

NJIT

SortingEnv: Extendable RL-Environment for an Industrial Process Milestone 5

Bruno Garofalo & Kerim Sever

DS 669-852

Professor Jing Li

May 11, 2025

Table of contents:	0
1. References and Acknowledgements	0
2. Application, MDP, and RL Model	5
3. Codebase, System, and Experiment Setup	6
4. Experiments	8
5. Conclusion	26

1. References and Acknowledgements

- **Reference of the paper:**
 - Paper Link: [2503.10466](#)
 - Presented at the 12th International Conference on Industrial Engineering and Applications (ICIEA-EU), Munich, 2025.
- **Links to codebase, datasets, etc.:**
 - Sortin_env GitHub: [GitHub - Storm-131/Sorting_Env: This is the code for the \(upcoming\) paper "SortingEnv: An Extendable RL-Environment for an Industrial Sorting Process" \(Maus et Al., 2025\)](#)
- **Other resources:** Stable baseline 3: [Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations — Stable Baselines3 2.6.1a0 documentation](#)
- **Workload distribution among group members:** Bruno 50% Kerim 50%

2. Application, MDP, and RL Model

Application introduction:

SortingEnv is a novel Reinforcement Learning framework designed to simulate complex industrial sorting systems typically found in manufacturing facilities, warehouses or distribution centers. This system was designed for training several RL agents, namely: Proximal Policy Optimization (PPO), Deep Q-Network (DQN) and Advantage Actor Critic (A2C), benchmarked against a Rule-Based Algorithm which represents the baseline model.

The industrial environment stimulates the flow of materials through 3 different components:

- *Input stage:* where the input mix of material A and B is generated

$$\textbf{Total input } I = I_A + I_B,$$

where I_A and I_B represent the quantity of material A and B

The input can be generated by a random generator or a seasonal pattern generator which introduces seasonality to the input mix

- *Conveyor belt or belt stage*: the component that transports the materials to the sorting module. Measured as the proportion of total belt occupied with material:

$$\text{Occupancy} = (B_A + B_B)/100,$$

where B_A and B_B represent the quantities of material A and B on the belt.

- *Sorting machine*: the module responsible for accurately sorting out the materials by type. The sorting accuracy is defined as α , which represents the proportion of correctly sorted material A and B, is calculated using the current formula but the way the parameters are derived depends on the environment used:

$$R = r_{\text{acc}} \cdot \left(\frac{\alpha - \text{threshold}}{1 - \text{threshold}} \right) + r_{\text{speed}} \cdot \left(\frac{v - 0.1}{0.9} \right) - \text{penalty}$$

Where r_{acc} is the reward for the accuracy, r_{speed} the reward for the speed, the *penalty* is added to minimize the number of belt adjustments, *threshold* is the minimum accuracy required (defaults to 0.7), v is the belt speed

The reward function will be discussed in more detail in the MDP section

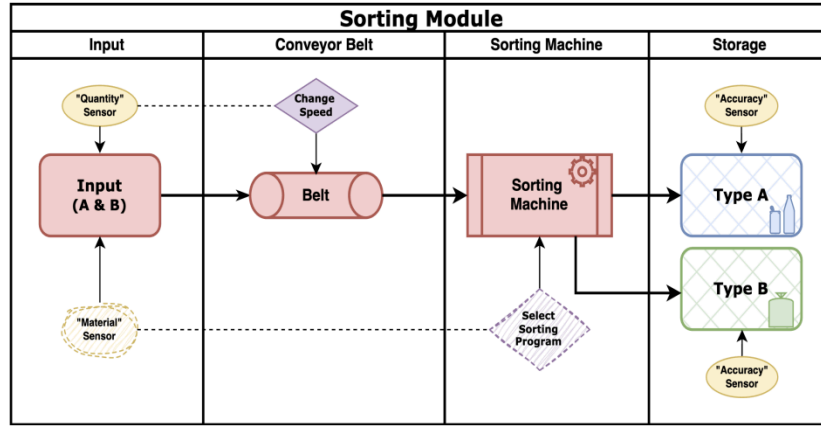


FIGURE 1. General overview of the Sorting Environment with the four compartments “Input”, “Belt”, “Sorting Machine” and “Storage”. The simple environment is described by the upper elements for sensors (yellow) and actions (purple). The configuration for the advanced environment is shown by additional sketched elements below, for another sensor (yellow) and action (purple).

The model can operate in 2 different environments. Both environments were implemented using Gymnasium (v0.29.1) to ensure the compatibility with the selected RL agents:

- *Basic environment*: focuses on the optimization of the belt speed and conveyor occupancy (quantity of material transported) to maximize the sorting accuracy and total reward

- Advanced environment: similar to the basic environment but utilizes 2 additional factors for the optimization: 1. The material ratio (quantity of material A / quantity of material B)
2. Sorting modes which depend on the material ratio

MDP formulation:

SortingEnv is based off MDP because of its sequential decision-making and results. The reasons why MDP makes sense for the issues at hand are below:

- **The environment has discrete states and action:**
 - This system considers how full the conveyor belt is based on the mix of inputs A and B, the speed of the conveyer belt moving, and the active sorting mode. These conditions determine the current state of the system and help the agent make decisions to maximize longterm reward. Previous states are no longer needed if the current state is well-defined following MDP.
- **The agent needs to make decisions over time:**
 - The goal is to have a strategy which is known as policy that works consistently over time. A situation where policy is helpful is to slow down the belt to improve accuracy even if it will slow down the process of moving materials.
- **The current state is enough to make decisions (Markov property):**
 - The agent considers the current state to make its decision on the what the next action is. It does not need previous decisions because the current state will have all the necessary information to make its best decision.
- **Rewards guide learning:**
 - The reward is given to the agent after each action. The reward is based on accuracy and efficiency and is used to determine what is the best policy.
- **It helps the agent get better over time:**
 - A policy, mapping from state to action, performs well over time fits SortingEnv's goals. The agent needs to learn to slow down, speed up, sorting setting all based on the systems state.

MDP overview:

The Markov Decision Process formulation (S, A, P, R, γ) changes based on the environment used.

Basic Environment MDP:

- **States representation (S):** given by the total amount of material inputted.

$$\text{Total input } I = I_A + I_B$$

- **Actions representations (A):** Ten discrete actions corresponding to belt speeds of 10% to 100%.

$$a \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$$

- **Transition probabilities (P):** modeled by the swallowing formula.

$$P(s_{t+1} | s_t, a_t) = f(s_t, a_t, \text{noise})$$

- **Reward function (R):** the agent must dynamically adjust and find the optimal speed in each setup. The final reward depends on the reward for achieving a good accuracy and the reward for achieving an optimal speed.

The accuracy α of the sorting process for materials A and B is influenced by both belt speed v and belt occupancy.

$$\alpha = \begin{cases} 1 - \text{Noise} & \text{if } O_{\text{Belt}} \leq O_{v_max} \\ \alpha_{>\text{limit}} = 1 - (O_{\text{excess}} * \lambda) - \text{Noise} & \text{if } O_{\text{Belt}} > O_{v_max} \end{cases}$$

Where O_{v_max} is the maximum belt occupancy limit per belt speed, O_{v_belt} is the belt speed at time step t , O_{v_excess} is the excess belt occupancy, calculated by subtracting the occupancy limit from the belt occupancy.

