# Machine Learning Driven Latency Optimization for Application-aware Edge Computing-based IoTs

Liang Zhang, Bijan Jabbari

lzhang36@gmu.edu, bjabbari@gmu.edu

Communications and Networks Laboratory,

Department of Electrical and Computer Engineering,

George Mason University,

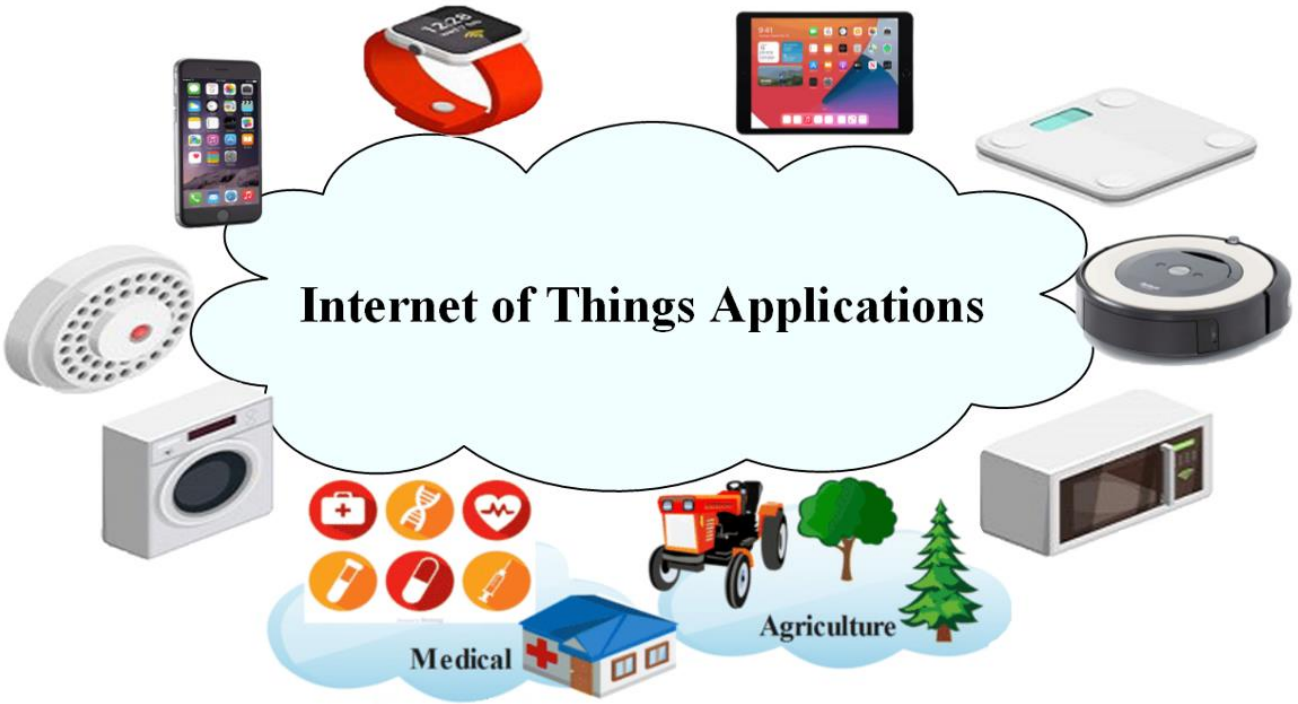
Fairfax, VA 22030 USA

# Outline

- **Background of Edge Computing for IoT**
- **System Model and Problem Formulation**
- **Algorithm and Analysis**
- **Evaluation Results**
- **Conclusions**

# Internet of Things Applications

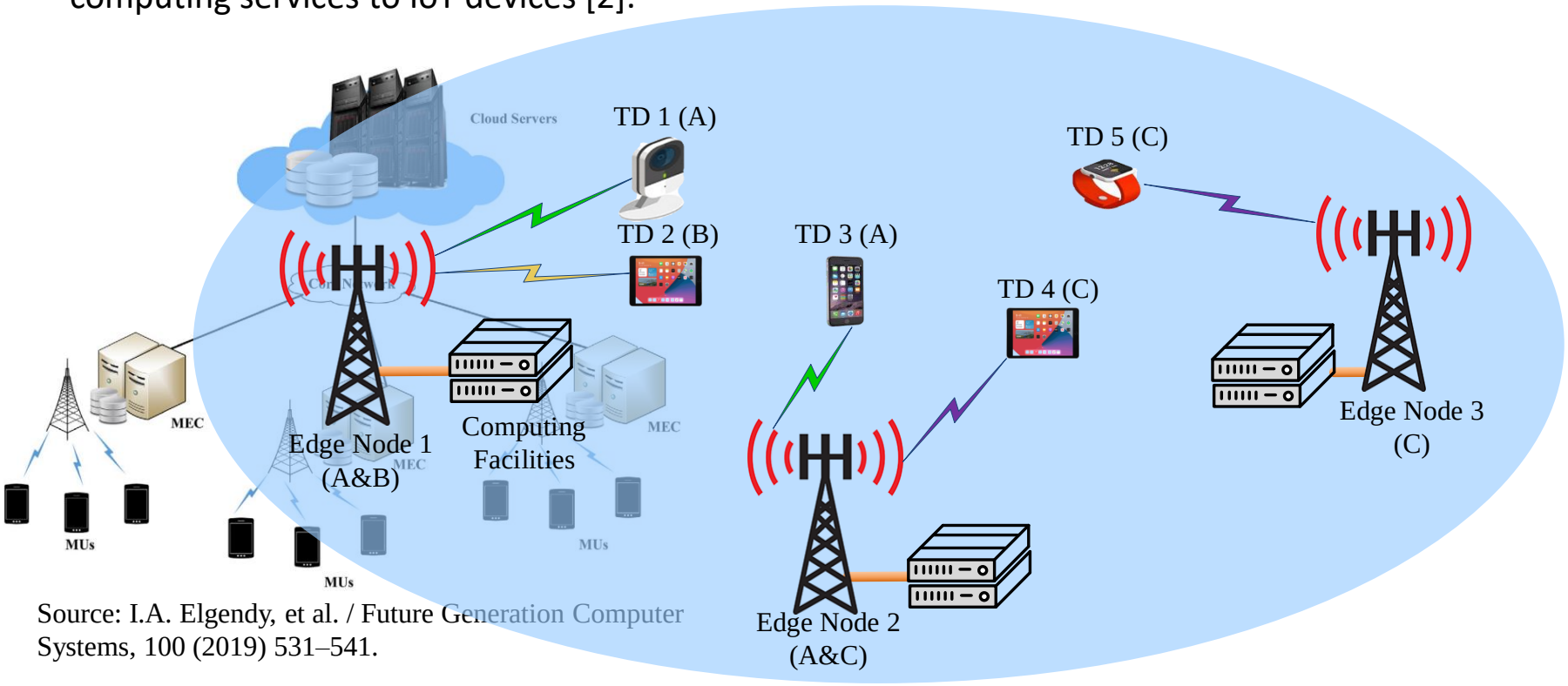
- Billions of Internet of Things (IoT) devices are connected to the Internet, and many IoT devices have limited power, computing and storage resources [1].
- Various IoT applications, have been widely studied to improve our daily life. Different applications may have various resource preference.




