A Generic Framework for Deploying Video Analytic Services on the Edge

Vassilios Tsakanikas, *Student Member, IEEE,* Tasos Dagiuklas, *Senior Member, IEEE*

Abstract—This paper introduces a novel distributed model for handling in real-time, edge-based Artificial Intelligence analytics, such as the ones required for smart video surveillance. The novelty of the model relies on decoupling and distributing the services into several decomposed functions which are linked together, creating virtual function chains (VFC model). The model considers both computational and communication constraints. Theoretical, simulation and experimental results have shown that the VFC model can enable the support of heavy-load services to an edge environment while improving the footprint of the service compared to state-of-the art frameworks. In detail, results on the VFC model have shown that it can reduce the total edge cost, compared with a Monolithic and a Simple Frame Distribution models. For experimenting on a real-case scenario, a testbed edge environment has been developed, where the aforementioned models, as well as a general distribution framework (Spark ©) and an edge-deployement framework (Kubernetes©), have been deployed. A cloud service has also been considered. Experiments have shown that VFC can outperform all alternative approaches, by reducing operational cost and improving the QoS. Finally, a caching and a QoS monitoring service based on Long-Term-Short-Term models are introduced and evaluated.

Index Terms—edge computing, AI applications, Virtual Function Chaining, caching, cost optimization, long-short term memory, QoS constraints

1 Introduction

Artificial Intelligence (AI), as expressed by the latest developments of Machine Learning (ML) and Deep Learning (DL), has produced numerous models which are mature enough to reach the market massively during the next few years. The main reasons for this, mainly involves the improvements on data-capturing devices, the re-engineering of several ML algorithms and the release of ML and DL toolkits, like PyTorch and TensorFlow [1].

Video analytics is an umbrella term for describing applications like object tracking, pedestrian detection, face recognition, behavioral analytics etc. The common business - technology model for deploying AI surveillance services nowadays is Cloud Computing [2], where the captured video streams are uploaded to a centralized data center. This imposes a round-trip time to the throughput of the service which, in many cases reduces significantly the Quality of Service (QoS). This leads the service providers to either reduce the processing frames per second (fps) or lower the resolution of the captured videos, nullifying the advances of new video sensors, like UHD and HDR sensors.

Edge Computing has been proposed as a computing paradigm according to which the data are processed *'near'* the generating data devices and comprises many low-capacity devices [3]. While this paradigm addresses the large round-trip times of Cloud Computing, the QoS is now limited due to the capacity of the edge devices. Not only academia, but lately industry has placed focus on Edge Computing, by providing software (e.g., Google Lite TensorFlow©) and hardware (e.g., NVIDIA Jetson AGX Xavier© and AWS DeepLens©) solutions which are suitable for edge processing. As discussed in several works ([4], [5]), current edge computing approaches face several challenges and limitations. These limitations mainly involve the insufficient computational capacities for heavy loaded tasks, like AI model training and the low storage space usually the edge devices are equipped with.

Cooperative and distributed models with the capacity to address the aforementioned limitations and enable real-time processing at the edge of the network are widely discussed in several recent research works [6]. The obvious reasons for targeting this research area mainly involve the efficient harvesting of the computational resources of the heterogeneous edge networks while reducing the total operational cost of such networks.

This paper proposes a novel distributing framework which explores the Virtual Function Chaining (VFC) concept inspired from the Software-Defined Networks and enables the real-time inference for surveillance applications at the Edge, supported by edge learning services build on deep learning models with the capacity to monitor, assess and predict the QoS of the supported services. In this model, an AI smart video analytics service is decomposed to a set of Virtual Functions (VFs), which can be deployed on different edge devices. Using these VFs, a VFC is created which process the streaming data in a distributed fashion.

The VFC framework deploy several modules which aim to optimal design the service, monitor its QoS metrics and fine-tune its configuration in order to avoid failures. More specifically, a computational engine is responsible for proposing the optimal setup of the VFC while an edge-learning service monitors the performance of the edge devices and propose possible alterations.

