FCPO: Federated Continual Policy Optimization for Real-Time High-Throughput Edge Video Analytics

1st Lucas Liebe, 1st Thanh-Tung Nguyen, 3rd Dongman Lee
School of Computing, *KAIST*, Daejeon, Republic of Korea
{lucasliebe, tungnt, dlee}@kaist.ac.kr

***Abstract*—The growing complexity of Edge Video Analytics (EVA) facilitates new kind of intelligent applications, but creates challenges in real-time inference serving systems. State-of-the-art (SOTA) scheduling systems optimize global workload distributions for heterogeneous devices but often suffer from extended scheduling cycles, leading to sub-optimal processing in rapidly changing Edge environments. Local Reinforcement Learning (RL) enables quick adjustments between cycles but faces scalability, knowledge integration, and adaptability issues. Thus, we propose FCPO, which combines Continual RL (CRL) with Federated RL (FRL) to address these challenges. This integration dynamically adjusts inference batch sizes, input resolutions, and multi-threading during pre- and post-processing. CRL allows agents to learn from changing Markov Decision Processes, capturing dynamic environmental variations, while FRL improves generalization and convergence speed by integrating experiences across inference models. FCPO combines these via an agent-specific aggregation scheme and a diversity-aware experience buffer. Experiments on a real-world EVA testbed showed over $5\times$ improvement in effective throughput, 60% reduced latency, and 20% faster convergence with up to $10\times$ less memory consumption compared to SOTA RL-based approaches.**

***Index Terms*—Federated Reinforcement Learning, Continual Learning, Edge Computing, Dynamic Batching, Visual Analytics**

I. Introduction

Video Analytics (VA) is widely regarded as a "killer application" in Edge Computing [1]. High-demand applications , such as traffic monitoring [2] and surveillance [3], generate substantial data volumes that require local processing to enhance data privacy, minimize network latency and improve throughput [4]. VA services are typically organized into a series of tasks as pipelines. To optimize latency and throughput of the whole pipeline, approaches such as [4]–[8] periodically perform scheduling to balance the workloads assigned to the edge server and devices, avoiding performance bottlenecks.

Once assigned, each device performs its assigned tasks until the next scheduling period, ranging from a few minutes [4] to 3 hrs [3]. We identify that dynamic real-time edge applications require continual performance optimization at the device level during this period due to multiple factors. These factors include fluctuating network conditions, workload variability, and the heterogeneity of computing devices (Figure 1), spanning a range of x86-64 servers and arch64 embedded computers with diverse computational capacities and resource availability [4]. Thus, on-device fast-paced and constant local optimization is

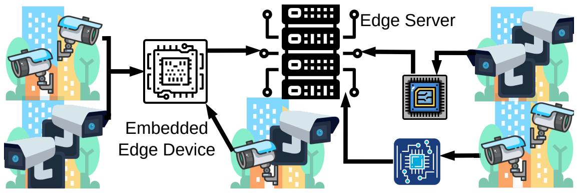


Fig. 1: A VA scenario with collaborating Edge devices. *Various sizes, shapes, and colors illustrate architecture, resource, and workload heterogeneity.*

essential to meet real-time requirements, as their violations can cause catastrophic failures and unsafe behavior [9].

To this end, several studies have proposed *Reinforcement Learning (RL)*-based approaches, adaptively learning near-optimal solutions at the edge device [10]–[13]. However, they have yet to overcome four major limitations.

(1) Offline training. Although RL can learn in real-time from newly obtained experiences, this capability is often limited by computational complexity (particularly in Deep RL) and concerns over reliability and stability [14]. For example, QoSAS [11], BCEdge [10], and DDQN [12] rely on offline-trained RL agents, utilizing only the inference phase at runtime. This approach restricts the real-time adaptability of RL, and these pre-trained agents may experience performance degradation, due to large environmental variations at the Edge.

(2) Poor Scalability. In existing approaches [10]–[12][1], adaptation is achieved by deploying one RL agent per device. Since these agents rely on offline training, they must continually collect and store experiences as data for retraining. As the number of inference models increases, the storage and time required for training also grow, with each iteration processing more data samples. This rapidly becomes a scalability bottleneck, restricting efficiency in large-scale deployments.

(3) Learning Divergence. In current approaches [10]–[12][1], when a new environment is encountered, a base agent is typically cloned and then the clone is tasked with optimization for that environment. Although these agents begin with similar initial understanding, their learning diverges over time due to *experience heterogeneity*. This makes it difficult for these agents to have a consistent unified policy across collaborating edge devices and the server.

(4) Cold start. Newly cloned agents often face significant cold start challenges when they are deployed in a new environment. These challenges occur because the agents are initialized with

[1]MagicPipe [13] does not provide an algorithm on how to scale up when new devices or models are introduced into the cluster.

TABLE I: Comparisons to the state-of-the-art RL-based systems

System	Online Learning	Scala-bility	Knowledge Fusion	Warm Start
QoSAS [11]	✗	✗	✗	✗
BCEdge [10]	✗	✗	✗	*Last*[‡]
MagicPipe [13]	✓	✗	✗	✗
DDQN [12]	✗	✗	✗	*Last*[‡]
FCPO	✓	✓	✓	✓

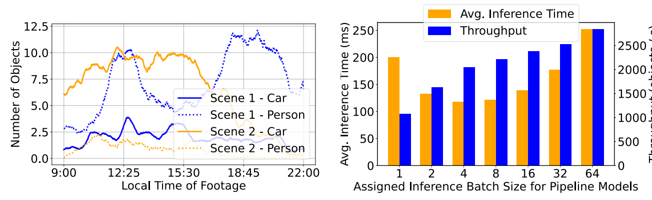
[‡] Last training checkpoint.

pre-trained parameters or policies from a previous setting. The severity of these cold start issues is closely related to the differences between the old and new environments. When there are substantial disparities in state distributions, action dynamics, or reward structures, agents typically experience a significant decline in performance. This drop in performance results from the mismatch between the agent's learned representations and the unfamiliar conditions it encounters in the new environment.

System Overview and Contributions. To address these limitations of the existing works, we propose FCPO – a system based on real-time *Federated Continual Reinforcement Learning* for *Policy Optimization* of pipeline configuration in latency-constrained EVA. In FCPO, each lightweight *inference agent (iAgent)* is specifically designed to manage and optimize a DNN inference model, rather than an entire device, fostering scalability even as the number of models grows. Each iAgent adapts dynamically to heterogeneous environments by selecting among three actions: (a) inference batch size selection, which balances throughput and latency; (b) input resolution selection, optimizing between accuracy and computational load; and (c) multi-thread processing, enabling efficient parallelism based on current resource availability. These actions represent complex and context-aware resource allocation strategies, allowing iAgents to continually adjust to changing conditions and workloads in real-time. Our iAgent-based approach allows for finer-grained control over resource allocations and enhances adaptability across varying models and workloads. To overcome the other limitations, we propose a novel learning method, *Federated Continual Reinforcement Learning (FCRL)*, which integrates *Continual Reinforcement Learning (CRL)* with *Federated Reinforcement Learning (FRL)*. Recent work has provided a foundational framework for CRL, defining a stable approach for continually training agents in dynamic environments [15]. Building on this, we incorporate FRL [16] to facilitate collaborative learning across agents in heterogeneous, non-IID environments with varying state transitions. To our knowledge, this is the first implementation of CRL combined with FRL in the novel FCRL framework for real-time systems.

Overall, our contributions and benefits to existing works are shown in Table I and summarized as follows:

- We propose FCPO, a scalable system for optimizing real-time performance of VA inference pipelines in large-scale edge systems by enabling fine-grained control over batch size, resolution, and multi-thread processing.
- We introduce FCRL, combining Federated and Continual Reinforcement Learning, to enable adaptive, collaborative learning in dynamic, non-iid edge environments for VA.
- We develop an agent-specific aggregation scheme that combines shared backbone knowledge with heterogeneous actions, handling large action spaces and environment-specific optimizations efficiently.
- We conduct real-world experiments, which show over 5× improvement in effective throughput, 60% reduced latency, and 20% faster convergence with up to 10× less memory consumption, demonstrating FCPO's superiority in managing complex, large-scale VA tasks with enhanced adaptability compared to SOTA RL-based approaches.



(a) Content dynamics in 2 traffic scenes (b) Batch latency/throughput trade-off

Fig. 2: Motivation for adaptation and dynamic batching.

II. Preliminaries and Motivations

A. Design Goals

Similar to prior works [10]–[13], FCPO aims to achieve high throughput and low latency to ensure efficient, real-time processing of VA tasks. In addition to these objectives, we emphasize minimizing convergence speed, decision latency, training latency, and optimizing resource utilization. Real-time decision-making and adaptive learning processes can impose significant demands on hardware resources, especially in constrained edge environments. Therefore, FCPO is designed to rapidly converge on optimal configurations with minimal decision and training latency, while keeping resource usage as low as possible to maximize the available capacity for workloads. This approach ensures that resources are primarily dedicated to VA processing rather than overhead, achieving both efficiency and responsiveness in large-scale deployments.

