OctoCross: Workload-Aware Request Offloading Scheduling in Cross-Camera Collaboration

Jinghan Cheng, Thanh-Tung Nguyen, Lucas Liebe, Yuheng Wu, Nhat-Quang Tau, Pablo Espinosa, Dongman Lee

Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea

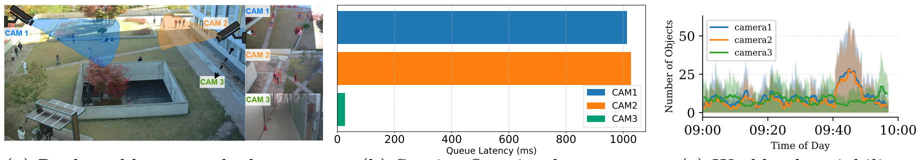
Abstract. Cross-camera collaboration is emerging as a crucial strategy to support dynamic and heterogeneous real-world workloads in real-time video analytics (VA) systems. However, existing offloading methods lack the capability to jointly model spatial and temporal dynamics, which limits their offloading accuracy, leading to low system performances. We propose OctoCross, a spatiotemporal offloading framework for request offloading in mutli-camera VA systems. OctoCross introduces a novel spatiotemporal model that can jointly learns spatial and temporal dynamics to produce an actionable offloading plan in an end-to-end manner. Extensive evaluations on real-world multi-camera datasets show that OctoCross achieves up to **5.8**× higher throughput and **3.2**× lower latency compared to baselines, while maintaining high SLO compliance across diverse deployment scenarios.Our code is available at `https://github.com/tungngreen/PipelineScheduler/tree/OctoCross-ICSOC2025`.

Keywords: Cross-camera collaboration · Spatio-temporal Scheduling · Task offloading · Visual Analytics.

1 Introduction

Real-time video analytics (VA) is crucial to smart city services like traffic control and surveillance [4] in which camera streams are streamed to embedded edge devices and processed locally for quick real-time analysis results [1]. To support complex applications, the concepts of pipelines are introduced where processing functions and DNN models are arranged as cascade directed acrylic graphs (DAGs) of microservices [1,20,12] instead of one monolithic executable.

This design significantly increases flexibility by enabling collaboration between embedded *edge devices* and edge server [12,4,21], and more recently among the edge devices themselves [20,7,9]. Particularly, an edge device with insufficient computational resources to process its local workload can offload incoming requests to nearby devices hosting the same microservice, thereby alleviating its burden. However, arbitrarily offloading requests without considering *where* and *when* to redirect them leads to suboptimal performance, manifesting as increased latency and imbalanced load distribution. Effective offloading, therefore, necessitates an understanding of the latent spatio-temporal correlations among



(a) Real-world camera deployment (b) Service Queuing latency (c) Workload variability

Fig. 1: Real-world examples of spatiotemporal workload variability.

devices [23]. These correlations are not merely geographical; they are fundamentally about the relationship between workload fluctuations and resource demands over time. A workload spike in the camera stream of one device, for instance, often precedes a similar surge at a neighbor within a predictable, albeit uncertain, time frame. Capturing this predictive relationship is the key to making informed and efficient offloading decisions. For instance, Figure 1a shows 3 cameras on a university campus with different queuing latencies. During lecture breaks, Cam 1 and 2's devices are currently experiencing high latency (Figure 1b) due to the surge of detected human objects (Figure 1c) in their streams. A few minutes later, before the lectures start again, Cam 3 overlooking the hallway experiences a surge while 1 and 2 now have low workload because by students are returning to the class rooms. Without careful offload scheduling, each individual camera will be under high load which leads to high processing latency, degraded analytics accuracy, and even crashes. On the contrary, adaptive collaboration via by capturing cross-camera spatio-temporal correlations enables offload scheduling so that a low-load neighbor device can help absorb the impact.

Recently, several scheduling approaches [20,7,9,3] have been proposed for cross-camera workload balancing and offloading. However, they exhibit critical limitations in workload-aware spatio-temporal modeling, as they fail to fully capture workload variability jointly across space and time. Specifically, Distream [20] and GNN_LSTM [9] rely on LSTM models to learn temporal correlations after first modeling spatial relationships—using a heuristic algorithm in the former and a GNN in the latter. In contrast, EdgeVision [3] focuses solely on spatial relationships derived from transmission bandwidths, overlooking the context-dependent co-occurrence of resource demands between devices.

Observations. The above approaches treat spatial and temporal dynamics in isolation or only partially, leading to suboptimal suboptimal offloading and resource allocation. ***A workload-aware offloading scheduling algorithm needs to consider both spatial and temporal information jointly, which enables capturing complex interdependencies and dynamic relationships often missed otherwise***. Joint learning benefits from end-to-end optimization for global performance by training with a single loss function [22].

To enable efficient joint learning, we identify the following **key challenges:**

(Challenge 1) Joint modeling of spatio-temporal relationships. Identifying features that effectively capture spatial and temporal dependencies is challenging. Prior work has focused on predicting future workloads of individual nodes [20,9]. However extending this to interacting nodes demands fusing temporal dynamics with spatial information that accounts for physical and communication topologies, which substantially increases the problem complexity.

(Challenge 2) Inferring the scheduling decisions. Even with meaningful spatio-temporal features, incorporating them into real-time scheduling remains challenging due to their high dimensionality, temporal variability, and the need for low-latency decision-making. These features may evolve rapidly and differ across nodes, making it difficult to design a strategy that adapts effectively under dynamic workloads.

(Challenge 3) Adaptability and Scalability. In dynamic Edge environments, nodes may be added or removed frequently due to changing conditions, such as congestion or intermittent connectivity. To remain effective, the system must support a flexible architecture that can quickly reflect these fluctuations in the spatial-temporal model as well as in the offloading decision making process. However, achieving this level of adaptability and scalability remains a significant and ongoing challenge.