Authors, in a recent publication [7] have presented the VFC architecture for effectively distributing an AI

(Corresponding author: VT)
Authors are with the Smart Internet Technologies (SuITE), Division of Computer Science and Informatics, School of Engineering, London South Bank University

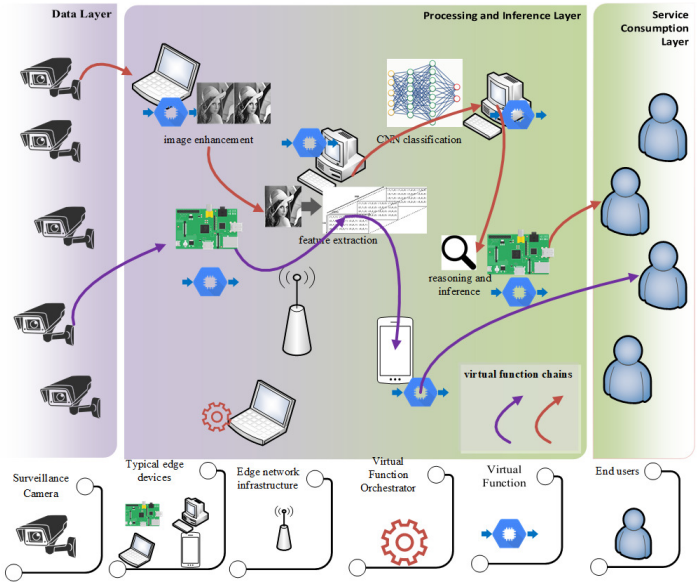

Fig. 1. Conceptual architecture of the proposed system.

surveillance service to the Edge. This architecture seamlessly integrates VFCs. The proposed architecture's services are mainly hosted on the Virtual Function Orchestrator (VFO). In the same publication, a model for designing the optimum setup for a VFC, in terms of VF instances and VF placement on the edge devices is presented. Each VF may appears in the VFC at several instances(replicas) and interconnected $VFCs$ (Fig. 2). This publication acts as the base for extending our work. The contributions of this manuscript mainly include:

1) An edge learning service is built on deep learning models for assessing the QoS of the deployed services and alerting in case a VFC is about to fail.
2) A caching mechanism, which demonstrates the scalability of the proposed model when multiple services are requested.
3) A prototype of the described architecture, which is used to evaluate the described models and provide a proof of concept, in terms of effectiveness and feasibility, on enabling $VFCs$ as a model for real-time AI surveillance applications.

Additionally, within this work an extensive comparison with generic distribution engines (Spark©, Kubernetes©) is presented, along with a set of new experiments.

While the proposed model is inspired by the Service Function Chaining (SFC) concept, it alters and extends several of its features, in order to meet the requirements of video analytics. First, it introduces a load-balancing mechanism connected with the desired QoS, which monitors the output of the service and rearranges the VFC automatically. Additionally, it extends the SFC model by allowing one VF instance to be part in several $VFCs$ (e.g. face detection, gender identification, etc.), so that several video analytic services can be deployed simultaneously. The different deployment modes of the $VFCs$ are presented in Fig. 2. The management of the VFC (autoscaling, QoS monitoring, etc.) is facilitated by an edge-learning model with the capacity to assess the performance of the edge devices within a specific time-frame and inform the VFO about possible failures.

The rest of this paper is organized as follows. Section 2 provides a brief state-of-the-art approaches for utilizing Edge for hosting AI services, while Section 3 describes the VFC model. Section 4 includes the implementation details of all the tools developed to evaluate the proposed concept. In Section 5, the results from the experiments are presented. Finally, the results are discussed in Section 6 and the conclusions and future work are drawn in Section 7.

