RL in Elevator Dispatch

Course Project Final Proposal
Reinforcement Learning and Decision Making Under Uncertainty,
University of Neuchatel, Spring 2025

William Dan*
University of Bern
william.dan@students.unibe.ch

23 May 2025

1 Introduction

Elevator dispatch is an NP-hard, stochastic scheduling task where even recent commercial heuristics still leave average passenger waits above 30s at peak traffic. Cutting those few extra seconds scales to thousands of saved person-hours, lower energy and higher tenant-satisfaction scores.

1.1 Brief explanation of what I do

The project will:

- model elevator group control as an MDP, implement a Gymnasium simulator,
- benchmark different RL models against the best heuristic optimization baselines.
- visualize the result and analyze the behavior of models

The state space and action space can be very small if we constraint the number of floors and elevators, also can be very large if we increase the number. I use small settings so that it's feasible for training.

All the codes are public on github. From this citation [Dan25], you can find the repository.

1.2 Disclaimer

Although there is paper regarding this topic [Wei+20], I cannot find their implementation of environment and RL model. I finish the environment, the baseline algorithm and the RL models all by myself. Generally speaking, the result is not good. But in some cases, RL models outperform baseline. I will analyze these cases and also mention if we design the reward differently, the RL models will have some shortcuts to get the higher rewards without learning anything.

2 Related Work

In elevator dispatch, there are already some mature heuristic algorithms. However, there are still some space in performance for improvement.[1] I have implemented FIFO and LOOK algorithm in the environment.

*University of Bern, 3012 CH-Bern, Switzerland.

Table 1. Classical algorithms of elevator dispatch

Algorithm	Description	Pros	Cons
FIFO (First-In-First-Out)	Processes requests in the order they are received.	• Simple to implement. • No starvation.	• Inefficient for scattered requests. • Longer wait times due to lack of prioritization.
SCAN	Moves in one direction, serves all requests, reverses at the end.	• Balanced load distribution. • Efficient for dense requests.	• Longer wait for opposite-direction requests. • Unnecessary stops at extreme floors.
LOOK	Similar to SCAN but reverses direction when no requests ahead.	• Avoids unnecessary end-floor stops. • Better efficiency than SCAN.	• Complex to implement. • Still delays reverse-direction requests.

3 Problem Statement

We model the problem into MDP, using the model from the previous work [Wei+20]. The state space is depicted in table 2[2] and the action space is depicted in table 3[3]. In the table 4, we show that the state space in this model is very large [4]. So, we use deep RL to solve this problem.

Table 2. MDP state representation for an N-floor, M-car elevator group

Symbol	Shape	Value / Type	Description
$\bar{B}$	$N \times M \times 2$	$\mathbb{R}_{\geq 0}$	*Hall-call matrix* after direction replication and truncation; $\bar{b}_{i,j}$ is cumulative (or binary) wait of callers at floor i in the travel direction of car j.
A	$N \times M$	$\{0, 1\}$	*Car-call matrix*; $a_{i,j} = 1$ iff a passenger in car j wishes to alight at floor i.
p	M	$\{1, \ldots, N\}$	Discrete current floor index of each car.
d	M	$\{-1, 0, 1\}$	Travel direction of each car (-1 down, 0 idle, $+1$ up).

The four tensors are stacked as $(\bar{B}, A, P, D)$, making a 3-D array of shape $(N, M, 5)$ which is then
flattened to a feature vector of length $5NM$ for the neural policy.

Table 3. MDP action space

Symbol	Domain / Encoding	Cardinality	Interpretation
$a = (i, j)$	$i \in \{1, \dots, N\},\ j \in \{1, \dots, M\}$	$\lvert\mathcal{A}\rvert = N \times M$	"Dispatch car j to stop next at floor i" (subject to non-reversal and capacity constraints).

