SortingEnv: An Extendable RL-Environment for an Industrial Sorting Process

Paper **link**: 2503.10466
Authors: Tom Maus, Nico Zengeler, Tobias Glasmachers
Github: SortEnv
Group 4: Bruno Garofalo & Kerim Sever

Milestone 2

What We Completed

Core Work Done:

- Selected *Sorting_Env* paper (Tom Maus et al.)
- Delved deep into the main paper objective: Optimize sorting capability of industrial conveyor belt by balancing belt speed and accuracy using RL
- Defined MDP for both basic and advanced RL environments:
- Set up basic and advanced environments
- Ran trials with PPO model on 1,000 time steps in both environments
- Develop plan for project completion
- Created and introduced in-depth presentation of Sorting_Env Simulation report.pptx - Google Slides
- Conducted full experiment cycles with PPO, DQN, A2C

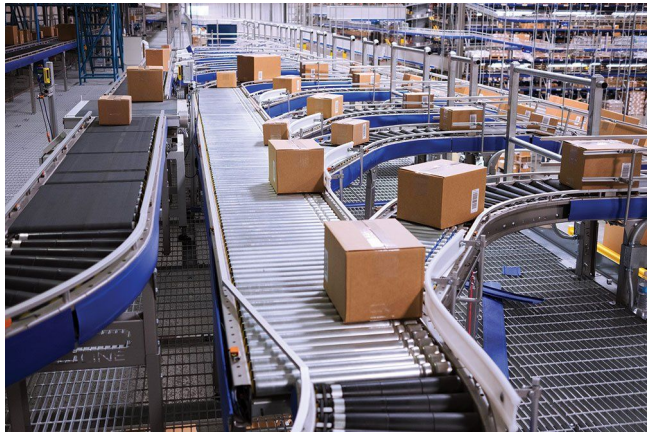
Feedback & Improvements

Feedback	Improvements
● Lack of clarity on the complexity difference between Basic and Advanced environments	● Described the basic and advanced environments in more visual terms in this report
● Needed clearer explanation of state, action, and reward representations	● Provided a more concise and clearer explanation of MPDs for both basic and advanced models in this report
● Presentation flow was dense and highly technical	● Will simplify the RL terminology for presentation clarity
● Professor: “Interpret the RL model results with more industrial context”	● Explained findings more in real-world terms
● Peer teams: Asked for more detailed comparison with the RBA baseline	● Results show side-by-side comparison between RBA and RL agents
● Needed stronger takeaway on real-world impact	● Explained findings more in real-world terms

The scenario

Complex industrial system using Reinforcement Learning (RL) to optimize **material sorting** performance based on:

1. ***Belt speed***
2. ***Material occupancy***
3. ***Sorting accuracy***



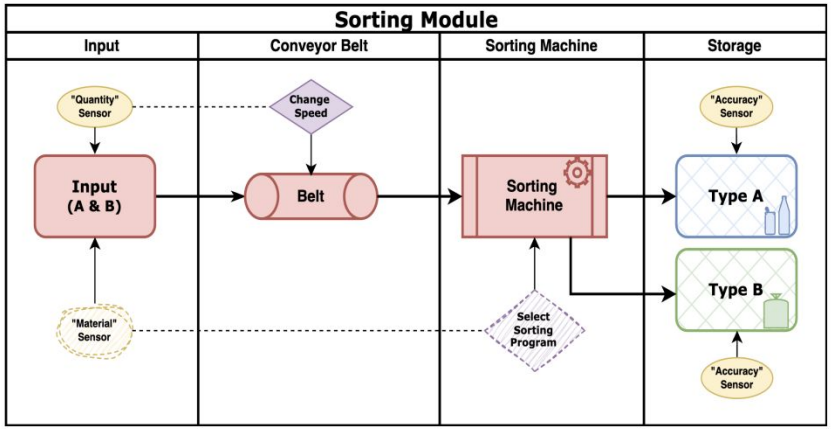
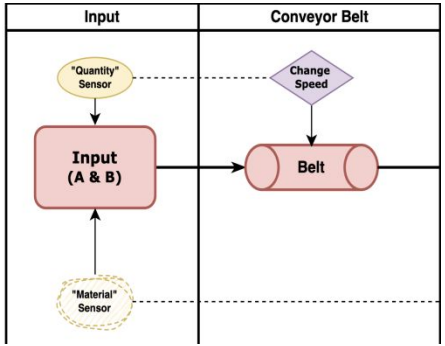