To address these challenges, we propose OctoCross, an end-to-end cross-camera offloading scheduling system for real-time Edge VA. The foundation of our system is a novel that jointly captures spatial and temporal relationships using an attention-based mechanism to predict system-wide workload fluctuations. We design a heuristic algorithm leveraging these predictions to proactively generate scheduling decisions, avoiding reactive adjustments. To tackle the dynamic nature of Edge environments, we introduce two adaptation mechanisms. First, a masking structure enables rapid, on-the-fly adjustments to the offloading schedule when nodes become unavailable. Second, a lightweight continuous fine-tuning protocol adapts the underlying spatio-temporal model to evolving workload patterns and changes in network topology, ensuring long-term predictive accuracy. The synergy of proactive scheduling and dynamic adaptation significantly reduces decision latency, alleviates queuing delays, and lowers internode communication overhead. Consequently, OctoCross achieves efficient, real-time video analytics while maintaining low scheduling complexity. Extensive evaluations on real-world heterogeneous smart cameras show that OctoCross achieves up to 5.8× higher throughput and 3.2× lower latency compared to baselines, while maintaining high SLO compliance across diverse deployment scenarios.

2 OctoCross's Design

Overview. As illustrated in Fig. 2, the overall architecture of OctoCross consists of a central server and a set of distributed edge devices, each connected to a real-time camera stream. The server, built on consumer-grade hardware, hosts the control plane, acting as a centralized controller that manages cluster-wide functions such as scheduling. On the data plane, each embedded edge device runs containerized microservices, which include processing functions and DNN models to analyze its local video stream. When a device reaches full resource utilization due to high workload, it can offload tasks to neighboring devices—or, in the *worst case*, to the server—to maintain timely analysis.

Detailed Workflow. Conceptually, a cycle of OctoCross's operations starts **S** with the video analytics microservices' being deployed at the devices

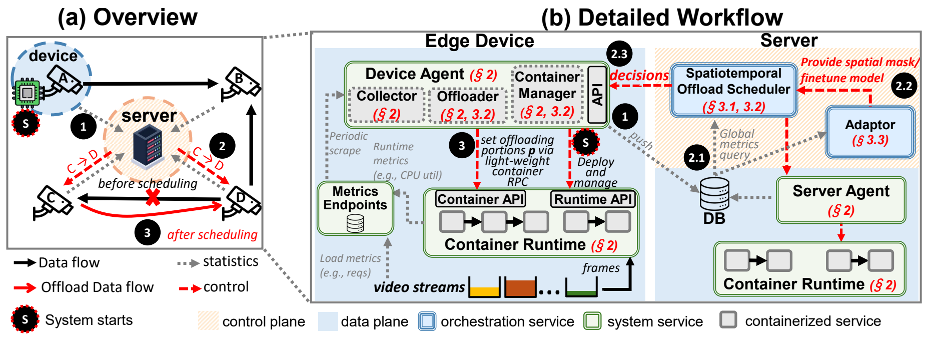


Fig. 2: OctoCross's Architecture

by the *Device Agent* and then run by *Container Runtime*(e.g., Docker, Containerd). During run time, 1 metrics such as request rates, CPU utilization, and memory usage are published to the *Metrics Endpoints*, scraped periodically and reported to the *Database* by *Device Agent*.

At the server, 2.1 the global metrics are queried to serve the *Offload Scheduler* and the *Adaptor*. The *Offload Scheduler* uses composed of a *Spatiotemporal Offloading model* and a scheduling algorithm, running periodically to determine the best offloading configurations within the cluster based on the newly collected workload patterns. *Adaptor* constantly monitors the global metrics for significant changes in the system topology. Particularly, when a device is disconnected, the *Adaptor* 2.2 constructs a spatial mask to allows the *Offload Scheduler* to temporarily ignore the device in its offloading decisions. On the contrary, when new devices are installed, the *Adaptor* collects and combines their workloads and runtime metrics with other devices. In both cases, the *Adaptor* then reflects the changes in the *Spatiotemporal Offloading model* by performing light-weight finetuning using newly collected workloads and runtime metrics. The offloading decisions of *Offload Scheduler* are 2.3 delivered to each *Device Agent* as well as the *Server Agent*—the server counter part of the *Device Agent*. These decisions 3 are conveyed to containers by the *Device Agent* via a lightweight container RPC API, after which portions of requests of the overloaded devices are redirected to other devices with available resources.

3 Spatiotemporal Offloading Scheduler

To support proactive and context-aware offloading in dynamic environments, we design a end-to-end *Spatiotemporal Offloading model* that captures evolving spatiotemporal workload dynamics among distributed devices (subsection 3.1) and an scheduling algorithm to transform the results of the model into the final offloading decisions (subsection 3.2). Lastly, we introduce an adaptation mechanism to tackle constant changes in the dynamic edge environments (subsection 3.3). The notations used in this section are summarized in Table 1.

3.1 Spatio-Temporal Model Architecture

As shown in Fig. 3, we formalize task offloading as a mapping function $\mathrm{F} : \mathbb{R}^{N\times\tau},\ \mathbb{R}^{N\times N} \rightarrow \mathbb{R}^{N\times N}$, which converts adjacency matrices A_m and historical workload H of the cluster into an offloading matrix M, where M_{ij} indicates

Table 1: Notation summary of spatiotemporal scheduling model.