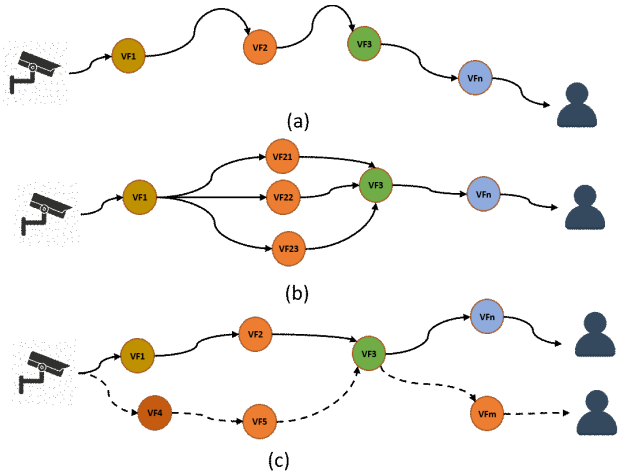

Fig. 2. Different modes of VFC deployment. (a) Basic VFC deployment, (b) VFC deployment with VF replicas and (c) Two $VFCs$ with caching enabled.

2 State of the Art

During the last decade, academia and industry have introduced a set of novel architectures which aim to efficiently utilize low-end hardware at the edge of the network, near the data generation sites. Such architectures, like Multi-access Edge Computing, Fog Computing, Cloudlet Computing, and Mobile Cloud Computing, while differ in several aspects, like communication network ownership, device mobility and device power supply, all share in common the same need for distributed architectures and approaches, which can enable the hosting of more demanding services [5]. Additionally, most of the relevant technologies which support Internet of Things and management of data coming from sensing devices are discussed in [8].

Several research studies have been focused on enabling edge computing to support demanding latency sensitive applications. Author in [9] has proposed a scheme for handling mass video data coming from city surveillance services on heterogeneous digital devices. Zhou *et al.* [10] have described a model for offloading cloud by utilizing an edge meshed network. Li *et al.* [11] have proposed a general virtualization architecture, based on VMs, mainly focusing on its networking aspects. Chen *et al.* [12] have described an architecture which explores fog computing as a processing infrastructure for supporting dynamic urban surveillance streams. Dautov *et al.* [13] have performed a comparison study among cloud, fog and edge computing

for supporting intelligent surveillance applications. Mao *et al.* [14] presented a wireless powered mobile-edge computing system where the users have the capacity to share communication and computation resources, under an efficient cooperation scheme, after formulating the relevant optimization problem and solving it. Finally, Chen *et al.* [15] presented an IoT-based smart grids model, which facilitate the connection, management and real-time analysis and processing of massive data, based on digitalization of power smart grids.

The authors in [16] provide a survey of the applications that can be supported from Edge Computing. Finally, the author in [17] provides a holistic vision about surveillance applications on edge/fog computing paradigms, where the basic concepts, challenges and opportunities are discussed.

Our model has compared with the current state-of-the art literature and does not require any special virtualization (e.g. Virtual Machines) [11] or distribution [18] (e.g. Apache Spark©) middle-ware in order to perform the real-time calculation of AI analytics, offloading the edge devices from the substantial overhead these approaches require. Additionally, surveillance applications are decomposed in virtual functions that are deployed in nodes with the available resources. Such functions are scaled up based on demands.

Deep learning techniques have been widely used during the last years to extract information from various kinds of data ([19]). Depending on the characteristics of input data, several architectures for deep learning have been proposed, such as the recurrent neural networks ([20]), convolutional neural networks ([21]), and deep neural networks ([22]). As deep and convolutional networks do not have the capacity to manage temporal information of input data, areas involving data such as text, audio or video, recurrent neural networks (RNNs) are usually applied. More specifically, there are two types of RNNs: discrete-time RNNs and continuous-time RNNs([23]). The main characteristic of the RNN architecture is a cyclic connection, which enables the RNN to possess the capacity to update the current state based on past states and current input data.

Long short-term memory (LSTM) networks have been proposed for input data which hold dependencies with a large temporal distance [24], which fit the problem described in the present model.

3 Virtual Function Chaining Model

Aiming to enable edge as a real time inference mechanism for AI video analytics services, a generalization distribution framework is proposed, according to which a video analytic service is decomposed to a set of VFs, creating a VFC. The proposed model aims to facilitate the efficient offloading of surveillance cloud services to a cooperative distributed edge environment, where heterogeneous edge devices formulate a service chain and jointly implement a service.

