Deep Reinforcement Learning Based Emergency Department Optimization

KANG, Xingjian
Medical Engineering (Std ID. 23079135)
Friedrich-Alexander University
Erlangen, 91052
xingjian.kang@fau.de

***Abstract*—This paper illustrated the research project of improving the medical resource distribution in Emergency Department by combining the Deep Reinforcement Learning (Deep RL) network with Discrete-Event-Simulation (DES) tools. Through agents trained by several Deep RL algorithms, a dynamic medical resource management mechanism is designed and tested to reduce the average waiting time and length of stay in ED for patients.**

***Index Terms*—healthcare management, emergency department; reinforcement learning; deep learning; simulation**

I. Introduction

The Emergency Department (ED) is a very essential unit of modern hospital system. Unfortunately, ED has often a bad reputation for its overcrowding and the long queuing time [1]. Moreover, patients always suffer from the prolonged Length of Stay (LOS) time in the ward, which not only challenges the capacity of medical team, but also may leads to harmful effects to patients' safety [2].

To model the real scenarios in ED, the Discrete-Event-Simulation (DES) could be used since it could successfully model a complex system by simulating the occurrence of discrete events over time [3]. DES has already been used by several optimization problems in health systems such as improvement of workflow in radiation therapy modelling [4] and hospital ward transfer simulation [5]. In this project, we developed a Python DES framework in ***SimPy***, then a simplified patient flow and medical resource assignment system simulation through 12 hours in ED can be well performed.

TABLE I
CTAS Levels and distributions

Acuity Level	*Label*	*Ratio*
I	Resuscitation	10%
II	Emergent	30%
III	Urgent	40%
IV	Less Urgent	10%
V	Not Urgent	10%

Aims at enhancing the medical resources assignment in the Emergency Department (ED), the utilization of networks based on Deep Reinforcement Learning (Deep RL) is employed. As shown in Fig 1, in the realm of Deep RL, an *agent* interacts with its *environment*, driven by the goal of devising a *policy* that optimizes the accumulation of long-term *rewards* [6]. For better training and validation, a custom environment based on the state-of-art reinforcement learning API ***OpenAI Gymnasium*** is developed. It owns the compatibility of interacting with previous implemented DES structure. In order to obtain a robust schedule system for the doctors and nurses in ED, three different variants of Deep Q-Network are trained and their performances was compared.

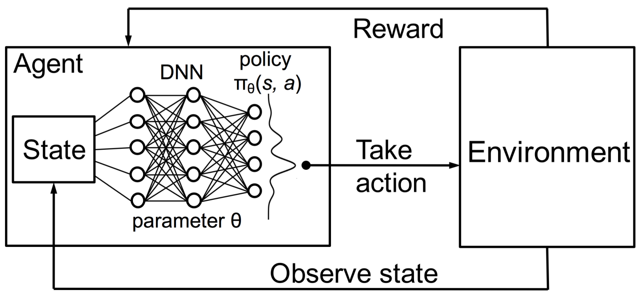


Fig. 1. Deep Reinforcement Learning

II. Methodology

A. Discrete-Event-Simulation of ED Patient Flow

The entire simulation for patient flow in ED requires a careful analysis of the real system. It involves not only the generation or arrival pattern of patients, but the capacity of medical resources as well. Before we build up the model, a certain set of time parameters must be determined for a clear representation of time series.

- **Patient Flow Analysis**

  Patient Flow is a concept used in healthcare services to denote the process by which patients are directed through a healthcare facility [7], [8]. A typical model of simplified patient flow in ED is shown in Fig 2. A single patient comes to the ED by following an exponential time distribution. The usual path a patient in ED takes begins with the triage stage, where an arriving patient is assessed by a triage doctor. Following the triage process, the patient is allocated a severity level based on the Canadian Triage and Acuity Scale (CTAS) [9], reflecting the seriousness of their signs and symptoms. With different acuity levels as Table I, patients might have different treatments. For example, severe patients with acuity level 1 will be immediately transferred to ICU, while patients with less severity may go through a relative longer treatment process before being discharged. After finishing all the treatments, the patient will be discharged and the whole flow will then come to an end.

