods with **Domain-Specific Visualization** to shed light on RL agent decisions.

- **Modular Toolbox**: Built for the Grid2Op ecosystem, integrating the *Captum* library for diverse attribution methods, the *Quantus* library for evaluation of explainability metrics, and a visualization that highlights critical features on the grid.

- **Two Pillars**: The framework consists of a suite of Feature Attribution Algorithms and a rigorous Evaluation Protocol

Methodology (2/3)

Feature Attribution Categories:

- **Perturbation-Based**: Measures importance by altering features (e.g., Feature Permutation, KernelSHAP).
- **Surrogate-Based**: trains interpretable models on local neighborhoods (e.g., LIME).
- **Gradient-Based**: Computes output gradients with respect to input features (e.g., Saliency, Input Gradient, Integrated Gradients).

Methodology (3/3)

To strictly validate the explanations, GridExplainer calculates the following metrics rather than relying solely on visual inspection

- **Faithfulness**: Assesses whether the feature importance scores provided by an explanation method accurately reflect the model's decision-making process
- **Robustness:** Evaluates the stability of explanations against small input perturbations, under the assumption that the model's output remains relatively unchanged
- **Complexity:** Indicates how concise the explanations are, i.e. how few features are used to explain a model prediction.
- **Randomization:** Assesses the validity of the method by comparing explanations of the trained model against those of an initialized (randomized) model.

Original contribution

- **Algorithm |** GridExplainer

  - **Short description:** Calculation of feature attribution values, quantifiying the importance of each feature (at each time step) for the decision of the agent, visualize and highlight the most important features on the grid.

  - **Contribution:** Novel application of feature attribution for grids under control of DRL agents as well as combination with quantitative evaluation and visualization

  - **Implemented WP2 features:** Explainability.

Algorithm – GridExplainer (1/2)

Motivation & Approach

- Deep Reinforcement Learning (DRL) agents provide recommendations for actions in a black-box manner with no insights on the reasons for taking certain actions
- Identify critical features and regions to enhance the interpretability and explainability of agent actions.
- Highlight the important features in a visual way
- Evaluate different explainability methods based on quantative metrics for explainability

Algorithm – GridExplainer (2/2)

The Framework

- **Input**: Pre-trained DRL Agent (e.g., CurriculumAgent) + Grid State (IEEE118).
- **Attribution**: Calculate feature importance using various gradient and perturbation-based methods.
- **Visualization**: Extend grid2viz to overlay importance scores directly on the grid topology.

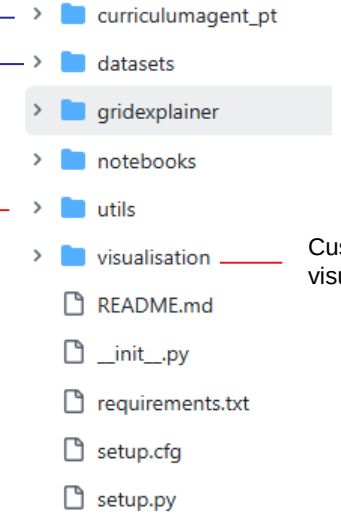
Overview of code structure

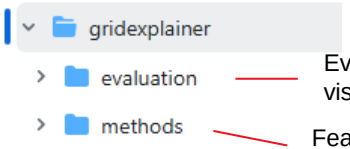
Pre-trained RL agent (CurriculumAgent)

Dataset classes for Grid2Op and XAI observations

Observation converters

Custom Plotly-based grid visualization tools





Evaluation metrics and radar chart visualizations

Feature attribution method wrappers

2 Notebooks:
- Visualizing agent decisions on the grid
- Benchmarking XAI methods

Input data

- Dataset for training models: available at the following Zenodo link: https://zenodo.org/records/13948340
- Input for each model: Grid2op Observation (l2rpn_wcci_2022 environment) of a IEEE 118 transmission grid
  - E.g., generator injections, loads, line status, line loading (ρ) and cooldown, temporal data (weekday, hour), agent type (i.e. which agent was used to collect this data point)
- Format: Numpy / Torch Tensor

Input data

- Dataset collected from curriculumagent is present in the repo under datasets/
- The framework ingests a comprehensive vector representing the instantaneous state of the power grid (derived from grid2op)

- For calculating the feature attributions, the respective methods require access to the Pytorch model of the agent. An example agent based on CurriculumAgent is present in the repo.

Output data

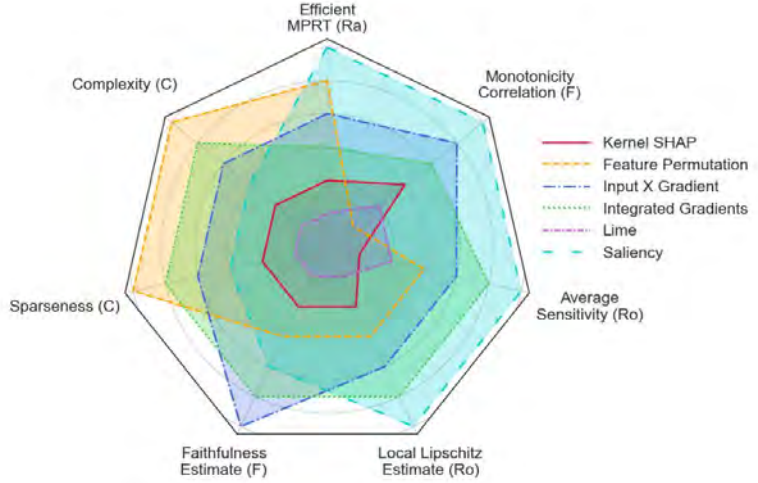
- The raw output is a Feature Importance Vector (same shape as input) which is then mapped to the physical grid.
- Attribution Scores: Quantified contribution of each element (e.g., "Line 42 flow contributed +0.8 to the decision").
- Visual Overlay: Scores for each feature are aggregated by component to generate Heatmaps on the grid topology (Red = High influence on failure/action).

Experiments

Comparison of diverse feature attribution methods

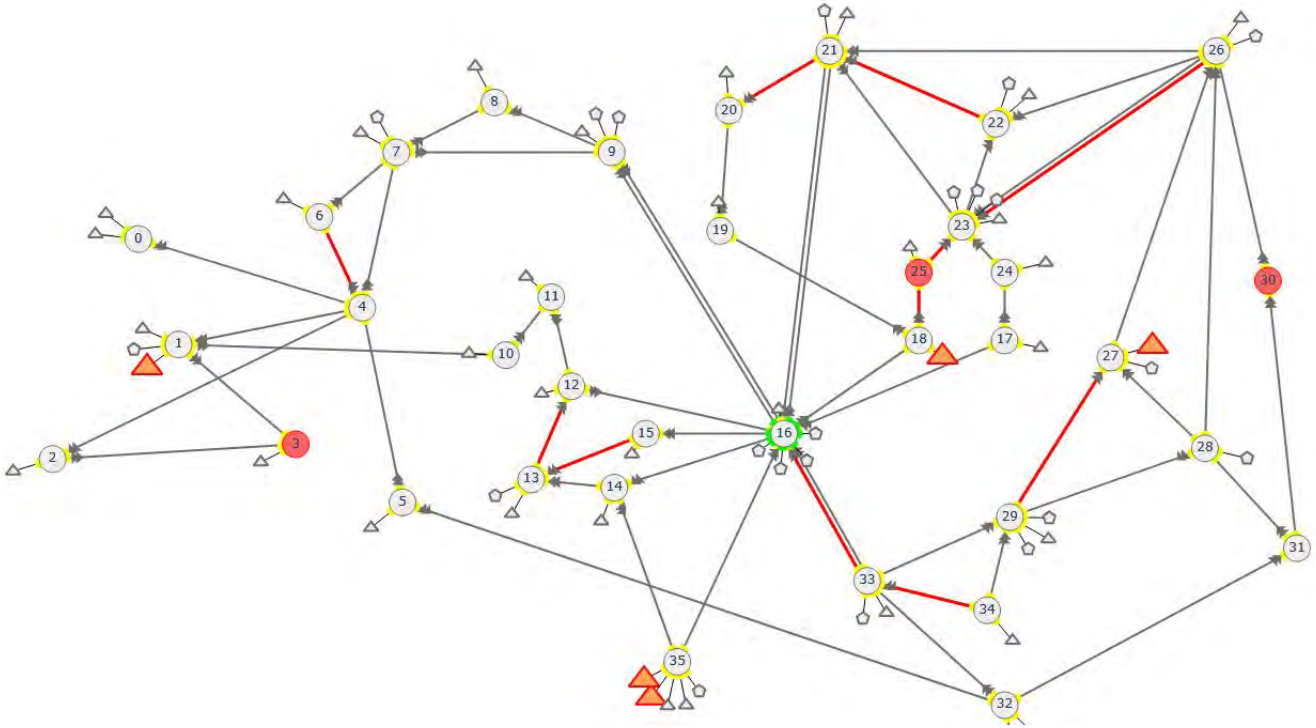
- Quantative Analysis: Evaluation of the explainability methods based on multiple metrics.

- Qualitative Analysis: Systematic plot and analysis of singular explanations using the developed visualization methods



Radar Chart plot of the different metrics calculated for each feature attribution method

Experiments



Example plot that highlights critical components (as identified by KernelShap) such as lines and loads in the power grid.

Reproducibility

- Major dependencies: Grid2op, grid2viz, pytorch, captum, quantus
- Clone the respository
- To run feature attribution methods and visualize the results, execute the notebook notebooks/qualitative_analysis.ipynb
- To calculate explainability metrics, run the notebook notebooks/qualitative_analysis.ipynb

Authors

Authors	Institution
Mohamed Hassouna	Fraunhofer/UKASSEL

Link to the repository

https://github.com/AI4REALNET/GridExplainer

TraceRL, Maze-Flatland (XGBoost)

EnliteAI

Outline

- AI4REALNET – Task 2.2 & 2.3
- Replicate the results
- Performance Comparison – NN vs XGBoost
- Trace-RL
- Link to Repositories

AI4REALNET – Task 2.2 & 2.3

2. → Environment engineering

- Improved environment engineering to **enhance AI-based agent development**;
- Designed multiple reward formulations **improving feedback frequency**, leading to more efficient training

3. → Transparent decision making with gradient-boosting-based model

- Integrates **XGBoost** without limiting agent capabilities. Full functionalities remain available.
- Lays the foundation for **human trust through transparency**, making **decision-making interpretable**.
- **Maintains comparable performance** when compared to deep dense networks for decision-making, while significantly **cutting training time** and **guaranteeing transparency**.
- Trace-RL: Developed a tool for interpretable and interactive analysis of RL agents in collaboration with T3.1. This tool enables the interactive visualization of trajectories as well as (with integration of T3.1 A3S)

While the first phase involved interconnected development, the maze-flatland repo is a foundation for ongoing development and future advancements in both tasks.