Notation	Description	Notation	Description
τ	Number of sampled timestamps	d	Embedding dimension for spatial and temporal representations
N	Number of devices (nodes)		
$H \in \mathbb{R}^{N\times\tau}$	Historical workload matrix	$\hat{H} \in \mathbb{R}^{N\times\tau}$	Predicted future workload
$A_m \in \mathbb{R}^{N\times N}$	Adjacency matrix of m (physical, network, and resource-based)	$M \in \mathbb{R}^{N\times N}$	Offloading matrix; M_{ij} is the proportion offloaded (device i to j)
ϕ_{spatial}	Learnable spatial embedding	ϕ_{temporal}	Learnable workload embedding
$S \in \mathbb{R}^{N\times d}$	Cluster spatial embeddings	$T \in \mathbb{R}^{\tau\times d}$	Temporal workload embeddings
$Z^{(l)} \in \mathbb{R}^{N\times\tau\times d}$	STAttBlock tensor at layer l	$A_t(\cdot), A_s(\cdot)$	Temporal and spatial attention
$E_{\text{st}} \in \mathbb{R}^{N\times d}$	Combined representations		

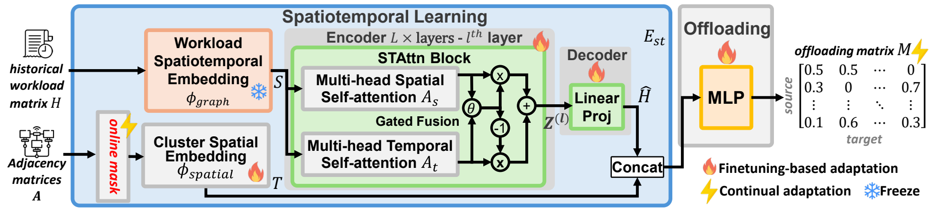

historical workload matrix H
Adjacency matrices A
online mask
Spatiotemporal Learning
Workload Spatiotemporal Embedding ϕ_{graph}
Cluster Spatial Embedding $\phi_{spatial}$
S
T
Encoder $L \times$ layers - l^{th} layer
STAttn Block
Multi-head Spatial Self-attention A_s
Multi-head Temporal Self-attention A_t
Gated Fusion
θ
-1
$Z^{(l)}$
Decoder
Linear Proj
E_{st}
$\hat{H}$
Concat
Offloading
MLP
offloading matrix M
source
target
0.5 0.5 ··· 0
0.3 0 ··· 0.7
0.1 0.6 ··· 0.3
Finetuning-based adaptation
Continual adaptation
Freeze

Fig. 3: Model architecture for Spatiotemporal Learning and Offloading

the proportion of tasks offloaded from device i to j. Achieving this mapping requires not only learning meaningful spatiotemporal patterns, but also effectively translating these patterns into actionable offloading proportions across devices

Input. For the *temporal* input, we construct a historical workload matrix $H \in \mathbb{R}^{N\times\tau}$ using the most recent τ workload samples per device.

For the *spatial* input, it is important to incorporate multiple topology views of the cluster, including: (1) the physical-space topology, capturing the physical layout (e.g., direct paths from the lobby to the yard); (2) the network topology, reflecting potential offloading latency; and (3) the resource availability topology, indicating whether target devices have capacity to handle new requests. Each topology is represented by an adjacency matrix $A \in \mathbb{R}^{N\times N}$.

Workload Spatiotemporal Embedding. To capture *spatiotemporal* correlations of workload variations within the cluster, we first embed historical workload samples into a d-dimensional space. Our goal is to transform H into an $N \times d$ device embedding matrix S by performing an embedding from $\mathbb{R}^N \to \mathbb{R}^d$, condensing the workload samples of each device into a latent dimension d to enable downstream modules to reason over spatial dependencies more effectively. This also transforms the workload matrix into a sequence, enabling quick adaptation to changes in number of devices in the cluster. A direct projection is computationally expensive for large matrices, which would reduce the system's scalability. Instead, we use a more efficient method. We first learn a linear map that summarizes each device's connectivity into a single value ($\mathbb{R}^N \to \mathbb{R}^1$) and then project this summary into the d-dimensional space. This drastically reduces the number of operations from $O(N \times d)$ to $O(N + d)$.

Encoder. The encoder is composed of L sequential *Spatiotemporal Multi-head Self-Attention Block* (STAttn Block). Each block contains a Spatial Attention A_S and a Temporal Attention A_t module both using the spatiotemporal embedding S as the input. The self-attention mechanism [17] was originally proposed for natural language processing. It computes a probability distribution over the input sequence to measure how much each word relates to others, allowing the model to dynamically focus on the most relevant parts when generating representations. Inspired by this, we apply self-attention to estimate each device's influence on workload changes across the cluster. Each self-attention module uses 8 heads, with each head capturing a distinct contextual pattern. For example, after a class, most students may exit through the front door, while others head to the basement to retrieve vehicles. A spatial attention head learns such patterns by estimating how likely objects seen by one camera are to appear on neighboring devices. These movements cause workload fluctuations, which temporal heads capture by attending to historical workload patterns.

After computing spatial and temporal attention outputs, it is important to combine them based on their contextual relevance. This is achieved through a gated fusion mechanism that adaptively balances their contributions. For instance, if the back entrance is closed during a special event, temporal patterns may become unreliable. The gate learns to down-weight temporal features and emphasize spatial cues—like redirected foot traffic—allowing the system to adjust in real time. The final encoder output is denoted as Z^L, carrying the fused spatiotemporal representations to the next stage.

Decoder. The original architecture [23] uses a multi-layer decoder with cross attention, which is computationally heavy and increases inference latency. Instead, we devise a lightweight linear-projection, which directly map the encoded spatiotemporal representations into the predicted workloads. This direct mapping not only reduces computational overhead but also simplifies the learning objective by encouraging the encoder to produce task-relevant representations, leading to more stable training and faster convergence. The output matrix $\hat{H}$ contains the predicted workloads of each nodes for the next τ timestamps.

Cluster Spatial Embedding. Before spatial embedding, each adjacency matrix is element-wise multiplied with a binary mask constructed at runtime to exclude unavailable nodes from future computation. To capture *spatial* correlations across physical-space, network, and resource availability topologies, we embed each adjacency matrix into an $N \times d$ latent space, summarizing each device's relationships with its neighbors in dimension d. Similar to the workload spatiotemporal embedding, this transforms adjacency matrices into variable-length sequences, enabling fast adaptation to dynamic cluster sizes. We adopt a similar embedding mechanism for computational efficiency.