[1] C. Qiu et al., "Networking integrated cloud–edge–end in IoT: A blockchain–assisted collective Q-learning approach," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12 694–12 704, Aug. 2021.

# MEC for IoT Applications

- MEC is a network architecture that pushes cloud computing capabilities at edge nodes that are close to users and connected to cloud servers via a core network.
- Mobile edge computing (*MEC*) is effective in reducing the latency for communications and computing services to IoT devices [2].




Source: I.A. Elgendy, et al. / Future Generation Computer Systems, 100 (2019) 531–541.


[2] P. Wang et al., "Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2872–2884, Apr. 2019.

# Outline

- Background of Edge Computing for IoTs
- System Model and Problem Formulation
- Algorithm and Analysis
- Evaluation Results
- Conclusions

# An Edge Computing Framework

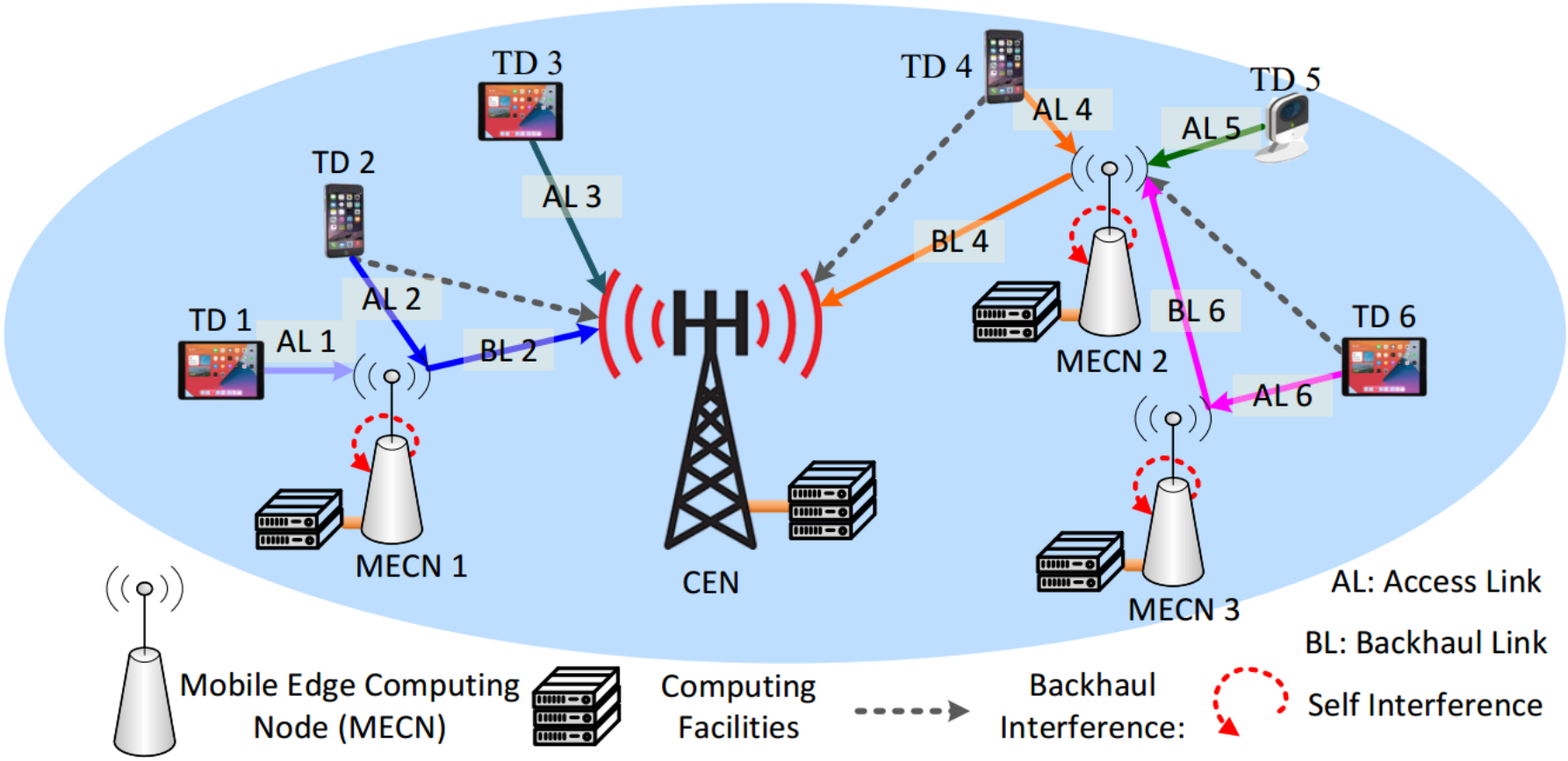



Fig. 3: A novel framework for edge computing based IoTs.

