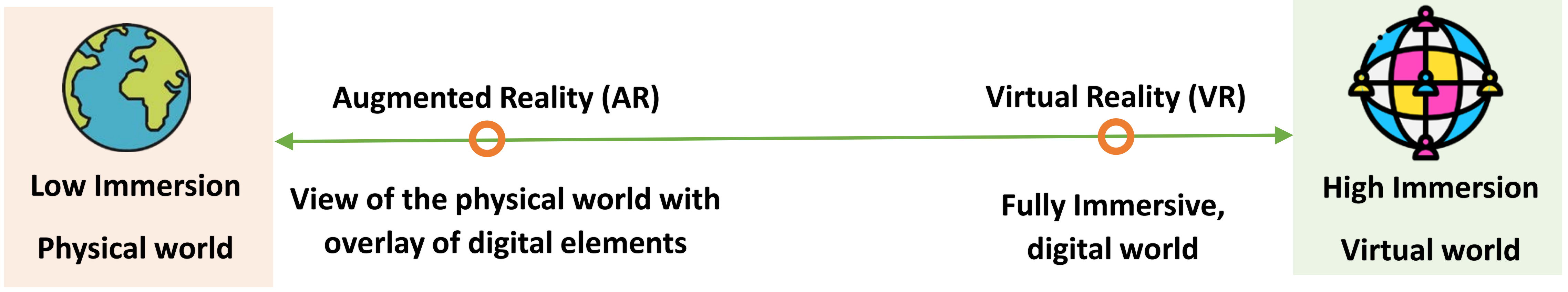
Asynchronous and User-centric **Reinforcement Learning** for the **Metaverse**

XR 3D real-time reconstruction in Metaverse

Extended Reality (XR) in Metaverse

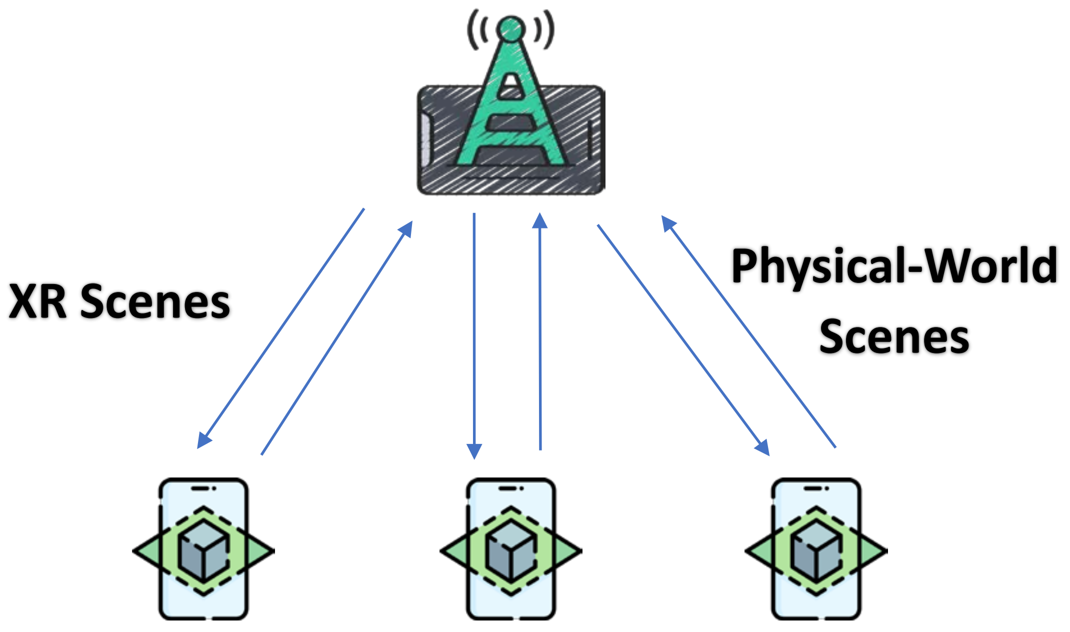
Extended Reality (XR)

Umbrella term for any alteration of real-world with digital elements



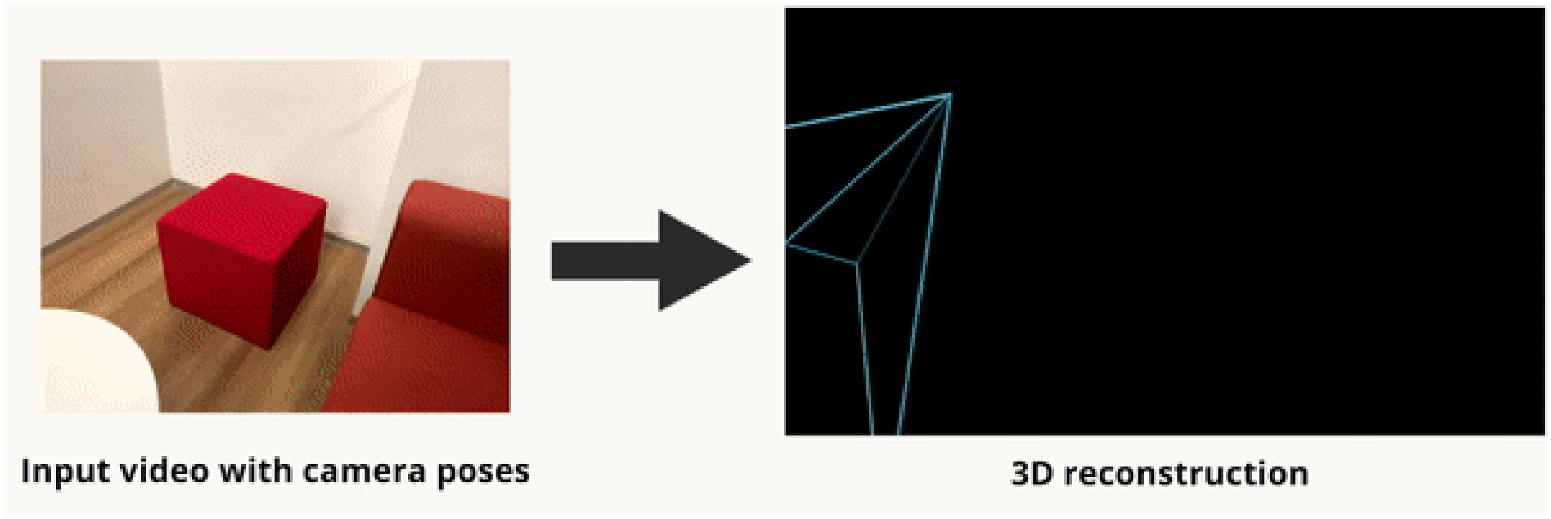
- **Metaverse is an extension, complete simulation, and mirror of the real world.**
- **XR improves interactivity and connectivity between people**
- **With the frequent demands of reflecting real objects in the virtual world, we need to consider an instant replication of 2D physical world images into 3D virtual world scenes.**

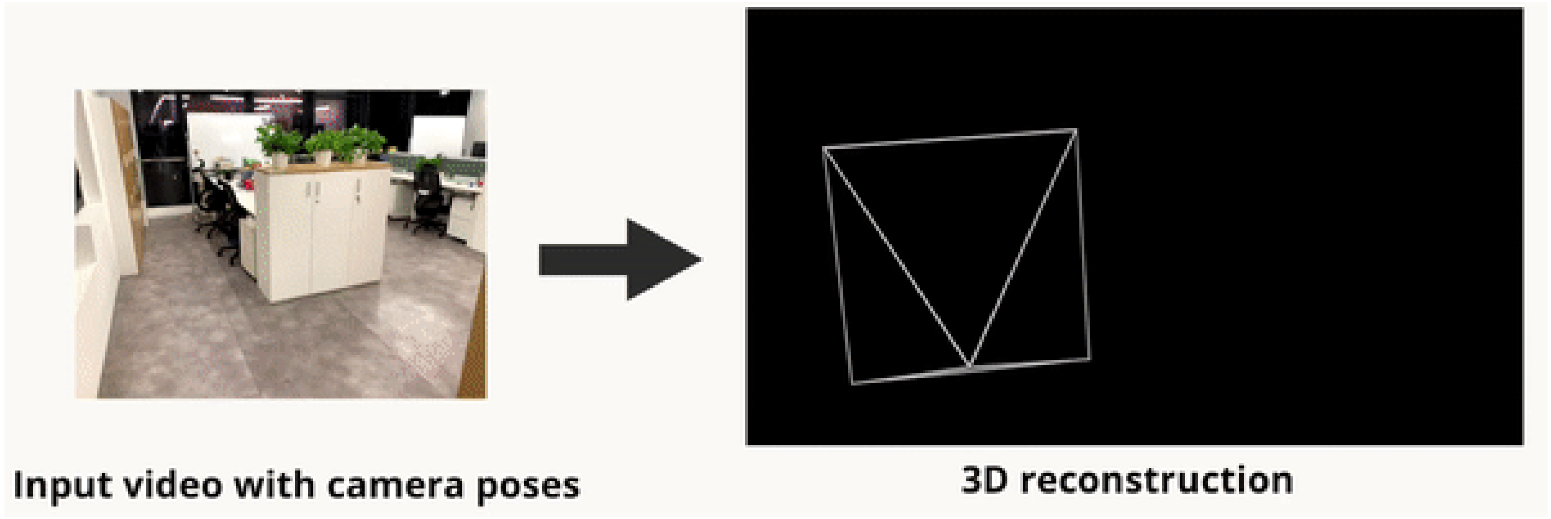
Computation offloading



- Devices reduce their weight at the sacrifice of their computation ability, leading to insufficient local computing power.
- Mobile devices capture and send the physical-world scenes to the server.
- Server handles physical-virtual world scene integration and sends virtual replicas back to users

