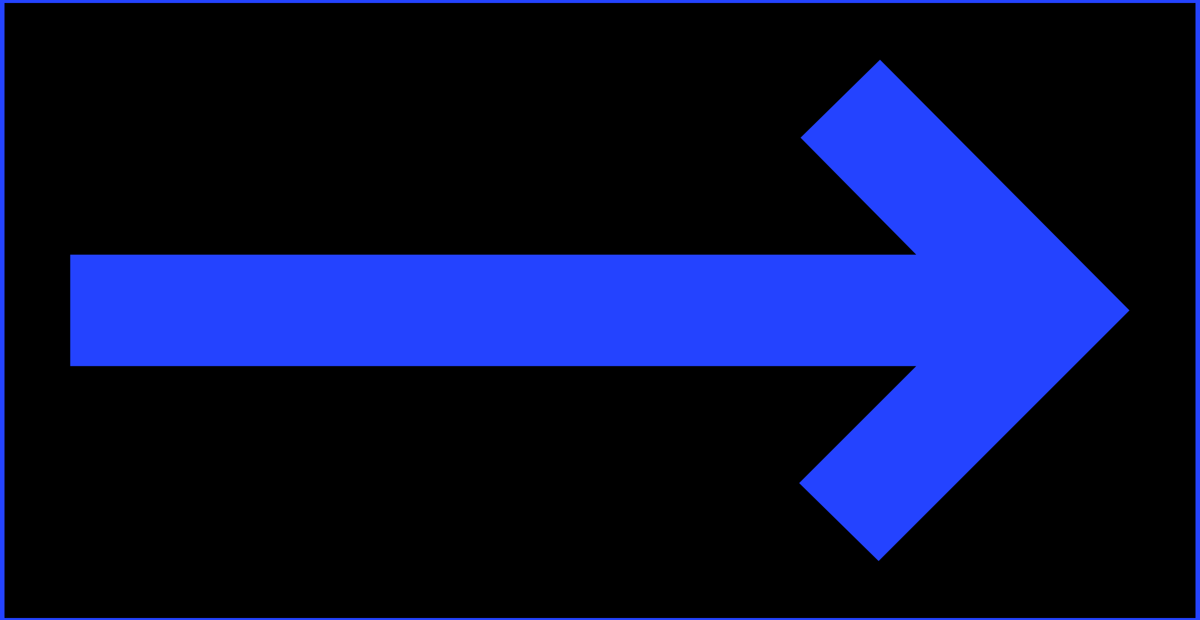
EdgeAIGC based multi-model ML using load balancing

Manas Viswajith - CB.SC.U4CSE23337
Nandagopal Menon - CB.SC.U4CSE23340
Nanditha - CB.SC.U4CSE23339
Jai Subiksha - CB.SC.U4CSE23327

Understanding the idea



The Rise of AIGC (Artificial Intelligence Generated Content)

- AIGC technologies like ChatGPT, DALL-E, and Sora are revolutionizing content creation.
- These models are powerful but have massive computational needs.

The Traditional Cloud-Based Problem

- Currently, most AIGC services run in centralized cloud data centers.
- This approach faces significant limitations:
  - **High Transmission Latency:** Sending requests and receiving generated content takes too long for real-time applications.
  - **High Service Costs:** Transmitting large amounts of data and renting cloud resources is expensive.

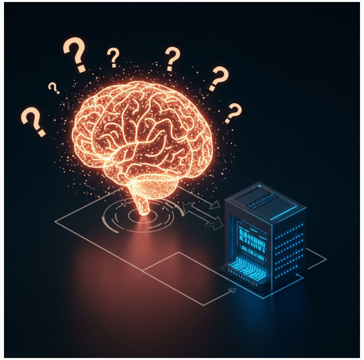
Our Goal: Can we bring AIGC closer to the user to solve these issues?

The Edge Computing Solution

- Edge computing processes data closer to the source, reducing latency and cost.
- By deploying AIGC models on edge servers, we can enable on-site service processing.