# Communication and Computing Model

- Let $s_{i,j}$, $\beta_{i,j}$ and $r_{i,j}$ be the signal to interference plus noise ratio (*SINR*), the assigned bandwidth, and the received data rate of user $i$ towards the edge node $j$, respectively.

$$d_{i,j} = \max\left(d_{i,j}^1, d_{i,j}^2\right)$$

$$\begin{cases} d_{i,j}^1 = f_0 b_{i,j}\, log_2(1 + s_{i,j}) \\ d_{i,j}^2 = \min\left(d_{i,j'}^a, d_{j',j}^b\right) \end{cases}$$

Here, $d_{i,j}^1$ is the data rate of one hop communications; $d_{i,j}^2$ is the data rate of two-hop communications; $d_{i,j'}^a$ and $d_{j',j}^b$ are the data rate of the access link and the backhaul link.

- Let $t_{i,j}^C$ and $t_{i,j}^T$ be the computation latency and the transmission delay from TD $i$ to edge node $j$. If TD $i$ is served by edge node $j$, the total delay is:

$$t_{i,j} = t_{i,j}^C + t_{i,j}^T$$

# Problem Formulation

- We formulate the **AP**plication-aware **E**dge-Io**T** (***APET***) problem with the objective of minimizing the average latency of all TDs.

$$\mathscr{P}_0 : \max_{\omega_{i,j}, b_{i,j}, \tau_{i,j}} \quad \frac{1}{|\mathscr{U}|} \sum_i \sum_j \omega_{i,j} t_{i,j}$$

$$s.t. :$$

$$C1 : \sum_j \omega_{i,j} \leq 1, \quad \forall i \in \mathscr{U},$$

$$C2 : \omega_{i,j} \leq q_{i,k} q^E_{j,k}, \quad \forall i \in \mathscr{U}, j \in \mathscr{E}, k \in \mathscr{K},$$

$$C3 : \sum_i \omega_{i,j} b_{i,j} \leq f_j^{max}, \quad \forall j \in \mathscr{E},$$

$$C4 : \sum_i \omega_{i,j} \tau_{i,j} \leq C_j, \quad \forall j \in \mathscr{E},$$

$$C5 : \sum_j \omega_{i,j} \beta_{i,j} \leq 1, \quad \forall i \in \mathscr{U},$$

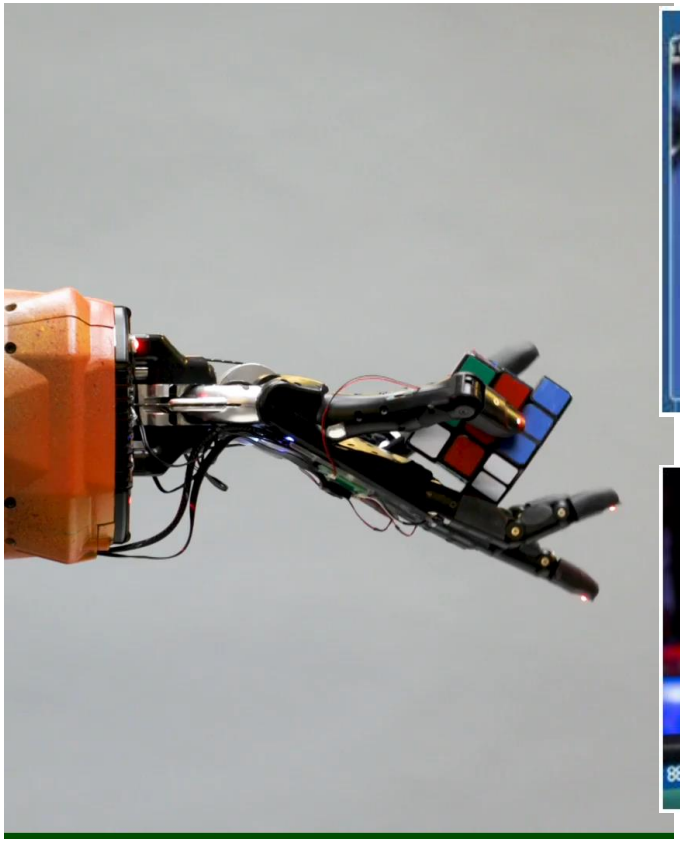
$$C6 : \omega_{i,j} \in \{0, 1\}, \quad \forall i \in \mathscr{U}, j \in \mathscr{E}. \qquad (9)$$

# Outline

- Background of Edge Computing for IoTs
- System Model and Problem Formulation
- Algorithm and Analysis
- Evaluation Results
- Conclusions

# Deep Reinforcement Learning

- Machine learning is a branch of artificial intelligence and computer science, which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy*.