To encode absolute device positions, we add a learnable $N \times d$ positional encoding to the structural embeddings. The resulting spatial embedding matrix S encodes both multi-view topology information and their influence on spatial correlations across devices.

Offloading head. The predicted load $\hat{H}$ is then concatenated with a spatiotemporal feature matrix S and passed through a residual network to produce the base offloading score matrix $M = \mathrm{MLP}([\hat{H}, T]) \in \mathbb{R}^{N \times N}$. A online binary mask is applied to ensure that offloading only occurs from overloaded to un-

Algorithm 1: Workload-Aware Offloading Scheduling

```
1  Function Main():
2      for every second do
3          if graph $\mathcal{G}$ has been updated then
4              $M' \leftarrow$ Online_Adaptation()
5          Sample workload $\rightarrow H$
6          Sample resources $\rightarrow A_m$
7      for every 15 seconds or nodes overloaded now do
8          $M \leftarrow Model(H_{t-32:t}, A_m, \theta)$
9          Execute $M$
10 Function Online_Adaptation():
11     if unavailable devices then
12         Update $Mask$
13         Wait $GracePeriod$
14     Offline_Finetune ()
15 Function Offline_Finetune():
16     Collect data
17     Finetune $Model$
```

derloaded nodes along valid edges. An iterative projection algorithm adjusts M based on the available capacities of each device, clamping the offloaded portions within a threshold θ to avoid over-generous offloading, which causes resource shortage at the offloaded target devices.

3.2 Scheduling Algorithm

OctoCross's scheduling algorithm To utilize the outputs of the spatiotemporal prediction model, OctoCross executes a scheduling loop that leverages the model-generated offloading matrix M to make workload transfer decisions in real time. As shown in Algorithm 1, the system continuously monitors the environment and performs scheduling at regular intervals. Every second, it samples the current workload statistics and updates the adjacency matrix A_m to reflect resource conditions and network connectivity (lines 5-6). If the underlying graph topology changes—such as when devices are added or become unavailable—the system triggers the `Online_Adaptation` procedure (line 3-4) to update Spatial $Mask$ and prepare for model finetuning.

The core scheduling logic runs every 15 seconds or when any node becomes overloaded (line 7). At each scheduling event, the system encodes the past 32 seconds ($\tau = 32$) of workload data $H_{t-32:t}$, which provides sufficient temporal context for accurate predictions without incurring excessive overhead. OctoCross then runs Spatiotemporal Offloading model to generate an offloading decision matrix M (line 8). The final offloading schedule is then sent to each *Device Agent*, which relays the information to the corresponding containers via a light RPC container API (line 9).

3.3 Continual Adaptation

To support scalable deployment in dynamic environments, OctoCross integrates both online adaptation and offline finetuning mechanisms to ensure consistent performance as the platform topology evolves.

Online Adaptation. OctoCross continuously monitors node availability and if the topology is altered, it triggers and `Online_Adaptation` (lines 7-12). If there are unavailable devices due to disconnection or crashing, it updates the Spatial $Mask$ to exclude these devices from future computation (line 10). It then waits for a $GradePeriod$ of 150 seconds to see whether these nodes are back online (line 11). Next, whether existing devices are unavailable or new ones are added, OctoCross still performs `Offline_Finetune` to reflect the long-term changes (line 14).

Offline Finetuning. To facilitate model finetuning, OctoCross begins by collecting data for training (line 14). In cases where a device becomes unavailable, historical workloads and associated spatial context—such as network conditions and resource availability—captured during the *GracePeriod* are immediately used as training data. Conversely, when new devices are added to the system, OctoCross gathers fresh data by sampling over a 48-hour period.

Since finetuning may be performed regularly, we design an efficient strategy that keeps the spatiotemporal representations of existing or unaffected nodes frozen. This significantly reduces computational overhead. Moreover, unlike full training, finetuning requires only a small amount of data, as the model already possesses a learned structure. Importantly, training labels are automatically extracted from historical workloads, eliminating the need for manual annotation.

In our experiments, finetuning over 1,000 epochs completes in under 30 seconds while preserving high prediction accuracy.

4 Implementation Details

Testbed. Our real-world testbed consists of a consumer-grade edge server equipped with an NVIDIA GTX 1080 GPU and 16 heterogeneous Jetson devices, including 3 Xavier AGXs, 5 Xavier NXs, 3 Orin NXs, 3 Orin Nanos, and 1 Orin AGX. Each device is connected to a real-time camera stream. To support larger-scale experiments, we additionally simulate 10 edge devices on a consumer-grade server with an RTX 3090 GPU.

Implementations. We implement OctoCross based on the open-source framework introduced in [12]. The control plane is written in Python, consisting of approximately 4K lines of code for model training, inference, and scheduling. The data plane is developed in C++ with around 20K lines of code, covering the implementation of the *Device Agent* and microservices. The *Device Agent*, running as a standalone process on each device, exposes a gRPC API used by the scheduler mainly to (1) deploy microservices and (2) configure offloading policies. Docker serves as the *Container Runtime*, with the *Device Agent* interfacing via the Docker API. Our system can also be integrated with other container runtimes (e.g., Containerd) and orchestration tools (e.g., Kubernetes, OpenShift) with minimal modification. Our inference microservice images are built on the official `cuda` base and include TensorRT for efficient inference. As pipelines process live video streams, we use gRPC for low-overhead inter-microservice communication. Each microservice follows a multi-threaded `preprocess` $\rightarrow$ `inference` $\rightarrow$ `postprocess` structure to maximize throughput via concurrency, though simpler architectures are also supported.