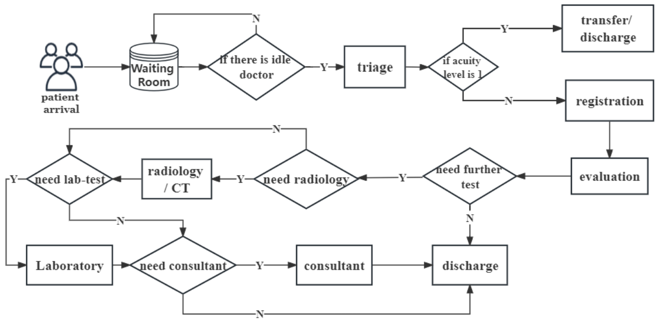


Fig. 2. Simplified ED Patient Flow

- **Delay Time Distribution**

Before starting the simulation, the different time distributions of multiple service events' delays should be selected. They are determined from historical records data and expert reviews [10]. Table II shows the chosen time distributions of the discrete events in virtual emergency room.

TABLE II
EVENTS' DELAY TIME DISTRIBUTIONS

Treatment/Event	***Time Distribution(min)***
Patient Arrival	Poisson(7)
Triage	Exponential(7)
Transfer	Exponential(5)
Registration	Exponential(5.5)
Evaluation	Normal($\mu = 14$, $\sigma = 6$)
CT-Scan	Normal($\mu = 29$, $\sigma = 14$)
Laboratory	Normal($\mu = 35$, $\sigma = 15$)
Consultation	Normal($\mu = 15$, $\sigma = 8$)
Discharge	Exponential(3)

- **Simulation Model**

Our simulation of the ED is mainly about a queuing system that follows the *First In First Out(FIFO)* rule. This system contains two parts:

*class **patient():***
The basic information of a single patient is encoded by this class, e.g. waiting time and LOS, etc. The patient is generated with certain acuity level and the probability ratios of certain level are showed in Table I.

*class **EmergencyDepartment():***
In this class, we implement the real time simulation of ED. Method *patient_generation()* contains the process of patient flow in ED. The patient arrival time can be modeled by *SimPy.timeout(delay)*. After the patient comes to the ED, We consider our medical resources have four types: doctors, nurses, CT-technicians and laboratory doctors. They are all encoded as *SimPy.Resource()* classes, so the management by a limited number of processes at a time can be archived [11]. If there's no more idle doctors for triage, the patient will be pushed into the waiting room until he/she is popped to the triage room. Then, the waiting time can be storage. The process of ED workflow is simulated by *patient_flow()*. With certain delay, the time consumption of treatments can be well visualized, which enables us to get a good search for the bottleneck of ED working process.

B. Deep Reinforcement Learning Agent Design

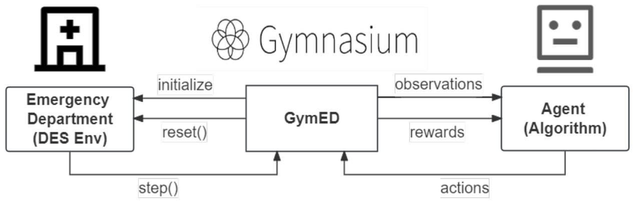


Fig. 3. Gymnasium ED Environment

In purpose of better scheduling of free medical resource in ED, an environment (*GymED*) of *OpenAI Gymnasium* [12] is developed as Fig 3. The essential tasks to be completed encompass designing the state representation, formulating the reward structure, and defining the available actions. This allows an agent to train in a well known Deep RL algorithm, i.e. Deep Q Network. Finally, for comparing and improving the performances of decision making mechanisms, two other modified DQN agents are employed.