FIGURE 1. General overview of the Sorting Environment with the four compartments "Input", "Belt", "Sorting Machine" and "Storage". The simple environment is described by the upper elements for sensors (yellow) and actions (purple). The configuration for the advanced environment is shown by additional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

Paper **link**: 2503.10466
Authors: Tom Maus, Nico Zengeler, Tobias Glasmachers
Github: SortEnv

Model Stages



Input stage: random mix of material A and B added to the belt

Input total = qty A + qty B

Input generation mechanisms for simulating the sorting environment:

1. **Random input generator**
2. **Seasonal pattern generator**

Belt stage: amount of material present on the belt at each time step

Occupancy = qty A + qty B

Model Stages

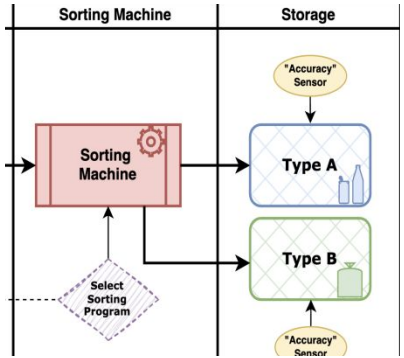
Sorting Stage:

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

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

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

α represents the proportion of correctly sorted material A and B



Basic Vs Advanced Environments

Basic:

- Only 1 sorting mode
- Action space given by 10 possible belt speeds

Advanced:

- 3 different sorting modes selected based on material ratio
- Action space 10 belt speeds x 3 possible modes = 30 possible actions t

Markov Decision Process Formulation - Base Environment

(MDP) = (S,A,P,R,γ)

States representation (S):

- Represents the total amount of material on the input belt (qty A + qty B)

Actions representation (A):

- Ten discrete actions corresponding to belt speeds of 10% to 100%.

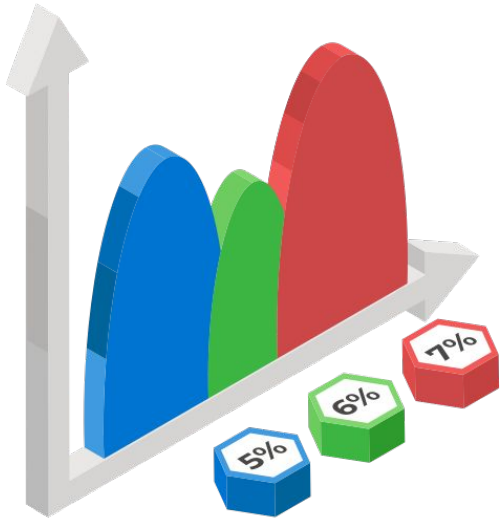
$$a \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$$

Markov Decision Process Formulation - Base Environment

Transition probabilities (P):

- The random noise added to the system makes the transitions stochastic

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



Markov Decision Process Formulation - Base Environment

Reward function (R):

- The agent must dynamically adjust and find the **optimal speed** in each setup
- The reward is calculated based on the **accuracy** for the material currently on the belt, and the belt **speed** *v*
- All values below a minimum accuracy **threshold** (0.7) will return a negative reward of -0.1 immediately
- A weight for the importance of both components can be set with the r**eward factors** r_{acc} and r_{speed}
- The **penalty** can be applied to limit the total number of speed changes in input-setups that are not fully random

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

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

1. α represents the sorting accuracy
2. O_{v_max} is the maximum belt occupancy limit per belt speed
3. **λ** is the accuracy abatement rate
4. **Noise factor** introduced to account for variability

Markov Decision Process Formulation - Advanced Environment

Premise:

- The basic environment allows only for discrete belt speed adjustments
- Advanced environment simulated a machine upgrade by introducing three sorting modes that depend on the ration of material A and B:

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

Markov Decision Process Formulation - Advanced Environment

Actions representation:

- Action space expanded to include a total of 30 actions: ten discrete actions corresponding to belt speeds of 10% to 100% x 3 sorting modes

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

States representation:

- Total amount of material on the input belt
- Ratio of material A to material B

$$\text{Input total} = \text{qty A} + \text{qty B}$$

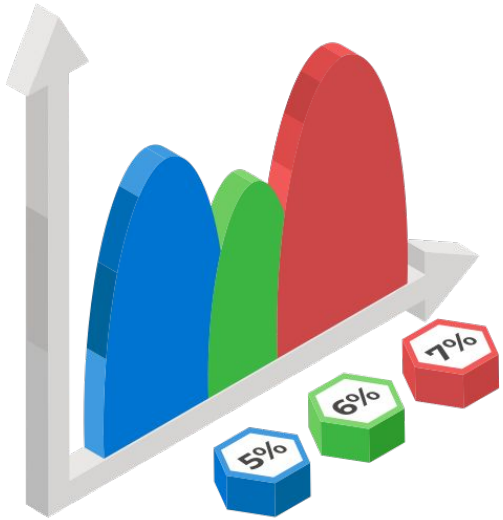
$$\text{Material ratio} = \text{qty A} / \text{qty B}$$

Markov Decision Process Formulation - Advanced Environment

Transition probabilities:

- The random noise added to the system makes the transitions stochastic

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



Markov Decision Process Formulation - Advanced Environment

Reward function:

- The reward function remains the same
- Sorting accuracy in the advanced environment is influenced by **belt speed**, **occupancy**, and the selected **sorting mode**

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

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

Challenges & How We Solved Them

Key challenges	Our solutions
● Poor documentation in original paper's code	● Switched to another paper which was more structured and had an easier-to-use GitHub repo
● Learning curve required to learn model and understand how to run it effectively	● Trial and error
● Training and testing of RL algorithms takes many hours per experiment	● Limited iterations for performance testing; reduced training time steps to 50k

Replicated Experiments

Markov Decision Process Formulation

Reward function (R):

- The agent must dynamically adjust and find the **optimal speed** in each setup
- The reward is calculated based on the **accuracy** for the material currently on the belt, and the **belt speed** *v*
- All values below a minimum accuracy **threshold** (0.7) will return a negative reward of -0.1 immediately
- A weight for the importance of both components can be set with the r**eward factors** r_{acc} and r_{speed}
- The **penalty** can be applied to limit the total number of speed changes in input-setups that are not fully random

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

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

1. α represents the sorting accuracy
2. O_{v_max} is the maximum belt occupancy limit per belt speed
3. **λ** is the accuracy abatement rate
4. **Noise factor** introduced to account for variability

Performance metric

Total reward

- Calculated as cumulative sum of all reward obtained for each time step in the model testing phase

```
total_reward = np.cumsum(reward_data['Reward'])
```

Experiment #1: Purpose & Setup

Objective:

- Replicate paper results using same parameters and settings

Why This Matters:

- Validate models run locally return the same results presented in the original paper

Our Setup:

- Used SortEnv GitHub repo
- Simulated via Gymnasium 0.29.1
- **Input** types: Random (r) or Seasonal (`s9`)
- **Environments**: basic or advanced (see detailed MDP description here Simulation report.pptx - Google Slides)
- **Action** Modes: Basic environment (10 speeds), Advanced environment (30 speed-mode pairs)
- **Total Reward**: Combines belt speed + sorting accuracy
    a. **Sorting Purity**: Accuracy of correctly sorted items (precision)
    b. **Mean Belt Speed**: Efficiency metric (higher = faster sorting)
- **Training Time Steps**: 100k timesteps per experiment (RBA is not trained)
- **Testing Time Steps:** 50
- 4 **models** tested: RBA (baseline mode), PPO, A2C, DQN
- **Training** running time: 5 min/experiment x 32 = 160 min
- **Testing** running time = 6 min

Output Interpretation:

- Evaluated al 4 models with 8 experimental conditions with
- Most models performed well and achieved a mean reward very close to the one in the paper
- PPO is the only model that performed poorly and did not return acceptable results
- This is probably due to the lack of model tuning

Challenges:

- Model training with 100k time steps takes 2.5 hours (~ 5 min/experiment x 32)
- Model tuning extremely time consuming (over 8 hours)

									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

Experiment #2: Purpose & Setup

Objective:

- Test the mean rewards obtained by training models in 50k time steps rather than 100k to find out whether or not the results are comparable

Why This Matters:

- If the results are comparable then we can run the subsequent experiments with 50k time steps Vs 100k resulting in a great time saving

Our Setup:

- Used SortEnv GitHub repo
- Simulated via Gymnasium 0.29.1
- Input types: Random (r) or Seasonal (`s9`)
- Environments: basic or advanced (see detailed MDP description here Simulation report.pptx - Google Slides)
- Action Modes: Basic environment (10 speeds), Advanced environment (30 speed-mode pairs)
- Total Reward: Combines belt speed + sorting accuracy
    a. Sorting Purity: Accuracy of correctly sorted items (precision)
    b. Mean Belt Speed: Efficiency metric (higher = faster sorting)
- **Training Time Steps**: 50k timesteps per experiment (RBA is not trained)
- **Testing Time Steps:** 50
- 4 models tested: RBA (baseline mode), PPO, A2C, DQN
- **Training** running time: 2.5 min/experiment x 32 = 80 min
- **Testing** running time = 6 min

Output Interpretation:

- Evaluated al 4 models with 8 experimental conditions with
- Most models performed well and achieved a mean reward very close to the one in the paper
- In some cases some models performed better
- PPO is the model that consistently performed poorly and did not return acceptable results
- A2C did not perform well in DEF-1-4

									Experiment 1		Experiment 2		
IDX	MODEL	TIME STEPS	Penalty	Input	Noise	Steps train	Threshold	Environment	100K TIMESTEPS total REWARD	PAPER REWARD	50K TIMESTEPS total REWARD	Delta paper Vs 100k exp	Reward delta paper Vs 50k exp
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

Lessons learned

- The results obtained with 50k training steps except for PPO which will not be used in the next experiments
- Model training with 100k time steps takes 2.5 hours (~ 5 min/experiment x 32)
- Model tuning is extremely time consuming (over 8 hours)
- Most models performed well and achieved a total reward very close to the one in the paper
- Model tuning process takes over 8 hours
- Surprised to see that PPO did not perform well
- The main challenge was understanding to get the models to work. Initially we could not get the results for RBA. Then we learned that we had to train first, then load trained model and test
- Models are designed to be run on CPU but do actually run on GPU. slowing down training process. For some reason we have been unable to switch device

New Experiments

Experiment #3: Purpose & Setup

Objective:

- Test models with increased noise level: baseline = 0.3, new noise levels 0.4 & 0.6

Why This Matters:

- The default noise level used in the paper is 0.3.
- Tested noise level in tuning process between 0 and 0.3.
- The noise level of 0.3 might be too low in some scenarios such as highly complex warehouses or distribution centers.
- We want to find out whether the models can handle higher levels of noise so that they can be used in more complex scenarios where there is more process variability

Experiment #3: Purpose & Setup

Our Setup:

- Used SortEnv GitHub repo
- Simulated via Gymnasium 0.29.1
- **Input** types: Random (r) or Seasonal (`s9`)
- **Environments**: basic or advanced (see detailed MDP description here Simulation report.pptx - Google Slides)
- **Action** Modes: Basic environment (10 speeds), Advanced environment (30 speed-mode pairs)
- **Total Reward**: Combines belt speed + sorting accuracy
    a. **Sorting Purity**: Accuracy of correctly sorted items (precision)
    b. **Mean Belt Speed**: Efficiency metric (higher = faster sorting)
- **Steps**: 50k timesteps per experiment
- 3 **models** tested: RBA, A2C, DQN
- Mean reward of **experiment #2** used as baseline results
- **Training Time Steps**: 50k timesteps per experiment (RBA is not trained)
- **Testing Time Steps:** 50
- **Noise levels tested**: 0.4, 0.6 (all experimental conditions with 0 noise discarded)
- **Training** running time: 2.5 min/experiment x 32 = 80 min
- **Testing** running time = 6 min