A specific scenario: Real-time 3D reconstruction

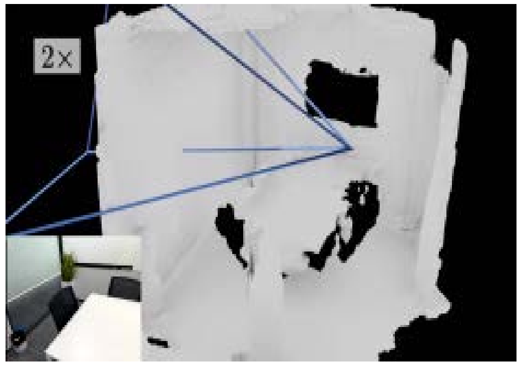


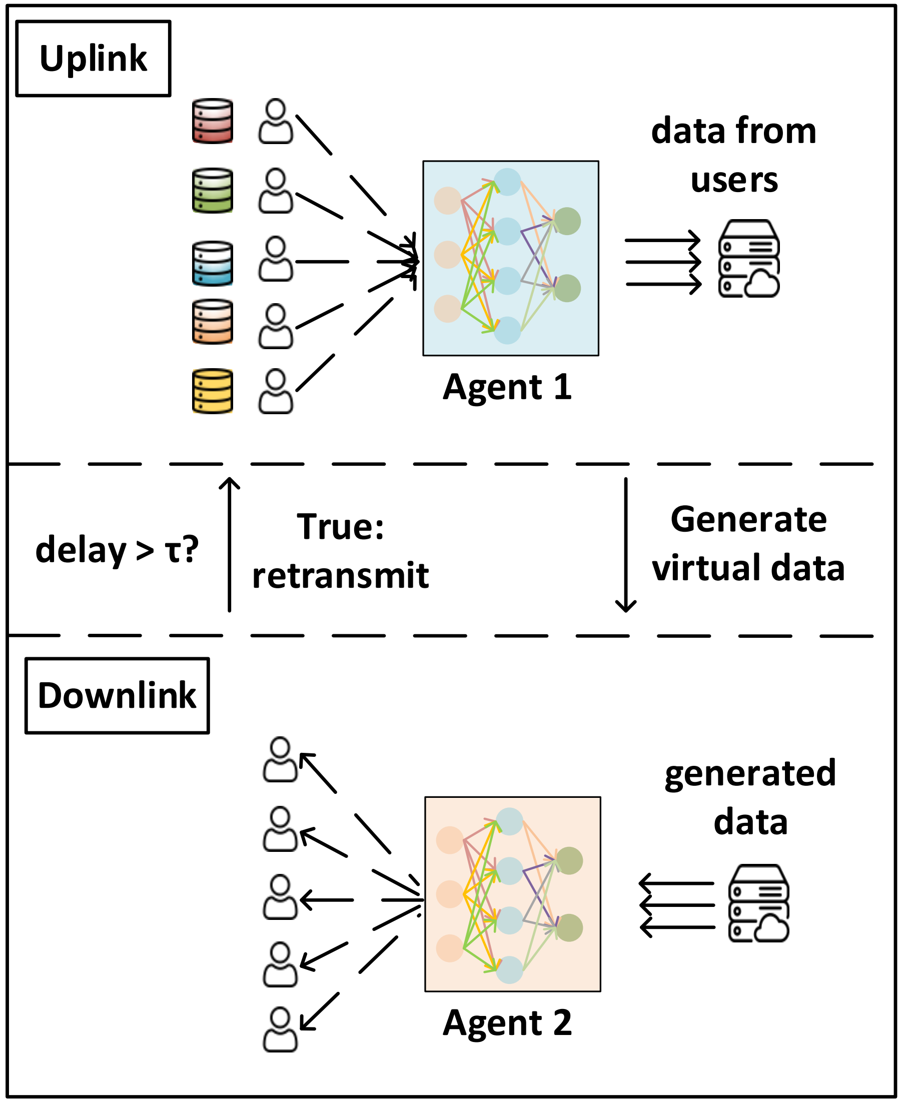


NeuralRecon
by Sun et al.
CVPR 2021

A specific scenario: Real-time 3D reconstruction

- UL: Users upload real scenes to the server to generate the digital replica and to be rendered.
- DL: The server transfers the generated models to each user.
- Problem arises: Asymmetric data sizes in UL-DL stages.

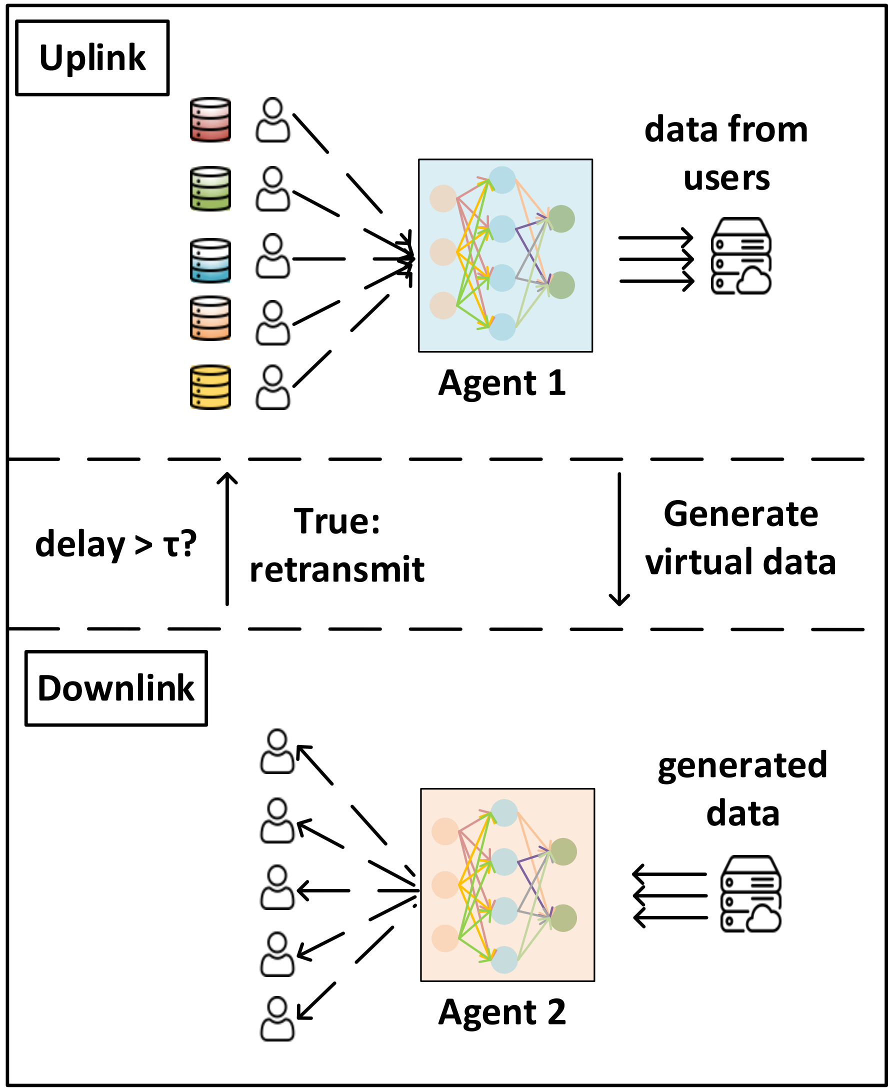




- We separate the transmission into multiple UL-DL turns.

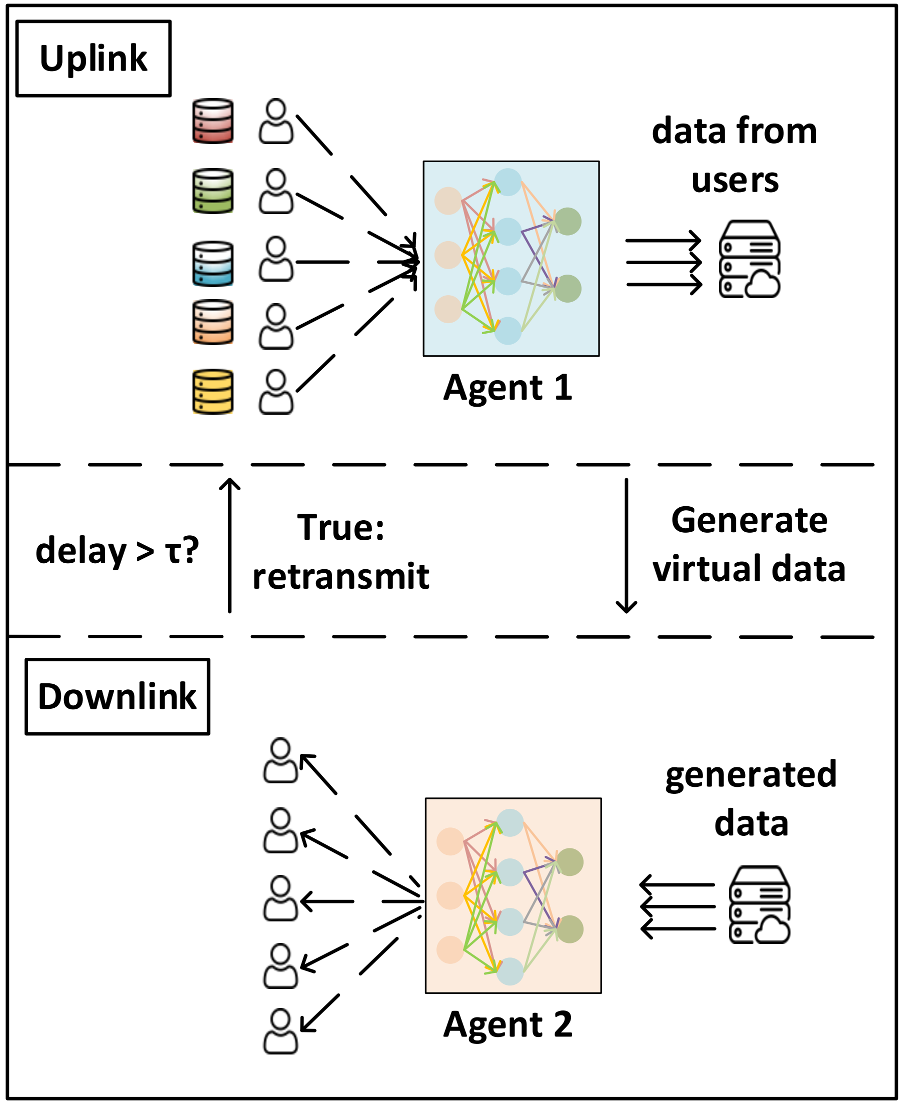
- Influence between the asynchronous processes can not be neglected. In such circumstances, the rendered 3D virtual objects can possibly be orders of magnitude larger than the uploaded scenes.

- This difference in data sizes introduces an issue of sensitivity, where a slight change in the UL transmission size may possibly induce a dramatic change in the DL data size.



- To ensure reliability, multiple processes need to be considered in tandem.

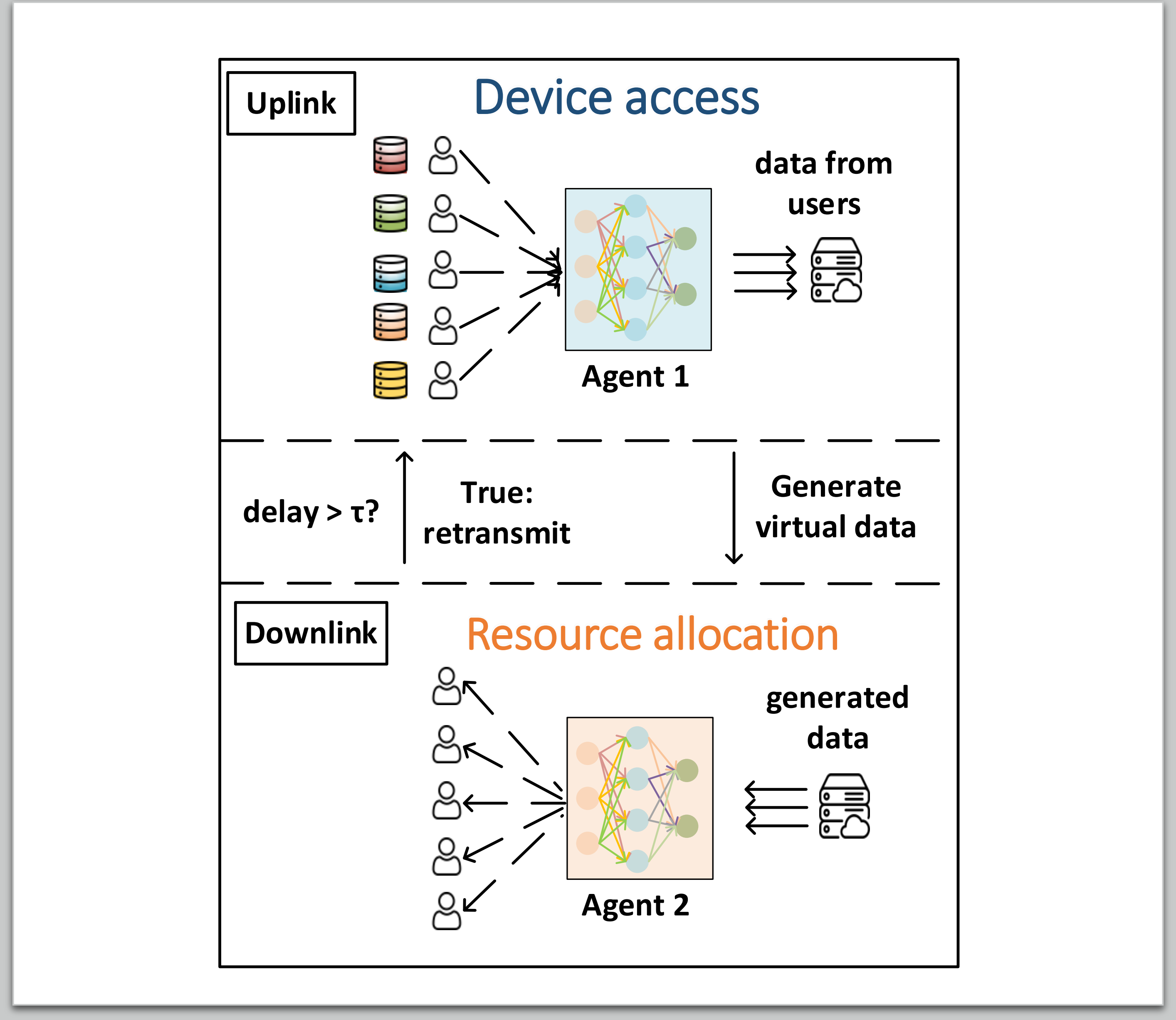
- To avoid the overlong delay in one DL transmission, we need to set a limit to each turn. And if the DL delay exceeds the time limit, the data uploaded in this turn is lost and will be retransmitted in future turns.