The basic principles of the proposed model (Figure 1) are the following:

- Each surveillance service is decomposed to set of processes. Each process implements certain tasks, like image enhancement, edge detection, AI model inference, etc.,
- Each process instantiates a VF and is deployed as an edge node. Each VF comprises three parallel threads: the Input Queue, the Output Queue and the Running agent (Fig. 3). The running agent implements the logic part of the VF (e.g. image filtering) while the Input and Output queues handle the packaging and communication of the VF with the previous VF and the next VF in the VFC respectively. The communication between the VFs is unidirectional, as surveillance services are streaming processes.
- A surveillance service is realized by a VFC, similar to the service function chaining proposed by the IETF WG [25]. A VFC must include at least one instance of each VF. The main concept of the model proposed by [25], includes network services, like firewall and packet filtering.
- VFO manages the VFs allocation to the physical devices and established the communication channels among them. Additionally, VFO monitors the performance criteria of the service (e.g. processed frames/sec, total cost, etc.) and performs actions in order to meet them, while hosting the edge learning AI model for QoS monitoring.

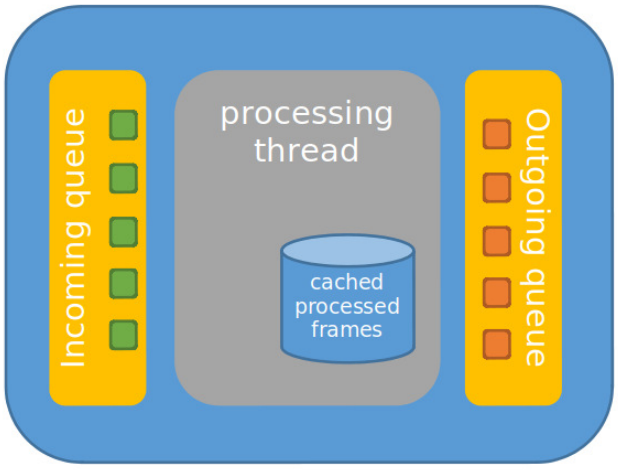

Fig. 3. Implementation of a VF.

3.1 Model Formulation

Each service is described by a VFC, and in general, a single VF can have multiple instances (replicas) within the edge environment. The rationale behind the replication of the VFs is that when a VF requires a processing time larger than the one prescribed by the required QoS (note that (i) the processing time of the whole VFC is actually equal with the processing time of the most demanding VF in the VFC, and (ii) the processing time of the VFs must be almost identical, due to the streaming nature of the described services. If VF_i produces data faster than VF_{i+1} can consume, then data will flood VF_{i+1} leading to the overflow of the input queue of the relevant edge device.)

When a user subscribes to a service (e.g. object detection, etc.), VFO instantiates the VFC by implementing the following tasks:

TABLE 1
Basic entities of the proposed VFC model.

Entity	Formulation	Description
Surveillance service	$\overrightarrow{S} = [fps, \{VF_j\}]$	fps is the processed frames / sec the service requires (QoS)
		$\{VF_j\}$ is the set of processes comprise the service
Virtual Function	$\overrightarrow{VF} = [cpu_{load}, outdata]$	cpu_{load} is the required CPU instructions per frame
		$outdata$ is amount of data virtual function produces after processing the input data
Edge Node	$\overrightarrow{K} = [m, c(l), r(l)]$	m is the CPU instructions / sec the device can execute
		$c(l)$ is the cost function of the device, when performing l CPU instructions. Cost is a general term which includes battery life, maintenance cost, etc.
		$r(l)$ is the required time to process l CPU instructions
Link	$\overrightarrow{W_{ab}} = [bw]$	bw is the communication bandwidth among edge nodes a and b

1) Calculates the required number of instances (replicas) for each VF, in order to meet the service's QoS criteria,
2) Allocates the VFs to edge devices and
3) Establishes communication channels between the edge devices.

A VF instance can be part of several service chains. Table 1 summarizes the formulations of the main entities of the proposed modeling framework. It is important to mention that we consider as the primary component of the QoS the number of frames the service can successfully process per second. This processing frame-rate influences the data volume injected in the edge network and thus is is correlated with the capacity of the distributed network to undertake specific load. As there are additional components that could be considered, as the resolution of the frame, we chose not to consider them for defining the QoS, due to the fact that frame-rate has a much higher contribution to the data volume.