Table 4. Size explosion of the elevator MDP (binary buttons)

Parameter	Formula	8 floors, 3 cars	20 floors, 4 cars
Hall calls	2^{2N}	2^{16}	2^{40}
Car calls	2^{NM}	2^{24}	2^{80}
Car positions	N^M	8^3	20^4
Car directions	3^M	3^3	3^4
State space $\lvert\mathcal{S}\rvert$	$2^{2N+NM} N^M 3^M$	1.5×10^{16}	1.7×10^{43}
Action space $\lvert\mathcal{A}\rvert$	$N\,M$	24	80
Q-table size $\lvert\mathcal{S}\rvert \cdot \lvert\mathcal{A}\rvert$	—	3.6×10^{17}	1.4×10^{45}

4 Implementation of Gym Environment

I implement the elevator dispatch in the Gym Env [Tow+24]. There are 3 types of event corresponding to a step: passenger spawning, door opening and door closing. The core logic is shown in algorithm 1[1].

Algorithm 1 Event-Driven Elevator Group Simulator

```
procedure AdvanceUntilNextEvent
    t_spawn ← NextSpawnTime()
    t_car ← NextCarEventTime()
    Δt, evt ← EarlierOf(t_spawn, t_car)
    AdvanceClockAndCars(Δt)
    if evt = SPAWN then
        SpawnPassengers()
    end if
    if evt = DOOR_OPEN then
        HandleArrival(CurrentCar)
    end if
    if evt = DOOR_CLOSE then
        FinalizeDoorClose(CurrentCar)
    end if
    return SnapshotReward(), evt
end procedure
```

5 Reinforcement Models

I use the policy Gradient with baseline and 1-step Actor Critic from Lab 9 of the course. Also, I use the PPO from 'Stable Baseline 3'.

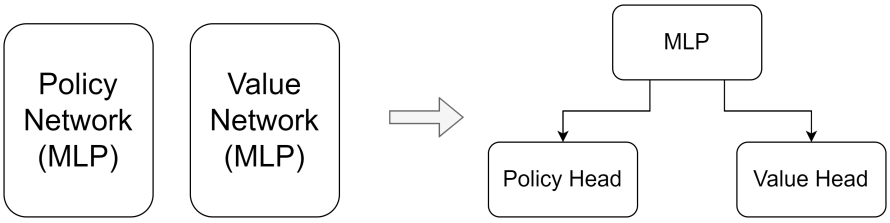


Figure 1. Archtecture of network

The difference is that I change the original network architecture to one MLP backbone plus policy head and value head. 1 This change will make the training more stable.

6 Experiments

The important part in this project is to design a reasonable reward to make the model learn and converge fast. What I have designed is very heuristic, which is also part of the reason why RL models perform poorly.

6.1 Waiting time as reward

At the end of each step, the environment will return a reward. The reward is relevant to the waiting time of all the passengers in the hall, the riding time of all the passengers in the cars.

Passenger clocks.

$$\forall \text{passenger p} : \; w_p(t) = t - \tau_p^{\text{req}} \quad r_p(t) = t - \tau_p^{\text{board}}$$

Instantaneous cost.

$$C(t) = \sum_{p \in \mathcal{W}(t)} \big[w_p(t)\big]^2 + \sum_{p \in \mathcal{B}(t)} \big[r_p(t)\big]^2$$

Simulator reward (per event).

$$R(t) \; = \; -\,C(t)$$

6.1.1 Motivation

Both FIFO and LOOK have delay problem. I design this reward, hoping that RL model can solve the problem. If the model aims to get higher rewards, it has to find a way to reduce the waiting time of the passengers. I square each passenger's waiting time, meaning there will be penalty if passengers wait for too long time.

6.1.2 RL models find a shortcut

The following figure illustrates the events triggering situation throughout the 100-steps time. A yellow line at a timestamp means that the event is triggered at this timestamp.