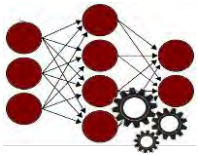
Maze-Flatland Training – Replicate the results

Offline dataset

Config file

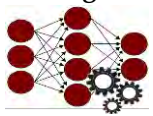
Input

Training of AI



Output

RL Agent



Maze-Flatland Training – Replicate the results

1) Pre-step

Install maze-**flatland**

```
conda env create --file environment.yml
conda activate maze-flatland
pip install -e .
pip install git+https://github.com/enlite-ai/maze.git@dev
```

Download the dataset

Download and extract
https://drive.google.com/file/d/1FW6FnAKHqXXu_LDbdeWQR32jtTQeFc5o

2) Imitation Learning

Neural Network - run the following command

```
maze-run -cn conf_train +experiment=offline/train/bc_train
trajectories_data=</path/to/dataset>
```

XGBoost - run the following command

```
maze-run -cn conf_train +experiment=offline/train/xgboost_train
trajectories_data=</path/to/dataset/>
```

3) Validation

Neural Network - run the following command

```
maze-run +experiment=multi_train/rollout/validation_torch_policy
input_dir=<path/to/policy>
```

XGBoost - run the following command

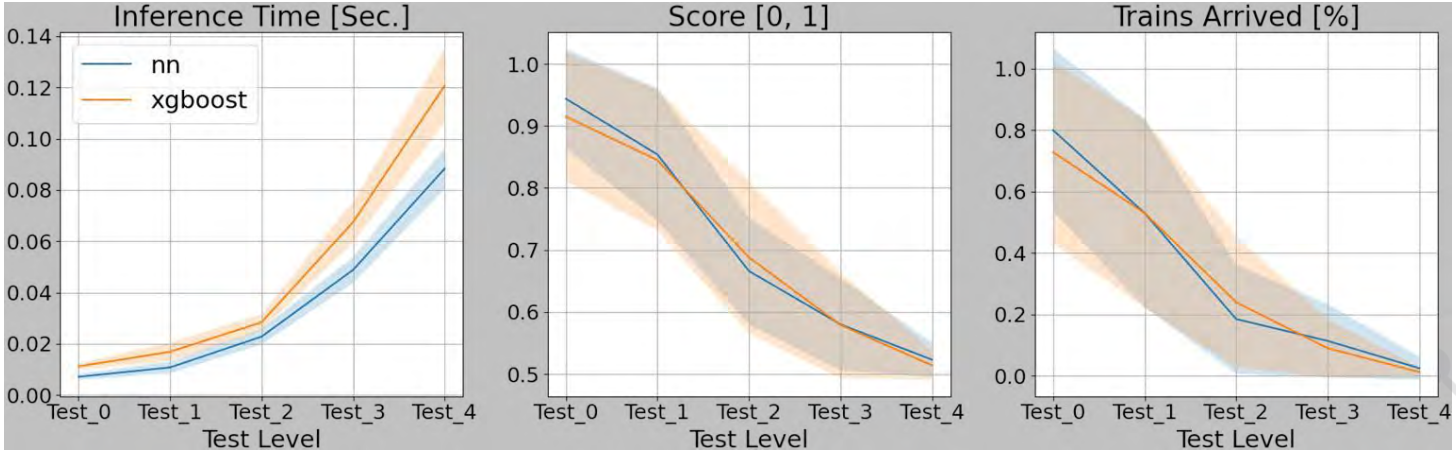
```
maze-run +experiment=multi_train/rollout/validation_xgboost
input_dir=<path/to/policy>
```

4) Compare

Analyse the *validation_stats.csv* files in the output directories

Repository link: https://gitlab.inesctec.pt/cpes/european-projects/ai4realnet/enliteai/beta_release/maze-flatland

Performance Comparison – NN vs XGBoost



Env setup for each level

	#agents	Map size	n_cities	Max Rail pairs in city	Max rails between cities	Malfunction rate	Malfunction interval
Test_0	7	30x30	2	2	2	1/540	[20, 50]
Test_1	10	30x30	2	2	2	1/900	[20, 50]
Test_2	20	30x30	3	2	2	1/1800	[20, 50]
Test_3	50	30x35	3	2	2	1/4500	[20, 50]
Test_4	80	35x30	5	2	2	1/7200	[20, 50]

TraceRL

A framework for interpretable and interactive analysis of RL agents

Main objective

- Visualize
- Understand
- Steer RL agents

Features

- Domain-agnostic
- Method-agnostic
- supports branching trajectories exploration

Phase 1
Record agent-env interactions with a wrapper

Phase 2
Store the trajectory

Phase 3
Visualize

Phase 4
What-IF scenario with human-input
(with T3.1 A3S)

Link to the repository

Link to repository:
https://github.com/AI4REALNET/maze-flatland
https://github.com/AI4REALNET/agent-as-a-service-trace-rl

Authors

Authors	Institution
Anton Fuxjaeger	EnliteAI
Alberto Castagna	EnliteAI

Explanations for Action Alternatives

University of Amsterdam (UvA)

Outline

- Context
- Methodology
- Results
- Overview of repo structure

Context

Definition

Successor Features (SFs) have been used in **Explainable Reinforcement Learning** to show what long-term outcomes we can expect to achieve under a given policy.

Motivation

An operator's **desired control policy is often not known**. For example because their goals and constraints are not fully defined. The correct outcomes are therefore uncertain to an AI system that wants to explain them, and thus we **must account for this uncertainty in the generated explanations**.

Use cases

Predicting long-term outcomes of actions available now helps operators control **complex networked systems** like power grids, railway networks and air traffic.

Methodology

- **Main idea** = Explain action outcomes (through successor features) even if we are uncertain about the policy that will be followed to achieve them.

Successor features capture outcomes under a policy π. Outcomes are defined as the expected discounted sum of state-action features over time.

$$\lambda(s,a) = \phi(s,a) + \gamma \mathbb{E}_{s' \sim \mathbb{T}(s'|s,a);a' \sim \pi(a'|s')}[\lambda(s',a')]$$

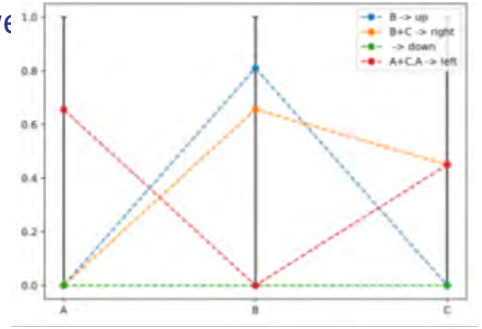
Where $\phi(s,a)$ defines a state-action feature function which can be arbitrarily defined. For example, for a power grid, we could define $\phi(s,a)$ to represent total line overload, so that the successor features $\lambda(s,a)$ capture the expected line overload in the future under policy π after taking action a.

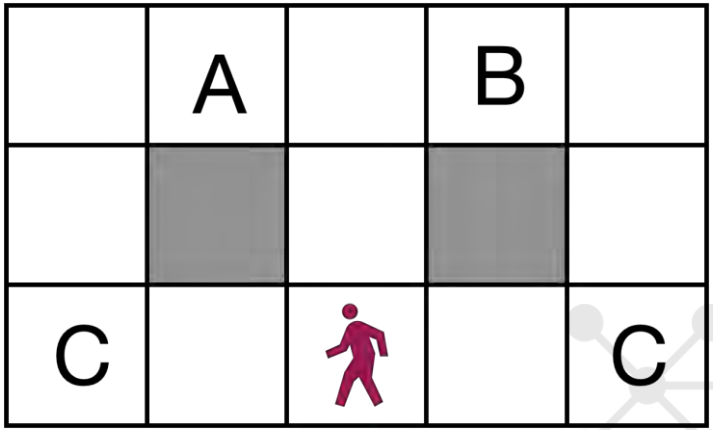
Successor features are a great way to **explain to operators what the future implications of their actions** are. However, they are specific to a given policy. How do we generate these explanations when we don't know π?

Methodology Aim

Consider the gridworld on the right. We aim to generate explanations that show which goals (A/B/C) can be achieved with each action (left/up/right) from the current state under any policy the operator is likely to follow.

Below, an example explanation shows which goals can be achieved for each action, incorporating in the predicted outcomes the four policies that can be followe





The above gridworld has four optimal policies, depending on which goals the operator would like to reach (A / B / A & C / B & C). The explanation must incorporate all four policies.

Original contribution

- **Problem formalization:** Definition of Successor Features under evolving beliefs about an operator.
- **Learning procedure**: ML-based learning of outcomes. Because outcome explanations determine future behavior but are also derived from the same future behavior, explanations form a fixpoint.
- **Implementation**: Implementation in Python for general-purpose Gym environments.

Methodology

Let ω be the unknown reward parameters of the operator. These define what the operator values in their reward function, and thus identifies their desired policy $\pi(a \mid s, \omega)$.

Consider an AI which has a current belief $B(\omega)$ about the operator. At each state the operator choosen an action based on the explanations given by the AI (and ω). From this interaction the AI can update its belief to $B'(\omega)$.

Correct outcomes can be calculated by tracking how the global state (s, B) of this interactive system evolves. Here, s is the state of the control problem, and determines which states can be encountered in the future. B are the AI's beliefs, which evolve as the AI interacts with the operator, and determine which optimal policies remain likely to be followed in the future. The corresponding successor features are:

$$\lambda(s, B, a) = \phi(s, a) + \gamma \mathbb{E}_{s' \sim \mathbb{T}(s' \mid s, a); a' \sim \hat{\pi}(a' \mid s')} [\lambda(s', B', a')]$$

where $\phi(s, a)$ remains as before and $\hat{\pi}$ is the mixture of operator policies under the belief B.

Following established work, we can learn to predict $\lambda(s, B, a)$ using neural networks.

Overview of repository structure

The code is located in two main folders:

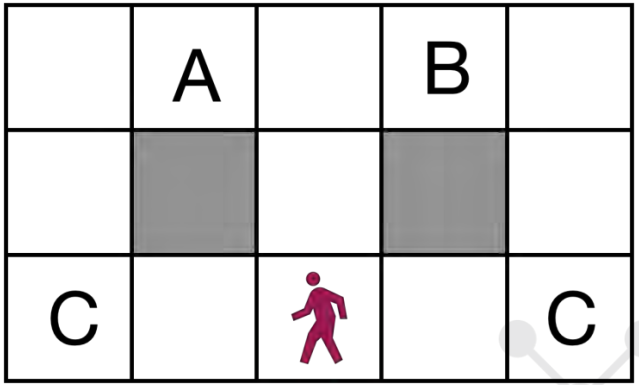
- **envs** contains the implementation of the two control problems we test on.
- **team_SF** contains the code for learning to predict outcomes $\lambda(s, B, a)$

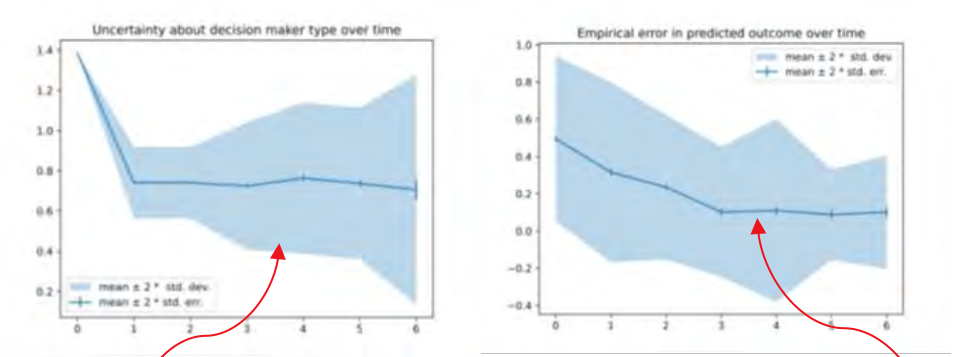
The main entrypoint is main.py, located at the root of the repository. Please find information on how to run the code in README.md

Results

GridWorld

3-objective GridWorld where the operator's desired combination of objectives (A / B / A & C / B & C) is not known a-priori.