- We can optimize the device access in the UL stage and optimize the computation resource allocation in the DL stage.

- The DL uses the same device access arrangement in UL.

- Therefore, DL optimization is totally based on UL, and some metrics in one turn (e.g., round delay) can only be obtained after all agents have selected actions.



Multi-stage joint optimization

● The users' left-to-be-transmitted data is continuous, which is sequential and causes the T (time steps) variable in the problem formulation. If we apply convex optimization strategies, there is daunting computation complexity.

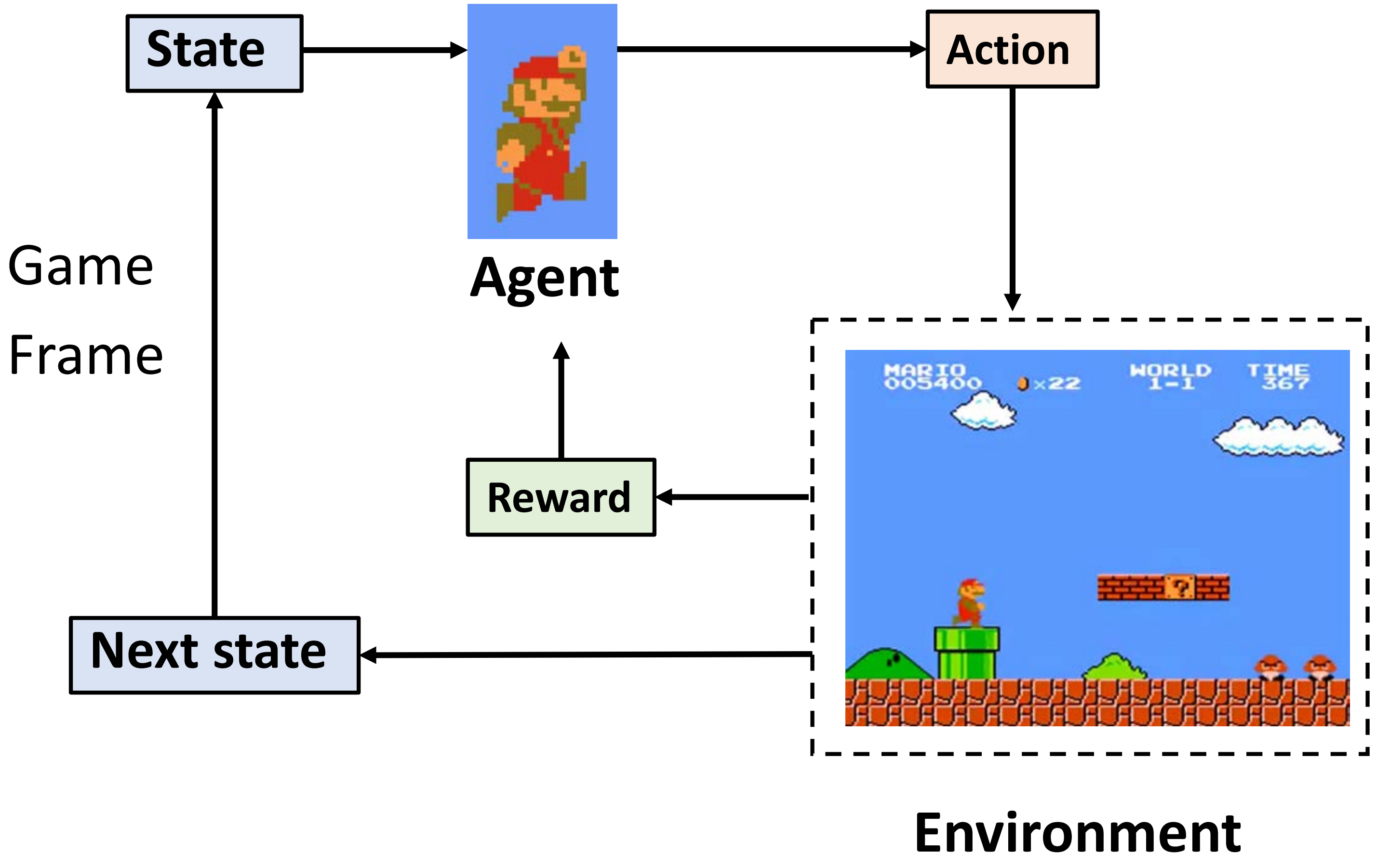
● Problems in the true-to-real-world scenario are usually naturally non-convex, relaxation is not always suitable, because it may lead to solutions that are far from the original problems. And there are many constraints.

● Model-free Reinforcement Learning (RL) can achieve near-optimal solutions with sufficient training, and it can handle complex joint optimization problems by well-designed algorithms and structures.

Reinforcement Learning Background

RL environment settings

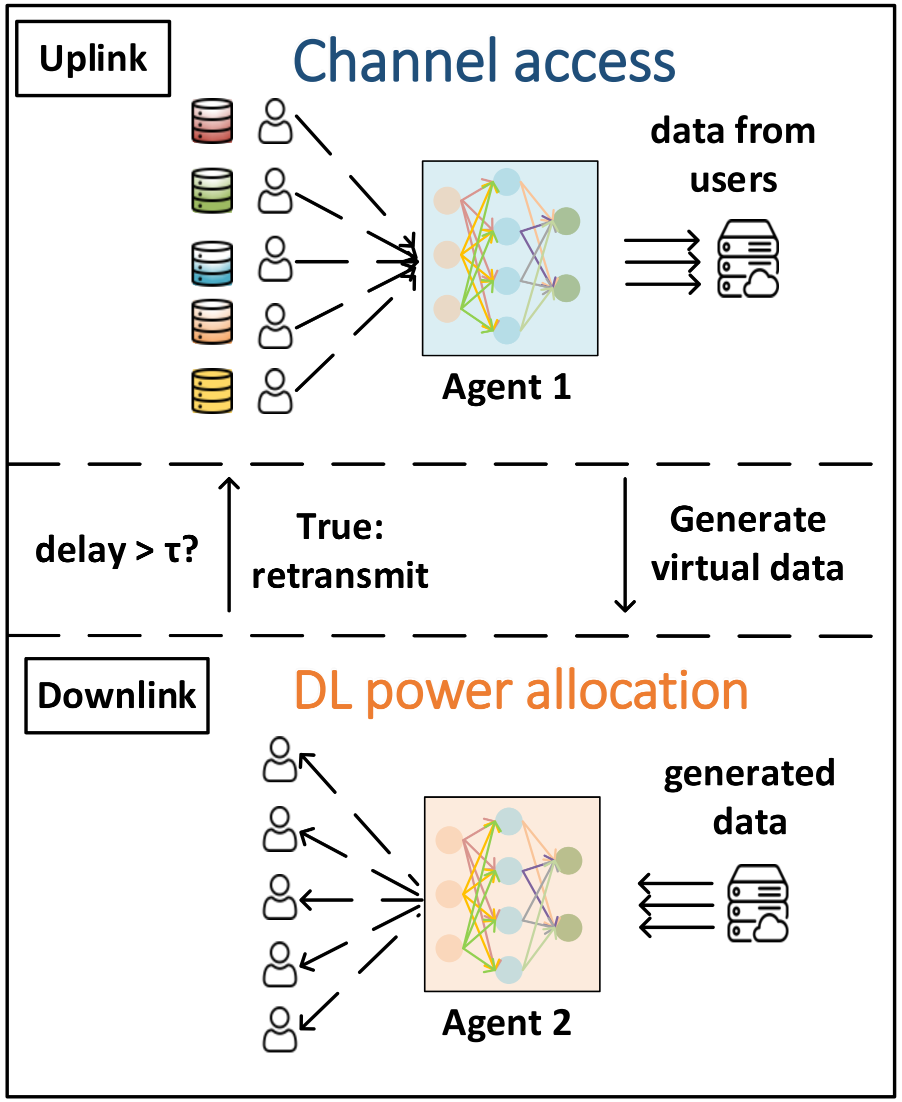
- The game frame is sent to Agent as the state, and Agent selects the best action according to the state.
- The environment provides the agent with the reward according to its action under state, as the feedback for the Agent to evaluate the selected action.
- The objective is to maximize the accumulated rewards.



RL environment settings

- States of the two Agents are different
- Actions, reward are different
- Device access: Discrete

  Resource allocation: Continuous



Basic Algorithms

Deep Q-learning network (DQN):

- $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$.

- Q is the action value, which is the value of action A under state S. α is learning rate, γ is reward discount factor with time steps.

- DQN must predict values of every action, which is infeasible in continuous action space.