The Core Challenge

- AIGC models have a **large parameter scale** and **complex computing requirements**.
- Edge servers have **limited storage and computational resources**.
- **Balancing the load** between edge and cloud



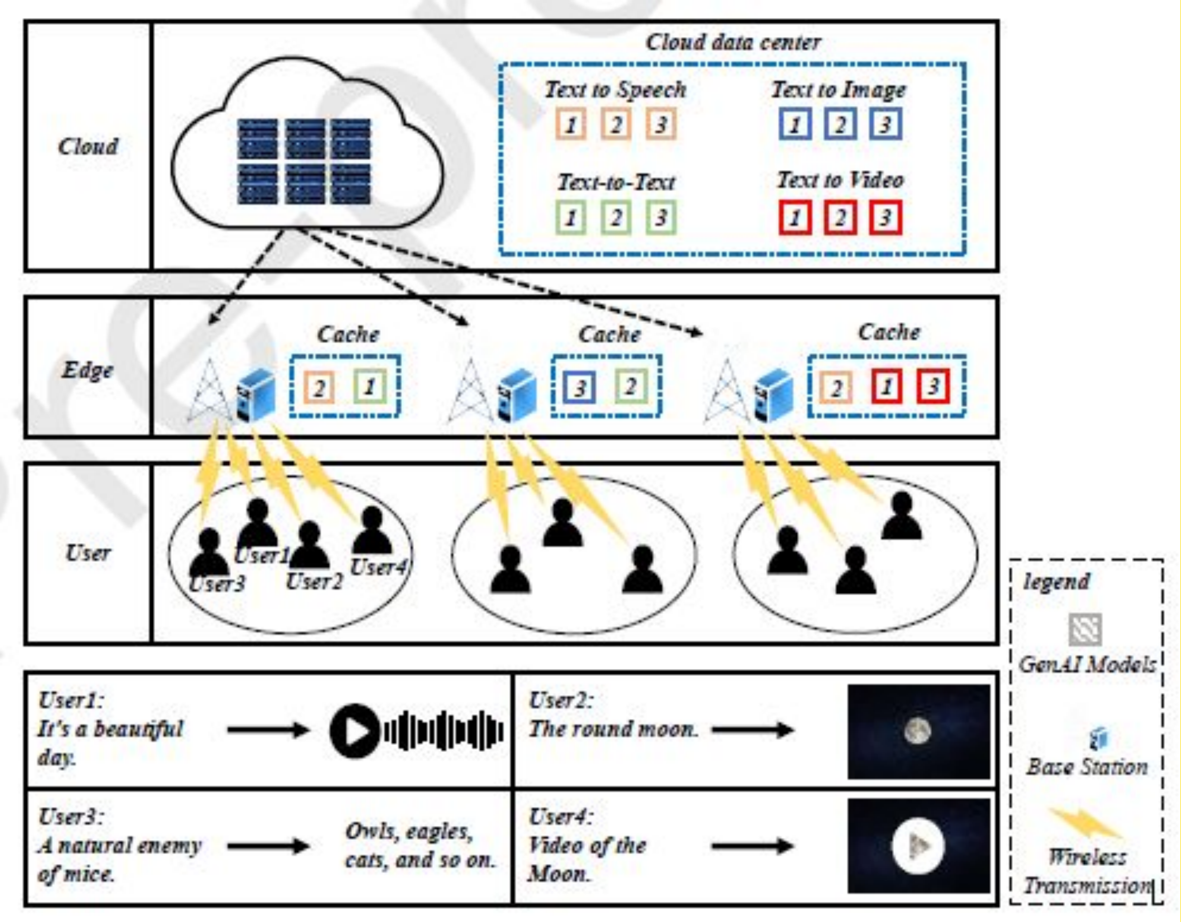
Use Case

What We Plan to Do: We will implement a smart traffic management system that uses the EdgeAIGC framework to process video feeds from roadside cameras in real-time.

The Models We Plan on Using: Our system will utilize a pipeline of three distinct machine learning models:

1. **Vehicle Detection Model**
2. **Vehicle Classification Model**
3. **Traffic Flow Analysis Model**

The Proposed EdgeAIGC Framework



We propose the **EdgeAIGC framework**, which consists of three layers:

- **User Layer:** Comprises multiple users, each generating AIGC service requests (e.g., "create an image of a cat").
- **Edge Service Layer:** A set of Edge Servers (ES) with limited storage and computing power. They can:
  1. Cache popular AIGC models for fast, local processing.
  2. Act as a relay, forwarding requests to the cloud if the model is not cached.
- **Cloud Service Center:** A powerful Cloud Server (CS) that stores all AIGC models and can handle any request, but at a higher latency and cost.

