# Edge-Cloud Collaborative Streaming Video Analytics with Multi-agent Deep Reinforcement Learning

Bin Qian, Yubo Xuan, Di Wu, Zhenyu Wen*, Renyu Yang, Shibo He, Jiming Chen, and Rajiv Ranjan

***Abstract*—Streaming video analytics focuses on the real-time analysis of streaming video data from multiple resources, such as security cameras, and IoT devices with video capabilities. It involves applications of various techniques to extract valuable information from live video streams. Edge computing and cloud computing facilitate video stream analytics by utilizing computation resources across both ends, enabling both high accuracy and low latency. However, video streaming behaviours are dynamic and constantly evolving across the edge and the cloud. The network conditions, computing resources, and video content can change rapidly, making it crucial to continuously adjust the analytics methods to provide accurate results. Previous works both based on deep neural networks (DNNs) or heuristic algorithms learn a suitable deployment plan for streaming video analytics applications from historical data or synthetic data and therefore are not able to capture the dynamics. Hence, we propose reinforcement learning-based methods that can adapt to ongoing changes in video streaming behaviours. To ensure the scalability of video analytics in distributed environments, we implement OSMOTICGATE2, a distributed streaming video analytics system that features optimized processing pipelines and multi-agent RL-based controllers for fast adapting the system configurations across the edge and the cloud. Experiments on a real testbed show that our method outperforms baselines, assuring real-time video analysis and high accuracy in dynamic and distributed environments.**

***Index Terms*—real-time video analytics, distributed system, multiagent deep reinforcement learning, edge cloud collaboration.**

## I. INTRODUCTION

VIDEO analytic is of utmost importance in a range of computer vision applications, including video surveillance [1], augmented reality, and autonomous driving. Presently, Deep Neural Networks (DNNs) serve as the foundational technology for cutting-edge video analytic algorithms, ensuring exceptional precision for the end users. Nevertheless, the deployment of DNN models for video analytics in real-world settings presents numerous obstacles. These models are highly computationally demanding, featuring hundreds of layers, which leads to significant inference latency. Moreover, the sheer magnitude of streaming video data gives rise to apprehensions regarding the transmission of raw data to the cloud for inference due to the exorbitant costs associated with bandwidth and the ensuing insufferable delays in transmission.

One promising means to tackle bandwidth expenses and data transmission delays in video analytic applications is deploying DNN models on edge nodes situated in close proximity to users. These edge nodes can swiftly receive streaming videos from such end devices as cameras or mobile phones with minimized delays. However, they inevitably fall short in processing capabilities and become overwhelmed especially during peak times with non-negligible delays. Hence, video analytics systems imperatively require elaborate considerations of both edge and cloud resources for optimal performance guarantee in overloaded situations.

There are many existing studies on video analytics pipelines in the continuum of edge and cloud computing. For instance, $A^2$ [2] and EdgeAdaptor [3] delved into the choice of distinct DNN models to optimize delay and accuracy. DeepDecision [4], FastVA [5], and Osmoticgate [6] investigated new offloading mechanisms between the edge and the cloud to fine-tune video analytics setups. They primarily take into account video preprocessing and model selection on the edge side to strike a balance between inference accuracy and delay. Reducto [7], and ClodSeg [8] employed frame filtering and resolution downsizing techniques to minimize communication costs during video transmission while preserving accuracy. However, these approaches can hardly adapt to dynamic environments where the fluctuation of available computing and communication resources is the norm rather than the exception across distributed edge nodes.

When it comes to optimizing video analytics pipelines in a highly distributed and dynamic environment, two main challenges remain unsettled: i) It is impractical to scale up the system due to the large configuration space in distributed systems. System states and configurations will drastically increase with the increment of the device number, and it is time-consuming for a centralized controller to search for an optimal configuration that maximizes the system resource usage across the edge and the cloud. For example, Chicago police analyze 30,000 camera streams in real time [9], and making decisions in a centralized manner is impossible. Thus the decentralized controlling system is imperative in such a distributed system. ii) Achieving a self-configuring plan for both cloud and edge nodes is challenging. In an edge-cloud setting, every agent, whether cloud or edge, strives to optimize their resource usage, which can impact the entire system's performance. For instance, if the network bandwidth

Bin Qian and Rajiv Ranjan are with School of Computing, Newcastle University, NE4 5TG, UK; Zhenyu Wen* (corresponding author), Yubo Xuan are with Zhejiang University of Technology, 310023, China; Di Wu is with University of St Andrews, KY16 9AJ, UK; Renyu Yang is with School of Software, Beihang University, 100191, China; Shibo He is with Zhejiang University, 310058, China; Jiming Chen is with Zhejiang University and Hangzhou Dianzi University, 310018, China.