Rewards	1-step AC	FIFO	LOOK
3 halls 1 car	-4.1e6	-8.4e4	-1.06e7

Table 5. Performance of AC model and baselines on the same setting (seed=0)

If we check how many passengers successfully alight on the desired floors, it shows that the number of RL model is zero. Then, I fix the random factors in the elevator environment to always have the same passengers spawning for testing. 2

In my environment, one step means one event triggered. I train the models with 100 steps. What the models do is filling the 100 steps with OPEN and CLOSE events (in the figure there seems to be only CLOSE event, this is because CLOSE events overlap with OPEN events) and quickly finishing this round. The total round consumes less time, so the waiting time is also low. That is the shortcut that RL models learn to maximize rewards. 3

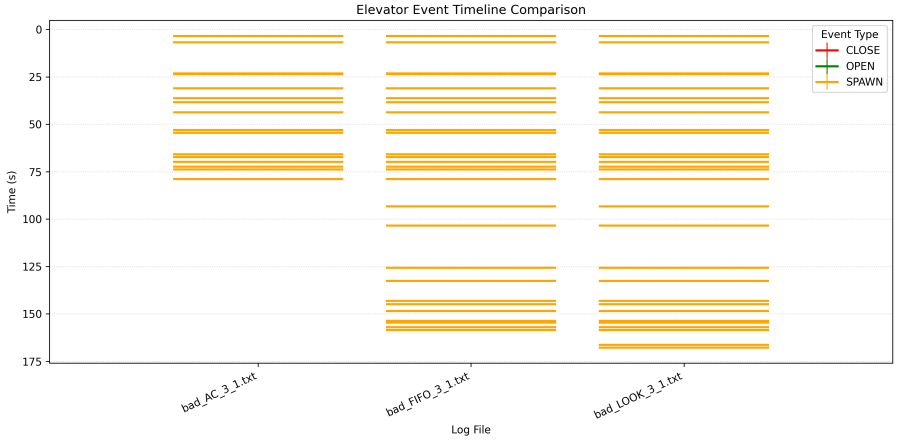


Figure 2. Passenger Spawning in 3 halls 1 car setting

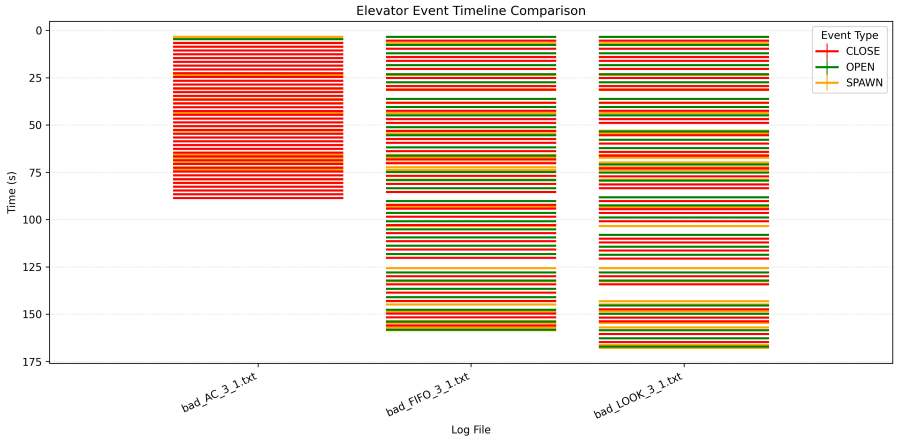


Figure 3. Shortcut case

6.2 Number of passenger boarding and alighting as reward

In every step, we count the number of passengers alighting and boarding. Then, we scale the number, add them together as a reward. After computing the reward, we set the numbers back to 0.

$$R_t \;=\; 10\,N_t^{\text{alight}} \;+\; N_t^{\text{board}}, \qquad N_t^{\text{alight}} \leftarrow 0,\; N_t^{\text{board}} \leftarrow 0.$$