For each possible belt speed, there is a predefined occupancy threshold (denoted as O_{max}) that determines the maximum allowable load on the belt to achieve the highest sorting accuracy. If the belt occupancy is within the acceptable speed range then the accuracy is set to 1.0, otherwise the accuracy decreases linearly according to this formula

$$\alpha_{>\text{limit}} = 1 - (O_{\text{excess}} * \lambda) - \text{Noise}$$

The noise factor has been introduced to account for variability. Once the accuracy and the belt speed are known then the reward can be calculated as:

$$R = r_{\text{acc}} \cdot \left(\frac{\alpha - \text{threshold}}{1 - \text{threshold}} \right) + r_{\text{speed}} \cdot \left(\frac{v - 0.1}{0.9} \right) - \text{penalty}$$

Where r_{acc} is the reward for the accuracy, r_{speed} the reward for the speed, the *penalty* is added to minimize the number of belt adjustments, *threshold* is the minimum accuracy required (defaults to 0.7), v is the belt speed

The quantities of correctly and incorrectly sorted material is then updated based on the sorting accuracy:

$$S_A = \alpha \times B_A + (1 - \alpha) \times B_B$$

$$S_B = \alpha \times B_B + (1 - \alpha) \times B_A$$

Where S_A and S_B represent the quantity of correctly sorted material A and B

Advanced Environment MDP:

The basic environment allows only for discrete belt speed adjustments and does not take into account the material mix, but real-world sorting processes are more unpredictable and unstable. The advanced environment introduces a layer of realism and complexity that adds depth to the whole simulation environment. The advanced environment simulates a machine upgrade by introducing three sorting modes that depend on the ratio of material A and B:

$$\text{Basic: } \frac{1}{3} \leq \frac{A}{B} \leq 3, \quad \text{Positive: } \frac{A}{B} \geq 3, \quad \text{Negative: } \frac{A}{B} \leq \frac{1}{3}$$

When the amount of material A is 3 or more times greater than B then the positive sorting mode is selected; if B is 3 times or more greater than A then the negative mode is selected; finally if the A/B ratio is between $\frac{1}{3}$ and 3 then the basic sorting mode is selected. These sorting modes expand the action space of the MDP as shown below.

- **States representation (S):** given by the total amount of material inputted, same as in the basic environment, and the material ratio.

$$\text{Total input } I = I_A + I_B$$

$$\text{Material ratio: } A/B$$

- **Actions representations (A):** the action space is expanded to include a total of 30 actions: 10 discrete actions corresponding to belt speeds of 10% to 100% x 3 sorting modes. The new action spaces can be represented as:

$$A \in \{(0.1, B), (0.1, P), (0.1, N), (0.2, B), (0.2, P), (0.2, N), \dots, (1.0, B), (1.0, P), (1.0, N)\}$$

- **Transition probabilities (P):** modeled by the swallowing formula, same as in the basic environment.

$$P(s_{t+1} | s_t, a_t) = f(s_t, a_t, \text{noise})$$

- **Reward function (R):** the reward function is calculated like in the basic environment but it now depends on the original two variables belt speed and belt occupancy plus the sorting mode selected by the agent.

$$R = r_{\text{acc}} \cdot \left(\frac{\alpha - \text{threshold}}{1 - \text{threshold}} \right) + r_{\text{speed}} \cdot \left(\frac{v - 0.1}{0.9} \right) - \text{penalty}$$

$$\alpha = \begin{cases} \min(\alpha + 0.15, 1.0) - \text{Noise} & \text{if mode correct} \\ \max(\alpha - 0.10, 0.0) - \text{Noise} & \text{if mode incorrect} \end{cases}$$

If the sorting mode selected by the agent is correct, then the sorting accuracy α takes the lower value between $\alpha + 0.15$ and 1 minus the noise. If the sorting mode selected by the agent is incorrect, then the sorting accuracy α takes the max value between $\alpha - 0.1$ and 0 minus the noise. For example, knowing the ratio of incoming materials allows the machine to optimize its sorting mechanism, increasing accuracy by 15% and decreasing noise to a range of 0 to 5%. Conversely, selecting an incorrect sorting mode can degrade performance, reducing accuracy by 10%.

RL approach:

RL is used to train in this project to train agents to control an industrial sorting process with different conditions. The model uses Stable Baseline 3 and is measured using both Basic and Advanced environments. The three main models are PPO, DQN, and A2C against a traditional rule-based baseline.

Proximal Policy Optimization (PPO)

PPO is a policy gradient algorithm that learns by trying different actions and slowly making improvements. The pros of PPO is that it makes small safe adjustments to keep previous learning. PPO works best in environments where the agent has to make complicated decisions like the Advanced Environment.

Pros:

- Better in complex environments.
- Better for continuous actions like changing belt speed.
- Stable and reliable during training.

Cons:

- Longer to learn.
- Requires more training steps to reach better results.

2. Deep Q-Network (DQN)

DQN is a value-based algorithm that calculates the Q-value using deep neural networks. It learns by calculating the value of each action and picks the highest score. It is best in small environments like the Basic Environment. This is because in the Basic environment it only has to choose from ten different belt speeds but in the Advance environment it did not do as well because of the different sorting modes and complex choices.

Pros:

- Easy to train.
- Fast learning in environments in Basic Env.

Cons:

- Struggles with continuous actions.
- Does not perform well when action depends on multiple factors.

3. Advantage Actor-Critic (A2C)

A2C uses both value and policy-based learning by running two separate networks: an actor (chooses actions) and a critic (evaluates actions). This architecture helps the agent learn faster and make better decisions because of the two networks. A2C does a good job in both environments but needs proper tuning in order to do so. PPO and A2C had close results in the Advanced Environment.

Pros:

- Learns faster.
- Good balance between new actions and previous actions.

Cons:

- Unstable if tuning is off.
- Doesn't perform better than PPO in complex tasks.

Why RL is a Good Fit for SortingEnv

The use of RL in SortingEnv is best because this project is based on an agent making real time decisions with constantly changing conditions. RL offers the agent to learn from trial and error, adapt to new situations, and can improve over time. This works well with the original intent of the paper by creating a way to sort materials with different conditions and environments, something a traditional system will struggle with especially when given complex decisions. The use of RL makes a strong and flexible solution to improve performance in a real word sorting system.

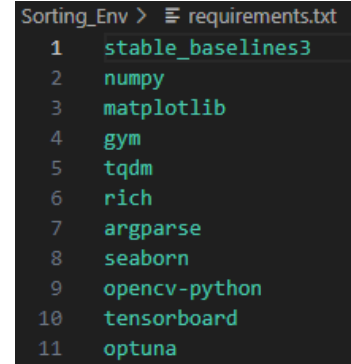
3. Codebase, System, and Experiment Setup**Libraries and system setup:**

Sorting_env model files: download from GitHub page [GitHub - Storm-131/Sorting_Env: This is the code for the \(upcoming\) paper "SortingEnv: An Extendable RL-Environment for an Industrial Sorting Process" \(Maus et AL., 2025\)](#)

All of these libraries are listed in the requirements.txt file (see below) and are automatically installed using the command:

[`pip install -r requirements.txt`](#)

- Simulated via Gymnasium 0.29.1
- Stable_baseline3
- Seaborn - latest version
- Numpy - latest version
- Matplotlib - latest version
- Optuna - latest version
- Tqdm - latest version
- Rich - latest version
- Opencv-python - latest version
- Tensorboard - latest version



```

Sorting_Env > requirements.txt
1  stable_baselines3
2  numpy
3  matplotlib
4  gym
5  tqdm
6  rich
7  argparse
8  seaborn
9  opencv-python
10 tensorboard
11 optuna

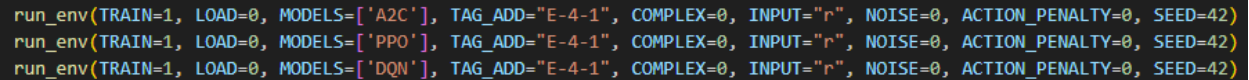
```

Hardware setup:

Sorting_Env was designed to run on CPU, the user should therefore ensure that the model is assigned to the correct device if a GPU is available.

Running the experiments:

In order for an experiment to be executed the models need to be trained first, then the pre-trained models are loaded and used for testing. To start, the training process for each experiment needs to be set-up in the main.py file.



```

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-4-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-4-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-4-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

```

This file contains a very long list of global and experiment specific variables. We will present only the variables that are required to reproduce our experiments.