Technical challenges:

- Training models takes a very very long time

Output Interpretation:

- **DQN** is the top performer under all tested noise levels and penalties, particularly in the advanced environment.
- **A2C** exhibits a solid performance and relatively stable behavior under increasing noise.
- **RBA** degrades sharply with added noise or constraints, making it impractical for real-world use unless in noise-free or unconstrained settings.
- Introducing penalties and noise reveals the adaptability advantage of deep **RL models**, especially when using more complex environments or inputs.

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

Experiment 4 & 5 background - refresh

Reward function (R):

- The agent must dynamically adjust and find the **optimal speed** in each setup
- The reward is calculated based on the **accuracy** for the material currently on the belt, and the **belt speed** *v*
- All values below a minimum accuracy **threshold** (0.7) will return a negative reward of -0.1 immediately
- A weight for the importance of both components can be set with the r**eward factors** r_{acc} and r_{speed}
- The **penalty** can be applied to limit the total number of speed changes in input-setups that are not fully random

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

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

1. α represents the sorting accuracy
2. O_{v_max} is the maximum belt occupancy limit per belt speed
3. **λ** is the accuracy abatement rate
4. **Noise factor** introduced to account for variability

Experiment #4: Purpose & Setup

Background: The action penalty can be applied to limit the total number of speed changes in input-setups that are not fully random, e.g. the seasonal input to enhance the realism of the simulation by encouraging the agent to maintain consistent speeds. Action penalty only applies to seasonal input

Objective:

- Test models with different action penalties: baseline = 0.5, new values: 0.6, 0.9

Why This Matters:

- 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.

Experiment #4: Purpose & Setup

Our Setup:

- Used SortEnv GitHub repo
- Simulated via Gymnasium 0.29.1
- **Input** types: Seasonal (s9)
- **Environments**: basic or advanced (see detailed MDP description here Simulation report.pptx - Google Slides)
- **Action** Modes: Basic environment (10 speeds), Advanced environment (30 speed-mode pairs)
- **Total Reward**: Combines belt speed + sorting accuracy
    a. **Sorting Purity**: Accuracy of correctly sorted items (precision)
    b. **Mean Belt Speed**: Efficiency metric (higher = faster sorting)
- **Steps**: 50k timesteps per experiment
- 3 **models** tested: RBA, A2C, DQN
- Mean reward of **experiment #2** used as baseline results
- Action **penalties**: baseline = 0.5, new values: 0.6, 0.9
- **Training** running time: 2.5 min/experiment x 32 = 80 min
- **Testing** running time = 6 min

Technical challenges:

- Training takes a long

Output Interpretation:

- **DQN** is the strongest performer but more **sensitive to high penalties** and noise.
- **A2C** 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.

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	0.47
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

Experiment #5: Purpose & Setup

Background: 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.

Objective:

- Test models with different threshold values: baseline = 0.7, new values: 0.5, 0.9

Why This Matters:

- 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.

Experiment #5: Purpose & Setup

Our Setup:

- Used SortEnv GitHub repo
- Simulated via Gymnasium 0.29.1
- **Input** types: Seasonal (s9)
- **Environments**: basic or advanced (see detailed MDP description here Simulation report.pptx - Google Slides)
- **Action** Modes: Basic environment (10 speeds), Advanced environment (30 speed-mode pairs)
- **Total Reward**: Combines belt speed + sorting accuracy
    a. **Sorting Purity**: Accuracy of correctly sorted items (precision)
    b. **Mean Belt Speed**: Efficiency metric (higher = faster sorting)
- **Steps**: 50k timesteps per experiment
- 3 **models** tested: RBA, A2C, DQN
- Mean reward of **experiment #2** used as baseline results
- Threshold **values**: baseline = 0.7, new values: 0.5, 0.9
- **Training** running time: 2.5 min/experiment x 32 = 80 min
- **Testing** running time = 6 min

Technical challenges:

- Training takes a long

Output Interpretation:

- **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 in basic environments, leading to negative rewards, which indicates that too high accuracy demands can hinder the agent's learning process (favors accuracy over speed)

MODEL	TIMESTEPS	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

Lessons learned

- All models appear to be somewhat affected by increasing **noise levels**, although RL agents seem to be more resilient, as indicated in the paper. This might pose challenges for the **real-world application** of these models, as large warehouses or distribution centers typically involve processes with higher variation and lower predictability.