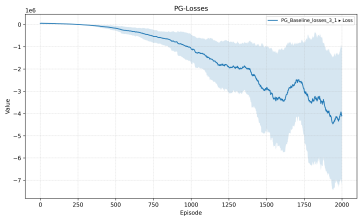
6.2.1 Motivation

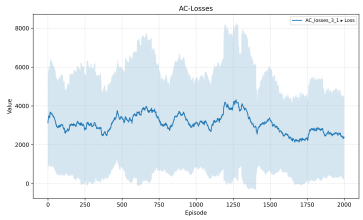
Since waiting time as reward does not make the models send any passengers to their desired floors, I directly use the number of passengers boarding and alighting as reward. I aim to let the models send as many passengers as possible to the right floors in a given time.

6.2.2 Result

I compare the performance of policy gradient with baseline, 1 step Actor Critic and PPO.

In the 3 halls 1 car setting, Policy Gradient with baseline and PPO are learning during training. Their losses are decreasing as the number of episodes grows. But 1 step Actor Critic does not seem to be learning based on its fluctuating losses. 6





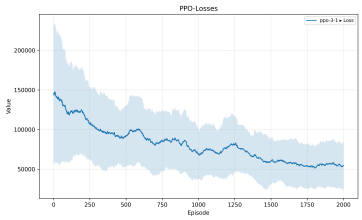
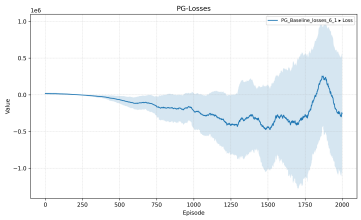


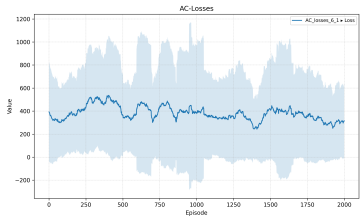
Figure 4. Policy Gradient with Baseline Losses in 3 halls 1 car setting

Figure 5. 1 step Actor Critic Losses in 3 halls 1 car setting

Figure 6. Proximal policy optimization Losses in 3 halls 1 car setting

In the 6 halls 1 car setting, Policy Gradient with baseline seems not able to learn based on its fluctuating losses. 9





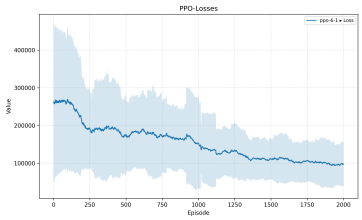


Figure 7. Policy Gradient with Baseline Losses in 6 halls 1 car setting

Figure 8. 1 step Actor Critic Losses in 6 halls 1 car setting

Figure 9. Proximal policy optimization Losses in 6 halls 1 car setting

In Policy Gradient with baseline and 1 step Actor Critic, we train the model in each episode until the clock time is larger than 100 seconds. In PPO, we use model from library ‘Stable Baseline 3‘ and we train the model in each episode exactly 100 steps.

Note that 100 steps and 100 seconds clock time are different. For example from the following figures, the elevator system can only have around 70 steps in 100 seconds. 11