Global variables that affect all experiments:

1. **THRESHOLD** : Threshold for accuracy. Default = 0.7
2. **TIMESTEPS** :Total Training Steps (Budget). Default = 100_000
3. **STEPS_TRAIN** : Steps per Episode (Training). Default = 250
4. **STEPS_TEST** : Steps per Episode (Testing). Default = 50
5. **SEED** :Random Seed for Reproducibility (can also be experiment specific). Default = 42

Experiment specific inputs:

1. **INPUT**: = r: random inputs, s3 or s9: seasonal input (simple or complex).
2. **NOISE**: noise level in observations (range 0 - 1).
3. **ACTION_PENALTY**: adds cost for extra actions to encourage efficiency. Default = 0.5

1. The first step is to set up the global variables. Select the required number of TIMESTEPS (total number of steps in the training process), STEPS_TRAIN (steps for each training episode), STEPS_TEST (steps for each test episode), THRESHOLD (Accuracy threshold).
2. The next step is to set up the individual experiments. Below we are showing a sample set up for a single experiment run with all 3 RL agents (RBA does not need training).

```
run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-4-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-4-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-4-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
```

3. Generally speaking, in order to obtain the results for RBA two cycles need to be set up.
Training cycle: within the `__main__` function of main.py, select TRAIN =1 and LOAD = 0. This will tell the model what process should be run. In this case we are selecting the training cycle. Then list out the specific model you want to get trained and assign a tag to the experiment. Alternatively, multiple RL agents can be passed to the MODEL variable like shown below.

```
run_env(TRAIN=1, LOAD=0, MODELS=['A2C', 'PPO', 'DQN'], TAG_ADD="E-4-1",
        COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
```

4. The next key step is to assign a unique ID to each experiment by modifying the TAG_ADD variable.
5. Then the type of environment needs to be selected: 0 for basic, 1 for complex.
6. The INPUT variable is next. Choose the proper material input generator: r for random generator, s3 for seasonal basic, s9 for seasonal complex.
7. Finally, select the required values for NOISE (in the paper it's set up at either 0 or 0.3) and the ACTION_PENALTY (in the paper it's either 0 or 0.5). One key caveat is the the penalty is only used if the model operates in a complex environment.
8. Now the model training can be executed. The training time changes based on the number of TIMESTEPS inputted. As the training is completed, the model will output a dashboard (which is saved in the img/figures folder), a model (which is saved in the models folder) and logs (which are saved in the logs folder).
9. Once the training is complete we can move on to the testing cycle. In order to do that, the first thing is to switch from training mode to test mode by setting TRAIN = 0 and TEST =1, LOAD = 1. LOAD is used to load all of the pre-trained models which should have been saved in the models folder.

```
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"],
        TAG_ADD="E-4-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
```

10. Next the models to be tested need to be assigned to the MODEL variable.
11. Change the tag to match the one used in the training cycle. If LOAD is set to 1 then all models that match the string in TAG_ADD will be loaded.

12. Finally, run [main.py](#) to test the models
13. As the testing is completed, the model will output a dashboard (which is saved in the img/figures folder), and logs (which are saved in the logs folder) for each model.

4. Experiments

Replicated experiments:

We ran 1 replicated experiment and 1 exploratory experiment:

Experiment #1:

Experiment #1 aims at replicating the paper results with the same settings used in the paper to validate that the model outputs the same identical results.

The experiment was set up as follows:

- Used [SortEnv GitHub repo](#)
- Simulated via Gymnasium 0.29.1
- **Training Time Steps:** 100k timesteps per experiment (RBA is not trained)
- **Testing Time Steps:** 50
- **Action Modes:** Basic environment (10 speeds), Advanced environment (30 speed-mode pairs)
- **Input types:** Random (r) or Seasonal (s9)
- **Environments:** basic or advanced
- **Threshold** = 0.7
- **SEED** = 42
- **Noise** range 0 or 0.3
- **Action** penalty 0 or 0.5
- **4 models** tested: RBA (baseline mode), PPO, A2C, DQN
- **Training** running time: 5 min/experiment x 24 = 120 min
- **Testing** running time = 6 min

```

##### TRAIN #####

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-1-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-1-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-1-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-1-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-1-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-1-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-1-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-1-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-1-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-1-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-1-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-1-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-1-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-1-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-1-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-1-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-1-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-1-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-1-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-1-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-1-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-1-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-1-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-1-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)

##### TEST #####

run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-1-1")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-1-2")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-1-3")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-1-4")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-1-5")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-1-6")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-1-7")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-1-8")

```

Results:

The table below shows the side by side comparison of our results (100K timesteps total reward) and the paper results (paper reward). The performance metric to compare the performance is the

total reward which is calculated as the cumulative sum of all rewards obtained for each time step in the model testing phase. The columns ‘Delta paper Vs 100k exp’ shows the difference, in terms of total reward, between ours and the paper results.

									Experiment 1		
IDX	MODEL	TIME STEPS	Penalty	Input	Noise	Steps train	Threshold	Environment	100K TIMESTEPS total REWARD	PAPER REWARD	Delta paper Vs 100k exp
DEF-1-1	RBA	100,000	0	random	0	250	0.7	basic	26.56	26.56	0
DEF-1-1	DQN	100,000	0	random	0	250	0.7	basic	24.53	23.9	-0.63
DEF-1-1	PPO	100,000	0	random	0	250	0.7	basic	21.03	26.32	5.29
DEF-1-1	A2C	100,000	0	random	0	250	0.7	basic	26	24.39	-1.61
DEF-1-2	RBA	100,000	0	random	0	250	0.7	advanced	36.48	36.48	0
DEF-1-2	DQN	100,000	0	random	0	250	0.7	advanced	33.09	31.01	-2.08
DEF-1-2	PPO	100,000	0	random	0	250	0.7	advanced	24.37	35.29	10.92
DEF-1-2	A2C	100,000	0	random	0	250	0.7	advanced	33.24	34.12	0.88
DEF-1-3	RBA	100,000	0.5	seasonal 9	0	250	0.7	basic	11.95	11.95	0
DEF-1-3	DQN	100,000	0.5	seasonal 9	0	250	0.7	basic	25.42	23.46	-1.96
DEF-1-3	PPO	100,000	0.5	seasonal 9	0	250	0.7	basic	20.46	28.06	7.6
DEF-1-3	A2C	100,000	0.5	seasonal 9	0	250	0.7	basic	18.62	18.33	-0.29
DEF-1-4	RBA	100,000	0.5	seasonal 9	0	250	0.7	advanced	27.33	27.33	0
DEF-1-4	DQN	100,000	0.5	seasonal 9	0	250	0.7	advanced	34.92	30.23	-4.69
DEF-1-4	PPO	100,000	0.5	seasonal 9	0	250	0.7	advanced	28.02	36.85	8.83
DEF-1-4	A2C	100,000	0.5	seasonal 9	0	250	0.7	advanced	29.31	30.61	1.3
DEF-1-5	RBA	100,000	0	random	0.3	250	0.7	basic	19.77	19.77	0
DEF-1-5	DQN	100,000	0	random	0.3	250	0.7	basic	23.75	23.34	-0.41
DEF-1-5	PPO	100,000	0	random	0.3	250	0.7	basic	22.19	23.79	1.6
DEF-1-5	A2C	100,000	0	random	0.3	250	0.7	basic	23.74	23.1	-0.64
DEF-1-6	RBA	100,000	0	random	0.3	250	0.7	advanced	29.5	29.5	0
DEF-1-6	DQN	100,000	0	random	0.3	250	0.7	advanced	32.75	31	-1.75
DEF-1-6	PPO	100,000	0	random	0.3	250	0.7	advanced	20.47	33.62	13.15
DEF-1-6	A2C	100,000	0	random	0.3	250	0.7	advanced	29.97	32.74	2.77
DEF-1-7	RBA	100,000	0.5	seasonal 9	0.3	250	0.7	basic	10.21	10.21	0
DEF-1-7	DQN	100,000	0.5	seasonal 9	0.3	250	0.7	basic	21.27	22.98	1.71
DEF-1-7	PPO	100,000	0.5	seasonal 9	0.3	250	0.7	basic	17.59	21.62	4.03
DEF-1-7	A2C	100,000	0.5	seasonal 9	0.3	250	0.7	basic	18.33	18.33	0
DEF-1-8	RBA	100,000	0.5	seasonal 9	0.3	250	0.7	advanced	22.83	22.83	0
DEF-1-8	DQN	100,000	0.5	seasonal 9	0.3	250	0.7	advanced	31.89	27.96	-3.93
DEF-1-8	PPO	100,000	0.5	seasonal 9	0.3	250	0.7	advanced	30.61	35.53	4.92
DEF-1-8	A2C	100,000	0.5	seasonal 9	0.3	250	0.7	advanced	28.96	25.67	-3.29