- **Gymnasium ED(GymED) Environment**

To model and optimizing the medical resource management, we initially set 1 doctors and 1 nurses at working state. We have totally 20 idle doctors and 20 idle nurses to assign. To simplify the simulation, we consider the numbers of working CT-Technician and laboratory doctors are fixed, which are 2 and 5. Each time step, the GymED takes the change of doctors and doctors, then runs a simulation for a fixed duration. The simulation time of a single roll-out is 720 minutes

(12 hours) for getting closer to the real ED shift schedule.

State Formulation: In *OpenAI Gymnasium* environment, the *observation_space* with *space.Box(shape=(2,))* can represent a 2D state space. The state at time point t s_t is defined as a matrix in the shape of 2. To describe the state of busy medical resources, the state vector encodes the number of current working doctors and nurses. It can be expressed as (1):

$$s_t = \{|\mathcal{MR}^1|_t, |\mathcal{MR}^2|_t\} \tag{1}$$

Where:

$$|\mathcal{MR}^1|_t : \text{number of working doctors,}$$
$$|\mathcal{MR}^2|_t : \text{number of working nurses}$$

Action Formulation: The action a_t of our agent can take are expressed in the Table III. We use *space.Discrete(5)* to represent our *action_space* since it can generate number in the range of [0, 5) which perfectly fits our action design.

TABLE III
ACTION LIST

Action Number	*Effect*
1	increase 1 working doctor
2	reduce 1 working doctor
3	increase 1 working nurse
4	decrease 1 working nurse
0	do not change the staffing

Reward Design: In terms of the model's objective, we aims to minimize the waiting time as well as the LOS of patients. As a result, the reward r_t at a given time step should first involve the negative sum of weighted waiting time and LOS (Table IV).

TABLE IV
WEIGHT OF WAITING TIME & LOS

Acuity Level (a)	*Waiting Time* c_1	**Length of Stay** c_2
I	0.5	0.5
II	0.4	0.4
III	0.3	0.3
IV	0.2	0.2
V	0.1	0.1

Additionally, we would like our schedule system can give treatments to as many patients as possible. Hence, the ratio of treated/discharged patients of total patients can also be a part of reward $r_{patient}$. Last but not least, for better optimizing the staffing, the number remaining free doctors and nurses should also play a role in the reward as $r_{resource}$.

The whole reward at time step t is showed in (2):

$$r_t = -(\sum_{\mathrm{p_i}} \mathrm{c}_1^{\mathrm{a}} WT_{\mathrm{p_i}} + \sum_{\mathrm{p_i}} \mathrm{c}_2^{\mathrm{a}} LOS_{\mathrm{p_i}}) + 0.1 \times (r_{patient} + r_{resource}) \tag{2}$$

Where:

$\mathrm{p_i}$: single patient

$\mathrm{c}_1^{\mathrm{a}}$,$\mathrm{c}_2^{\mathrm{a}}$: weights of the waiting time and LOS as Table 4

- **Deep Q Network(DQN)**

Within this segment, we introduce the Deep Q-network (DQN) within the context of the Markov Decision Process (MDP) framework. Combined with deep neural network, DQN is introduced as a way to address the challenges of scaling up Q-learning, a classical RL algorithm, to handle complex and high-dimensional state spaces [13]. The main innovation of DQN is the use of deep neural networks to estimate the Q-values $Q(s_t, a_t)$, which denotes the anticipated total reward achievable through selecting a particular action within a specific state and subsequently adhering to a designated policy. Algorithm 1 shows the overall DQN algorithm in ED environment.

We use a Temporal Difference Error term (TD Error δ_t) to express the optimization goal of DQN, which is the difference between the result from target network and prediction. The object function (3) to minimize is as follow:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\overbrace{\left(\underbrace{r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)}_{\text{Target Q Value } \mathcal{T}} - \underbrace{Q(s_t, a_t; \theta)}_{\text{Prediction } \mathcal{P}}\right)}^{\text{TD Error } \delta_t}\right]^2 \tag{3}$$

Where:

$$\theta \text{ are the network weights,}$$
$$\theta^- \text{ are the target network weights}$$
$$\gamma \text{ is the discount factor}$$

The deep neural network structure we used is showed as Fig 4. A simple feed forward network with 3 hidden layers (32, 64 and 2 neurons for each layer) can perform good search for value-function approximation.

