AI fundamental blocks – beta release

Deliverable D2.2

Distribution level: Public
Version: 1.0
Date of delivery: 29/03/2025

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.

List of contributions

- Politecnico di Milano (POLIMI)
- Fraunhofer Institute/ University of Kassel (FHG/UKASSEL)
- INESC TEC
- IRT SystemX (IRTSX)
- University of Amsterdam (UVA)
- enlite AI

Distributed RL

Main developer: POLIMI

Outline

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

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

Collect **large number of samples** from the environment → Find **subproblems** = state/action variables that can be used to predict new state variables → Extend **state-of-the-art algorithms** to solve each subproblem with limited communication

Find subproblems ↕ Use **information-theoretic measures** (e.g., mutual information)

Extend state-of-the-art algorithms ↕ **Distributed/Hierarchical RL**

- **Preliminary experiments** on Grid2Op, Flatland

Original contribution

- **Algorithm #1 |** State and Action Factorization (SAF)
  - ➢ **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

- **Algorithm #2 |** Distributed Q-learning (DQL)
  - ➢ **Short description:** Distributed version of the standard Q-learning algorithm
  - ➢ **State-of-the-art:** Other distributed RL algorithms[3]
  - ➢ **Contribution:** Proposed specific implementation for turn-based MDPs
  - ➢ **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)

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

Overview of repository structure

The code is composed of two different folders, each containing one algorithm

- 📂 **distributed_q_learning** contains the code for the Distributed Q Learning algorithm (DQL)
- 📂 **state_action_factorization** contains the code for the State and Action Factorization algorithm (SAF)

A description of how to use the code is provided separately in the README file of each folder.

Authors

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

Algorithm #1
State and action factorization

State and Action factorization

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

$$
\begin{array}{c}
\begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & a_1 & a_2 & a_3 \end{array} \\
\begin{array}{c} s'_1 \\ s'_2 \\ s'_3 \\ s'_4 \\ s'_5 \end{array}
\left[\begin{array}{cccccccc}
 & & & & & & & \\
 & & \square & & & & & \\
 & & & & & & & \\
 & & & & & & & \\
 & & & & & & &
\end{array}\right]
\end{array}
\qquad \square = \mathrm{MI}\left(s'_2, s_3\right)
$$

(estimated on the dataset $\mathcal{D}$)

3. Transform it into a pseudo block-diagonal matrix. Each block is an independent MDP

Overview of code structure

```
state_action_factorization
├── cluster              → block diagonalization procedure
│   └── ...
├── extract_data         → extract dataset from Grid2Op
│   └── ...
├── grid2op_patch        → patch for Grid2Op
│   └── ...
├── mutual_information   → mutual information estimator
│   └── ...
├── main.py              → run factorization on data extracted from Grid2Op
```

Input data

- String identifying Grid2Op environment
- number of episodes and number of samples used to extract data from Grid2Op
- threshold for the adjacency matrix

Output data

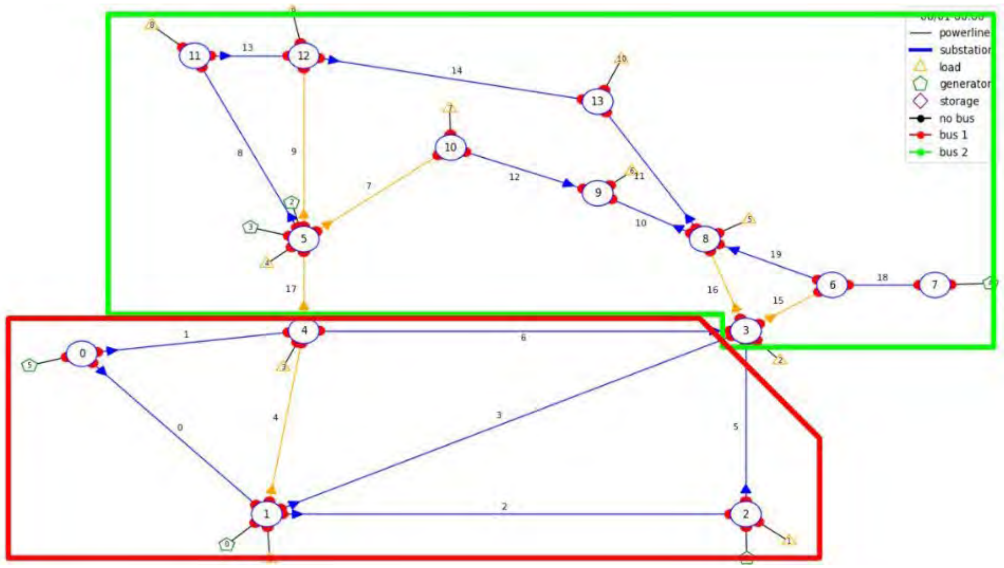
- one folder for each substation containing the data extracted from the Grid2Op environment

- three different versions of the adjacency matrix (original, shuffled, unbiased)

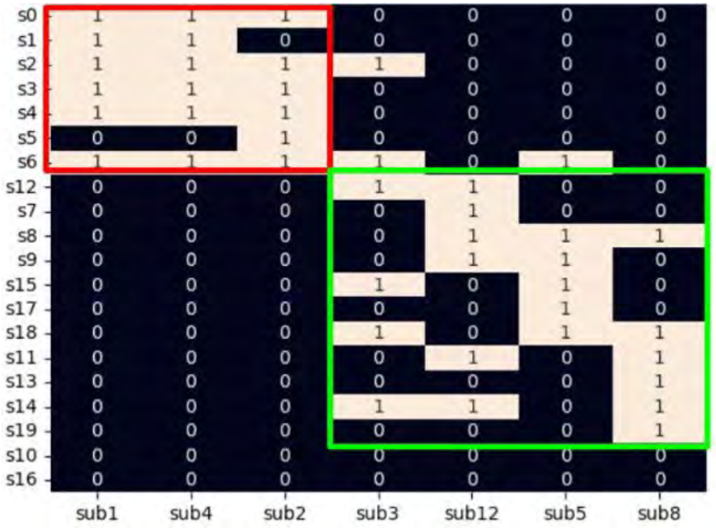
- a plot of the final diagonal matrix in which blocks corresponds to MDPs

Experiment

- Tested on Grid2Op, environment IEEE case14 benchmark (14 substation, 20 lines, 6 generators, 11 loads)



state variables = lines capacity



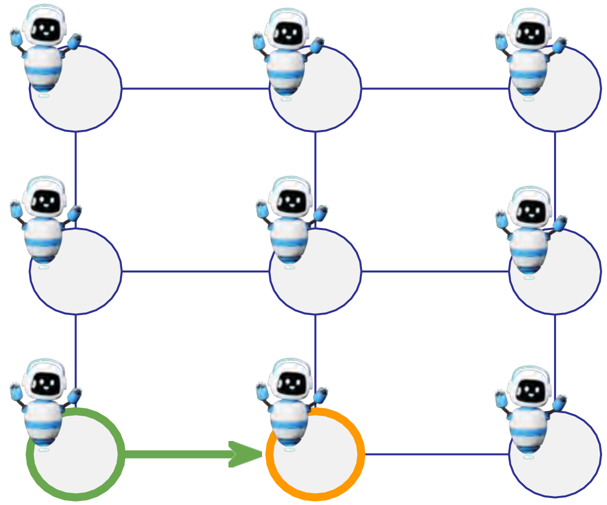
action variables = valid substation topology changes

- The grid is split into two parts. The results are the same as the domain-expert analysis [Marot et. al., Guided machine learning for power grid segmentation, 2018]

Algorithm #2
Distributed Q-learning

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]

Overview of code structure

```
distributed_q_learning
├── flatland tools      ──→ contains a wrapper to Flatland
│   └── ...
├── training _utils     ──→ auxiliary functions for training
│   └── ...
├── plot                ──→ script for plotting the training curve
│   └── ...
├── train.py            ──→ train the DQL algorithm (on Flatland)
```

Input data

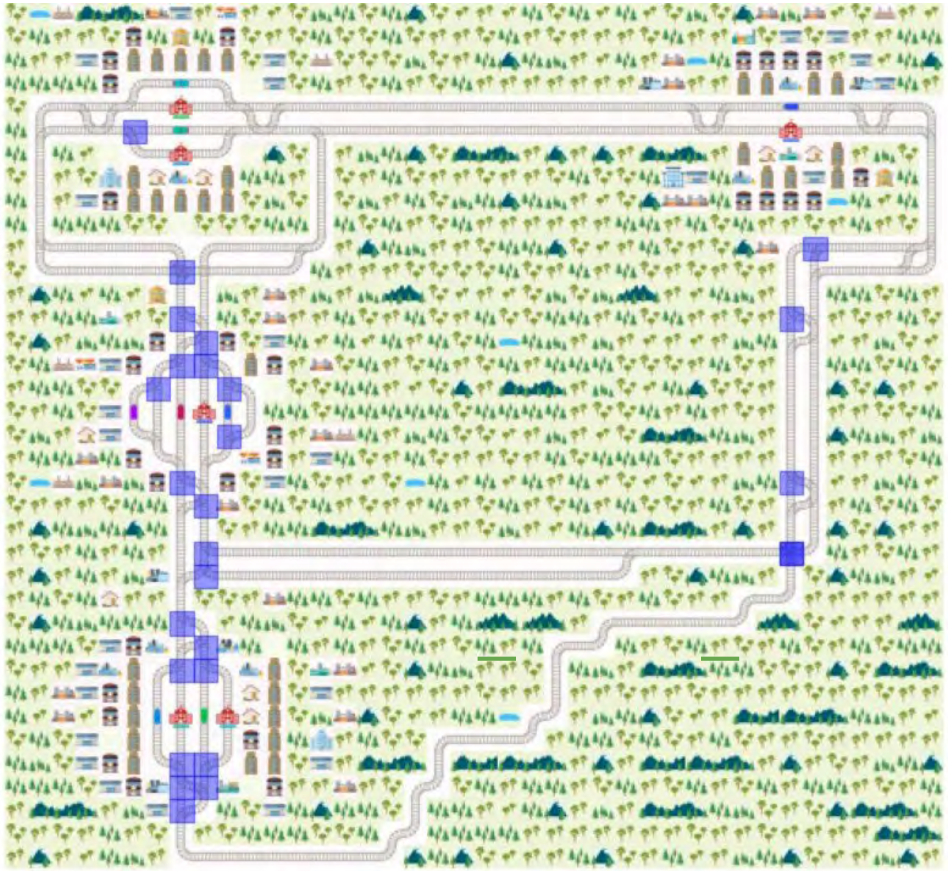
- Parameters of the Flatland environment (grid size, number of trains, …)
- Hyperparameters of the DQL algorithm (learning rate, epsilon decay, …)

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 log file and the seeds.

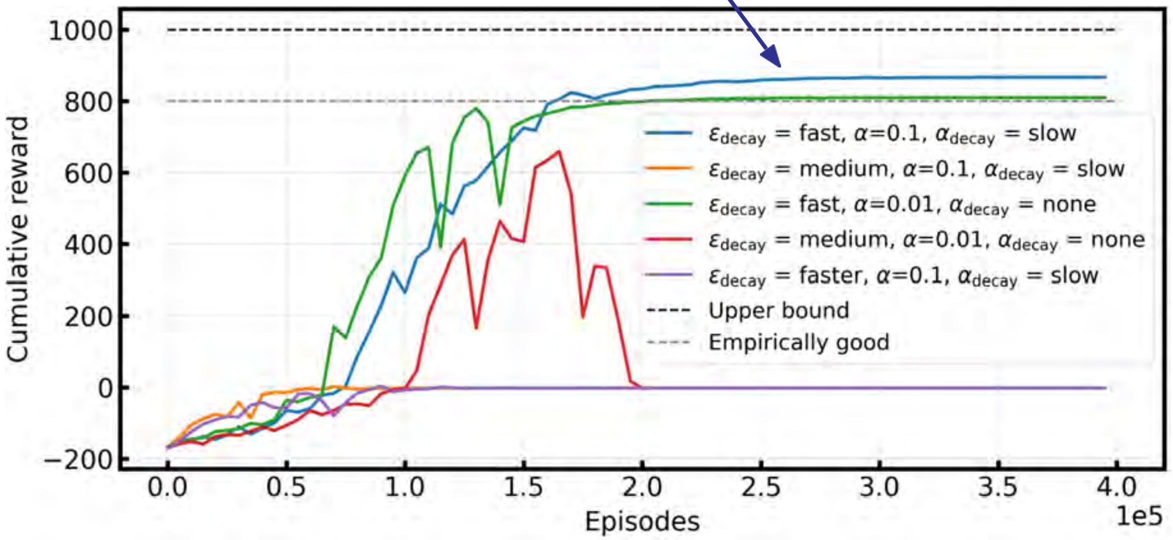
- a global log file containing the computation time and the configurations of each experiment.

Experiments



one agent for each junction node

- Tested on Flatland grid 40x40 with 5 trains, 7 stations
- Hyperparameters: epsilon decay, learning rate, learning rate decay
- Convergence to optimal solution, i.e., all the trains arrive at destination with no delay.



Link to the repository

https://github.com/AI4REALNET/distributed_rl

Explainable and Transparent Failure Prediction of Agents

Main developer: Fraunhofer/UKASSEL

Authors

Authors	Institution
Mohamed Hassouna	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 oberservations and the agent index/type
- Identify critical features and regions to enhance the global interpretability of grid failures.

Overview of code structure

models

- Metrics.ipynb
- README.md
- cemc_final_metrics.py
- cemc_hyperparam.py
- gandalf_final_metrics.py
- gandalf_hyperparam.py
- lightgbm.ipynb
- lightgbm_hyperparam.py
- rf_hyperparam.py
- xg_hyperparam.py

Evaluation of Boosting models

Evaluation of neural network models

Training/Hyperparameter Tuning

Analysis of the best model (LightGBM) + Extraction of feature importances + plots

models

- dataset_colnames.npy
- lightgbm_new.pkl

Feature names

Best LightGBM model checkpoint

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
- Mutliple 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

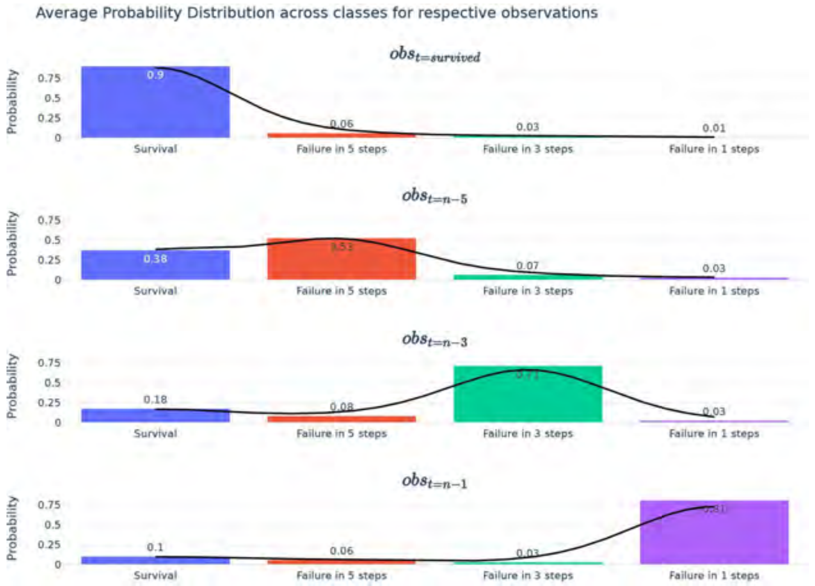
- 2 Neural Network Architectures: Feed-forward NN, GANDALF [2]
- Neural Network models underperform, LightGBM performs best.