Actor-Critic Structure

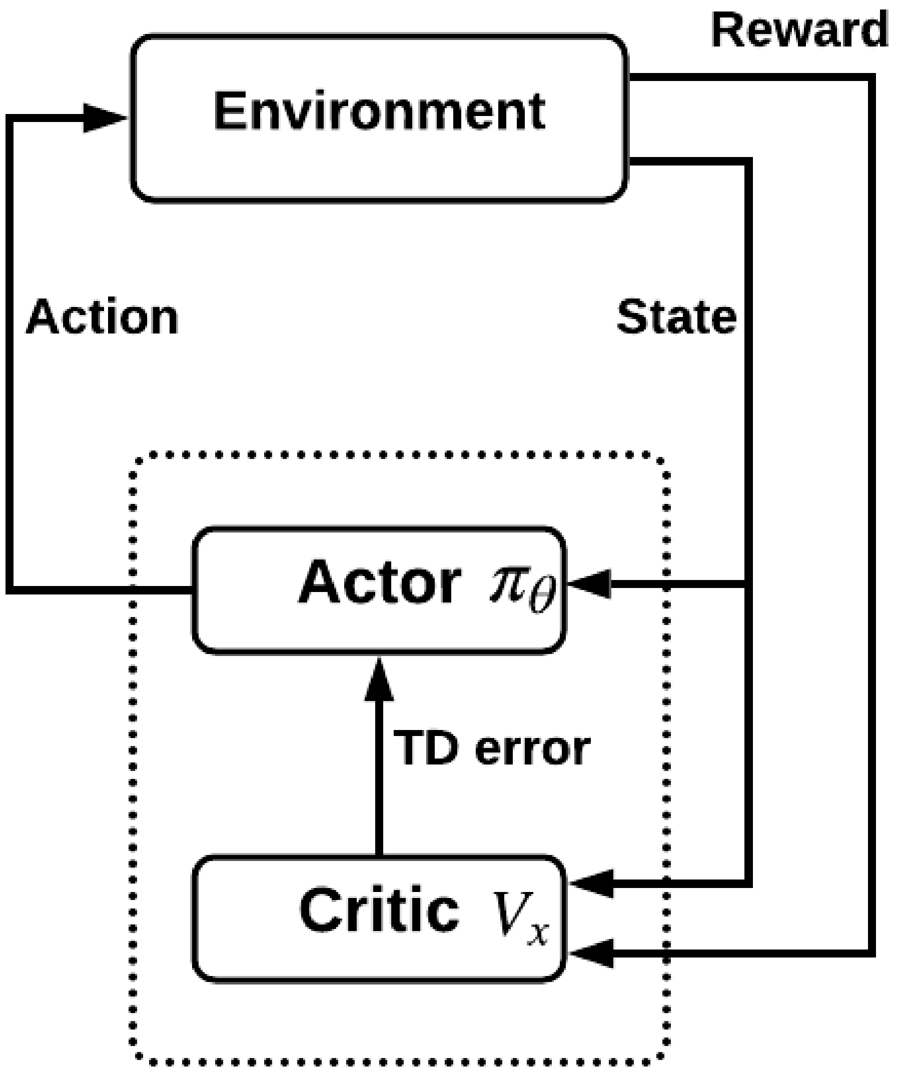
- Actor: responsible for action selection, update through policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \boxed{G_t}\right] \rightarrow G_t = \sum \gamma^t \times R_t$$

- Critic: responsible for evaluating the action, updated by Mean Square Error (MSE) loss between the estimated Q value and factual Q value

$$R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$$

$$R_{t+1} + C_{\phi'}(S_{t+1}, A_{t+1}) - C_\phi(S_t, A_t)$$

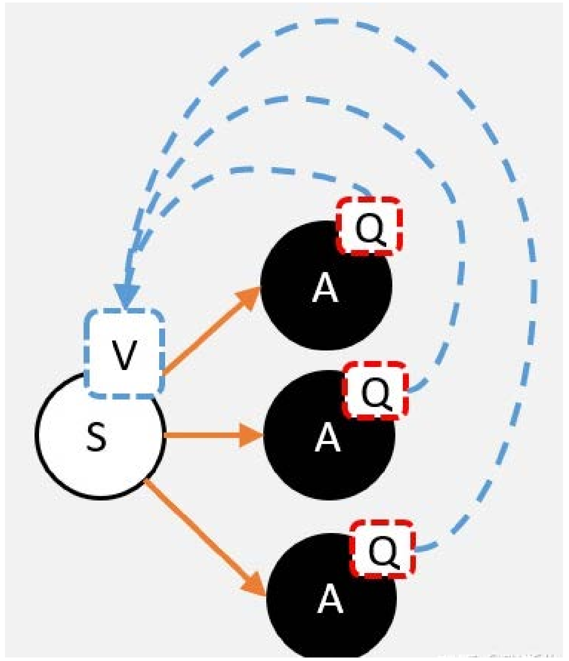


Advantage Actor-Critic Structure

- It uses a baseline to evaluate the advantage of the action under state S.

- The widely used baseline is the state value, V(s). Advantage is A = Q(S,A) - V(S)

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\boxed{G_t}] \rightarrow A_t$$



Asynchronous MDP

Reinforcement Learning

- Good at solving complicated sequential problems in randomly evolving environments.
- Actor-critic methods can optimize both discrete and continuous variables.
- Easy to consider different aspects by setting the reward function.

$$\begin{tikzcd}
 & \text{Asynchronous multiple processes} \arrow[dl] \arrow[dr] & \\
\text{Asynchronous MDP} & & \text{Multi-agent RL}
\end{tikzcd}$$

Asynchronous MDP

Traditional MDP

T : $S_1 \times A_1 \times R_1 \times S_2 \times A_2 \times R_2 \times S_3 \ldots \times A_N \times R_N$

Asynchronous MDP

T : $S_1^1 \times A_1^1 \times R_1^1 \times S_1^2 \times A_1^2 \times R_1^2 \ldots \times A_1^N \times R_1^N \times S_2^1$

X_t^k: X of agent k at time step (iteration) t.

- Agent 1: Channel Access Arrangement
- Agent 2: Power Allocation, and use the channel assignment selected by agent 1
- Asynchronous, iterative multiple processes that are connected and influenced by each other

Centralized Training Decentralized Execution

CTDE (widely used Multi-agent RL framework)

T : S × A_1 × A_2 ×… × A_N × R_{sum} × S’

Asynchronous MDP

T : $S_1^1 \times A_1^1 \times R_1^1 \times S_1^2 \times A_1^2 \times R_1^2 \ldots \times A_1^N \times R_1^N \times R_1^G \times S_2^1$

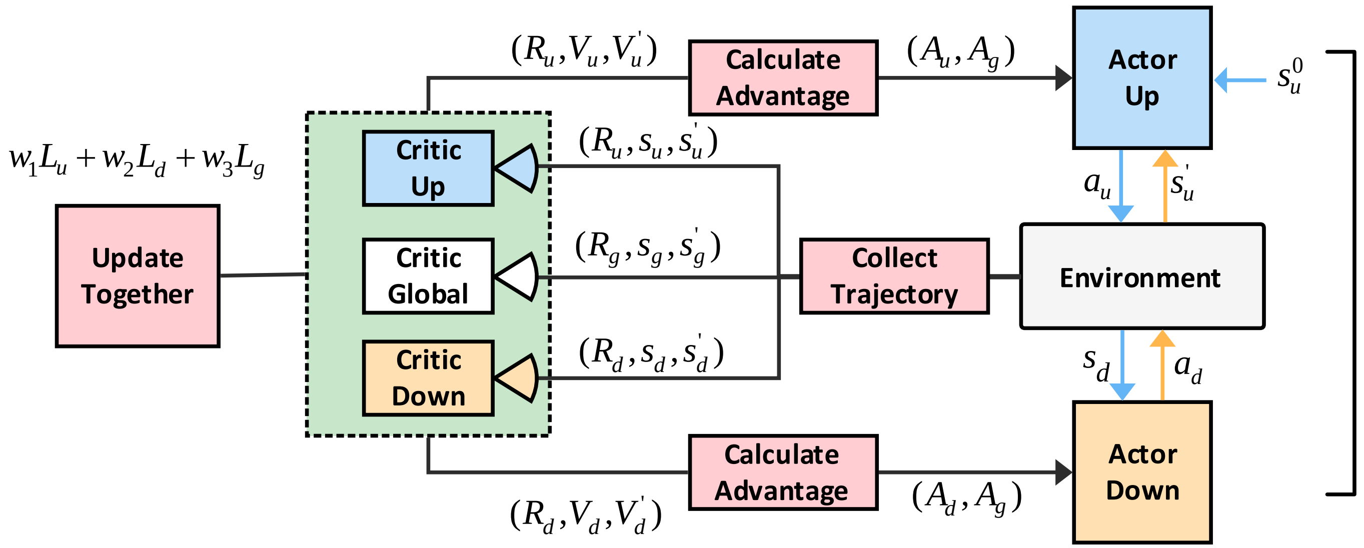
In our scenario:

- States of different agents are different.
- State of Agent 2 is based on the action of Agent 1.

  e.g., channel access, and uploaded data sizes.
- Specifically, the *Global Reward* can only be obtained after all agents’ actions, and it is related to both Agents.

Asynchronous Actor Hybrid Critic

Asynchronous Actor Hybrid Critic



- Use asymmetric Actors. They take in their own current state and output the selected action.
- Use multiple input and output Critic (Hybrid Critic).

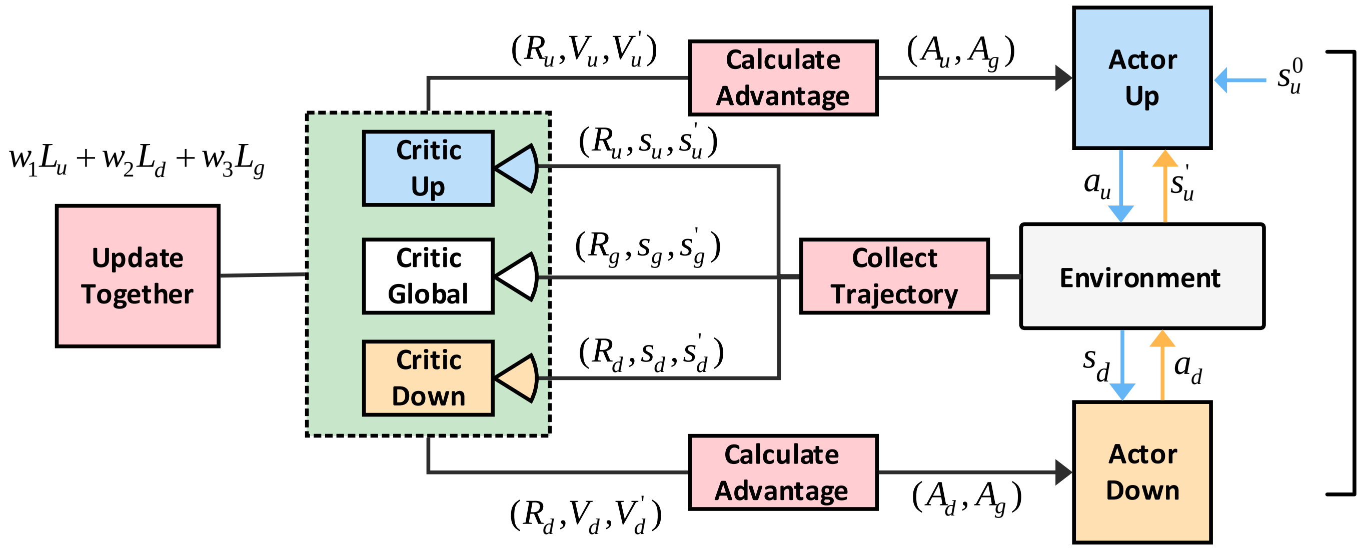
  The Critic takes in the UL state, DL state, and global state, and evaluates the values of the three kinds of states, respectively.

Asynchronous Actor Hybrid Critic

- In such a hybrid-reward scenario, directly giving the agent extra information from the other agent is not always advisable, the redundant information from the other Agent will even impinge on the action evaluation.