- The real-world application of the **action penalty experiment** is particularly interesting. The action penalty is intended to reduce the number of **belt speed changes**, which can, over time, increase wear and tear on the equipment. The ideal RL agent is one that performs well even under high action penalties, in this case, A2C.

- A lower **threshold** corresponds to a more forgiving accuracy requirement. With a lower threshold (0.5), all models have an easier time earning positive rewards, since they are less likely to fall below the threshold, even when operating at suboptimal belt speeds. This enables the agent to explore a broader range of belt speeds, **potentially favoring higher speeds** that increase throughput, even at the expense of some sorting precision.

- Before submitting the final report, we would like to complete the **model tuning**, which we have not yet finished, and re-run all experiments. This will (hopefully!) allow us to better match the results reported in the paper. This will require running the models on the **CPU** rather than GPU

Incomplete Experiments

Model tuning

Purpose

- Fine-tune hyperparameters (learning rate, gamma, entropy coefficient, etc.) to improve model performance.
- Essential to replicate paper results more closely and enhance generalization in noisy or constrained environments.

Setup

- Use `Tuning` module in `utils/tuning.py`
- Parameters tested using Optuna-based tuning loop
- Experiment involves:
    - RL agents: A2C, PPO, DQN
    - Tuning on both basic and advanced environments
    - 50k timesteps per run × 32 combinations = high runtime demand
    - CPU-only (GPU not supported in current setup)

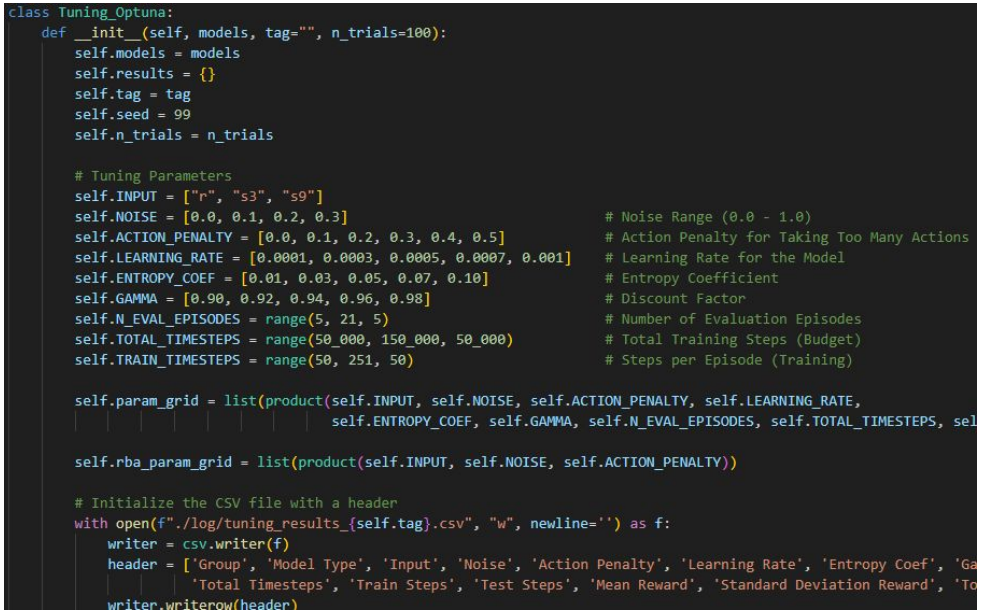
Model tuning

What We Would Want to do In the Future

- Comparison of tuned vs. baseline model results
- Performance metrics: total reward, stability, convergence time
- Visuals showing performance before/after tuning
- Explanation of best-performing hyperparameter combinations