□ 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

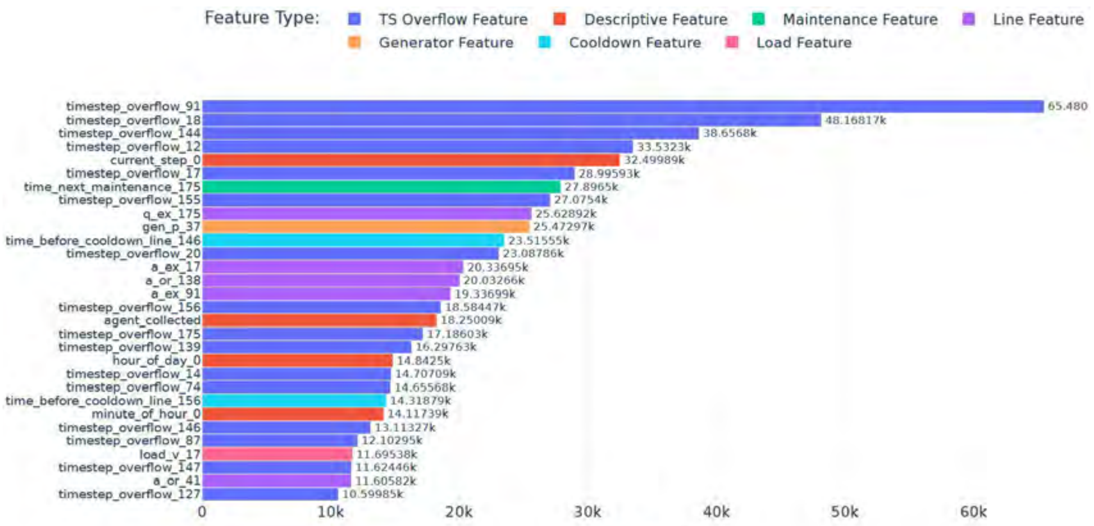
- Used the gain metric to measure how much each feature improves accuracy.
- Helps understand key patterns and relationships in the data.

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.

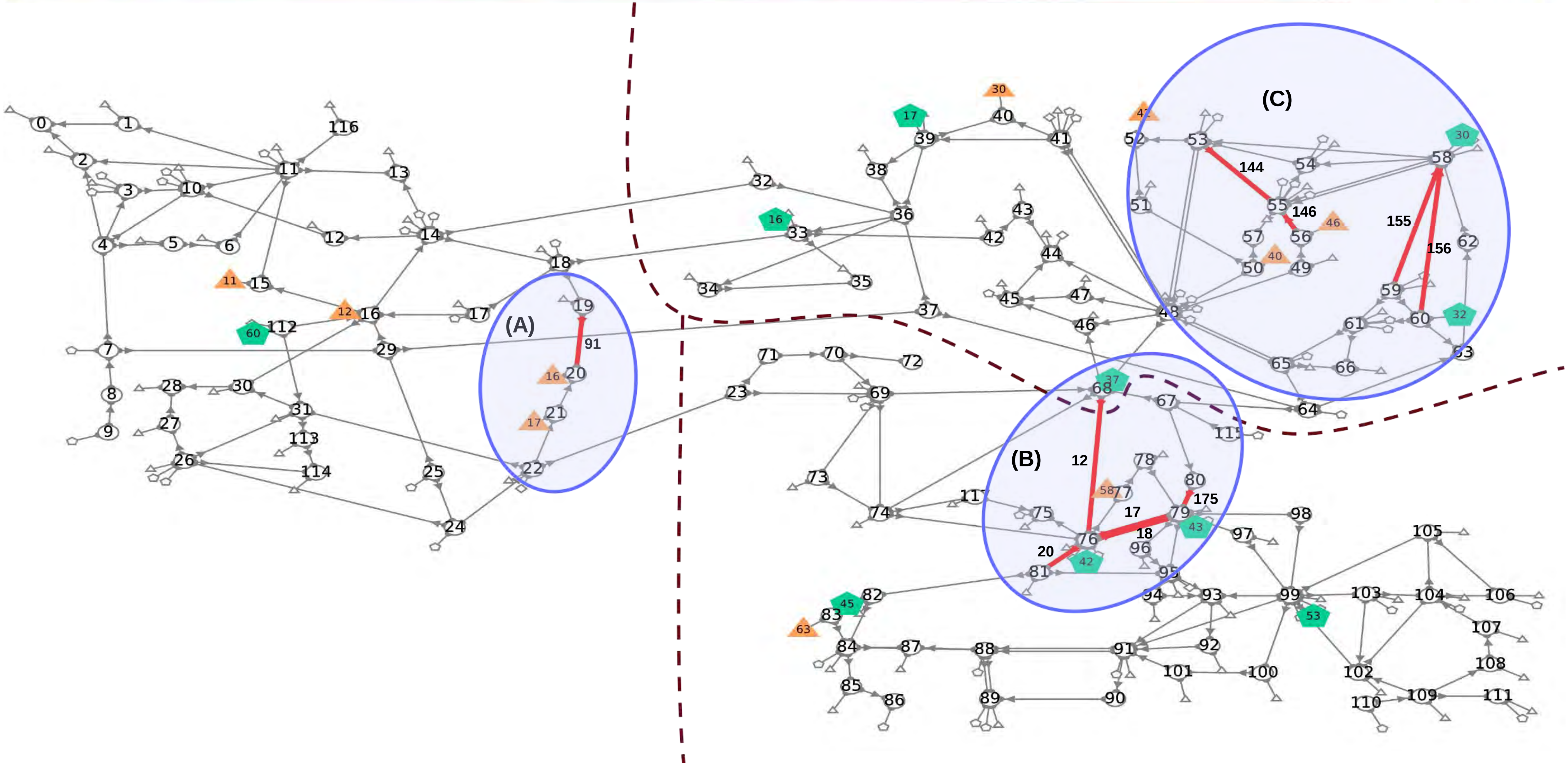
Dominance of line features

- 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 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

Link to the repository

https://github.com/AI4REALNET/failure_prediction

Human Assessment Model

Main developer: INESC TEC

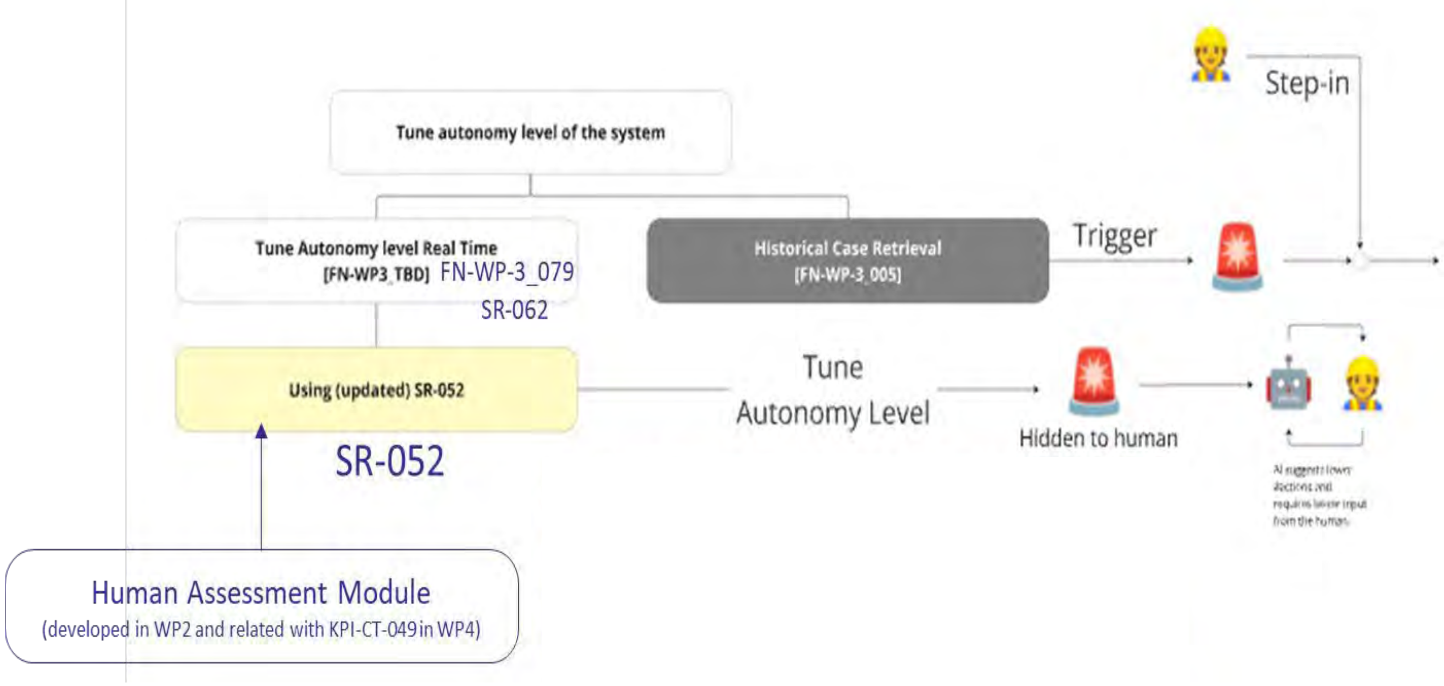
Authors

Authors	Institution
Duarte Dias	INESC TEC
Adriana Arrais	INESC TEC
João Paulo Cunha	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 the level of autonomy of the system in a seamless and implicit way – without interacting with the operator.



Context (2/2)

Motivation

If the system automatically, without any direct intervention of the human (implicit interaction), is able to adapt itself to the operator we believe it can contribute for a higher level a empathy and trust with the AI system, leading to a higher acceptability.

Use cases

Power grids, railway networks and air traffic networks can be integrated with HAM. It is not depedent of the use case, only on which use cases the tune autonomy level system will be integated.

Methodology

Main idea = <u>Create a Human Assessment Model</u>

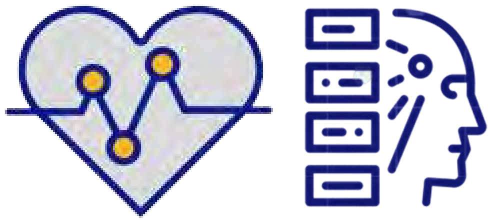
↓

The creation of a personalised model is essencial, since response to stress differs between individuals.

Develop the **personalisation module** where prediction of cognition and stress from physiology is personalised

→

Real-time application of **Human Assessment Module**



- **Preliminary experiments:** Development and optimization of cognitive performance assessment based on ECG features. Dataset from 11 Air Traffic Controllers was used from previous data collection (2018, Rodrigues et al.).

Original contribution (1/3)

- **Algorithm #1 | Personalised Human Assessment Module**

Algorithm #1.1 | Quantification of Cognitive Performance

- ➢ **Short description:** Personalised regression to quantificatify cognitive performance from electrocardiogram.
- ➢ **State-of-the-art:** Modulation of cognition in ill individuals [1].
- ➢ **Contribution:** Novel method to modulates cognitive performance from ECG physiology.
- ➢ **Implemented WP2 features:** Human Cognitive Performance Assessment.

Algorithm #1.2 | Classification of Stress Level

- ➢ **Short description:** Personalised classification of stress from electrocardiogram.
- ➢ **State-of-the-art:** Non-personalised modulation of stress [1, 2]
- ➢ **Contribution:** Novel method to modulates stress from ECG physiology on working environments.
- ➢ **Implemented WP2 features:** Human Stress Assessment.

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)

Original contribution (2/3)

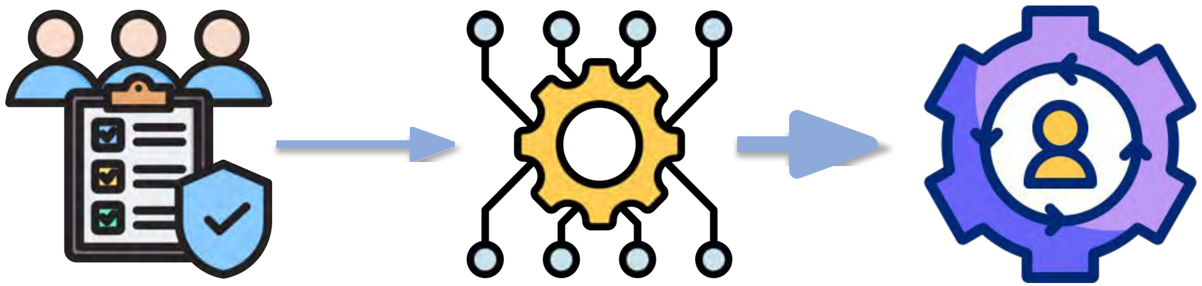
- **Real-time Implementation | ** Algorithm real-time implementation
  - ➢ **Short description:** Real-time implementation of human assessment – cognition and stress.
  - ➢ **State-of-the-art:** There are no solutions that use ECG to predict cognition. Other approached have more complex data gathering (Ex. EEG) [3, 4] .
  - ➢ **Contribution:** Novel method to modulates cognitive performance and stress from ECG physiology in real-time, in working environment.
  - ➢ **Implemented WP2 features:** Human Cognitive Performance and Stress Assessment.