B. Action Space for Quick Adaptation

In the highly dynamic edge environment, fluctuating network conditions and variable workloads, as shown in Figure 2a, introduce significant challenges to maintaining consistent performance. These factors necessitate continual optimization to adapt to rapid changes in real-time as they can lead to degraded throughput, increased latency, and inefficient resource use. To this end, we consider three methods with different throughput trade-offs as actions for the agent, which are described in the next sections.

Dynamic Batched Inference. Batched inference has long been regarded an effective method to increase the inference throughput of DNN models [17], as it efficiently leverages the parallel capabilities of GPUs. By dynamically adjusting the batch size, it is possible to achieve desirable throughput levels. Figure 2b demonstrates the impact of batching on throughput and end-to-end latency within an EVA traffic monitoring pipeline, highlighting two key insights: (1) Larger batch sizes improve throughput via vector-level parallelism but increase latency, as the first query in a batch must wait for the last. (2) Smaller batch sizes reduce individual inference latency but can create pipeline bottlenecks, causing higher end-to-end latency, as seen with batch sizes of 1 and 2. Thus, while dynamic batching can be an effective tool for balancing throughput and latency, careful tuning is crucial for optimization.

Resolution Adjustments. Resizing input data (e.g., video frames) to improve throughput has been widely studied [18]–[20]. However, traditional DNNs often lack support for variable input sizes, and using multiple inference engines can lead to excessive resource demands. FCPO addresses this with frame packing, combining smaller images into a single frame for inference models. A similar method is proposed by Peng et al. [20] and Gokarn et al. [19].

Multi-thread Processing. DNN-based query processing comprises three steps: (1) pre-processing (e.g., normalization), (2) DNN inference, and (3) post-processing (e.g., decoding and filtering). While pre- and post-processing cannot be batched, throughput can be improved via concurrency, using multiple threads to prevent bottlenecks. However, on resource-limited embedded devices, excessive threading may degrade performance due to resource contention. Carefully managing thread allocation is crucial to balance concurrency benefits with resource efficiency, avoiding system overload.

C. Challenges for Continual Adaptation in Real-Time

Continual learning of RL agents in a real-world environment, rather than relying on simulations, is critical due to the inherent variability of edge scenarios. Simulations often fail to capture the whole stochastic nature of network conditions, hardware heterogeneity, and workload variability. However, achieving continual learning with real-time training presents four critical challenges not yet overcome by existing works:

1) ***Exponential Action Combinations****:* The combination of actions required for optimal performance in edge environments creates an exponential search space, making naive random exploration infeasible. In FCPO, the agent must dynamically adjust the batch size, resolution, and multi-threading level. Selecting the optimal batch while simultaneously tuning frame packing and thread counts results in a vast array of configurations, each affecting latency, throughput, and resource usage. Efficient exploration strategies are thus essential to identify the best action combinations without excessive sampling.

2) ***Learning Overhead****:* Real-time learning introduces computational overhead which can create contention for critical resources such as CPU cycles, memory bandwidth, and GPU processing power, potentially leading to slower inference times and degraded performance.

3) ***Non-IID Experiences****:* Each agent's experience is unique, as it encounters distinct sequences of states and actions due to varied exploration paths, workload patterns, and environmental conditions. This non-iid nature complicates knowledge aggregation and fusion across agents, as they evolve based on distinct data distributions.

4) ***Heterogeneous Action Spaces****:* Due to device and model heterogeneity, resource constraints may limit viable actions (e.g., batch size or resolution choices), making the action spaces inconsistent across agents.

The next section covers our approach to address these challenges, which is essential to ensure that RL agents can effectively learn in real-world edge systems, maintaining performance and reliability across diverse and variable conditions.

III. Real-time Inference Serving System

A. System Overview

Figure 3a shows the components for controlling the real-time edge VA system. FCPO follows a popular setup for edge computing systems, consisting of multiple clusters [5]. Each cluster consists of a local server and multiple heterogeneous edge devices, connected to various data sources (e.g., real-time cameras). The local server runs the *System Controller* responsible for model allocation, system-wide scheduling, and executing *Agent-Specific FL Aggregation* (subsection IV-D). In this paper, we focus on perform continual local real-time optimization and leverage [4] for global periodic scheduling.

Both the server and devices are capable of hosting VA inference models. Due to their heterogeneous resource availability, the number of models hosted at each machine varies significantly. Each host device is controlled by a single *Device Control*, responsible for running the models according to the scheduled provided by *System Controller* and collecting device run-time statistics. In BCEdge [10] this component contains the intelligent agent, while in FCPO, this component is responsible for real-time metrics collection and FL participation. Communication from *System Controller* to the workload is coordinated through the *Device Control*, ensuring fair and sequential execution of system changes.

Each *Workload Model* is piggybacked with a light-weight *Continual RL (CRL)* agent, called *iAgent*, in charge of organizing the structure between processing threads and collects workload metrics. *iAgent* constantly observes the model's performance against its environment and adaptively learns the optimal configuration to improve the performance.

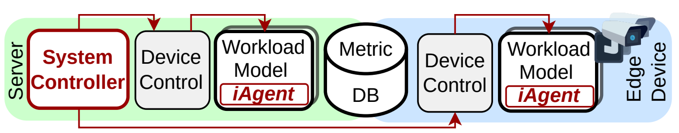
The final component in the system is a distributed *Metric Database (DB)* for storing real-time metrics that are used to evaluate the system and for updating allocation strategies.

B. System Procedure

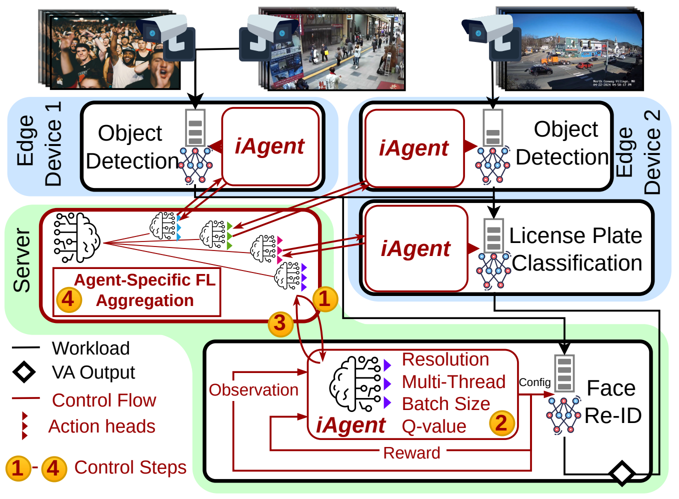
Figure 3b illustrates the architecture of FCPO, a system based on *Federated Continual Reinforcement Learning (FCRL)* for continual Policy Optimization in real-world edge environments. The workflow of the components in FCPO follows a typical FL procedure.

① At each round's start, the *System Controller* distributes the aggregated *Global Model (GM)* to all *iAgent*s. This helps *iAgent*s optimize their respective inference models, particularly assisting newly deployed models that might otherwise face cold starts.

② Each *iAgent* interacts with its specific environment using CRL to efficiency adapt itself to the environment.



(a) Overview of the main components.



(b) Detailed workflow and architecture for FCPO's real-time pipeline processing and *iAgent* Federated Continual Reinforcement learning (FCRL).

Fig. 3: System-wide FCPO architecture overview.

③ Once local training is complete, selected *iAgent*s upload the adapted model parameters to the server.

④ The *System Controller* aggregates the local updates into a new GM. As introduced in section I, the knowledge learned by one agent can greatly benefit the others. FCPO employs Federated Learning to combine the collective knowledge learned by agents. To avoid overgeneralization causing each agent to lose the environment-specific knowledge, FCPO is equipped with an *Agent-Specific FL Aggregation* algorithm.

In the next section, we go into the details of FCPO by first introducing the architecture of *iAgent* in subsection IV-A and how its learning can be formalized into a Markov Decision Process (MDP) in subsection IV-B. Then we introduce the novel FCRL method to allow effective continual adaptation in subsection IV-C and IV-D.

IV. Federated Continual Policy Optimization

A. *iAgent's Model Architecture*

The model architecture of *iAgent* is shown in Figure 4. It takes in an input of size 8, which includes the current actions, the arrival rate, and the number of drops from the full queue. It comprises a backbone, one value head, and three action heads. The backbone is implemented with two linear layers, featuring a hidden dimension of 64 and an output dimension of 48. These layers serve as a feature extractor for the subsequent layers, aiming to capture the overall dynamics of the environment. The value head is designed as a single linear layer to estimate the cumulative reward.

While all three actions are closely related, choosing all three actions together using a single action head results in a

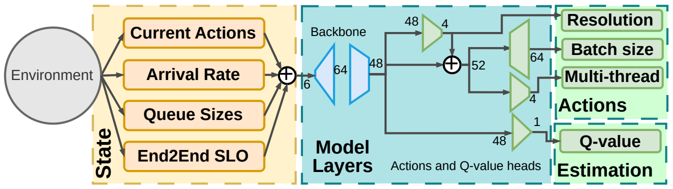


Fig. 4: Architecture of *iAgent*.

large exploration space for this head and consequently slower convergence. Thus, to generate the probability distribution for action sampling, the network has one linear layer for each action, followed by a softmax activation function. To facilitate feature sharing and alternating optimization among the heads, we take inspiration from Faster R-CNN model architecture with cascading outputs [21].