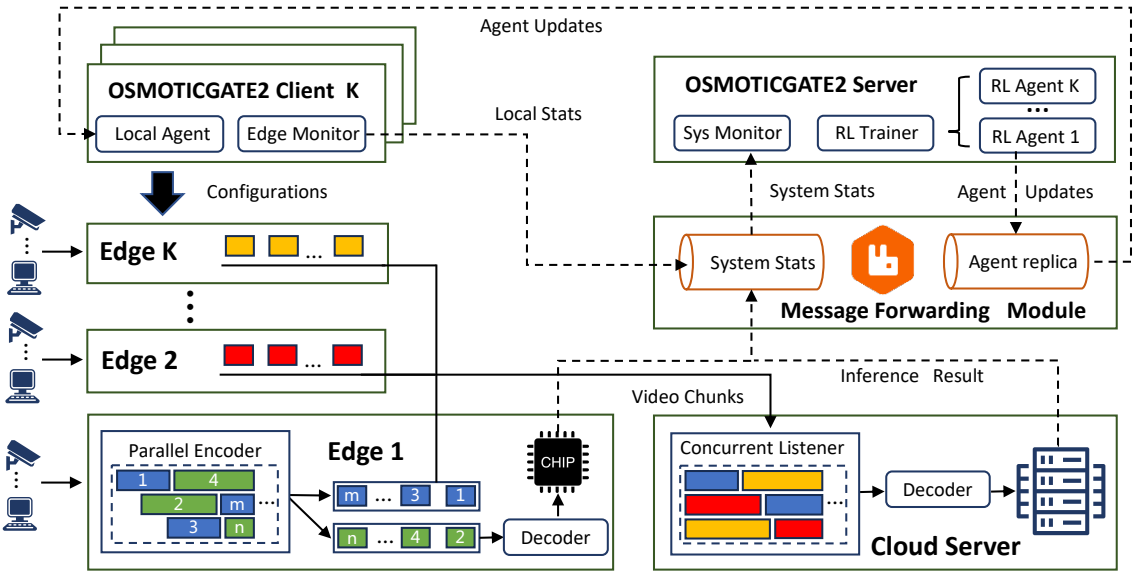



Fig. 1: RL-based Edge-Cloud Collaborative Video Analytics in OSMOTICGATE2. The streaming videos are encoded in the edge nodes and then processed on both the edge and the cloud. Our OSMOTICGATE2 agents control the edge behaviors with various configurations. The two modules are communicated via a message forwarding module across the edge and the cloud.

is sufficient, each edge node may be more inclined to offload more data to the cloud, which may saturate the cloud and reduce the performance of the entire system.

To address the aforementioned challenges, we present OSMOTICGATE2, a distributed orchestration system for optimizing video analytics across the edge and the cloud. In our system, multiple edge devices work collaboratively and choose the appropriate configurations to maximize system Quality of Service (QoS) performance. To improve the system scalability, we deploy on each edge node a controller for generating video analytics configurations, based on local status only. Additionally, we have developed a mechanism grounded in multi-agent reinforcement learning, which utilizes the Centralized Training and Decentralized Execution (CTDE) approach. This strategy allows all agents to engage in collaborative learning within the OSMOTICGATE2 server. The resulting OSMOTICGATE2 offers notable advantages, including exceptional scalability, adaptability, and efficiency, all achieved through a carefully optimized system-algorithm co-design.

Experiments on a real testbed with edge-cloud collaboration validate the system's effectiveness in enabling real-time and accurate video analytics. The paper's main contributions are summarized as follows:

- Design OSMOTICGATE2, an edge-cloud collaborative video analytics system in which the edge nodes and the cloud server can collaborate for video encoding and video analytics.
- Design an online multi-agent reinforcement learning algorithm (MAPPO) for orchestrating the system configurations within the systems. The agents of multiple edge nodes collaboratively learn the optimal policy by sharing their information to maximize their respective rewards.
- We evaluate the performance the proposed algorithm with real-world datasets and testbeds. Our method can achieve the best performance in terms of rewards, inference accuracy, as well as the target system latency.

## II. SYSTEM OVERVIEW

An overview of the proposed streaming video analytics architecture OSMOTICGATE2 is depicted in Fig. 1, where a large number of edge devices work collaboratively with the cloud server to perform video analytics jobs on streaming video generated from end devices. The streaming video is first encoded into small chunks and then processed locally on edge devices, with all results aggregated on the cloud server in the end. When the edge device is unable to process all the video streams in real time, a proportion of the video chunks are transmitted to the cloud for processing as well.

The extra video transmission overhead for cloud computing raises challenges on balancing the communication and computation resources in edge-cloud collaborative analytics. As compared to edge computing, cloud computing incurs extra communication overhead but benefits lower processing latency. The balance between the communication and computation is further complicated when the system scales with more edge devices in the system. Thus, an adaptive mechanism that learns the system dynamics is necessary in consideration of the system scalability. OSMOTICGATE2 develops a novel mechanism capable of adaptively generating system configurations to improve the system QoS metrics, i.e., accuracy and latency. It is composed of three major components: the *video analytics modules*, the *multi-agent controllers* as well as the *message-forwarding modules*. The three components are modularized and communicated via the *message-forwarding modules*, featuring seamless "monitoring - updating" across the edge-cloud computing paradigm, enabling the smooth running of the OSMOTICGATE2. In general, OSMOTICGATE2 has three main design considerations:

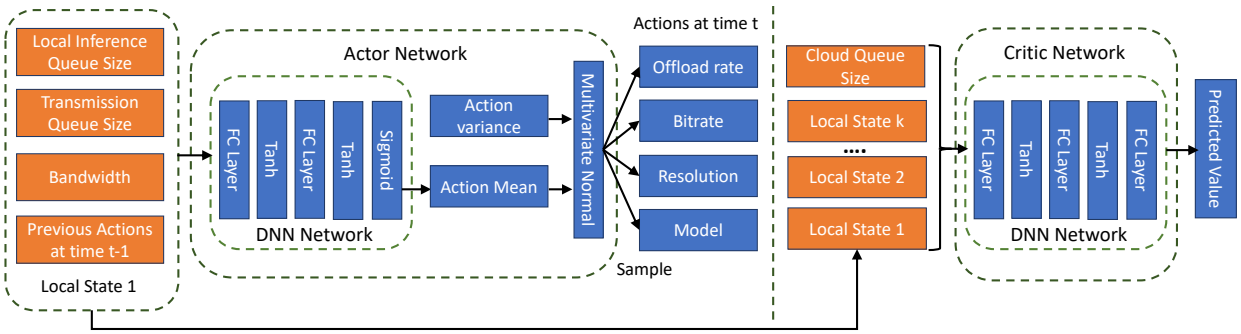



Fig. 2: RL Agent architecture

TABLE I: Notation

| Notation | Description |
|---|---|
| $t$ | Current time step $t$. |
| $K$ | The total number of edge devices. |
| $E_k$ | Edge device $k$, $k \in$K. |
| $S$ | The cloud server. |
| $r_k^t$ | Video resolution of edge device $k$ at time step $t$. |
| $b_k^t$ | Video bitrate of edge device $k$ at time step $t$. |
| $m_k^t$ | Inference model candidate of edge device $k$ at time step $t$. |
| $\mu_k^t$ | The offloading ratio for edge device $k$ at time step $t$. |
| $QI_k^t$ | The size of local inference queue for edge device $k$ at time step $t$. |
| $QT_k^t$ | The size of transmission queue for offloading at time step $t$. |
| $Q_S^t$ | The size of inference queue for server $s$ at time step $t$. |
| $B_k^t$ | The network bandwidth of edge device $k$ at time step $t$. |
| $A_k^t$ | The action for edge device $k$ at time step $t$. |
| $acc_k^t$ | The average accuracy from edge device $k$ at time steps $t$. |
| $l_k^t$ | The inference latency of edge device $k$ at time steps $t$. |
| $l_k^{target}$ | The latency target of edge device $k$. |
| $O_k^t$ | The local state of edge device $k$ at time step $t$. |
| $G^t$ | The global state of the server $S$ at time step $t$. |
| $R_k^t$ | The reward of edge device $k$ at time step $t$. |
| $l^{thres}$ | The latency threshold in reward function |
| $F$ | The reward penalty when latency exceeds threshold $l^{thres}$ |
| $M$ | Maximum training episode |
| $T$ | Maximum steps in one episode |

**Adaptability:** We design multi-agent RL-based controllers for generating configurations and updating controllers based on local and global system status.

**Scalability:** We implement centralized training and decentralized execution (CTDE) strategies for the controllers, alleviating the problems of generating configurations from high-dimension search spaces, especially when there are many edge devices in the system.

**Efficiency:** We implement on-device acceleration techniques on both computation and communication for video processing across the edge and the cloud server.

## III. Multi-agent RL-based Controllers

We consider a classic real-time video analytics application, as depicted in Fig. 1. This application comprises multiple edge devices denoted as $E_k$, for $k \in \{1, 2, ..., K\}$, and a cloud server referred to as $S$. Each $E_k$ is responsible for receiving video streams and performing video analytics jobs in real time. Before the encoded video streams are input into the deep learning models for video analytics, they are placed in task queues, where they await decoding into image frames at both endpoints. Then an appropriate model is selected to meet the quality of service (QoS) requirements specific to the application. All the outcomes generated are then consolidated and sent to the cloud monitor, alongside other system status information.

For N chunks of video streams generated from $E_k$, the overall average processing latency $l_k^t$ averages over two components: 1). *Edge processing latency* edge processing pipeline includes four components: the video encoding latency, the waiting time in the queue, the video decoding latency, and the inference latency. 2). *Cloud processing latency* cloud processing places extra system overhead. During cloud inference, except aforementioned latency, one extra latency comes from the transmission latency between the edge and the cloud. The notation table is shown in Table I.

### A. Optimization Objective

Throughout the whole $T$ running time within the system, we consider two optimization objectives: the average system processing latency and the average inference accuracy. Assume the average inference accuracy during one time interval $\Delta t$ is $acc_k^t$, our optimization objective is formulated as follows:

$$\begin{aligned} \underset{A_k^t}{\text{maximize}} \quad & \frac{1}{T}\sum_{t=1}^{T} acc_k^t \\ \text{subject to} \quad & A_k^t = \{r_k^t, b_k^t, m_k^t, \mu_k^t\}; \\ & |l_k^t - l_k^{target}| \leq \epsilon, \forall t \in T \end{aligned} \tag{1}$$

For each $E_k$, we aim to maximize the average inference accuracy during the system running time $T$, while ensuring the average processing latency is close to a preset latency threshold $l_k^{target}$. We set a slack variable $\epsilon$ such that the small latency variance $|l_k^t - l_k^{target}|$ is acceptable during the application production. Subject to different QoS metrics of the real-time stream processing systems, $l_k^{target}$ can be variably adjusted as well for each edge device. The generated configurations for video processing include the video bitrate $b_k^t$, frame resolution $r_k^t$, and the choice of model $m_k^t$ used for video analytics. Additionally, the smart agent must make decisions about the offloading rate $\mu_k^t$, representing the proportion of stream data to be transferred from device $k$ to the server $S$.

The key problem for the RL agent lies in figuring out the optimal value, denoted as $A_k^t$, for $E_k$ at every time step, with the goal of enhancing the precision of system predictions. The difficulty lies in factoring in dynamic system limitations, like computational resources and network capacity, while being

contextually aware. The objective is to strike the right balance between ensuring stable system response times and achieving the highest level of accuracy by making informed choices regarding video analytics configurations.

Based on the system model and the optimization objective as represented in Equation 1, we elaborate on the intricate blueprints of the fundamental elements in the reinforcement learning (RL) agent, i.e., the architecture of the RL agent, the RL state, the RL action, the reward function. In addition, we also demonstrate how to train the RL agent in the system.