- Therefore, we use the Global Advantage as a connection between the two Agents, to equip them with a global view of the whole system

Asynchronous Actor Hybrid Critic

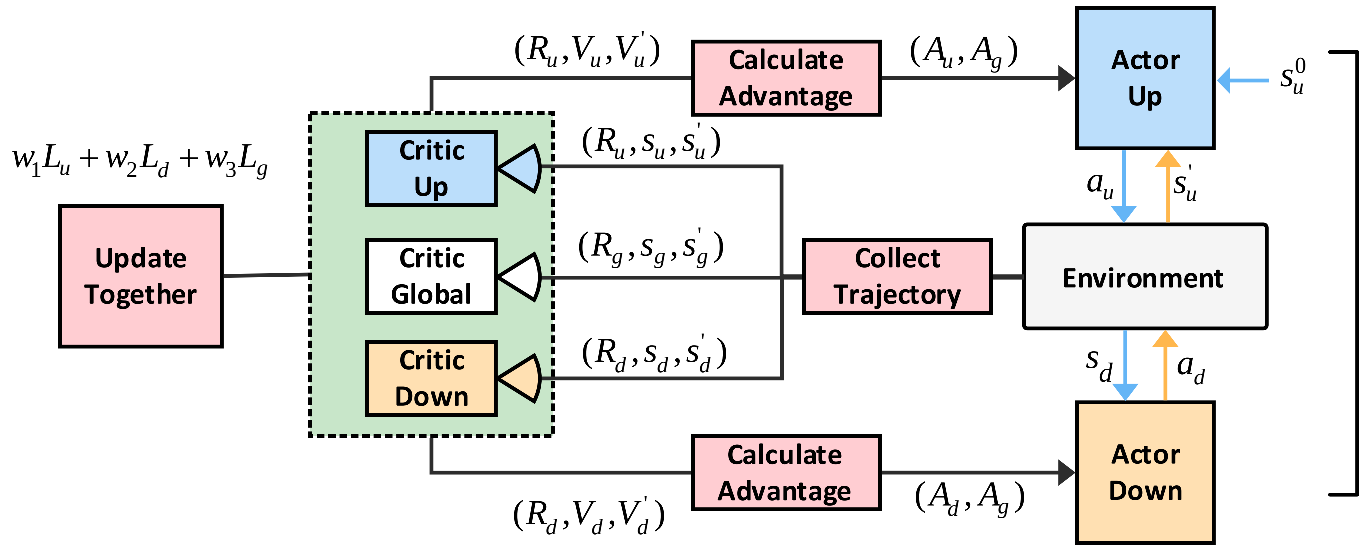


- The Actors update with the sum of their own and the Global Advantage.

$$\Delta\theta_1 = \mathbb{E}_{(s_u^t, a_u^t)\sim\pi_{\theta_1'}}[\nabla f^t(\theta_1, \boxed{(A_u(s_u^t) + A_g(s_g^t))}] \rightarrow \text{Uplink + Global Advantage}$$

$$\Delta\theta_2 = \mathbb{E}_{(s_d^t, a_d^t)\sim\pi_{\theta_2'}}[\nabla f^t(\theta_2, \boxed{(A_d(s_d^t) + A_g(s_g^t))}] \rightarrow \text{Downlink + Global Advantage}$$

Asynchronous Actor Hybrid Critic



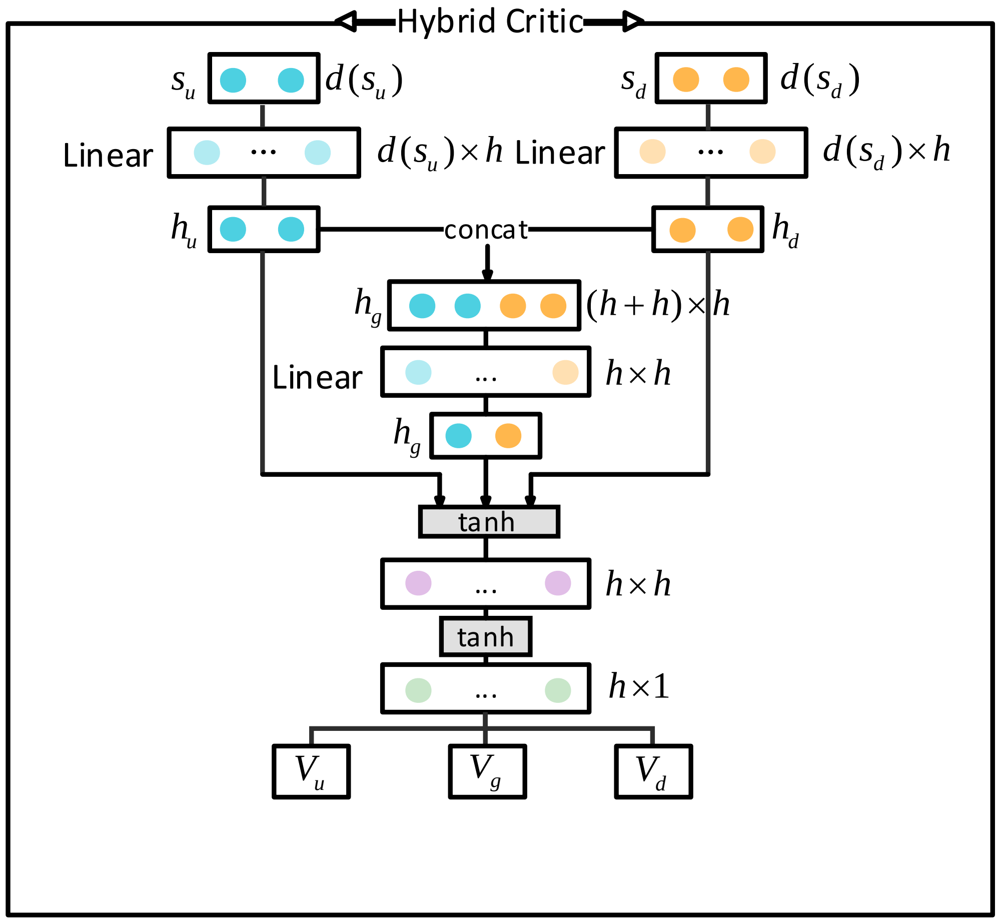
- The Critic update with the weighted losses of the three parts.

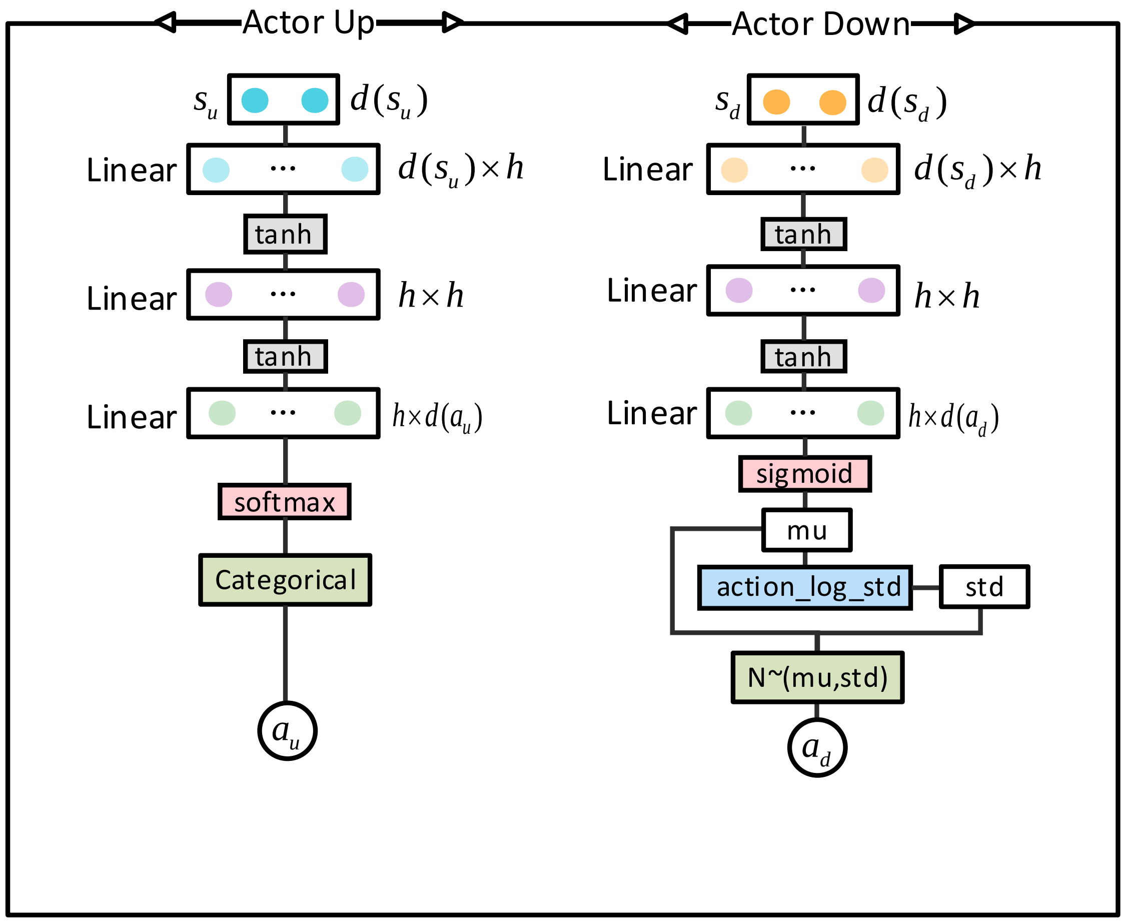
$$L^u(\phi) = (V_\phi(s_u^t) - A_u^t - \gamma V_{\phi'}(s_u^t))^2,$$
$$L^d(\phi) = (V_\phi(s_d^t) - A_d^t - \gamma V_{\phi'}(s_d^t))^2,$$
$$L^g(\phi) = (V_\phi(\{s_u^t; s_d^t\}) - A_g^t - \gamma V_{\phi'}(\{s_u^t; s_d^t\}))^2,$$
$$L(\phi) = w_1 \times L^u(\phi) + w_2 \times L^d(\phi) + w_3 \times L^g(\phi),$$

Three heads:

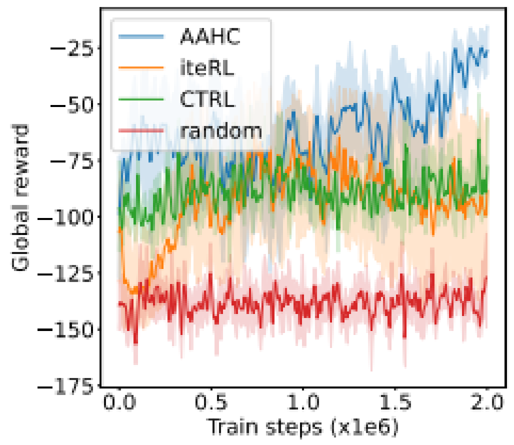
- input: UL state; output: value for the UL state
- input: DL state; output: value for the DL state
- input: Global state (concatenation of UL+DL); output: value for the Global state

