Machine Learning Based Modeling of Terahertz Aerial Communications for 6G Systems

Kemal Berkay Elçik kemalberkayelcik@hotmail.com
Electrical and Electronics Engineering / Boğaziçi University, 2019

Özgür Gürbüz
Electronics Engineering / Sabancı University

Akhtar Saeed
Electronics Engineering / Sabancı University

Access the github repository

Abstract

At most 100-word summary of the problem and the findings.
Keywords: At most 5 keywords.

1 Introduction

The THz band, a trillion cycles per second, is the frequency range between 0.1 to 10 THz. This band offers higher data rates compared to the currently used radio frequencies. Because of the high data rate demands, 6G and beyond technologies can gain much from the THz band.

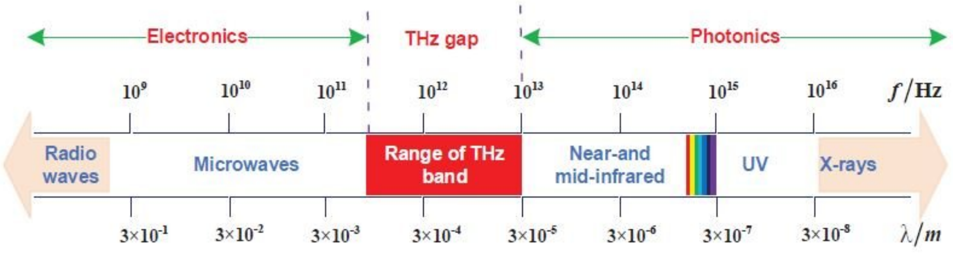

Figure 1: THz band in the EM spectrum.
[Huq et al., 2019]

However, the huge bandwidth advantage of the THz band also comes with its downsides. The high frequency of the THz band causes high path loss and noise at the ground level. The path loss is made up of two components as seen in Eq. 1.

$$A_{pathloss} = A_{spread} \cdot A_{absorption} \tag{1}$$

One is the spread loss, also called the free space path loss, caused by the attenuation of the electromagnetic wave as it propagates in the medium. It can be formulated as in Eq. 2.

$$A_{spread} = \frac{4 \cdot \pi \cdot f \cdot d}{c} \tag{2}$$

The other one is the absorption loss, caused by absorption of the electromagnetic radiation by the water vapor in the atmosphere. Spread loss is not related to the atmospheric conditions, it is only related to the distance the electromagnetic wave travels. However, since absorption loss is caused by the water vapor in the atmosphere, it is affected by water vapor concentration. The effect of the absorption loss is mitigated at higher altitudes, due to low water vapor concentration. Therefore, we aim to use the THz band for aerial communication. The model for the absorption loss is given in Eq. 3.

$$A_{absorption}(f,d) = \frac{1}{\tau(f,d)} \tag{3}$$

$$\tau(f,d) = \frac{P_0}{P_i} \tag{4}$$

τ is the medium's transmittance value. f and d denote the frequency and distance traveled by the electromagnetic wave respectively, P_0 is the power of the incident electromagnetic wave and P_1 is the power of the electromagnetic wave after traveling the distance d. Therefore, the transmittance corresponds to the fraction of the electromagnetic wave's power that remains after passing through the medium for distance d, where the electromagnetic wave is of frequency f.