Focus Areas for Tuning

- **Learning Rate Schedules:** Test static vs. dynamic (e.g., linear decay, exponential decay)
- **Entropy Coefficient Annealing:** Gradually reduce exploration over time
- **Dynamic Action Penalties:** Start low and increase over episodes to simulate mechanical wear scenarios
- **Threshold Sensitivity:** Tune penalty trigger levels (accuracy cutoffs) to balance quality vs. throughput



```python
class Tuning_Optuna:
    def __init__(self, models, tag="", n_trials=100):
        self.models = models
        self.results = {}
        self.tag = tag
        self.seed = 99
        self.n_trials = n_trials

        # Tuning Parameters
        self.INPUT = ["r", "s3", "s9"]
        self.NOISE = [0.0, 0.1, 0.2, 0.3]                              # Noise Range (0.0 - 1.0)
        self.ACTION_PENALTY = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]           # Action Penalty for Taking Too Many Actions
        self.LEARNING_RATE = [0.0001, 0.0003, 0.0005, 0.0007, 0.001]   # Learning Rate for the Model
        self.ENTROPY_COEF = [0.01, 0.03, 0.05, 0.07, 0.10]             # Entropy Coefficient
        self.GAMMA = [0.90, 0.92, 0.94, 0.96, 0.98]                    # Discount Factor
        self.N_EVAL_EPISODES = range(5, 21, 5)                         # Number of Evaluation Episodes
        self.TOTAL_TIMESTEPS = range(50_000, 150_000, 50_000)          # Total Training Steps (Budget)
        self.TRAIN_TIMESTEPS = range(50, 251, 50)                      # Steps per Episode (Training)

        self.param_grid = list(product(self.INPUT, self.NOISE, self.ACTION_PENALTY, self.LEARNING_RATE,
                                       self.ENTROPY_COEF, self.GAMMA, self.N_EVAL_EPISODES, self.TOTAL_TIMESTEPS, sel

        self.rba_param_grid = list(product(self.INPUT, self.NOISE, self.ACTION_PENALTY))

        # Initialize the CSV file with a header
        with open(f"./log/tuning_results_{self.tag}.csv", "w", newline='') as f:
            writer = csv.writer(f)
            header = ['Group', 'Model Type', 'Input', 'Noise', 'Action Penalty', 'Learning Rate', 'Entropy Coef', 'Ga
                      'Total Timesteps', 'Train Steps', 'Test Steps', 'Mean Reward', 'Standard Deviation Reward', 'To
            writer.writerow(header)
```

Model tuning

Optimization Strategy

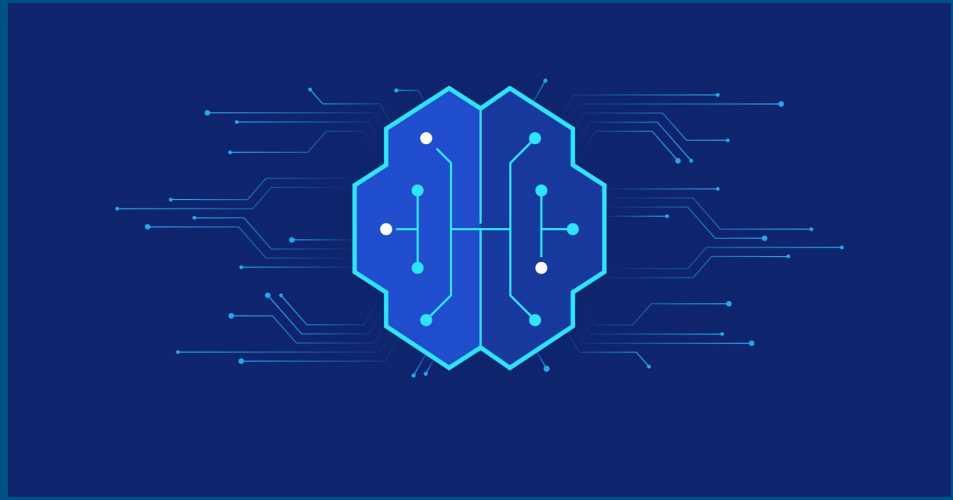
- Use **Optuna** to automate search across multiple parameters
- Enable **early stopping (pruning)** to cut unpromising runs
- Apply **model-specific tuning**:
    - PPO: clip range, learning rate, target KL
    - A2C: value/entropy loss weights
    - DQN: learning rate decay, replay buffer size

