OsmoticGate

Qian, Bin; Wen, Zhenyu; Tang, Junqi; Yuan, Ye; Zomaya, Albert; Ranjan, Rajiv

DOI:
10.1109/TC.2022.3193630

Citation for published version (Harvard):
Qian, B, Wen, Z, Tang, J, Yuan, Y, Zomaya, A & Ranjan, R 2023, 'OsmoticGate: Adaptive Edge-based Real-time Video Analytics for the Internet of Things', *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 1178-1193. https://doi.org/10.1109/TC.2022.3193630

OsmoticGate: Adaptive Edge-based Real-time Video Analytics for the Internet of Things

Bin Qian, Zhenyu Wen*, Junqi Tang, Ye Yuan, Albert Y. Zomaya, Rajiv Ranjan

Abstract—Edge computing has gained momentum in recent years, and can provide more immediate analysis of streaming video data. However, the edge devices often lack the computing capabilities (processing power, memory) to guarantee reasonable performance (e.g., accuracy, latency, throughput) for complex video analytics tasks. To alleviate this critical problem, the prevalent trend is to offload some video analytics tasks from the edge devices to the cloud. However, existing offloading approaches fail to consider the dynamic nature of the video analytical tasks (e.g., varying encoding format for different video content) and are unable to adapt system dynamics (e.g., varying workload between the edge and the cloud).

To overcome the limitation of existing approaches, we develop an edge-cloud offloading performance model based on the concept of hierarchical queues. The resource constraints (e.g., computing capacity and network bandwidth) of each edge nodes and dynamic edge-cloud network conditions are used to parameterize the performance model. Since finding optimal solutions for the performance model is NP-hard, we develop a two-stage gradient-based algorithm and compare it with some state-of-the-art (SOTA) solutions (e.g., FastVA, DeepDecision, Hill Climbing). Experiments have shown our performance model's advantages and the stability of the proposed offloading approach given different systems (edge-cloud) and video analytics application dynamics.

Index Terms—Video processing, Offloading, Optimization, IoT

1 INTRODUCTION

The adoption of Internet-of-Things (IoT) devices, including set-top boxes, embedded computers, and mobile devices, has increased in the context of video generation, delivery, and processing applications. A case in point, each minute 300 hours of video are uploaded onto Youtube. As noted by ourselves [1] and others [2], deep learning technologies and their offshoots are becoming cornerstone for enabling IoT video analytics applications such as security surveillance, image object tracking, home or industrial building automation.

Although the computing power of IoT devices has improved significantly, more research is required to optimize the execution of deep learning models on the same. For example, there is a need to develop optimization techniques that can balance the IoT devices' resource constraint (e.g., processing power and memory) and deep learning model's complexity and performance (e.g., processing latency, accuracy). HUAWEI Mate 10 pro only has 200MB memory for its *Neural Processing Units* (NPU) which is not enough to run many advanced deep learning models, such as Faster R-CNN [3]. Moreover, NVIDIA Jetson Nano can only process five video frames in each second by using ResNet model [4], far from meeting the requirements of real-time processing. This bottleneck will be dramatically amplified as the IoT data volume and velocity continue to grow exponentially.

Developing an offloading technique to balance the video analytics workload, driven by the edge and cloud resources' computing and processing capabilities, has evolved as a promising approach [5], [6]. Although offloading techniques have been widely used to address the resource limitation of IoT applications [7], [8], [9], these works rely on numerical modeling and fail to consider that the heavy transmission workload is posted by video analytics, where a huge amount of streaming video is uploaded to the cloud server. DeepDecision [5] is the first attempt that combines the low power edge devices with more powerful cloud servers to execute a deep learning model both locally and remotely. However, this approach does not support fine-grained offloading of video frames across edge and cloud within a single window, which may lead to inefficient resource utilization. The inefficiency is caused by the possible idle time of either edge devices or cloud servers when the window size is not well tuned (see more details in §2.2). FastVA [6] proposes a local frame buffer for fine-grained splitting of video data within a window. This solution is not suitable when the number of frames within a window can change over time, for example, when cameras are configured to adaptive bitrate protocols. Furthermore, FastVA fails to coordinate video analytics workload across multiple edge nodes which may cause queuing delay in the cloud. We discuss this issue through a primary benchmark in §2.2. VideoPipe [10], VideoEdge [11], VideoStorm [12] and Chameleon [13] focus on scheduling and configuring the video analytics jobs (queries) on edge or cloud computing clusters. These approaches have two key limitations: (i) they may run out of capacity at the edge layer as most of these devices are resource constrained and (ii) they are unable to run complex deep learning models on resource-constrained devices, which may be required for more complex IoT application scenarios (e.g., city level traffic modeling).

In the practical deployment of edge computing, an offloading policy needs to consider *three facts*: 1) *heterogeneity of edge node* – Each edge node has a different processing rate based on current working situations (e.g., number of IoT

- *B. Qian, Newcastle University, UK. E-mail: b.qian3@ncl.ac.uk*
- *Z. Wen, Zhejiang University of Technology, CN. E-mail: Zhenyuwen@zjut.edu.cn, corresponding author.*
- *J. Tang, University of Cambridge, UK. E-mail: jt814@cam.ac.uk*
- *Y. Yuan, Beijing Institute of Technology, CN. E-mail: yuan-ye@bit.edu.cn*
- *A. Zomaya, Sydney University, E-mail: albert.zomaya@sydney.edu.au*
- *R. Ranjan, Newcastle University, UK. E-mail: raj.ranjan@newcastle.ac.uk*

devices to ingest data from). The proportion of the video offloaded to cloud should consider the computing and network load of each individual edge (e.g., CPU, upstream link utilization). 2) *interplay among edge nodes and cloud servers* – all edge nodes may forward the video to the cloud simultaneously, without considering others' offloading policies. This may cause starvation on the cloud server where the video from some edge nodes may be delayed for processing. 3) *modern video streaming protocols adaption* – video streaming protocols are essential for video delivery. They break video into small segments, send to target servers and reassemble them at destination. Video analytical framework needs to carefully adapt to this protocol, in particular needs to consider how varying number of frames within a segment impact the offloading policy.

Objective. Our approach OSMOTICGATE therefore considers the facts in practical deployment of edge computing, and proposes a novel technique to uncover the influence of these facts to offloading policy design. In particular, we develop a hierarchical queue model to capture the heterogeneity of edge node, in which the resource constraints (e.g., computing capacity and network bandwidth) of each edge node are used to parameterize the *local queue* and the *global queue* performance models. Thereafter, we attempt to minimize the processing latency of each video stream to achieve the real-time analytics. Specially, we formulate our problem as a non-smooth, non-convex, constrained min-latency optimization problem. To find an approximate solution of this problem efficiently, we develop a two-stage gradient-based algorithm and compare it with the state-of-the-art (SOTA) solutions (e.g., FastVA, DeepDecision, HillClimb [14]). Our evaluation demonstrates the advantages of workload-based modeling and the proposed algorithms reduce 2 × latency compared with SOTA methods in 5G network.

The contributions of this paper are as follows:

- We develop a new hierarchical queue model to describe the system dynamics of a video analytic system in edge-cloud environment. The model is adapted to bitrate-based video streaming and focuses on modeling the processing latency and throughput of a video analytics system (§3).
- We formulate the task offloading problem as a non-smooth, non-convex and constrained optimization problem (§4), and propose a gradient-based algorithm to solve this problem efficiently (§5).
- We feed the model parameter through real-world benchmark and compare our algorithms with SOTA methods in both simulated environment and real-world testbed (§6).

Position of the work. Our framework is generic for modeling the workload of complex video analytic system while resolving the optimal workload balance problem. The focus of this work is to minimize the system latency while maintaining the throughput within the user-defined bounds. In the future, the framework can be easily extended to include new decision variables including accuracy, energy consumption, resolution, bitrate as well as as the choice of deep learning models.

2 Background and Motivation

2.1 Edge-Cloud Paradigm for Video Analytics

In this work, we aim to provide a general solution for efficiently performing video analytics on the emerging edge-cloud computing paradigm [1] as shown in Fig. 1. In this computing paradigm, a set of video generating devices (e.g., traffic surveillance cameras, drones, mobile phones) are generating live video stream which can be processed either on low-power edge computing devices (e.g., Raspberry pi, Jetson Nano, computing chips), or a cloud data center with GPU cluster. Extensive research [3], [15] has been conducted in deep learning and adapted to assist video analytics. The communication among edge nodes and data center is via Wide Area Network (WAN), using various video stream transmission protocols.

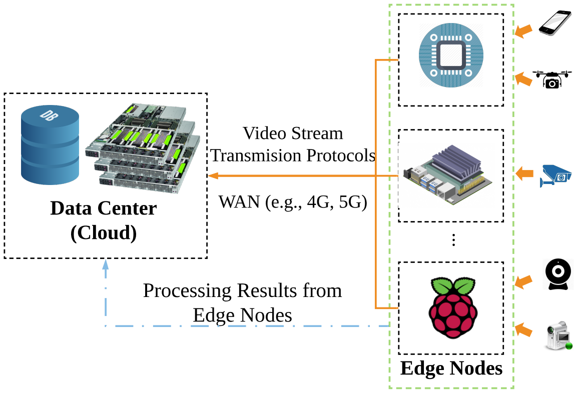
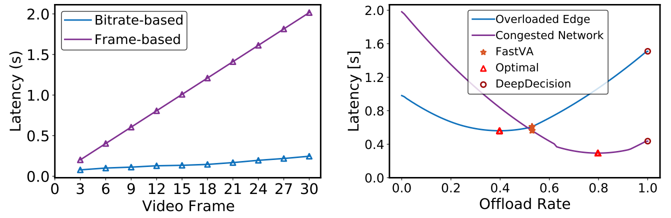


Fig. 1: Video Analytics in Edge-Cloud Computing Paradigm

2.2 Motivation

In order to perform video analytics efficiently on edge-cloud computing paradigm, the key is *when* and *how* to offload the video streaming to the data center. In the following, we use a real-world traffic monitoring application and conduct two benchmark experiments to study the impact of the three complexities mentioned in §1.

Setup. We deploy a retrained YOLOv3-tiny (detailed in §6.1) on both a Jetson Nano and a GPU server to count the number of cars in each video frame, with an emulated 5G network environment.



(a) Compare Transmission Latency (b) Heavy System Workload

Fig. 2: What is Affecting the Performance of Edge-Cloud Video Analytics System?

Bitrate-based V.S. Frame-based Video Transmission. We crop input video into segments containing 3 to 30 frames and deliver them from the edge to the cloud server using two types of transmission methods: 1) bitrate-based method that compresses the input video in H.264 format before transmission; and 2) frame-based method that transfers the video frames sequentially. As shown in Fig. 2(a), with the increase of video size, the difference of the transmission time between bitrate-based and frame-based method increases dramatically. The bitrate-based one is almost 10× faster than the frame-based one, when the number of frame is 30.

Understanding the System Workload is Essential. In this experiment, we inject video traffic for 60 seconds between edge node and cloud server while emulating following system dynamics: i.e., I) Heavy traffic between the edge node and server: we inject 10 seconds of video chunks to the communication queue beforehand; II) Heavy load in the edge: we inject 10 seconds of video chunks to the edge processing queue beforehand. Next, we benchmark two SOTA offloading techniques (FastVA and DeepDecision) and an exhaustive method. Fig. 2(b) shows that changing system workload impacts the optimal offloading rate. Moreover, we conclude that SOTA solutions are unable to achieve optimal offloading rate when subjected to system dynamics I and II.