Particularly, the first action head utilizes the backbone's features to determine the resolution. Its output is then concatenated into the backbone features to determine the actions of the other two heads. This approach enables the agent to learn the dependencies among the actions. For instance, if the request rate is static and the resolution is lowered to accommodate two images in a frame, the throughput is doubled and the batch size can be reduced. Additionally, lowering the resolution increases the preprocessing rate, resulting in the necessity for an additional preprocessing thread to avoid bottlenecks.

B. *Markov Decision Process of iAgent*

We describe the MDP of *iAgent* as consisting of states S, actions A, a transition probability distribution P, and a reward function R. Every step $n \in \mathbb{N}$, *iAgent* collects the state and chooses an action. Multiple steps within an episode are used to optimize the behavior in the next episode, as shown in Figure 4 and Figure 5. *iAgent* learns the environment defined through the MDP and identifies an optimal policy $\pi^* = \text{argmax}_a Q^*(s,a)_n$, that maximizes the cumulative expected reward for a state-action-pair at step n. The optimal Q-function $Q^*(s,a)_n$ defines the reward by choosing action $a \in A$ in state $s \in S$, which is approximated through a neural network.

State Space S. A state space captures the characteristics of an environment. At the beginning of a step, the agent constructs a state $s_n \in S$ as a vector of size 8, $s_n \in S \subset \mathbb{R}^8$. The most important state metric is the request rate, as it determines the throughput the agent needs to achieve. Other metrics guide the agent in its decision-making process, such as the current resolution configuration, batch size, and thread configuration. The current queue sizes between processing steps are also incorporated to help the agent identify processing bottlenecks. While these inputs may not be strictly necessary for a well-trained agent, they can significantly improve the learning speed, which is crucial for online-training. The last state value is the end-to-end *Service Level Objectives (SLO)* of the pipeline, it is used to give the agent context for what kind of processing speed is required.

Action Space A. An action refers to the behavior chosen by the agent for the current step n. For *iAgent*, each action

$a_n \in A \subset \mathbb{N}^3$ is represented by a three-tuple:

$$[RES(\text{resolution}),\ BS(\text{batch size}),\ MT(\text{multi-thread})]$$

All actions can be changed instantly at runtime, and have a direct impact on inference throughput. The optimal choice is to increase $a_n[1] = BS$, as decreasing the resolution may decrease accuracy, and adding threads allocates additional resources. Nevertheless, RES and MT may be necessary to handle a high workload with low latency.

Transition Probability Distribution $\boldsymbol{P}$**.** A transition probability $p_{s_{n+1}} \in P$ defines how likely an agent will transition from current state s_n to the next state s_{n+1} based on the chosen action a_n. $p_{s_{n+1}} = P(s_{n-1}|s_n, a_n)$ captures the environment dynamics as well as the effects of an action.

Reward Function $\boldsymbol{R}$**.** The reward function returns a scalar value r_n, rating the last action, and is designed to directly reflect the design goals by subtracting latency and an oversize penalty from the throughput. These values are aggregated as the cumulated expected reward $\mathbb{E}[\sum_{n=0}^{|C|} \gamma^n r_n]$, where r_n is the reward calculated after step n. The reward is normalized between -1 and 1, resulting in the following equation for R with lat as the estimated weighted average of the local latency:

$$r_n = \frac{1}{2} * (\vartheta \frac{Throughput_n}{RequestRate_n} - \varsigma\ lat - \varphi \frac{a[1]_n}{RequestRate_n}) \quad (1)$$

Compared to BCEdge [10] we do not directly include the model SLO into the reward function, because (a) *iAgent* should learn the relationship of throughput and latency and (b) each models deadline within a pipeline is ambiguous. However, the oversize penalty is increased by the number of requests that exceed the local SLO. This way the reward is indirectly decreased for not meeting SLOs.

Finally, the ***optimal reward function*** can be written as:

$$Q^*(s,a)_n = \mathbb{E}_{\gamma \sim p(s_{n-1})}[r_n + \gamma \max_{a_{n-1}} Q^*(s,a)_{n-1}] \quad (2)$$

where γ is the discount factor to balance between the immediate and future rewards.

C. Continual Reinforcement Learning (CRL)

As long as dynamic changes follow a consistent pattern, they can be represented within a single MDP. For example, over short intervals, unstable network bandwidth can be captured by a single probability distribution [22]. However, averaging this distribution over longer periods may lead to imprecision, as edge deployments often face varying patterns.

CRL allows agents to be defined across multiple environments represented by different MDPs—a challenge for traditional RL definitions. For instance, permanent road construction alters content dynamics in traffic monitoring, impacting the transition probability distribution and changing the MDP. If the MDP shifts during or after training, traditional RL agents, such as those offline-trained in existing methods [10]–[13], cannot adapt to the new environment.

Based on the notation on Abel et al. [15] the state space S is called observations O, which combined with actions A create histories $h \in H$ of sequential pairs $h = o_0 a_0 ... o_n a_n$ for

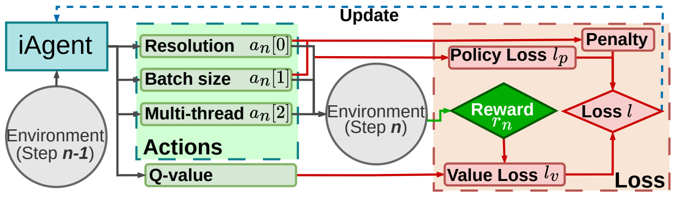


Fig. 5: The training framework of FCPO's *iAgent*(Figure 4).

$o_n \in O$ and $a_n \in A$. An agent is represented as a function $\lambda \in \Lambda : H \to \Delta(A)$, with Δ representing a probability distribution over a countable set. The environment is a function $e \in E : H \times A \to \Delta(O)$ that can capture the MDP (subsection IV-B). The learning process is described as agents searching for the set of optimal agents $\Lambda^* \subset \Lambda$, and the notation $\Lambda_1 \vdash_e \Lambda_2$ refers to $\Lambda_1 \subset \Lambda$ converging to $\Lambda_2 \subset \Lambda$ by observing e. Proof that the presented *iAgent* is a valid CRL problem is presented in the Supplementary Material.

Extension to FCRL. The formulation of CRL can be extended to FCRL by defining it over multiple instances. Within every round of federated learning, the model sent from server to instance i should be interpreted as the basis Λ_B for the local CRL problem $(e_i, v_i, \Lambda, \Lambda_B)$. The aggregated model parameters serve as a basis that is not optimal without personalization and quickly adapted with continual learning.

This notation captures how every *iAgent* solves it's personal CRL problem based on shared general knowledge. While an instance can encounter different environments and performance evaluations, the set of all agents Λ is identical. In FCPO the performance function is also identical $(\forall i_1, i_2 : v_{i_1} = v_{i_2})$, forming a special case of FCRL problem where instances target an identical goal.

Loss Architecture. The loss calculated based on the steps in the current episode, using the $ratio$ of explored to exploited actions. Besides policy and value loss, *iAgent* receives a direct penalty for actions 1 (resolution) and 3 (multi-threading), which is visualized in Figure 5. The total loss l is determined as:

$$l = l_p + l_v + \omega \cdot \frac{1}{n} \sum_n (a_n[0] + a_n[2]) \quad (3)$$

where ω is a hyperparameter to adjust the penalty. The policy loss l_p is the average of the clipped ratio $\varepsilon \cdot ratio$, balancing exploration, multiplied with the Generalized Advantage Estimation to learn transition probability distribution [23]. The exponential of the negative reward is included as a factor to provide more direct feedback of the total reward value and fast continual adaptation to slight changes.

$$l_p = \frac{1}{n} \sum_n \min\left(\varepsilon \cdot \text{ratio}, \text{ratio}\right) \cdot \left(\text{GAE} + e^{-r_n}\right) \quad (4)$$

The mean squared error $mse()$ between the estimated Q-values and the collected rewards forms the value loss l_v. By optimizing this, *iAgent* learns policies with better performance. This estimation is also used for sharing knowledge of the overall environment trends across federated agents.

$$l_v = \text{mse}(Q(s,a)_n, r_n) \quad (5)$$

Algorithm 1: Agent-specific Aggregation - Server

```
1  Function Main():
2      select and await selected clients
3      for $l \in$ base_network layers do
4          agg_layers.add($l$)
5      for $m \in$ client models $M$ do
6          for $l \in \{layer_1, layer_2, layer_{value}\}$ do
7              agg_layers[$l$] $+= m[l]$
8          for $l \in \{layer_{a[0]}, layer_{a[1]}, layer_{a[2]}\}$ do
9              $factor = \left(\text{LOSS}_l - \frac{\text{LOSS\_TOTAL}}{|M|}\right)^{-1}$
10             agg_layers[$l$] $+= factor \cdot m[l]$
11             LOSS_TOTAL $+=$ LOSS$_l$
12     agg_layers $/= |M| + 1$
13     for $m \in$ client models $M$ do
14         for $l \in \{layer_1, layer_2, layer_{value}\}$ do
15             $m[l]$.load($agg\_layers[l]$)
16         send $m$ to client
17     base_network.load(agg_layers)
```

The reward function is complex, and adding more components makes it harder to optimize and balance the parameters effectively. To address this, the loss penalty serves as a way to ensure the batch size is optimized first, while other actions are only used when the improvement in the primary objectives is substantial enough to justify their trade-offs. The secondary goals of accuracy and resource consumption are more challenging to evaluate from a local perspective, without labels to assess accuracy and resource consumption that may seem acceptable to one task but is detrimental to others.