Inside each container, we devise a lightweight load balancing mechanism with a fixed-policy round-robin strategy, where the proportion of requests offloaded to each target is configured by the *Device Agent* via a lightweight RPC API similar to [2]. Runtime and workload metrics are collected via Docker, CUDA, and JetPack APIs, and pushed to a cluster-wide PostgreSQL 14 database enhanced with TimescaleDB. To simulate realistic network conditions, we extract and apply LTE bandwidth traces from [15] using the Linux `tc` utility.

Our complete source code and setup instructions are publicly available at: https://github.com/tungngreen/PipelineScheduler/tree/OctoCross-ICSOC2025.

5 Evaluation

We evaluate OctoCross through a real-world deployment covering 1) heterogeneous edge hardware, 2) multiple surveillance environments, 3) increasing camera scales, and 4) competitive baselines and ablation studies. This evaluation reveals the following key findings:

1) OctoCross scales efficiently, achieving up to **5.68**× improvement over non-collaborative baselines across campus and city deployments.

2) OctoCross is latency-aware and robust, cutting end-to-end and tail delays while keeping median latency stable under bursty loads.

3) OctoCross benefits from joint spatio-temporal modeling: ablating spatial or temporal components drastically degrades performance.

5.1 Experiment Setups

Baselines. We compare OctoCross against two state-of-the-arts offloading strategies and a no-offload baseline. (1) Distream [20] uses LSTM to forecast workload and performs offloading based on queue lengths and bandwidth estimation. (2) GNN_LSTM [9] extends this by modeling spatial correlations among devices via GNN for coordinated scheduling. (3) *No Offloading* serves as a non-collaborative baseline where all tasks are processed locally.

Dataset. We use two *real-world datasets* to evaluate deployment scenarios.

– *Campus* scenario: The MTMMC dataset [**?**] contains a cluster of 16 RGB 15-FPS cameras covering the surroundings (i.e., back and front yards), the lobby, and 3 floors of a campus building. Our motivation experiments Figure 1 show that there exists strong spatiotemporal correlations among the cameras.

– *City traffic* scenario: The AI City Challenge 2022 [10] provides offers city-scale multi-camera traffic footage at 10 FPS. Its spatiotemporal properties have studied in several works in tracking [5]. We extract 26 longest video streams for our experiments. The lengths of these streams are uneven which display the dynamic nature of Edge environments, which is address in Figure 8.

5.2 Evaluation Metrics

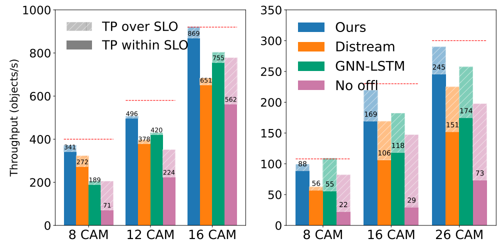
Service Throughput (OPS). Number of targets processed per second across all service stages, reflecting system scalability and efficiency.

End-to-End Latency. Total time from task generation to result delivery, capturing overall system responsiveness.

Latency SLO Miss Rate. Percentage of tasks exceeding the 200ms latency threshold, indicating violations of real-time service requirements.

5.3 Evaluation Results

Consistent Throughput Gains Across Scales and Scenarios. Figure 4 compares the throughput of OctoCross against three common SOTA baselines under different system scales. Evaluations are conducted on two scenarios: *Campus* building setup with small-scale human movements and a *City traffic* setup



(a) *Campus* (b) *City traffic*

Fig. 4: Throughput comparisons in *Campus* and *City traffic* scenarios under different scales. *The dashed red line denotes the ideal throughput.*

Table 2: Throughput gain and latency reduction in *Campus* (upper) and *City traffic* (lower).

	OctoCross Throughput Gain (×)			OctoCross Latency Reduction (×)		
Compared to	8 cams	12 cams	16 cams	8 cams	12 cams	16 cams
Distream	1.25×	1.31×	1.33×	1.19×	1.13×	1.52×
GNN_LSTM	1.08×	1.18×	1.15×	1.47×	1.19×	1.28×
No offloading	4.80×	2.21×	1.54×	3.24×	2.45×	2.71×

	OctoCross Throughput Gain (×)			OctoCross Latency Reduction (×)		
Compared to	8 cams	16 cams	26 cams	8 cams	16 cams	26 cams
Distream	1.57×	1.59×	1.62×	1.16×	1.37×	1.42×
GNN_LSTM	1.60×	1.43×	1.41×	2.36×	1.50×	1.32×
No offloading	4.00×	5.82×	3.36×	2.38×	3.26×	2.47×

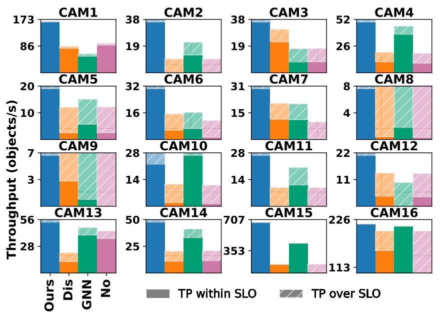
representing large-scale urban environments. By capturing spatiotemporal correlations, OctoCross consistently achieves the highest throughput and scales efficiently, reaching **92.4%** and **89.1%** of ideal throughput in the 16-camera *Campus* and 26-camera *City traffic* settings, respectively.

Shaded bars indicate requests that were successfully processed but violated the latency SLO threshold. The proportion of SLO violation is significantly higher in *City traffic* due to bursty traffic and fast-moving objects, which increase cross-camera transfer and narrow the offloading window. More importantly, traffic scenarios are inherently more complex with vehicles frequently stopping in the section between cameras, rendering the spatiotemporal correlations more difficult to be captured.