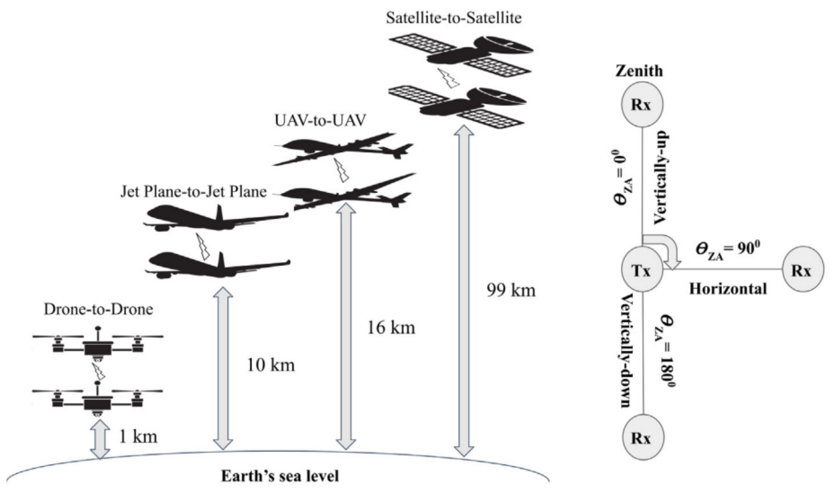

Figure 2: Aerial vehicle communication cases that can be considered by the model. [Saeed et al., 2020]

Efficient allocation of resources is necessary to accomplish high-quality communication to overcome the high losses of the THz band. The movement of the transmitter and the receiver, antenna beam misalignment, atmospheric conditions, battery, memory, and computing capacity of the devices make this a very complex problem. Because of its complexity, machine learning models need to be incorporated to solve this problem. Movement of the transmitter and the receiver, beamwidth of the antenna, channel selection and power allocation are a few of the parameters we can control to achieve higher capacity. For simplicity, this project deals only with channel selection and power allocation. Other parameters could be incorporated into the model in the future. Only drone altitude is considered for simplicity as well. The model can be improved to work on different altitudes, in the future.

...to be continued...

1.1 Related Work

For the channel model and transmittance data, we used [Saeed et al., 2020]. This paper makes use of different tools such as LBLRTM, HITRAN, and am-HITRAN to find and examine transmittance values on different altitudes, frequencies, and distances. Then, uses the transmittance data to calculate the absorption and path loss.

[Saeed et al., 2021] Is continued on top of the work on [Saeed et al., 2020]. The paper Chooses frequency bands that have a relatively constant path loss and a relatively constant noise power spectral density(PSD). An illustration of this is given in Fig. 3. The idea is that a higher capacity could be achieved via communication over these channels.

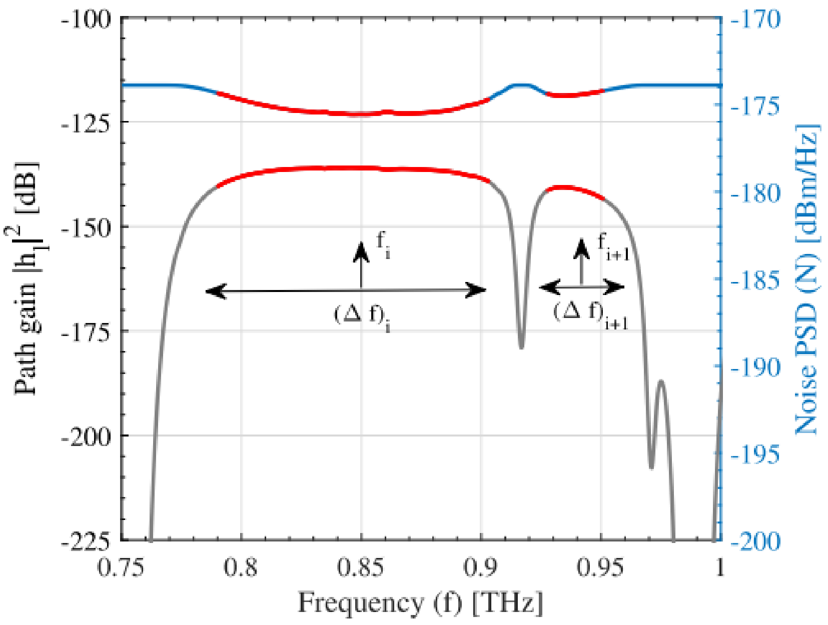

Figure 3: Visual illustration of the common flat bands among the path gain and the noise. [Saeed et al., 2021]

[Saeed et al., 2024] solves the exact problem we are trying to solve in this project. The paper suggests an algorithm called MaxActive that selects between the common flat band channels described in [Saeed et al., 2021] based on SNR. The paper also shows the capacity results of MaxActive and other channel selection methods used with Equal Power Allocation(EP) and Water Filling Power Allocation(WF).