Overhead Minimization. Deploying CRL on embedded devices introduces overhead that can reduce the performance of the VA system. Training neural networks is particularly challenging due to limited resource availability. To address this, *iAgent*s employ a loss gate that executes back-propagation only when the improvement is significant. If the loss magnitude falls below a specified threshold, the network update is minimal and can be skipped. However, to prevent the learning process from stagnating in sub-optimal positions, the FL update is always executed, as described in the next section.

Between RL updates, storing *experiences* increases memory consumption—a critical issue on embedded devices. Each agent maintains its experiences in a fixed-sized buffer, which imposes an upper limit on memory usage. This issue is even more pronounced in RL systems performing offline updates over extended periods, as they require large buffers to collect numerous experiences. For example, for each update, BCEdge [10] and DDQN [12] store over 5000 experiences.

In FCPO, online CRL training allows the buffer to remain small and to be emptied frequently, significantly reducing memory overhead. Additionally, *iAgent*'s buffer is populated based on experience diversity to maximize training efficiency. After each forward pass, diversity d is calculated as follows:

$$d = \alpha \cdot D_M(s_n, s_{n-1}, \cdots, s_0) + \beta \cdot D_{KL}(\pi) \tag{6}$$

where D_M is the Mahalanobis distance between the new state and stored states, which emphasizes novelty, and D_{KL} is the KL-Divergence of policy distributions, which captures

Algorithm 2: Agent-specific Aggregation - Client

```
1  Function Main():
2      $m_{local}$ = $local\_update(m_{local})$; send $m_{local}$ to server
3      while await ($m_{aggregated}$) from server do
4          continue inference
5          history_states += state; history_actions += action
6      POLICY $= m(history\_states)$
7      LOSS = neg_log_likelihood(POLICY, history_actions)
8      $m_{aggregated}$.freeze($layer_1, layer_2, layer_{value}$)
9      $m_{aggregated}$.update(LOSS); iAgent.load($m_{aggregated}$)
```

deviations in action spaces. This lightweight buffer design eliminates sequential dependencies between experiences and improves their IID distribution. Thus, *iAgent*s do not require an additional replay buffer of past episodes like BCEdge [10].

D. Federated Reinforcement Learning (FRL)

Knowledge Aggregation. As shown in algorithm 1, each federated update step start by selecting clients for aggregation. Next, local weights and experiences are collected after a last local update (line 2). During aggregation, only the backbones and Q-value head are aggregated equally (lines 6-7, 12). These layers are designed to extract features from a state and provide a general understanding of the environment that remains consistent across agents. Equal aggregation ensures that all agents contribute equally to this shared knowledge, preventing any single agent from dominating the model's understanding. A weighted aggregation, in contrast, could prioritize local information from specific agents, leading to imbalances and potentially diminishing the performance of other agents.

On the other hand, action heads are aggregated using a loss-based aggregation across all agents with the same output dimensions (lines 8-11). It is essential to store different action heads based on their dimensions since weights for selecting batch sizes of 1-64 or 1-16 are not directly comparable.

After this, the updated network is transferred back to the agents, replacing the current network (line 17). During this process, the agents do not perform any local updates, and any experiences collected during this time are discarded. The latest aggregated network, which includes the updated action heads, is then stored on the server (line 18).

Action Head Fine-Tuning. Using the aggregated layers directly to make decisions can lead to unpredictable performance because the output from the backbone is not aligned with the action heads. To address this, we fine-tune the model with experiences collected locally at each agent, focusing solely on the policy loss while freezing the value head and backbone (algorithm 2 lines 6-9). This fine-tuning step, is performed on the edge devices, as it is faster than a regular update, and therefore should not introduce additional overhead. Also fine-tuning at the server would require sending local experiences to the server, adding additional network overhead.

Large-Scale FL. FCPO combines hierarchical FL with client selection to minimize network overhead caused by the frequent exchange of model parameters. The Edge inherently represents a hierarchical network structure where edge devices are co-located with cameras and connect to a local cluster

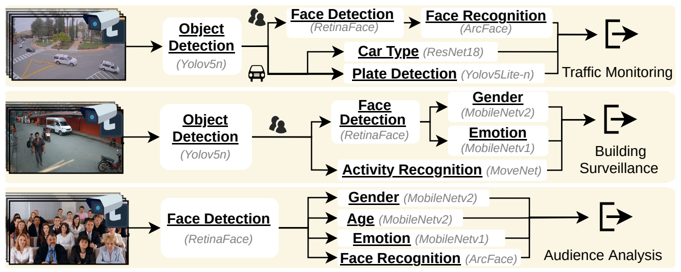

Object Detection (Yolov5n)
Face Detection (RetinaFace)
Face Recognition (ArcFace)
Car Type (ResNet18)
Plate Detection (Yolov5Lite-n)
Traffic Monitoring
Object Detection (Yolov5n)
Face Detection (RetinaFace)
Gender (MobileNetv2)
Emotion (MobileNetv1)
Activity Recognition (MoveNet)
Building Surveillance
Face Detection (RetinaFace)
Gender (MobileNetv2)
Age (MobileNetv2)
Emotion (MobileNetv1)
Face Recognition (ArcFace)
Audience Analysis

Fig. 6: Sample pipelines for traffic, surveillance, and audience analysis.

edge server. This edge server can then communicate with other clusters or the cloud to distribute workload.

This topology inherently forms clusters of devices, which can be leveraged to reduce the number of participating devices. Additionally, client selection that considers memory, computing availability, and data heterogeneity as FedHybrid [24] reduces the risk of stragglers. The considerations of FedHybrid align well with FCPOs efforts to reduce device overhead and prioritizing diversity. To further lower network overhead, we extend the utility function proposed in [24] by incorporating bandwidth (normalized to $10Mbit/s$) for client c as follows:

$$TotalUtil(c) = Util(c)_{[24]} * \sqrt{Bandwidth(c)/10} \qquad (7)$$

The *System Controller* will determine the participation of a device, considering the utility sum of all agents in a single device based on $TotalUtil(c)$. A chosen *Device Control* makes the final decision, which of the local *iAgent*s will participate in the next FL round based on memory availability.

Taking these factors into account, FCPO carries out fine-grained client selection within each cluster. Once multiple aggregation rounds are completed within a cluster, the updates are shared with other clusters through the cloud, as proposed in [25]. This step follows the same aggregation process used at the edge server and the same number of local rounds.

V. Evaluation

In our evaluation, we assess FCPO on a real-world testbed and aim to answer the following questions:

- **Q1** (V-B1): *Does* FCPO *outperform other Edge VA systems with and without local optimization?*
- **Q2** (V-B2): *How fast and reliable is* FCPO*s FL procedure?*
- **Q3** (V-B3): *Can* FCPO *adapt to strict real-time SLOs?*
- **Q4** (V-B4): *Does* FCPO *enable measurable warm starts?*
- **Q5** (V-B5): *How much overhead does* FCPO *incur on the devices? And does that prohibit scalability?*
- **Q6** (V-C1): *Can* FCPO *adapt quickly in drastically changing environments? And if so, how does CRL help?*
- **Q7** (V-C2): *How does FRL benefit learning convergence?*

A. Experimental Methodology

1) ***Real-world Testbed****:* We evaluate FCPO with an edge server with 4 consumer-grade GPU NVIDIA RTX 3090 and 12 heterogeneous devices consisting of 3 Jetson Xavier AGXs, 5 Jetson Xavier NXs, 3 Jetson Orin Nanos and an "On-Premise" desktop PC equipped with a GTX 1080Ti. We emulate real-world network bandwidth conditions using the data transfer benchmark of an Irish 5G dataset [26].

TABLE II: Summary of FCPO's implementation and training parameters.

Parameters	Description
$n_s = 10$	Number of steps in each episode
$LR = 10^{-3}$	Learning rate of *iAgent*
$\vartheta, \varsigma, \varphi = 1.1, 10, 2$	Weights to calculate the reward (Eq. 1)
$\gamma, \lambda = 0.1$	Weights of reward temporal relation (Eq. 2 + GAE)
$\omega = 0.2$	Weight of loss penalty (Eq. 3)
$\varepsilon = 0.9$	Clip value in policy loss (Eq. 4)
$\alpha, \beta = 0.5$	Weights to calculate experience diversity (Eq. 6)

2) ***System Implementation****:* FCPO's implementation is build based on PipelineScheduler [27] in C++ and adds over 5,000 new lines of code to the system. We deploy each inference model within a container and leverage a microservice architecture to harness its inherent flexibility and robustness, with Docker serving as the container runtime. However, FCPO can also operate in a monolithic architecture with minimal modification. We implement and run the *Controller* as a separate process on the server to supervise the operation of the entire cluster. Since the focus of FCPO is continual local adaptation, we utilize [4] to make global scheduling decisions at the *Controller*, distributing the workload across devices every 5 minutes. On each edge device and the server where inference containers are hosted, we deploy a *Device Agent* to manage and monitor the containers.