### B. Architecture of RL agents

**Actor-critic framework:** Figure. 2 illustrates the architecture of RL agents in our system. We adopt an architecture of actor-critic paradigm where multiple actor networks are deployed on each edge device $E_k$ for outputting the optimal policy. Following the design in [10], each edge device $k$ has its own pair of actor and critic network on the central server $S$. The advantage of this design is that each edge device $E_k$ can focus on optimizing its own policy while sharing the learning from local trails through a critic network on the cloud. We use the Proximal Policy Optimization (PPO) [11] as the actor-critic RL algorithm in this paper and denote the architecture of PPO used as the multiple agents PPO (MAPPO).

The actor network $\pi$ is responsible for learning the agent's policy. It maps agent observations $O$ to the mean and standard deviation of a Multivariate Gaussian Distribution, from which an action is sampled, in continuous action spaces. The actor network maximizes the expected cumulative reward over time by adjusting its network parameters in a way that leads to better actions in various states, ultimately improving the agent's performance. In this work, the input of the actor network is the local state of each $E_k$, and the output is probability distributions of 4 actions. They are later cast to the system configurations in the environment. The architecture of the actor network includes a 2-layer DNN network with a sigmoid activation function and a hyper-parameter action variance.

We sample the actions via a multivariate normal distribution with the action mean generated from the DNN network and the action variance. There have been many techniques for enabling efficient model exploration during training, as reflected in [12]. In this paper, we employ a parameterized exploration technique for action sampling modules in agent actors. Specifically, the distribution variance exponentially decreases along with different training episodes, i.e., standard deviation $std = 0.5 * 0.96^{episode}$. As compared to exploration techniques such as extra entropy loss in optimization objective, the benefit of such mechanism is enabling fast training of the algorithm, which is especially important in real-world systems.

The critic network, on the other hand, estimates the value of being in a particular state or taking a particular action in a given state. It takes states and actions as input and outputs a value, which represents the expected cumulative reward that can be obtained from that state-action pair. By comparing the estimated values with the actual rewards, the agent can update its policy network to maximize the expected rewards. In this work, the input of the critic network is an embedding layer that concatenates all local states and the cloud queue size. After performing feedforwarding with a two-layer DNN network, the output of the critic network is the predicted value given the global state.

### C. RL States and Actions

The system state reflects the working status within the system at every time step $t$. Concretely, we distinguish two states, i.e., the local state ($O_k^t$) on each edge device $E_k$ and the global state ($G^t$) on the server $S$. On the edge side, each device $E_k$ takes the local state $O_k^t$ as the input at time step $t$ to output the optimal action $A_k^t$. The global state $G^t$ is used as the input of the global critic network at time step $t$.

The local state $O_k^t$ for each $E_k$ includes the size of the transmission queue $QT_k^t$, the size of the local inference queue $QI_k^t$, the average network bandwidth $B_k^t$ between $E_k$ and the cloud $S$, and the agent's action at the previous time step $t-1$. At time slot $t$, we denote the local state of each agent $k$ as $O_k^t = \{QI_k^t, QT_k^t, B_k^t, A_k^t\}$

On the cloud server $S$, the global state $G^t$ includes all local states and the size of cloud queue $Q_S^t$ from all edge devices. The global critic network takes the global state and estimates the value of state-action value function, i.e., the expected cumulative reward an RL agent can achieve following its policy. The global state $G^t$ at time step $t$ is defined as $G^t = \{Q_S^t, O_1^t, O_2^t, ..., O_k^t\}$. where $k$ is the number of all edge devices.

The RL action generated by each edge device $k$ should consider the critical decisions with the system model, i.e., the resolution of generated images $r_k^t$, the bitrate for encoding and decoding the video stream $b_k^t$, the candidates of different models $m_k^t$, and the offloading ratio of the generated images $\mu_k^t$. The RL agent action for each edge device $k$ at time step $t$ can be expressed as $A_k^t = \{r_k^t, b_k^t, m_k^t, \mu_k^t\}$. Specifically, we design three decision variables $r_k^t$, $b_k^t$, and $m_k^t$ as discrete variables. The resolution choices $r_k^t$ are three-folds: 240P, 360P, and 540P. The video bitrate $b_k^t$ has three choices as well: 500kbps, 1000kbps, and 3000kbps. The model choice on the edge side $m_k^t$ includes three model variants of YOLOv8, named YOLOv8-m, YOLOv8-s, YOLOv8-n. While on the cloud side, we use YOLOv8-m only, with the best accuracy and largeset inference latency as well. The data offloading ratio $\mu_k^t$ is designed as a continuous variable from [0,1] to concisely control the proportion of data transferred to the cloud. During system implementation, $\mu_k^t$ portions of the video chunks are offloaded to the cloud while the rest are processed locally. The combination of the chosen actions can cover a wide range of accuracy and latency requirements that are suited for different types of QoS requirements, i.e., highest accuracy or moderated accuracy with acceptable system latency.

### D. Reward Function

The reward function guides the optimization direction and is the core of the RL agent. Based on our system optimization

**Algorithm 1:** RL agent training in OSMOTICGATE2

**Input:** the number of the edge device $K$, maximum episode M, maximum time step T
**Output:** the actor of each device k, $[\pi_k, \forall k \in K]$
**for** *each episode* $m \in M$ **do**
  $\tau_1, ..., \tau_K = []$ empty list
  **for** *each device* $E_k, k \in K$ *in parallel* **do**
    Initialize parameters $\theta_k$ for the actor $\pi_k$, and $\phi_k$ for the critic $V_k$
    **for** *each time step* $t \in T$ **do**
      1. observe the agent's local state $O_k^t$ and global state $G_k^t$
      2. Get actions $A_k^t = \pi_k(O_k^t; \theta_k^t)$
      3. execute the action $A_k(t)$ in environment
      4. observe the agent's local state $O_k^{t+1}$ and global state $G_k^{t+1}$
      5. get the reward with $R_k^t$ with Eq. 2
      6. $\tau_k += [A_k^t, O_k^t, G_k^t, R_k^t, O_k^{t+1}, G_k^{t+1}]$
    **end**
    Update $\theta_k$ via Adam
    Update $\phi_k$ via Adam
  **end**
**end**
**return** *RL agents*