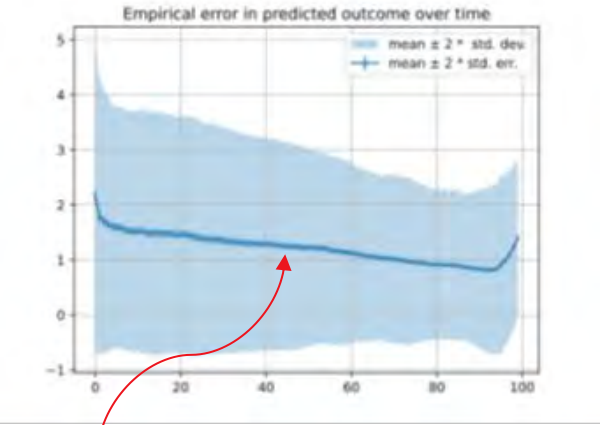


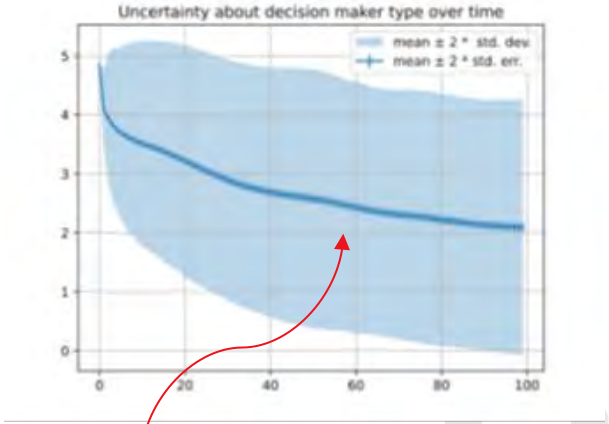


Error in predicted outcomes is low … and decreases as AI becomes more certain about operator's desired objectives.

Dam Control

Realistic 3-objective control problem of a Dam. Objectives are to (1) minimize flooding, (2) maximize irrigation, and (3) maximize hydro power. The operator's weighting of these three is not known.





As before, error in predicted outcomes is low … and decreases as AI learns more about the operator.

Authors

Authors	Institution
Sebastiaan De Peuter	UvA
Herke van Hoof	UvA

Link to the repository

https://github.com/AI4REALNET/T2.3_explaining_action_alternatives

Safe and Constrained RL

Politecnico di Milano (POLIMI)

Outline

- Context
- Original Contribution
- Methodology
- Algorithm
- Overview of repo structure

Context

Definition

Safe Reinforcement Learning (Safe RL) introduces ways to learn satisfying reliability constraints.

Motivation

Large-scale networks are characterized by constraints that the deployed **agents must fullfill.** This cannot ensured by standard RL algorithms.

Use cases

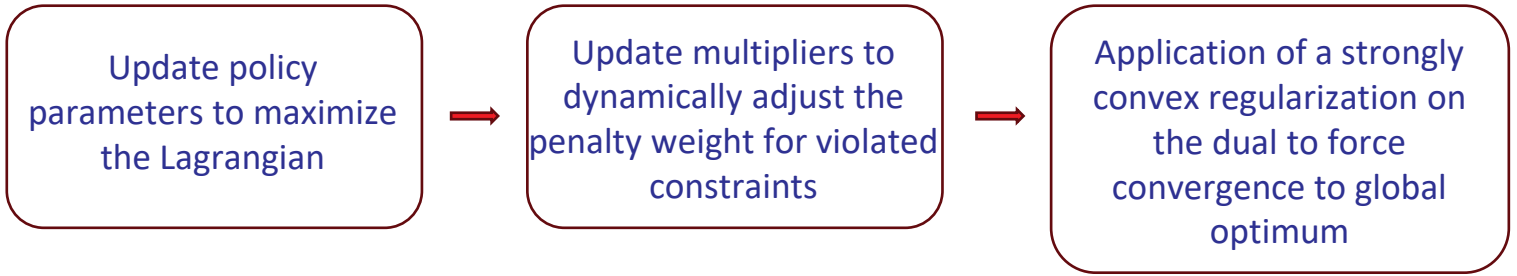
Power grids and railway networks require to be operated with **agents with constraints and working with risk measures**.

Original contribution

- **Algorithm |** Safe and Constrained Policy Gradient (C-PG)
  - ➢ **Short description:** Implementation of a Safe PG algorithm called C-PG supporting constraints and risk measure optimization
  - ➢ **State-of-the-art:** Right now, algorithms support 1 constraint only or discrete state and action spaces
  - ➢ **Contribution:** First safe RL method working with multiple constraints and on continuous state and action spaces factorization of control problems
  - ➢ **Implemented WP2 features:** Safe AI

Methodology

- **Main idea** = A policy gradient method alternating ascent/descent scheme between the policy (primal) and the Lagrange multipliers (dual), enhanced by a regularization term to stabilize the learning process.



C-PG Algorithm - Detailed View

- The C-PG algorithm can be implemented in both the Action-based (C-PGAE) and Parameter based exploration (C-PGPE) declinations

Algorithm 1: C-PGAE.

Input : Iterations K; batch size N; regularization ω; initial parameters: $\boldsymbol{\theta}_0$, $\boldsymbol{\lambda}_0$, and $\boldsymbol{\eta}_0$; step sizes: $\zeta_{\boldsymbol{\theta},k}$, $\zeta_{\boldsymbol{\lambda},k}$, $\zeta_{\boldsymbol{\eta},k}$.

for $k \in [\![K]\!]$ **do**
  Collect N trajectories $\{\tau_i\}_{i\in[\![N]\!]}$ with $\pi_{\boldsymbol{\theta}_{k-1}}$
  if k is odd **then**
    $\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_{k-1} - \zeta_{\boldsymbol{\theta},k-1}\widehat{\nabla}_{\boldsymbol{\theta}}\widetilde{\mathcal{L}}_{\mathrm{A},\omega}(\boldsymbol{\theta}_{k-1},\boldsymbol{\lambda}_{k-1},\boldsymbol{\eta}_{k-1})$
    $\boldsymbol{\eta}_k \leftarrow \boldsymbol{\eta}_{k-1} - \zeta_{\boldsymbol{\eta},k-1}\widehat{\nabla}_{\boldsymbol{\eta}}\widetilde{\mathcal{L}}_{\mathrm{A},\omega}(\boldsymbol{\theta}_{k-1},\boldsymbol{\lambda}_{k-1},\boldsymbol{\eta}_{k-1})$
  else
    $\boldsymbol{\lambda}_k \leftarrow \boldsymbol{\lambda}_{k-1} + \zeta_{\boldsymbol{\lambda},k-1}\widehat{\nabla}_{\boldsymbol{\lambda}}\widetilde{\mathcal{L}}_{\mathrm{A},\omega}(\boldsymbol{\theta}_{k-1},\boldsymbol{\lambda}_{k-1},\boldsymbol{\eta}_{k-1})$
  end
end
Return $\boldsymbol{\theta}_K$

Algorithm 2: C-PGPE.

Input : Iterations K; batch size N; regularization ω; initial parameters: $\boldsymbol{\rho}_0$, $\boldsymbol{\lambda}_0$, and $\boldsymbol{\eta}_0$; step sizes: $\zeta_{\boldsymbol{\rho},k}$, $\zeta_{\boldsymbol{\lambda},k}$, $\zeta_{\boldsymbol{\eta},k}$.

for $k \in [\![K]\!]$ **do**
  Sample N parameters $\{\theta_i\}_{i\in[\![N]\!]}$ with $\nu_{\boldsymbol{\rho}_{k-1}}$
  With each $\{\pi_{\theta_i}\}_{i\in[\![N]\!]}$ collect a trajectory τ_i
  if k is odd **then**
    $\boldsymbol{\rho}_k \leftarrow \boldsymbol{\rho}_{k-1} - \zeta_{\boldsymbol{\rho},k-1}\widehat{\nabla}_{\boldsymbol{\rho}}\widetilde{\mathcal{L}}_{\mathrm{P},\omega}(\boldsymbol{\rho}_{k-1},\boldsymbol{\lambda}_{k-1},\boldsymbol{\eta}_{k-1})$
    $\boldsymbol{\eta}_k \leftarrow \boldsymbol{\eta}_{k-1} - \zeta_{\boldsymbol{\eta},k-1}\widehat{\nabla}_{\boldsymbol{\eta}}\widetilde{\mathcal{L}}_{\mathrm{P},\omega}(\boldsymbol{\rho}_{k-1},\boldsymbol{\lambda}_{k-1},\boldsymbol{\eta}_{k-1})$
  else
    $\boldsymbol{\lambda}_k \leftarrow \boldsymbol{\lambda}_{k-1} + \zeta_{\boldsymbol{\lambda},k-1}\widehat{\nabla}_{\boldsymbol{\lambda}}\widetilde{\mathcal{L}}_{\mathrm{P},\omega}(\boldsymbol{\rho}_{k-1},\boldsymbol{\lambda}_{k-1},\boldsymbol{\eta}_{k-1})$
  end
end
Return $\boldsymbol{\rho}_K$

Overview of code structure

algorithms/: Contains the core implementation of C-PG (Constrained Policy Gradient) derivations such as C-PGAE and C-PGPE and the primal-dual update logic, together with other baselines that can be adopted in these settings.

envs/: Custom wrappers for Grid2Op environments to handle observation normalization and constraint definitions.

data_processors/: Utilities for data processing.

policies/: Definitions for neural network architectures (e.g., MLP) used for both action and parameter exploration.

docs/: Detailed documentation.

run.py: The main entry point to configure hyperparameters, cost functions, and launch training sessions.

Input data

- Desired environment configuration
- Parameters of the C-PG algorithm
- Constraints

Output data

- Learned Policy
- Simulations data

Authors

Authors	Institution
Alessandro Montenegro	POLIMI
Marco Mussi	POLIMI
Alessio Russo	POLIMI
Alberto Maria Metelli	POLIMI
Marcello Restelli	POLIMI

Link to the repository

https://github.com/AI4REALNET/safe-constrained-policy-gradient

MORL - Framework integration of metrics for ethical dimensions

ZHAW

Context (1/2)

Definition

Multi-objective reinforcement learning (MORL) is a form of reinforcement learning in which an agent simultaneously optimizes multiple, potentially conflicting objectives by learning policies that balance trade-offs between them rather than maximizing a single scalar reward.

Motivation

Ethical dimensions in AI are often evaluated separately despite being interdependent, leaving trade-offs implicit and hard to assess. Multi-objective optimization, particularly MORL, addresses this by explicitly modelling competing objectives, enabling transparent trade-off analysis and supporting holistic, use-case-aware trustworthiness-by-design.

Use cases

This implementation is specific to Power Grid domain and power grid simulation environment Grid2OP.

Context (2/2)

Motivation

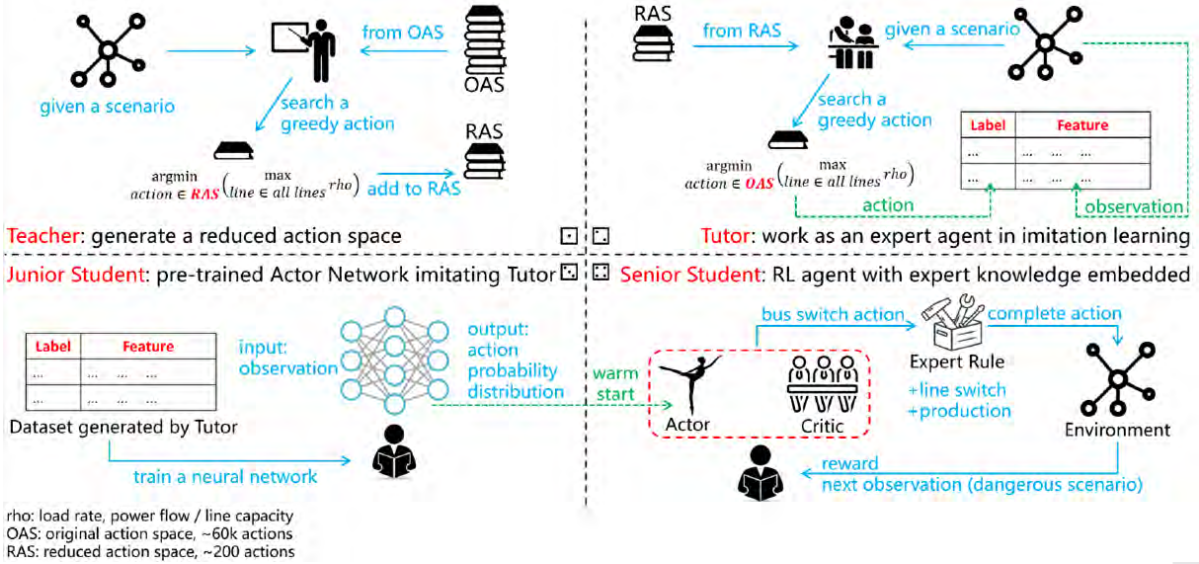
Ethical dimensions in AI are often evaluated separately despite being interdependent, leaving trade-offs implicit and hard to assess. Multi-objective optimization, particularly MORL, addresses this by explicitly modelling competing objectives, enabling transparent trade-off analysis and supporting holistic, use-case-aware trustworthiness-by-design.