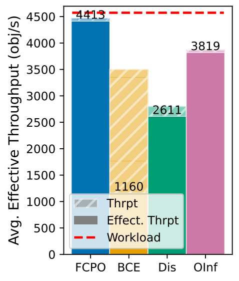
Within each container, we deploy each inference model in a single process. We use OpenCV (4.8.1) for image (video frame) pre- and post-processing tasks. Different inference engines, such as TensorRT, ONNX, TF Lite, and OpenVINO, are supported in an easy plug-and-play manner. For the experiments in this paper, we use TensorRT (8.4.3.1) to load and run inference models. As described in section IV, we attach a lightweight *iAgent* (implemented with LibTorch) to each inference process to optimize its operation. The parameters used for the experiments are detailed in Table II. The code base can be applied to various visual inference tasks requiring little modification.

3) ***Edge VA Workloads****:* As shown in Figure 6, we use traffic monitoring, building surveillance, and audience analysis as three distinct representative edge VA applications to evaluate FCPO. We set a strict end-to-end SLO of 250ms for all pipelines to reflect the prompt nature of Edge VA applications and services. For data, we collected a total of 23 continual 4-hour videos from real-world public online streams to represent 23 data sources with diverse object distributions and content dynamics. During the experiments, the videos are streamed at 15 FPS to simulate real-time video sources across devices.

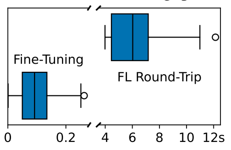
We also leverage the vehicle tracking data from the 2022 AI City Challenge [28] to evaluate *iAgent*, which was unsupervisedly trained on our collected dataset, in its ability to adapt to a new domain Figure 10. We use 9 6-min videos at 10 FPS.

4) ***Baselines****:* For all experiments, we leverage OctopInf [4] to distribute the workload across multiple devices every 5 minutes. Our goal is to demonstrate that combining *infrequent global scheduling* with *continual local optimization* provided by FCPO can significantly improve performance.

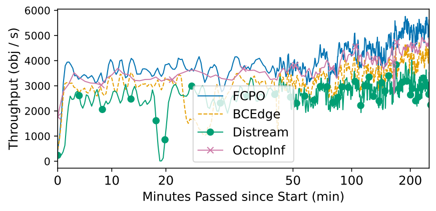
- *BCEdge* [10] represents state-of-the-art RL-based throughput optimization. We selected BCEdge as a baseline



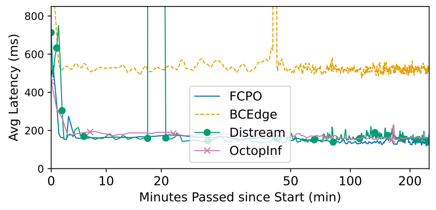
(a) Effective throughput



(b) FL roundtrip latency



(c) Averaged total end-to-end throughput.



(d) Average end-to-end latency of arrived requests.

Fig. 7: End-to-end system performance comparisons over 4h.

because, to the best of our knowledge, it is the most recent work on this topic and exhibits the most desirable qualities among the SOTAs compared in Table I. We deploy one agent per device, which was trained offline using profiling results, and is updated every 7000 experiences. To prevent catastrophic failures by excessive GPU resource requests, we limit the concurrency and shared memory actions to only two configurations each.

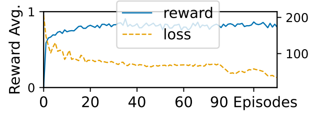
- *OctopInf* [4] serves as a global scheduling baseline without local optimizations, but global parameter settings.
- *Distream* [5] is a popular baseline without any detailed runtime optimization of parameters like batch size.

*5) **Metrics**:* Every reported metric is the average value over three runs. We validate FCPO by comparing its end-to-end performance against baselines using the following:

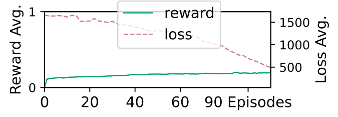
- *End-to-End Latency.* This is the time elapsed from the generation of a video frame at the source to the moment its inference results arrive at the designated *sink*, including all network, queuing, and processing latencies. It measures the responsiveness of the system in handling video streams.
- *Throughput.* Our scenarios involve pipelines that analyze attributes of objects-of-interest using inference models. Therefore, we define ***throughput*** as the total number of objects that are analyzed by the pipelines within one second. However, results that arrive late—where their end-to-end latency exceeds their SLO (set at 250ms)—are no longer useful. Thus, we also measure the ***effective throughput***, which is *the total number of inference results that arrive on time within one second*. This number highlights the degree of real-time processing.

To prove the scalability and real-time capability of FCPO, we analyze the overhead incurred by *iAgent* using the following metrics for RL and system resources:

- *Convergence Speed.* Convergence refers to the state of accurately approximating optimal decision. Its speed indicates how quickly *iAgent* learns the environment.
- *Memory Consumption.* Memory allocated to *iAgent*.
- *Power Consumption. iAgent*'s power consumption on edge devices.
- *Decision Latency.* Time taken to choose a single action.



(a) FCPO (*online training*)



(b) BCEdge [10] (*offline training*)

Fig. 8: Learning performance with averaged loss and rewards.

- *Training Latency.* This is the time required to update the network on-device locally without FL.

B. Results

*1) **Processing Throughput and Inference Latency Improvements**:* Figure 7 shows the overall throughput performance of FCPO. FCPO significantly outperforms the baselines in terms of **throughput** and **effective throughput**. FCPO implements local continual optimization on top of OctopInf's global workload distribution. By continually adapting to the environment via actions such as batch size adjustment FCPO manages to output nearly 2 times the throughput and better latency (Figure 7d) most of the time, which proves the validity of our framework. However, this is not the case for BCEdge, which also employs a local adaptation approach on top of OctopInf (Figure 7a and c). By adjusting batch sizes, compared to Distream, BCEdge reduces the number of queue drops and thus achieves higher throughput. However, only a small portion of this is effective throughput because its average latency is significantly worse (Figure 7d). While increasing throughput, batched inference raises the latency of each inference request [18]. The effect is compounded over multiple stages of the pipeline, resulting in nearly 2.5 times higher latency.

We further shed light on the root cause by analyzing Figure 8. The reward and loss of FCPO show gradual improvement and importantly noticeable fluctuation as it continually tries to adapt to the ever-changing environment dynamics. Contrary, BCEdge is **trained offline** with profiling data and its reward quickly converges because profiling data is obviously less diverse in workload patterns and cannot capture all the conditions of devices as well as the interactions among various applications (processes) running on the devices.

Another cause is that there is only one BCEdge agent on each device to make decisions for all workloads, which becomes the bottleneck. Contrary, FCPO has one light-weight *iAgent* for each workload increasing the timeliness of its decision. We will further discuss the scalability shortly.

*2) **Federated Learning Latency**:* The FL latencies in Figure 7b show that the average FL round completes in approximately 4–8 seconds. This aligns with findings from a recent study [29], where an FL round using FedAvg [30] over 5G networks took 43 seconds with 6 nodes and a model size of approximately 3MB. In contrast, the *iAgent* model is only 53KB in size, though it is aggregated across more nodes. Despite this, the FL optimization in FCPO achieves efficient round-trip times—from sending weights to receiving the aggregated model. Because local *iAgents* continue operating during this period, FL latency does not hinder real-time processing. Furthermore, on-device fine-tuning after aggregation

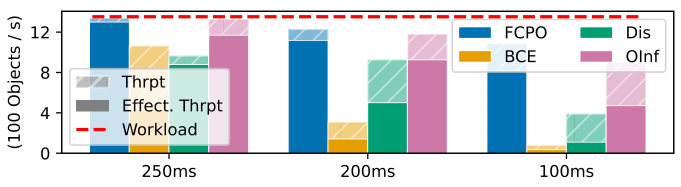


Fig. 9: Throughput comparisons under increasingly restricted SLOs.

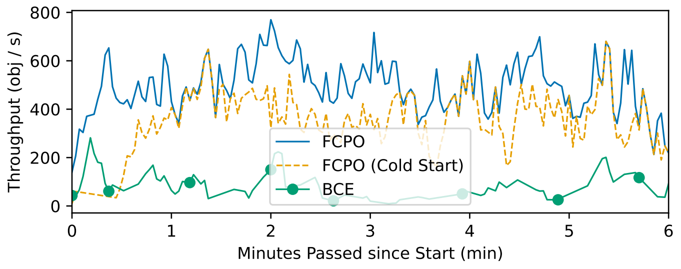
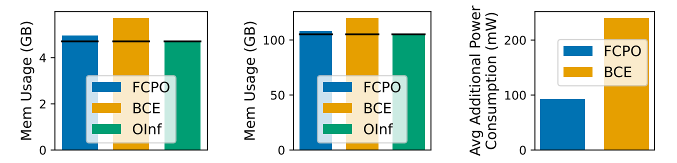
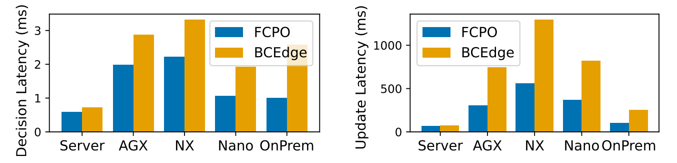


Fig. 10: Performance on a traffic dataset [28] with *out-of-distribution* workload patterns.