goal in Equation 1, for each time step $t$, we formulate the reward $R_k^t$ as follows:

$$R_k^t = \begin{cases} acc_k^t - \omega * |l_k^t - l_k^{target}| & l_k^t \leq l^{thres} \\ -\omega * F - (l_k^t - l^{thres}) & \text{otherwise} \end{cases} \quad (2)$$

where $acc_k^t$ is the average accuracy at time step $t$, and $\omega * |l_k^t - l_k^{target}|$ is the latency constraint with a positive $\omega$ balancing the weights between the accuracy and latency. When the latency $l_k^t$ exceeds the pre-defined threshold $l^{thres}$, we place another linear penalty term $-\omega * F - (l_k^t - l^{thres})$, where with $F$ a positive number to penalize the action that leads to high latency. $\omega$ weighting term is included to ensure that the rewards under normal conditions are not overshadowed. By doing so, the agent is motivated to complete the task as close to the desired time as possible while maximizing accuracy.

### *E. Centralized Training and Decentralized Execution (CTDE) in* OSMOTICGATE2

In our paper, the number of edge devices, the number of bitrates, resolutions, model choices and the offloading rates all could increase the complexity of the decision-making. In MAPPO, we have designed centralized training and decentralized deployment strategy for alleviating the scalability problems. Specifically, by using centralized training, the agents can learn and coordinate their actions based on a global view of the environment. This allows them to capture complex interactions and dependencies more effectively during the learning process. In our paper, for each configuration we select three values that could reveal performance variations across them. Increasing the number of configurations will not raise scalability issues. Since the agents operate in a decentralized manner, making decisions based on their local observations and learned policies. Each agent acts independently and does not require access to the full state or information of other agents. In our system, we implement in OSMOTICGATE2 server the RL trainer and deploy in OSMOTICGATE2 client the local agents.

Algorithm 1 illustrates how we train the actor and critic network of each RL agent. First of all, we initialize the environment for each device, and initialize the parameters $\theta_k$ of the actor $\pi_k$, and the parameters $\phi_k$ of the critic $V_k$. When the RL agents are ready, both the server and the edge nodes start receiving the video streams. During the training process, each edge node is allocated a buffer list $\tau_k$. At time step $t$, the agent observes the local state $O_k^t$ and the global state $G_k^t$, then the action is generated $A_k^t = \pi_k(O_k^t; \theta_k^t)$. The $A_k^t$ is executed in the system until time step $t+1$. At this time step, the new local state $O_k^{t+1}$, global state $G_k^{t+1}$, as well as the reward $R_k^t$ are observed in the cloud server. All these stats are collected in the system as one buffer for updating the model parameters. The agents continue to interact with the system until the end of the episode where $t == T$. After one episode, for each pair of actor $\theta_k$ and $\phi_k$, all collected samples in the trajectory are fed to the objective function of the respective actor and critic networks, and the network parameters are updated with ADAM optimizers.

## IV. IMPLEMENTATION DETAILS

### *A. Video Analytics Module*

The video analytics module is built upon the work in *Osmoticgate* [6] while we extend the framework with several parallel techniques to speed up the video processing operations. The processing process in OSMOTICGATE2 includes video encoding, inference, and video transmission across the edge and the cloud server. We illustrate in the following subsections the techniques employed to speed up the processing within the system.

*1) Parallel Video Encoder:* When the video streams are fed into the edge devices, they are encoded within the system with the configuration command from the OSMOTICGATE2 clients. The encoding process entails compressing the video streams to target bitrate and resolutions, in order to accelerate the inference speed. As encoding computation is highly CPU-intensive, we utilize the multi-core architecture and implement multi-process encoding for accelerating the encoding process in the first place. Specifically, the video streams are split into small chunks and encoded concurrently in different processes. The results are then pushed to the local processing queue for local inference or transmission queue to be processed by the cloud server.

*2) Inference Engine:* We implement inference engines for both the edge and cloud devices for making predictions on the encoded video chunks. On the edge devices, models of various configurations are loaded into the memory. OSMOTICGATE2 clients decide the appropriate model to be used for processing the current chunks. On the cloud server, only a large model with the best performance is deployed for making predictions.

All received video chunks are decoded into batches of frames before being loaded into the models for inference.

*3) Concurrent Listener:* When transmitting the raw video contents from the edge to the cloud, the packet re-assembly may take much time and it is easy for the socket to get stuck at this time. This is especially true when multiple clients are transmitting to the cloud simultaneously, where the clients have to wait for the other clients to send all the content.

Thus, we design a socket pool with each containing a worker process listening to connections and receiving the packets. The concurrent processes are continuously monitoring a whole socket pool, re-assembling the received packets. All received packets are sent to the queue, then the cloud cluster for further processing.

### B. Multi-agent Controllers

The core of the OSMOTICGATE2 is the coordination of components for alleviating the computation and communication bottlenecks across the edge and cloud servers. In order to do so, we deploy on each edge device an OSMOTICGATE2 *client* and a global OSMOTICGATE2 *server* on the cloud. Each OSMOTICGATE2 *client* contains an agent and generates system configurations for controlling the local processing operation in real-time. The configurations are generated based on local system stats collected via the *Edge Monitor* and the previously generated configurations.