Main findings:

- RBA matched the paper results in all experiments.
- Most RL agents performed well and achieved a total reward very close to the one in the paper with the exception of PPO which performed poorly in all experiments and did not return acceptable results (in most cases it returned a delta of 5 or more).
- The larger delta of the PPO agent is most likely due to the fact that the models were not optimized (tuning) during the training process.
- The tuning process is a highly time consuming process. Several experiments have shown that the tuning of a single model takes well over 8 hours making the model optimization highly challenging.
- Overall, the training-testing time required for the experiments above was of about 2 hours for 100.000 training time-steps; this is due to the fact that the Sorting_Env model has been programmed to be executed in the CPU but on our machines run on the GPU, resulting in much longer times than expected. Unfortunately we have been unable to find

an efficient way to switch device, for this reason we have decided to run experiment #2.

Experiment #2:

Due to the long times required to complete the pre-training of the models within 100.000k time-steps, we have decided to run a second experiment with the same identical conditions of experiment #1 except for the number of training time-steps which was set to 50.000. The hypothesis is that if the results are comparable then we can run the subsequent experiments with 50k time steps versus 100k resulting in a great time saving.

The experiment was set up as follows:

- Used [SortEnv GitHub repo](#)
- Simulated via Gymnasium 0.29.1
- **Training Time Steps:** 50k timesteps per experiment (RBA is not trained)
- **Testing Time Steps:** 50
- **Action** Modes: Basic environment (10 speeds), Advanced environment (30 speed-mode pairs)
- **Input** types: Random (r) or Seasonal (s9)
- **Environments:** basic or advanced
- **Threshold** = 0.7
- **SEED** = 42
- **Noise** range 0 or 0.3
- **Action** penalty 0 or 0.5
- **4 models** tested: RBA (baseline mode), PPO, A2C, DQN
- **Training** running time: 5 min/experiment x 24 = 120 min
- **Testing** running time = 6 min

```

##### TRAIN #####

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-2-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-2-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-2-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-2-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-2-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-2-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-2-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-2-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-2-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-2-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-2-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-2-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-2-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-2-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-2-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-2-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-2-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-2-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-2-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-2-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-2-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="DEF-2-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="DEF-2-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="DEF-2-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)

##### TEST #####

run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-2-1")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-2-2")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-2-3")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-2-4")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-2-5")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-2-6")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-2-7")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="DEF-2-8")

```

Results:

The table below expands the results table of experiment #1 to include the total reward for the experiment #2 (“50k TIMESTEPS total reward”) as well as the difference between the paper total reward and the total reward from experiment #2 (“Reward delta paper Vs 50k exp”).

IDX	MODEL	TIME STEPS	Penalty	Input	Noise	Steps train	Threshold	Environment	Experiment 1		Experiment 2	Delta paper Vs 100k exp	Reward delta paper Vs 50k exp
									100K TIMESTEPS total REWARD	PAPER REWARD	50K TIMESTEPS total REWARD		
DEF-1-1	RBA	100,000	0	random	0	250	0.7	basic	26.56	26.56	26.56	0	0
DEF-1-1	DQN	100,000	0	random	0	250	0.7	basic	24.53	23.9	23.83	-0.63	0.07
DEF-1-1	PPO	100,000	0	random	0	250	0.7	basic	21.03	26.32	22.01	5.29	4.31
DEF-1-1	A2C	100,000	0	random	0	250	0.7	basic	26	24.39	25.96	-1.61	-1.57
DEF-1-2	RBA	100,000	0	random	0	250	0.7	advanced	36.48	36.48	36.48	0	0
DEF-1-2	DQN	100,000	0	random	0	250	0.7	advanced	33.09	31.01	30.12	-2.08	0.89
DEF-1-2	PPO	100,000	0	random	0	250	0.7	advanced	24.37	35.29	21.61	10.92	13.68
DEF-1-2	A2C	100,000	0	random	0	250	0.7	advanced	33.24	34.12	32.62	0.88	1.5
DEF-1-3	RBA	100,000	0.5	seasonal 9	0	250	0.7	basic	11.95	11.95	11.95	0	0
DEF-1-3	DQN	100,000	0.5	seasonal 9	0	250	0.7	basic	25.42	23.46	23.15	-1.96	0.31
DEF-1-3	PPO	100,000	0.5	seasonal 9	0	250	0.7	basic	20.46	28.06	18.33	7.6	9.73
DEF-1-3	A2C	100,000	0.5	seasonal 9	0	250	0.7	basic	18.62	18.33	20.46	-0.29	-2.13
DEF-1-4	RBA	100,000	0.5	seasonal 9	0	250	0.7	advanced	27.33	27.33	27.33	0	0
DEF-1-4	DQN	100,000	0.5	seasonal 9	0	250	0.7	advanced	34.92	30.23	33.42	-4.69	-3.19
DEF-1-4	PPO	100,000	0.5	seasonal 9	0	250	0.7	advanced	28.02	36.85	30.61	8.83	6.24
DEF-1-4	A2C	100,000	0.5	seasonal 9	0	250	0.7	advanced	29.31	30.61	22.5	1.3	8.11
DEF-1-5	RBA	100,000	0	random	0.3	250	0.7	basic	19.77	19.77	19.77	0	0
DEF-1-5	DQN	100,000	0	random	0.3	250	0.7	basic	23.75	23.34	22.95	-0.41	0.39
DEF-1-5	PPO	100,000	0	random	0.3	250	0.7	basic	22.19	23.79	19.34	1.6	4.45
DEF-1-5	A2C	100,000	0	random	0.3	250	0.7	basic	23.74	23.1	23.37	-0.64	-0.27
DEF-1-6	RBA	100,000	0	random	0.3	250	0.7	advanced	29.5	29.5	29.5	0	0
DEF-1-6	DQN	100,000	0	random	0.3	250	0.7	advanced	32.75	31	29.94	-1.75	1.06
DEF-1-6	PPO	100,000	0	random	0.3	250	0.7	advanced	20.47	33.62	20.69	13.15	12.93
DEF-1-6	A2C	100,000	0	random	0.3	250	0.7	advanced	29.97	32.74	29.1	2.77	3.64
DEF-1-7	RBA	100,000	0.5	seasonal 9	0.3	250	0.7	basic	10.21	10.21	10.21	0	0
DEF-1-7	DQN	100,000	0.5	seasonal 9	0.3	250	0.7	basic	21.27	22.98	27.64	1.71	-4.66
DEF-1-7	PPO	100,000	0.5	seasonal 9	0.3	250	0.7	basic	17.59	21.62	18.33	4.03	3.29
DEF-1-7	A2C	100,000	0.5	seasonal 9	0.3	250	0.7	basic	18.33	18.33	17.59	0	0.74
DEF-1-8	RBA	100,000	0.5	seasonal 9	0.3	250	0.7	advanced	22.83	22.83	22.83	0	0
DEF-1-8	DQN	100,000	0.5	seasonal 9	0.3	250	0.7	advanced	31.89	27.96	26.27	-3.93	1.69
DEF-1-8	PPO	100,000	0.5	seasonal 9	0.3	250	0.7	advanced	30.61	35.53	26.43	4.92	9.1
DEF-1-8	A2C	100,000	0.5	seasonal 9	0.3	250	0.7	advanced	28.96	25.67	27.61	-3.29	-1.94

Main findings:

- RBA matched the paper results in all experiments.
- Most RL agents performed well and achieved a total reward very similar to the one in the paper with the exception of PPO which performed poorly in all experiments, exactly like in experiment #1, and did not return acceptable results (in most cases it returned a delta of 3 or more).
- All other RL agents returned a total reward similar to the one in experiment #1 or better.
- The larger delta of the PPO agent is most likely due to the fact that the models were not optimized (tuning) during the training process.
- Since the results from experiment #2 are very close to those obtained from experiment #1, we've decided to use this experiment as our baseline for all subsequent experiments which were executed with 50.000 training time-steps.
- The PPO model was discarded because it did not return acceptable results in any of the preliminary experiments and will not be used in experiment # 3 through #5.