3 Cogwear. (2023). Cognitive state assessment using wearable EEG technology. Retrieved February, 2025, from https://cogweartech.com/

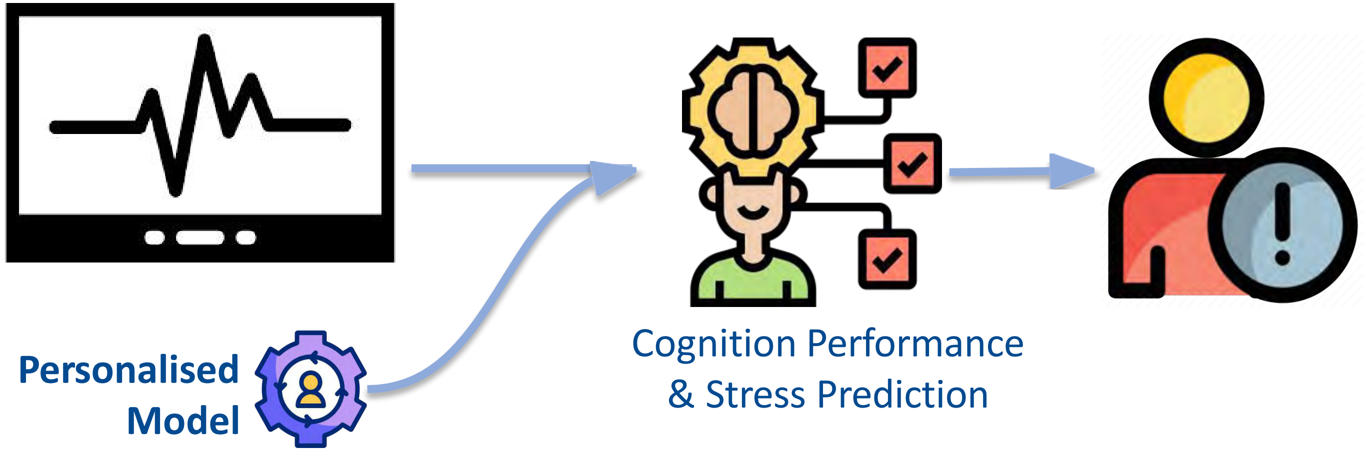
4 EMOTIV. (2025). How EEG tech aids workplace wellness. Retrieved February, 2025, from https://www.emotiv.com/blogs/news/workplace-wellness-using-eeg-technology

Original contribution (3/3)

Algorithm #1



Real-time Implementation

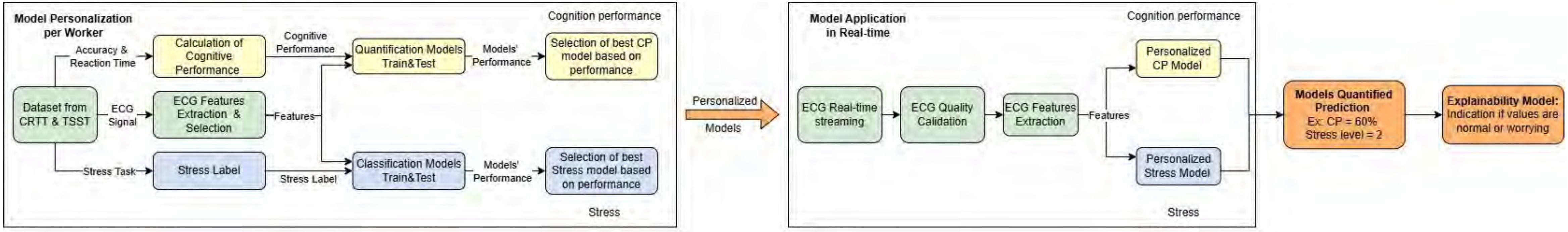


Block Diagram – Generic Overview (1/3)

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

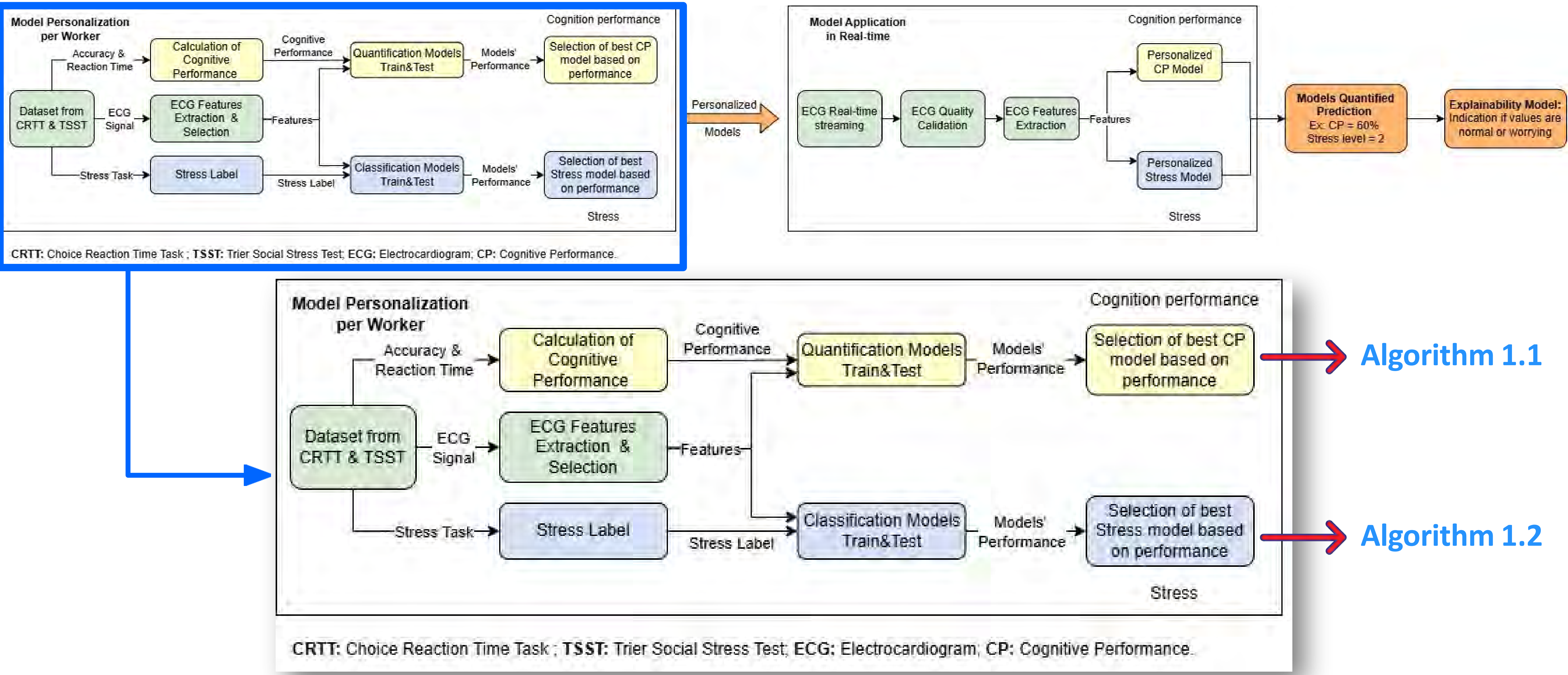
The implementation is divided in two main phases:

1. Model Personalisation (Algorithm #1): Modelling of cognition of stress for each individual.
2. Model Implementation (Real-time Implementation): Real-time implementation of model.

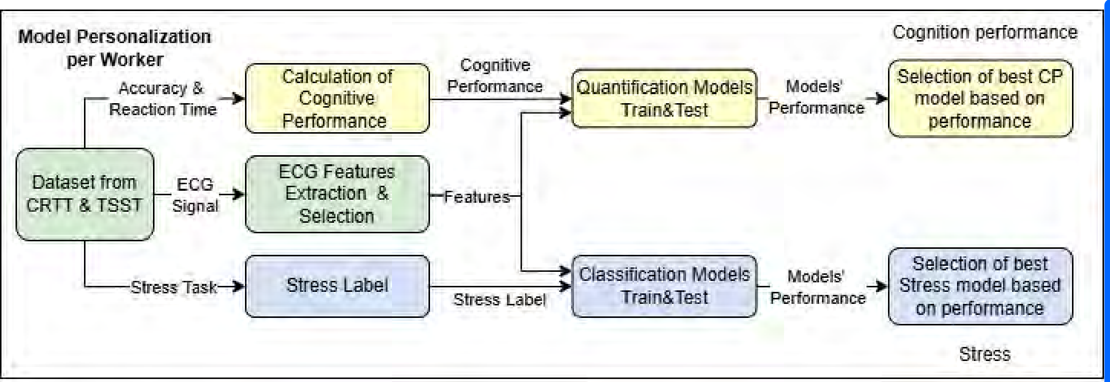


CRTT: Choice Reaction Time Task ; **TSST:** Trier Social Stress Test; **ECG:** Electrocardiogram; **CP:** Cognitive Performance.

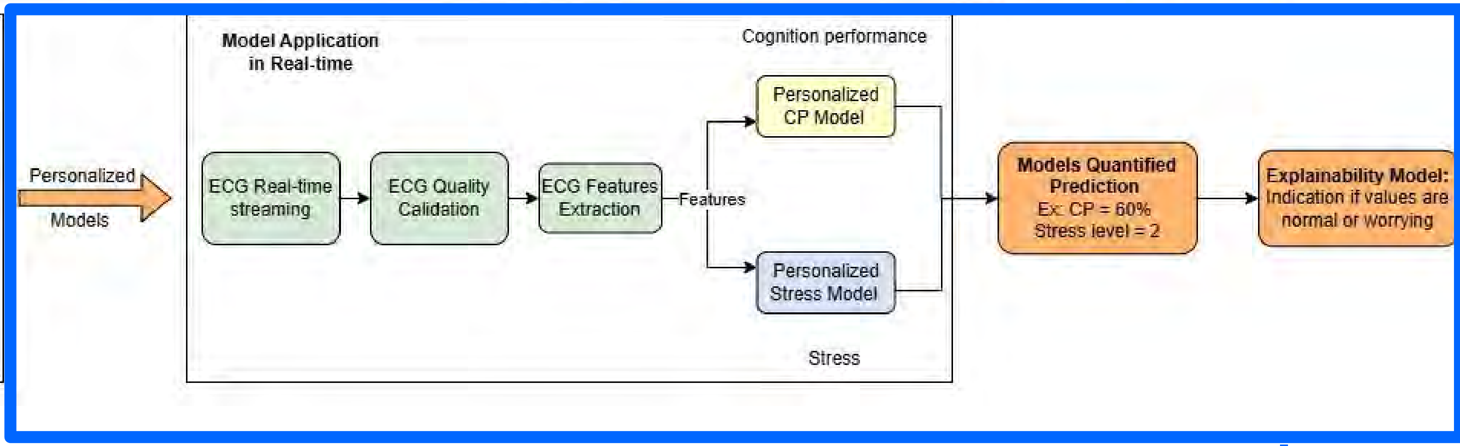
Block Diagram – Algortihm #1 (2/3)

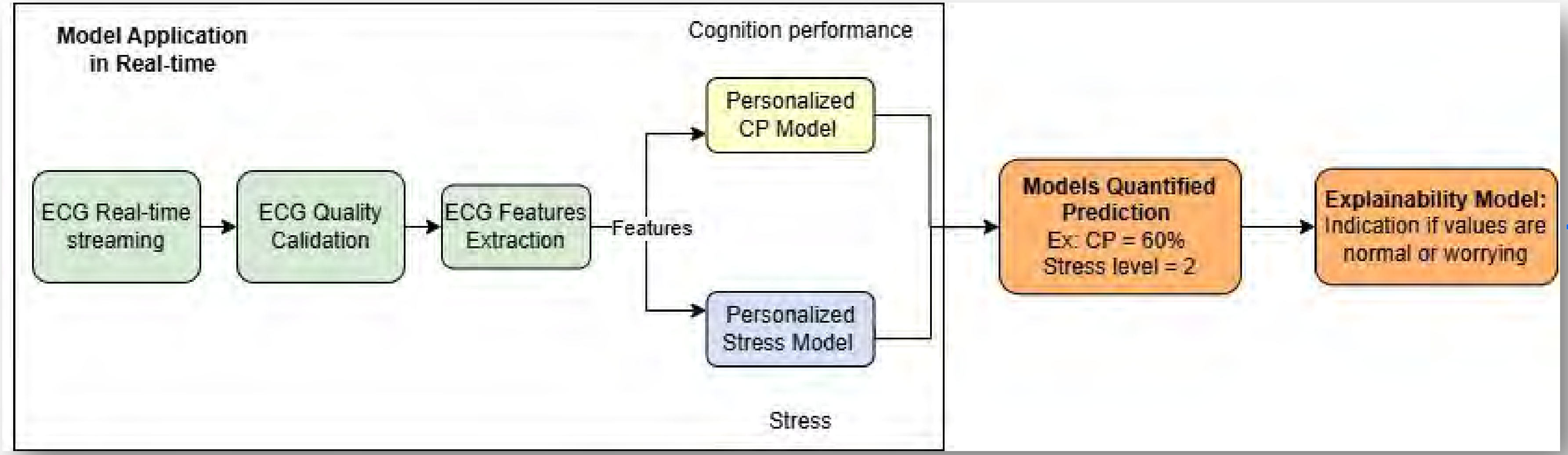


Block Diagram – Real-time Implementation (3/3)



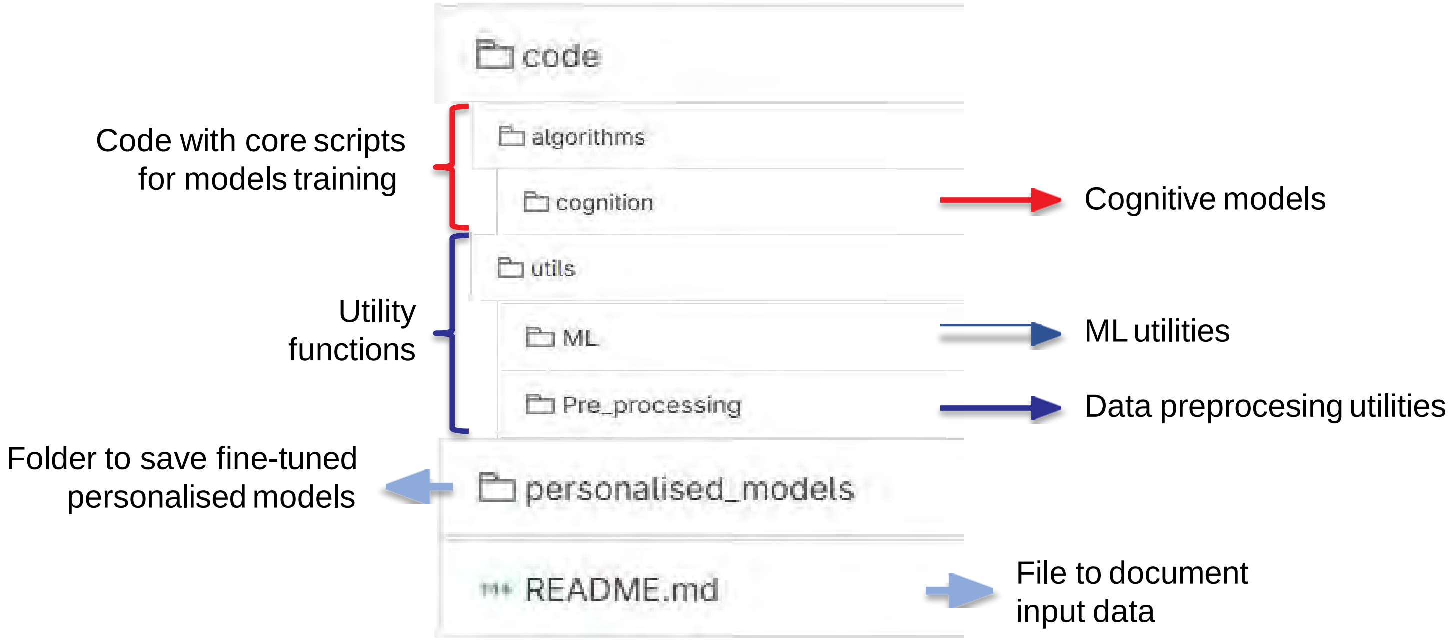
CRTT: Choice Reaction Time Task ; TSST: Trier Social Stress Test; ECG: Electrocardiogram; CP: Cognitive Performance.