(a) Average *device* memory consumption (b) Average *server* memory consumption (c) Avg consumption for *real-time local optimization*



(d) Decision latencies per step. (e) Update latencies per episode.

Fig. 11: Agent overhead comparisons averaged over 4h. (*Portions above the black line in (a) and (b) are memory consumption of RL agents*)

introduces negligible overhead—less than 300ms across all devices—which is still below the highest update latency shown in Figure 11e. Thus, *iAgent* can reliably support continual adaptation in real-world deployments without compromising responsiveness.

*3) **Adaptation to Stricter Real-Time Requirements**:* We evaluate *iAgent*'s ability to adapt to real-time constraints by tightening the SLO from 250ms to 200ms and 100ms, respectively. As shown in Figure 9, both Distream and OctopInf experience a severe drop in performance. This is because they make periodic scheduling decisions based solely on average workload statistics from the previous period. Under stricter SLOs, the system dynamics become significantly more volatile, requiring immediate and responsive adaptation.

By incorporating the end-to-end SLO as a state variable, *iAgent* learns to associate tighter latency requirements with appropriate system configurations, enabling it to respond more effectively to increased urgency and maintain better performance. In contrast, although BCEdge incorporates SLO information into its reward function, it fails to capture the diversity of environment patterns, often leading to suboptimal decisions. As a result, it accumulates only minor rewards (Figure 8), which are insufficient to drive effective learning.

*4) **Analysis of Warm Starts**:* In real-world scenarios, environmental changes can occur both gradually and abruptly. To evaluate FCPO's ability to achieve a "warm start" even during abrupt changes, we replace the data sources with videos from the AI City Challenge dataset [28], which exhibits *out-of-distribution* workload patterns. As shown in Figure 10, the trained *iAgent* maintains high throughput throughout the experiment. We then increase the difficulty by evaluating a blank *iAgent* (cold start). Although it initially exhibits low throughput, it quickly adapts to the new environment and, by the end of the experiment, achieves performance only slightly below that of the warm-start *iAgent*. In contrast, BCEdge performs worse than it did on the original dataset and fails to adapt to the new environment. These results demonstrate that *iAgent* not only adapts effectively to previously unseen conditions but also supports both warm and cold start scenarios, making it highly suitable for deployment in dynamic, real-world settings.

*5) **Analysis of On-Device Agent Overhead**:* To prove the scalability of FCPO show the overhead comparisons in Figure 11. Regarding memory consumption, all FCPO's *iAgent*s combined take less than 3% of the total memory allocated for the experiments at both the Edge devices and the server. This is owing to its lightweight structure and small fixed-size experience buffer. On the other hand, even though there is only one BCEdge agent per device, it takes significantly more memory (up to 10×) due to a bulky structure and a large experience buffer (7000 experiences). The BCEdge agent is deeper and wider compared to *iAgent*. It also requires an additional branch to analyze the state value, which results in more intermediate layers, which in turn increases memory consumption. Moreover, since BCEdge only needs to train offline the server, we disable it at the Edge devices to save resources for pipeline processing. Otherwise, the memory consumption at the edge is expected to be even higher.

We collect edge power consumption metrics through Jtop and calculate the agent overhead by subtracting the workload power consumption. On average, BCEdge consumes more than twice as much energy as FCPO on each edge device. This highlights how much more expensive an inference of BCEdge is on limited resources and how efficient *iAgent* operates.

Regarding latencies, on each of the edge devices, BCEdge takes $1.5-2\times$ more time to make a decision due to its more complex model. The training time of FCPO can safely execute until the next decision time after one second even on low-end devices like Orin Nano. This result shows, how the lightweight design of FCPO is capable of real-time workload configuration. The update latencies of BCEdge are collected offline, as it is not designed for online learning. Running it‘s training at the same time of inference will result in worse performance due to co-location interference [4].

C. Ablation Study

To provide further insights into FCPO, we conduct an ablation study, where we remove two heads and use a single

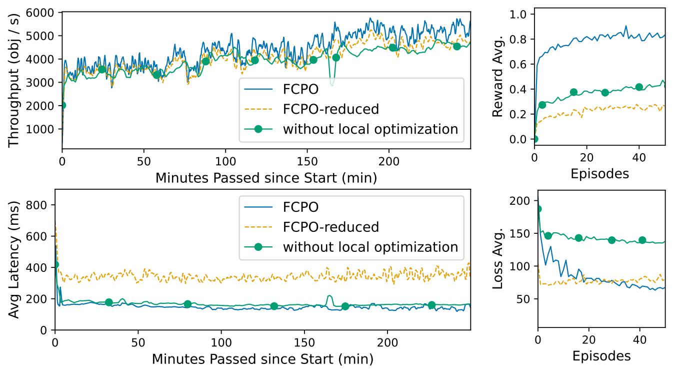


Fig. 12: Ablation study comparisons with a reduced configurations.

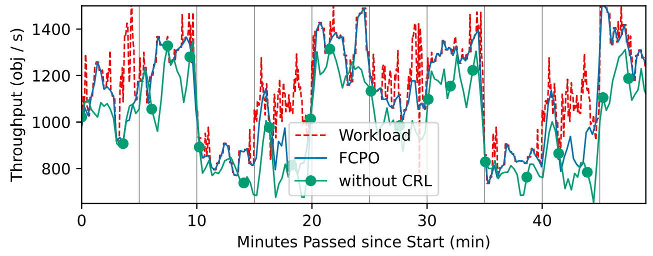


Fig. 13: Impact of Continual Learning on Effective Throughput. (*Vertical lines indicate a significant context switch*).

head for all three actions in the FCPO-reduced version. As shown in Figure 12, removing certain aspects of FCPO leads to significantly worse performance. Specifically, deploying *iAgent* at the server making optimizations every 5 minutes results in less responsive updates, leading to suboptimal decisions. On the other hand, FCPO-reduced, while still able to perform updates, struggles to understand the action space, resulting in low rewards and high latency. These findings support our argument that a single head introduces a highly complex action space, where *iAgent* fails to effectively explore its decisions.

1) ***Benefits of Continual Learning on Performance:*** To capture the performance impact of including CRL in the design, we concatenate multiple 5-minute video segments from different sources, causing the underlying distributions to change drastically. We evaluate two *trained* instances of FCPO at the same checkpoint over 50 minutes of traffic monitoring at 3 sources. One agent is *frozen* without CRL, while the other is allowed to *learn* continually using CRL. Figure 13 shows that, although the frozen agent is unable to adapt, it still makes near-optimal decisions because it was trained beforehand. However, it is outperformed by the learning agent throughout the entire experiment. The CRL agent shows better adaptation to workload spikes, but unfamiliar scenes like 40 to 45 minutes can still present a challenge. While both agents follow similar patterns and have comparable performances at each context switch, the CRL agent can quickly adapt to the new context and perform significantly better until the next switch. This proves the effectiveness of CRL in adapting to changing environments.

2) ***Convergence Speed with Federated Learning:*** Figure 14 illustrates the convergence speed, at varying numbers of pipelines used during FRL training in the traffic pipeline.

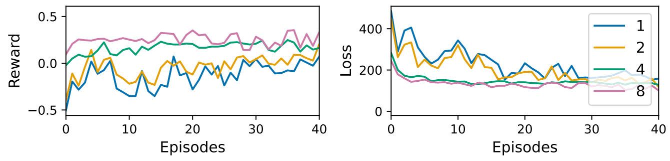


Fig. 14: Average loss and reward with different FRL instance counts.

Results are averaged across all running models and all parameters are identical across configurations. In the single-pipeline scenario (1) federated aggregation is disabled, the other configurations aggregate every second episode.

As the number of pipelines—and consequently *iAgents*—increases, both reward and the rate of loss reduction improve, with the most significant gains observed between 4 and 8 pipelines. This suggests that learning scales effectively with more federated agents, though diminishing returns emerge at higher numbers. This effect may reflect an upper limit on achievable improvements, as loss and reward converge quickly. Each additional pipeline introduces slight variations in dynamics, increasing the complexity of optimizing the global model. These variations are also the cause of learning divergence. *Instead of diverging*, FRL combines these differences into a more general scene understanding and smooth the learning curves, resulting in less performance fluctuations.

VI. Discussion

A. *Feasibility and Scalability in Real-world Environments*

In this section, we further examine the feasibility and scalability of FCPO in real-world environments, where sensitivity to system overhead, large-scale collaborative learning, and heterogeneity in applications and devices are key challenges.

Overheads. From the experimental results, each device, on average, incurs less than 3% total memory usage, less than 1.7% total power consumption, and approximately 53KB of network overhead every 300s for continual adaptation—resulting in more than a 13% improvement in throughput. These overheads are negligible, thanks to our lightweight design of *iAgent* and efficient Agent-specific Federated Learning (FL) aggregation mechanism. This minimal resource footprint is crucial for practical real-world deployments, where computational and energy budgets are constrained.

Large-scale Collaboration. In dense Edge networks, vanilla Federated Learning (FL)—where all agents participate in every round—is typically infeasible due to the significant network and computation overhead imposed on the centralized server, which becomes a bottleneck. To address this, FCPO is designed to selectively involve only a subset of agents with the most valuable experiences in each round, thereby maintaining constant system complexity. Meanwhile, the remaining agents continue to perform local optimizations independently.