Asynchronous Actor Hybrid Critic

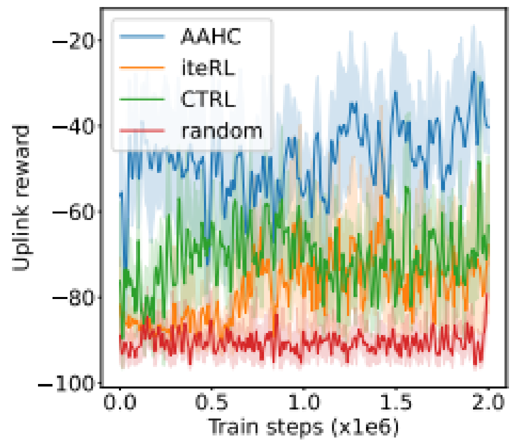




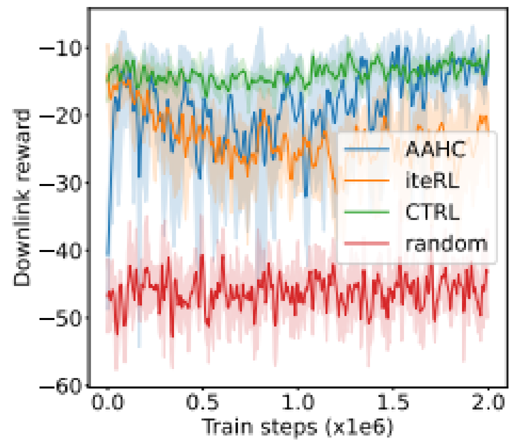
Asynchronous Actor Hybrid Critic



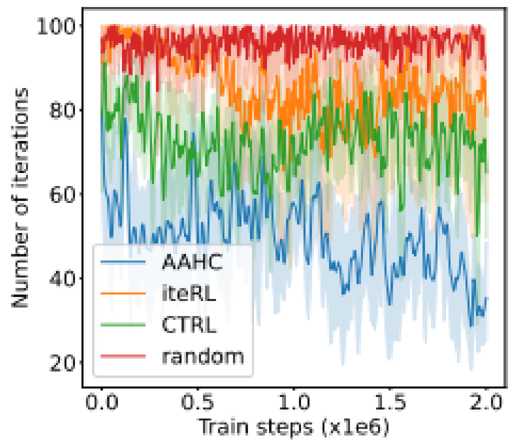
(a) 3-8 global reward.



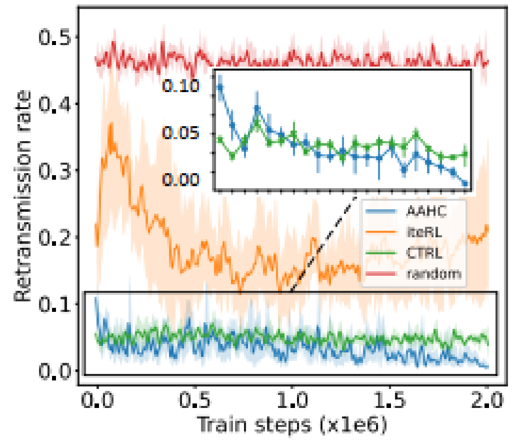
(b) 3-8 uplink reward.



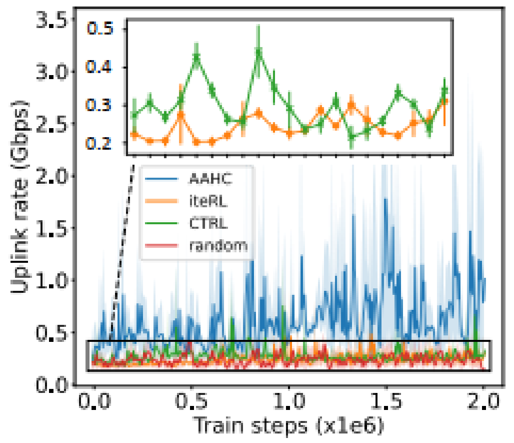
(c) 3-8 downlink reward.



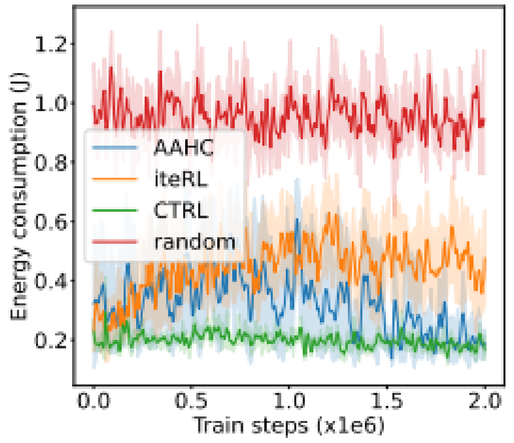
(d) 3-8 number of iterations.



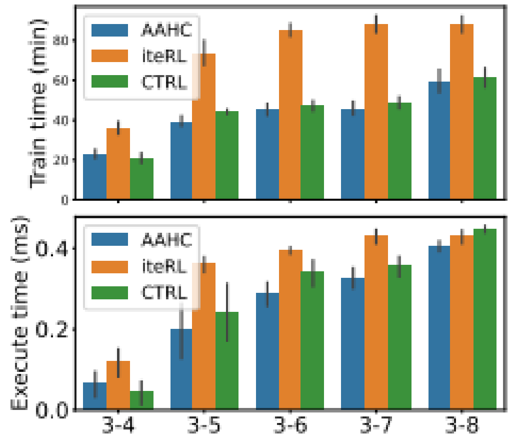
(e) 3-8 re-transmission rate.



(f) 3-8 uplink rate.



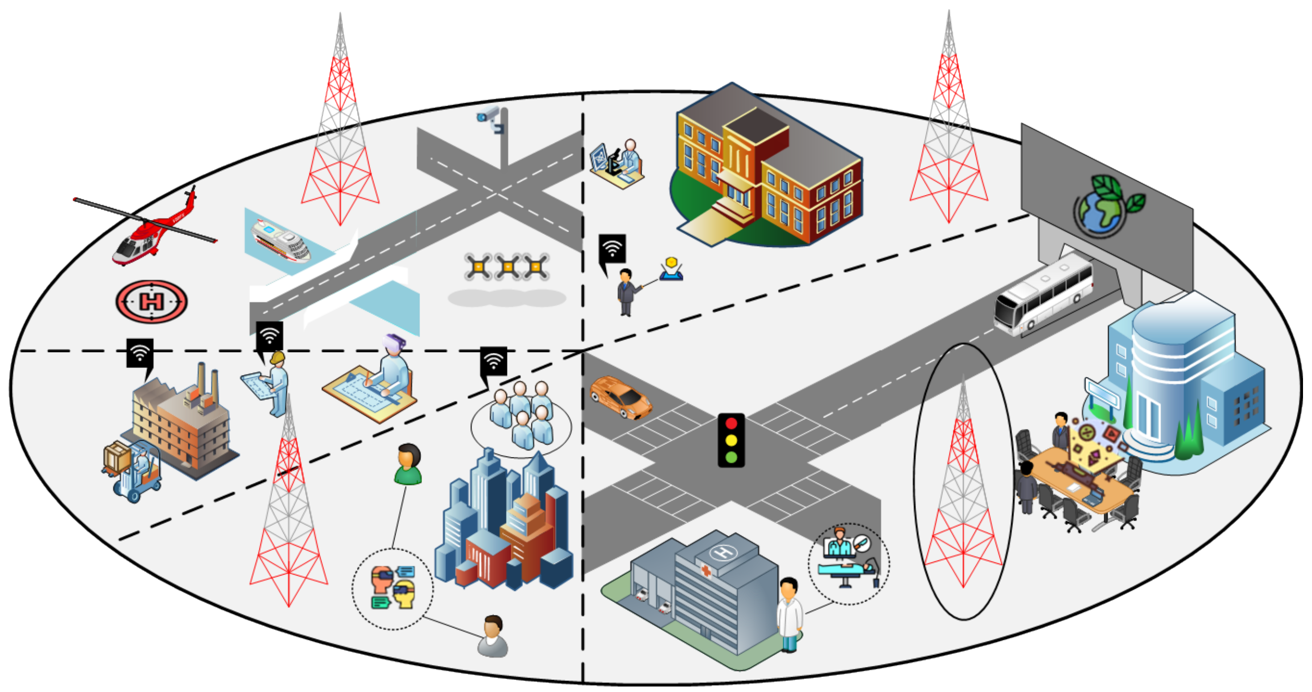
(g) 3-8 energy consumption.



(h) Train time & Execute time.

User-centric Metaverse

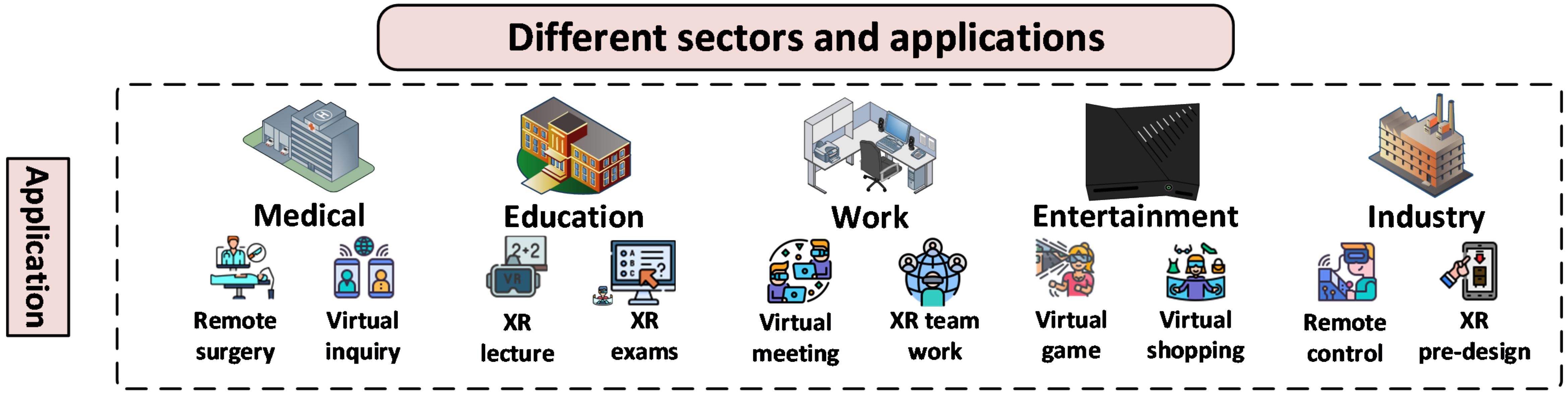
User-centric in Metaverse



- Metaverse is a user-centric application by design. As such, every component of the multiverse should place the human user at its core.
- Metaverse is a universe that every user can access freely at any time and any place.