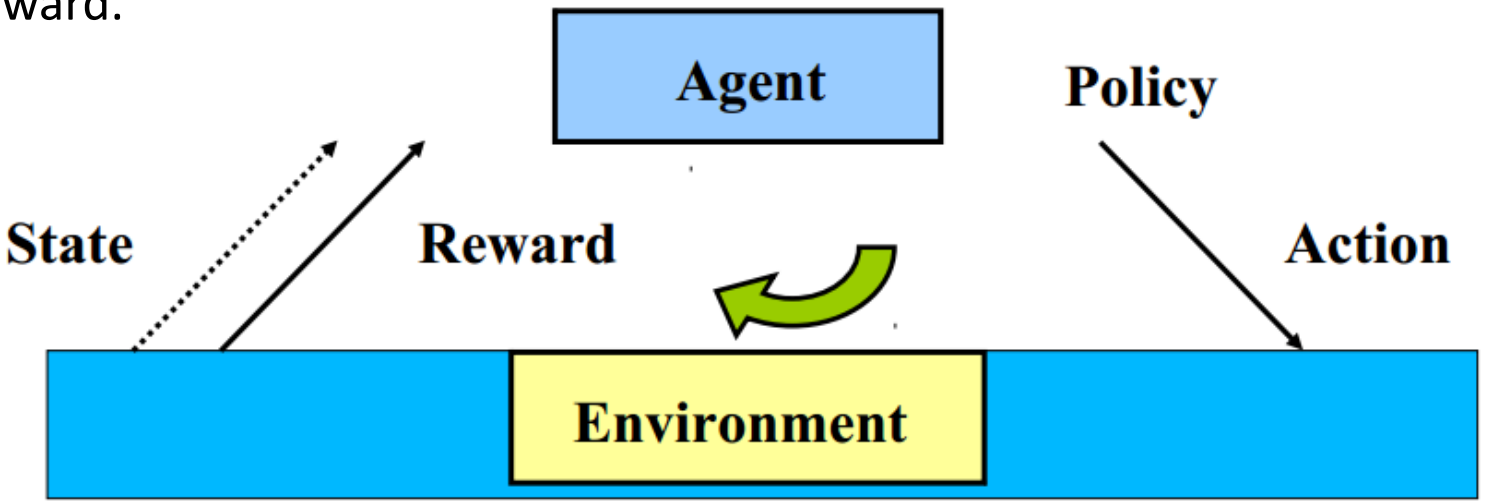




[*] Source: https://www.ibm.com/cloud/learn/machine-learning

# Elements of Deep Reinforcement Learning

- **Agent**: Intelligent programs
- **Environment**: External condition
- A typical fully connected neural network includes three layers: the input layer, the hidden layer and the output layer.
- **The output** (action $a(t)$) is determined by the input (state $s(t)$), **the reward** ($r(t)$) is determined by the output, and **the weights** are updated based on the reward.



# The framework of DDPG

- State:

$$s(n) = \{H(E), H(U), r_i, c_i, q_i, q_i^E, t_{i,j}\}$$

- Action:

$$a(n) = \{\omega_{i,j}\}$$

- Reward:

$$g(t) = \frac{1}{|U|}\sum_i \sum_j t_{i,j}$$

- **Actor network**: input is s, output is a
- **Critic network**: input is (s, a), output is Q(s,a)
- **Target actor network and target critic network:**

  input is actor network and critic network

  it is used to calculate the loss function of the critic network

# The framework of DDPG

- The critic network is updated based on the loss function:

$$\begin{cases} L(\theta^{Q}) = \dfrac{1}{K}\sum_{k=1}^{K}(g(k) - Q(s(k), a(k)|\theta^{Q} + \gamma A_1)^2, \\ A_1 = Q(s(k+1), a(k+1)|\theta^{Q^-})). \end{cases} \tag{6}$$

- The actor network is updated based on the gradient policy as follows.

$$\begin{cases} \nabla_{\theta^{\varphi}} J(\theta^{\varphi}) = \dfrac{1}{K}\sum_{k=1}^{K}(A_2 \cdot A_3), \\ A_2 = \nabla_a Q(s, a|\theta^{Q})\big|_{s=s(k),a=\varphi(s(k))}, \\ A_3 = \nabla_{\theta^{\varphi}} \varphi(s|\theta^{\varphi})\big|_{s=s(k)}. \end{cases} \tag{7}$$

- Then, the target networks are updated as:

$$\begin{cases} \theta^{\varphi^-} \leftarrow \eta\theta^{\varphi} + (1-\eta)\theta^{\varphi^-}, \\ \theta^{Q^-} \leftarrow \eta\theta^{Q} + (1-\eta)\theta^{Q^-}. \end{cases} \tag{8}$$

# The DDPG-APET Algorithm

**Algorithm 1:** DDPG-APET

**Input :** $\mathscr{B}$, $\mathscr{U}$, $u_i(t)$, $\varphi(s(t)|\theta^{\varphi})$, $Q(s(t),a(t)|\theta^{Q})$, $\varphi(s(t)|\theta^{\varphi-})$ and $Q(s(t),a(t)|\theta^{Q-})$;

**Output:** $g(n+1)$;

**for** *epoch m* **do**
- Initialize the actor network, the critic network, the target actor network and the target critic network with the weights $\theta^{\varphi}$, $\theta^{Q}$, $\theta^{\varphi}$, and $\theta^{Q}$, respectively;
- Initialize the replay buffer $\eta = 0$;
- Initialize $s(n)$, $a(n)$ and $g(n)$;
- **for** *training step n* **do**
  - Obtain $H(\mathscr{E})$, $H(\mathscr{U})$, $r_i$, $c_i$, $q_i$, $q_j^E$, and $t_{i,j}$;
  - Generate $s(n+1)$;
  - Record the sample $(s(n),a(n),g(n),s(n+1))$ in replay buffer;
  - $\eta = \eta + 1$;
  - **if** $\eta \geq \eta^{batch}$ **then**
    - Update weight of the critic network $\theta^{Q}$ by Eq. (10);
    - Update weight of the actor network $\theta^{\varphi}$ by Eq. (11);
    - Update weights of target networks by Eq. (12);
  - Generate action $a(n+1)$;
  - Add noise to action $a(n+1) = a(n+1) + a'(n+1)$;
  - Obtain $g(n+1)$ based on $a(n+1)$ and Algorithm 2;
- return the largest $g(n+1)$ in all epoches;
- Calculate $\omega_{i,j}$ based on the final action $a(n+1)$;
- Obtain $t_{i,j}$ based on $a(n+1)$ and Algorithm 2;