Use cases

This implementation is specific to Power Grid domain and power grid simulation environment Grid2OP.

Staged Teacher–Tutor–Junior Student–Senior Student RL pipeline

The algorithm implemented a staged Teacher–Tutor–Junior Student–Senior Student architecture[1], combining expert action generation, imitation learning, and reinforcement learning. Learning is structured as a staged process that progressively embeds expert knowledge, reduces combinatorial complexity, and stabilises exploration before full reinforcement learning is applied.



1. Y. Hu, B. Chen, and K. Tang, Neurips 2020: Learning to run a power network (l2rpn) — ppo solution, https://github.com/Aspirin96/L2RPN_NIPS_2020_a_PPO_Solution, Competition solution repository (Huawei EI Innovation Lab). Accessed: 01.11.2025, 2020.

Staged Teacher–Tutor–Junior Student–Senior Student RL pipeline

Stage 1 – Safety and Robustness Stage. Agent used: Teacher. Designed to ensure stable and secure grid operation by prioritizing objectives related to system survival, overload prevention, and risk minimization. This stage dominates in critical or near-failure states and prevents the agent from trading safety for secondary gains.

Stage 2 – Performance and Structural Efficiency Stage. Agent used: Tutor. Designed to optimize operational efficiency and limit unnecessary or risky control actions once the system operates within safe limits. This stage encourages effective grid management without compromising stability.

Stage 3 – Trustworthiness Objectives Stage. Agents used: Junior Student, Senior Student. Designed to promote longer-term ethical considerations such as fairness and sustainability, including balanced service provision and reduced reliance on carbon-intensive generation. This stage becomes active only when safety and operational constraints are satisfied.

Reward engineering

Block aggregates:

- Survival: metrics describe the number of environment steps before termination and proxy based on episode length.
- Fairness: metrics describe the regional variance of average transmission line loading and variance of curtailment magnitude.
- Sustainability: metrics include a relative emissions proxy and maximum observed line loading.
- Structural: various metrics describing economic cost, complexity proxy and line loading.

Original contribution

- **Algorithm #1 |** Multi-objective reinforcement Learning
  - **Short description**: Evaluation of MORL methods for topology control in transmission grids.
  - **State-of-the-art**: Grid operation is sequential, nonlinear, and stochastic, with decisions made at every moment influencing future system states. RL agents improve continuously by interacting with simulated or real environments, making them adaptive to changing grid conditions.[1]
  - **Contribution**: Model of trade-offs between functional and non-functional objectives based on the analysis of agent's learning behaviour.
  - **Implemented WP2 features**: Staged MORL pipeline with 4 reward blocks.

1 E. van der Sar, A. Zocca, and S. Bhulai, "Multi-agent reinforcement learning for power grid topology optimization," in IEEE PES General Meeting, 2023.

Simulation environment

Grid2Op is an open-source simulation framework specifically designed for reinforcement learning research in power system operation.

Grid2Op represents a transmission power system as a collection of interacting physical components, each corresponding to a standard element in power system modeling.

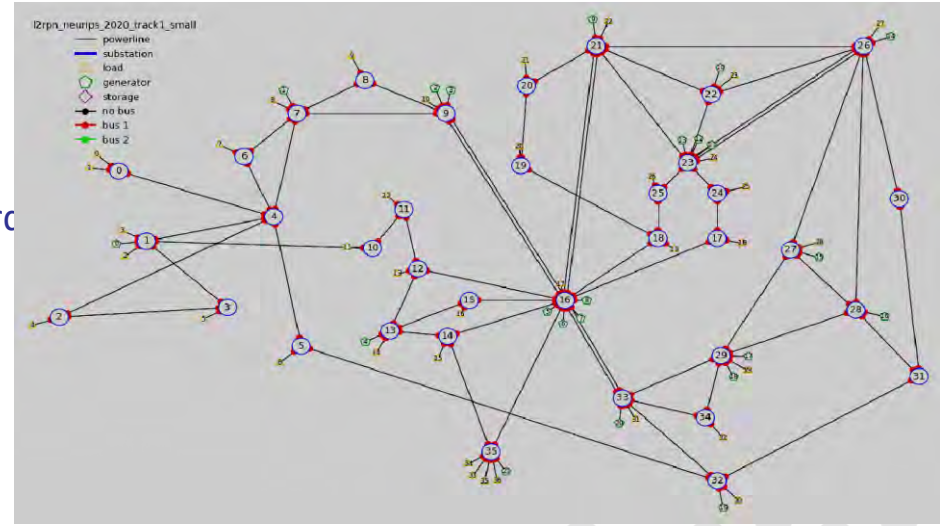
Used environment:

36 substations,

59 powerlines,

22 generators and

37 loads



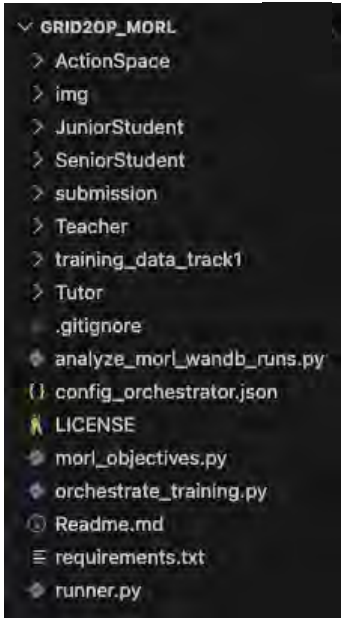
1. Grid2Op Contributors, Grid2op documentation, https://grid2op.readthedocs.io/en/latest/, Accessed: January 2026, 2024.

Overview of code structure

Teacher-Tutor-Junior Student-Senior Student framework

Evaluation of MORL models =>

Core script to run full pipeline =>

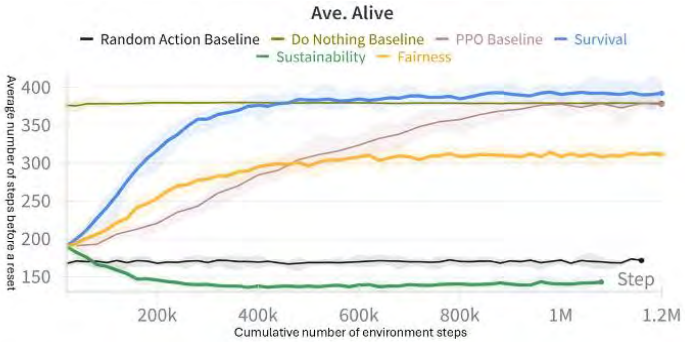


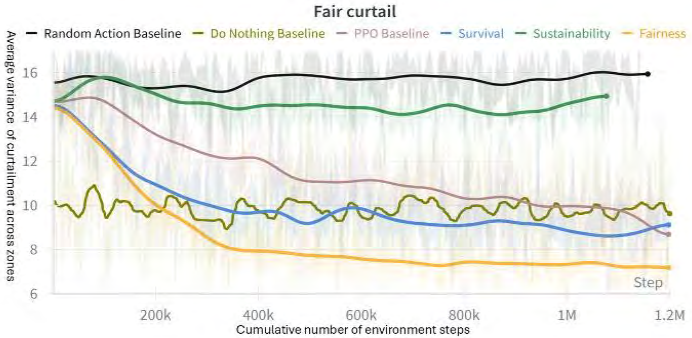
<= configuration file for the orchestrator

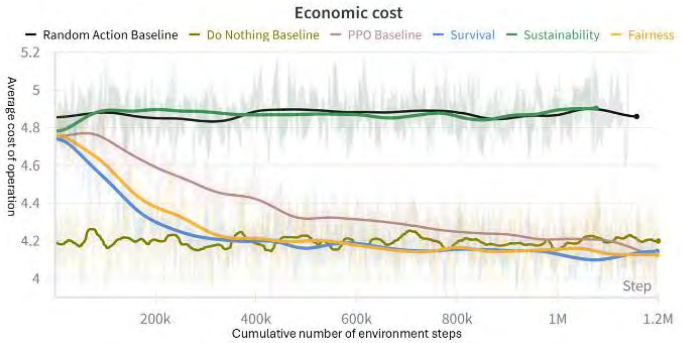
Reproducibility

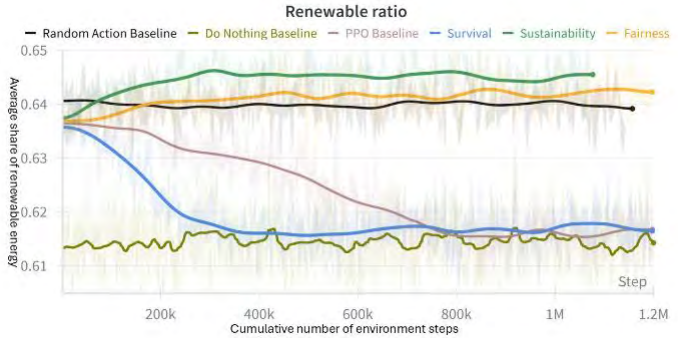
- Major dependencies: Grid2op, pytorch, gymnasium, git lfs
- Training and model data are stored in the git large file storage, make sure that the dependency is installed
- Clone the repository
- Edit config_orchestrator.json to select which stage to execute
- Run orchestrate_training.py

Results









No single control policy optimizes all objectives, but instead a set of Pareto-optimal policies emerges that represent different trade-offs between competing goals.

Perspectives

- Next steps
  - Revisit the **Kassel MORL approach** using the new **multi-objective reward formulation**.
  - Use the **default Grid2Op chronics** instead of a curated subset.
  - The current **regional partitioning of the grid** used for fairness- and structure-related metrics is **imperfect**.

- Future research
  - Apply MORL to **larger Grid2Op benchmarks**.