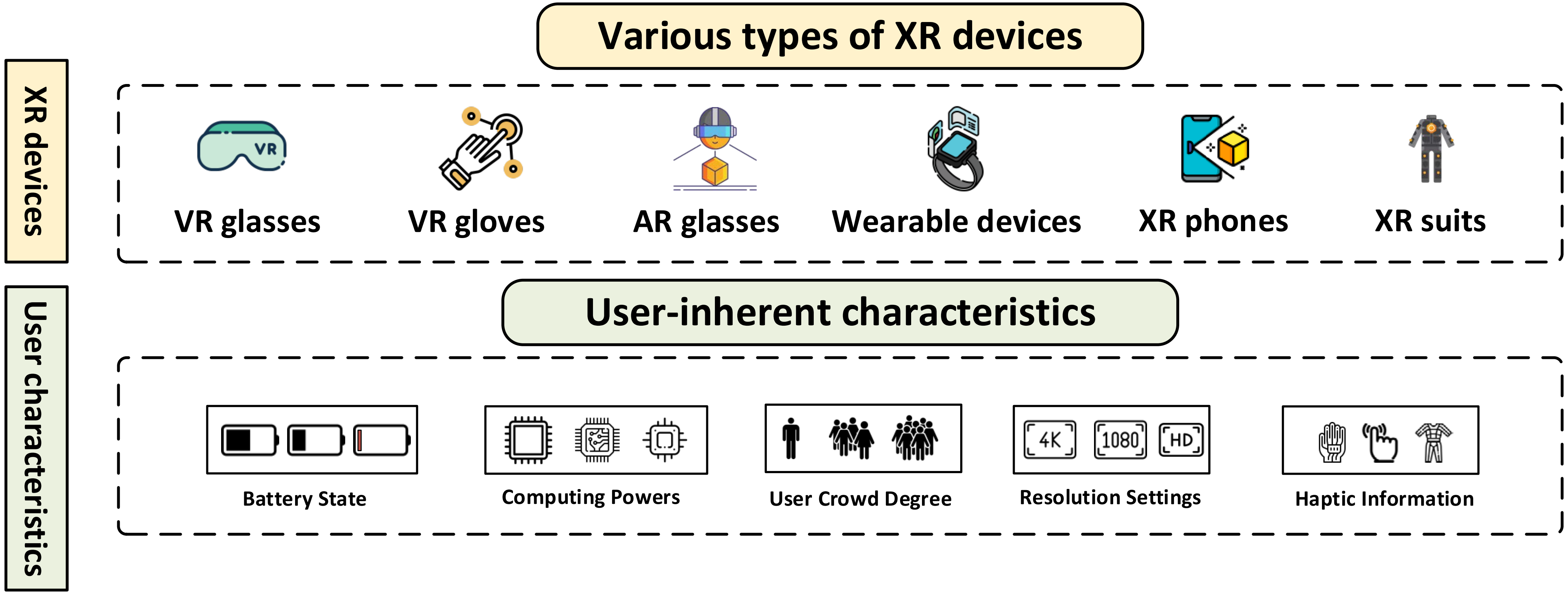
Reasons for User-centric

- Various applications across different sectors.
- Different levels of computation and communication demands.
- e.g., remote surgeries require extremely high frame rate, alongside with hyper-realistic and ultra-low latency feedback as it is a safety-sensitive application. While an XR lecture is not safety-sensitive with higher tolerance.



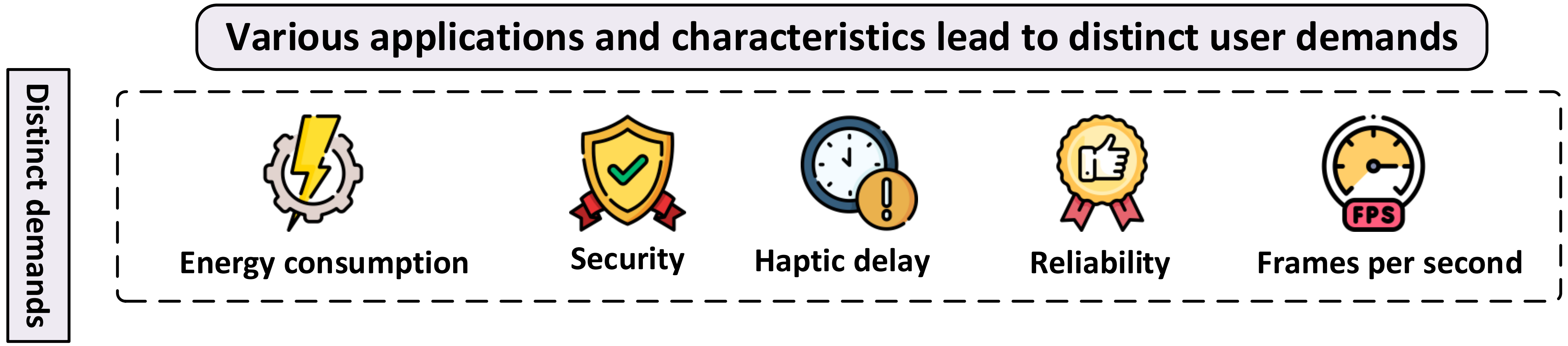
Reasons for User-centric

- Variations across XR device hardware specifications and user-inherent characteristics lead to different computation power and battery capacity.
- These variations are to be taken into account when designing user-centered optimization algorithms



Reasons for User-centric

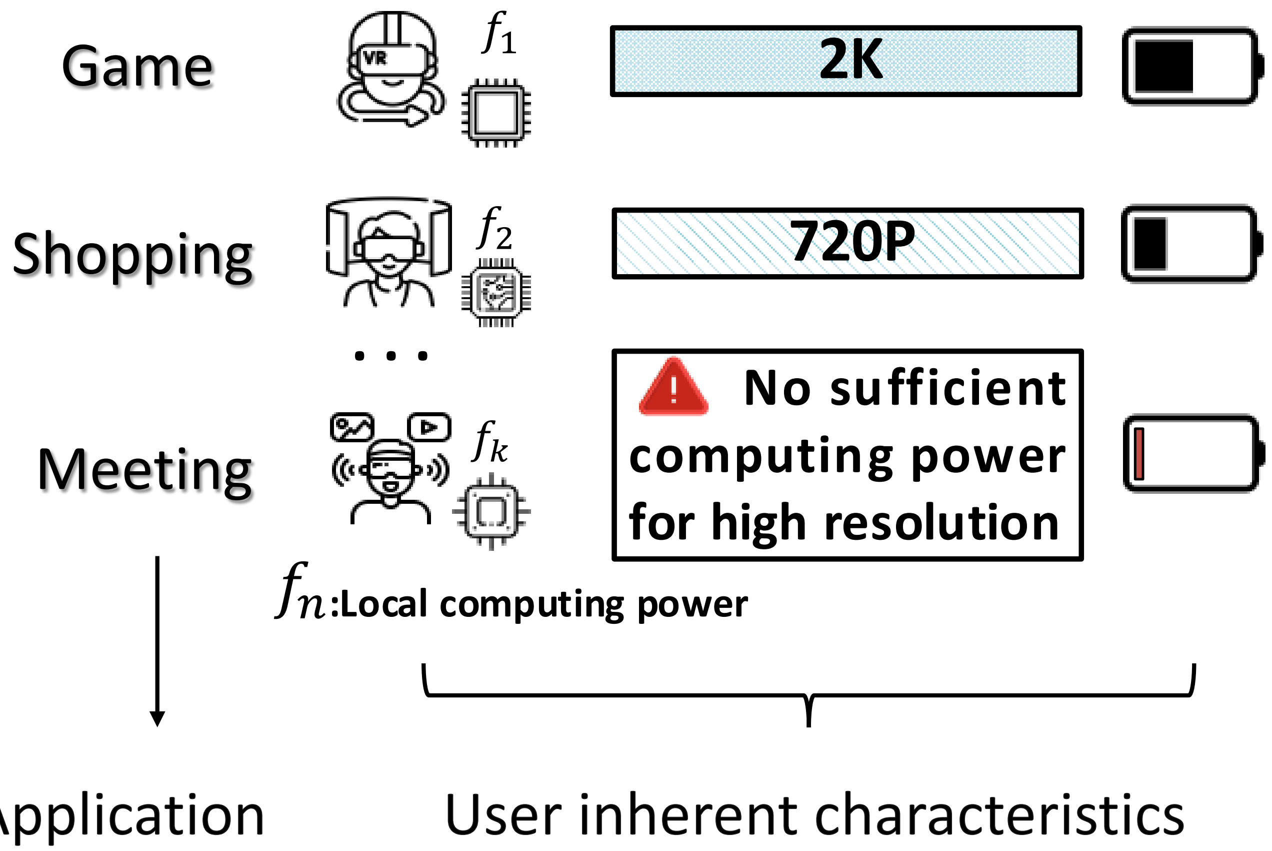
- There are distinct demands among different users.
- e.g., Consider a VR scenario, where user A is playing a game while user B is having a VR chat. Their required frames per second (FPS) are obviously different. For the game, the most comfortable FPS should be about 90 FPS, while a virtual meeting only needs about 40 FPS.



Problem arises

● The user-centric scenario arises a user-diverse problem. When allocating resources to different users, they are not equal. It requires us to develop a more user-centric solution.

e.g., Allocating remote computation resources to the following users.



Game
Shopping
Meeting
f_1
f_2
f_k
2K
720P
No sufficient computing power for high resolution
f_n:Local computing power
Application
User inherent characteristics

● The different users have various application scenarios, and user inherent characteristics.

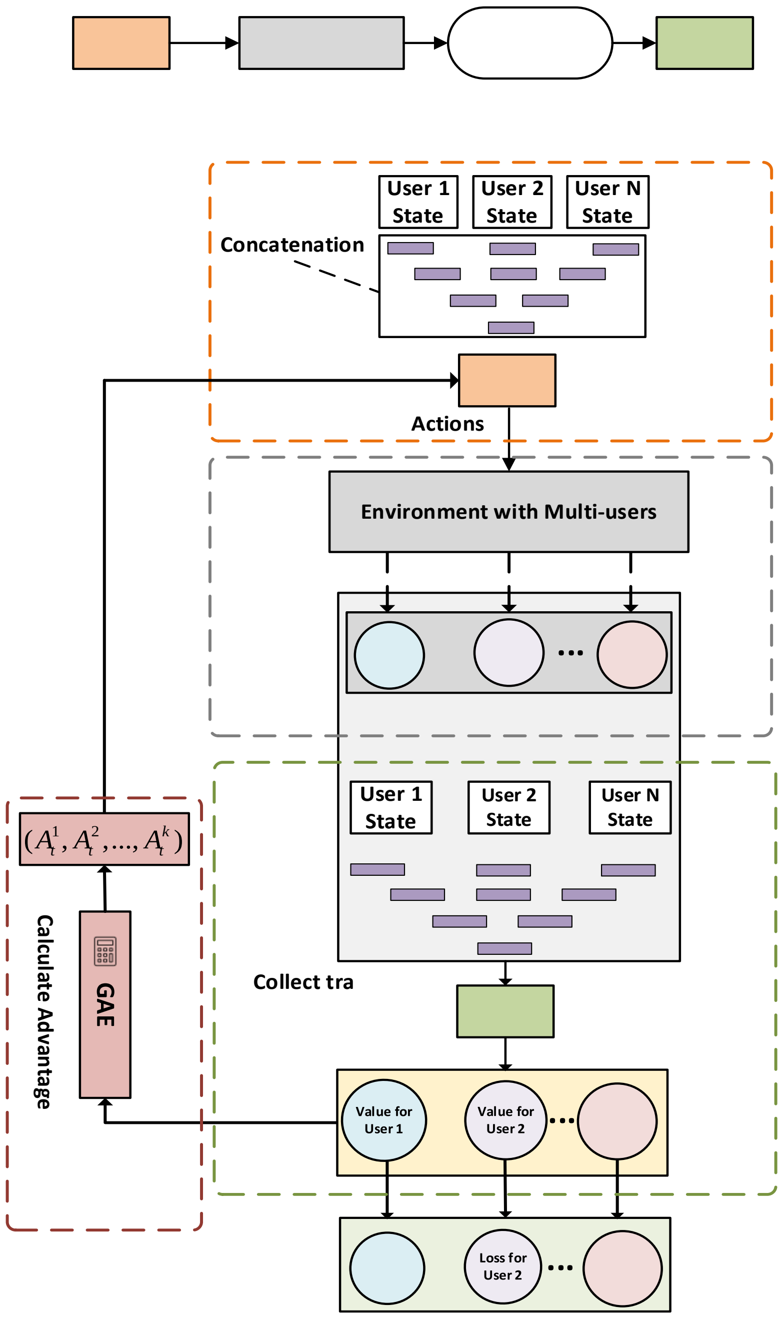
● Divide one second into T frames. How to allocate the remote computation resources in each frame is a worth studying problem in this scenario.