Terahertz communications is a relatively new research area, and there are not many papers on channel selection for 2 devices to increase capacity in the THz band using machine learning models. However, there are lots of papers using machine learning models for channel selection on different problems. [Farhad and Pyun, 2023] is a comprehensive survey about the subject. Taking those researches as benchmarks can help create the foundations of a new machine learning model for our problem. Most of the papers in the literature use deep Q learning for channel selection on networks to improve communication over multiple agents. The one paper that this research is built on is [Tan et al., 2020].

2 Method

Our main objective is to solve joint resource allocation problem for Dr2DR communication in the THz band with the use of machine learning models. The resources we consider are channel selection and power allocation. We use deep Q learning as our machine learning model.

2.1 Deep Q Learning

2.1.1 Q learning

Q learning is a form of reinforcement learning. It is achieved by letting an agent interact with an environment and rewarding or punishing the agent based on the consequences of its actions. It is achieved by assigning a Q value to each state-action pair of the environment. The Q value represents the total expected reward assuming the agent always takes the best possible actions after taking action a at state s. It can be denoted as in Eq. 5.

$$Q(s_t, a_t) = r_t + \gamma\, Q(s_{t+1}, a_{t+1}) \tag{5}$$

Here, r_t denotes the immediate reward after taking action a_t in state s_t. $Q(s_t, a_t)$ denotes the Q value of the action a_t in state s_t, which also corresponds to the total expected reward assuming the agent always takes the best possible actions after taking action a_t at state s_t as mentioned before. $Q(s_{t+1}, a_{t+1})$ corresponds to the future rewards starting from the state s_{t+1}. We can write $Q(s_{t+1}, a_{t+1})$ in open form to achieve Eq. 6. This is called the Bellman Equation.

$$Q(s_t, a_t) = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \gamma^3\, r_{t+3}... \tag{6}$$

This way, we can calculate the Q values for each state-action pair. If the agent chooses the action with the highest Q value for a given state, it takes the optimal action. Then, the changes in the environment due to the action are observed, and a new state is obtained. This new state is again used to find the state-action pair with the highest Q value. If this loop is continued, the agent always takes the actions that maximize its rewards.

2.1.2 Deep Q network

Since our problem is complex with so many state-action pairs, that we can't determine the reward for each state-action pair. Therefore, we apply deep Q learning. In deep Q learning, Q values are determined by a deep neural network(DNN). An illustration of the deep Q learning algorithm is given in Fig. 4. In our implementation, the DQN has 3 hidden layers with sizes of 300, 200, and 100. The RelU activation function is used for the first two layers, and the last layer is linear.

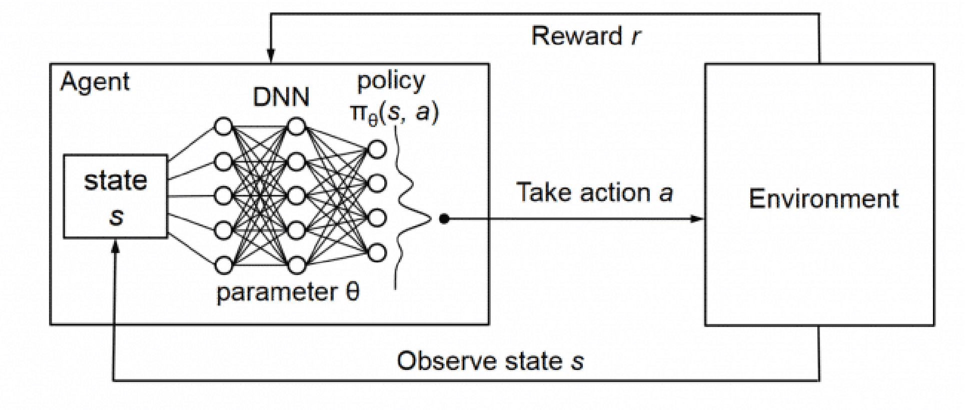

Figure 4: General deep Q learning algorithm.
[Bi et al., 2023]