Specifically, the *System Monitor* is deployed on the OSMOTICGATE2 *server* collecting all local stats. A *RL Trainer* is deployed on the cloud server along with all local agents. Via collected information from the *System Monitor*, the *RL Trainer* is able to jointly update all agents, such to achieve the goal of optimal system performance.

### C. Message-forwarding Module

In order to realize the seamless control between the *video analytics module* and the *multi-agent controllers* mentioned above, we implement the *Message-forwarding module* as a middleware to exchange information between these two modules across the edge and the cloud.

The module is based on RabbitMQ where we initiate two queues: 1) *System Stats* queue for forwarding the inference results from all devices as well as the local/global stats. All stats from both the edge devices and cloud server are then aggregated in the system monitor to assist the agent training process 2) *Agent Replica* queue for forwarding and deploying the updated agents to the edge side. OSMOTICGATE2 *server* contains all agent replicas within the system. Once the agents are updated via the RL trainer, the newly generated models are sent to the target device, ensuring all agents are up-to-date, conforming to the system states.

## V. PERFORMANCE EVALUATION

### A. Experimental Setting

**Testbed:** We use NVIDIA Jetson Xavier NX (with ARMv8.2 CPU and 8GB RAM) as the edge node and the cloud server is a bare metal Ubuntu machine, with 8 cores (Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz), GeForce RTX3090 graphics card and 48 GB RAM. The cloud operating system is Ubuntu 20.04.

TABLE II: The average inference accuracy and processing latency for each video chunk with different models, including the encoding, decoding latency. This does not include the queue waiting time, and the transmission latency.

| Edge | Accuracy | | | Latency(s) | | |
|---|---|---|---|---|---|---|
| Model(YOLOv8) | m | s | n | m | s | n |
| 540P | 0.844 | 0.794 | 0.746 | 2.117 | 1.646 | 1.025 |
| 360P | 0.817 | 0.763 | 0.721 | 1.512 | 1.306 | 0.851 |
| 240P | 0.762 | 0.731 | 0.653 | 1.289 | 1.174 | 0.754 |
| Cloud | Accuracy | | | Latency(s) | | |
| Bitrate(kbps) | 3000 | 1000 | 500 | 3000 | 1000 | 500 |
| 540P | 0.846 | 0.814 | 0.741 | 0.707 | 0.677 | 0.668 |
| 360P | 0.842 | 0.818 | 0.775 | 0.569 | 0.548 | 0.542 |
| 240P | 0.815 | 0.797 | 0.767 | 0.483 | 0.473 | 0.467 |

**Dataset:** We use road traffic video datasets from UA-DETRAC [13] to construct our video analytics system, which contains 10 hours of video captured at 24 different locations at Beijing and Tianjin in China [1]. The videos are recorded at 30 frames per second (fps), with a resolution of 960×540 pixels. We subtract from the simple and medium category a total of 120s of video data to construct our training dataset.

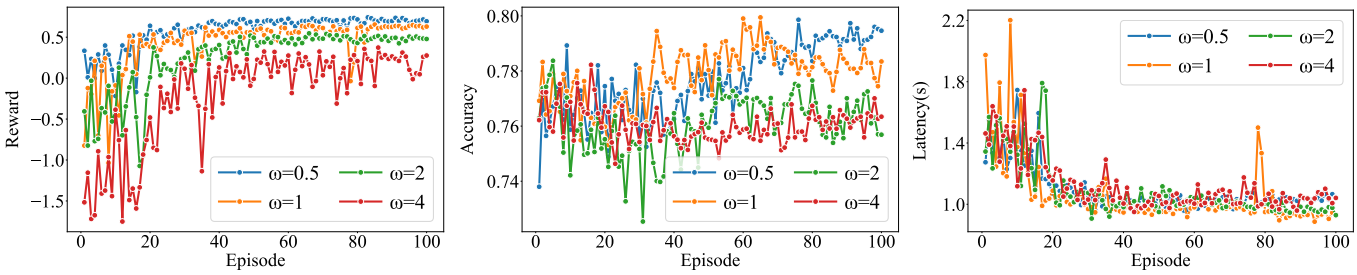
In order to simulate real-world bandwidth connection, we use public traces from oboe [14] with TC to control the bandwidth between the edge and the cloud server. We sample a sub-trace from all the traces and repeatedly emulate the bandwidth throughout the whole training process. In particular, during the training of each episode, the network condition changes every 20 seconds, and the duration of each time step $t$ is set to 10 seconds.

**System Configuration:** We consider object detection when analyzing the video data in UA-DETRAC and deploy yolo [15] models for detection. We use the same architectures as in [15] and deploy on the edge device YOLOv8-n, YOLOv8-s, YOLOv8-m models with 3.2M, 11.2M, 25.9M parameters respectively. As cloud has enough computation power, we only deploy the YOLOv8-m model with highest accuracy. The models are pre-trained on COCO dataset fine-tuned on UA-DETRAC subset later. During the fine-tuning process, we start with a learning rate of 0.01 and a batchsize of 64 and train for 300 epochs, and then adjust the learning rate to 0.001 and continue training 300 epochs to get the final model.

We show in Table II the running performance of different models in the system. Due to the different processing pipelines, the inference accuracy of different edge models are correlated with the video frame resolution as encoding to smaller bitrate is unnecessary. Whereas the cloud performance is influenced by both the resolution and video stream bitrate. The system configuration space is further complicated by the offloading rate as well. In the reward function, $F$ is set to 1 and $l^{thres}$ is set to 1.5 in all experiments.