There are also some especial structures and tricks that can be employed to improve the estimation of DQN:

Replay Buffer: We use a buffer $\mathcal{D}$ to store past experiences for training [14]. It randomly samples transitions to break temporal correlations, stabilizing learning, enabling better exploration, and allowing the

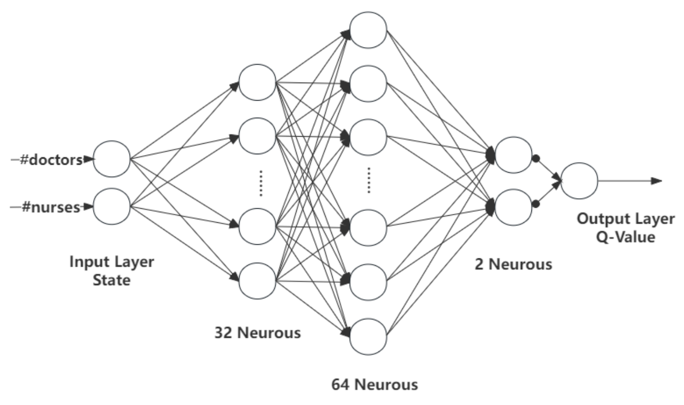


Fig. 4. Network Architecture of Agent

agent to learn from a diverse range of situations. This helps improve convergence and overall training efficiency.

Target Network: A target network [15] is a duplicate network used for estimating future Q-values. It can also stabilizes training by providing a fixed Q-value target for comparison, reducing the issue of "moving targets."

Epsilon(ϵ) Greedy and Decay: Epsilon-greedy (4) is a common exploration strategy in reinforcement learning. It balances exploration and exploitation by choosing the best action (greedy) most of the time, but occasionally selecting a random action (exploration) with a probability ϵ. This gradual transition can also improve learning efficiency and help achieve policy convergence.

$$a_t = \begin{cases} \arg\max_a Q(s_t, a; \theta), & probability > \epsilon, \\ random\ action, & probability \leq \epsilon, \end{cases} \tag{4}$$

Moreover, We gradually decrease the exploration parameter ϵ over time with a parameter γ. The agent shifts from early exploration to refined exploitation, striking a balance between exploring new actions and exploiting the best-known actions.

- **DQN Extensions**

Variants of DQN encompass refinements and adaptations to the fundamental DQN algorithm (also *Vanilla DQN*), aimed at tackling diverse challenges and elevating performance. Here we implemented two of them:

Double DQN (DDQN): DDQN reduces the overestimation of Q-values by using a separate network for selecting the best action and evaluating its value [16].It improves the stability of training by slightly changing the target Q value.

Algorithm 1 Deep Q-Network with Replay Buffer

```
1: Initialize replay buffer $\mathcal{D}$ with capacity $N$
2: Initialize Q-network with random weights $\theta$
3: Initialize target Q-network with weights $\theta' = \theta$
4: for $episode = 1$ to $M$ do
5:     Reset the ED environment to initial
6:     for $t = 1$ to $T$ do
7:         $\epsilon$ greedy search of action $a_t$
8:         if $\mathcal{MR}^1$ and $\mathcal{MR}^2$ is not zero then
9:             Execute action $a_t$, change the idle resource
10:        end if
11:        Run DES simulation
12:        Observe reward $r_t$ and next state $s_{t+1}$
13:        Put transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{D}$
14:        if $\mathcal{D}$ contains more than mini-batch size $B$ transitions then
15:            Sample $B$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $\mathcal{D}$
16:            for each transition $(s_i, a_i, r_i, s_{i+1})$ do
17:                Compute target value:
                   $y_i = r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \theta')$
18:                Predict Q-value:
                   $Q_i = Q(s_i, a_i; \theta)$
19:                Gradient of loss function:
                   $\nabla_\theta \mathcal{L}_i = 2(Q_i - y_i)\nabla_\theta Q(s_i, a_i; \theta)$
20:                Update Q-network weights:
                   $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_i$
21:            end for
22:            Every $C$ steps, update target Q-network: $\theta' \leftarrow \theta$
23:        end if
24:        $s_t \leftarrow s_{t+1}$
25:    end for
26: end for
```