Real-time Implementation

Overview of code structure



*** Source code is under analysis for license and sharing on project repository – estimation for sharing is mid-March.**

Input data (Algorithm #1)

- These raw datasets were created from data gathered following a specific protocol described in Rodrigues et al. (2018)
- **Tests Performed:**
  - **Baseline**: Sit comfortably for 10 minutes.
  - **2-Choice Reaction Time Task (CRTT)**: 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 recorded.
  - **Trier Social Stress Test (TSST)**: An acute psychosocial stress paradigm.
- **Task Order: Baseline ⟶ CRTT1 ⟶ TSST ⟶ CRTT2**
- Psychological scale assess
- File Format:
  - File Type: .json

Keys	Description
ECG	Raw electrocardiogram (ECG) signal.
VAS	Visual Analogue Scale rating (self perception of stress).
STAI_6items	State-Trait Anxiety Inventory (6-item version) score, measuring anxiety levels.
Right_ans	Correct responses.
Answers	Responses provided by the participant.
Answer_reaction_time	Time taken (in seconds) to respond.
Answer_timing	Time when the visual stimuli was presented (initiating in the beginning of each CRTT task).

Output data (Algorithm #1.1)

```
Selected Features:
['nni_20', 'cvnni', 'sdnn', 'std_hr', 'lf']

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: -0.02528120940270484
RMSE: 0.27385694596104804

linear_regression
-------------------
Fitting 2 folds for each of 2 candidates, totalling 4 fits
Best hyperparameters: {'fit_intercept': False}
R2: -0.15371996517932374
RMSE: 0.2905042334870019

polynomial_regression
-------------------
Fitting 2 folds for each of 6 candidates, totalling 12 fits
Best hyperparameters: {'linearregression__fit_intercept': False, 'polynomialfeatures__degree': 2}
R2: -3.6881123071068664
RMSE: 0.5856002656927668

KNRegression
-------------------
Fitting 2 folds for each of 2496 candidates, totalling 4992 fits
Best parameters: {'leaf_size': 20, 'metric': 'euclidean', 'n_neighbors': 14, 'p': 1, 'weights': 'uniform'}
R2: -0.11750671486611997
RMSE: 0.28590867463672137

SVRegression
-------------------
Fitting 2 folds for each of 384 candidates, totalling 768 fits
Best parameters: {'C': 100, 'degree': 2, 'epsilon': 0.01, 'gamma': 'scale', 'kernel': 'linear'}
R2: -0.11817947540526408
RMSE: 0.2859947229489761
```

...

...

```
ridge_regression
-------------------
Fitting 2 folds for each of 10 candidates, totalling 20 fits
Best alpha: 0.46415888336127775
R2: -0.09260223770006615
RMSE: 0.28270487945282163

huber_regression
-------------------
Fitting 2 folds for each of 75 candidates, totalling 150 fits
Best parameters: {'alpha': 0.0001, 'epsilon': 2.0, 'max_iter': 1000}
R2: -0.09945050952403012
RMSE: 0.28358947201544693

RANSAC_regression
-------------------
Fitting 2 folds for each of 9 candidates, totalling 18 fits
Best parameters: {'min_samples': 0.1, 'residual_threshold': 1.0}
R2: -0.08736475847157554
RMSE: 0.28202648087091164
-------------------

Best Model
Model: bayesian_regression
RMSE: 0.27385694596104804
Insert a numeric code to identify the controller:
12
File 'model_C12.pkl' save with success in personalised_models\model_C12.pkl!
```

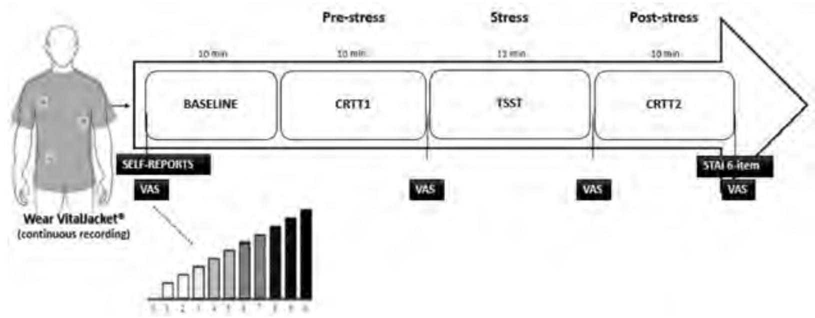
Saves in defined folder:
- Python **Pickle file** with **personalised algorithm**

PKL

Experiments

- ## Algorithm #1.1 | Quantification of Cognitive Performance

11 Subjects



Stress and Cognition Assessment Protocol
(dataset collected previously as described in 2018, Rodrigues et al.)

Model Personalisation

- Features design and optimisation
- ECG Features : HRV & fiducial features (41)
- Personalised physiological normalisation
- Cognitive performance metrics
- ML-Driven Quantification Models

```
Best Model
Model: bayesian_regression
RMSE: 0.27385694596104804
Insert a numeric code to identify the controller:
12
File 'model_C12.pkl' save with success in personalised_models\model_C12.pkl!
```

Personalised Model

Link to the repository

Link to the repository:
https://github.com/AI4REALNET/d2.2-human-assessment-module-v0.5

Graph Neural Solver: Power Grid

Main developer: IRTSX

Authors

Authors	Institution
Milad Leyli-abadi	IRTSX

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

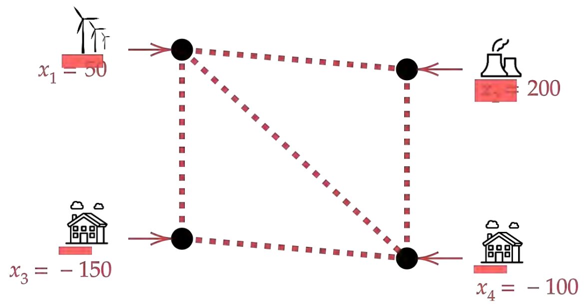
Context

- Currently used physical simulators
  - Inputs / outputs

X = injections
(productions + loads)
T = topology

Numeric Solver
(physics model)

Y = power flows



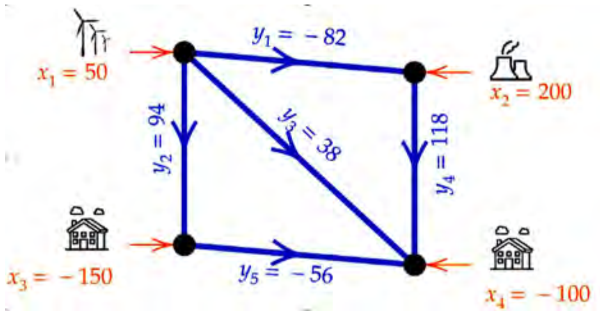
Context

- Currently used physical simulators
  - Inputs / outputs

X = injections
(productions + loads)
T = topology

Numeric Solver
(physics model)

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

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

Context

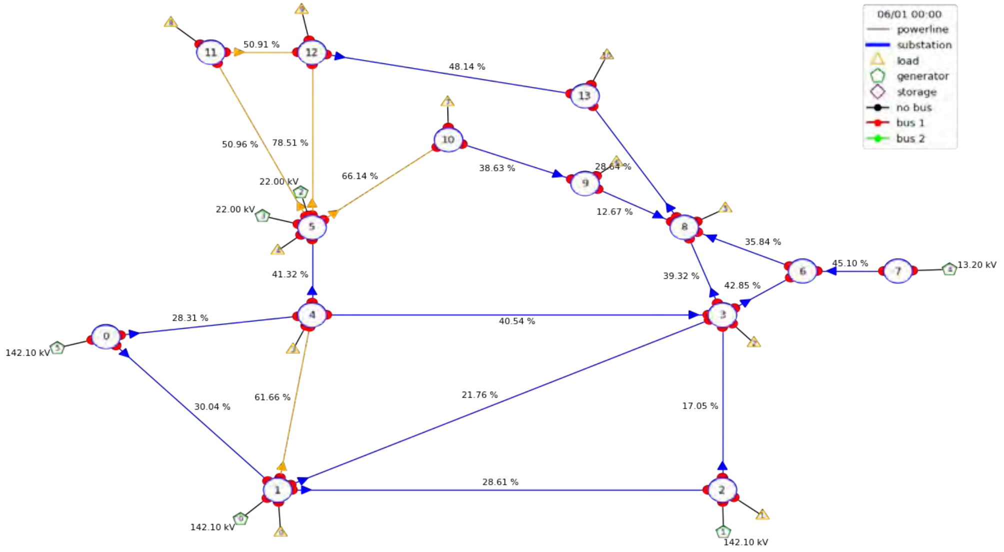
- Need → Simulations allowing to predict the grid state in real time with respect to
  - Action (topological changes)
  - Exogenous factors (wind, temperature, anomalies, etc.)
- Problem → Physics simulation are computationally intensive
  - Need to accelerate the simulation [x100 à x1000]
- Solutions → 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

Accuracy Acceleration

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

Generate a set of synthetic data using the Data Module

Use the Graph Neural Solver to learn from the data (injections, voltage angles)

A physics informed neural network to optimize on local conservation law

Evaluate the performance using a set of 4 categories of evaluation criteria

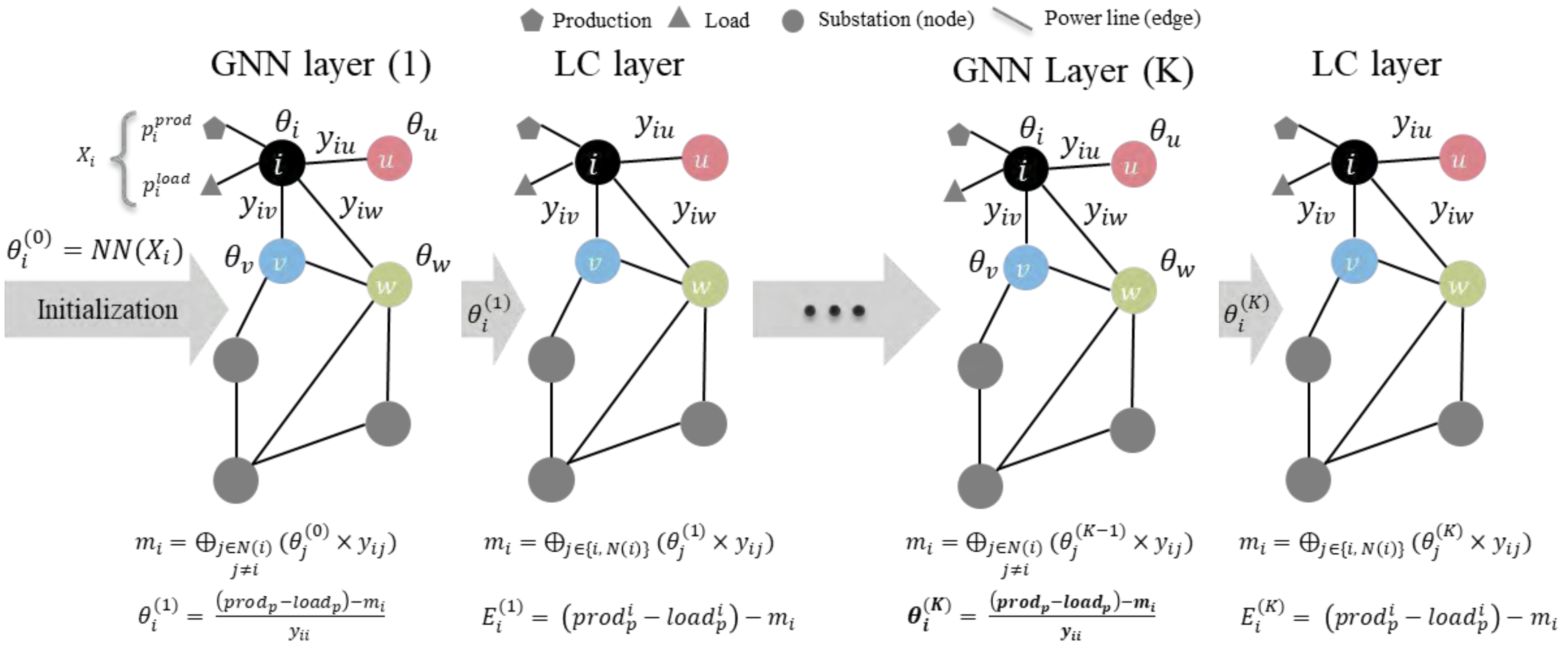
Benchmarking enabled using LIPS framework

- **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	$\frac{1}{L}\sum_{\ell}^{L}\mathbb{1}_{(\hat{a}^{\ell}_{or,ex}<0.)}$	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	$\frac{1}{L_{disc}}\sum_{\ell_{disc}}^{L_{disc}}\mathbb{1}_{(\lvert\hat{x}^{\ell}_{ex}\rvert+\lvert\hat{x}^{\ell}_{or}\rvert>0.)}$	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	$\text{MAPE}((Prod-Load)-(\sum_{\ell=1}^{L}(\hat{p}^{\ell}_{ex}+\hat{p}^{\ell}_{or})))$	Mean energy losses residual
P7	Local Conservation	$\text{MAPE}((p^{prod}_{k}-p^{load}_{k})-(\sum_{l\in neig(k)}\hat{p}^{\ell}_{k}))$	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

```
.
├── LICENSE
├── README.md
├── configs
│   ├── gnn.ini
│   ├── l2rpn_case14_sandbox.ini
│   └── l2rpn_neurips_2020_track1_small.ini
├── getting_started
│   ├── 0_generate_data.ipynb
│   ├── 1_example_gnn_without_nn.ipynb
│   └── 2_gnn_powergrid.ipynb
├── gnn_powergrid
│   ├── __init__.py
│   ├── dataset
│   ├── evaluation
│   └── gnn
└── imgs
    ├── gnn_eq_powergrid.png
    └── gnn_scheme_powergrid.png
```

License (Mozilla Public License) that is used currently

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 cofiguration 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 for 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


```json
{
    "power": {
        "ML": {
            "MAE_avg": {
                "p_ex": 0.0022464103531092405,
                "p_or": 0.0022464103531092405
            },
            "MAPE_10_avg": {
                "p_ex": 0.00017036871994216223,
                "p_or": 0.00017036871994216223
            },
            "MAPE_90_avg": {
                "p_ex": 0.0004267319175032469,
                "p_or": 0.0004267319175032469
            },
            "MAPE_avg": {
                "p_ex": 57037578240.0,
                "p_or": 57037578240.0
            },
            "MSE_avg": {
                "p_ex": 0.00014822339289821684,
                "p_or": 0.00014822339289821684
            }
        },
        "Physics": {
            "CHECK_GC": {
                "mae": 1.0375976671639364e-05,
                "violation_percentage": 0.0,
                "wmape": 1.0
            },
            "CHECK_LC": {
                "mae": 0.0017422774608998484,
                "mape": 8.329480616426182e-05,
                "violation_percentage": 5.0
            },
            "CHECK_LOSS": {
                "violation_percentage": 0.0
            },
            "DISC_LINES": {
                "p_ex": 0.0,
                "p_or": 0.0,
                "violation_proportion": 0.0
            },
            "LOSS_POS": {
                "violation_proportion": 0.0
            }
        }
    },
    "theta": {
        "MAPE10": 0.00011485389095878186
    }
}
```