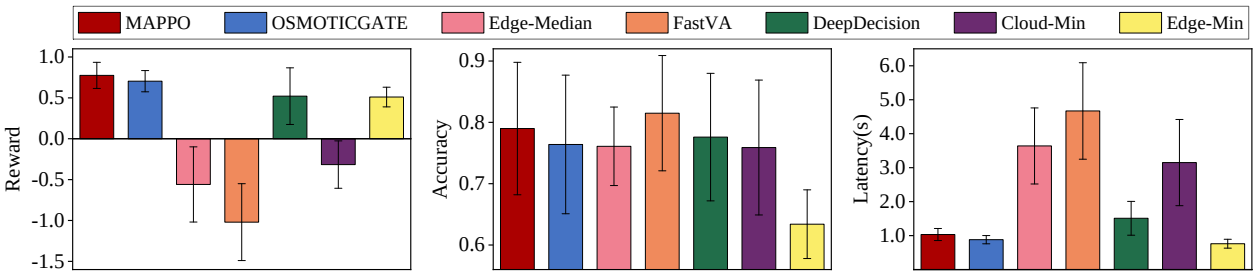
**Baseline methods:** We assess the performance of our approach by contrasting it with the following baseline.

[1] https://detrac-db.rit.albany.edu/




(a) Reward Curve (b) Accuracy Curve (c) Latency Curve

Fig. 3: Convergence and Performance of OSMOTICGATE2 under Different Penalty Weights




(a) Average reward (b) Average inference accuracy (c) Average latency

Fig. 4: Performance of OSMOTICGATE2 and baseline with Penalty Weights of 0.5

1) OSMOTICGATE [6]: the cloud server models the system and collects global information, using a two-stage gradient-based algorithm to provide offloading strategies for each edge node.

2) FastVA [5]: considers making the most use of network transmission to offload the video chunks to the cloud.

3) Edge-Median: edge nodes choose the smallest offload rate, the medium resolution, the medium bitrate, and the medium model.

4) DeepDecision [4]: addresses an optimization challenge wherein video processing occurs selectively either at the edge or the cloud side, contingent upon the prevailing system throughput within each time interval.

5) Cloud-Min: edge nodes choose the largest offload rate, and choose the smallest resolution and the lowest bitrate.

6) Edge-Min: edge nodes choose the smallest offload rate, resolution, lowest bitrate, and the smallest model.

### *B. Convergence and Performance under Different Penalty Weights*

We analyze the convergence of the proposed OSMOTICGATE2 under different penalty weights $\omega$, i.e., 0.5, 1, 2, 4. We set $l^{target}$ as 1 in this experiment. As seen in Fig. 3, all 4 sets of experiments can converge to a stable policy. The convergence can be fast at around 40 episodes, which validates the effectiveness of the proposed MAPPO algorithm. In Fig. 3a, the overall reward decreases when $\omega$ gets larger, with around 0.77, 0.63, 0.48, and 0.23 reward for 4 penalty weights respectively. This is mainly due to the larger penalty incurred from the deviations of the latency target. However, the reward does not fully reflect the system performance as it combines both the latency and accuracy of the system.

From the accuracy curve and latency curve in Fig. 3b and 3c, when the value of $\omega$ is set to 0.5, our algorithm attains its highest accuracy of 0.79. This is primarily because a smaller value of $\omega$ places greater emphasis on achieving high accuracy. During the training process, the accuracy curve gradually converges to a point and the latency is stable at the end. As reflected in Fig. 3c, the convergence latency is around 1s, which satisfies the pre-defined $l^{target}$. The training curve reveals that our algorithm learns configurations that prioritize meeting the latency target over maximizing accuracy.

### *C. Performance Comparison with Baselines*

In this experiment, we compare the performance of the proposed algorithm with several baselines. We set $\omega$ to be 0.5, and $lt^{target}$ to be 1. We report the average reward for different methods in Fig. 4a and the detailed system performance in Fig. 4b and 4c.

From Fig. 4a, we can see that MAPPO outperforms baselines in all experiments. OsmoticGate is the most competitive baseline in the literature. However, as its primary goal was to minimize system processing latency, OsmoticGate finally achieves 0.88 latency and 0.76 accuracy, an inferior performance as compared to us. FastVA achieves the highest accuracy 0.815 and latency 4.67, as it tends to transmit as many as possible video chunks to the cloud, but it leads to cloud overload. Other simple heuristics such as Edge-Median and Edge-Min all lead to worse performance in our experiments. In conclusion, our OSMOTICGATE2 with MAPPO method strictly conforms to the system performance target, i.e., maximize accuracy while ensuring around 1.0 system latency. OSMOTICGATE2 can ensure real-time streaming video processing while maximizing the prediction accuracy.

The experimental results validate the superiority of the multi-agent reinforcement learning in our OSMOTICGATE2.

## VI. CONCLUSION

In this paper, we study the problem of real-time video analytics across the edge and cloud environments. We propose OSMOTICGATE2 for orchestrating the configurations and performing task offloading across the edge and the cloud. In order to adapt to distributed and dynamic environments, we introduce a sophisticated online multi-agent reinforcement learning system crafted. Powered by the cutting-edge multi-agent reinforcement learning algorithm MAPPO, all agents are actively engaged in real-world environments, continually learning from these interactions. This dynamic learning process enables the system to optimize its performance effectively in a changing environment. Through rigorous experimentation, we show exceptional adaptability to varying system configurations while maintaining stable runtime performance.

## ACKNOWLEDGMENTS

This work was supported in part by National Key Research and Development Program of China under Grant 2022YFE0196000; in part by China Postdoctoral Science Foundation under Grant 2023M743403; in part by Zhejiang Provincial Natural Science Foundation of Major Program (Youth Original Project) under Grant LDQ24F020001; in part by the Fundamental Research Funds for the Central Universities.