**Update Critic network**

**Update Actor network**

**Update Target Actor network**

**Update Target Critic network**

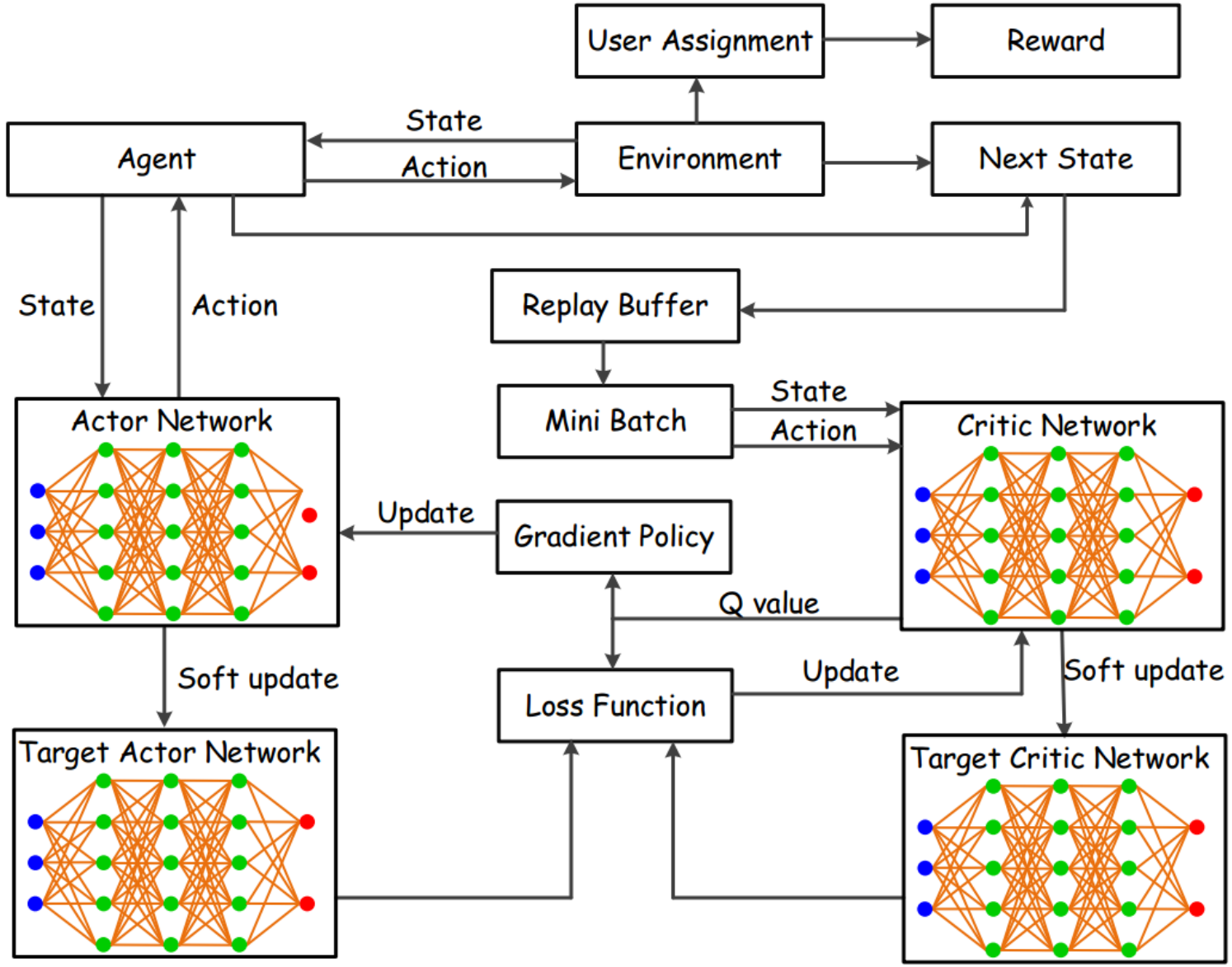
# Joint Resource Assign Algorithm

- An optimal resource assignment strategy is designed for the computing and communication resource allocation based on the given TD assignment.

**Algorithm 2:** Joint Resource Assign Algorithm

**Input** : $\mathscr{E}$, $\mathscr{U}$, $f_j^{max}$, $C_j$, $\omega_{i,j}$;
**Output:** $b_{i,j}$ and $\tau_{i,j}$;
**for** *edge node* $j$ *in* $\mathscr{E}$ **do**
  Obtain the TD set $\Omega_j = \{\omega_{i,j} = 1\}$;
  Initialize $\Delta b$, $\Delta\tau$, $\Delta t_{i,j}$ and $t_{i,j}$;
  $f_j^R = f_j^{max}$, $C_j^R = C_j$, $f_j^U = 0$ and $C_j^U = 0$,;
  **while** $f_j^R \geq \Delta b$ *and* $C_j^R \geq \Delta\tau$ **do**
    **for** *TD* $i$ *in* $\Omega_j$ **do**
      Obtain $t'_{i,j}$ if $\Delta b$ and $\Delta\tau$ are assigned to TD $i$;
      Calculate $\Delta t_{i,j} = t_{i,j} - t'_{i,j}$;
    Find $i' = \underset{i}{\operatorname{argmax}}\, \Delta t_{i,j}$ ;
    Update $b_{i',j} = b_{i',j} + \Delta b$ ;
    Update $\tau_{i',j} = \tau_{i',j} + \Delta\tau$;