Authors

Authors	Institution
Ricardo Chavarriaga	ZHAW
Anna Fedorova	ZHAW
Tobias Brandt	ZHAW
Minh-Khan Nguyen	ZHAW

Link to the repository

https://github.com/AI4REALNET/Grid2Op_MORL

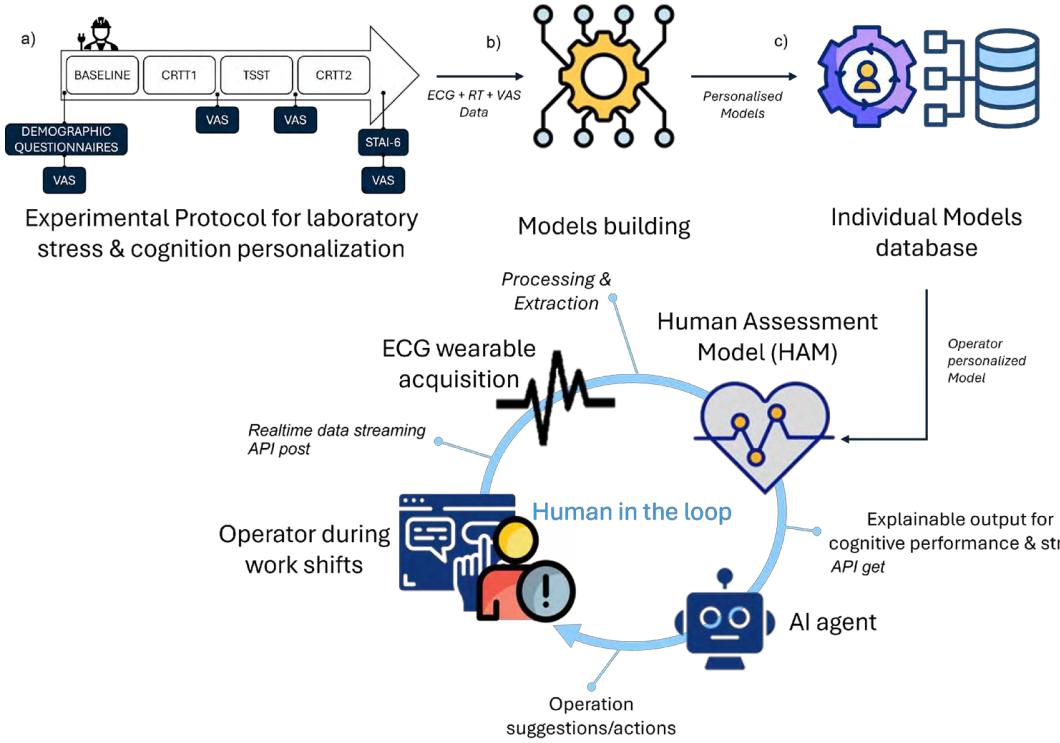
Human Assessment Model

INESC TEC

Context (1/2)

Definition

Human Assessment Model (HAM) aims to provide a real-time quantification of the cognitive and stress level of the operator to support AI agents' autonomy in a seamless and implicit way – i.e., without interacting with the operator.



Context (2/2)

Motivation

AI agents should be able to automatically adapt to provide for timely and adequate decision and suggestion to the human operators, without any direct intervention from them (e.g., through explicit prompt). Such an approach can be achieved with systems like HAM, contributing for a higher empathy and trust with the AI system, potentially leading to a stronger acceptability.

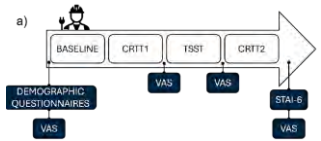
Use cases

Power grids, railway and air traffic networks management software can integrate with the **HAM** through a series of **explainable outputs** describing the current psychophysiological state of the worker**.**

Methodology

Main idea = develop a personalised approach to quantify cognitive performance and stress in real-time

Stress personalised characterisation with ECG recordings in laboratory settings



Model building for Cognitive Performance Evaluation and Stress Detection

Real-time model deployment

- **Preliminary experiments** performed on previously collected ECG signals from 11 Air Traffic Controllers[1].

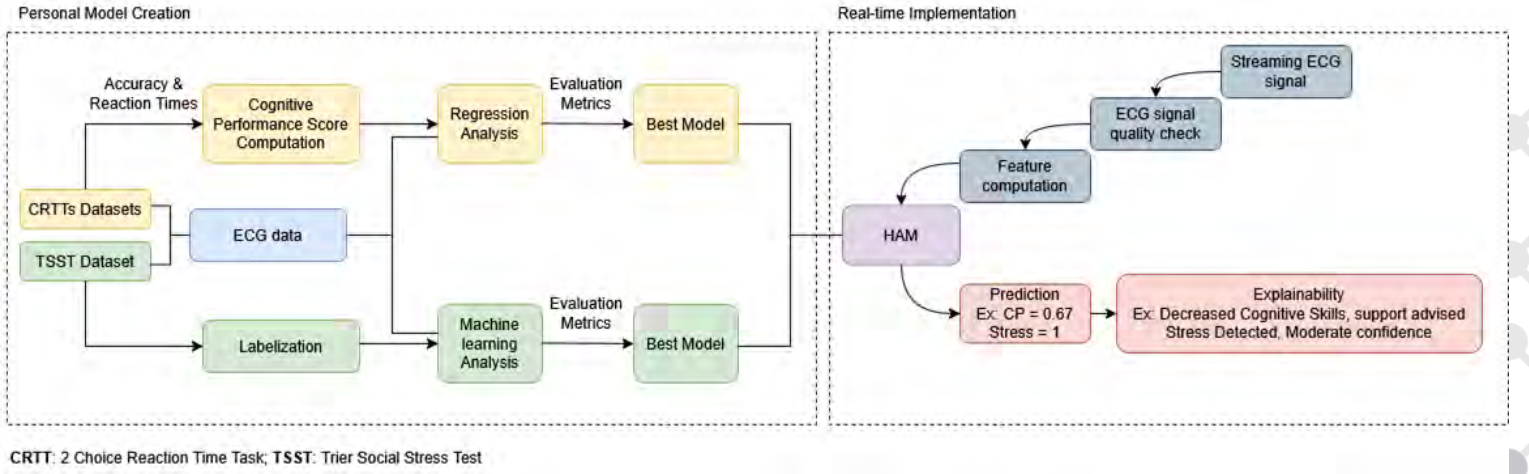
1. Rodrigues, S., Paiva, J. S., Dias, D., Aleixo, M., Filipe, R. M., & Cunha, J. P. S. (2018). Cognitive impact and psychophysiological effects of stress using a biomonitoring platform. International journal of environmental research and public health, 15(6), 1080.

Block Diagram – General Overview

This Block Diagram gives the generic overview of data flow, from the dataset input to the predicted human stress and cognition levels.

The implementation is divided in two main phases:

1. **Personal Model Creation** (Algorithm #1): Cognition (#1.1) and stress (#1.2) modeling for each individual.
2. **Real-time Implementation** (Algorithm #2): Real-time implementation of the HAM.



CRTT: 2 Choice Reaction Time Task; **TSST**: Trier Social Stress Test

Original contribution (1/2)

- **Algorithm #1 |** Personal Model Creation
  - Algorithm #1.1 | Cognitive Performance Evaluation
    - **Short description**: A personalised regressor to estimate the Cognitive Performance score through ECG-derived selected features.
    - **State-of-the-art**: Physiological measures extracted from wearable devices can estimate cognitive skills in patients with cognitive impairments[1].
    - **Contribution**: Personalised regression analysis using ECG and HRV features.
    - **Implemented WP2 features**: Human Cognitive Performance Assessment model.

  - Algorithm #1.2 | Stress Detection
    - **Short description**: A personalised machine learning model to detect stress using ECG-derived selected features.
    - **State-of-the-art**: Machine Learning can successfully detect the presence of stress with HRV parameters[2].
    - **Contribution**: Personalised machine learning model using ECG and HRV features.
    - **Implemented WP2 features**: Human Stress Assessment model

1 Rykov et al., Predicting cognitive scores from wearable-based digital physiological features using machine learning: data from a clinical trial in mild cognitive impairment (2024)

2 Fernandes et. al., HealthSense: Unobtrusive Continuous Stress Monitoring Using a Novel Dual Ecg-PPG Patch (2024)

Algorithm #1: Input data

- The prototypal features datasets were created from data collected according to a specific experimental protocol[1] and saved in .json format.
- Experimental Protocol pipeline:
  - **Baseline**: open eyes rest state for 10 minutes.
  - **2-Choice Reaction Time Task** (**CRTT1**): A selective-attention task where participants identified either the large, global letter or the small, local letters of a hierarchically organized visual object. Response time and accuracy were stored.
  - **Trier Social Stress Test** (TSST): An acute psychosocial stress paradigm involving free speech and arithmetic tasks.
  - **CRTT2**
- Psychological scale assessments (Visual Analogue Scale and STAI-6) were conducted throughout the protocol, and results are stored in the raw dataset.

1. Rodrigues, S., Paiva, J. S., Dias, D., Aleixo, M., Filipe, R. M., & Cunha, J. P. S. (2018). Cognitive impact and psychophysiological effects of stress using a biomonitoring platform. International journal of environmental research and public health, 15(6), 1080.

Algorithm #1: Output data (1/2)

```
ATC 1
['qt_interval', 'tp_segment', 'rr_interval', 'mean_hr', 'median_nni']

bayesian_regression
-------------------
Fitting 2 folds for each of 256 candidates, totalling 512 fits
Best parameters: {'alpha_1': 0.001, 'alpha_2': 1e-06, 'lambda_1': 1e-06, 'lambda_2': 0.001}
R2: -0.26491105093656087
RMSE: 0.19490176915445762

linear_regression
-------------------
Fitting 2 folds for each of 2 candidates, totalling 4 fits
Best hyperparameters: {'fit_intercept': True}
R2: -0.7145291003919052
RMSE: 0.2269123942403238

polynomial_regression
-------------------
Fitting 2 folds for each of 6 candidates, totalling 12 fits
Best hyperparameters: {'linearregression__fit_intercept': True, 'polynomialfeatures__degree': 4}
R2: -44.03810062498398
RMSE: 1.1629895738929719

▪ ▪ ▪

-------------------
-------------------
Best Model
Model: KNRegression
RMSE: 0.10874356716761976
Insert a numeric code to identify the controller:
1
```

Participant Identifier Number

ECG-Derived parameters after feature selection

Model Iteration Block with:

- Type of model.
- Hyperparameters tuning results.
- Coefficient of determination R^2.
- Root Mean Square Error (RMSE)

Best Model Summary

Code Insertion for .pkl file saving

Algorithm #1: Output data (2/2)

```
ATC 1
['pr_segment', 'nni_20', 'range_nni', 'max_hr', 'lfnu']
Insert a numeric code to identify the controller:
1

logistic_regression
------------------
Fitting 3 folds for each of 30 candidates, totalling 90 fits
Best parameters: {'C': 10, 'penalty': 'l1', 'solver': 'saga'}
Balanced Accuracy: 0.875
Weighted F1-Score: 0.9056277056277057
Weighted TPR: 0.9090909090909091

KNN
------------------
Fitting 3 folds for each of 24 candidates, totalling 72 fits
Best parameters: {'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'distance'}
Balanced Accuracy: 0.875
Weighted F1-Score: 0.9056277056277057
Weighted TPR: 0.9090909090909091

SVM
------------------
Fitting 5 folds for each of 32 candidates, totalling 160 fits
Best parameters: {'C': 1, 'gamma': 'scale', 'kernel': 'linear'}
Balanced Accuracy: 0.875
Weighted F1-Score: 0.9056277056277057
Weighted TPR: 0.9090909090909091

The average TPR for detecting stress across all models is: 0.922077922077922
```

Participant Identifier Number

ECG-Derived parameters after feature selection

Code Insertion for .pkl file saving (all models)

Model Iteration Block with:

- Type of model.
- Hyperparameters tuning results.
- Balanced Accuracy.
- Weighted F1-score.
- Weighted TPR.