## REFERENCES

[1] L. Zhang, J. Xu, Z. Lu, and L. Song, "Crossvision: Real-time on-camera video analysis via common roi load balancing," *IEEE Transactions on Mobile Computing*, pp. 1–13, 2023.

[2] J. Jiang, Z. Luo, C. Hu, Z. He, Z. Wang, S. Xia, and C. Wu, "Joint model and data adaptation for cloud inference serving," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 279–289.

[3] K. Zhao, Z. Zhou, X. Chen, R. Zhou, X. Zhang *et al.*, "Edgeadaptor: Online configuration adaption, model selection and resource provisioning for edge dnn inference serving at scale," *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5870–5886, 2023.

[4] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE conference on computer communications*. IEEE, 2018, pp. 1421–1429.

[5] T. Tan and G. Cao, "Fastva: Deep learning video analytics through edge processing and npu in mobile," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1947–1956.

[6] B. Qian, Z. Wen, J. Tang, Y. Yuan, A. Y. Zomaya, and R. Ranjan, "Osmoticgate: Adaptive edge-based real-time video analytics for the internet of things," *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 1178–1193, 2022.

[7] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang *et al.*, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2020, p. 359–376.

[8] Y. Wang, W. Wang, J. Zhang, J. Jiang, and K. Chen, "Bridging the Edge-Cloud barrier for real-time advanced vision analytics," in *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

[9] "Can 30,000 cameras help solve chicago's crime problem?" https://www.nytimes.com/2018/05/26/us/chicago-police-surveillance.html, accessed: 2023-10-31.

[10] C. Yu, A. Velu, E. Vinitsky, J. Gao *et al.*, "The surprising effectiveness of ppo in cooperative multi-agent games," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 611–24 624, 2022.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[12] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," *Information Fusion*, vol. 85, pp. 1–22, 2022.

[13] L. Wen, D. Du, Z. Cai, Z. Lei, M. Chang, H. Qi *et al.*, "UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking," *Computer Vision and Image Understanding*, 2020.

[14] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett *et al.*, "Oboe: Auto-tuning video abr algorithms to network conditions," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 44–58.

[15] "Yolov8," https://github.com/ultralytics/ultralytics, accessed: 2024-03-19.

## BIOGRAPHIES

**Bin Qian** [Student Member, IEEE] (bin.qian0718@gmail.com) received the MSc degree in data science from the University of Southampton, U.K, in 2018. He is currently working toward the PhD degree in computer science from Newcastle University, Newcastle Upon Tyne, U.K. His research interests include IoT, machine learning, task offloading.

**Yubo Xuan** (221122120292@zjut.edu.cn) is currently pursuing a master's degree in computer science and technology at the School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China. His research interests include IoT, machine learning and Edge-Cloud collaboration based on Machine Learning

**Di Wu** [Student Member, IEEE] (dw217@st-andrews.ac.uk) is currently pursuing a PhD degree in computer science at University of St Andrews, UK. He received a B.S. degree in Information System and Information Management from Northeast Forestry University, China in 2015, and an M.S. degree in Data Science from University of Southampton, UK in 2018. His major interests are in the areas of federated learning, distributed machine learning, edge computing, model compression, and Internet of Things.

**Zhenyu Wen** [Senior Member, IEEE] (wenluke427@gmail.com) is currently a Tenure-Tracked Professor with the Institute of Cyberspace Security, Zhejiang University of Technology. His current research interests include IoT, crowd sources, AI system, and cloud computing. For his contributions to the area of scalable data management for the Internet of Things, he was awarded the the IEEE TCSC Award for Excellence in Scalable Computing (Early Career Researchers) in 2020.

**Renyu Yang** [Member, IEEE] (renyuyang@buaa.edu.cn) is currently an Associate Professor in the School of Software, Beihang University, China. He was with the University of Leeds UK, Alibaba Group China and Edgetic Ltd. UK, having industrial experience in building large-scale resource scheduling systems. He is a recipient of Alan Turing Post-Doctoral Enrichment Award, 2022. His research interests include parallel and distributed computing, and deep learning systems and applications. He is a member of IEEE.

**Shibo He** [Senior Member, IEEE] (s18he@zju.edu.cn) received the Ph.D. degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2012. He is currently a Professor with Zhejiang University. He was an Associate Research Scientist from March 2014 to May 2014, and a Postdoctoral Scholar from May 2012 to February 2014, with Arizona State University, Tempe, AZ, USA. From November 2010 to November 2011, he was a Visiting Scholar with the University of Waterloo, Waterloo, ON, Canada. His research interests include Internet of Things, crowdsensing, big data analysis, etc.

**Jiming Chen** [Fellow, IEEE] (cjm@zju.edu.cn) received the B.Sc. and Ph.D. degrees in control science and engineering from Zhejiang University, Hangzhou, China, in 2000 and 2005, respectively. He is currently a Professor with the College of Control Science and Engineering, and the Deputy Director of the State Key Laboratory of Industrial Control Technology, Zhejiang University. His research interests include the Internet of Things, sensor networks, networked control, and control system security

**Rajiv Ranjan** [Senior Member, IEEE] (rranjans@gmail.com) is an Australian-British computer scientist, of Indian origin, known for his research in Distributed Systems (Cloud Computing, Big Data, and the Internet of Things). He is University Chair Professor for the Internet of Things research in the School of Computing of Newcastle University, United Kingdom. He is the director of Networked and Ubiquitous Systems Engineering (NUSE) Group, jointly with Dr. Graham Morgan, in the School of Computing. He is a fellow of Academia Europaea and the Asia-Pacific Artificial Intelligence Association. He is also the Founding Director of the International Centre (UK-Australia) on the Internet of Energy (IoE), funded by EPSRC.