In the case of Double DQN, the Q-value estimates are obtained from another online Q-network, and the target Q-values are estimated using the target Q-network to mitigate overestimation bias.

Mathematically, we modify the target Q value by (5):

$$\mathcal{T} = r_t + \gamma Q\left(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'; \theta); \theta^-\right) \tag{5}$$

Dueling DQN: Dueling DQN separates the Q-network into two parts: one estimating the value function(*Policy Network*) and another estimating the advantage function(*Target Network*) [17].

We extend the target Q value by (6):

$$\mathcal{T} = r_t + \gamma \left(V(s_{t+1}; \theta_v) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s_{t+1}, a'; \theta_a) \right) \tag{6}$$

Where:

$$V(s_{t+1};\theta_v)\text{: value estimation,}$$
$$A(s_{t+1},a';\theta_a)\text{: advantage estimation,}$$

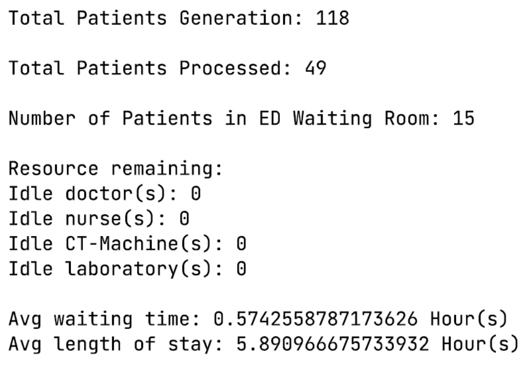
This allows the agent to learn state values and action advantages separately, enhancing learning efficiency.

III. Results

This section presents the results of ED medical resource system designed by RL algorithms. For acceleration of training with GPU, the DQN agent was coded in *PyTorch* framework. In the context of the experiment, we compared the performance of three distinct DQN agents.

A. Output of Discrete Event Simulator

In order to test if the DES could successfully simulate the patient flow of ED, we set up SimPy-based environment by certain initialization. After running the SimPy-based environment for 720 min, we had the following output in Fig 5:



```
Total Patients Generation: 118

Total Patients Processed: 49

Number of Patients in ED Waiting Room: 15

Resource remaining:
Idle doctor(s): 0
Idle nurse(s): 0
Idle CT-Machine(s): 0
Idle laboratory(s): 0

Avg waiting time: 0.5742558787173626 Hour(s)
Avg length of stay: 5.890966675733932 Hour(s)
```

Fig. 5. Output of SimPy ED Environment

B. Hyperparameters

Precisely defining hyper-parameters holds a great significance, as they have vital influence over the performance of the deep Q-network agents. The values in Table V were obtained through a random search process, yielding appreciable enhancements in performance metrics.

TABLE V
Hyperparameters

Hyperparameters	***Value***
Learning Rate	1×10^{-3}
Random Seed	258
Optimizer	Adam
Batch Size	100
Time steps	2×10^{4}
Discount Factor	0.99
Update Frequency	10
Replay Buffer Size	1×10^{3}
Epsilon Start	1
Epsilon End	0.01
Decay Factor	0.9999

C. Performance of Agents

- ***Vanilla DQN:*** The output of the trained conventional DQN algorithm is shown in Fig 6. Evidently, the performance of the exemplified network demonstrates improvement over successive time steps (model runs). Approximately the maximum reward that can be achieved is at the level of 600. However, As the beginning of the curve shows, our agent didn't have many negative rewards. This suggests that the DQN agent tends to exhibit a tendency of overestimating the Q-value.