Implementation Strategy

AI- Powered Optimization

1. The Inputs (System State) The agent constantly observes the current network **State** to make informed decisions. This includes:

- The popularity trend of different AIGC models.
- Active user requests, including the model type and data sizes.
- Real-time network channel quality between users and edge servers.

2. The Decisions (System Action) Based on the state, the agent takes a specific **Action**. This is a comprehensive plan that defines:

- **Model Caching Decisions**: Which specific models to store on which edge servers.
- **Resource Allocation Plan**: The precise percentage of bandwidth and computing power each user is allocated

3. The Goal (System Reward) The agent learns by receiving a **Reward** for its actions. The goal is to maximize this reward, which is calculated based on minimizing the total user response time and overall service cost.

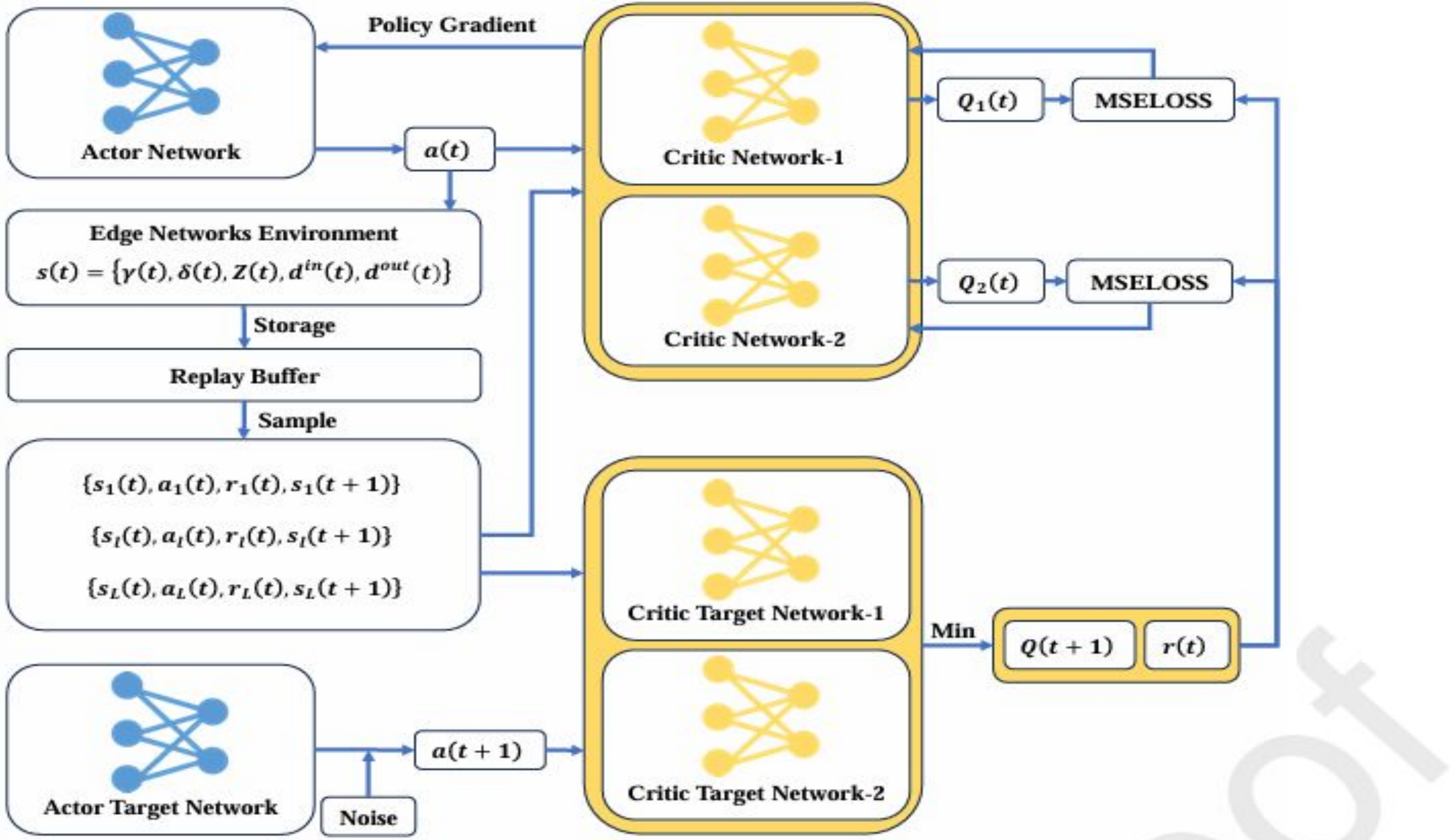
The Engine: The TD3 Algorithm

We use the **Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm** as our decision-making engine. It is specifically chosen for its stability and strength in handling continuous decisions, like allocating a precise percentage of resources. This intelligent strategy is how our framework achieves a model hit rate improvement of at least **41.06%** over baseline methods.

How we plan to implement?

We'll start by building a virtual model of the network using a simulator like iFogSim. This allows us to test and validate our TD3 algorithm in a controlled environment to prove its effectiveness.

As a future goal, we'll create a small-scale, real-world testbed using Raspberry Pi devices as edge servers. This will let us measure true performance and understand the practical challenges of the system.



Policy Gradient
Actor Network
$a(t)$
Critic Network-1
Critic Network-2
$Q_1(t)$
$Q_2(t)$
MSELOSS
MSELOSS
Edge Networks Environment
$s(t) = \{\gamma(t), \delta(t), Z(t), d^{in}(t), d^{out}(t)\}$
Storage
Replay Buffer
Sample
$\{s_1(t), a_1(t), r_1(t), s_1(t+1)\}$
$\{s_l(t), a_l(t), r_l(t), s_l(t+1)\}$
$\{s_L(t), a_L(t), r_L(t), s_L(t+1)\}$
Critic Target Network-1
Critic Target Network-2
Min
$Q(t+1)$
$r(t)$
Actor Target Network
$a(t+1)$
Noise

State-of-the-Art Literature

Wu Wen, Huang, Y., Zhao, X., et al. (2025). ***EdgeAIGC: Model caching and resource allocation for Edge Artificial Intelligence Generated Content*****, Digital Communications and Networks.**