Overall Best Performance

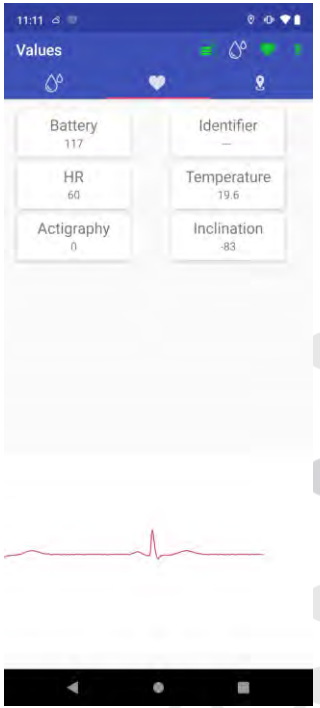
Original contribution (2/2)

- **Algorithm #2 |** Real-time Implementation
  - **Short description**: Personalised models are loaded. Features are extracted from streaming ECG data to make real-time prediction of Cognitive Performance and Stress Detection. HAM output is converted in interpretable strings to better interact with AI agents.
  - **State-of-the-art**: Stress can be predicted within very short windows (10s) in streaming ECG data[1].
  - **Contribution**: An algorithm that retrieves and segment ECG data from server through APIs, computes features and make interpretable prediction of both Cognitive Performance and Stress each 30s.
  - **Implemented WP2 features**: Real-time Human Assessment Model

1. He, J., Li, K., Liao, X., Zhang, P., & Jiang, N. (2019). Real-time detection of acute cognitive stress using a convolutional neural network from electrocardiographic signal. *IEEE Access*, *7*, 42710-42717.

Algorithm #2: Input data

Data is collected by a wearable device with ECG data streaming to mobile App (video with screen casting of (simulated) ECG streaming). The mobile App shares the data to the data endpoint which is then retrieved by the Real-time Implementation algortihm (#2).



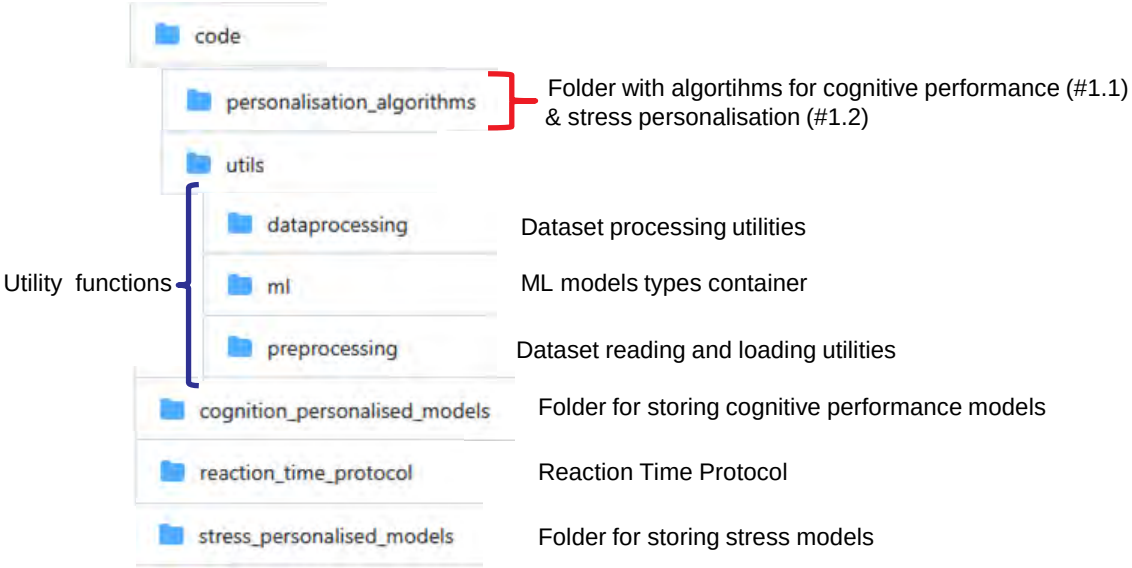
https://drive.inesctec.pt/s/YxLfyZsq9EtNEPj

Algorithm #2: Output data

https://drive.inesctec.pt/s/FSMWPiBQbR2aP8B

- Worker's ID insertion to load their personal model
- Retreving data from the server
- Wait until a 1 minute ECG buffer is obtained
- Signal quality check
- Feature computation
- Cognitive Performance Estimation and Stress Detection
- Posting results on the server

Overview of code structure



Link to repository

https://github.com/AI4REALNET/Human-Assessment-Module

Authors

Authors	Institution
Federico Calà	INESC TEC
Duarte Dias	INESC TEC
Adriana Arrais	INESC TEC
João Paulo Cunha	INESC TEC

Domain Transparency in Airspace Sectorization

Delft University of Technology (TUD)

Outline

- Context
- Methodology
- Original Contribution
- Overview of repo structure

Context: ATM use case on sectorization

Dynamic Airspace Sectorization (DAS): "a continuous restructuring of airspace sectors that ensures efficient allocation of scarce resources (e.g., Air Traffic Controllers) considering operational, economic and ecological constraints in both nominal and variable air traffic conditions." (Gerdes *et al.*, 2018)

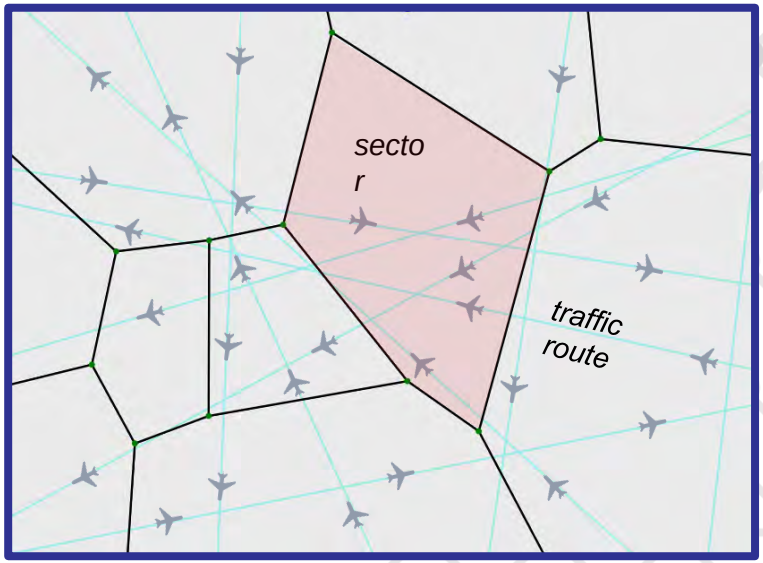
Human Role: flow and capacity manager

Objectives:

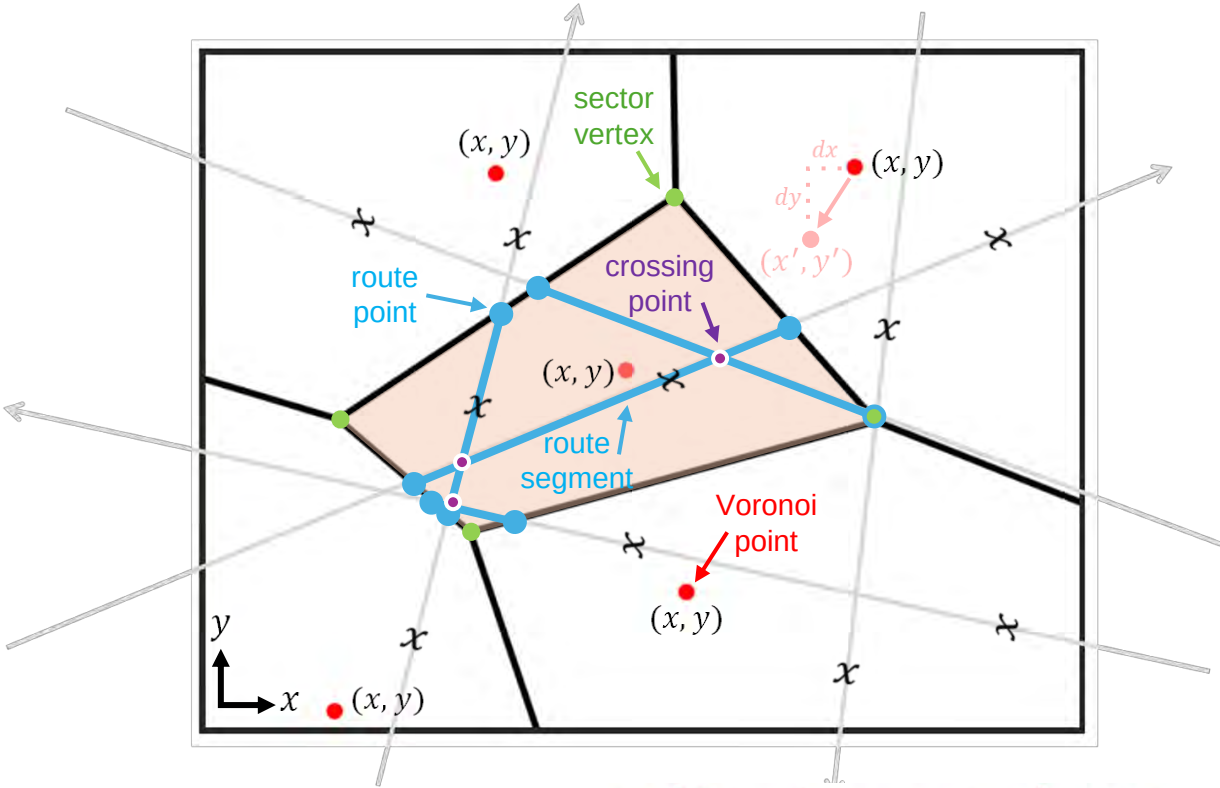
- Keep predicted controller workload within acceptable bounds (min/max)
- Balance predicted workload across sectors (low standard deviation)
- Minimize the number of handover points (coordination workload)

Operational constraints:

- Number of airspace sectors equal the number of available controllers
- Ensure convex airspace sectors
- Sectors should not be too small
- Avoid route crossings too close to sector boundaries
- Ensure sufficient route lengths within sectors
- Avoid route segments to coincide with sector vertices
- Avoid shallow crossing angles between routes and sector boundaries



Methodology: Voronoi Airspace Tessellation



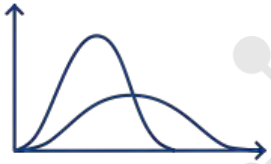
The airspace can be partitioned in Voronoi cells, where each cell is an airspace sector under the control of an Air Traffic Controller (ATCO).

Per sector, predicted ATCO taskload can be calculated using planned traffic routes, allowing determination of:

- Sector entry and exit times
- Aircraft occupancy over time
- Constraint violations (e.g., too short route segments, route crossing too close to sector borders, etc.)