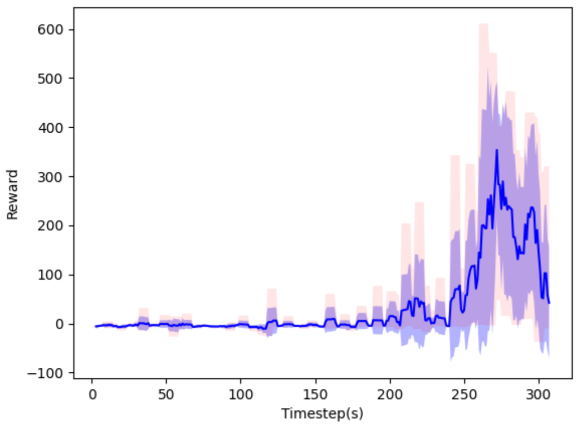


Fig. 6. Reward Plot of DQN

- ***Double DQN:*** Fig 7 demonstrates the performance of Double DQN agents. Compared to Vanilla DQN, DDQN agent has more negative Q-values at the early stage of training. This proves that the Double DQN can address the overestimation problem in a considerable level. Besides, the highest reward is significantly improved.

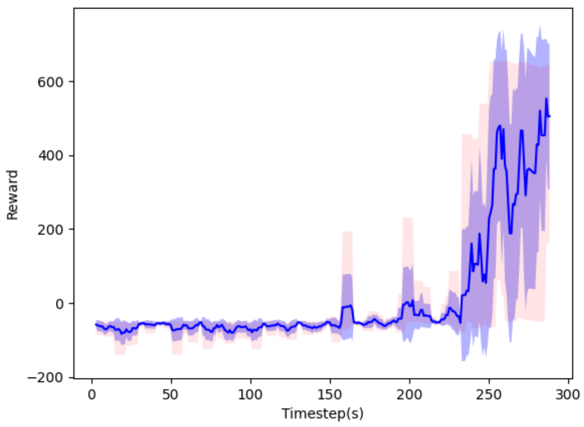


Fig. 7. Reward Plot of DDQN

- ***Dueling DQN:*** As demonstrated in Fig 8, agent with Dueling DQN has a higher maximum return than the former two algorithms. Part of the reason for this is that Dueling DQN learns state value functions more efficiently. At each iteration, the function is updated, which also affects the values of other actions. Traditional DQNs only update the value of one action, but not the values of other actions. Therefore, Dueling DQN is able to learn the state value function more frequently and accurately.

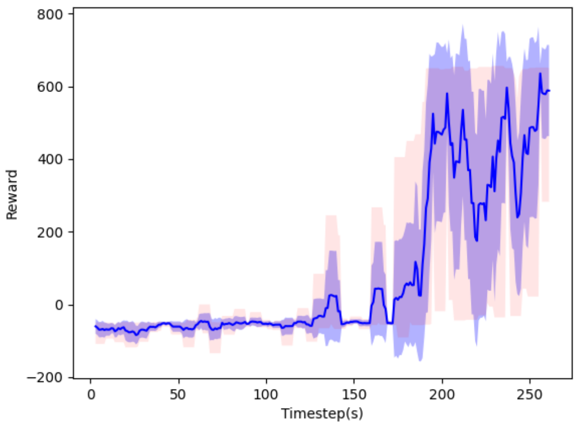


Fig. 8. Reward Plot of Dueling DQN

IV. Discussion

The scope of this study is restricted due to the assumptions necessitated by the absence of comprehensive medical resource data. Despite the adaptability of medical resource numbers within the time window, the required information was absent from existing literature. In the future, with more detailed data from real scenarios, a more precise simulation of patient flow and ED working process could be extended.

The fusion of a Deep Reinforcement Learning Network with Discrete-Event Simulation holds substantial promise for addressing the medical resource dispatching problem [10]. In this paper, our focus is not on presenting a finely tuned Deep RL model or an intricate emergency department simulation. Instead, we aim to introduce a DES framework that aligns with OpenAI Gymnasium environments. This framework allows for the straightforward application of a multiple methods that have been devised and assessed within such environments.