Emergence of AIGC Models (2023–2024)

- Models like **ChatGPT, DALL·E, Sora**, and **Claude** have redefined content generation across text, images, and video.
- These models are **resource-heavy**—often requiring a lot of VRAM and multi-GPU setups.

Edge Computing for AIGC

- Recent studies have explored **deploying AIGC models on edge servers** to reduce latency.
- There were challenges faced with model size, heterogeneous edge capabilities and dynamic user demand.

Prior Research Directions

- **Edge Content Caching**: Based on content popularity or mobility (e.g., EICache, DMCPA)
- **Resource Allocation**: Focused on optimizing bandwidth or compute using MDPs, federated learning, or heuristics (e.g., JORA, PAES)

Key Contribution

- Proposes an **integrated framework** for edge-side AIGC that jointly handles:
  - Intelligent model caching decisions
  - Multi-user resource allocation (bandwidth and compute)
  - Adaptation to real-time model popularity and network dynamics

Technical Innovation

- Applies **TD3 (Twin Delayed Deep Deterministic Policy Gradient)** for optimization
  - Supports continuous action spaces (e.g., partial GPU/bandwidth allocation)
  - Learns from environment feedback to improve caching and load balancing policies over time

Experimental Results

- Achieves at least **41.06% improvement in model cache hit rate**
- Demonstrates reduced user response time and cost across dynamic AIGC workloads

Research Gap

- Most existing approaches handle **caching** and **resource allocation** in isolation
- Few works address them **jointly**, especially in real-time AIGC scenarios with dynamic system states

Task Allocation

Architectural Design

Jai Subiksha

Algorithm Design

Nandagopal Menon

Simulation Analysis

Nanditha

Documentation

Manas Viswajith

Q&A

Key points to read while you get the inquisitive thoughts running

- **AIGC models are powerful but resource-intensive**, making cloud-only approaches slow and expensive.
- **Edge computing** can solve this, but we face challenges with limited resources and model placement.
- Our **EdgeAIGC framework**, powered by **TD3**, jointly optimizes **model caching** and **resource allocation**.
- It intelligently learns from the environment to **minimize user wait time** and **reduce service cost**.