Figure 2 shows the throughput gains and latency reductions achieved by OctoCross in the *Campus* and *City Traffic* scenarios. We observe that the throughput gains over both Distream and GNN_LSTM increase as the cluster size grows. This trend can be attributed to the limitations of their 2-stage spatiotemporal modeling approaches, which treat spatial and temporal dependencies separately and sequentially. As the cluster topology becomes more complex, such decoupled models struggle to capture the joint dynamics of workload and resource variations across time and space, leading to suboptimal scheduling decisions. In contrast, OctoCross's unified attention-based architecture enables it to model these dependencies holistically, resulting in more efficient workload redistribution and greater performance gains in larger deployments.

On the other hand, there is no clear trend in latency reduction across different cluster sizes when compared to the baselines. We hypothesize that this is due to the inherent variability in real-time systems: even when the scheduler correctly identifies target devices with available resources, the offloaded requests remain subject to multiple sources of non-determinism. These include fluctuations in network latency, dynamic queuing delays at the inference backends, and contention from concurrent requests—factors that can obscure the impact of improved scheduling alone. As a result, latency improvements may appear inconsistent, even when workload balancing is effectively achieved.

Per-Camera Contributions and Overall System Latency Analysis. Figure 5 and 6 present a detailed per-camera analysis for both the *Campus* and *City Traffic* scenarios. Consistent with earlier results, OctoCross gener-


(a) Per-camera throughput comparisons

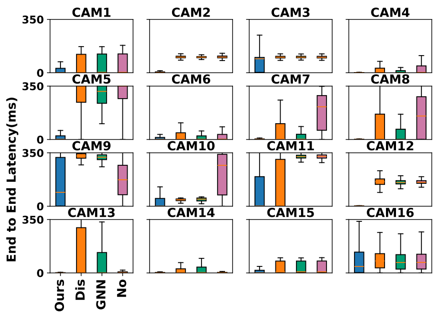
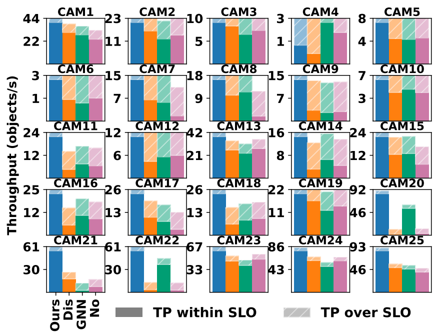

(b) Per-camera latency comparisons.

Fig. 5: Per-camera detailed analysis in full 16-cam *Campus* scenario.


(a) Per-camera throughput comparisons

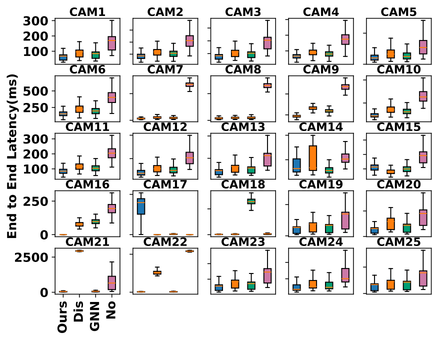

(b) Per-camera latency comparisons.

Fig. 6: Per-camera detailed analysis in full 26-cam *City traffic* scenario

ally outperforms all baselines across the majority of cameras, with a few notable exceptions—such as **CAM10** in Figure 5a, **CAM4** in 6a, and **CAM17** in Figure 5b. These exceptions can be attributed to the baselines' tendency toward local optimization (e.g., greedy algorithm), driven by their limited ability to capture global spatiotemporal correlations. In contrast, OctoCross leverages a unified attention-based architecture that enables global optimization of workload distribution across the cluster. However, this global perspective can sometimes lead to locally suboptimal decisions for specific nodes, particularly when their individual behavior diverges from the broader traffic pattern captured by the model. We noticed that this behavior of OctoCross becomes more noticeable without clamping the offload portions to a device.

Training Convergence. Figure 7 illustrates the loss convergence of the spatiotemporal model during training in two scenarios with 16 and 26 cameras, respectively. OctoCross's model consistently converges within a similar number of epochs, demonstrating stable and efficient learning behavior regardless of cluster size. Notably, due to the model's lightweight architecture, training for 3,000 epochs completes in approximately 200 seconds on an NVIDIA RTX 3090 GPU. In practice, however, full training is rarely needed and only fast finetuning is typically required, which we will discuss shortly.

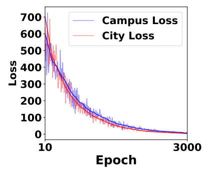
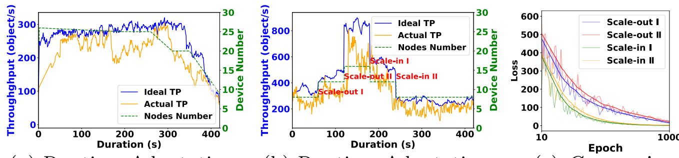


Fig. 7: Full training process



(a) Runtime Adaptations of *City* (b) Runtime Adaptations of *Campus* (c) *Campus's* finetuning loss

Fig. 8: Runtime Adaptation in real-world settings

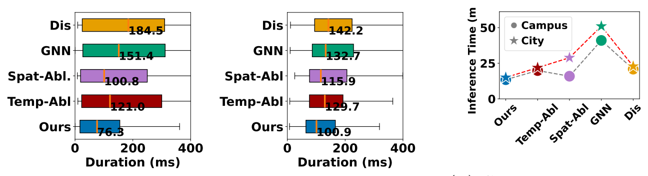
Runtime Adaptation. Figure 8a and 8b illustrate how OctoCross adapts to dynamic changes in cluster topology. In the *City Traffic* scenario, natural variations in stream durations result in cameras gradually becoming unavailable. In contrast, the *Campus* scenario introduces controlled changes, where 4 devices are added or removed at a time.