Methodology: environment modelling

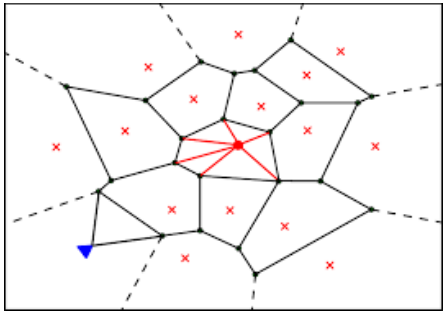
- **Trajectory generation:**
  - Interpolation of flight plan data (geometric calculations)
  - Historical flight data (distributions on FIR airspace entry/exit times)
  - Incorporate trajectory uncertainty (uncertainty distributions on position, arrival times and flight speed)



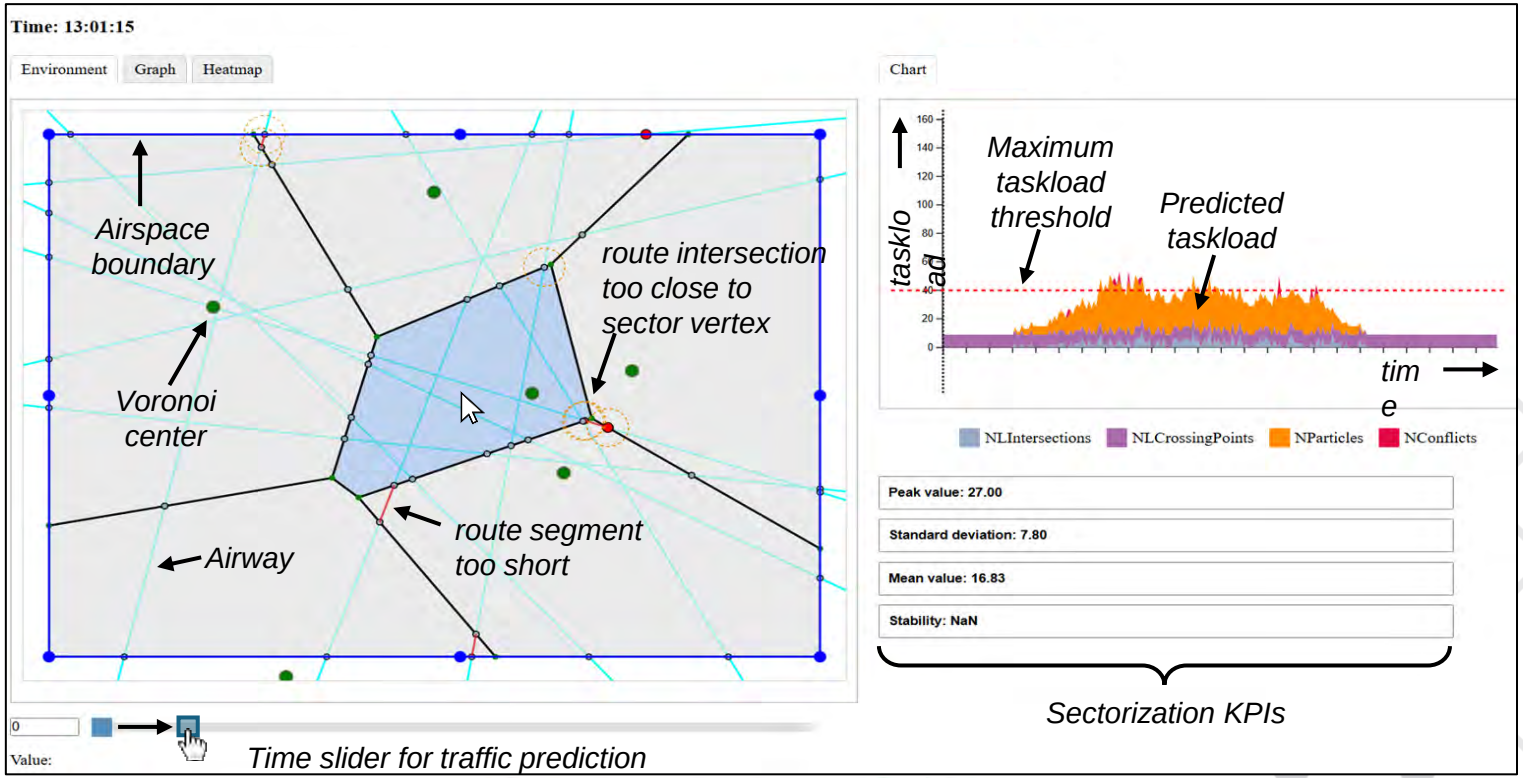
- **Taskload prediction**: (weighted) sum of complexity (CX) factors
  - CX_1: Number of route crossings
  - CX_2: Number of coordination points (= route-airspace crossings)
  - CX_3: Number of predicted flights per sector, per time slice
  - CX_4: Number of predicted conflicts per sector, per time slice
  - …

$$TL = \sum w_i \cdot CX_i$$

- **Convex sector geometry**:
  - Voronoi partitioning
  - Fortune's algorithm (fast!)
  - Number of centers = number of available controllers



Original contribution: HMI for sectorization



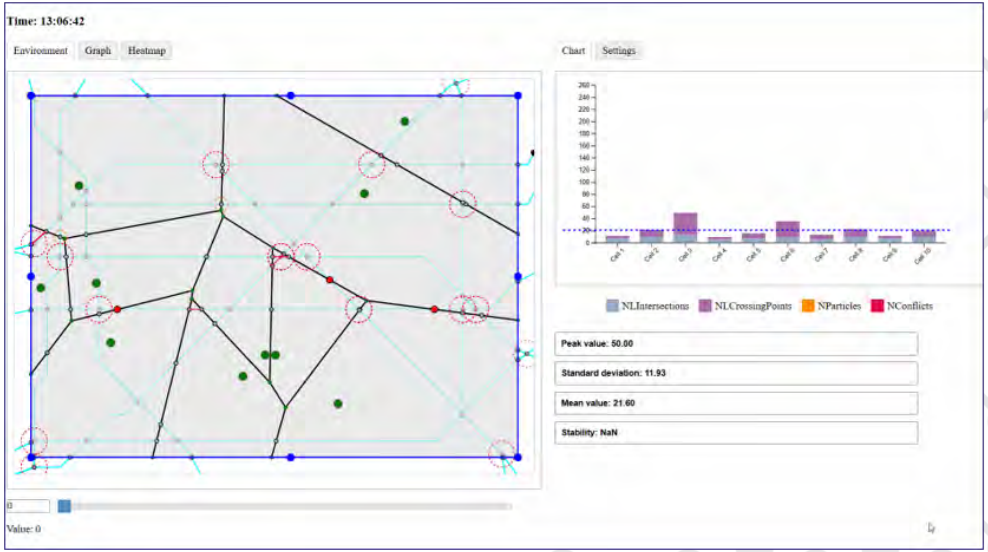
Original contribution: Domain Transparency

Ecological approach: identify and visualize system operational envelope and constraints for domain transparency (independent of human or AI agents performing actions).

Direct Manipulation Interface reduces (cognitive) distance between information processing and user's intentions through (direct) engagement.

Designed for a (generic?) sectorization/ partitioning context.

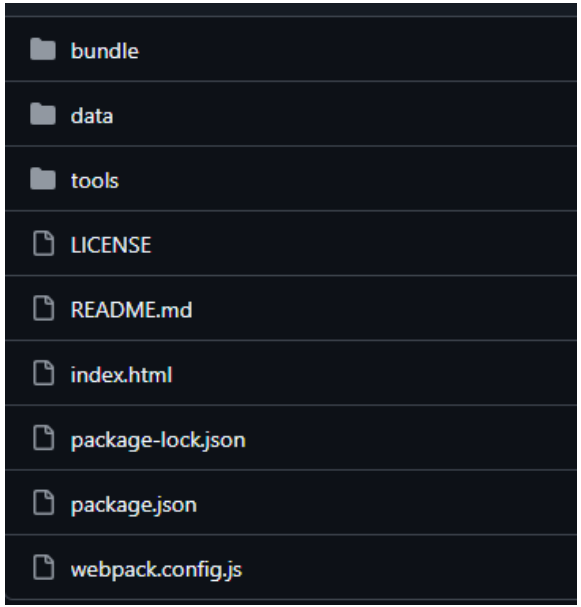
Helps with understanding and monitoring AI agents performing actions in WP3 (T3.3) and WP4 (T4.3).



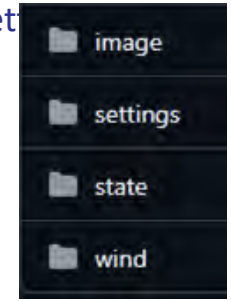
Overview of the repository

JavaScript source

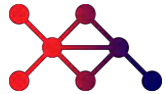
Main HTML file to open in browser



Data files & set



Authors



Authors	Institution
Clark Borst	Delft University of Technology

Link to the repository

edu.nl/k7jvb

Transparency trough Tokens

University of Applied Science Northwestern Switzerland (FHNW)

Outline

- Context
- Methodology
- Contribution
- Examples of token interactions

Context

Definition

Token Communication between Multi-Agent Systems refers to a structured mechanism where agents exchange discrete tokens encoding intent, priority, and resource states. In the implementation, tokens formalize negotiation steps, making interactions explicit and traceable. This enables clear analysis of coordination, conflict resolution, and decision-making. The approach improves transparency and understanding of agent negotiation in railway simulations

Motivation

We employ **token-based communication** within the multi-agent railway simulation to enable a formally structured and analyzable representation of inter-agent negotiation processes. Tokens encapsulate negotiation primitives such as intent, priority, and resource allocation states, thereby making interaction dynamics explicit and machine-interpretable.

This approach facilitates a **fine-grained examination of coordination, conflict resolution, and decision-making** strategies among agents under operational constraints. Consequently, it enhances the interpretability and systematic analysis of negotiation behavior in complex, distributed railway systems

Context: Railway conflict negotiation

Why negotiation is organized through tokens

Railway disruptions generate many local conflicts over limited junction, track, and time-slot capacity. Negotiation-based mixed-initiative control addresses this by letting AI resolve operational conflicts as they emerge, while human operators retain authority at the policy level. Token communication provides the formal language for this process: it represents requests, priorities, and constraints in a discrete form that is both operationally useful and analytically transparent.

Core idea

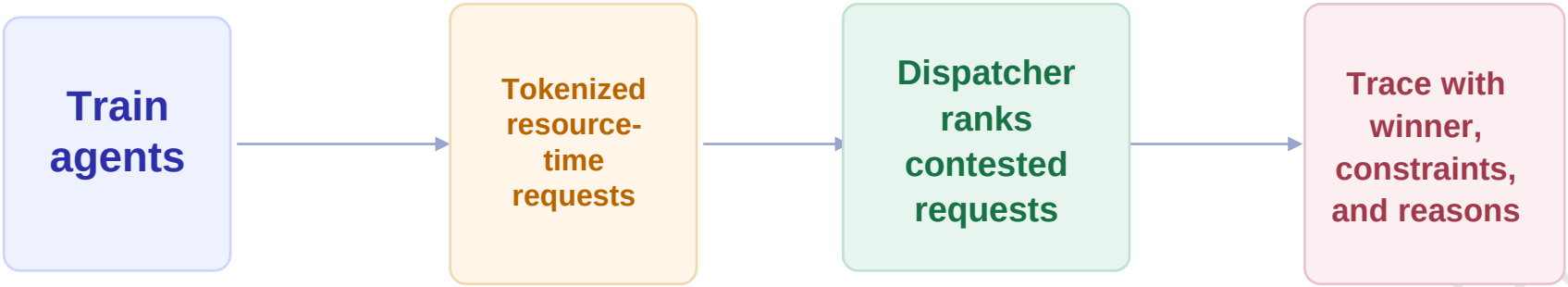
Local conflicts are easier to resolve than a brittle global re-plan with tokens.

Mixed-initiative control separates human governance from algorithmic execution and connect them at the same time.

Tokens make (conflict) negotiation explicit, traceable, and machine-interpretable.

Methodology: Communication trough tokens

Machine-level tokens in the MARL negotiation loop



a. Each agent first proposes an action; contested requests are expressed as discrete tokens tied to a resource and time window.
The dispatcher evaluates requests with explicit components such as priority, delay proxy, fairness loss, and anti-starvation state.
Because allocation is deterministic, every outcome can be reconstructed from the negotiation trace rather than inferred post hoc

Linking human primitives to machine tokens

The same architecture connects human directives and automated negotiation.

Human- Machine primitive (Token)

Ordered context tokens define the meaning and syntax of the directive.

Negotiation proxy

- interprets directive type
  maps tokens to policy parameters
  passes bounded changes into negotiation

Machine-level effects

- weight adjustments fairness or priority constraints
- targeted overrides or routing bias

Example: a human directive such as **“[Prioritize] [regional] [trains]”** is encoded as a *primitive* and translated into *machine tokens* that modify the negotiation outcome without forcing low-level manual control.

Decision Transparency

Why tokenization improves understanding and governance

Helps with understanding and monitoring AI agents performing actions in WP3 (T3.3) and WP4 (T4.3).