The update equation for the Q function can be seen in Eq. 7.

$$\underbrace{\text{New}Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underset{\text{Learnin Rate}}{\alpha} [\underbrace{R(s,a)}_{\text{Reward}} + \underset{\text{Discount rate}}{\gamma} \overbrace{\max Q'(s',a')}^{\text{Maximum predicted reward, given new state and all possible actions}} - Q(s,a)] \quad (7)$$

The learning algorithm can be summarized as in Algorithm 1.

Algorithm 1 Q-learning: Learn function $Q : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$

```
Require:
  States $\mathcal{X} = \{1, \ldots, n_x\}$
  Actions $\mathcal{A} = \{1, \ldots, n_a\}$,      $A : \mathcal{X} \Rightarrow \mathcal{A}$
  Reward function $R : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$
  Black-box (probabilistic) transition function $T : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{X}$
  Learning rate $\alpha \in [0, 1]$, typically $\alpha = 0.1$
  Discounting factor $\gamma \in [0, 1]$
  procedure QLEARNING($\mathcal{X}$, $A$, $R$, $T$, $\alpha$, $\gamma$)
    Initialize $Q : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$ arbitrarily
    while $Q$ is not converged do
      Start in state $s \in \mathcal{X}$
      while $s$ is not terminal do
        Calculate $\pi$ according to Q and exploration strategy (e.g. $\pi(x) \leftarrow argmax_a Q(x, a)$)
        $a \leftarrow \pi(s)$
        $r \leftarrow R(s, a)$                                    ▷ Receive the reward
        $s' \leftarrow T(s, a)$                                   ▷ Receive the new state
        $Q(s', a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a'))$
        $s \leftarrow s'$
      end while
    end whilereturn $Q$
  end procedure
```

2.1.3 ϵ-greedy algorithm

This algorithm uses an ϵ greedy policy that is mentioned as ”exploration strategy” in Algorithm 1. This policy ensures that the agent will choose totally random actions with a probability of ϵ. This probability ϵ can take any value between $[0, 1]$. If we choose ϵ to be 1, it means all actions are random. If we choose it to be 0, it means all actions are chosen according to our deep Q network(DQN) which is called a greedy policy. In that case, the DQN might get stuck in a local maximum. For example, it can learn that an action returns high rewards, and keep choosing that action all the time, even if there are better actions available. Since the weights of the DQN are initialized randomly, the DQN gives out random Q values for state-action pairs. The DQN can’t learn the Q values for an action-state pair that it has never tried before. The policy that finds the middle ground between these two is called the ϵ-greedy policy. This way, the agent chooses random actions from time to time, therefore we ensure that the agent discovers different actions, and the DQN does not get stuck on a local maximum. In our implementation, we chose a decaying ϵ. It starts at 1 and linearly decreases until it reaches 0.01. The decay rate is chosen such that it takes a quarter of the total actions for ϵ to reach its minimum value.

2.1.4 Experience replay

Tthis algorithm can also be improved for better learning of the Q function. If we only use the last state-action pair chosen by the agent to update the Q function, the algorithm might get stuck in a local maximum. Just as mentioned in the previous subsection, the agent can learn that an action returns high rewards, and keep choosing that action all the time, even if there are better actions available. To overcome this we can store a bunch of previous state, action, and reward trios and choose a batch of these values to minimize the loss over that batch, just like traditional DNN algorithms. The DQN will learn more accurate Q values. This is called experience replay. In our implementation, we store the previous 2000 state, action, and reward trios and choose a batch of 256 of them to minimize the loss function at each step.