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	⊖	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	⊖	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)

Link to the repository

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

Neural Prioritized Planning: Flatland

Main developer: UvA

Authors

Authors	Institution
Herke van Hoof	UvA
Marius Captari	UvA

Context

Definition

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

Motivation

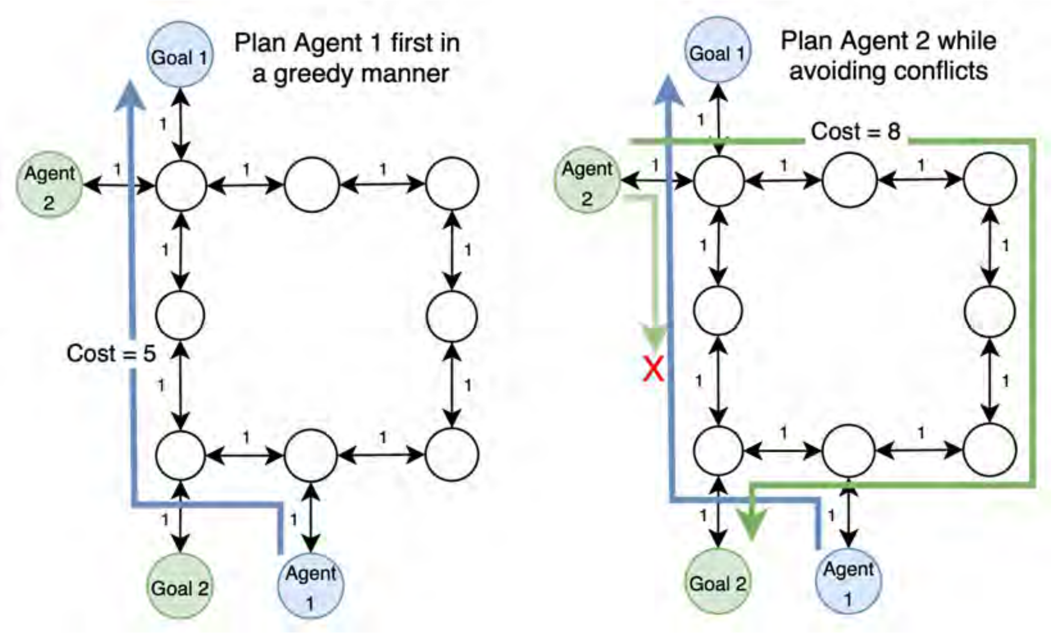
Efficiently scheduling agents in congested networks (e.g., railway systems) is crucial for minimizing delays and optimizing flow. While optimal solvers do exist, they **fail to scale** to problems with more than a few dozen agents. More scalable solvers are needed.

Use cases

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

Motivation: small scale example

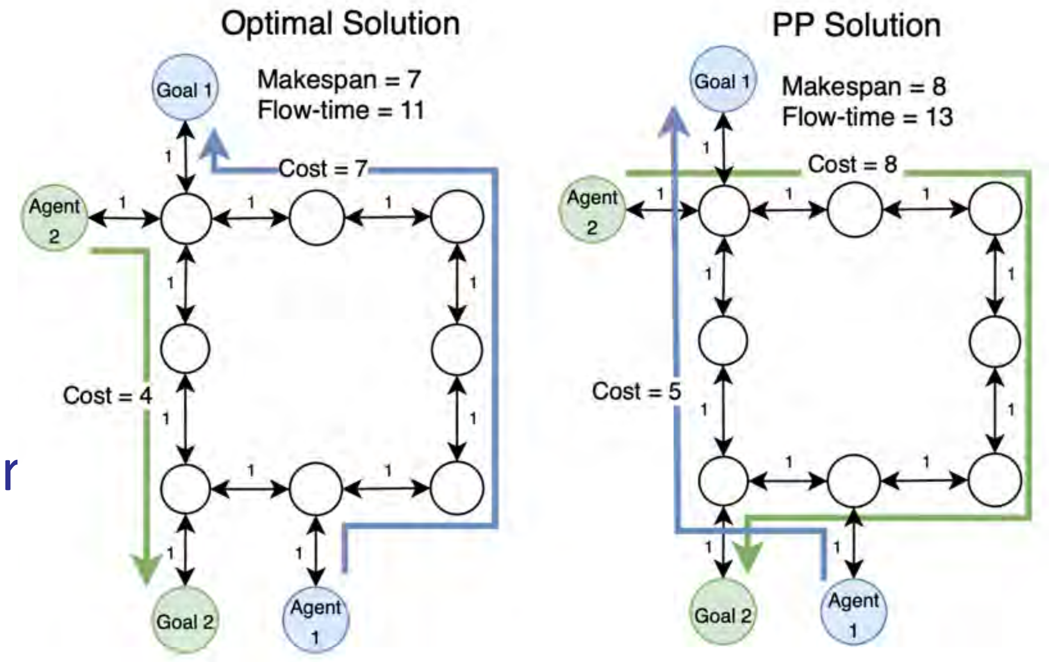
- Plan Agent 1 **greedily**, by using shortest distance to goal
- Agent 2 has to **avoid collision** with Agent 1
- **Prioritized Planning** (PP) is such an algorithm, that plans agents in sequence, based on the assigned priorities.
- Often times, PP it is **not** optimal.



Motivation: small scale example

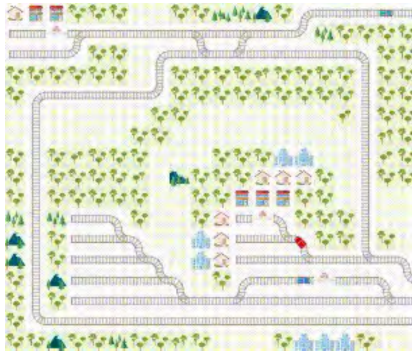
- In this example the optimal path is for Agent 1 to take the slightly longer route, while Agent 2 takes the much shorter one.
- **Makespan** = max length of all paths
- **Flow-time** = sum of length of all paths

- 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 (PP)

Differentiable loss calculation between the 2 solutions

Expert (optimal) solution

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

Set **updated weights** for PP

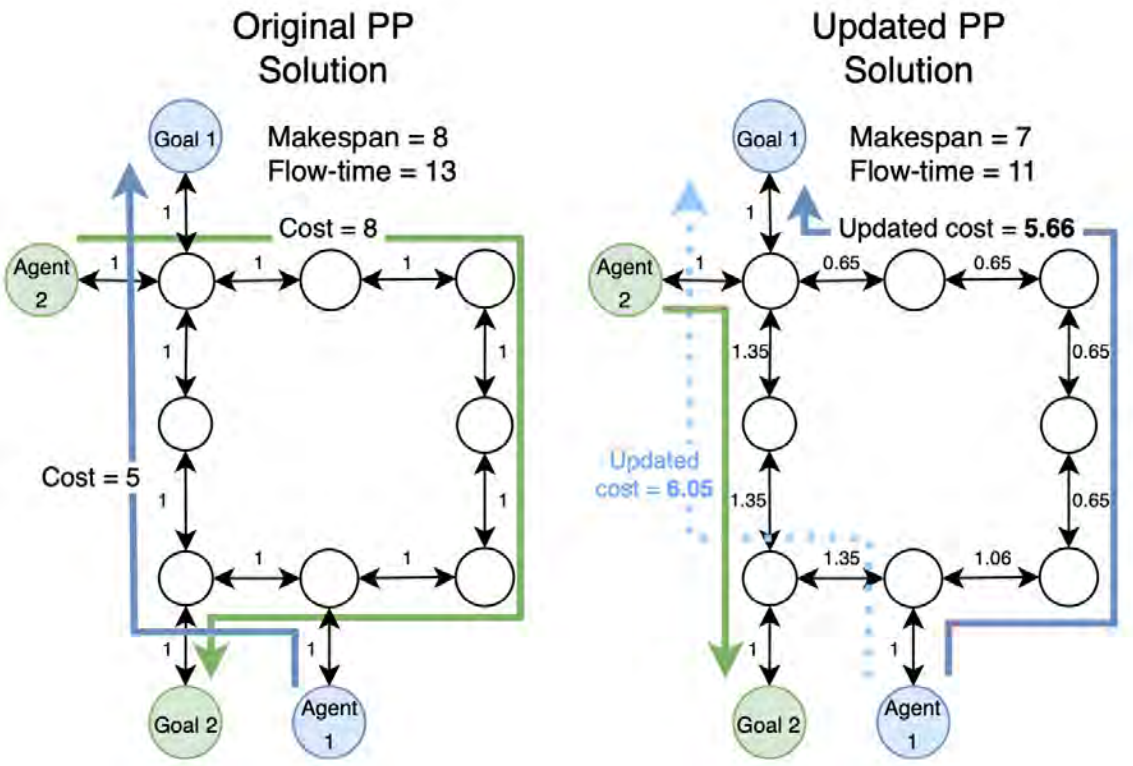
Use **Conflict Based Search** (CBS) to generate optimal paths.

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.



Preliminary 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

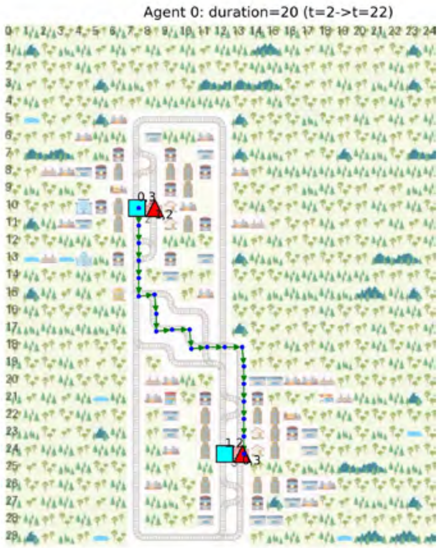
Example on 30x30 Flatland map with 4 agents

- <u>End</u> Stations are represented with a , the numbers represents which agents has to finish there.
- <u>Start</u> 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:**

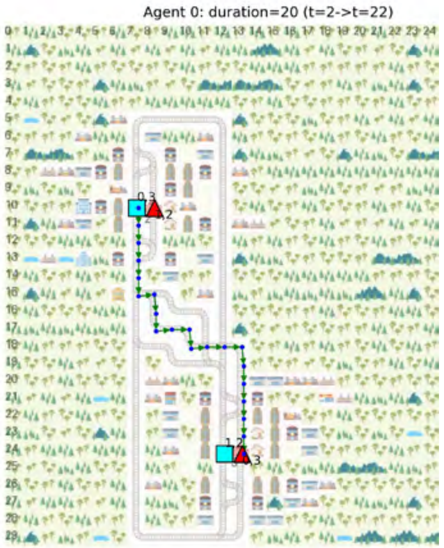
Original PP

Vs

CBS



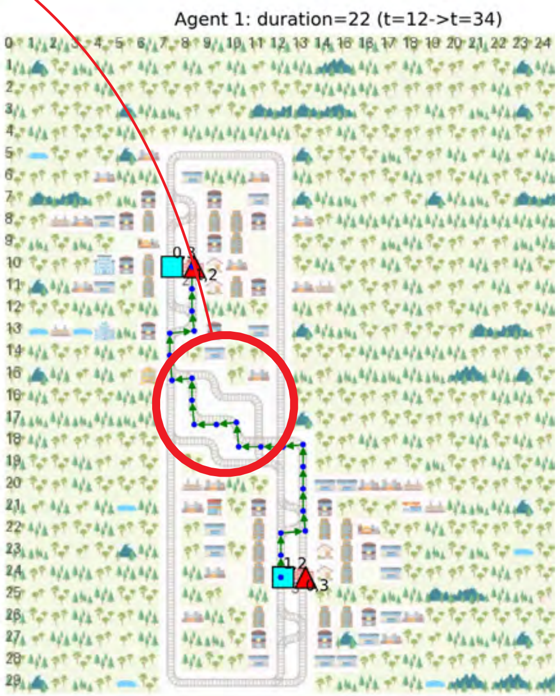
PP Trained (matches CBS Agent 0 path)



Preliminary experiments on Flatland

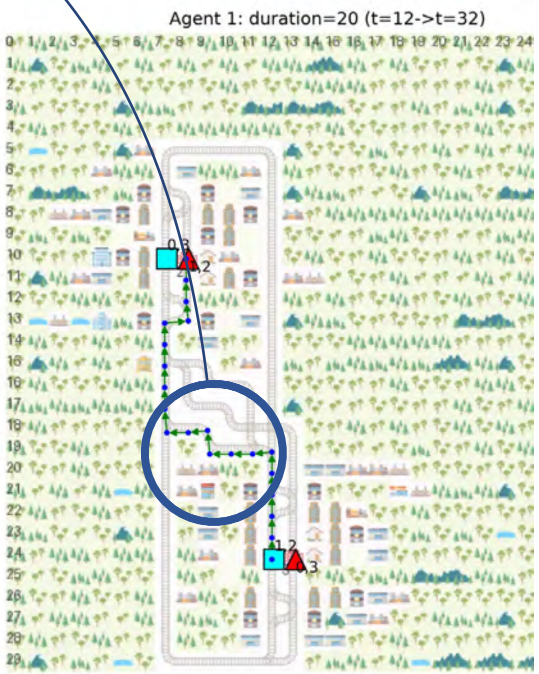
- So that Agent 1 can take the shorter route (**20 length**) using the **updated weights**, instead of the old PP route (**22 length**) to arrive at the goal, avoiding the conflict with Agent 0:

Original PP



Vs

PP Trained



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


```json
{
  "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
  ]
}
```

Perspectives

- Next steps:
  - Working in Flatland simulation:
    - Learn based on **dynamic start/current positions** of agents.
      - Edge weights as a function of the current agent positions
    - If breakdowns happen, **replan** using updated positions/weights.
    - Learned weights should reflect possibility of **breakdowns**.
  - Learn to assign **priorities** (learn to rank approach).
- Future work:
  - Optimal solvers such as CBS dont scale into more realistic scenarios with tens/hundreds of agents at the same time.
  - Have an **Reinforcement Learning loss** instead of relying on expert optimal trajectories.

Link to the repository

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

Maze-Flatland

Main developer: enliteAI

Authors

Authors	Institution
Anton Fuxjäger	enliteAI
Alberto Castagna	enliteAI
Marcel Wasserer	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.