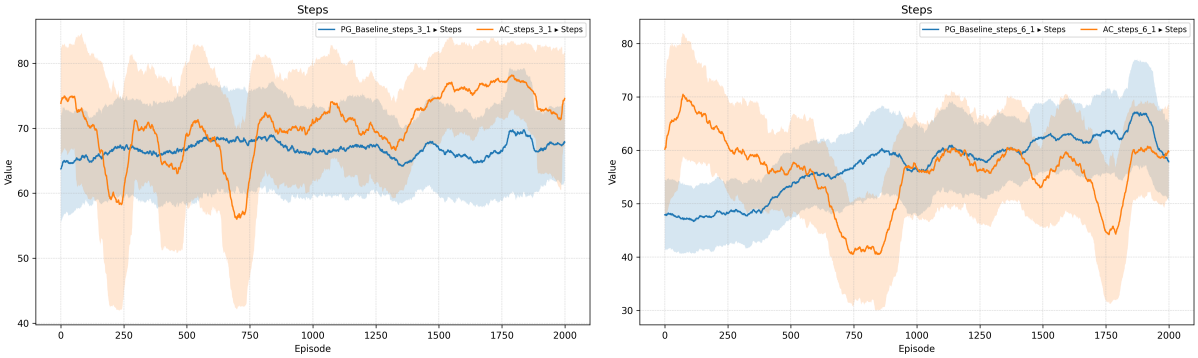


Figure 10. The number of steps per episode in 3 halls 1 car setting

Figure 11. The number of steps per episode in 6 halls 1 car setting

Because of the different steps between PG, AC and PPO (PG, AC: approx. 70 steps, PPO: 100 steps), the rewards they get are also different. PPO gets more rewards because it has more steps to send more passengers to the desired floors. 13

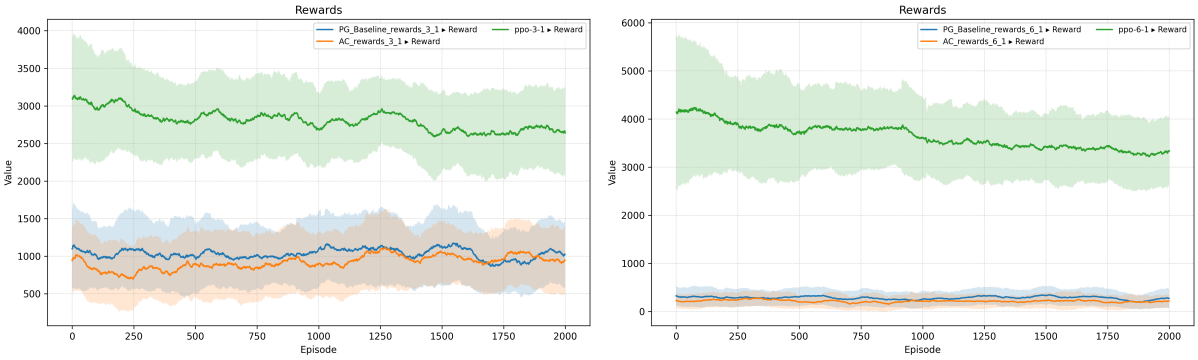


Figure 12. Rewards of models in 3 halls 1 car setting

Figure 13. Rewards of models in 6 halls 1 car setting

In the analysis of performance, I run each model and algorithm 1000 times to get the rewards distribution. The average performance of each models is not better than baseline FIFO, LOOK. But, in some cases, they can reach very high scores and beat the baseline. 15