2.1.5 Quasi-static target network

Another improvement that can be made is the implementation of a quasi-static target network. Instead of using the same DQN for prediction and target values while calculating loss, we can use a different DQN for the calculation of target values. If we use the same DQN for both these values, we are basically chasing a target that can never be reached, because the update of the DQN also results in a new target. This makes the learning process unstable. Another problem with this is that some of the state, action, reward trios stored in our agent's memory might have non-optimal actions chosen. A batch that is randomly chosen might coincidentally have lots of these non-optimal data, therefore updating the DQN in an unfavourable way. In order to overcome these issues we can employ a quasi-static target network. In this implementation, the target value is calculated with a different DQN. This DQN is updated every few steps to match the current DQN that is used to calculate the predictions.

Algorithm 2 Quasi-static target network

```
Create a trained DQN and a target DQN with weights $\theta$ and $\theta'$ respectively. Initialize $\theta$ randomly, let
$\theta' = \theta$ and choose some integer N
for $k \leftarrow 1$ to $N$ do
    apply Algorithm 1 but use target DQN with weights $\theta'$ for the second Q term in Eq. 7.
end for
$\theta' \leftarrow \theta$
```

In our implementation, we choose N to be 100.

3 Implementation

We used OpenAI's Gymnasium framework and PyTorch to implement this project. The codes can be found in the link at the top of the first page of the paper.

The observation space consists of currently active channels, the distance between the transmitter and the receiver, the total path loss, and noise power values for each channel at the specified distance. The atmospheric condition data that is used to calculate the total path loss and the noise power is obtained from the International Telecommunication Union (ITU). The action space consist of 2 integers. The first integer is the channel chosen to be added to the active channels on the current step, the second integer is the channeş chosen to be removed from the active channels on the current step. The agent can choose to not add or remove a channel for the current step by choosing 0 for the corresponding integers.

Using the Gymnasium framework, the observation space and the action space is defined as:

```
import gymnasium as gym
from gymnasium import spaces

self.observation_space = spaces.Dict(
            {
                "channels": spaces.MultiBinary(self.n_channels),
                "distance": spaces.Box(low=1,high=11, dtype=np.int32),
                "loss": spaces.Box(low=0,high=1e18, shape=(self.n_channels,), dtype=np.float32
                    ),
                "noise": spaces.Box(low=3e-12, high=5e-12, shape=(self.n_channels,), dtype=np.
                    float32),
            }
        )

self.action_space = spaces.MultiDiscrete([self.n_channels+1, self.n_channels+1], dtype=np.
    int32)
```

After selecting the channels and allocating power to the channels, we can calculate the SNR of a channel as in Eq. 8.

$$\gamma(f_i, z_1, z_2, d, (\Delta f)_i) = \frac{P_T^i G_T}{P_n(f_i, z_1, z_2, d, (\Delta f)_i) A_{pathloss}} \tag{8}$$

P_T^i denotes the power allocated to channel i with center frequency f_i, G_T is the total antenna gain(transmitter and receiver) which is constant for our model, z_1 and z_2 are the altitudes of the transmitter and receiver respectively, d is the distance between the transmitter and the receiver, $(\Delta f)_i$ is the spectral resolution, which is chosen to be 1GHz for our project.

After finding the SNR for each channel, we can calculate the capacity using Eq. 9.

$$C(z_1, z_2, d) = \sum_{q=1}^{Q} (\Delta f) \log_2[1 + \gamma(f_i, z_1, z_2, d, (\Delta f)_i)] \tag{9}$$

which is a sum over all the selected channels.

We chose an equal power allocation scheme(EP) for the sake of simplicity. After this project, the model can be improved via the implementation of different power allocation schemes.