Challenges. Bitrate-based video stream transmission is efficient and commonly used in real-world applications. However, adapting the video analytics pipeline to this transmission protocol requires video encoding/decoding process and smooth feeding of the data into the analytics model. Hence, the first requirement (**R1**) is to study and design a mechanism for splitting the video streaming processing task to a granular level that allows efficient encoding/decoding, assisting optimal offloading performance tuning. The second requirement (**R2**) is to develop a novel model that automatically captures the dynamic workload in the proposed video analytics system. In the practical deployment, the edge node heterogeneity, network variation, as well as the interplay between the edge nodes and cloud servers can induce complicated and dynamic workload within the system. Finally, the third requirement (**R3**) is to design a new algorithm which adapts to the proposed model and optimizes the system performance.

3 System Model

In this section, we first show a new design that allows our OsmoticGate to adapt to bitrate-based video streaming, and then propose a *Hierarchical Queue Model (HQM)* to describe the system behaviors for video analytics: we formulate the offloading policies and system performance optimization goal. Finally, we model *two* key metrics for measuring system performance: *processing latency* and *system throughput*.

3.1 Adapting Bitrate-based Video Streaming

A video chunk is a segment of video stream which contains a sequence of video frames. In order to utilize the bitrate-based video streaming and meet the requirement **(R1)**, we split a time window of video into n chunks. We therefore are capable of offloading a proportion of the chunks to the server for processing.

On edge node, each video is parameterised by a bitrate b **encoding** as defined in Eq. (1):

$$b = \alpha \cdot f \cdot r \cdot c \tag{1}$$

f is the video frame rate, and r is the resolution of a raw video frame. α represents the bits required to represent each image pixel, c is the compression rate with video encoding. Next, for each **split**, the start time and chunk duration are set. As video encoding must store key frame (i.e., I-frame) within the video as well as cross-frame information variation at a fixed time interval, we locate I-frames as well as the group of pictures (GOPS)[1] within the duration. Different video encoding techniques can adapt c and α to different settings. After this stage, video chunks can either be offloaded to cloud or processed locally. Finally, the **decode** is the process of converting video chunks into video frames. The extracted frames are processed by deep learning models where real detection happens.

3.2 Hierarchical Queue Model (HQM)

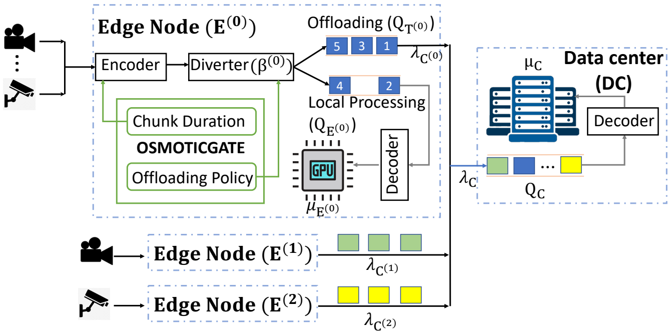


Fig. 3: Hierarchical Queue Model in OsmoticGate

To meet **(R2)**, we develop OsmoticGate and HQM as shown in Fig. 3. The design of OsmoticGate follows the pipelines in §3.1, with an *Encoder* and a *Diverter* on each edge node for generating and allocating video chunks. *Decoder* and *Inference Engines* are deployed on both edge and cloud data center for processing the encoded video chunks.

The core of OsmoticGate is a Hierarchical Queue Model (HQM), which uncovers the system bottleneck in edge-cloud video processing system. Our system considers K edge nodes, each with the following queues for real-time video analytics: 1) Offloading Queue Q_T; 2) Local Processing Queue Q_E for buffering the allocated video chunks. Q_C is the Cloud Processing Queue in the data center that receives the video chunks from any Q_T. All video in Q_E and Q_C must be decoded into frames in *Decoder* before being processed.

3.2.1 System Offloading Policy $\boldsymbol{\beta}$

In order to meet **(R3)**, each *Diverter* generates an offloading policy that controls the inputs of any Q_E and Q_T, which impacts the system performance. We formulate the system offloading policy $\boldsymbol{\beta}$ as:

$$\boldsymbol{\beta} := [\beta^1, \beta^2, ..., \beta^k] \tag{2}$$

β^k indicates offloading policy for each edge node E^k. We allow β^k to be any real numbers between 0 and 1 in order to fit our continuous optimization schemes.

The input rate of any Q_E and Q_T are noted by $\lambda_E(\beta)$ and $\lambda_T(\beta)$ respectively:

$$\begin{aligned} \lambda_E(\beta) &= \beta \cdot n/\Delta t \\ \lambda_T(\beta) &= (1-\beta) \cdot n/\Delta t \end{aligned} \tag{3}$$

for n chunks to be processed during time interval Δt, a proportion β is placed at the edge node E. As they can contain decimal numbers, we round the inputs to closest integer number throughout the following paper.

3.2.2 System Performance Optimization Goal

In HQM, we aim to minimize the system latency $T(\boldsymbol{\beta})$ (modeled in §3.3), while achieving the defined minimal throughput constraint $I(\boldsymbol{\beta})$ (in §3.4). To simplify the problem, we fix the video encoding configurations and optimize the offloading parameters $\boldsymbol{\beta}$. After we finish the optimization stage, we round them to the nearest values which will

[1] https://en.wikipedia.org/wiki/Group_of_pictures

enforce $\lambda_E(\beta)\Delta t$ to be integers. Our formulation for the optimization objective reads:

$$\begin{aligned}
&\arg\min_{\boldsymbol{\beta}\in\mathbb{R}^K} \; T(\boldsymbol{\beta}) \\
&\qquad s.t. \quad C_1 : I(\boldsymbol{\beta}) \geq I^* \\
&\qquad\qquad\;\;\, C_2 : \boldsymbol{\beta} \in [0,1]^K
\end{aligned} \tag{4}$$

C_1 represents that the system throughput must be equal to or greater than a predefined constraint I^* (detailed in §3.4). C_2 denotes that the video offloading rates $\boldsymbol{\beta}$ are restricted between $[0, 1]$.

3.3 Latency Model

In this subsection, we aim to model the end-to-end latency of processing each video chunk. For example, there are n chunks of video that are injected to our system during interval Δt (including those already in the system). The latency is the average time of processing all n chunks. We define the cumulative latency for processing all video chunks generated during time Δt in the system as $T(\boldsymbol{\beta})$:

$$T(\boldsymbol{\beta}) \quad = \quad \sum_{k=1}^{K}[T_E^k(\beta^k) \;+\; T_T^k(\beta^k)] \;+\; T_C(\boldsymbol{\beta}) \tag{5}$$

$T_E^k(\beta^k)$ and $T_T^k(\beta^k)$ are expected queueing latency for Local Processing Queue and Offloading Queue at any edge node E^k. $T_C(\boldsymbol{\beta})$ is the queueing latency for Cloud Processing Queue.

3.3.1 Processing Latency for Individual Component

First, we assume that the size for the video chunk is s (bits/chunk). The processing latency l_E for each video chunk on the edge node E has positive correlation with s as formalized in Eq. (6) and is assumed the same for chunks in the same Q_E:

$$l_E = a_E + b_E \cdot s \tag{6}$$

where b_E is a constant that defines relation between the processing latency and the size of each chunk. a_E is the oscillation introduced by the underlying hardware such as the video decoding latency.

Similarly, the processing latency l_C in the server can be expressed as :

$$l_C = a_C + b_C \cdot s \tag{7}$$

b_C indicates the coefficient between the processing latency and the size of a given chunk, and a_C is the underlying hardware influence.

Assuming uploading bandwidth between E and DC is B, the average transmission latency per chunk l_T is:

$$l_T = \frac{s}{B} \tag{8}$$

l_T is the same for all chunks in Q_T.

3.3.2 Queueing Latency for Local Processing Q_E

Let $Size(Q_E)$ denotes the number of video chunks in Q_E at time t. The input rate of Q_E is $\lambda_E(\beta)$, linearly correlated with β (see Eq. (3)).

Given system parameters, the queuing latency of i_{th} video chunk injected to Q_E during Δt (with respect to the offloading rate β) is denoted by $T_{Q_E}(i, \beta)$:

$$T_{Q_E}(i,\beta) = (\text{Size}(Q_E) + i) \cdot l_E - \frac{i}{\lambda_E(\beta)} \tag{9}$$

$(Size(Q_E)+i)\cdot l_E$ indicates the timestamp that the i_{th} chunk is popped out from Q_E, $\frac{i}{\lambda_E(\beta)}$ is the timestamp of the chunk arrived at the queue.

The sum of the processing latency for all items injected into the queue can be expressed in Eq. (10).

$$T_E(\beta) = \sum_{i=1}^{Size(Q_E)} i \cdot l_E + \sum_{i=1}^{\lambda_E(\beta)\Delta t} \max\{T_{Q_E}(i,\beta), l_E\} \tag{10}$$

T_E is the cumulative latency with two components: the first part is the latency for processing (including queueing latency) the remaining video chunks in Q_E, i.e., $i \cdot l_E$ is the cumulative latency of popping the i_{th} chunk out of the queue; the second part is the cumulative time cost of popping the injected video chunks during time Δt out of the queue. The max term indicates that minimum processing latency must be greater than pure edge processing latency l_E (processed instantly after entering the queue).

3.3.3 Queueing Latency for Offloading Q_T

Similar computation applies to the *Offloading Queue*. Denote $Size(Q_T)$ as the number of video chunks in the *Offloading Queue* at time t, the input rate is $\lambda_T(\beta)$ (see Eq. (3)). The queuing latency for i_{th} chunk in the *Offloading Queue* is computed through:

$$T_{Q_T}^k(i,\beta^k) = (\text{Size}(Q_T^k) + i) \cdot l_T^k - \frac{i}{\lambda_T^k(\beta^k)} \tag{11}$$

The accumulated processing latency $T_T(\beta)$ is computed as:

$$T_T(\beta) = \sum_{i=1}^{Size(Q_T)} i \cdot l_T + \sum_{i=1}^{\lambda_T(\beta)\Delta t} \max\{T_{Q_T}(i,\beta), l_T\} \tag{12}$$

3.3.4 Queueing Latency for Cloud Processing Q_C

Assume that $Size(Q_C)$ represents the number of video chunks in the cloud queue at time t, $\mu_T^k = \frac{1}{l_T^k}$ indicates the output rate of Q_T^k on edge node E^k. During Δt, the total number of video chunks coming into each *Offloading Queue* is denoted by $N_{Q_T}^k(\beta^k)$:

$$N_{Q_T}^k(\beta^k) = \lambda_T^k(\beta^k) \cdot \Delta t + Size(Q_T^k) \tag{13}$$

which adds up the injected chunks $\lambda_T^k(\beta^k) \cdot \Delta t$ and the chunks already in the queue $Size(Q_T^k)$.

Thus, the overall input rate $\lambda_C(\boldsymbol{\beta})$ of Q_C is defined by:

$$\lambda_C(\boldsymbol{\beta}) = \sum_{k=1}^{K} \lambda_C^k(\beta^k) = \sum_{k=1}^{K} \min\{\mu_T^k, \frac{N_{Q_T}^k(\beta^k)}{\Delta t}\} \tag{14}$$

input rate $\lambda_C^k(\beta^k)$ from each node is limited by the minimum between the video transmission rate μ_T^k and average producing rate $N_{Q_T}^k(\beta^k)/\Delta t$ from each *Offloading Queue*. $\lambda_C(\boldsymbol{\beta})$ sums up the output from all K *Offloading Queues*.