Application Heterogeneity. The state and action spaces of FCPO are tailored for VA applications. However, its adaptation mechanism—Federated Continual Reinforcement Learning—is general and applicable to a broad range of tasks. FCPO learns the environment state and selects actions such as batching, multi-threading, and resolution adjustment to maximize throughput. Similar strategies are widely used in machine learning, including for time-series prediction and natural language processing. For example, batching has proven effective for RNNs [31], [32] and Transformers [33], [34], both of which benefit from parallel processing. This suggests FCPO can be extended to support such applications efficiently.

Hardware and Software Heterogeneity. A key challenge to real-world scalability lies in the heterogeneous mix of hardware and software platforms from different manufacturers. FCPO is built around generalizable actions whose effectiveness can be consistently evaluated across major platforms. While batched inference is a relatively recent innovation from the deep learning era—unlike established techniques like resolution adjustment and multi-threading—it is now widely supported. Major hardware such as GPUs [4], CPUs [35], [36], TPUs, and NPUs [37], as well as inference engines like TensorRT [4], [18], OpenVINO [35], [36], and TensorFlow [38], offer native batching support, making it a scalable optimization across diverse environments.

B. Future Work

Besides further valuation and fine-tuning to evaluate the performance on other real-time systems, we outline directions for future work aimed at advancing the deployment of FCPO in increasingly complex and dynamic real-time edge scenarios.

Decentralized and Hybrid Federated Learning (FL). We aim to enable even greater scalability through decentralized or hybrid (FL), where the aggregation of *iAgent*s could occur within networking hardware or on edge devices. While this concept has been explored in prior work, it has not been considered for real-time inference. Additionally, further investigation is needed to fully understand the trade-offs and overheads associated with such decentralized aggregation strategies.

Global Reward Learning. As application, hardware, and software heterogeneity grows, designing the reward function through manual hyperparameter tuning (see Appendix B) may become increasingly impractical. To address this, we plan to integrate fast local optimization with global reward learning. This approach allows the reward function to self-adapt, reducing manual effort while potentially achieving better performance than hand-designed optimization functions.

C. Threats to Validity

Experiment scenarios. Despite our best effort, selected videos and scenarios may not fully capture the real-world variability, which could limit the generalization of our findings due to potential data bias. To mitigate this issue and strengthen the robustness of our conclusions, we collected 23 4-hour videos in different 3 domains–traffic, building surveillance, and audience analysis–with diverse workload patterns (Figure 2). We also incorporated the 2022 AI City Challenge Dataset [28], particularly Track 1: Multi-Camera Vehicle Tracking, to challenge *iAgent*'s learning mechanism on *out-of-distribution* data. Experimental results from subsubsection V-B4 show that *iAgent* can swiftly adapt to the new dataset.

Multiple treatment. Another challenge we face is the risk of multiple treatment interference, as several changes have been introduced—some of which are fundamental.This makes it difficult to isolate the effects of each individual adjustment. To address this, we conduct ablation studies to evaluate each proposed technique separately. The results indicate that each method contributes positively to overall performance, although accurately quantifying the specific impact remains challenging.

VII. Related Work

Edge Workload Distribution. For real-time inference, Zhao et al. [39] propose a reinforcement learning-based scheme that enhances throughput by selectively offloading parts of a model that cannot fully run on-device. Works like [4]–[6], [8] focus on balancing VA pipelines between edge devices and servers, by considering network conditions, workload changes, or energy consumption. CoEdge [40], combines DNN performance estimation with an inference scheduler for containerized co-execution and batching. Guan et al. [41] propose federated scheduling for DAG Tasks as in Figure 6 in mixed-critically systems. As solutions differ in resource centralization, execution flexibility, and throughput gains; self-learning approaches like FCPO can adapt accordingly.

Real-Time Reinforcement Learning (RL). Many areas like robotics rely on real-time on-device RL for modern applications under resource constraints [42]. To improve the performance of these applications, Liu et al. [43] present a new formulation for deadline-safe RL execution and Shirvani et al. [44] balance speed, accuracy, and energy dynamically. However, FCPO is too light-weight to benefit from these for significant performance improvements.

Edge Federated Learning (FL) faces unique challenges in heterogeneous edge environments. Lightweight agents can be achieved using symmetric conversion modules or early exits [45]. Zhang et al. [46] address bottleneck devices with a 3-stage client selection process, while hierarchical FL clusters agents into groups [47]. Asynchronous FL eliminates rounds to handle timing challenges, but suffers from inconsistent and biased updates. Hu et al. [48] address this through branch models, which are then aggregated into the global model. FCPO encounters much heterogeneity at runtime, benefiting from advanced FL approaches.

Edge Continual Learning (CL). Deng et al. [49] propose semi-supervised on-device CL for classification, while Yu et al. [50] explore lightweight unsupervised clustering using hyperdimensional computing. Adapting to embedded device constraints, Kwon et al. [51] present a hardware-aware meta CL system. These FL and CL advancements highlight their feasibility in Edge environments. However, applying CL in reinforcement learning, a distinct learning paradigm, remains an open question requiring evaluation through FCPO.

VIII. Conclusion

In this paper, we present FCPO, an optimization system that combines FRL and CRL to enhance the throughput of real-time Edge Visual Analytics (VA) inference. The dynamics of unstable network conditions and content variations provide significant opportunities for continual configuration adaptation. Each step in a VA pipeline is locally optimized by an *iAgent*, which dynamically adjusts input resolution, batch size, and multi-thread configuration. The experimental results show FCPO significantly outperforms recent baselines.

FCPO highlights the importance of local optimization in real-time edge systems, improving the speed and efficiency of machine learning tasks while ensuring better resource utilization. This results in more sustainable and cost-effective computing solutions, marking a step forward in efficient real-time distributed inference on embedded devices.

Reproducibility

The implementation used for the evaluations is provided as an open-source project on GitHub[1], together with detailed documentation to help reproduce the experimental results. Moreover, this code is archived on Zenodo[2] for long-term preservation. Though the data is not made publicly available due to ownership and data privacy concerns, interested researchers and developers can still recreate nearly identical datasets for academic purposes with our provided scripts.

Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (under grant No. RS-2019-II191126, Self-learning based Autonomic IoT Edge Computing), the KAIST Convergence Research Institute Operation Program, and KAIST Deep Mobility Consortium. The authors acknowledge the use of different AI systems (incl. GPT-4o) limited to editing and grammar enhancement.

References

[1] G. Ananthanarayanan, P. Bahl, P. Bodik, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-Time Video Analytics: The Killer App for Edge Computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[2] T.-T. Nguyen, S. Y. Jang, B. Kostadinov, and D. Lee, "PreActo: Efficient Cross-Camera Object Tracking System in Video Analytics Edge Computing," in *2023 IEEE Int. Conf. on Pervasive Computing and Communications (PerCom)*. IEEE, 2023, pp. 101–110.

[3] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 115–131.

[4] T.-T. Nguyen, L. Liebe, T.-N. Quang, Y. Wu, J. Cheng, and D. Lee, "OCTOPINF: Workload-Aware Real-Time Inference Serving for Edge Video Analytics," in *The 23rd International Conference on Pervasive Computing and Communications (PerCom 2025)*. IEEE, 2025.

[5] X. Zeng, B. Fang, H. Shen, and M. Zhang, "Distream: scaling live video analytics with workload-adaptive distributed edge intelligence," in *Procs. of the 18th SenSys*, 2020, pp. 409–421.

[6] X. Hou, Y. Guan, and T. Han, "Dystri: A Dynamic Inference based Distributed DNN Service Framework on Edge," in *ACM Int. Conf. Proceeding Series*. ACM, aug 2023, pp. 625–634.

[7] Y. Ma, R. Fu, A. Zou, J. Li, C. Chen, C. Lu, and X. Guan, "Performance optimization and stability guarantees for multi-tier real-time control systems," in *2024 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2024, pp. 187–200.

[8] C. Gao, N. Kumar, and A. Easwaran, "Energy-efficient real-time job mapping and resource management in mobile-edge computing," in *2024 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2024, pp. 15–28.

[1] https://github.com/tungngreen/PipelineScheduler
[2] https://doi.org/10.5281/zenodo.14789255

[9] J. Chen, A. Zou, Y. Xu, and Y. Ma, "Scenic: Capability and scheduling co-design for intelligent controller on heterogeneous platforms," in *2024 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2024, pp. 1–14.

[10] Z. Zhang, Y. Zhao, H. Li, and J. Liu, "Bcedge: Slo-aware dnn inference services with adaptive batch-concurrent scheduling on edge devices," *IEEE Transactions on Network and Service Management*, 2024.

[11] Z. Fang, T. Yu, O. J. Mengshoel, and R. K. Gupta, "Qos-aware scheduling of heterogeneous servers for inference in deep neural networks," in *Procs. of the 2017 ACM on Conf. on Information and Knowledge Management*, 2017, pp. 2067–2070.

[12] Y. She, T. Shi, J. Wang, and B. Liu, "Dynamic batching and early-exiting for accurate and timely edge inference," in *2024 IEEE 99th Vehicular Technology Conf. (VTC2024-Spring)*, 2024.