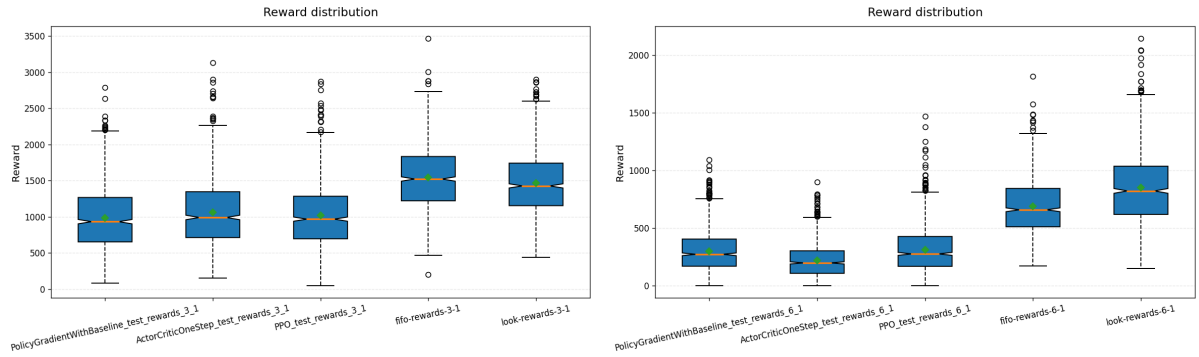


Figure 14. Average Performance of models and FIFO, LOOK in 3 halls 1 car setting

Figure 15. Average Performance of models and FIFO, LOOK in 6 halls 1 car setting

6.2.3 Cases where RL models beat baseline

I use the same seed to produce the same passenger spawning time and spawning number. The following figure illustrates the events triggering situation throughout the 100-seconds time. A yellow line at a timestamp means that the event is triggered at this timestamp. We can see that the timestamps of SPAWN event triggered are exactly the same in all the models and baselines. I want to make the elevator environment deterministic to guarantee the fairness of the test. 19

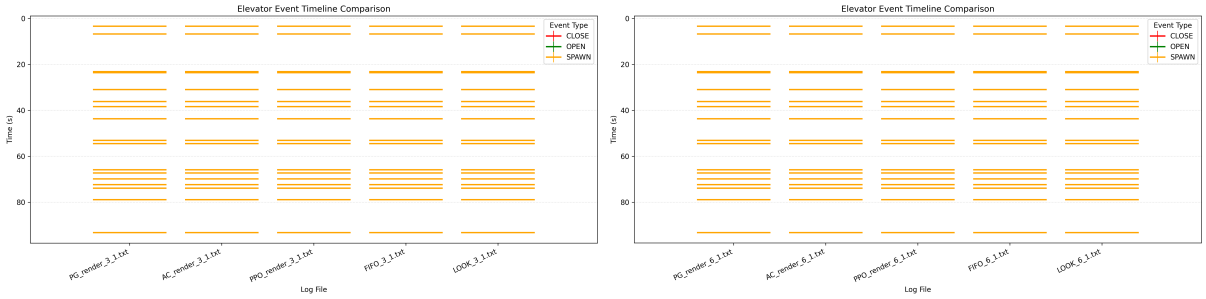


Figure 16. The Passenger Spawning in 3 halls 1 car setting

Figure 17. The Passenger Spawning in 6 halls 1 car setting

Based on that spawning setting, I test all the models as well as the baselines to see which can get higher rewards. From the table 6, PPO outperforms others in this very case.

Rewards	PG with baseline	1-step Actor Critic	PPO	FIFO	LOOK
3 halls 1 car	1056.0	1197.0	1222.0	1218.0	882.0
6 halls 1 car	283.0	30.0	846.0	641.0	760.0

Table 6. Rewards of models and baselines on the same setting (seed=0)

There is an interesting observation in the event timeline comparison in 6 halls 1 car setting 19. The baselines' events are sparse while the models' events are dense. This is because the baselines have already finished sending all the passengers while the models are still in process.

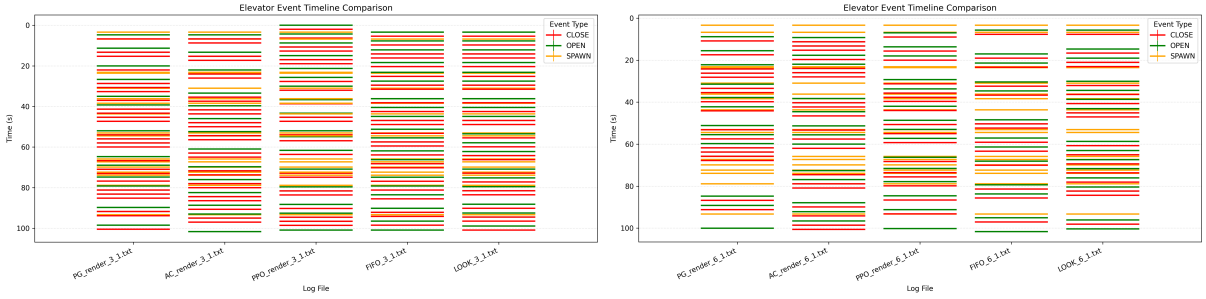


Figure 18. Event Timeline Comparison in 3 halls 1 car setting

Figure 19. Event Timeline Comparison in 6 halls 1 car setting

We can observe PPO's behaviors. Here we choose OPEN event, record the loads of elevator in this event and show them in the figure.