# The Diagram of the DDPG-APET Algorithm



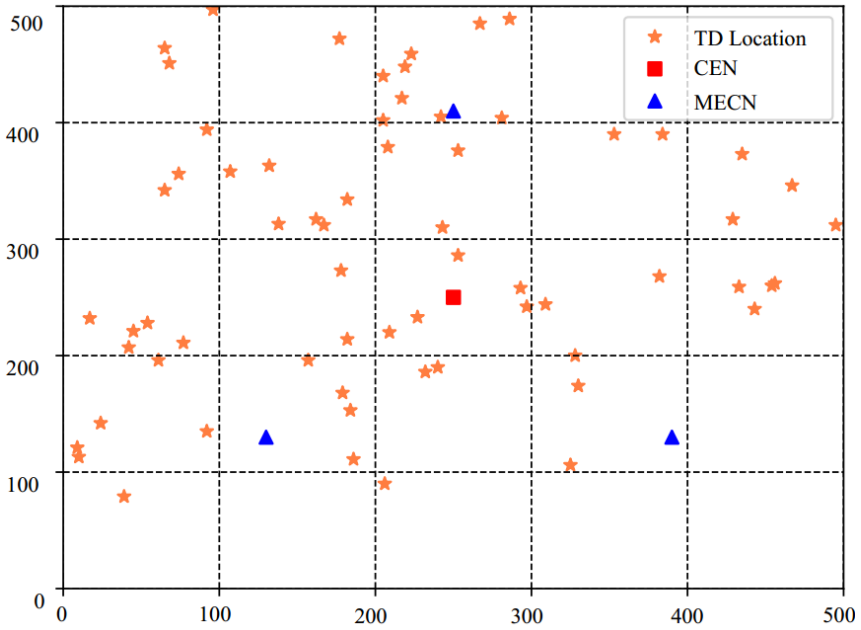
# Outline

- Background of Edge Computing for IoTs
- System Model and Problem Formulation
- Algorithm and Analysis
- Evaluation Results
- Conclusions

# Simulation Settings

- We use Python3.7 and Pycharm 2020.3.2 to run our simulations, and utilize Tensorflow 2.4 (optimizer is tf.keras.optimizers.Adam).
- The coverage area is set as 500 $m$ × 500 $m$, and all edge nodes are placed in fixed locations as shown in figure below.
- The same parameters are utilized to initialize the actor networks and the critic networks.
- Three baseline algorithms are utilized to evaluate the performance of the proposed algorithm.

  1) Best-APET
  2) Fair-APET
  3) ) S-Edge



# Evaluation Results

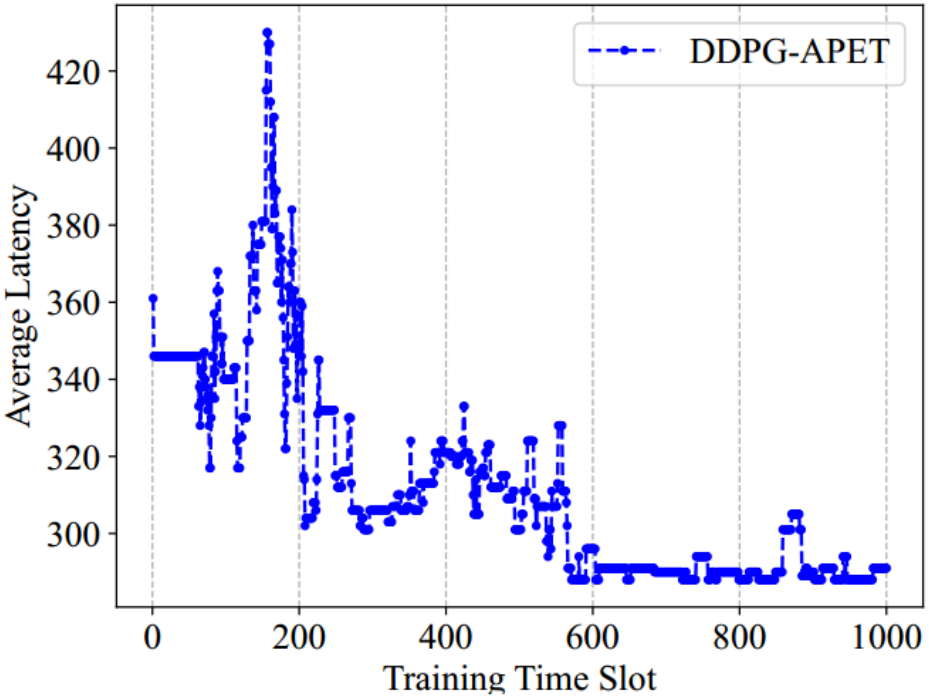



Fig.3: Convergence results versus different batch size.