Again, we compute the queuing latency of i_{th} chunk arrived at Q_C by:

$$T_{Q_C}^k(i,\beta^k) = (Size(Q_C) + i) \cdot l_C - \frac{i}{\lambda_C(\beta^k)} \tag{15}$$

The cumulative processing latency of Q_C is:

$$T_C(\boldsymbol{\beta}) = \sum_{i=1}^{Size(Q_C)} i \cdot l_C + \sum_{i=1}^{\lambda_C(\beta)\Delta t} \max\{T_{Q_C}^k(i,\beta^k), l_C\} \tag{16}$$

3.4 Throughput Model

We define system throughput as the number of chunks that can be processed within time Δt. It is total throughput for the proposed system and is defined as $I(\boldsymbol{\beta})$:

$$I(\boldsymbol{\beta}) = \sum_{k=1}^{K} I_E^k(\beta^k) + I_C(\boldsymbol{\beta}) \tag{17}$$

where $I_E^k(\beta^k)$ indicates the throughout for each edge node E^k, $I_C(\boldsymbol{\beta})$ indicates the throughput for cloud DC.

3.4.1 Throughput of Edge Node E

Assume that the output rate of Q_E is $\mu_E = \frac{1}{l_E}$, total number of chunks in Q_E during Δt is:

$$N_E(\beta) = \lambda_E(\beta)\Delta t + Size(Q_E) \tag{18}$$

which is affected by offloading ratio β.

The throughput of the E is shown in Eq. (19):

$$I_E(\beta) = \min\{\mu_E, \frac{N_E(\beta)}{\Delta t}\} \tag{19}$$

If the queued chunks are greater than the processing capacity μ_E, the throughput I_E equals μ_E. Otherwise, I_E equals the processing rate of available chunks, i.e., $N_E(\beta)/\Delta t$.

3.4.2 Throughput of Cloud DC

The number of chunks in Q_C at time t is constrained by cumulative number of chunks 1) injected into, 2) popped out from all Q_T^k, which are defined by $N_{Q_{Tin}}$ and $N_{Q_{Tout}}$ respectively:

$$N_{Q_{Tin}} = \sum_{k=1}^{K}(\lambda_T^k \Delta t) \text{ and } N_{Q_{Tout}} = \sum_{k=1}^{K}(\mu_T^k \Delta t) \tag{20}$$

The cloud throughput is affected by the output rate of Q_C and the total number of chunks in Q_C, which can be formalized by the following:

$$I_C(\boldsymbol{\beta}) = \min\{\frac{N_{Q_{Tin}}+Size(Q_C)}{\Delta t}, \frac{N_{Q_{Tout}}+Size(Q_C)}{\Delta t}, \mu_C\} \tag{21}$$

The first two terms indicate two upper limits for all available chunks, which includes transmitted chunks from E^k and chunks already in the cloud queue $Size(Q_C)$. $\mu_C^k = \frac{1}{l_C}$ is the output rate of Q_C.

4 Constrained Min-Latency Problem

Recall in §3.2.2 we formulated the optimization problem as in Eq. (4), which aims to minimize the system latency while ensuring the minimal system throughput. In this section, we list the challenges to solve the problems and our transformations of the problem.

4.1 Challenges in the optimization task

Constrained optimization. In our formulation, we enforce constraints C_1 and C_2 that regularize the decision boundary of the input variable $\boldsymbol{\beta}$. Hence, the optimization algorithms must take the constraints into account and ensure feasibility and convergence simultaneously.

Non-smooth optimization with Lipschiz continuity. Our objective function $T(\cdot)$ is a Lipschitz-continuous function which satisfies:

$$\|T(x) - T(y)\|_2 \leq M\|x-y\|_2, \forall x, y \in \mathbb{R}^K \tag{22}$$

When $M < +\infty$, Eq. (4) is a non-smooth function globally, because its components have a piece-wise smooth structure of the form $\max\{f_1(\boldsymbol{\beta}), f_2(\boldsymbol{\beta})\}$. For example, Eq. (10) includes the term $\max\{(\text{Size}(Q_E)+i)\cdot l_E - \frac{i}{\lambda_E(\beta)}, l_E\}$, where we have $f_1(\beta) = (\text{Size}(Q_E)+i)\cdot l_E - \frac{i}{\lambda_E(\beta)}$, $f_2(\beta) = l_E$ which is non-differentiable when $f_1(\beta) = f_2(\beta)$. Since both $f_1(\beta)$ and $f_2(\beta)$ are smooth functions, $T(\cdot)$ is smooth in most time when the following condition is met: $x \in \mathbb{R}^K$ which satisfies $f_1(x) \neq f_2(x)$, there exists a radius $\varepsilon(x) > 0$:

$$\|\nabla T(x) - \nabla T(y)\|_2 \leq L(x)\|x-y\|_2, \forall y : \|x-y\|_2 \leq \varepsilon(x) \tag{23}$$

where we denote $L(x) < +\infty$ as the local-smoothness parameter at the point x which is the smallest possible positive value to ensure Eq. (23) to hold.

Non-convex optimization. We observe that objective function $T(\boldsymbol{\beta})$ (see Eq. (4)) is also non-convex. Since each component of $T_E(\beta)$, $T_T(\beta)$ and $T_C(\boldsymbol{\beta})$ includes a sum of non-convex functions. For example, in Eq. (10) we have $\sum_{i=1}^{\lambda_E(\beta)\Delta t}\max\{(\text{Size}(Q_E)+i)\cdot l_E - \frac{i}{\lambda_E(\beta)}, l_E\}$, where $\lambda_E = \beta \cdot n/\Delta t$, and then the first term in the $\max$ is a concave function in β, while l_E is a constant. Hence all the elements in the sum is either non-convex or a constant function. As a result, It is a non-convex optimization problem that is a NP-Hard problem.

Theorem 4.1. *To solve Eq. (4) is a NP-hard problem.*

Proof. Non-convex Quadratic Programming (QP) is known to be an NP-hard problem [16]. It is defined as follows:

$$\begin{aligned} \min \quad & \frac{1}{2}x^T Q x + c^T x \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{24}$$

where x is the variable and (Q, A, b, c) are the data. When Q is an indefinite symmetric matrix, QP is non-convex and NP-hard [17].

Consider the definition of the QP problem as above, it can be transformed into the optimization problem in Eq. (4) in a polynomial time. The transformation is as follows. i) change all variable x into the offloading rate $\boldsymbol{\beta}$; ii) adapt $Ax \leq b$ into the system throughput constraint C_1 in Eq. (4): with the expansion from Eq. (28), $I(\boldsymbol{\beta})$ is either linearly correlated or irrelevant of $\boldsymbol{\beta}$, which satisfies the linear constraint in QP; iii) adapt the generic quadratic function $\frac{1}{2}x^T Q x + c^T x$ into the optimization goal $T(\boldsymbol{\beta})$ in Eq. (4): $T(\boldsymbol{\beta})$ is correlated with $T_E(\beta), T_T(\beta)$ and $T_C(\beta)$ in Eq. (10), (12) and (16). All three equations are transformed into quadratic forms with respect to β after the adaptation of $\sum(\cdot)$ in Eq. (25). $f_1(\beta)$ is linearly correlated with β and $f_2(\beta)$ is element-wise non-convex function with respect to β. The multiplication $\frac{1}{2}(1+f_1(\beta))\cdot f_1(\beta)\cdot f_2(\beta)$ reveals that $T(\beta)$ fits the quadratic form with respect to $\boldsymbol{\beta}$ with indefinite matrix Q. Thus, in non-convex QP with indefinite matrix Q, Q can be transformed into our problem setting which also contains indefinite matrix Q. This transformation can be achieved in polynomial time. Our problem is at least as hard as non-convex QP problem, which is NP-hard, making our optimization problem in Eq. (4) NP-hard. □

4.2 Problem Transformation

To solve the non-convex problem through a gradient-based method, in this section, we derive the adaptation of the objective function and the gradient accordingly. The adaptation discussed in this section will assist in derivation of gradient information as discussed in §5.

Adaptation of $\sum(\cdot)$. In Eq. (10), (12) and (16) we have the summation in the form of $\sum_{i=1}^{f_1(\beta)}(i \cdot f_2(\beta))$ which can not be directly used for computation against the offloading rate $f(\beta)$. To adapt them into an expression with respect to the offloading rate $f(\beta)$, such that gradient can be computed wherever possible.

For Eq. (10), we have $T_E(\beta) = a + \sum_{i=1}^{f_1(\beta)} f_2(i,\beta)$ where a is a constant, $f_1 = \beta \cdot \Delta t$ and $f_2(i,\beta) = \max\{(\text{Size}(Q_E) + i)\cdot l_E - \frac{i}{\lambda_E(\beta)}, l_E\}$. We first extract the i from $f_2(i,\beta)$ into the form $f_2(\beta)$, so that:

$$\begin{aligned} f(\beta) &= \sum_{i=1}^{f_1(\beta)} i \cdot f_2(\beta) \\ &= \tfrac{1}{2}(1 + f_1(\beta)) \cdot f_1(\beta) \cdot f_2(\beta) \end{aligned} \quad (25)$$

The gradient of $f(\beta)$ is expressed as $g(\beta)$:

$$\begin{aligned} g(\beta) = \quad & (\tfrac{1}{2} + f_1(\beta)) \cdot f_2(\beta) + \\ & \tfrac{1}{2}(1 + f_1(\beta)) \cdot f_1(\beta) \cdot f_2^{'}(\beta) \end{aligned} \quad (26)$$

To ease the computation of gradient, we smooth the value space of β by transforming it from discrete to continuous space. We choose the closest discrete value after deciding the final offloading ratio β.

Relaxation of $\min(\cdot)$ and $\max(\cdot)$. The non-smoothness in our optimization function is introduced by $\min(\cdot)$ and $\max(\cdot)$ terms. We need to relax these terms to remove the non-smoothness. For example in $f_2(\beta^k) = \max\{f_3(\cdot), f_4(\cdot)\}$. where $f_3(\cdot) = \text{Size}(Q_E^k) + i) \cdot l_E^k - \frac{i}{\lambda_E^k(\beta^k)}$ and $f_4(\cdot) = l_E^k\}$. The relaxation can be implemented by:

$$\max(f_3(\cdot), f_4(\cdot)) = \begin{cases} f_3(\cdot) & f_3(\cdot) > f_4(\cdot) \\ f_4(\cdot) & f_3(\cdot) < f_4(\cdot) \end{cases} \quad (27)$$

With the system parameters and offloading rates, the value of $f_3(\cdot)$ and $f_4(\cdot)$ are determined, and thus the $min(\cdot)$ term is relaxed. By doing so, we can ensure that non-smoothness in the function is removed while not changing the problem statement of formulation. However, there is no expression when $f_3(\cdot) = f_4(\cdot)$. This problem needs to be resolved with our Alg. 2 (detailed in §5.3) . Similarly, $min(f_3(\cdot), f_4(\cdot))$ terms in Eq. (14) and (21) can be relaxed as the following:

$$\min(f_3(\cdot), f_4(\cdot)) = \begin{cases} f_3(\cdot) & f_3(\cdot) < f_4(\cdot) \\ f_4(\cdot) & f_3(\cdot) > f_4(\cdot) \end{cases} \quad (28)$$

Relaxation of L_C. To enable the adaptation of Eq. (25) in Eq. (16), we make the following relaxation. l_C is the latency of processing video chunks in server side. To simplify the computation, we assume that $l_C(r)$ is the same for any types of chunks from various edge nodes, i.e., $L_C = \sum_{k=1}^{K} L_C^k / K$.