In EP, the total power P_T is allocated over selected channels equally. The power of each selected channel is found as in Eq. 10.

$$P_T^i = \frac{P_T}{\sum_{i=1}^{I} (\Delta f)_i} (\Delta f)_i \tag{10}$$

Another scheme that can be implemented in the project in the future is the water-filling (WF) power allocation scheme. In WF, the power of each selected channel is found as in Eq. 11.

$$\frac{P_T^i}{P_T} = \begin{cases} \frac{1}{\gamma_o} - \frac{1}{\bar{\gamma}_i}, & \bar{\gamma}_i \geq \gamma_o, \\ 0, & \bar{\gamma}_i < \gamma_o, \end{cases} \quad \text{s.t.} \quad \sum_{i=1}^{I} P_T^i \leq P_T \tag{11}$$

where γ_o denotes the SNR threshold. Which is found as in Eq. 12.

$$\sum_{i=1}^{I} \left(\frac{1}{\gamma_o} - \frac{1}{\bar{\gamma}_i} \right) = 1. \tag{12}$$

We determined the capacity difference on each step as the reward for the environment. If the capacity increases compared to the previous state, the agent receives a positive reward, if the capacity decreases the agent receives a negative reward. This causes the agent to take actions that improve the capacity or minimize the depreciation of the capacity if no better action is available.

4 Results

Figure 5 shows the change in ϵ and the running average of the capacity throughout the training. By looking at the decrease in the average capacity towards the end of the training, we can say that a higher minimum value or rather a slower decay for the ϵ might be beneficial. Of course, it is impossible to know without running another training with this change applied.

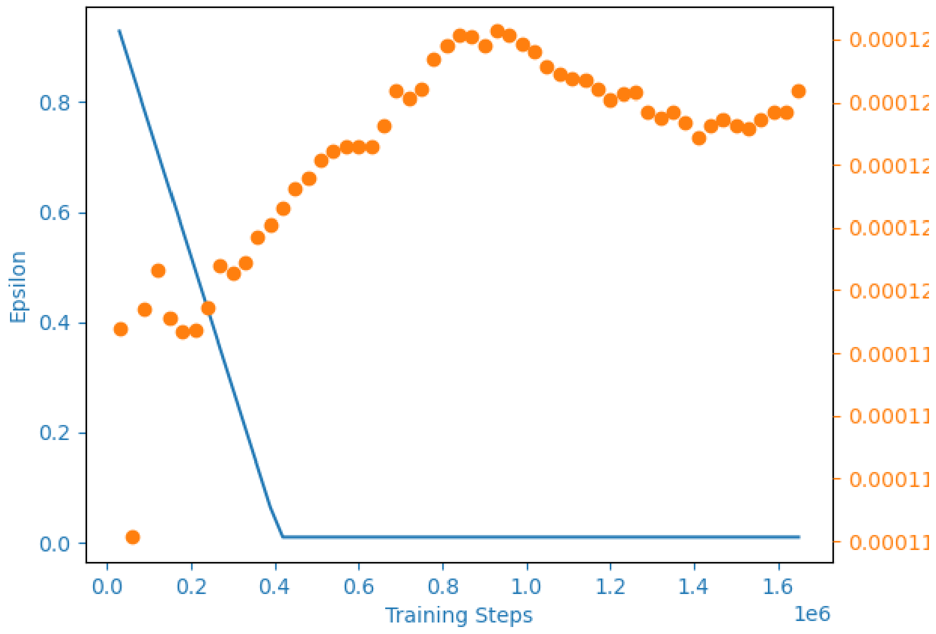

Figure 5: Capacity and epsilon change over the training

Figure 6 shows the change in ϵ and the running average of the rewards throughout the training. By looking at the decrease in rewards throughout training, we can arrive at the same conclusion as the previous paragraph. Also, we can see that the rewards started increasing towards the end, this might mean that we didn't give the DQN enough time to be trained. A longer training might result in better rewards towards the end of the training. We can also see that the rewards increase or remain constant for 3000 steps but then drop significantly every 30000th step. This is because every 3000th step, the environment is reset, and a random new observation is given to the agent to train for the next 3000 steps. As we can see 3000 steps is not enough time for the agent to make enough changes on the active channels to achieve optimal capacity. Increasing the number of steps before the environment is reset is an option, but it would probably take too many steps for the agent to reach the optimal channel allocation for capacity maximization in the current model. Instead, a change in the design of the environment would make much more sense. Specifically, changing the action space to a continuous action space which allows the agent to choose intervals. The first reason current action space format is problematic for a few reasons. Firstly, it takes too many steps to change the active channels from a random selection to the desired selection, therefore it is unpractical. Secondly, there are too many possible actions which makes the learning take impractically long since the DQN can't learn which action results in how much of a reward without trying them first.