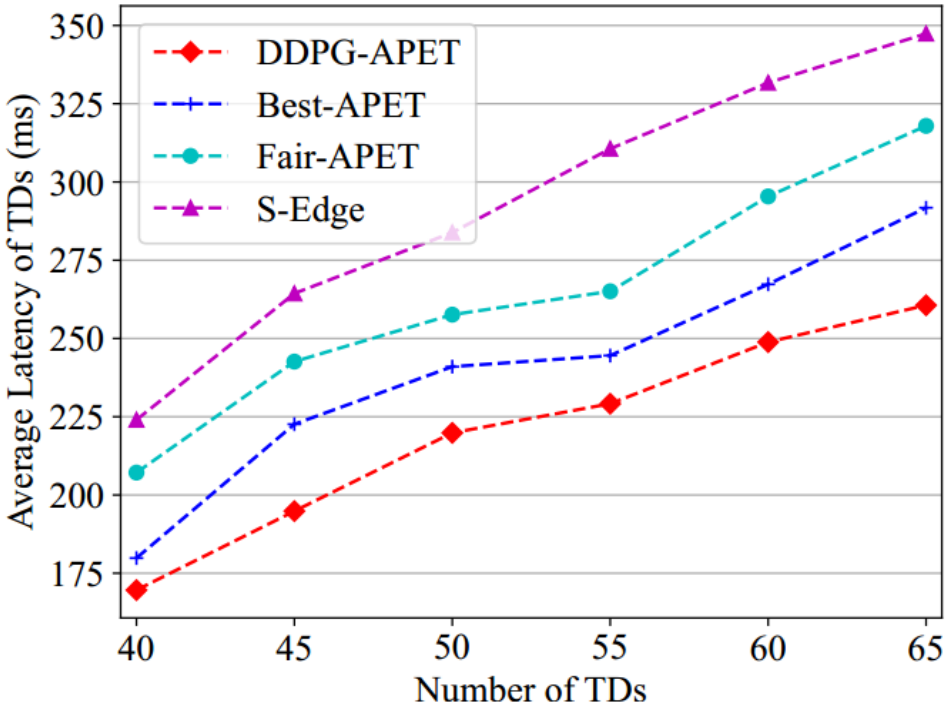



Fig.4: Average QoE versus number of users.

# Evaluation Results (Cont'd)

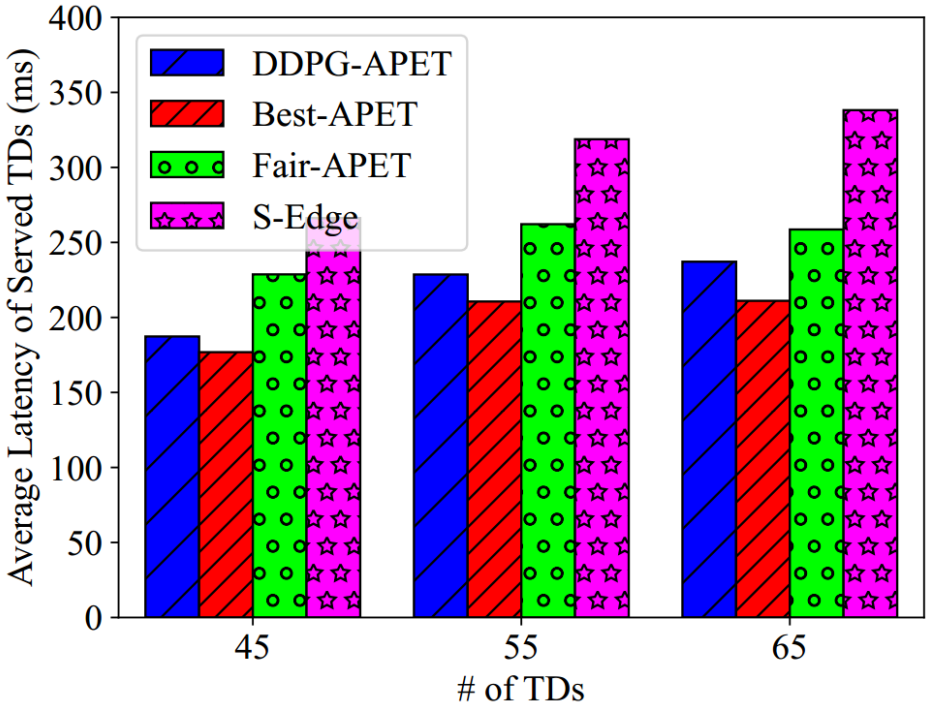



Fig. 5: Average latency of served TDs versus # of TDs.

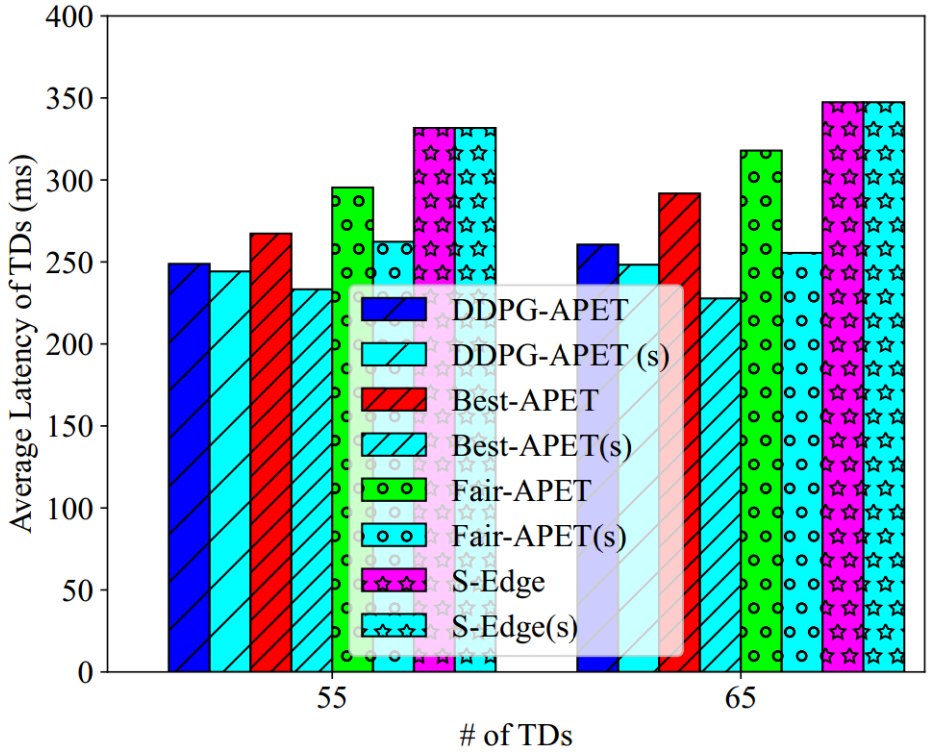



Fig. 6: Comparison of the average delay.