5 Two-stage Algorithm Design

In the previous section we have presented the formulation of the constrained min-latency optimization objective. Now in this section we complete the algorithmic design by adapting the gradient-based optimization methods as the solver according to the characteristics of our objective. The proposed formulation of the optimization task and the adaptation of the gradient-based algorithms jointly form a core contribution of this work.

Why Gradient-based Algorithm. The gradient-based algorithms [18], [19], although they do not have theoretical guarantees for finding the global optima, have been adapted to provide numerical solutions for many non-convex optimization problems. Motivated by the characteristics of our objective function, a two-stage gradient algorithm is proposed for solving Eq. (4). We use the gradient-based methods to find an approximate solution. The first motivation behind such choice is that the objective function's gradients can be efficiently computed ($O(\sum_{k=1}^{K} n^k)$ floating point operations). Meanwhile as it is locally smooth almost everywhere, gradient-based methods can guarantee decrease at each iteration with suitable step-sizes and fast convergence rates.

We consider the gradient-based methods to be the most suitable choices for our optimization task, due to their numerical efficiency and simplicity of implementation. One may consider the zeroth-order methods such as the Hill-Climb, which does not exploit the gradient information (note that in our setting the gradient evaluation is as efficient as the function value query) and has poor convergence rates [20] especially in high dimensions. It is unclear whether the higher-order methods such as the Newton-type methods are suitable for our tasks, since they typically need significantly more computational cost [21] for deriving the descent direction and handling the constraints compared to gradient-based methods – we leave this as a future direction of research.

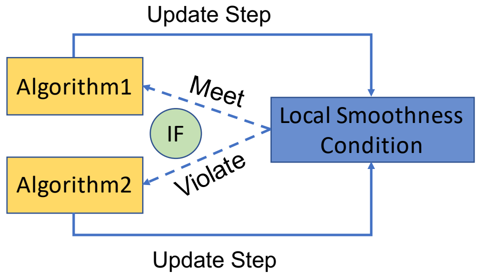


Fig. 4: High-level Overview of Two-stage Algorithm

Alternatively, one may consider adapting the stochastic gradient descent (SGD) methods [22], [23] to our task. Such methods compute efficient approximations of the gradients $\nabla T(\cdot)$ on randomly subsampled minibatches of the loss function $T(\cdot)$ as the descent direction. The SGD methods are tailored for huge-scale tasks such as training deep neural nets on a large dataset. Since our optimization task is rather mild-scale, and meanwhile it is unclear how to implement a stable and efficient line-search scheme for stochastic gradients in our task (and also considering the practical limitations and downsides[2] of SGD discussed in [24]), we leave the investigation of the practicality of using SGD-type iterations for potentially improving the convergence rates in our task as a future direction.

5.1 Overview of Two-Stage Gradient Algorithm

Denoting $\mathcal{Q}$ as the constraint set resulted by C_1 and C_2:

$$\mathcal{Q} := \{v \in [0,1]^K \mid I(v) \geq I^*\}, \quad (29)$$

we define the projection operator as the following:

$$\mathcal{P}_{\mathcal{Q}}(x) := \arg\min_{y \in \mathcal{Q}} \frac{1}{2}\|x - y\|_2^2. \quad (30)$$

The projection operator takes any point $x \in \mathbb{R}^K$ and returns its closest point within the constraint set. Then our algorithms for solving Eq. (4) can be generally expressed as the following iterative form:

$$\boldsymbol{\beta}_{j+1} = \mathcal{P}_{\mathcal{Q}}(\boldsymbol{\beta}_j - \eta_j \boldsymbol{\gamma}_j), \quad (31)$$

where η_j is the step size chosen at iteration j, while $\boldsymbol{\gamma}_j$ is some descent direction which seeks to decrease the function value $T(\cdot)$. The most simple and efficient choice is the gradient direction:

$$\boldsymbol{\gamma}_j = \nabla T(\boldsymbol{\beta}_j), \quad (32)$$

Since the objective function is globally non-smooth, for each step we need to choose a step-size η_j that adapts to

[2] Compared to the full gradient descent methods, SGD methods can have slower convergence rates in some scenarios/regimes [24], and require significantly more frequent calls on the projection operators [25]. Moreover, they are less compatible with the line-search schemes, and have less parallelizability [19].

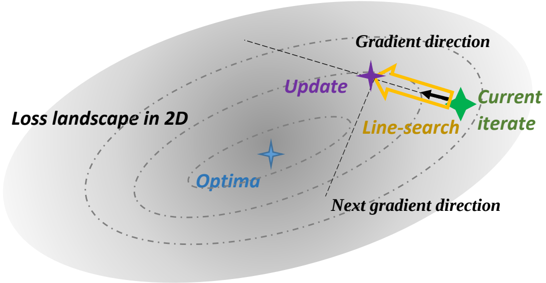


Fig. 5: Illustration of the Weak-Wolfe line search mechanism, which in each iteration seeks a step-size to optimally decrease the objective function value and make sure that the next gradient direction to be as orthogonal as possible to current gradient direction.

the local smoothness as described in Eq. (23). Therefore, our proposed algorithm consist of two stages as shown in Fig. 4. If the local-smoothness condition, a generalized sufficient decrease property, is met, the efficient algorithm PGD-VAO (§5.2) is applied for updating the steps. Otherwise, PGS-VAO (§5.3) is applied for updating the steps. Based on this switch, the algorithm converges after several iterations. The detail of how to switch between two algorithm is discussed in §5.4. Finally, we analyze the algorithm complexity in §5.5.

5.2 Projected Gradient Descent for Video Analytic Offloading (PGD-VAO)

We first present our projected gradient descent algorithm tailored for solving optimization task Eq. (4) in Alg. 1. In **Step 0**, we first compute the gradient at point $\boldsymbol{\beta}_j$ by transferring $\min(\cdot)$ and $\max(\cdot)$ through Eq. (27) and Eq. (28). Then, the gradient can be computed by Eq. (26). Since the local smoothness parameter defined in Eq. (23) at point $\boldsymbol{\beta}$ – the $L(\boldsymbol{\beta})$ is not known for each update but the practical step-size choice of gradient step is dependent on this parameter, we adopt the weak-wolfe line-search [26] scheme which estimates the local smoothness and allow us to use an adaptive step-size throughout iterations (see **Step 1**). Also, we discuss how to choose line search algorithms to improve the performance of PGD-VAO in §5.2.1. In **Step 2**, we compute next point y_{j+1}. Then, in **Step 3**, we check whether y_{j+1} meets the constraint, if not it will be returned to the closet point within the constraint set, denoted $\boldsymbol{\beta}_{j+1}$.

5.2.1 *Choice of Line Search*

In practice the theoretically ideal step-size of $\eta_j = O(1/L(\boldsymbol{\beta}_j))$ cannot be exactly computed since it is computationally intractable and unnecessary to get the exact value of $L(\boldsymbol{\beta}_j)$, hence the line-search schemes have been widely-adopted as numerical solutions. For instance, the back-tracking line-search scheme, being the most simple yet widely-applied choice, can be expressed as finding step size η_j such that a sufficient decrease is numerically enforced:

$$T(\boldsymbol{\beta}_j - \eta_j\boldsymbol{\gamma}_j) \leq T(\boldsymbol{\beta}_j) - w_1\eta_j\langle\nabla T(\boldsymbol{\beta}_j), \boldsymbol{\gamma}_j\rangle, \tag{33}$$

with parameter $w_1 \in (0, 1)$ configurable, while the (weak) Wolfe line-search scheme [26] uses an additional condition:

$$\langle\nabla T(\boldsymbol{\beta}_j - \eta_j\boldsymbol{\gamma}_j), \boldsymbol{\gamma}_j\rangle \leq w_2\langle\nabla T(\boldsymbol{\beta}_j), \boldsymbol{\gamma}_j\rangle, \tag{34}$$

with a tunable parameter $w_2 \in [w_1, 1)$. The idea behind the Wolfe-type line-search is simple, as we can observe from above experession, that besides seeking a decrease in functional value, it enforce the gradient direction of the forthcoming iteration to be as orthogonal as possible w.r.t the current descent direction $\boldsymbol{\gamma}_j$ (which in the case of PGD-VAO $\boldsymbol{\gamma}_j = \nabla T(\boldsymbol{\beta}_j)$). We also illustrate in 2D the mechanism of such a line-search method in Figure 5. The Wolfe-type line search methods are known for providing a more accurate estimation of the ideal step-sizes compare to the backtracking line-search.

Algorithm 1: Projected Gradient Descent for Video Analytic Offloading (PGD-VAO)

```
Input:
Initial point $\boldsymbol{\beta}_0$ at which $T$ is differentiable,
weak wolfe line search parameter $w_1, w_2$, constraints $\mathcal{Q}$,
    total number of iterations $J$
for $j = 1, 2, ..., J$ do
    Step 0: Compute the gradient
    $\boldsymbol{\gamma}_j = \nabla T(\boldsymbol{\beta}_j)$
    Step 1: Step length calculation
    $\eta_j = \text{line\_search}(\boldsymbol{\beta}_j, \boldsymbol{\gamma}_j, w_1, w_2)$
    Step 2: Update $y_{j+1} = \boldsymbol{\beta}_j - \eta_j\boldsymbol{\gamma}_j$
    Step 3: Projection $\boldsymbol{\beta}_{j+1} = \arg\min_{x\in\mathcal{Q}} \frac{1}{2}||x - y_{j+1}||_2^2$
end
```

5.3 Projected Gradient Sampling for Video Analytic Offloading (PGS-VAO)

In this subsection we present our PGS-VAO algorithm tailored for solving our constrained optimization task as Alg. 2. The gradient sampling was proposed in [27] whereby addressing the non-smooth property of the objective function.

Step 1: With sampled points in **Step 0**, we compute all gradients at these points following the same procedure as **Step 0** in Alg. (1) . Following the recent research in [28], we use the non-normalized search direction $-\boldsymbol{\gamma}_j$ as opposed to $-\boldsymbol{\gamma}_j/||\boldsymbol{\gamma}_j||_2$ in [27]. If the norm of the gradient $||\boldsymbol{\gamma}_j||$ is smaller than the optimality tolerance ν_j, we terminate this loop, reduce the sampling radius ϵ_j and optimality tolerance ν_j, continue next iteration.

Step 2: We numerically observe that the original backtracking line search applied in [28] does not provide consistent results. We hence replace backtracking line search by bisection line search satisfying Weak Wolfe conditions.

Step 3: After each update, if the resulting points is out of boundary $y_{j+1} \notin \mathcal{Q}$, we implement projection step 4, otherwise we continue next loop.

Step 4: We solve for each step, y_{j+1} that do not meet both C_1 and C_2 will be projected back to the points within the original variable space $\mathcal{Q}$ that are closed to the current position y_{j+1} in terms of euclidean distance and we obtain $\boldsymbol{\beta}_{j+1}$ to move to the next iteration.