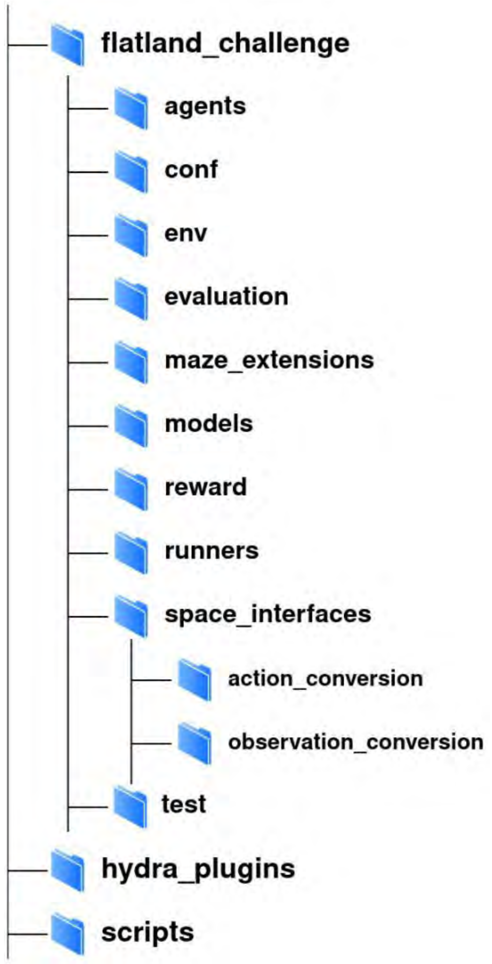
[GitHub - enlite-ai/maze: Maze Applied Reinforcement Learning Framework](https://github.com/enlite-ai/maze)

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

Outline

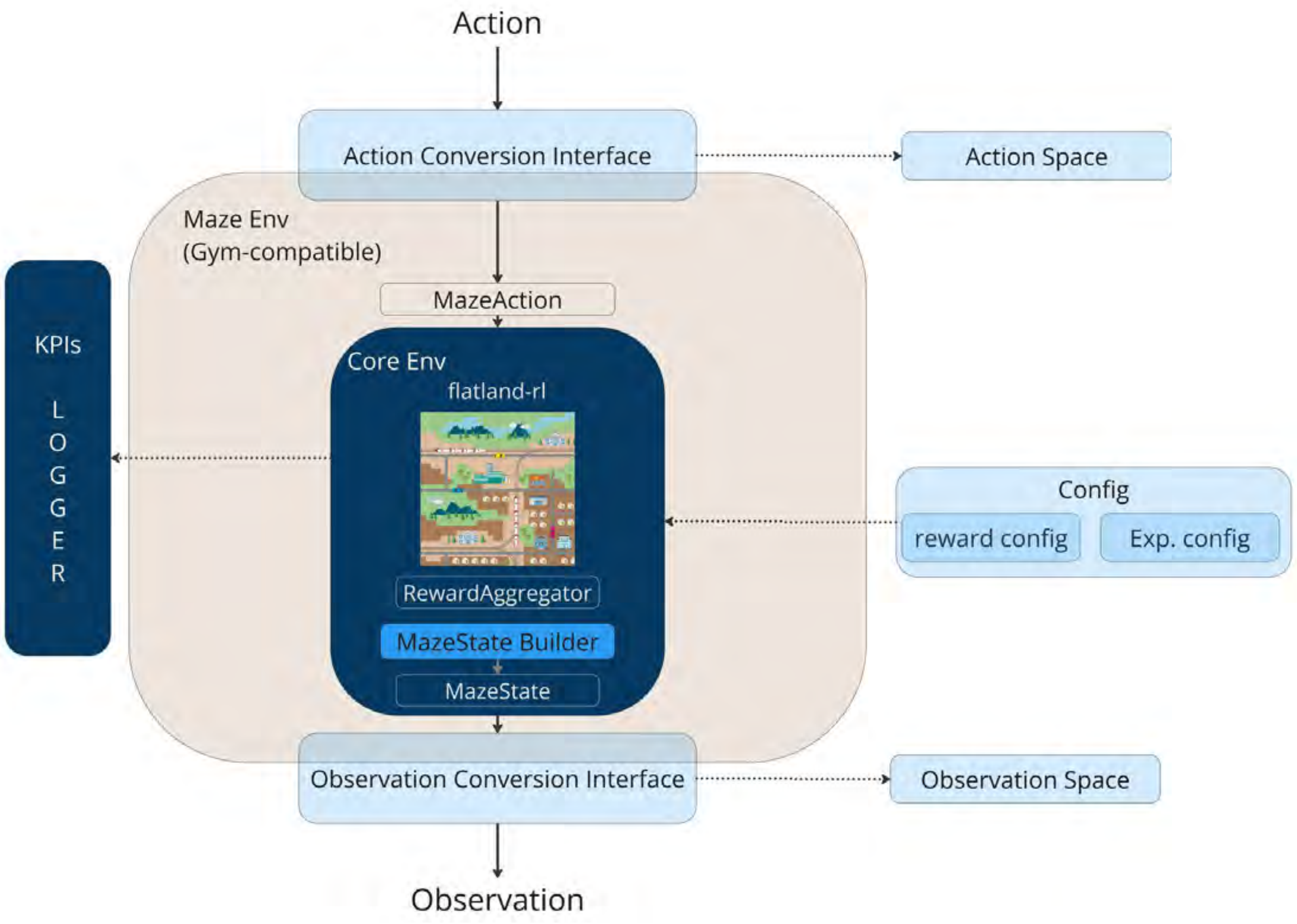
- Repository overview
- Environment architecture
- Multi-Agent formulation
- Wrappers
- Built-in masking logic
- Agent-Environment Interaction
- Maze-Flatland KPIs
- Offline Training
- Validation and Testing
- AI4REALNET – Task 2.2 & 2.3
- Replicate the results
- Performance Comparison – NN vs XGBoost
- Summary

Maze-Flatland – Repository Overview



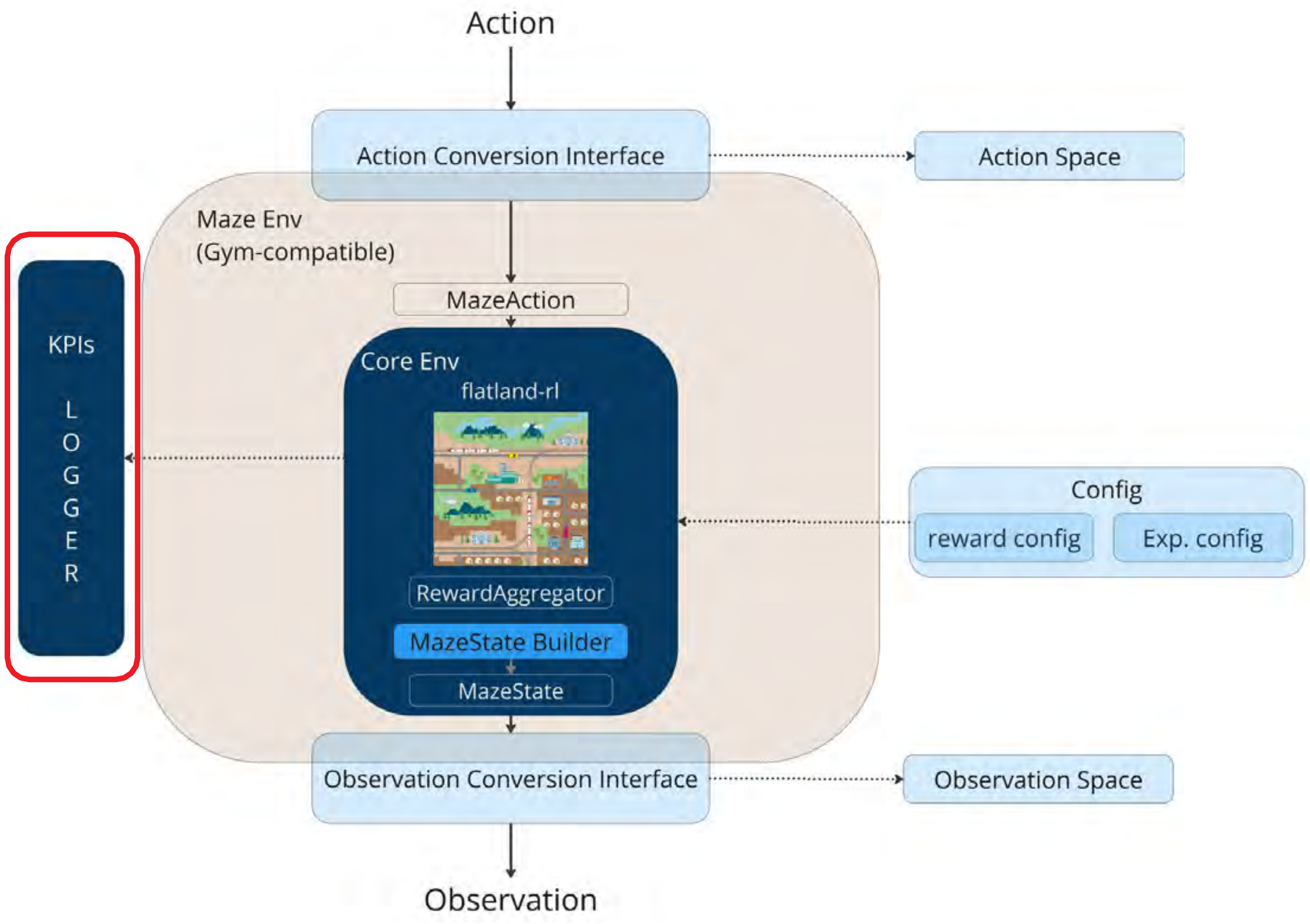
- ➢ YAML-based configuration
- ➢ Extensive logging support
- ➢ Automated testing
- ➢ Modular architecture
- ➢ Designed for scalability and future extensions

Maze-Flatland – Environment Architecture: Flow



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

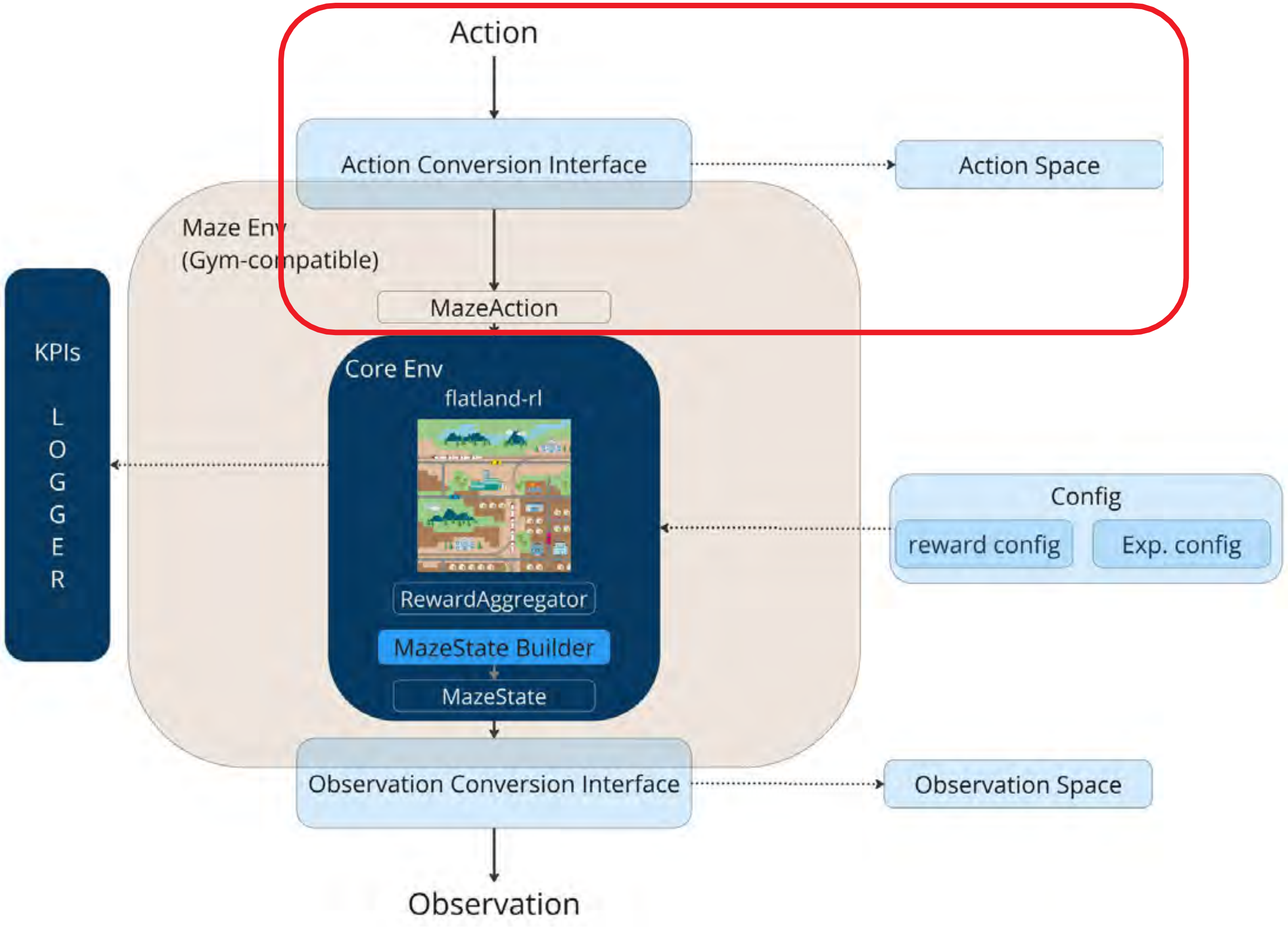
Maze-Flatland – Environment Architecture: Flow



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

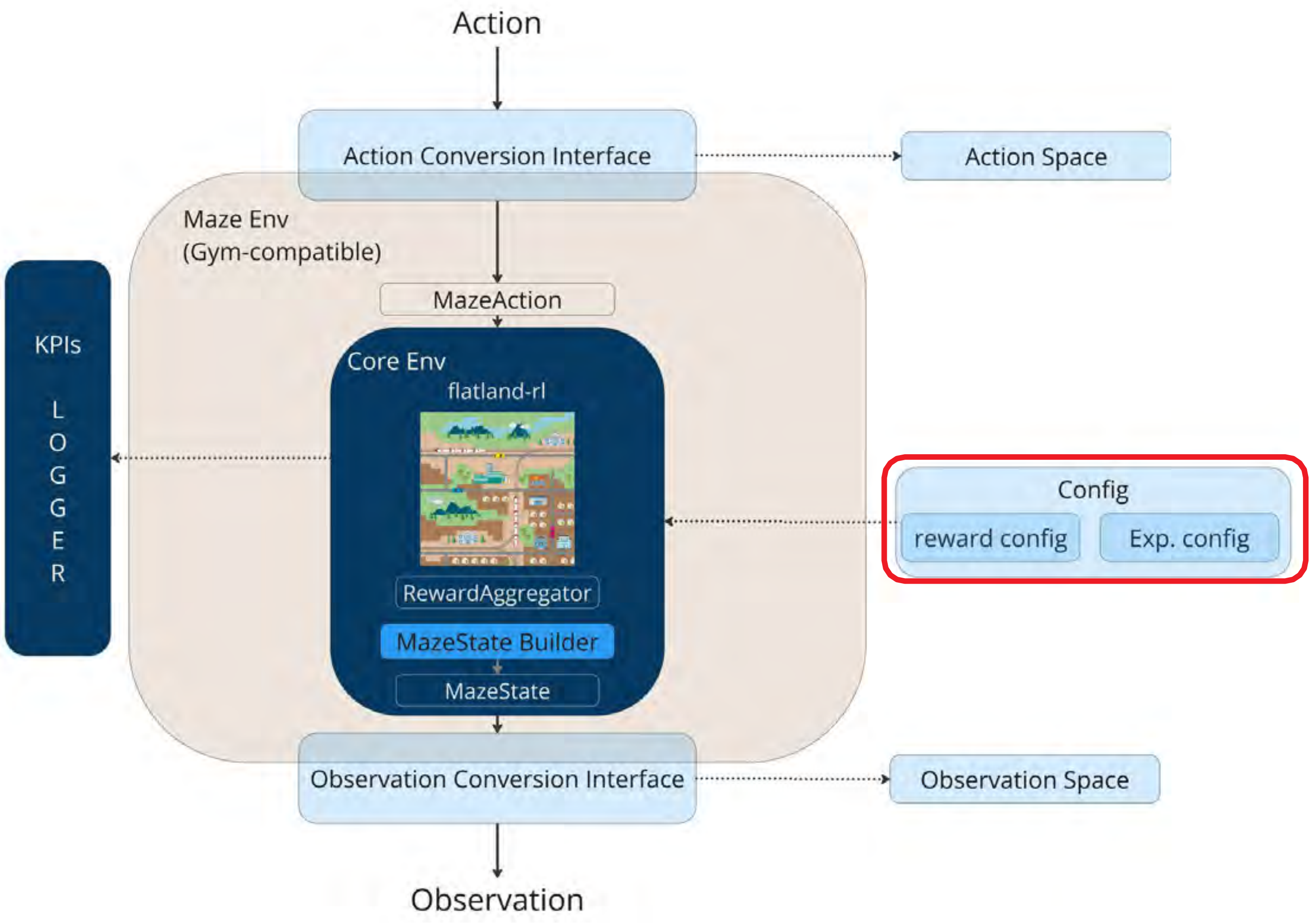
Maze-Flatland – Environment Architecture: Flow