# Evaluation Results (Cont'd)

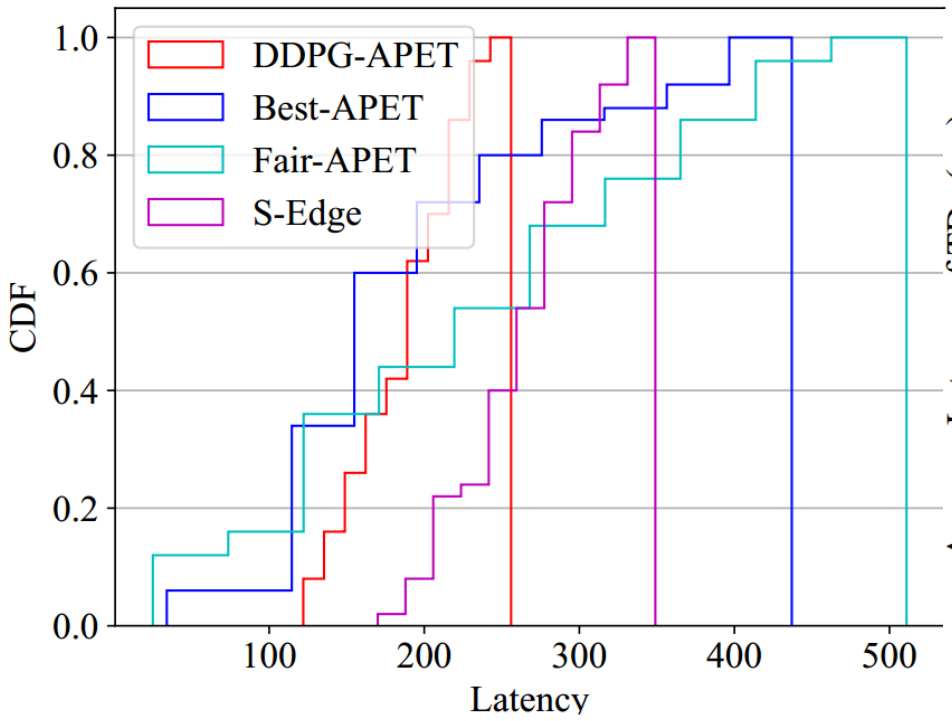



Fig. 7: Cumulative distribution function results.

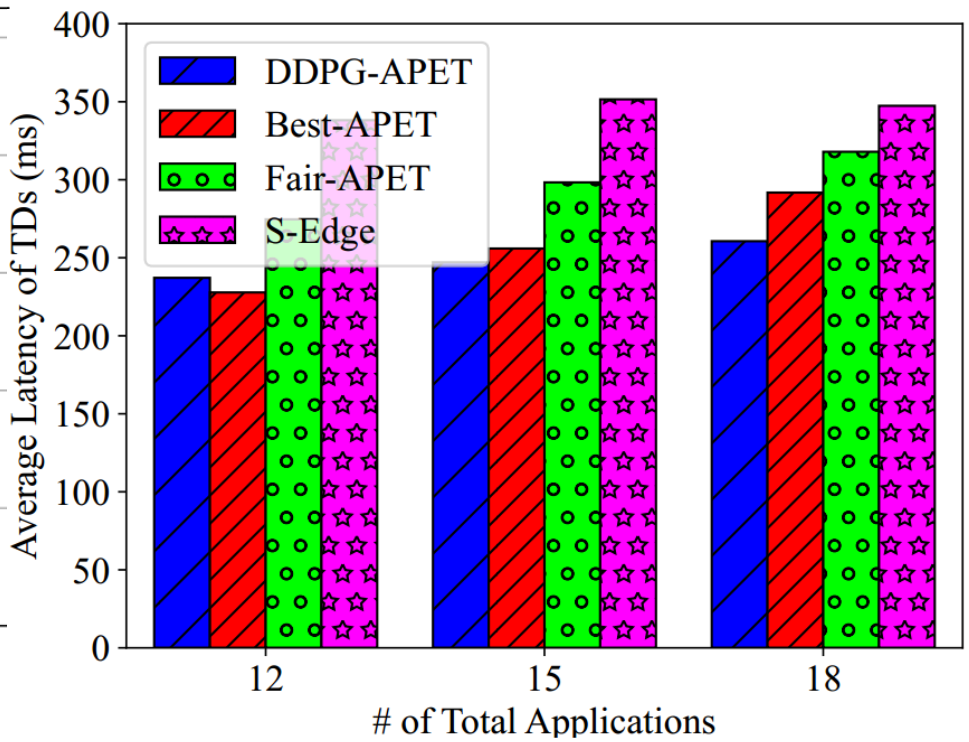



Fig. 8: Average latency of TDs versus # of applications.

# Outline

- Background of Edge Computing for IoTs
- System Model and Problem Formulation
- Algorithm and Analysis
- Evaluation Results
- Conclusions

# Conclusions

- We have proposed a novel edge-computing framework to serve IoT devices with different types of applications, and formulated the application-aware edge-IoT problem with the target to optimize the average latency of all IoT devices.

- we have proposed a deep deterministic policy gradient algorithm to solve the application-aware edge-IoT problem and designed an optimal joint resource assignment strategy to assign resources.

- We demonstrate that the proposed deep deterministic policy gradient algorithm has up to 27% average delay improvement as compared to baseline algorithms.