5.4 Switching between PGD-VAO and PGS-VAO

In this subsection we discuss the condition that makes the PGD-VAO non-convergent meaning we have to switch to the PGS-VAO iterations. Once our algorithm reaches the point that the PGD iterations are able to make progress, our algorithm may switch back to PGD-VAO.

A theoretical analysis on the convergence of PGD-VAO. We start by presenting in Lemma 5.1 a generalized sufficient decrease property when the local smoothness holds, which is essential for the convergence of our algorithm. The proof strategy of this lemma is standard and follows similar

Algorithm 2: Projected Gradient Sampling for Video Analytic Offloading (PGS-VAO)

```
Input:
Initial point $\boldsymbol{\beta}_0$ at which $T$ is differentiable, closed unit ball $\mathbb{B}$, maximum step $N$.
initial sampling radius $\epsilon_0 > 0$, sample radius reduction factor $\theta_\epsilon \in [0,1]$, sample size $m \geq K+1$.
Optimality tolerance $\nu_j \geq 0$. Optimality tolerance reduction factor $\theta_\nu \in [0,1]$
weak wolfe line search parameter $w_1$, $w_2$, constraints $\mathcal{Q}$
for $j \in N$ do
    Step 0: Gradient Sampling
    Independently sample $\{\boldsymbol{\beta}_{j,1}, \boldsymbol{\beta}_{j,2}, ..., \boldsymbol{\beta}_{j,m}\}$ from $\mathbb{B}(\boldsymbol{\beta}_j, \epsilon_j)$;
    Step 1: Search direction calculation
    Compute $\boldsymbol{\gamma}_j$ as the solution of $\arg\min_{g\in\mathcal{G}} ||g||_2^2$, where
    $\mathcal{G} = \mathrm{Conv}\{\nabla T(\boldsymbol{\beta}_{j,0}), \nabla T(\boldsymbol{\beta}_{j,1}), ..., \nabla T(\boldsymbol{\beta}_{j,m})\}$;
    if $||\boldsymbol{\gamma}_j|| \leq \nu_j$ then
        $t_j$ = 0, set $\nu_{j+1}$ = $\theta_\nu \nu_j$, $\epsilon_{j+1}$= $\theta_\epsilon \epsilon_j$;
        go to step 3
    else
        set $\nu_{j+1}$ = $\nu_j$, $\epsilon_{j+1}$ = $\epsilon_j$;
        go to step 2
    end
    Step 2: Step length calculation
    $\eta_j$ = line_search($\boldsymbol{\beta}_{j,0}, \boldsymbol{\gamma}_j, w_1, w_2$)
    Step 3: Update  $y_{j+1}$ = $\boldsymbol{\beta}_j - \eta_j \boldsymbol{\gamma}_j$
    if $y_{j+1} \notin \mathcal{Q}$ then
        go to step 4;
    else
        set $\nu_{j+1}$ = $\theta_\nu \nu_j$, $\epsilon_{j+1}$= $\theta_\epsilon \epsilon_j$;
        continue
    end
    Step 4: Projection $\boldsymbol{\beta}_{j+1} = \arg\min_{x\in\mathcal{Q}} \frac{1}{2}||x - y_{j+1}||_2^2$
end
```

steps as the sufficient-decrease result in [29] which was originally derived for globally-smooth composite objectives. We include the proof in the Appendix A for completeness. Next we will use this lemma to analyze when the projected gradient descent with practical line-search schemes will converge to a stationary point.

Lemma 5.1 (Local Sufficient Decrease Property). *Let $x \in \mathcal{Q}$ and $T(\cdot)$ is locally smooth around x with a radius $\varepsilon(x)$ such that:*

$$\|\nabla T(x) - \nabla T(v)\|_2 \leq L(x)\|x - v\|_2, \forall v : \|x - v\|_2 \leq \varepsilon(x) \tag{35}$$

where we denote $L(x) < +\infty$ as local-smoothness parameter at the point x which is the smallest possible positive value to ensure (35) to hold, and $z = \mathcal{P}_\mathcal{Q}(x - \eta\gamma)$ with the step-size η is suitably chosen such that $\|z - x\|_2 \leq \varepsilon$, then for any $a > 0$ we have:

$$\begin{aligned} 0 \quad \leq \quad & T(x) - T(z) + \frac{1}{2aL(x)}\|\boldsymbol{\gamma} - \nabla T(x)\|_2^2 \\ & + [\frac{L(x)(a+1)}{2} - \frac{1}{2\eta}]\|x - z\|_2^2 \end{aligned}$$

We now apply the local sufficient decrease property by Lemma 5.1 to show that when local smoothness holds with a lower-bounded radius $\varepsilon(\boldsymbol{\beta}_j) \geq \varepsilon > 0$, the updating sequence $\boldsymbol{\beta}_j$ generated by Alg.1 converges to a stationary point. We start by defining the generalized gradient map at any position x as:

$$G(x) := \frac{1}{\eta}[x - \mathcal{P}_\mathcal{Q}(x - \eta\nabla T(x))]. \tag{36}$$

When the vector $x - \eta\nabla T(x)$ is in the constraint set $\mathcal{Q}$, it is clear that $G(x) = \nabla T(x)$.

Theorem 5.2 (Convergence of PGD-VAO under local-smoothness)**.** *Suppose for all iteration j, the updates $\boldsymbol{\beta}_j$ admit local smoothness with a radius lower bounded as $\varepsilon(\boldsymbol{\beta}_j) \geq \varepsilon > 0$, $\eta_j = \frac{1}{2aL(\boldsymbol{\beta}_j)}$ for some $a > 1$, then the accumulative average gradient norm, $\boldsymbol{G}(J) := \frac{1}{J}\sum_{j=1}^{J} \|G(\boldsymbol{\beta}_j)\|_2^2$ converges at a rate $O(1/J)$:*

$$\boldsymbol{G}(J) \leq \frac{8aT(\boldsymbol{\beta}_0)\max_{j\in[J]} L(\boldsymbol{\beta}_j)}{J}, \tag{37}$$

and meanwhile $\|G(\boldsymbol{\beta}_J)\|_2^2 \to 0$ as $J \to +\infty$.

We provide the proof in Appendix B. Theorem 5.2 suggests that for the case where we have local-smoothness lower bounded above 0 throughout the iterations, the sequence generated by Alg.1 strictly converges to a stationary point of Eq. (4). However, when the iteration violates this condition, the gradient norm $\boldsymbol{G}(J)$ will be unbounded and we cannot guarantee convergence for stationary point at this case. This potential weakness of PGD-VAO motivates us to adopt the class of gradient sampling algorithms of Burke et al [27], [28] which is tailored for addressing such a non-convergent issue of line-search gradient descent methods in non-smooth optimization [3]. While the PGD-VAO is computationally efficient, our PGS-VAO algorithm demands significantly more computation cost. Ideally we wish to run the efficient PGD-VAO algorithm whenever the local-smoothness holds with a non-decreasing radius.

Practical implementation. From our analysis we can see that it is easy to check in practice when should we switch from PGD-VAO to PGS-VAO. We can observe from the proof of Theorem 5.2 presented in the supplemental material, that only when $\eta_j = O(1/L(\boldsymbol{\beta}_j))$ can we derive the bound in Eq. (37). Meanwhile if the radius $\varepsilon(\boldsymbol{\beta}_j) \to 0$, then the local sufficient decrease cannot hold unless the step size must also shrink $\eta_j \to 0$. In the context of Theorem 5.2, it is equivalent to regard this case as $\eta_j = \frac{1}{2aL(\boldsymbol{\beta}_j)}$ with $a \to +\infty$, and hence the right-hand-side of (37) become unbounded: $\frac{8aT(\boldsymbol{\beta}_0)\max_{j\in[J]} L(\boldsymbol{\beta}_j)}{J} \to +\infty$, and at such case the PGD-VAO cannot have guaranteed convergence to stationary point and we need to switch to PGS-VAO.

Recall that we use a line-search algorithm to estimate the local-smoothness and adaptively determine the step-size. If we observe that the step-sizes given by the line-search scheme keep decreasing towards 0 for a number of iterations, then it suggests that sequence arrives at a regime where the local smoothness fails to hold. At such case we need to switch to PGS-VAO for further progress.

To be more specific, the most practical scheme for determining the switching point could be: first selecting a small step-size threshold η_a which is close to 0; then if the step-sizes chosen by the line-search scheme are below this threshold consecutively for a number of iterations, we may switch to the Alg 2. Similarly, after a few iterations of using costly Alg 2, if the step-sizes chosen by the line-search scheme are above this threshold, we may switch back to use the efficient PGD-VAO iterations.

[3] We refer the readers to [28, Figure 1] for an illustration of the non-convergent issue of gradient descent (with line-search) and how gradient sampling can overcome this.

Discussion. Although the gradient-based methods cannot guarantee convergence to the global optima for non-convex objectives in general, numerically we found that our algorithms consistently converge to well-performing solutions which are sufficient in practice. This phenomenon seems to suggest that our objective function is likely to be a well-behaved non-convex function, such that local minimas are almost as good as global optima. However, we plan to investigate this problem in the future at a greater depth.

5.5 The Complexity of the Algorithms

The complexity per iteration of the PGD-VAO algorithm is relatively low. To evaluate the gradient of $T_E^k(\beta^k)$, $T_T^k(\beta^k)$ and $T_C(\boldsymbol{\beta})$ in E.q 10, 12 and 16, the number of element-wise gradient evaluation is $\beta^k n^k$, $(1-\beta^k)n^k$ and $\sum_{k=1}^{K} \min\{\mu_T^k, (1-\beta^k)n^k + \text{Size}(Q_T^k)\}$ respectively. Denote $\bar{n} = \frac{1}{K}\sum_{k=1}^{K} n^k$ (where K is the total number of edge nodes) we can see that the gradient evaluation takes $O(\bar{n}K)$ floating point operations, while the line search takes the same order of complexity and projection step takes $O(K)$ floating point operations, and hence the total complexity per iteration of PGD-VAO is $O(\bar{n}K)$.

However, the PGS-VAO algorithm is much more computationally expensive per iteration. It first needs to compute at least $K+1$ gradients which cost $O(\bar{n}K^2)$, then solving the QP subproblem in step 1 takes $O(K^3)$ floating point operations. The complexity of PGS-VAO per iteration is $O(\bar{n}K^2 + K^3)$. Hence the PGS-VAO iterations are much more computationally expensive than PGD-VAO iterations.

6 Evaluation

In the evaluation, we first benchmark system performance of the cloud-edge video processing system (as shown in Fig 1) in a real world test-bed and then feed the benchmarked parameters to our model. Next, we evaluate the performance of OsmoticGate through simulations, and compare its performance with the SOTA solutions.

6.1 Obtaining the parameters for HQM via real-world benchmark

Parameters. To make the HQM capture the system behaviors of the cloud-edge video processing system, we conduct a set of real-world benchmark experiments to obtain the modeling parameters. Specifically, in §3.2, we model the relationship between the processing latency of system components against the video size. Hereby we benchmark three sets of parameters including edge inference latency (Eq. (6)), cloud inference latency (Eq. (7)) and network transmission latency (Eq. (8)). We also add 10% of oscillation to the network bandwidth to simulate real-world condition.