New experiments:

Experiment #3:

In most real-world warehouses or distribution centers the sorting process exhibits a high complexity due to the process variation as well as to unpredictable product mix. Because of this, we believe that in order for Sorting_Env to be able to be deployed to an industrial setting, the model needs to be tested with a higher noise level. The default setting in the paper is either 0 or

0.3, and the authors tested all other values within that range (0, 0.1, 0.2, 0.3). In this experiment we will test the effect that higher noise levels (0.4, 0.6) have on Sorting_Env to test its reliability in more complex scenarios.

The experiment was set up as follows:

- Used [SortEnv GitHub repo](#)
- Simulated via Gymnasium 0.29.1
- **Training Time Steps:** 50k timesteps per experiment (RBA is not trained)
- **Testing Time Steps:** 50
- **Action Modes:** Basic environment (10 speeds), Advanced environment (30 speed-mode pairs)
- **Input** types: Random (r) or Seasonal (s9)
- **Environments:** basic or advanced
- **Threshold** = 0.7
- **SEED** = 42
- **Noise** range 0.4 or 0.6 (all experiments with 0 noise were discarded)
- **Action** penalty 0 or 0.5
- **3 models** tested: RBA (baseline mode), A2C, DQN
- **Training** running time: 5 min/experiment x 24 = 120 min
- **Testing** running time = 6 min

Noise = 0.4

```
##### TRAIN #####

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-1-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-1-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-1-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-1-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-1-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-1-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-1-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-1-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-1-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-1-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-1-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-1-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-1-5", COMPLEX=0, INPUT="r", NOISE=0.4, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-1-5", COMPLEX=0, INPUT="r", NOISE=0.4, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-1-5", COMPLEX=0, INPUT="r", NOISE=0.4, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-1-6", COMPLEX=1, INPUT="r", NOISE=0.4, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-1-6", COMPLEX=1, INPUT="r", NOISE=0.4, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-1-6", COMPLEX=1, INPUT="r", NOISE=0.4, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-1-7", COMPLEX=0, INPUT="s9", NOISE=0.4, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-1-7", COMPLEX=0, INPUT="s9", NOISE=0.4, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-1-7", COMPLEX=0, INPUT="s9", NOISE=0.4, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-1-8", COMPLEX=1, INPUT="s9", NOISE=0.4, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-1-8", COMPLEX=1, INPUT="s9", NOISE=0.4, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-1-8", COMPLEX=1, INPUT="s9", NOISE=0.4, ACTION_PENALTY=0.5, SEED=42)

##### TEST #####

run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-1-1")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-1-2")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-1-3")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-1-4")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-1-5")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-1-6")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-1-7")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-1-8")
```

Noise = 0.6

```
##### TRAIN #####

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-2-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-2-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-2-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-2-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-2-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-2-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-2-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-2-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-2-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-2-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-2-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-2-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-2-5", COMPLEX=0, INPUT="r", NOISE=0.6, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-2-5", COMPLEX=0, INPUT="r", NOISE=0.6, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-2-5", COMPLEX=0, INPUT="r", NOISE=0.6, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-2-6", COMPLEX=1, INPUT="r", NOISE=0.6, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-2-6", COMPLEX=1, INPUT="r", NOISE=0.6, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-2-6", COMPLEX=1, INPUT="r", NOISE=0.6, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-2-7", COMPLEX=0, INPUT="s9", NOISE=0.6, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-2-7", COMPLEX=0, INPUT="s9", NOISE=0.6, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-2-7", COMPLEX=0, INPUT="s9", NOISE=0.6, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-2-8", COMPLEX=1, INPUT="s9", NOISE=0.6, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-2-8", COMPLEX=1, INPUT="s9", NOISE=0.6, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-2-8", COMPLEX=1, INPUT="s9", NOISE=0.6, ACTION_PENALTY=0.5, SEED=42)

##### TEST #####

run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-2-1")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-2-2")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-2-3")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-2-4")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-2-5")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-2-6")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-2-7")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-2-8")
```

Results:

The results table below shows the results for experiment #3. As previously discussed, the conditions were tested only with RBA, A2C and DQN. The total reward baseline for experiment #2 (noise = 0.3) are in the green column (“baseline noise = 0.3 total reward”), whereas the results for the experiments with noise levels or 0.4 and 0.6 are in the yellow columns.

MODEL	TIMESTEPS	Penalty	Input	Steps train	Environment	Baseline Noise = 0.3 total reward	Noise = 0.4 total reward	Noise = 0.6 total reward
RBA	50,000	0	random	250	basic	19.77	16.69	13.33
DQN	50,000	0	random	250	basic	22.95	22.84	21.8
A2C	50,000	0	random	250	basic	23.37	22.49	18.7
RBA	50,000	0	random	250	advanced	29.5	26.22	21.9
DQN	50,000	0	random	250	advanced	29.94	30.31	29.86
A2C	50,000	0	random	250	advanced	29.1	29.58	27.04
RBA	50,000	0.5	seasonal 9	250	basic	10.21	9.42	6.19
DQN	50,000	0.5	seasonal 9	250	basic	27.64	26.72	22.23
A2C	50,000	0.5	seasonal 9	250	basic	17.59	20.46	17.59
RBA	50,000	0.5	seasonal 9	250	advanced	22.83	20.91	17.34
DQN	50,000	0.5	seasonal 9	250	advanced	26.27	31.57	26.24
A2C	50,000	0.5	seasonal 9	250	advanced	27.61	25.67	22.62

Main findings:

- **DQN** is the top performer under all tested noise levels, particularly in the advanced environment.
- **A2C** exhibits a solid performance and relatively stable behavior under increasing noise.
- **RBA** degrades sharply as the noise increases, making it impractical for real-world use unless in noise-free or simple operational environments with limited variation
- The experiment clearly shows that **RL agents A2C and DQN** have a higher adaptability advantage to noise over RBA, especially when using more complex environments or inputs.
- As observed in the previous experiments, all models tend to perform better when learning in advanced environments

Experiment #4:

In real-world industrial sorting systems, frequent changes in conveyor belt speed can result in greater mechanical wear, higher energy consumption, and potential disruptions to the sorting process. To address this, the action penalty in the simulation encourages the agent to maintain

more consistent speeds by discouraging unnecessary adjustments. This approach enhances the simulation's realism by reflecting operational constraints and promoting efficiency. By incentivizing the agent to recognize and adapt to input patterns rather than constantly altering speeds, the model more accurately mirrors practical industrial operations. It's important to note that action penalties only apply to seasonal input

The objective of this experiment is to test the model performance in advanced environments when higher action penalties are applied (0.6, 0.9) instead of the default 0.5 in order to see whether the model is capable of performing acceptably in operational conditions that have highly limited machine adjustment capabilities.