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

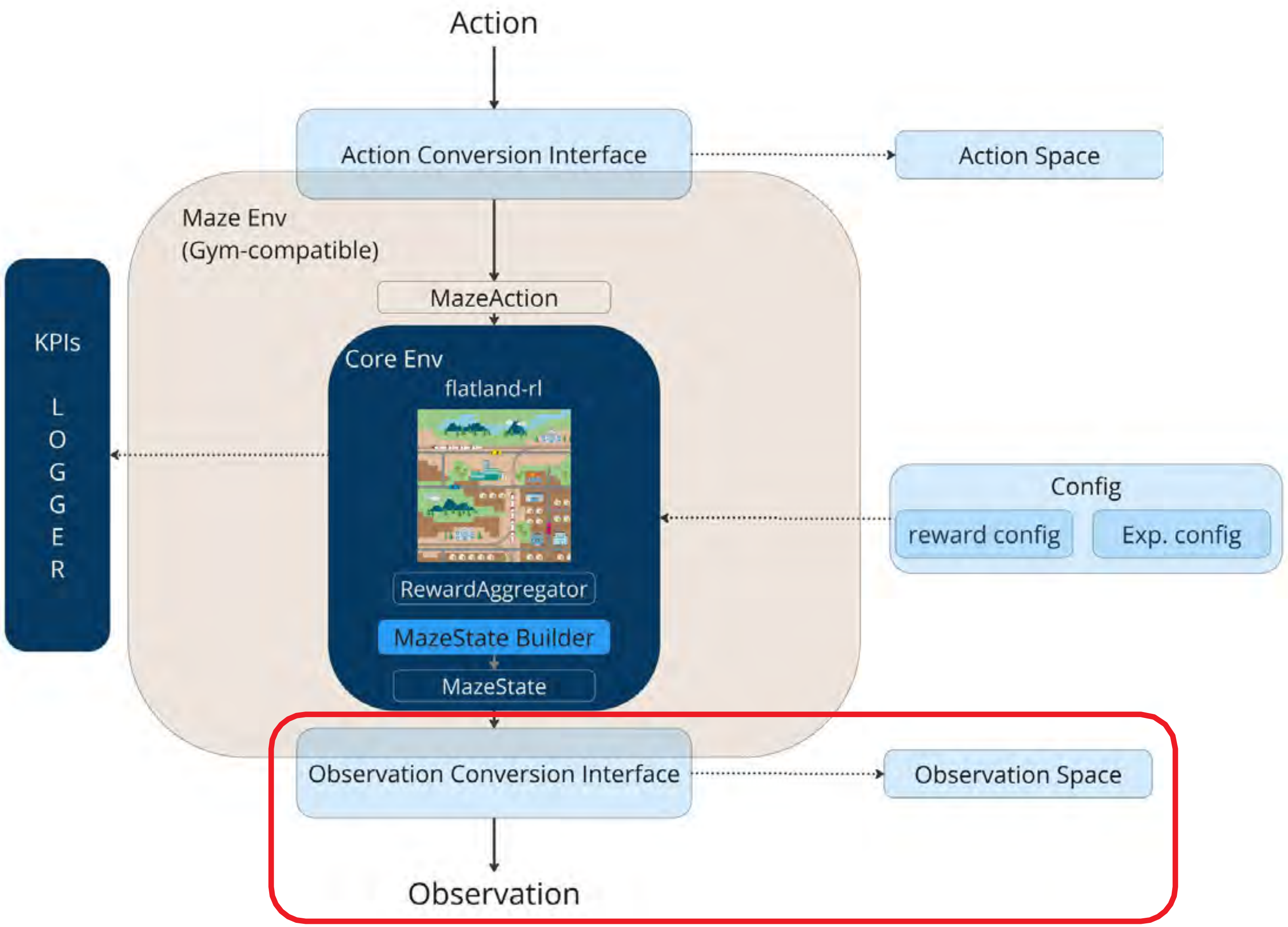
Maze-Flatland – Environment Architecture: Flow



Set rail config, #trains and reward used for experiment

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

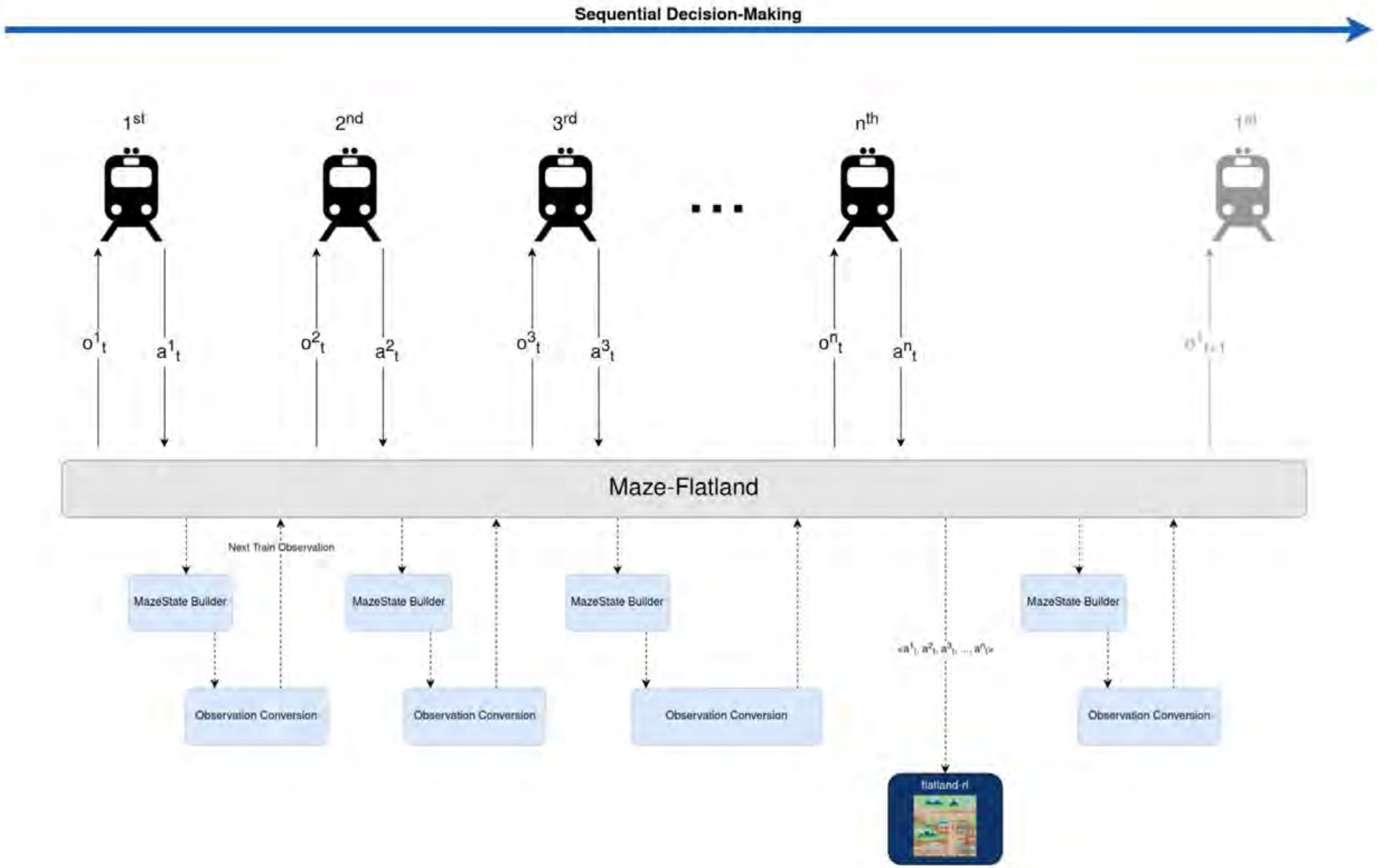
Maze-Flatland – Environment Architecture: Flow



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

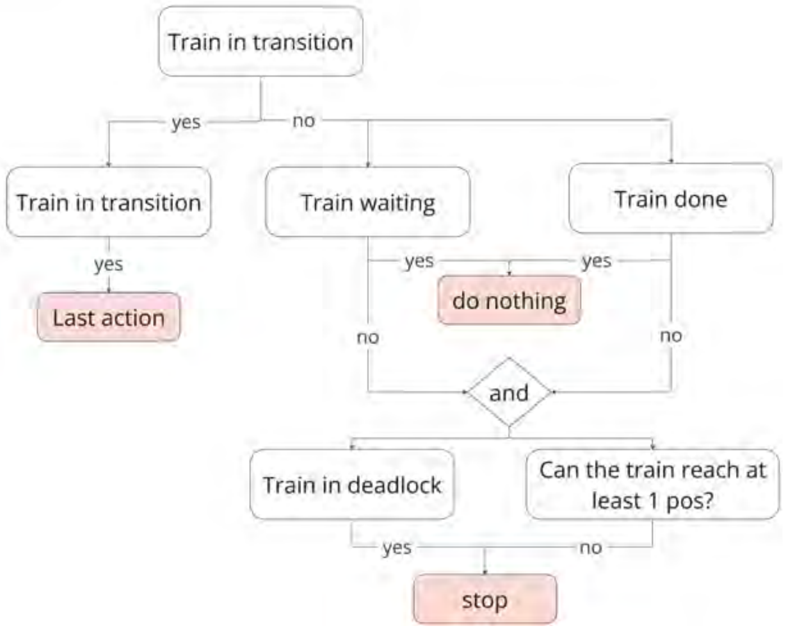
- Prevent unnecessary decision-making – when actions lead to the same outcome
- Improve sample efficiency – Agent is trained only on meaningful state transitions.
- Leading to faster learning and more robust policy

Rendering Wrapper

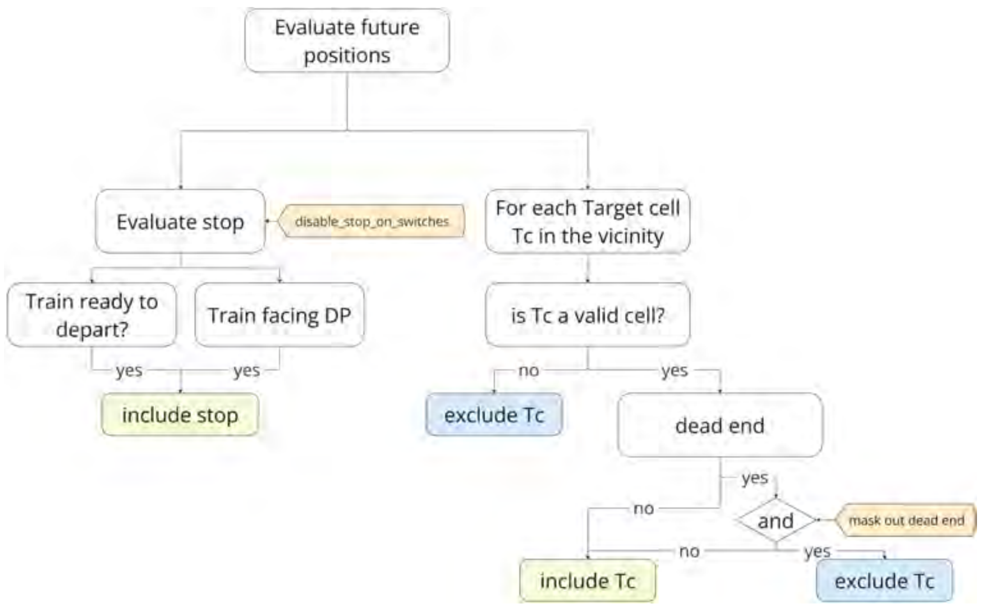
- Support custom visualization – allowing for different level of details

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.