OctoCross handles both types of changes seamlessly, without requiring full system reconfiguration or retraining. This marks a significant improvement over baselines such as GNN_LSTM, which assume a fixed graph structure; any topology change in those models necessitates reconstructing the graph and retraining from scratch. OctoCross avoids this limitation through its masking mechanism and embedding modules. Specifically, the embedding modules convert the input matrices into variable-length sequences, enabling the model to flexibly adjust to changes in the number of active devices. Meanwhile, the masking mechanism instantly excludes unavailable nodes from future computation, ensuring graceful degradation and continuity in scheduling.

In the *Campus* experiment, we simulate two scale-out events that effectively double the number of devices. Although OctoCross typically collects data from new devices before performing finetuning, in this test we skip the collection period to focus on runtime adaptation. Even under these conditions, the finetuned model adapts well to the expanded workload. Figure 8c shows the loss convergence during four finetuning episodes in the *Campus* scenario. While all instances exhibit similar final performance, the convergence curves for scale-out events are smoother—suggesting that the model generalizes better when scaling up, possibly due to richer spatial patterns and more diverse workloads introduced by the additional devices.

Ablation Study. We conduct an ablation study by (*Temp-Abl*) removing our Workload Temporal Embedding module and use a separate LSTM model similar to the baselines and (*Spat-Abl*) replacing our Cluster Spatial Embedding with Distream's greedy algorithm. Figure 9a and b show *Temp-Abl* and *Spat-Abl* still outperforms the baselines but there are sigfinicant drops compared with the full OctoCross. For *Spat-Abl*, with a greedy algorithm for spatial exploration, the system tends to make local optimization. For *Temp-Abl*, the separate LSTM model cannot generalize well to diverse workloads, leading to higher susceptibility to workload bursts.

As shown in Figure 9c, using an end-to-end model is also more efficient in terms of scheduling time, as it eliminates intermediate processing steps such as separate spatial and temporal modeling—allowing scheduling decisions to be computed in a single forward pass. This design fully leverages GPU par-



(a) Queuing latency of *Campus* (left) and *City traffic* (right)

(b) Scheduling overhead

Fig. 9: Ablation Study

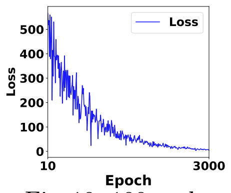


Fig. 10: 100-node large scale training

allelism, resulting in minimal added latency even as the number of devices increases. In contrast, *GNN_LSTM* incurs higher overhead because both of its modeling stages execute on the GPU. Since GPU–CPU communication is costly, invoking these stages separately introduces significant overhead due to repeated data transfers. Meanwhile, *Distream* suffers from latency because it sequentially traverses all node workloads during scheduling, making it inherently slower—especially as cluster size grows.

6 Discussion

Scalability and Generalizability. OctoCross demonstrates strong scalability through its lightweight attention-based architecture and online runtime masking, enabling low-latency scheduling even in dynamic 100-node deployments (Figure 10). Although this paper focuses on applying OctoCross to video analytics inference services, its design holds promise for generalization to other domains. The core spatiotemporal learning mechanism and offloading scheduling algorithm operate on general system-level signals—such as request rates, network conditions, and resource availability—that are commonly available across services and platforms. Overall, we envision OctoCross serving as a generalized, scalable solution across diverse real-time distributed applications.

Future Work. However, we recognize a limitation in our current implementation. Specifically, we adopt the contain implementation from [2], in which each container exposes a lightweight RPC API to communicate with the control plane. Through this API, containers receive offloading commands and self-adjust their outbound workloads accordingly. While effective, this approach assumes that all containers are instrumented to support such APIs—a constraint that may not hold in general-purpose environments. In future work, we plan to integrate OctoCross with Kubernetes, leveraging service objects that operate independently of individual containers. This would enable more flexible and efficient load balancing without imposing assumptions on the container internals, making OctoCross more broadly generalizable in heterogeneous infrastructures.

7 Related Works

Distributed Workload Orchestration and Offloading. Efficient distributed task execution leverages diverse offloading strategies to optimize energy, cost, and QoS [6], including decentralized reactive methods for dynamic online scenarios [14]. AI-driven video analytics techniques address offloading and partitioning of collaborative multi-DNN applications [19], schedule DNN inference on heterogeneous processors by predicting workload patterns [18], and apply reinforcement learning for dynamic task management [24]. While foundational, these works do not fully address the complexity of joint spatio-temporal modeling required for advanced collaborative intelligence.

Collaborative Systems for Spatiotemporal Video Analytics. Several works exploit spatiotemporal correlations in dynamic environments [16,13,8,11]. Recent systems focus on cross-camera video analytics: Distream [20] provides a workload-adaptive framework, EdgeVision [3] supports cooperative task placement and model selection, and GNN_LSTM [9] uses GNNs and LSTMs for workload prediction. GASTO [7] adopts adaptive graph-based learning for offloading. However, limitations persist: Distream and GNN_LSTM model space and time sequentially, and EdgeVision mainly captures spatial context. OctoCross jointly models spatio-temporal patterns via graph-based attention, enabling adaptive, workload-sensitive offloading under dynamic conditions.

8 Conclusion

We propose OctoCross, a spatiotemporal scheduling framework that enables adaptive task offloading in cross-camera systems. By leveraging graph-based modeling and attention-driven coordination, it proactively captures dynamic workload patterns for efficient, context-aware scheduling. Real-world results show up to **5.8**× higher throughput and **3.2**× lower latency compared to non-collaborative baselines. OctoCross offers a practical foundation for scalable, responsive video analytics, with future work exploring stage-level scheduling and multi-pipeline orchestration.