[13] G. Coviello, Y. Yang, K. Rao, and S. Chakradhar, "Magic-pipe: Self-optimizing video analytics pipelines," in *Procs. of the 22nd Int. Middleware Conference*, 2021, pp. 79–90.

[14] P. Osinenko, G. Yaremenko, G. Malaniya, and A. Bolychev, "An actor-critic framework for online control with environment stability guarantee," *IEEE Access*, vol. 11, pp. 89 188–89 204, 2023.

[15] D. Abel, A. Barreto, B. Van Roy, D. Precup, H. P. van Hasselt, and S. Singh, "A definition of continual reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[16] H. Jin, Y. Peng, W. Yang, S. Wang, and Z. Zhang, "Federated Reinforcement Learning with Environment Heterogeneity," *Procs. of Machine Learning Research*, vol. 151, pp. 18–37, 2022.

[17] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Batch: Machine learning inference serving on serverless platforms with adaptive batching," in *SC20: Int. Conf. for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–15.

[18] V. Nigade, P. Bauszat, H. Bal, and L. Wang, "Jellyfish: Timely inference serving for dynamic edge networks," in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 277–290.

[19] I. Gokarn, H. Sabbella, Y. Hu, T. Abdelzaher, and A. Misra, "Mosaic: Spatially-multiplexed edge ai optimization over multiple concurrent video sensing streams," in *Procs. of the 14th Conf. on ACM Multimedia Systems*, 2023, pp. 278–288.

[20] H. Peng, Y. Zhan, P. Li, and Y. Xia, "Tangram: High-resolution video analytics on serverless platform with slo-aware batching," in *2024 IEEE 44th Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 2024, pp. 645–655.

[21] R. F. Mansour, J. Escorcia-Gutierrez, M. Gamarra, J. A. Villanueva, and N. Leal, "Intelligent video anomaly detection and classification using faster rcnn with deep reinforcement learning model," *Image and Vision Computing*, vol. 112, p. 104229, 2021.

[22] R. Lubben, M. Fidler, and J. Liebeherr, "A foundation for stochastic bandwidth estimation of networks with random service," in *2011 Proceedings IEEE INFOCOM*. IEEE, apr 2011, pp. 1817–1825.

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

[24] K. Tam, C. Tian, L. Li, H. Zhao, and C. Xu, "Fedhybrid: Breaking the memory wall of federated learning via hybrid tensor management," in *Procs. of the 22nd SenSys*, 2024, pp. 394–408.

[25] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020-2020 IEEE international conference on communications (ICC)*. IEEE, 2020, pp. 1–6.

[26] D. Raca, D. Leahy, C. J. Sreenan, and J. J. Quinlan, "Beyond throughput, the next generation: A 5g dataset with channel and context metrics," in *Procs. of the 11th ACM multimedia systems Conf.*, 2020, pp. 303–308.

[27] T.-T. Nguyen, L. Liebe, T. N. Quang, Y. Wu, chengjinghan, and cdsnlab, "tungngreen/pipelinescheduler: v0.1.0 - percom 2025 implementation," Feb. 2025. [Online]. Available: https://doi.org/10.5281/zenodo.14789255

[28] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M. Chang, Y. Yao, L. Zheng, M. S. Rahman, A. Venkatachalapathy, A. Sharma, Q. Feng, V. Ablavsky, S. Sclaroff, P. Chakraborty, A. Li, S. Li, and R. Chellappa, "The 6th ai city challenge," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE Computer Society, June 2022, pp. 3346–3355.

[29] R. J. Hayek, J. Chung, K. Comer, C. R. Murthy, R. Kettimuthu, and I. Kadota, "Federated learning over 5g, wifi, and ethernet: Measurements and evaluation," *arXiv preprint arXiv:2504.04678*, 2025.

[30] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[31] C. Gao, A. Rios-Navarro, X. Chen, S.-C. Liu, and T. Delbruck, "EdgeDRNN: Recurrent Neural Network Accelerator for Edge Inference," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 4, pp. 419–432, Feb. 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9268992

[32] C. Holmes, D. Mawhirter, Y. He, F. Yan, and B. Wu, "GRNN: Low-Latency and Scalable RNN Inference on GPUs," in *Proceedings of the Fourteenth EuroSys Conference 2019*. Dresden Germany: ACM, Mar. 2019, pp. 1–16. [Online]. Available: https://dl.acm.org/doi/10.1145/3302424.3303949

[33] "NVIDIA/FasterTransformer," May 2025, original-date: 2021-04-02T21:36:33Z. [Online]. Available: https://github.com/NVIDIA/FasterTransformer

[34] "Mastering LLM Techniques: Inference Optimization," Nov. 2023. [Online]. Available: https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

[35] "Automatic Batching — OpenVINO™ documentationCopy to clipboardCopy to clipboardCopy to clipboardCopy to clipboardCopy to clipboardCopy to clipboardCopy to clipboardCopy to clipboardCopy to clipboardCopy to clipboardCopy to clipboard — Version(2024)." [Online]. Available: https://docs.openvino.ai/2024/openvino-workflow/running-inference/inference-devices-and-modes/automatic-batching.html

[36] A. Demidovskij, A. Tugaryov, A. Suvorov, Y. Tarkan, M. Fatekhov, I. Salnikov, A. Kashchikhin, V. Golubenko, G. Dedyukhina, A. Alborova, R. Palmer, M. Fedorov, and Y. Gorbachev, "OpenVINO Deep Learning Workbench: A Platform for Model Optimization, Analysis and Deployment," in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, Jan. 2020, pp. 661–668, iSSN: 2375-0197. [Online]. Available: https://ieeexplore.ieee.org/document/9288163

[37] M. Dörrich, M. Fan, and A. M. Kist, "Impact of Mixed Precision Techniques on Training and Inference Efficiency of Deep Neural Networks," *IEEE Access*, vol. 11, pp. 57 627–57 634, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10146255

[38] "Tensorflow Serving Configuration | TFX." [Online]. Available: https://www.tensorflow.org/tfx/serving/serving_config

[39] Z. Zhao, K. Wang, N. Ling, and G. Xing, "Edgeml: An automl framework for real-time deep learning on the edge," in *Procs. of IoTDI*, 2021, pp. 133–144.

[40] Z. Jiang, N. Ling, X. Huang, S. Shi, C. Wu, X. Zhao, Z. Yan, and G. Xing, "Coedge: A cooperative edge system for distributed real-time deep learning tasks," in *Procs. of the 22nd Int. Conf. on Information Processing in Sensor Networks*, 2023, pp. 53–66.

[41] F. Guan, J. Lee, C. J. Xue, J.-M. Wu, and N. Guan, "Mixed-criticality federated scheduling for relaxed-deadline dag tasks," in *2024 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2024, pp. 362–374.

[42] Z. Li, A. Samanta, Y. Li, A. Soltoggio, H. Kim, and C. Liu, "\mathrm {R}^{3}: On-device real-time deep reinforcement learning for autonomous robotics," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 131–144.

[43] M. Liu, P. Lu, X. Chen, O. Sokolsky, I. Lee, and F. Kong, "Deadline-safe reach-avoid control synthesis for cyber-physical systems with reinforcement learning," in *2024 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2024, pp. 96–108.

[44] S. Shirvani, A. Samanta, Z. Li, and C. Liu, "Duojoule: Accurate on-device deep reinforcement learning for energy and timeliness," in *2024 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2024, pp. 109–122.

[45] J. Liu, H. Huang, C. Wang, R. Li, T. Car, Q. Yang, and Z. Zheng, "Can federated learning clients be lightweight? a plug-and-play symmetric conversion module," in *2024 IEEE 44th Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 2024, pp. 809–820.

[46] M. Zhang, H. Zhao, S. Ebron, R. Xie, and K. Yang, "Ensuring fairness in federated learning services: Innovative approaches to client selection, scheduling, and rewards," in *2024 IEEE 44th Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 2024, pp. 762–773.

[47] S. Zhang, Z. Zheng, Q. Li, F. Wu, and G. Chen, "Mobility-aware device sampling for statistical heterogeneity in hierarchical federated learning," in *2024 IEEE 44th Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 2024, pp. 656–667.

[48] M. Hu, Z. Xia, D. Yan, Z. Yue, J. Xia, Y. Huang, Y. Liu, and M. Chen, "Gitfl: Uncertainty-aware real-time asynchronous federated learning using version control," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 145–157.

[49] Y. Deng, S. Yue, T. Wang, G. Wang, J. Ren, and Y. Zhang, "Fedinc: An exemplar-free continual federated learning framework with small labeled data," in *Procs. of the 21st ACM SenSys*, 2023, pp. 56–69.

[50] X. Yu, A. Thomas, I. G. Moreno, L. Gutierrez, and T. S. Rosing, "Intelligence beyond the edge using hyperdimensional computing," in *2024 23rd ACM/IEEE Int. Conf. on Information Processing in Sensor Networks (IPSN)*, 2024, pp. 1–13.

[51] Y. D. Kwon, J. Chauhan, H. Jia, S. I. Venieris, and C. Mascolo, "Life-learner: Hardware-aware meta continual learning system for embedded computing platforms," in *Procs. of the 21st ACM Conf. on Embedded Networked Sensor Systems*, 2023, pp. 138–151.