Environment Set-up. We use NVIDIA Jetson Nano (with ARM Cortex-A57 CPU and 4GB RAM) as the edge node and the cloud server is a bare metal Ubuntu machine, with 20 cores (Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz), GeForce GTX1080 Ti graphics card and 32 GB RAM. The network is configured by by using Linux traffic control (TC) as shown in Table 1, according to real world measurement[4]

Dataset and Deep Learning Model. We aim to achieve real-time road traffic monitoring which requires to detect the number of cars in each video frame. We therefore train a YOLOv3-tiny, a variant of the YOLOv3 [15] with 9 layers by using Jackson video dataset [30]. Since the dataset does not provide the labels, we extract image frames from the video and manually annotate 2500 frames for training purpose. The labeled dataset is available at [31]. The models are implemented with Darknet[5] and trained on our server. The trained YOLOv3-tiny is deployed on both server and edge. Moreover, TensorRT[6] is used to optimize the deployment on Jetson Nano for high-performance inference.

Network	Latency	Upload Bandwidth
2G	500ms	0.1Mbps
3G	80ms	1Mbps
4G	40ms	8Mbps
5G	20ms	20Mbps

TABLE 1: Emulated Network Configuration

Benchmarking Results. Table. 2 shows the inference speed of the cloud server and edge node, we record end-to-end latency from video decoding to completion of the inference procedure. We see that the inference latency increases linearly against the chunk size. Also, the cloud server has 10x of processing speed, compared to that of the edge node. We feed benchmarked parameters to model in Eq. (6) and (7).

Also, we record end-to-end transmission latency under difference network environments. With the benchmark results, we can approximate the relation between the chunk size and transmission latency as formulated in Eq. (8).

Chunk Size to Chunk Duration. As illustrated in Eq. (6), (7) and (8), we are interested in the relation between the latency and the chunk size. However, chunk size is not directly configurable with FFMpeg processing. We resolve this by mapping the chunk duration to chunk size as shown in Table 2, such that all experiments can be implemented with chunk duration as a control variable. Our benchmark result indicates that chunk size maintains a linear relation with the chunk duration, and that the processing latency fits our modeling proposition in Eq. (6), (7) and (8).

Chunk Duration (s)	**Chunk Size (KB)**	**Cloud Inference Latency(s)**	**Edge Inference Latency (s)**	**Transmission Latency (s)**			
				2G	**3G**	**4G**	**5G**
0.1	315	0.065	0.497	15.62	1.565	0.196	0.114
0.2	330	0.08	0.698	19.81	2.013	0.252	0.142
0.3	341	0.095	0.923	21.98	2.242	0.28	0.162
0.4	348	0.11	1.143	25.41	2.573	0.322	0.183
0.5	352	0.125	1.353	26.11	2.683	0.336	0.191
0.6	358	0.14	1.552	28.44	2.916	0.364	0.211
0.7	365	0.165	1.751	31.9	3.363	0.42	0.242
0.8	371	0.17	1.952	38.11	3.921	0.49	0.281
0.9	378	0.185	2.151	41.13	4.364	0.546	0.313
1.0	384	0.20	2.354	47.43	4.923	0.616	0.355

TABLE 2: Testbed Benchmarking

Model Accuracy. Video that is encoded with lower bitrate can have less latency and bandwidth cost as well as higher throughput, compared to that is encoded with higher bitrate. In order to mitigate the accuracy loss while maximizing the system throughput, video encoding configuration has to be carefully designed.

DeepDecision [5] has evaluated that resolution does not impact greatly on model inference accuracy. In the following, we evaluate the relation between model inference accuracy and bitrate. The model prediction accuracy is computed against the annotated car objects in the video frame. The overlapping between the detection and annotation bounding boxes is computed using IoU (Interaction of

[4] https://www.opensignal.com/reports/2018/04/uk/state-of-the-mobile-network

[5] http://pjreddie.com/darknet

[6] https://developer.nvidia.com/tensorrt

Bitrate (kb/s)	Precision	Recall	F1
1000	0.9626	0.9338	0.9433
500	0.9661	0.9247	0.9400
250	0.9670	0.9052	0.9286
100	0.9607	0.7358	0.8121

TABLE 3: Model Accuracy under Various Bitrates and the Video Resolution is 1080P.

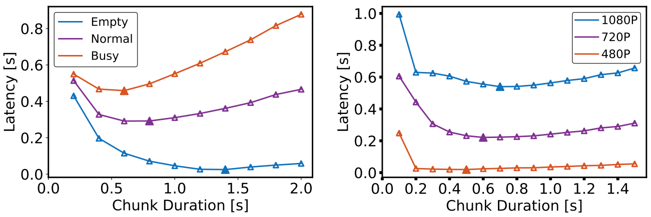
Union) metrics. We identify one detection as true positive when computed IoU metric exceeds a threshold (0.7 in out setting). As both the prediction accuracy and coverage are important, we consider the F1 score as a metric to evaluate model performance. Table 3 shows that the prediction precision does not decrease too much with the reduction of bitrate. Note that the accuracy decrease dramatically when the bitrate is lower than 100kb/s. In this paper, we set the bitrate as 1000kb/s for the rest of experiments. We are aware that accuracy of YOLOv3-Tiny could be inferior to YOLO. However, for our developed real-world dataset, yolov3-tiny already achieves acceptable accuracy. Also, the same detection model is deployed at both the edge and the cloud. Once the video is encoded, the same configuration would lead to the same accuracy for both sides, which applies to all baselines in §6.2, and not only HQM. The above benchmarking result motivates modeling the accuracy as an observation parameter instead of an variable parameter in the HQM.

Determine the Chunk Duration. As discussed in §3.2, we adapted video analytics task offloading based on bitrate video streaming. The following evaluates the impacts of the video split granularity (i.e., chunk duration) to the system performance. To this end, we first configure the input rate of video stream and network as 30 frame/s and 5G, and consider *two* scenarios below: 1) *Varying system workload*: We set three system status, i.e. no workloads (*empty*), normal workloads (*normal*) and heavy workloads (*busy*) across the whole system. We also encode video bitrate and resolution at 1000kb/s and 1080P, receptively. 2) *Varying video resolution*: we adjust the resolution of input video among 1080P, 720P and 480P, and set the system in normal condition. In both scenarios, we vary the chunk duration from 0.2 second to 2 second and report its influence to latency.

Fig 6(a) shows that the optimal chunk duration decreases with the increase of the system workload, i.e., the optimal chunk duration for busy, normal and empty are 0.6 seconds, 0.8 seconds and 1.4 seconds, respectively. In Fig. 6(b), our finding is that the smaller the video resolution, the smaller optimal chunk duration. Due to the fact that the lower video resolution requires less time to process each video chunk, the whole system configuration should be reduced as well for achieving smaller system latency.

Another finding from Fig. 6 is that when the video chunk is too small, the latency increases dramatically. It is because only a few frames are included in a chunk and the benefit of video compression is not sufficiently utilized. Also, more computing resources are wasted in encoding and decoding.

In the following evaluations, we use the obtained optimal settings of chunk duration. For more complex environment, in future, we can train reinforcement learning models to decide the optimal chunk duration automatically [32].



(a) Varying System Workload (b) Varying Video Resolution

Fig. 6: Latency V.S. Chunk duration

6.2 Evaluation

Simulation Configuration. We configure the system parameters according to Table 2, the input rate of each edge node is 30 frame/s, and video resolutions are 1080P, 720P and 480P, respectively. The network is the same as in benchmarking experiments (see Table 1). The system workload is set as no workloads (*empty*), normal workloads (*normal*) and heavy workloads (*busy*), with 0, 5 and 10 video chunks in the respective queues. Time interval Δt is 30 seconds for all experiments.

Evaluation Metric. We mainly consider two metrics in our evaluation: 1) *Latency* – the average latency of processing each video frame; this includes the data transmission time and processing time either on edge or cloud. 2) *Throughput*– the number of the video frame processed in each second.

Algorithms. Although we proposed a two-stage algorithmic strategy to overcome a potential issue that the PGD iterations could converge at non-stationary points, our extensive numerical experiments do not observe such a case. Hence we compared our algorithms PGD-VAO and PGS-VAO separately with *three* baseline solutions below. We implemented and parameterized these baseline techniques based on our system configurations.

- DeepDecision: considers the optimization problem that during each time interval, video is processed only on the edge side or the cloud side depending on system throughput.
- FastVA: considers making most use of network transmission to offload the video chunks to the cloud.
- HillClimb: utilizes the Hill Climbing [14] algorithm to solve our optimization problem.

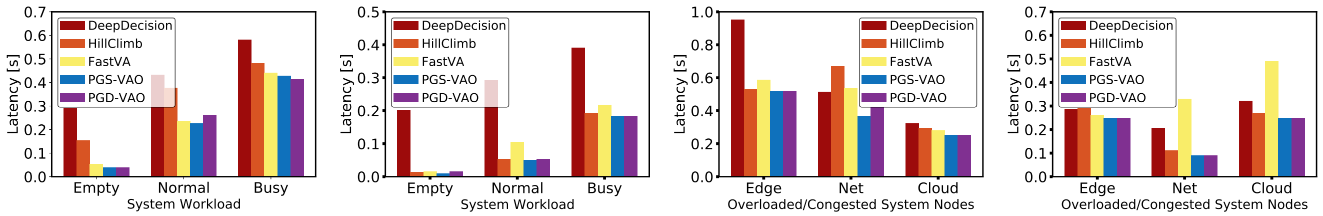
6.3 Comparison With Existing Approaches

In this subsection, we analyze the algorithm performance under various system conditions. For all experiments, optimal chunk duration is chosen based on the benchmarking results as shown in Fig. 6. We configure the number of edge node , cloud server and video resolution as 10, 1 and 720P respectively, if not otherwise stated.

6.3.1 The Impact of Network Bandwidth

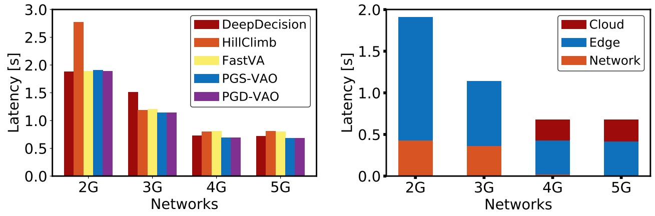
Fig. 8(a) shows that with the increase of the network bandwidth, the latency is greatly reduced. In 3G condition, the latency is 2× of that in 4G and 5G. However, when network is 4G and 5G, there is no obvious performance difference. This is because more data are transmitted to the cloud when the network speed increase, more computing resources from the cloud server are utilized until it is saturated.

In order to better understand the trends, we show the latency introduced by different components using PGS-VAO, according to our HQM in Fig. 8(b). In 2G and 3G network, the latency is introduced by data queuing for



(a) Uniform System Workload under 3G Network (b) Uniform System Workload under 5G Network (c) Imbalanced System Workload under 3G Network (d) Imbalanced System Workload under 5G Network

Fig. 7: Performance under Different System Workloads



(a) Algorithm Comparison (b) System Latency Proportion (PGS-VAO)

Fig. 8: Performance under Different Network Bandwidth

processing in the edge nodes and transmitting to the cloud. As the improvement of network, our algorithm can adapt to this change, offloading more data to the cloud to utilize the computing resources from cloud, thereby reducing latency.