Why This Matters

- Dynamic hyperparameters improve adaptability over time
- Better generalization to noisy and constrained environments
- Helps finalize the most deployable model

```python
class Tuning:
    def __init__(self, models, tag="", n_trials=100):
        self.models = models
        self.results = {}
        self.tag = tag
        self.seed = 99
        self.n_trials = n_trials

        # Initialize the CSV file with a header
        with open(f"./log/tuning_results_{self.tag}.csv", "w", newline='') as f:
            writer = csv.writer(f)
            header = ['Group', 'Model Type', 'Input', 'Noise', 'Action Penalty', 'Learning Rate', 'Entropy Coef', 'Ga
                      'Total Timesteps', 'Train Steps', 'Test Steps', 'Mean Reward', 'Standard Deviation Reward', 'To
            writer.writerow(header)

        # Tuning Parameters
        # Input Type r=random, s3=simple_saisonal, s9=complex_saisonal
        self.INPUT = ["r", "s3", "s9"]
        self.NOISE = [0.0, 0.1, 0.2, 0.3]                           # Noise Range (0.0 - 1.0)
        self.ACTION_PENALTY = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]        # Action Penalty for Taking Too Many Actions
        self.LEARNING_RATE = [0.0001, 0.0003, 0.0005, 0.0007, 0.001] # Learning Rate for the Model
        self.ENTROPY_COEF = [0.01, 0.03, 0.05, 0.07, 0.10]          # Entropy Coefficient
        self.GAMMA = [0.90, 0.92, 0.94, 0.96, 0.98]                 # Discount Factor
        self.N_EVAL_EPISODES = range(5, 21, 5)                      # Number of Evaluation Episodes
        self.TOTAL_TIMESTEPS = range(100_000, 200_001, 50_000)      # Total Training Steps (Budget)
        self.TRAIN_TIMESTEPS = range(50, 251, 50)                   # Steps per Episode (Training)

        self.param_grid = list(product(self.INPUT, self.NOISE, self.ACTION_PENALTY, self.LEARNING_RATE,
                                       self.ENTROPY_COEF, self.GAMMA, self.N_EVAL_EPISODES, self.TOTAL_TIMESTEPS, self.TRAIN_

        self.rba_param_grid = list(product(self.INPUT, self.NOISE, self.ACTION_PENALTY))
```

Project Reflections and Future Applications



Key Learnings

Technical Skills Gained

- RL frameworks: Stable-Baselines3, Gymnasium
- MDP design for real-world simulation
- Environment configuration and benchmarking techniques
- Working on a professional RL model
- Hands-on debugging of RL pipelines

Conceptual Insights

- Reward shaping and penalty design significantly impact learning dynamics
- PPO is sensitive to hyperparameters; A2C offers stability under constraints
- Simulated constraints (noise, penalties, thresholds) mimic real-world industrial trade-offs well

Applying RL to Real-World Problems

Key Considerations

- **Sensor noise & real-time variability** → Must be modeled as stochasticity
- **Reward structures** → Should reflect true cost functions (e.g., energy, wear)
- **Environment realism** → State/action space should align with hardware/operational limits
- **Training time** → Must balance exploration depth with business timelines
- **Hardware limitations** → Need GPU access or cloud resources for scalability

Practical Takeaway

> “It's not enough for the model to *work*—it must also make *operational sense*.”

Final Thoughts

Project Experience

- Tackled a deeply technical topic and made it tangible
- Experienced end-to-end RL experimentation: from setup to simulation, tuning, and analysis
- Developed confidence in dissecting and improving existing research environments

Message to Class

- Don't be discouraged by the complexity of RL—once the pipeline runs, the rest is refinement and enhancements
- Debugging RL setups is **half the challenge** (and half the fun)

Quote to Close

> “The best models don't just perform—they adapt. And that's what makes RL powerful.”