3.2 Problems definitions and solutions

VFO node needs to assign the VFC to the edge environment. In order to achieve this, the following general assignment constrained problem needs to be solved:

Problem 1*: Determine the number of VF instances (replicas) and assign each instance to an edge device, such that the video analytics are generated while maintaining the required fps (as imposed by the QoS), and the total edge environment cost be minimum.*

Problem 1 can be formulated as:

$$min \left\{ \sum_{i=1}^{n} \sum_{j=1}^{y} x_{i,j} c_{i,j} \right\} \quad (1)$$

$$min \left\{ y = \sum_{j=1}^{m} instances_j \right\} \quad (2)$$

$$\sum_{j=1}^{y} x_{i,j} = 1 \quad (3)$$

$$\sum_{i=1}^{n} x_{i,j} \leq 1 \quad (4)$$

$$time_{computational} + time_{communication} \leq \frac{1}{fps} \quad (5)$$

, with

$$time_{computational} = \max_{i=1..n}^{j=1..y} \{x_{i,j} t_{i,j}\}$$

$$time_{communication} = \max_{i,i'=1..n}^{j=1..y-1} \left\{ x_{i,j} x_{i',j+1} \frac{output_j}{W_{i,i'}} \right\}$$

, where n is the number of edge devices, m is the number of the VFs, $x_{i,j} = \begin{cases} 1 & \text{if } VF_j \text{ is assigned on node } n_i \\ 0 & \text{otherwise} \end{cases}$,

$t_{i,j}$ is the required time for device D_i to execute VF_j and process the data produced from a single frame and $instances_j$ is the number of the required instances for VF_j. Finally, $W_{i,i'}$ is the bandwidth of the communication link between $node_i$ and $node_{i'}$, which host two adjacent VFs, j and $j+1$. This formulation describes a model which aims to minimize the total cost of the service (eq. 1) while meeting the QoS constraints (eq. 5), with the minimum number of VF instances (eq. 2), requiring that each VF instance must by assigned to exactly one edge device (eq. 3) and each edge device can undertake no more than one VF instance (*eq. 4*). It would be possible to allow an edge device to execute more than one VF by eliminating *eq. 4*. In the context of this work, edge nodes are considered low-end devices with limited resources, thus overwhelming them with more than one VF is out of scope.

This is a NP-Hard problem [26], which requires a substantial computational time to be solved. In order to acquire a feasible solution within an acceptable time-frame, we decouple *Problem 1* to two sub-problems: (A) VF instances sub-problem and (B) the assignment (placement) sub-problem.

Sub-problem A aims to identify the minimum number of instances for each VF. As discussed in the previous section, one instance from each VF must be deployed on the VFC, in order to support the service. Let $GVF = [VF_1, VF_2, ..., VF_n]$ be the set of the first instances of each VF. Each one of these VFs will be deployed to a different edge device. At this point of the assigning workflow, the allocation cost is not considered. Yet, we seek if there is a feasible solution of the placement, such that the QoS constraint is met on a specific edge instance. This results to the following relaxation.

$$min \left\{ \max_{i=1..n}^{j=1..y} \{x_{i,j} t_{i,j}\} + \max_{i,i'=1..n}^{j=1..y-1} \left\{ x_{i,j} x_{i',j+1} \frac{output_j}{W_{i,i'}} \right\} \right\} \quad (6)$$

$$\sum_{j=1}^{m} x_{ij} = 1 \quad (7)$$

$$\sum_{i=1}^{n} x_{ij} \leq 1 \quad (8)$$

Regarding $t_{i,j}$, it can be calculated using the $r_d()$ function of the edge device d. Thus $t_{i,j} = r_i(N_j)$, where N_j refers to the CPU instructions required by VF_j to complete its task.