Interpretability

“Step-wise” JSON transcripts and compact reason codes show which requests competed, which constraints were active, and why one allocation won.

Governance

Operators steer the system through policy settings, safeguards, and high-level directives rather than manual conflict-by-conflict intervention.

Guardian supervision layer

- monitors congestion, no-move streaks, and unfair outcomes
- applies bounded policy patches when risk rises
- preserves deterministic negotiation behavior

System insight

A token language shared across humans, agents, and supervision makes the MARL system observable, steerable, and easier to trust in safety-critical settings.

Tokens bridge human intent and machine coordination.

Hybrid Planning with Token-Based Transparency

Goal

Transparent and controllable decision-making

Problem

Why is transparency needed? Multi-agent systems are complex. Decisions are often hard to understand and difficult to control. E.g.: Why did one train wait and another move?

Planning decisions appear as a black box

Approach

Making planning transparent with Tokens. Introduce Tokens as human inputs, they represent: clear, explicit decisions. E.g. :PRIORITY → “this agent should go first” WAIT → “this agent should pause”. Tokens express human intent in a structured way.

Transparency through explicit interaction

- System behavior is not hidden:
  - changes are triggered by explicit Tokens
- Clear cause → effect:
  - Token → Replan → Changed behavior
- Human can always answer:
  - “Why did this agent move?” → because of a specific Token
- **No unexplained system decisions**

Co-Learning Approach

Transparent Learning through Human Interaction

- Learning through Human Interaction
- System behavior is shaped continuously
- Human interaction directly influences future behavior
- The system is not fixed, it adapts over time, based on human guidance

Transparent Learning Process

- Why learning is understandable:
  - changes do not happen internally
  - changes happen during interaction
- You can observe:
  - when behavior changes
  - what caused the change

Co-Learning Approach

Learning via feedback loop:

- Human interacts
- System adapts
- Adaptation affects future behavior

- Not a one-time decision: but a continuous refinement process

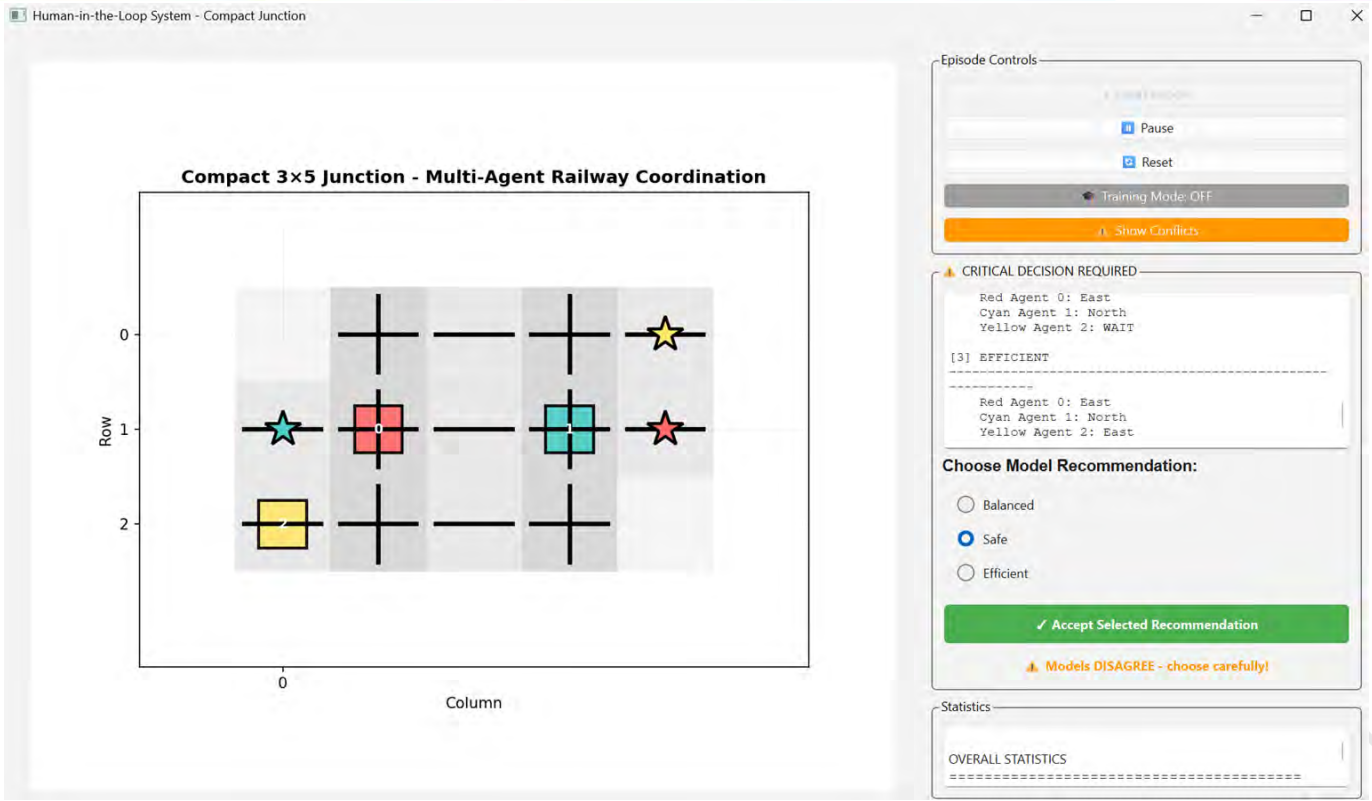
Human provides:

- Corrections
- Preferences
- Guidance

- System behavior reflects accumulated human input over time

- **Transparency of learning, not just actions**

Co-Learning Approach: Example



Tokens in Hybrid Planning

Hybrid planning combines:

Conflict-Based Search (CBS)
→ computes an initial conflict-free plan
→ ensures a consistent and robust baseline

Prioritized Planning (PP)
→ enables fast replanning after changes
→ reacts efficiently to human input

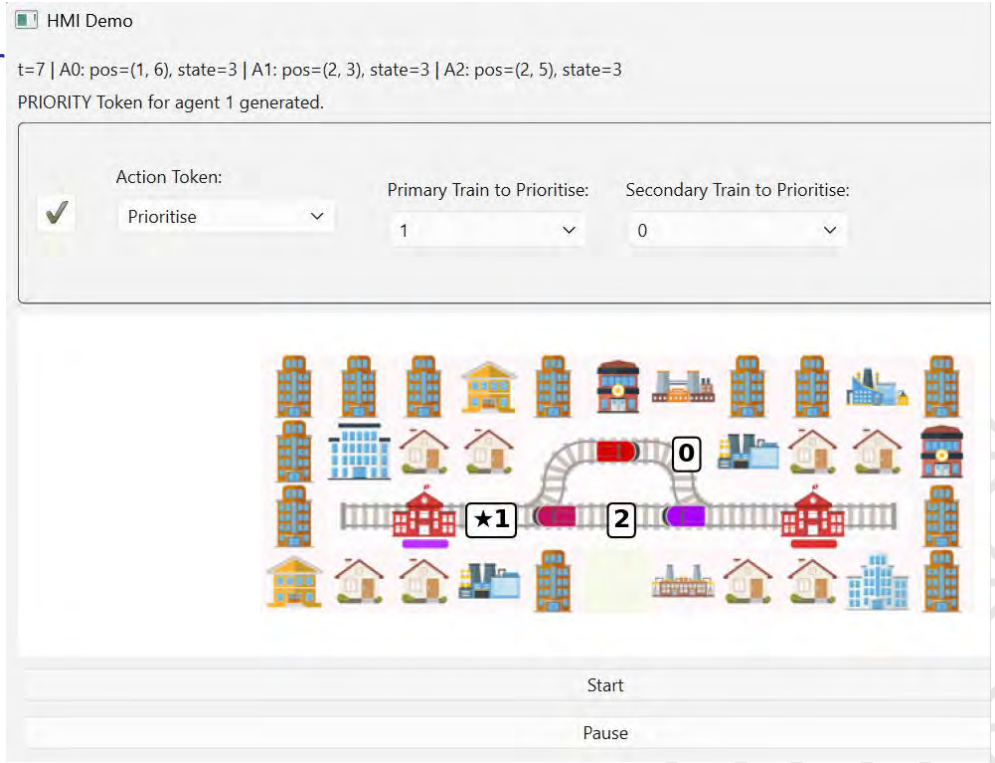
Tokens trigger replanning: system adapts quickly using PP, while maintaining a CBS-based structure

Workflow

From Human Input to System Behavior

1. Initial plan computed using CBS
2. Simulation runs
3. Human applies Token
4. System triggers replan()
5. PP performs fast adaptation
6. Updated behavior becomes visible

Fast response + stable baseline



In the example on the right, Train 1 was prioritized (marked by the star)

Code structure: Hybrid Approach

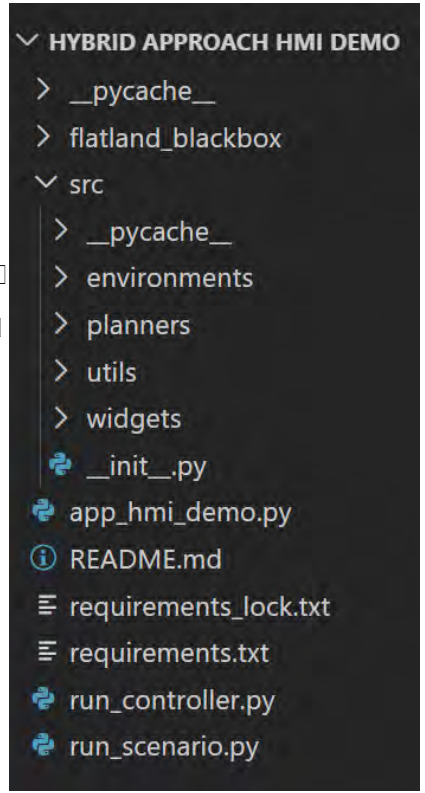
External Blackbox planner (CBS / PP implementation)

Scenario loading and environment setup

Hybrid planning using CBS and PP

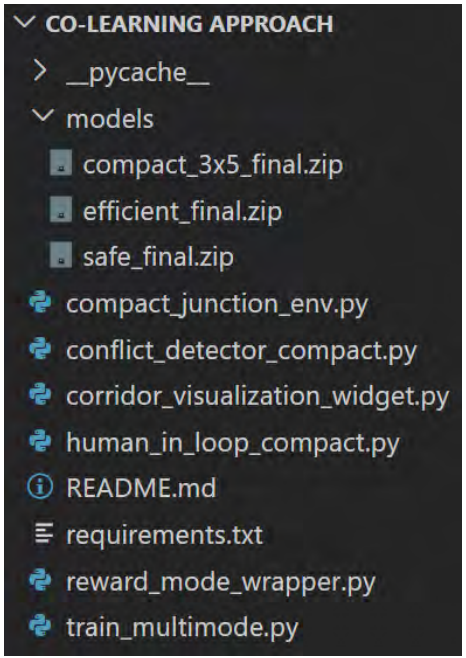
Token-based user interaction and control

Controller and simulation execution



Helper functions and environment utilities

Code structure: Co-Learning Approach



Pretrained models for different training objectives

Conflict detection and decision support □

Human interaction and intervention logic

Multi-agent learning and reward handling

□ Simulation environment

Authors

Authors	Institution
Manuel Renold	FHNW
Julia Stadelmann	FHNW
Janick Marti	FHNW

Link to the repository

Github Repo : https://github.com/AI4REALNET/Tokener

AI4REALNET

AI4REALNET has received funding from European Union's Horizon Europe Research and Innovation programme under the Grant Agreement No 101119527 and from the Swiss State Secretariat for Education, Research and Innovation

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.