The experiment was set up as follows:

- Used [SortEnv GitHub repo](#)
- Simulated via Gymnasium 0.29.1
- **Training Time Steps:** 50k timesteps per experiment (RBA is not trained)
- **Testing Time Steps:** 50
- **Action Modes:** Basic environment (10 speeds), Advanced environment (30 speed-mode pairs)
- **Input types:** Random (r) or Seasonal (s9)
- **Environments:** basic or advanced
- **Threshold** = 0.7
- **SEED** = 42
- **Noise** range 0.4 or 0.6 (all experiments with 0 noise were discarded)
- **Action** penalty 0.6 or 0.9
- **3 models** tested: RBA (baseline mode), A2C, DQN
- **Training** running time: 5 min/experiment x 24 = 120 min
- **Testing** running time = 6 min

Performance metrics:

1. **Total reward**, as calculated in the previous experiments
2. **Speed belt adjustments:** measures as the the number of times a speed change is made by the model during the testing cycle

Action penalty= 0.6

```
##### TRAIN #####

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-3-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-3-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-3-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-3-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-3-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-3-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-3-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.6, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-3-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.6, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-3-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.6, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-3-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.6, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-3-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.6, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-3-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.6, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-3-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-3-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-3-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-3-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-3-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-3-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-3-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.6, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-3-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.6, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-3-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.6, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-3-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.6, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-3-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.6, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-3-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.6, SEED=42)
```

```
##### TEST #####

run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-3-1")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-3-2")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-3-3")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-3-4")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-3-5")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-3-6")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-3-7")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-3-8")
```


Action penalty = 0.9

```
##### TRAIN #####

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-4-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-4-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-4-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-4-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-4-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-4-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-4-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.9, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-4-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.9, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-4-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.9, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-4-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.9, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-4-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.9, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-4-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.9, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-4-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-4-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-4-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-4-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-4-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-4-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-4-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.9, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-4-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.9, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-4-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.9, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-4-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.9, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-4-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.9, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-4-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.9, SEED=42)

##### TEST #####

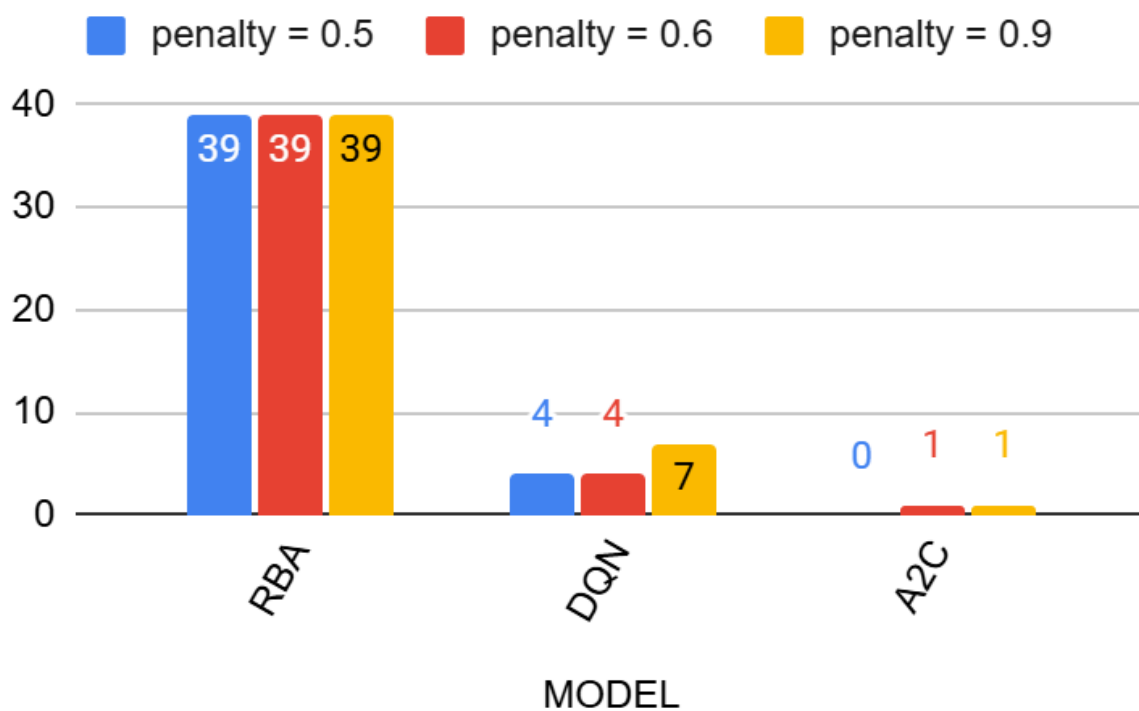
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-4-1")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-4-2")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-4-3")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-4-4")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-4-5")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-4-6")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-4-7")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-4-8")
```

Results:

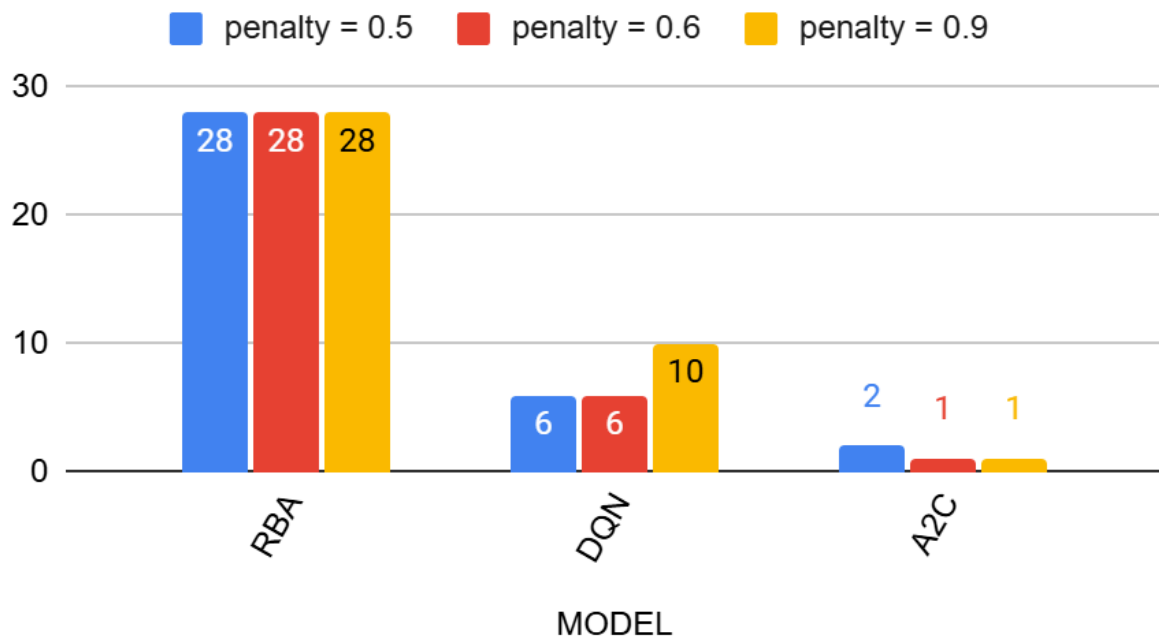
The results table below shows the results for experiment #4. The total reward baseline for experiment #2 (action penalty = 0.5) is in the green column (“baseline penalty = 0.5”), whereas the results for the experiments with penalty levels of 0.6 and 0.9 are in the yellow columns.