6.3.2 The Impact of System Workload

We first test the general performance of the algorithms with uniform system workloads, i.e., *empty*, *normal* and *busy* workload for all system queues. Then, we tune the workload imbalance by setting overloaded edge and cloud nodes, and congested network conditions respectively. We set these nodes as *busy* while keeping others as *normal*. We conduct the experiments in both 3G and 5G networks.

Fig. 7(a) and 7(b) indicate that with the increase of the workload, the system latency is increasing as well. It is also obvious that HQM-based algorithms (e.g., PGS-VAO, PGD-VAO) perform better than non-HQM-based algorithms (e.g., FastVA), which performs better in 5G (see Fig. 7(b)).

Fig. 7(a) illustrates that our proposed algorithm can capture the the system bottleneck which is the network speed and understand that maximizing the network utility is the best strategy. FastVA shares the similar offloading strategy, thereby achieving similar performance compared to our algorithms. However, FastVA cannot adapt to the change of system status, using the same offloading strategy in 5G network (as shown in Fig. 7(b)). Since the network is not the bottleneck, the ideal offloading policy should be based on the processing capacity of both edge nodes and cloud server.

To further highlight the advantage of our algorithms, we imbalance the workload on different components modeled by our HQM. In particular, when the bottleneck is on the cloud/network, i.e., a lot of data is queuing for either transmitting or processing on the cloud, both FastVA and DeepDecision fail to make the right offloading decision. In 5G network (see Fig 7(d)), the situation is magnified because FastVA is able to push more data to the cloud server, without considering the queued data on network or cloud.

6.3.3 The Impact of Computing Resources

In this experiment, we evaluate the performance of the algorithms with different computing resources. We first fix the number of cloud servers from 1 to 5 and vary the number of edge nodes from 10 to 50 to report the system latency. Then we set edge nodes as 10 and 30 and vary the cloud nodes from 1 to 5. The experiment is conducted in 5G network, and results are shown in Fig. 9.

The findings are two-folds. First, from Fig. 9(a) and 9(b), we can see that the cloud processing capability has great impact on the overall system processing latency. When there is only one cloud node, the processing latency is approximately 5 × than the system with 5 cloud nodes. However, if the capacity of the cloud server becomes sufficient, adding more cloud resources will not affect the overall system processing latency (see Fig. 9(c) and 9(d)). When the ratio of number of edge node and cloud server is 10:1, the system latency can be reduced to less than 0.1s. The latency reaches 0.2s when the ratio is around 15:1. This is also revealed in Fig. 9(c) and 9(d), when the ratio exceeds 15:1, the overall latency increases dramatically. This indicates that one cloud server is not sufficient for supporting more than 15 edge nodes. In order to scale up the system, appropriate cloud servers should be added with the increase of edge nodes .

Moreover, our proposed method always performs better than baseline methods in any condition. There is no significant performance difference between PGS-VAO and PGD-VAO, but as discussed earlier, PGS-VAO is more stable in dealing with non-smooth value functions, it is still worth using PGS-VAO when the computation reaches break point.

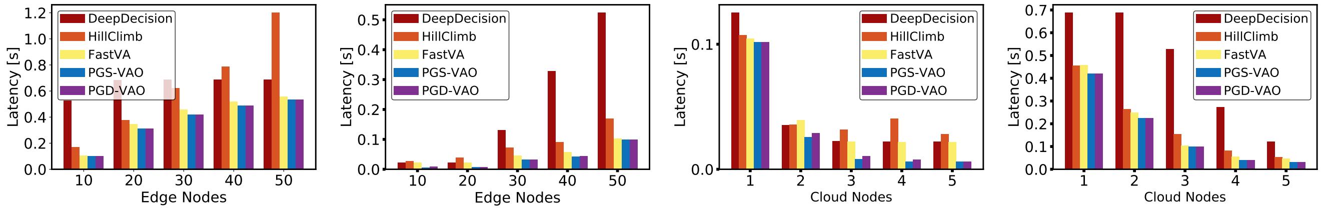
6.3.4 The Impact of Video Resolution

In this subsection, we vary the video resolution from 1080P to 480P and show its impact on latency in Fig. 10(a). Our proposed algorithm outperform all other baseline methods. In general, the latency of 720P and 480P video is about 1/2 and 1/6 of that in 1080P video, near linear reduction with resolution decreasing. The reduction of the latency is caused by two reasons: 1) less data needs to be transmitted 2) faster inference time. Video resolution also affects the system throughput as we will discuss in the next subsection.

6.4 Impact of Throughput Constraint

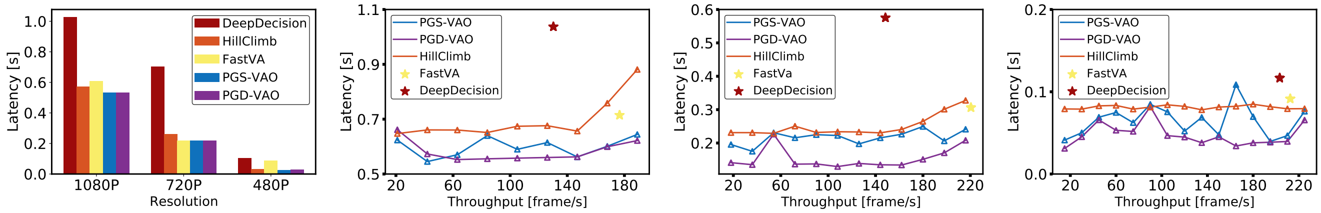
Recall E.q 4, our optimization problem is to minimize the system latency which is bounded by a pre-defined throughput threshold (I^* in C_1). In order to study the impact of the I^* on latency, our experiments are conducted under different video resolution, while we adjust the throughput constraint with different values.

Fig. 10(b), 10(c) and 10(d) shows that our proposed algorithms can achieve less latency, compared to other methods,



(a) 1 Cloud Node (b) 5 Cloud Nodes (c) 10 Edge Nodes (d) 30 Edge Nodes

Fig. 9: The Latency with Various Edge Nodes and Cloud Servers



(a) Impact of Resolution on Latency (b) Input Video Resolution (1080P) (c) Input Video Resolution (720P) (d) Input Video Resolution (480P)

Fig. 10: Impact of Throughput Constraint on System Latency with Varying Resolution

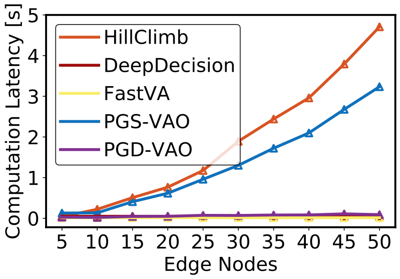
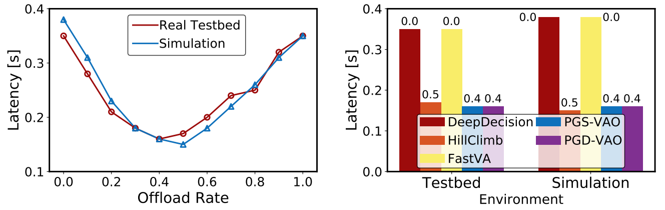


Fig. 11: Algorithm Computation Latency with Different Edge Nodes



(a) System Latency with Different Offloading Rate (b) Algorithm Performance in Different Environments

Fig. 12: Testbed vs Simulation

and ensure the latency is fluctuating in a certain range with the varying of throughput constraints. This fluctuation is caused by the network oscillation that we add into the simulated network environments. HillClimb is implemented based on HQM, but it fails to obtain the optimal solution in some cases. For example, Fig. 10(b) and 10(c) show that when the throughput is set higher than 180, the latency increases significantly. Moreover, FastVA and DeepDecision are not based on HQM, and they therefore do not have throughout constraint. We report the monitored throughput by using two algorithms.

Additionally, it is a challenge to set an optimal throughput constraint that is affected by many factors such as input stream rate, network conditions and video resolution etc. Throughout, in this paper, for each set of the experiment, we manually set the throughput constraint below the maximum system throughput to ensure smooth running of the optimization algorithms. A more advance method is desired to maximize the throughput in future work.

6.5 The Complexity Analysis of the Algorithms

In §5.5, we analyzed the algorithm complexity and we validate the analysis in this subsection. Fig. 11 shows the execution time of the HQM-based algorithms to compute an offloading solution. PGD-VAO outperforms the other two algorithms in terms of efficiency; its computation time increase *linearly* as the edge nodes increase. The computation time of PGS-VAO has a *superlinear growth* when the number of edge node increase. The results experimentally confirm our time complexity analysis for both PGD-VAO and PGS-VAO.

6.6 Real-world Test-bed

We evaluate the performance of the OsmoticGate using a lab test-bed in order to compare the real-system performance against simulation.

Lab Test-bed Configuration. The real-world test-bed is configured with Jetson Nanos and servers. We connect 4 Jetson-Nanos to the GPU server and set the connection as 5G with TC. To emulate the workload of the transmission queues, we maintain all transmission queues with 30 video chunks to saturate the cloud server. Time interval Δt is set as 30 seconds.

Fig. 12(a) reports the average queuing latency for both the simulation and the test-bed environment given different offloading rates. Generally, the computed simulation latency is quite close to the real test-bed. The difference between the two is less than 0.01 seconds for most cases and the two curves nearly overlap at several points. Small oscillation has been noted for the real test-bed as well. The complexity of the data transmission and processing pipelines, as well as the varying working conditions of the HQM, are all factors that introduce oscillation to the test-bed performance.

Fig. 12(b) compares our proposed algorithm and other baselines in both test-bed and simulation environments. Overall, the HQM based algorithms (i.e., PGS-VAO, PGD-VAO, HillClimb) can achieve better performance as compared to non-HQM based algorithms (i.e., DeepDecision, FastVA). HillClimb, PGS-VAO and PGD-VAO all reach near-optimal point around 0.5 and 0.4, as seen in Fig. 12(a), while DeepDecision and FastVA consider transmitting all processing tasks to the cloud. PGS-VAO and PGD-VAO outperform the other baselines by 2x. This huge difference has proved the necessity to consider system workload dynamics when making offloading decisions.

7 Related Work

Content Delivery Network. Content Delivery Network (CDN) has been well studied in many areas including vehicle monitoring [33], Unmanned Aerial Vehicle (UAV) monitoring [34] and smart city [35]. The main goal of CDN is to use various technologies such as caching [36], [37] or machine learning (ML) [38], [39] to optimize streaming data delivery at the network level. Our OsmoticGate can be built up on these underlying systems to have better performance of video analytics task offloading.

Video Streaming System. Video streaming systems aim to deliver video data under different network conditions, while meeting various QoS requirements, including latency, throughput and system re-buffering level. To this end, the adaptive video streaming algorithms aim to configure the video stream to achieve efficient video delivery. The existing work usually considers two factors: i) Rate-based algorithms that decide the bitrate based on network bandwidth assumption [40], [41], [42]. ii) Buffer-based algorithms that consider the client's playback buffer [43], [44]. This method keeps the system buffer at a stable level without sacrificing the video quality at large. Our work is built upon the modern video streaming technologies. Unlike the traditional video streaming systems that focus the QoS on video delivery, instead, we aim to offload the video processing tasks over edge-cloud environment.

Video Analytics Task Offloading. [10], [11], [12], [13] have been conducted on scheduling and configuring the video analytics jobs (queries). DeepDecision [5] considers both edge and cloud for conducting video analytics. However, it chooses CNNs only on one node (edge/cloud) for processing the videos. FastVA [6] considers offloading from local NPU to edge server, CrowdVision [45] considers client-server task offloading. However, They all fail to consider the system workload thus can not deliver optimal decisions under heavily-loaded systems. Also, these works treated video as a sequence of images, whereas we consider bitrate-based video analytics that benefit from modern video streaming protocols.