User-centric Reinforcement Learning

※ The key point of the user-centric structure is the Critic. It takes in the state and outputs multiple values for each user.

i.e., The Critic not only evaluates the overall value for all users but evaluates the value for each user, and then calculates the losses for each user, updating with sum losses.

User-centered RL Overview



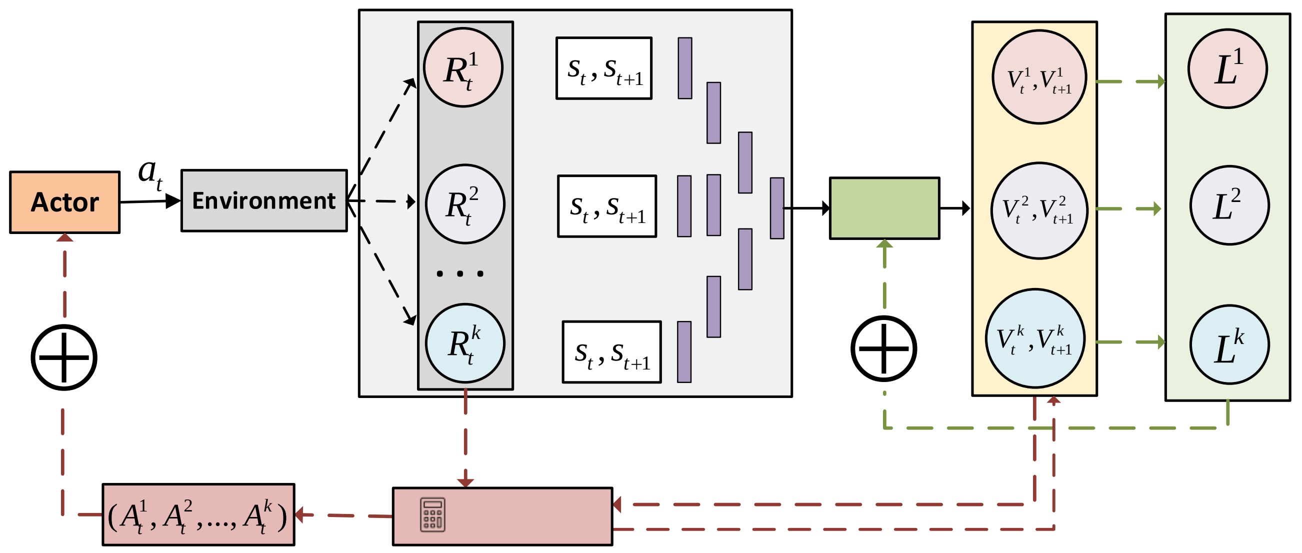
The **Actor** takes in the states of all users, then chooses actions for **all users**.

With the actions from all users, each user will get a **separate reward**.

The hybrid **Critic** evaluates the action under the state for **each user**, and then compute the losses for **each user.**

The rewards are used to calculate the advantages for **each user**, and then sent to the Actor for updating.

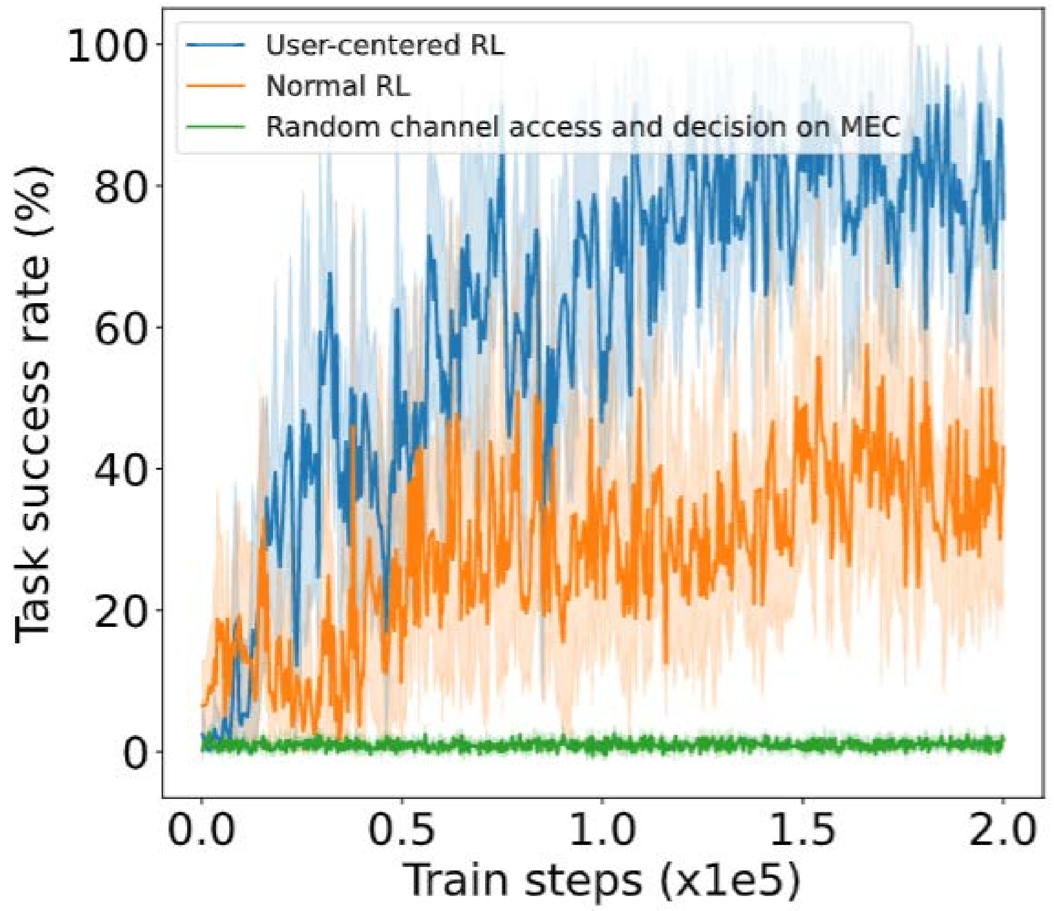
User-centric Reinforcement Learning



Actor: $$\Delta\theta = \mathbb{E}_{(s^t,a^t)\sim\pi_{\theta'}}[\nabla f^t(\theta, (\sum_{n=1}^{N} \boxed{A_n^t(s^t)})]. \xrightarrow{} $$ Sum all users' Advantages

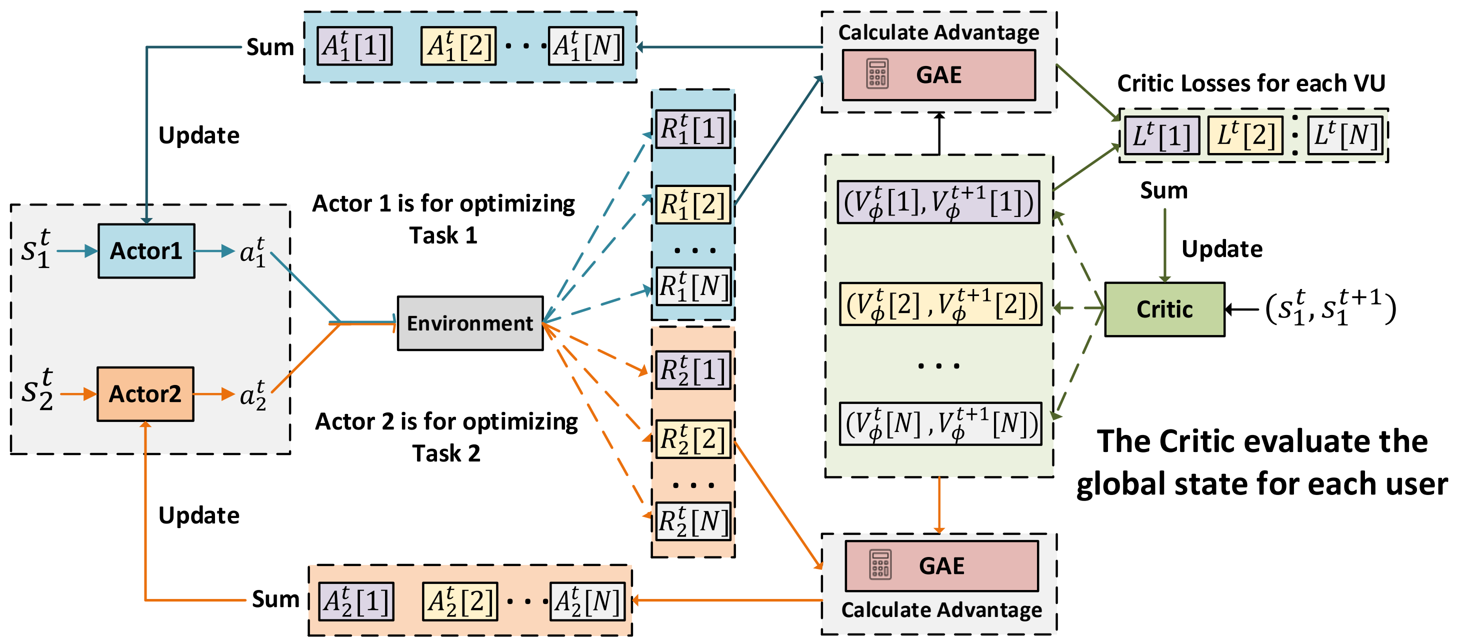
Critic: $$L(\phi) = \sum_{n=1}^{N} \left(V_{\phi,n}(s^t) - (A_n^t + V_{\phi',n}(s^t))\right)^2 .$$ Sum all users' Losses

User-centric Reinforcement Learning



(a) Performance of User-centered DRL.

Multi-task/agent User-centric RL structure


Sum
$A_1^t[1]$ $A_1^t[2]$ · · · $A_1^t[N]$
Calculate Advantage
GAE
Critic Losses for each VU
$L^t[1]$ $L^t[2]$ $L^t[N]$
Update
$R_1^t[1]$
$R_1^t[2]$
$R_1^t[N]$
Actor 1 is for optimizing Task 1
$(V_\phi^t[1], V_\phi^{t+1}[1])$
Sum
Update
s_1^t
Actor1
a_1^t
Environment
$(V_\phi^t[2], V_\phi^{t+1}[2])$
Critic
(s_1^t, s_1^{t+1})
s_2^t
Actor2
a_2^t
$R_2^t[1]$
$R_2^t[2]$
$R_2^t[N]$
$(V_\phi^t[N], V_\phi^{t+1}[N])$
Actor 2 is for optimizing Task 2
The Critic evaluate the global state for each user
Update
GAE
Calculate Advantage
Sum
$A_2^t[1]$ $A_2^t[2]$ · · · $A_2^t[N]$

- In the real-world scenario, we often need to optimize multiple tasks similar to that in the real-time 3D reconstruction scenario.
- For Agent with task 1, we use the sum of advantages calculated from task 1 process among all users, while the Agent 2 we use the sum advantages from task 2.