Equation 7 reflects the fact that each VF from the GVF set must be appointed only to one node and (eq. 8) that each edge node can not undertake more than one VF. *Sub-problem A* can be solved in a polynomial time by modeling it as a Min Cost Flow problem [27]. The utilized solver is a typical Hungarian algorithm. This process results to an allocation of the GVF with the minimum required time that the network can support. Let t^* be the resulting time. If $t^* \leq \frac{1}{fps_{serv}}$, then the edge network can support the service without having to replicate a subset of the VFs. In this case, we can re-formulate the assignment problem as a constrained mixed integer problem, with the following formulation:

Sub-problem B:

$$min \left\{ \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} c_{i,j} \right\} \quad (9)$$

$$\sum_{j=1}^{m} x_{i,j} = 1 \quad (10)$$

$$\sum_{i=1}^{n} x_{i,j} \leq 1 \quad (11)$$

$$\max_{i=1..n}^{j=1..y} \{x_{i,j} t_{i,j}\} + \max_{i,i'=1..n}^{j=1..y-1} \left\{ x_{i,j} x_{i',j+1} \frac{output_j}{W_{i,i'}} \right\} \leq \frac{1}{fps_{serv}} \quad (12)$$

The objective function (eq. 9) of this formulation aims to minimize the total cost of the VFC deployment to edge network, while fulfilling the QoS constraints of the service (eq. 12) and assigning all VFs to a device (eq. 10) while limiting the number of deployed VFs to a device (eq. 11). This problem, can be solved by utilizing Constrained Programming [28] in a polynomial time.

3.2.1 Solving Problem 1

If $t^* > \frac{1}{fps_{serv}}$, then the computational capacity of the edge devices is insufficient to support the service's QoS, if only one instance of each VF is deployed. In order to tackle this, we draw inspiration from the recently launched concept of Cloud-native functions [29], which handle dynamically the number of their instances aiming to handle the incoming requests. Thus, we propose a mechanism according to which a VF can be launched to multiple devices, and share the data coming from the previous VF of the VFC following a round-robin approach. According to this mechanism, VF replicas are deployed to different nodes, and each replica undertakes a portion of the data produced by the previous VF on the chain.

Thus, if a second instance of VF_j is deployed, the required time for the function to process the data related to a single frame changes from $r_k(l)$ to $\frac{r_k(l)}{2} + b$, where b is the time overhead implied for handling the data separation on VF_{k-1} (previous VF in the chain) and data merging on VF_{k+1} (next VF in the chain), assuming that the two instances of the VF is deployed to identical nodes.

This case leads as to the formulation of a new sub-problem (*sub-problem C*). Its objective is to identify the minimum number of replicate instances for each VF that need to be deployed on the edge network, in order to meet the QoS constraints. Let $\overrightarrow{S} = [s_1, s_2, ..., s_m]$ represent the number of instances for each one of the VFs, with s_i being an integer larger or equal to one ($s_i \geq 1$). Thus, we derive the following problem formulation:

$$min \left\{ \sum_{j=1}^{m} s_j \right\} \quad (13)$$

$$\sum_{j=1}^{m} \left(\frac{r_f(N_j)}{s_j} + (s_j - 1)b + \frac{output_j}{W_{ff'}} \right) \leq \frac{1}{fps_{serv}} \quad (14)$$

, where N_j are the required instructions per frame required for executing VF_j on device f, with f' undertaking VF_{j+1}. Equation 13 drives our model to produce solutions with the minimum total new instances of the VFs, while (eq. 14) satisfies the QoS of the surveillance service. Unlike the problem formulated by *eq. (9-12)*, this is a non-linear mixed integer problem, which requires the utilization of the active set solver APOPT [30]. Let the result of this sub-problem be $instances = [ins_1, ins_2, ..., ins_m]$. Using $instances$, we can revisit *Sub-problem B* and solve the allocation problem as before.

The discrepancy of the latest described sub-problem is that the utilized nodes f and f' are unknown. This is rational, because we seek the number of the VF instances with regard to the computational time, which depends not only on the VF load but also on the node that will undertake the VF. We resolve this deviation by using Algorithm 1. This algorithm can provide two approaches: (a) worstCase scenario, where the edge device used to calculate (eq. 14) is the one with the lowest processing capacity and (b) bestCase scenario, where the highest processing capacity device is utilized.