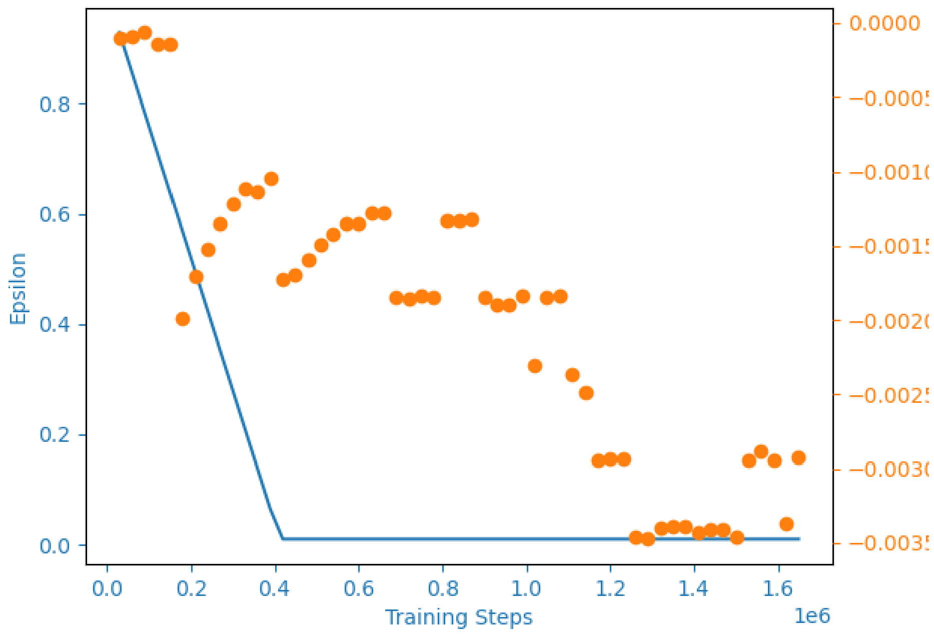

Figure 6: Received reward and epsilon change over the training

5 Discussion and Conclusions

References

Bi, Z., Guo, X., Wang, J., Qin, S., & Liu, G. (2023). Deep reinforcement learning for truck-drone delivery problem. *Drones*, *7*(7), 445.

Farhad, A., & Pyun, J.-Y. (2023). Terahertz meets ai: The state of the art. *Sensors*, *23*(11), 5034.

Huq, K., Busari, S. A., Rodriguez, J., Frascolla, V., Bazzi, W., & Sicker, D. (2019). Terahertz-enabled wireless system for beyond-5g ultra-fast networks: A brief survey. *IEEE Network*, *33*, 89–95.

Saeed, A., Erdem, M., Saleem, A., Gurbuz, O., & Akkas, M. A. (2024). Joint resource allocation for terahertz band drone communications. *IEEE Transactions on Vehicular Technology*.

Saeed, A., Gurbuz, O., & Akkas, M. A. (2020). Terahertz communications at various atmospheric altitudes. *Physical Communication*, *41*, 101113.

Saeed, A., Gurbuz, O., Bicen, A. O., & Akkas, M. A. (2021). Variable-bandwidth model and capacity analysis for aerial communications in the terahertz band. *IEEE Journal on Selected Areas in Communications*, *39*(6), 1768–1784.

Tan, J., Liang, Y.-C., Zhang, L., & Feng, G. (2020). Deep reinforcement learning for joint channel selection and power control in d2d networks. *IEEE Transactions on Wireless Communications*, *20*(2), 1363–1378.