References

1. Microsoft rocket for live video analytics (2020), `https://www.microsoft.com/en-us/research/project/live-video-analytics/`
2. Crankshaw, D., Wang, X., Zhou, G., Franklin, M.J., Gonzalez, J.E., Stoica, I.: Clipper: A Low-Latency online prediction serving system. In: 14th USENIX Symposium on Networked Systems Design and Implementation. pp. 613–627 (2017)
3. Gao, G., Dong, Y., Wang, R., Zhou, X.: EdgeVision: Towards Collaborative Video Analytics on Distributed Edges for Performance Maximization. IEEE Transactions on Multimedia **26**, 9083–9094 (2024)
4. Hung, C.C., Ananthanarayanan, G., Bodik, P., Golubchik, L., Yu, M., Bahl, P., Philipose, M.: VideoEdge: Processing camera streams using hierarchical clusters. 3rd ACM/IEEE Symposium on Edge Computing pp. 115–131 (2018)
5. Li, F., Wang, Z., Nie, D., Zhang, S., Jiang, X., Zhao, X., Hu, P.: Multi-camera vehicle tracking system for ai city challenge 2022. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 3265–3273 (2022)
6. Li, H., Dong, C., Wen, W.: Energy-efficient task offloading with statistic qos constraint through multi-level sleep mode in ultra-dense network. In: Int. Conf. on Service-Oriented Computing (ICSOC). pp. 378–392. Springer (2023)
7. Li, Y., Li, J., Lv, Z., Li, H., Wang, Y., Xu, Z.: Gasto: A fast adaptive graph learning framework for edge computing empowered task offloading. IEEE Transactions on Network and Service Management, vol. 20, no. 2, pp. 932–944 (2023)
8. Marchese, A., Tomarchio, O.: Extending the kubernetes platform with network-aware scheduling capabilities. In: Int. Conf. on Service-Oriented Computing (ICSOC). pp. 465–480. Springer (2022)

9. Miao, W., Zeng, Z., Zhang, M., Quan, S., Zhang, Z., Li, S., Zhang, L., Sun, Q.: Workload prediction in edge computing based on graph neural network. In: 2021 IEEE ISPA. pp. 1663–1666. IEEE (2021)
10. Naphade, M., Wang, S., Anastasiu, D.C., Tang, Z., Chang, M., Yao, Y., Zheng, L., Rahman, M.S., Venkatachalapathy, A., Sharma, A., Feng, Q., Ablavsky, V., Sclaroff, S., Chakraborty, P., Li, A., Li, S., Chellappa, R.: The 6th ai city challenge. In: 2022 IEEE/CVF CVPR Workshops. pp. 3346–3355 (June 2022)
11. Nguyen, T.T., Jang, S.Y., Kostadinov, B., Lee, D.: PreActo: Efficient Cross-Camera Object Tracking System in Video Analytics Edge Computing. In: IEEE Int. Conf. on Pervasive Comput. and Comm. (PerCom). pp. 101–110 (2023)
12. Nguyen, T.T., Liebe, L., Tau, N.Q., Wu, Y., Cheng, J., Lee, D.: OctopInf: Workload-Aware Inference Serving for Edge Video Analytics. In: 2025 IEEE Int. Conf. on Pervasive Computing and Communications (PerCom). pp. 128–137 (2025)
13. Panda, S.P., Ray, K., Banerjee, A.: Service allocation/placement in multi-access edge computing with workload fluctuations. In: Proceedings of 19th Int. Conf. on Service-Oriented Computing (ICSOC). pp. 747–755 (2021)
14. Peng, Q., Xia, Y., Wang, Y., Wu, C., Luo, X., Lee, J.: A decentralized reactive approach to online task offloading in mobile edge computing environments. In: Int. Conf. on Service-Oriented Computing (ICSOC). pp. 232–247. Springer (2020)
15. Raca, D., Leahy, D., Sreenan, C.J., Quinlan, J.J.: Beyond throughput, the next generation: A 5g dataset with channel and context metrics. In: Procs. of the 11th ACM multimedia systems Conf. pp. 303–308 (2020)
16. Sedlak, B., Morichetta, A., Wang, Y., Fei, Y., Wang, L., Dustdar, S., Qu, X.: Slo-aware task offloading within collaborative vehicle platoons. In: Int. Conf. on Service-Oriented Computing (ICSOC). pp. 72–86. Springer (2024)
17. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, u., Polosukhin, I.: Attention is All you Need. In: Advances in Neural Information Processing Systems (2017)
18. Xu, D., Li, Q., Xu, M., Huang, K., Huang, G., Wang, S., Jin, X., Ma, Y., Liu, X.: Niagara: Scheduling dnn inference services on heterogeneous edge processors. In: Int. Conf. on Service-Oriented Computing (ICSOC). pp. 67–85. Springer (2023)
19. Yang, Z., He, X., Wang, T., Wang, Z.: An energy-efficient partition and offloading method for multi-dnn applications in edge-end collaboration environments. In: Int. Conf. on Service-Oriented Computing (ICSOC). pp. 54–68. Springer (2024)
20. Zeng, X., Fang, B., Shen, H., et al.: Distream: scaling live video analytics with workload-adaptive distributed edge intelligence (2020), proceedings of the 18th Conf. on Embedded Networked Sensor Systems, ACM, pp. 409–421, Japan
21. Zhang, H., Ananthanarayanan, G., Bodik, P., Philipose, M., Bahl, P., Freedman, M.J.: Live video analytics at scale with approximation and delay-tolerance. In: Proceedings of the 14th USENIX NSDI. p. 377–392. USA (2017)
22. Zhang, J., Zheng, Y., Qi, D.: Deep Spatio-Temporal Residual Networks for City-wide Crowd Flows Prediction. In: Proceedings of the AAAI Conf. on Artificial Intelligence. vol. 31 (Feb 2017)
23. Zheng, C., Fan, X., Wang, C., Qi, J.: GMAN: A Graph Multi-Attention Network for Traffic Prediction. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 1234–1241 (Apr 2020), number: 01
24. Zhou, B., Cheng, L.: Deep reinforcement learning-based scheduling for same day delivery with a dynamic number of drones. In: Int. Conf. on Service-Oriented Computing (ICSOC). pp. 34–41. Springer (2023)