Task Offloading in Edge and Cloud Computing. Offloading techniques have been widely used to address the resource limitation of IoT applications [7], [8], [9]. These works are limited when the heavy transmission workload is posted by video analytics, where a huge amount of streaming video is uploaded to the cloud server. The complex encoding/decoding video processing pipeline complicates the system modeling as the change of video configurations may affect the computation and transmission at the same time. [7] studies computation offloading for Internet of Vehicles and proposes to solve a mixed-integer nonlinear programming problem with Edmonds–Karp algorithm. [8] studies generic edge-cloud task offloading and decomposes the optimization problem in a convex form where global optimal can be achieved via KKT conditions. [9] is another generic task offloading work and proposes simple logistics to optimize the QoS metrics. These works decompose the optimization problems into simpler convex forms and thus can be solved by heuristic or other simpler algorithms. However, as modeled in our paper, video analytics task offloading can be formulated as a constrained, non-smooth and non-convex optimization problem, can not be solved with above methods.

8 Conclusions

In OsmoticGate, we investigated video streaming processing task offloading in cloud-edge computing paradigm. Based on bitrate-based video streaming protocols, we propose a HQM that is capable of capturing system workload dynamics. We model the system latency and throughput and then formulate a non-smooth, non-convex, constrained min-latency optimization problem. A two-stage gradient-based algorithm has been proposed which features switching between PGS-VAO and PGD-VAO algorithms. We have analyzed the convergence bound of PGA-VAO. Using this bound, we give practical implementation criteria for switching between the two algorithms. Extensive benchmarking has been conducted that serve as the foundations of our experiments. Simulation results showed that the our algorithm outperforms baseline works. Also, the two-stage algorithm is stable given different throughput constraints and various system conditions, which confirmed its effectiveness.

Acknowledgment

This research is funded and aligned with the following EPSRC projects, EP/W003325/1 & EP/T021985/1, currently co-led by Prof. Ranjan and Prof. Zomaya. Zhenyu Wen is supported by the NSFC (Grant No. 62072408) and Ye Yuan is supported by the NSFC (Grant No. 61932004) and the Fundamental Research Funds for the Central Universities (Grant No. N181605012).

References

[1] B. Qian, J. Su *et al.*, "Orchestrating the development lifecycle of machine learning-based iot applications: A taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 53, pp. 1–47, 2020.

[2] K. He, X. Zhang *et al.*, "Deep residual learning for image recognition," in *(CVPR) 2016*.

[3] S. Ren, K. He *et al.*, "Faster r-cnn: towards real-time object detection with region proposal networks," in *(NeurIPS) 2015*.

[4] Nvidia, *Deep Learning Inference Benchmarks*. https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks

[5] X. Ran, H. Chen *et al.*, "Deepdecision: A mobile deep learning framework for edge video analytics," in *(INFOCOM) 2018*.

[6] T. Tan and G. Cao, "Fastva: Deep learning video analytics through edge processing and npu in mobile," in *(INFOCOM) 2020*.

[7] X. Wang, Z. Ning *et al.*, "Offloading in internet of vehicles: A fog-enabled real-time traffic management system," *(TII) 2018*.

[8] J. Ren, G. Yu *et al.*, "Collaborative cloud and edge computing for latency minimization," *(TVT) 2019*.

[9] X. Xu, Q. Huang *et al.*, "Intelligent offloading for collaborative smart city services in edge computing," *(IoT-J) 2020*.

[10] M. Salehe, Z. Hu *et al.*, “Videopipe: Building video stream processing pipelines at the edge,” in *(Middleware) 2019*.

[11] C.-C. Hung, G. Ananthan *et al.*, “Videoedge: Processing camera streams using hierarchical clusters,” in *(SEC) 2018*.

[12] H. Zhang, G. Ananthan *et al.*, “Live video analytics at scale with approximation and delay-tolerance,” in *{USENIX} ({NSDI} 17)*.

[13] J. Jiang, G. Ananthanarayanan *et al.*, “Chameleon: scalable adaptation of video analytics,” in *(SIGCOMM) 2018*, 2018, pp. 253–266.

[14] S. Russell and P. Norvig, “Artificial intelligence: a modern approach,” 2002.

[15] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv preprint arXiv:1804.02767*, 2018.

[16] J. Chen, “Globally solving nonconvex quadratic programming problems via completely positive programming,” *(MPC) 2012*.

[17] M. Parda, “Global optimization algorithms for linearly constrained indefinite quadratic problems,” *(Comp. Math. Appl.) 1991*.

[18] Y. Nesterov, “Gradient methods for minimizing composite functions,” *(Mathematical Programming) 2013*.

[19] J. Tang, M. Golb *et al.*, “Gradient projection iterative sketch for large-scale constrained least-squares,” in *(ICML) 2017*.

[20] S. U. Stich, C. L. Muller *et al.*, “Optimization of convex functions with random pursuit,” *(SIOPT) 2013*.

[21] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[22] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[23] J. Tang, M. Golb *et al.*, “Rest-katyusha: Exploiting the solution's structure via scheduled restart,” in *(NeurIPS) 2018*.

[24] J. Tang, K. Egia *et al.*, “Practicality of stochastic optimization in imaging inverse problems,” *(Trans. Comput. Imag.) 2020*.

[25] J. Tang and M. Davies, “A fast stochastic plug-and-play admm for imaging inverse problems,” *arXiv*, 2020.

[26] A. S. Lewis and M. L. Overton, “Nonsmooth optimization via quasi-newton methods,” *(Mathematical Programming) 2013*.

[27] J. V. Burke, A. S. Lewis *et al.*, “A robust gradient sampling algorithm for nonsmooth, nonconvex optimization,” *(SIOPT) 2005*.

[28] J. V. Burke, F. E. Curtis *et al.*, “Gradient sampling methods for nonsmooth optimization,” in *Numerical Nonsmooth Optimization*.

[29] D. Driggs, J. Tang *et al.*, “Spring: A fast stochastic proximal alternating method for non-smooth non-convex optimization,” *arXiv*.

[30] C. Canel, T. Kim *et al.*, “Scaling video analytics on constrained edge nodes,” *(SysML '19)*, 2019.

[31] Training dataset. https://drive.google.com/file/d/11CqVAl7R1523823F5smLU6gQSCOmfZ8w/view?usp=sharing

[32] H. Mao, R. Netravali *et al.*, “Neural adaptive video streaming with pensieve,” in *(SIGCOMM) 2017*.

[33] W. Xiong, C. Shan *et al.*, “Real-time processing and storage of multimedia data with content delivery network in vehicle monitoring system,” in *(WINCOM) 2018*. IEEE, 2018, pp. 1–4.

[34] A. Asheralieva and D. Niyato, “Game theory and lyapunov optimization for cloud-based content delivery networks with device-to-device and uav-enabled caching,” *(TVT) 2019*.

[35] M. Chen and L. Wang, “A computing content delivery network in smart city: Scenario, framework, analysis,” *(IEEE Netw.) 2019*.

[36] J. Ni and D. H. Tsang, “Large-scale cooperative caching and application-level multicast in multimedia content delivery networks,” *IEEE Communications Magazine*.

[37] T.-Y. Ku, J. D. Shin *et al.*, “Hybrid cache architecture using big data analysis for content delivery network,” in *(BdCloud) 2014*.

[38] Z. Chang and L. Lei, “Learn to cache: Machine learning for network edge caching in the big data era,” *(WCM) 2018*.

[39] A. Sadeghi, G. Wang *et al.*, “Drl for adaptive caching in hierarchical content delivery networks,” *(TCCN) 2019*.

[40] Z. Li, X. Zhu *et al.*, “Probe and adapt: Rate adaptation for http video streaming at scale,” *(JSAC) 2014*.

[41] Y. Sun and X. Yin, “Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction,” in *(SIGCOMM) 2016*.

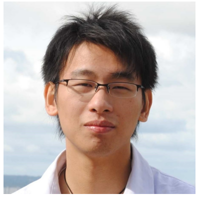
[42] J. Jiang, V. Sekar *et al.*, “{CFA}: A practical prediction system for video qoe optimization,” in *13th {USENIX} ({NSDI})*.

[43] J. Huang and R. Johari, “A buffer-based approach to rate adaptation: Evidence from a large video streaming service,” in *(SIGCOMM) 2014*.

[44] K. Spiteri, R. Urgaonkar *et al.*, “Bola: Near-optimal bitrate adaptation for online videos,” *TON*.

[45] Z. Lu, K. Chan *et al.*, “Crowdvision: A computing platform for video crowdprocessing using deep learning,” *(TMC) 2018*.

Bin Qian (Member, IEEE) received the M.Sc. degree in Data Science from University of Southampton, U.K, in 2018. He is currently pursuing the Ph.D. degree in computer science from Newcastle University, Newcastle Upon Tyne, U.K. His research interests include IoT, machine learning, task offloading.



Zhenyu Wen received the M.Sc. and Ph.D. degrees in computer science from Newcastle University, Newcastle Upon Tyne, U.K., in 2011 and 2016, respectively. He is currently a Professor with the Institute of Cyberspace Security and college of Information Engineering, Zhejiang University of Technology, China. His current research interests include IoT, crowd sources, AI system, and cloud computing.

Junqi Tang received the M.Sc. and Ph.D. from the Institute for Digital Communications, University of Edinburgh, U.K., in 2015 and 2019, respectively. Prior to that he received B.Eng in Communication Engineering from Sichuan University, China, in 2014. He is currently a postdoctoral research associate with the Department of Applied Mathematics and Theoretical Physics (DAMTP), University of Cambridge, U.K. His research interests include machine learning, explainable AI, large-scale optimization. , with applications in computer vision and image processing.

Ye Yuan received his BS, MS and PhD degrees in Computer Science from Northeastern University, China in 2004, 2007 and 2011, respectively. He is now a Professor at the College of Information Science and Engineering at Beijing Institute of Technology. His research interests include graph databases, probabilistic databases, data privacy-preserving and cloud computing.

Albert Y. ZOMAYA is Peter Nicol Russell Chair Professor of Computer Science and Director of the Centre for Distributed and High-Performance Computing in the School of Computer Science at the University of Sydney, Australia. To date, he has published over 700 scientific papers and articles and is (co-)author/editor of over 30 books. A sought-after speaker, he has delivered over 250 keynote addresses, invited seminars, and media briefings. He is currently the Editor in Chief of the ACM Computing Surveys and served in the past as Editor in Chief of the IEEE Transactions on Computers (2010-2014) and the IEEE Transactions on Sustainable Computing (2016-2020).

Rajiv Ranjan is a Full professor in Computing Science at Newcastle University, United Kingdom. Before moving to Newcastle University, he was Julius Fellow (2013-2015), Senior Research Scientist and Project Leader in the Digital Productivity and Services Flagship of Commonwealth Scientific and Industrial Research Organization (CSIRO C Australian Government's Premier Research Agency). Prior to that he was a Senior Research Associate (Lecturer level B) in the School of Computer Science and Engineering, University of New South Wales (UNSW). Prof. Ranjan has a PhD (2009) from the department of Computer Science and Software Engineering, the University of Melbourne.