From the figure below 21, we can see that PPO always has the highest loads of elevator. This is because PPO always lets the number of passengers grow and then send them in one time.

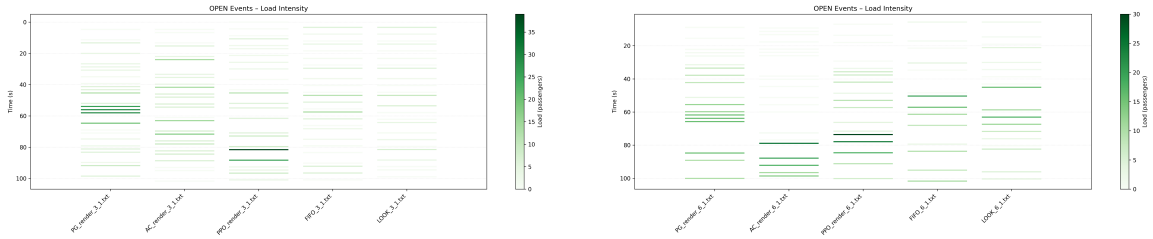


Figure 20. Elevator Load Comparison in 3 halls 1 car setting

Figure 21. Elevator Load Comparison in 6 halls 1 car setting

7 Discussion

7.1 How we can avoid designing reward in modern RL

Writing a numeric reward often turns out to be the hardest part of an RL project: misspecification leads to reward-hacking, unsafe behaviour, and long engineering cycles. Avoiding explicit rewards therefore means

1. faster iteration,
2. safer alignment with human intent,
3. the ability to exploit large, unlabelled data sets

Recently, the popular ways to avoid designing reward are:

1. Reinforcement Learning from Human Feedback (RLHF) – learning a reward model from pairwise human preferences and optimising the policy against it, the technique that now underpins large-language-model alignment and many robotics demos.
2. Curiosity-driven intrinsic motivation – giving the agent its own internal "curiosity" reward (e.g., prediction-error bonuses or Random Network Distillation) so it can explore when the environment offers no external reward.
3. Inverse Reinforcement Learning (IRL) – inferring the hidden reward function that makes expert demonstrations near-optimal, then using that learned reward to train new policies.
4. World-model pre-training (e.g., Dreamer V3) – first fit a latent dynamics model entirely without task rewards, then plan or fine-tune inside that model; has become a go-to recipe for robotics and embodied-AI benchmarks.

8 Conclusion

From this project, I have learned how to design a gym environment for Reinforcement Learning and how to analyze data in different perspectives. While training, the RL models will have some emergent behaviors. Some are very interesting and worthy for illustrating.

I also get a deeper insight into Policy Gradient, Actor Critic and PPO models.

The most tricky part is to design the reward properly. Now in modern RL, we also have some ways to avoid designing reward.

References

[Dan25] W. Dan. *elevator-gym. An RL gym environment for the elevator-dispatch problem*. Version latest. If you use this software, please cite it. Apr. 27, 2025. URL: `https://github.com/william-dan/rl-elevator`.

[Tow+24] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al. "Gymnasium: A Standard Interface for Reinforcement Learning Environments". In: *arXiv preprint arXiv:2407.17032* (2024).

[Wei+20] Q. Wei, L. Wang, Y. Liu, and M. M. Polycarpou. "Optimal Elevator Group Control via Deep Asynchronous Actor-Critic Learning". In: *IEEE Trans. Neural Networks Learn. Syst.* 31.12 (2020), pp. 5245–5256. DOI: 10.1109/TNNLS.2020.2965208.