A potential extension of our optimization efforts could involve exploring alternative Deep RL algorithms that our agent can employ. For example, we can apply more DQN variants, e.g. *Noisy DQN* [18], *Rainbow* [19], *etc.* Moreover, we may change the decision making mechanism by modify the framework into a *Multi-Agent RL* environment incorporating other factors, such as operating rooms and bed situation [20].

References

[1] G. Savioli, I. F. Ceresa, N. Gri, G. Bavestrello Piccini, Y. Longhitano, C. Zanza, A. Piccioni, C. Esposito, G. Ricevuti, and M. A. Bressan, “Emergency department overcrowding: understanding the factors to find corresponding solutions,” *Journal of personalized medicine*, vol. 12, no. 2, p. 279, 2022. I

[2] R. Otto, S. Blaschke, W. Schirrmeister, S. Drynda, F. Walcher, and F. Greiner, “Length of stay as quality indicator in emergency departments: analysis of determinants in the german emergency department data registry (aktin registry),” *Internal and Emergency Medicine*, vol. 17, no. 4, pp. 1199–1209, 2022. I

[3] J. I. Vázquez-Serrano, R. E. Peimbert-García, and L. E. Cárdenas-Barrón, “Discrete-event simulation modeling in healthcare: A comprehensive review,” *International journal of environmental research and public health*, vol. 18, no. 22, p. 12262, 2021. I

[4] B. Vieira, D. Demirtas, J. B van de Kamer, E. W. Hans, and W. Van Harten, “Improving workflow control in radiotherapy using discrete-event simulation,” *BMC medical informatics and decision making*, vol. 19, pp. 1–13, 2019. I

[5] M. Allen and T. Monks, “Integrating deep reinforcement learning networks with health system simulations,” *arXiv preprint arXiv:2008.07434*, 2020. I

[6] Y. Li, “Deep reinforcement learning: An overview,” *arXiv preprint arXiv:1701.07274*, 2017. I

[7] R. Hall, “Patient flow,” *AMC*, vol. 10, no. 12, p. 4, 2013. II-A

[8] R. El-Bouri, T. Taylor, A. Youssef, T. Zhu, and D. A. Clifton, “Machine learning in patient flow: a review,” *Progress in Biomedical Engineering*, vol. 3, no. 2, p. 022002, 2021. II-A

[9] M. J. Bullard, E. Musgrave, D. Warren, B. Unger, T. Skeldon, R. Grierson, E. van der Linde, and J. Swain, “Revisions to the canadian emergency department triage and acuity scale (ctas) guidelines 2016,” *Canadian Journal of Emergency Medicine*, vol. 19, no. S2, pp. S18–S27, 2017. II-A

[10] S. Lee and Y. H. Lee, “Improving emergency department efficiency by patient scheduling using deep reinforcement learning,” in *Healthcare*, vol. 8, p. 77, MDPI, 2020. II-A, IV

[11] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, “Sympy: symbolic computing in python,” *PeerJ Computer Science*, vol. 3, p. e103, Jan. 2017. II-A

[12] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, “Gymnasium,” Mar. 2023. II-B

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015. II-B

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013. II-B

[15] J. Fan, Z. Wang, Y. Xie, and Z. Yang, “A theoretical analysis of deep q-learning,” in *Learning for dynamics and control*, pp. 486–489, PMLR, 2020. II-B

[16] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016. II-B

[17] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, “Dueling network architectures for deep reinforcement learning,” in *International conference on machine learning*, pp. 1995–2003, PMLR, 2016. II-B

[18] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, *et al.*, “Noisy networks for exploration,” *arXiv preprint arXiv:1706.10295*, 2017. IV

[19] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018. IV

[20] S. V. Albrecht, F. Christianos, and L. Schäfer, *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2023. IV