MODEL	TIMESTEPS	Input	Noise	Steps train	Threshold	Environment	Baseline penalty 0.5	Penalty = 0.6 total REWARD	Penalty = 0.9 total REWARD
RBA	50,000	seasonal 9	0	250	0.7	basic	11.95	8.06	-3.65
DQN	50,000	seasonal 9	0	250	0.7	basic	23.15	27.66	21.65
A2C	50,000	seasonal 9	0	250	0.7	basic	20.46	22.55	21.56
RBA	50,000	seasonal 9	0	250	0.7	advanced	27.33	24.53	16.13
DQN	50,000	seasonal 9	0	250	0.7	advanced	33.42	31.05	27.3
A2C	50,000	seasonal 9	0	250	0.7	advanced	22.5	27.63	22.38
RBA	50,000	seasonal 9	0.3	250	0.7	basic	10.21	6.71	-3.79
DQN	50,000	seasonal 9	0.3	250	0.7	basic	27.64	22.13	21.06
A2C	50,000	seasonal 9	0.3	250	0.7	basic	17.59	18.33	18.33
RBA	50,000	seasonal 9	0.3	250	0.7	advanced	22.83	19.93	11.23
DQN	50,000	seasonal 9	0.3	250	0.7	advanced	26.27	26.49	20.2
A2C	50,000	seasonal 9	0.3	250	0.7	advanced	27.61	24.96	21.42

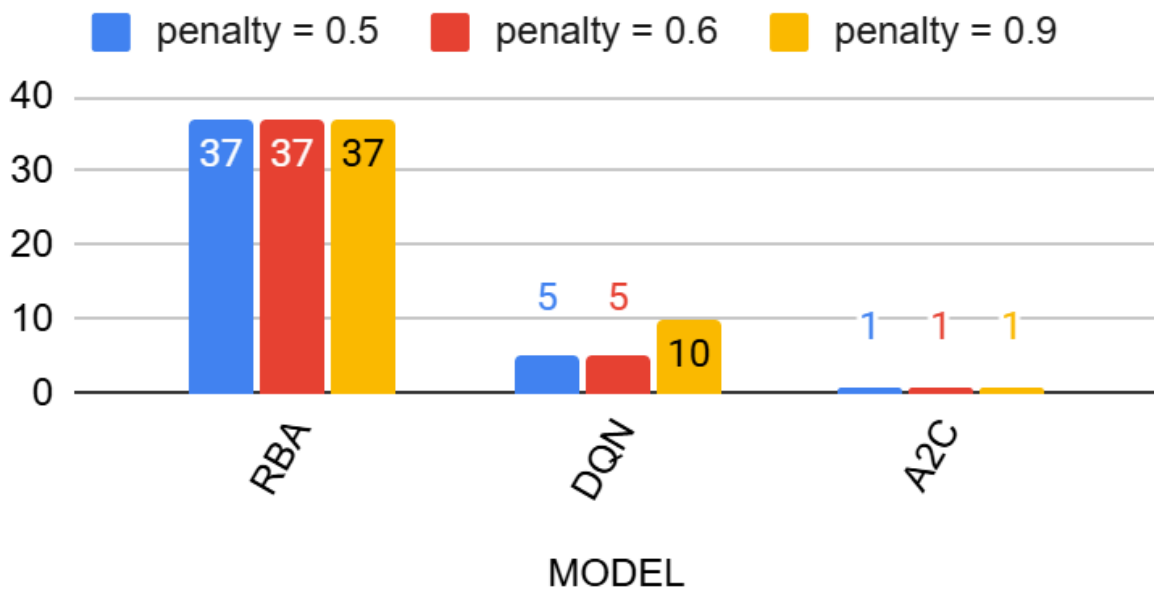
Number of speed adjustments (s9, noise =0, basic env)



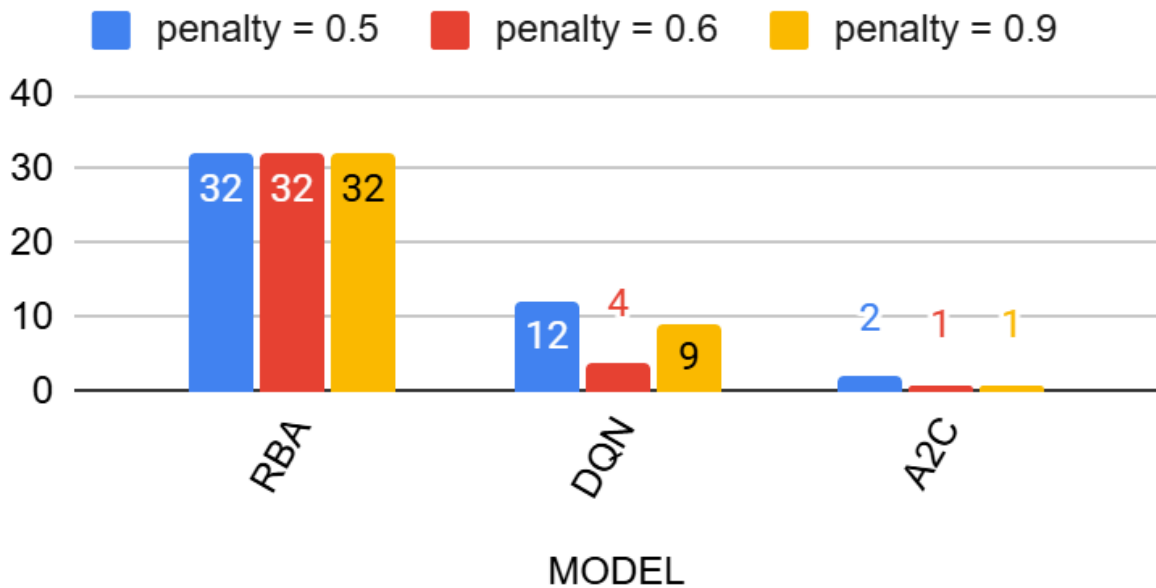
Number of speed adjustments (s9, noise =0, adv env)



Number of speed adjustments (s9, noise =0.3, basic env)



Number of speed adjustments (s9, noise =0.3, adv env)



Main findings:

- As observed in previous experiments, the highest total rewards are achieved when the agents operate in advanced environments
- DQN performed better than RBA and A2C with penalty of 0.6 in both basic and advanced environments, but its performance tends to drop when the penalty becomes more extreme
- A2C performed acceptably well at penalty levels of 0.6 and 0.9 indicating that this agent is more stable and adaptable, making it a strong candidate for environments with operational constraints or uncertainty.
- RBA is not competitive under most real-world-like conditions and fails under stricter constraints in both basic and advanced environments.
- RBA is unable to maintain the number of speed belt adjustments below an acceptable count during the testing cycle of 50 steps across all experimental conditions, resulting in lower total rewards and poor sorting performance
- A2C is the only model able to maintain the number of speed adjustments below 5 across all experiments, but its total reward is almost always lower than DQN, suggesting that the reduced number of speed adjustments is resulting in a lower sorting accuracy. DQN appears to be the model that provides the best balance in terms of speed belt adjustments, accuracy and total reward

Experiment #5:

The threshold defines a cutoff value for the sorting accuracy (α). If the actual accuracy falls below this threshold: the agent immediately receives a negative reward, specifically -0.1. This punishes poor sorting, regardless of the belt speed. Exploring different threshold values will allow us to check how tolerant the system is.

- A lower threshold should allow for more inaccuracies before punishing the agent and encourages faster belt speeds, since the penalty is less likely to be triggered.
- Higher thresholds require very high accuracy to avoid penalties and forces the agent to slow the belt when needed to ensure quality.