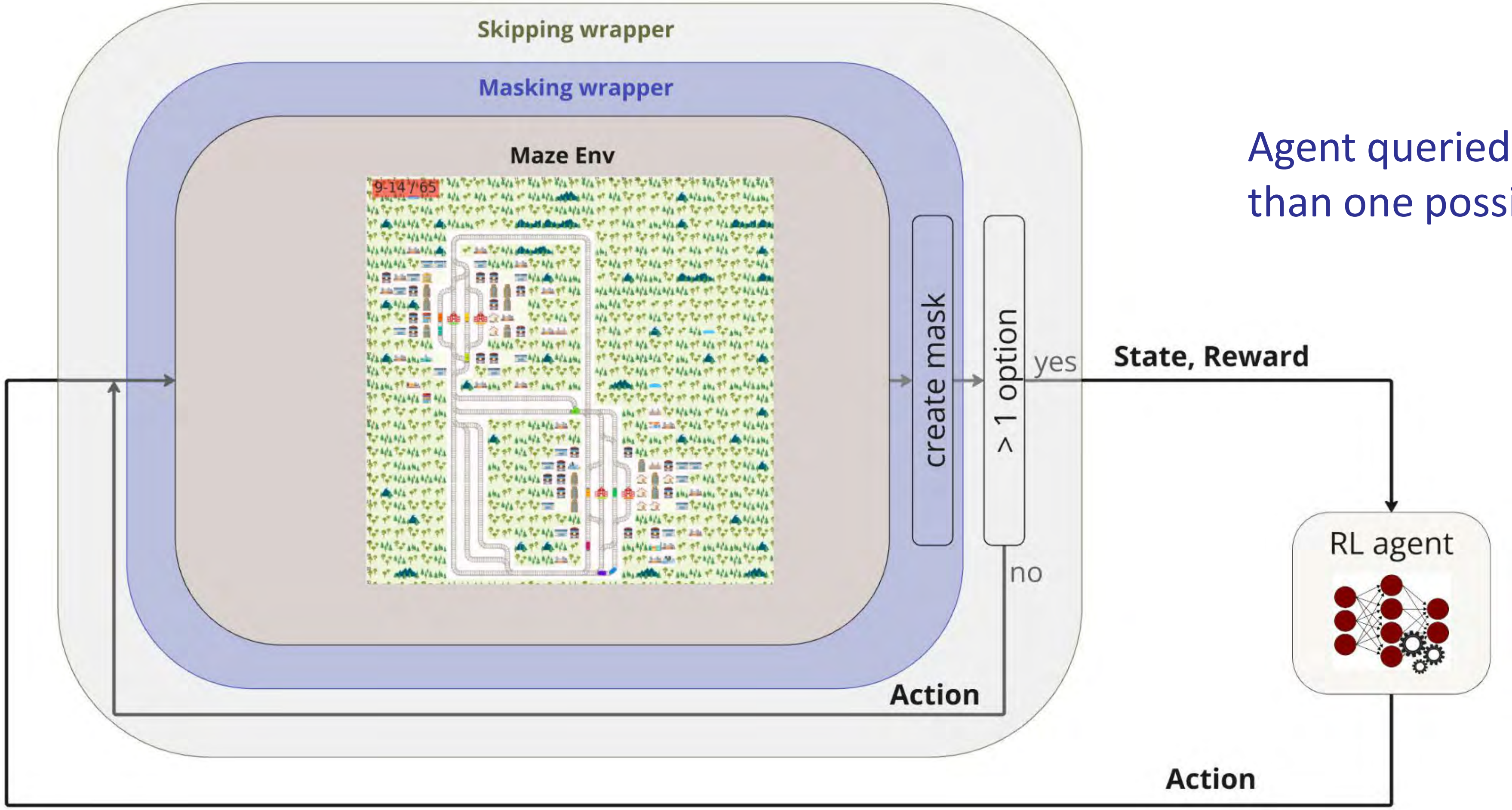
Terminal state

Flag used in masking to change the behavior

Non Terminal State – not included in the masking

Terminal State included in the masking

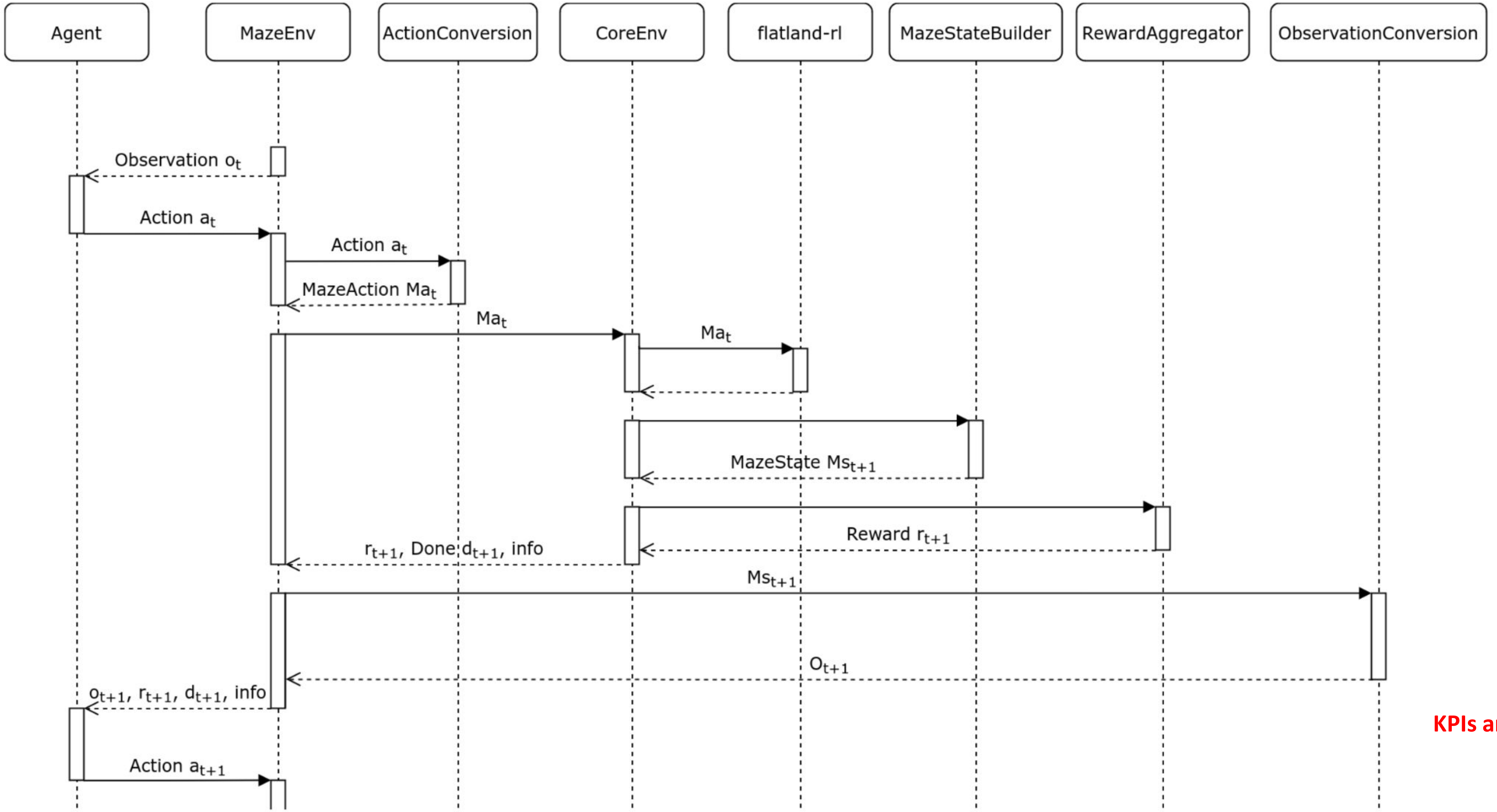
Maze-Flatland – Agent-Env Interaction



Agent queried only when there is more than one possible option

Learn more at https://maze-rl.readthedocs.io/en/latest/reference_docs/core_wrappers.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	25030.000
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	0.080
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	0.087
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:5006, µ:1.5]
1	rollout_stats	ActionEvents	discrete_action	step_key_train_move/agent_1/..	[len:5006, µ:1.6]
1	rollout_stats	ActionEvents	discrete_action	step_key_train_move/agent_2/..	[len:5006, µ:1.6]
1	rollout_stats	ActionEvents	discrete_action	step_key_train_move/agent_3/..	[len:5006, µ:1.6]
1	rollout_stats	ActionEvents	discrete_action	step_key_train_move/agent_4/..	[len:5006, µ:1.7]
1	rollout_stats	SkipEvent	sub_step	sum_skipped	18936.000
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	-182.560
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

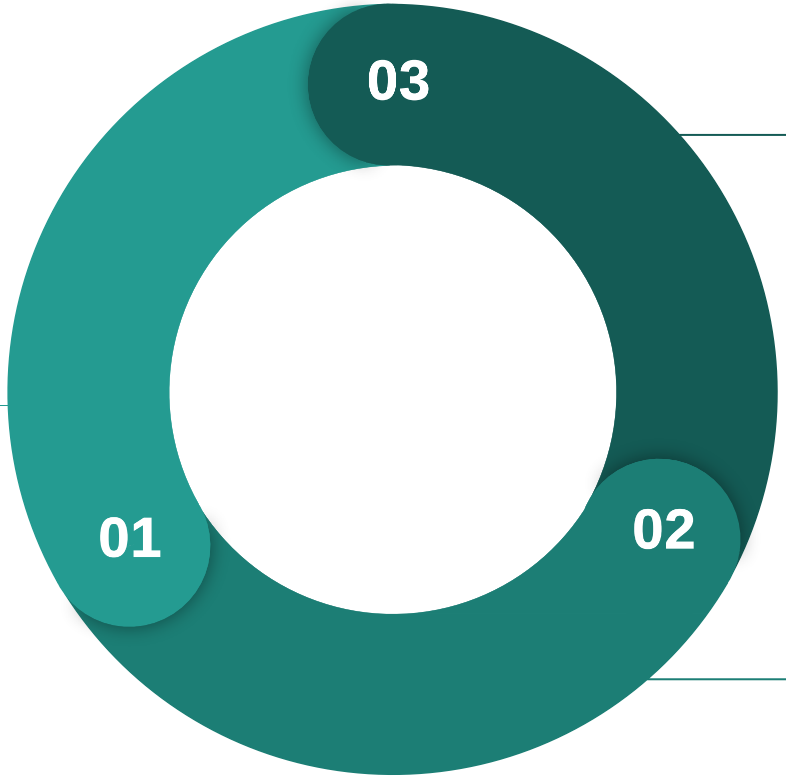
Can be extended!!!

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



Offline training

Use the provided tool to learn from the collected experience.

Preprocessing

Apply a pre-processing filter when loading the stored trajectories

Create dataset

Use a heuristic or a policy to collect the 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

AI4REALNET – Task 2.2 & 2.3

2. → Environment engineering

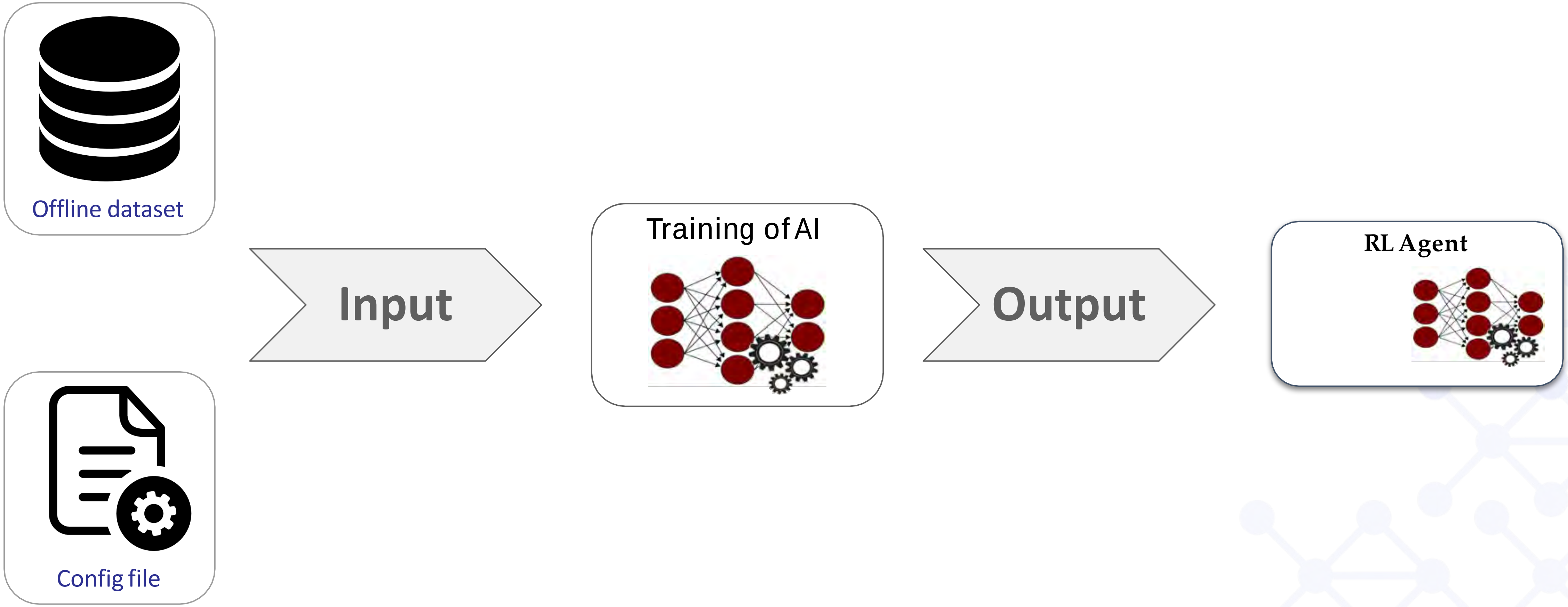
- Improved environment engineering to **enhance AI-based agent development**;
- Designed multiple reward formulations **improving feedback frequency**, leading to more efficient training

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**.

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



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/1FW6FnAKHgXXu_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

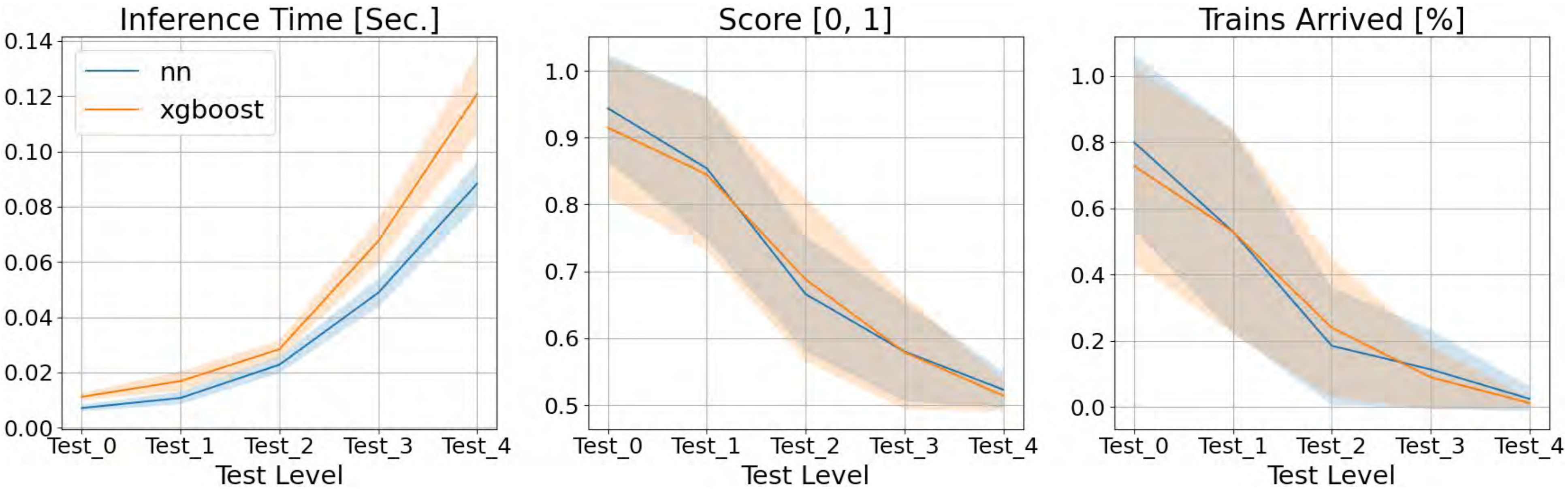
```
maze-run
+experiment=multi_train/rollout/validation_xgboost
input_dir=<path/to/policy>
```

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 customisation: reward, observation and action;
- Integrated XGBoost for interpretable decision making

	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	X	X	X	✔	X	X	X

Link to the repository

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

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.