We tested the models with different threshold values (0.5, 0.9) and compared it to the total reward obtained with threshold value of 0.7 for all 3 models (RBA, A2C and DQN)

The experiment was set up as follows:

- Used [SortEnv GitHub repo](#)
- Simulated via Gymnasium 0.29.1
- **Training Time Steps:** 50k timesteps per experiment (RBA is not trained)
- **Testing Time Steps:** 50
- **Action Modes:** Basic environment (10 speeds), Advanced environment (30 speed-mode pairs)
- **Input types:** Random (r) or Seasonal (s9)
- **Environments:** basic or advanced
- **Threshold** = 0.5 or 0.9 (default = 0.7)
- **SEED** = 42
- **Noise** range 0.4 or 0.6 (all experiments with 0 noise were discarded)
- **Action** penalty 0.5
- **3 models** tested: RBA (baseline mode), A2C, DQN
- **Training** running time: 5 min/experiment x 24 = 120 min
- **Testing** running time = 6 min

Threshold = 0.5

```
COMPLEX = 0          # Complex Environment (1) or Simple Environment (0)
INPUT = "r"          # Input Type r=random, s3=simple_saisonal, s9=complex_saisonal
THRESHOLD = 0.7      # Threshold for Accuracy
NOISE = 0            # Noise Range (0.0 - 1.0)
```


Threshold = 0.9

```
##### TRAIN #####

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-6-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-6-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-6-1", COMPLEX=0, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-6-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-6-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-6-2", COMPLEX=1, INPUT="r", NOISE=0, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-6-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-6-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-6-3", COMPLEX=0, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-6-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-6-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-6-4", COMPLEX=1, INPUT="s9", NOISE=0, ACTION_PENALTY=0.5, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-6-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-6-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-6-5", COMPLEX=0, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-6-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-6-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-6-6", COMPLEX=1, INPUT="r", NOISE=0.3, ACTION_PENALTY=0, SEED=42)

run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-6-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-6-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-6-7", COMPLEX=0, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)


run_env(TRAIN=1, LOAD=0, MODELS=['A2C'], TAG_ADD="E-6-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['PPO'], TAG_ADD="E-6-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
run_env(TRAIN=1, LOAD=0, MODELS=['DQN'], TAG_ADD="E-6-8", COMPLEX=1, INPUT="s9", NOISE=0.3, ACTION_PENALTY=0.5, SEED=42)
```

```
##### TEST #####

run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-6-1")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-6-2")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-6-3")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-6-4")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-6-5")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-6-6")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-6-7")
run_env(TRAIN=0, LOAD=1, BENCHMARK=0, TEST=1, MODELS = ["RBA", "A2C", "PPO", "DQN"], TAG_ADD="E-6-8")
```

Results:

The results table below shows the results for experiment #5. The total reward baseline for experiment #2 (threshold = 0.7) is in the green column (“baseline threshold = 0.7 total reward”), whereas the results for the experiments with threshold levels of 0.5 and 0.9 are in the yellow columns.

MODEL 	TIMESTEP S	Penalty	Input	Noise	Steps train	Environment	Threshold = 0.5 total reward	Baseline Threshold = 0.7 total reward	Threshold = 0.9 total reward
RBA	50,000	0	random	0	250	basic	30.87	26.56	-5
DQN	50,000	0	random	0	250	basic	29.17	23.83	-5
A2C	50,000	0	random	0	250	basic	29.61	25.96	-5
RBA	50,000	0	random	0	250	advanced	37.43	36.48	31.64
DQN	50,000	0	random	0	250	advanced	33.16	30.12	25.46
A2C	50,000	0	random	0	250	advanced	33.52	32.62	24.35
RBA	50,000	0.5	seasonal 9	0	250	basic	18.46	11.95	-5
DQN	50,000	0.5	seasonal 9	0	250	basic	32.18	23.15	-5
A2C	50,000	0.5	seasonal 9	0	250	basic	22.25	20.46	-5
RBA	50,000	0.5	seasonal 9	0	250	advanced	29.86	27.33	16.84
DQN	50,000	0.5	seasonal 9	0	250	advanced	32.66	33.42	25.11
A2C	50,000	0.5	seasonal 9	0	250	advanced	29.83	22.5	23.86
RBA	50,000	0	random	0.3	250	basic	24.3	19.77	-5
DQN	50,000	0	random	0.3	250	basic	28.2	22.95	-5
A2C	50,000	0	random	0.3	250	basic	27.51	23.37	-5
RBA	50,000	0	random	0.3	250	advanced	32.65	29.5	23.56
DQN	50,000	0	random	0.3	250	advanced	32.26	29.94	26.66
A2C	50,000	0	random	0.3	250	advanced	34.14	29.1	25.86
RBA	50,000	0.5	seasonal 9	0.3	250	basic	16.2	10.21	-5
DQN	50,000	0.5	seasonal 9	0.3	250	basic	30.15	27.64	-5
A2C	50,000	0.5	seasonal 9	0.3	250	basic	22.25	17.59	-5
RBA	50,000	0.5	seasonal 9	0.3	250	advanced	23.34	22.83	17.69
DQN	50,000	0.5	seasonal 9	0.3	250	advanced	32.53	26.27	24
A2C	50,000	0.5	seasonal 9	0.3	250	advanced	28.12	27.61	20.11

Main findings:

- **Threshold 0.5** provides the highest rewards for most setups, especially in advanced environments, meaning less strict accuracy requirements lead to better performance. (favors speed over accuracy)
- **Threshold 0.9** severely penalizes the models learning in basic environments, resulting in negative rewards. This indicates that when the required accuracy is too high, the agent's learning process can be severely hindered (favors accuracy over speed)
- Interestingly, at a threshold of 0.9, all models operating in the basic environment fail whereas when operating in the advanced environment all models return quite high total rewards. This once more confirms the importance of training the models above in complex environments, probably due to the fact these provide much richer information the models can learn from in the training process.

5. Conclusion

What have you learned?

After running all of our experiments, we discovered that **noise levels** make it more challenging for all models to perform well. A2C and DQN are able to handle the noise levels better than PPO because A2C uses two networks to guide the learning and DQN learns from previous actions. For real world scenarios, like warehouses and factories, randomness is a big factor to consider, meaning models that can handle higher variation and lower predictability are better.

We also tested action penalties, which can allow changes in the belt speed and affect performance. DQN performed well when the action penalty was 0.6, meaning that the agent would be best for deployment in sorting operations with an average number of belt adjustments. DQN did perform worse however when the penalty was too high but A2C in a limited environment has the best result because of its two network setup.

Another thing we tested was the accuracy threshold. A lower threshold of 0.5 gives the models more flexibility and allows the model to make minor adjustments for long term performance and efficiency. All models are able to earn more positive rewards because of the threshold being at 0.5 and allows the models to perform at higher belt speeds and continue to stay relatively accurate. A higher threshold of 0.9 makes it harder for the models to perform well even in the basic environment where there is less information but in advanced environments the agent grabs more useful information to help learn and perform better.

Obstacles and solutions:

Some of the challenges we faced throughout this project was discovering how the model works within the code base. The logic behind the classes and functions are very complex and the code base is created in a very object oriented programming style. This made it a bit confusing figuring out exactly what we wanted to run, how to tune the model, what parameters to change, and a few other questions we had to discover.

Another challenge we encountered was our models not being tuned properly, specifically PPO and lead to poor performances in our experiments. There is another file in the utils folder called [tuning.py](#) with a class called Tuning and Optuna Tuning. By allowing the model to run with the tuning parameters it would take around 8 hours per model with a few iterations just because it would rerun the model with a bunch of different parameters automatically.

Another issue was SortingEnv code is built specifically for CPU and we were not able to configure our systems to run the model on our CPU instead we ran it on GPU. This caused the

experiments to run longer and our work around to this was to reduce the training time from 100k to 50k timesteps.

Applying RL to real-world problems:

To apply this RL project in a real world environment there are a few key things we wanted to highlight. We believe we would need to make the environment realistic, Watch Training time, Include Real-life expectations, Measure other aspects besides reward, and finally scalability. To make the environment realistic it is important to include random inputs by including different types of material to match real-world conditions. For training time it is important to find a balance between accuracy and speed since real world events have a strict timeline and need to fulfill the business needs. For real like limits it is important to make sure the belt speeds are not too fast for the machines or else it could potentially break the machines. When going over results it is important to make sure that the use of energy from the machines, maintenance costs for the machines, and accuracy is monitored and checked as well to ensure that the results are consistent. The final reason for scalability is that if a company were to use this RL approach within their company we would want to make sure that the code is applicable in multiple factories and warehouses and will still keep up with the accuracy and efficiency for the business.

SortingEnv with the use of RL is a powerful tool that can help warehouse business and factories with their sorting systems. To make it work in a real world environment it is important to make the experiments as realistic as possible and to have a lot of stress testing and quality control to ensure that the models will perform as expected.