Algorithm 1: Online placement optimization algorithm.

```
Input: fps, x_ij, ε (error tolerance)
deploy(x_ij);
fps_current = measure_fps();
if |fps − fps_current| < ε then
    return x_ij
else
    while (|fps − fps_current| > ε) do
        sleep(1);
        if (fps < fps_current − ε) then
            x_ij = addReplicate(x_ij)
        else if (fps > fps_current + ε) then
            x_ij = removeReplicate(x_ij)
        else
            return x_ij
        end
        deploy(x_ij);
        fps_current = measure_fps();
    end
end
```

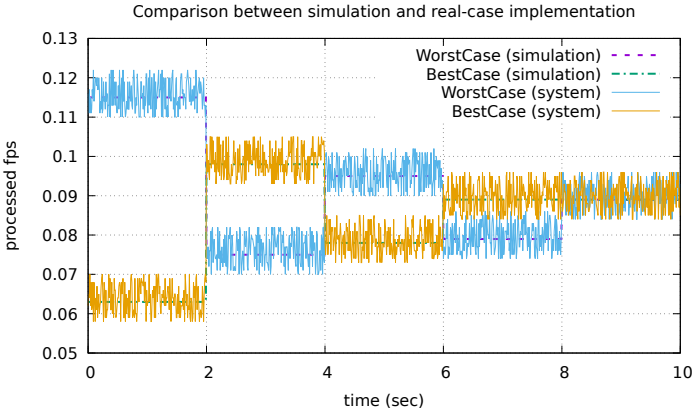

Fig. 4. Algorithm 1 comparison between modeling and real-case system implementation.

Algorithm 1 receives as input the processing fps implied by the QoS and the allocation of the initial instances of the VFs. As depicted in Fig. 4, both approaches converge to the desired processing fps. The reported results have been derived by a simulation framework that has been developed in order to evaluated the reported approach (simulation plots) and by an experimental edge network setup (system plots). Details on the utilized edge environment are provided in Section 4.2.

As far as the two functions ($addReplicate()$ and $removeReplicate()$) used in Algorithm 1, they calculate for each VF either the improvement (for the $addReplicate()$) or the regression (for the $removeReplicate()$) a new instance will have to the total cost. Let $\{v_i\}$ be these values. Then, we choose the VF which minimizes the difference $|fps_{current} - v_i|$. In each iteration, *Sub-problem B* is solved.

Aiming to evaluate the accuracy of the proposed algorithm for solving *Problem 1*, a set of scenarios (edge networks and $VFCs$) have been setup, solving *Problem 1* both using a naive greedy algorithm (which calculates the optimum solution) and the proposed approach. Five different $VFCs$ have been used. For each VFC, 15 scenarios have been established by setting the parameters presented in Table 2. The parameters were acquired from the experimental setup, after following the measurement approaches suggested in [31] and in [32]. Figures 5 and 6 present the results of this comparison. The reported results are the average values of the total cost among the 15 different scenarios. The total average extra cost imposed by the proposed approach was $\approx$ 1.97%, while the required time for each algorithm to solve the problem is $\approx 5.61sec$ for the proposed approach and $7.89 \times 10^3 sec$ for the greedy algorithm (Fig. 6). The solvers, which have been implemented using GEKKO suite [33], have been executed on a Intel i7 2.8GHz (8core) on 8GB of RAM.

3.3 QoS monitoring and failure avoidance

The model described in the previous sections is used in order to initialize and instantiate a VFC for serving a surveillance service. Yet, during the execution time of a service, the edge environment, unlike cloud infrastructures, is highly dynamic. The edge devices, due to low resources,

TABLE 2
Parameters on experiment setup.

Parameter	Value
cpu_{load}	$10^4\mathcal{N}(10, 0.8)$
m_k	$10^3\mathcal{N}(20, 0.9)$
Wtt'	$10^5\mathcal{N}(10, 0.7)$
$output$	$10^3\mathcal{N}(200, 0.65)$

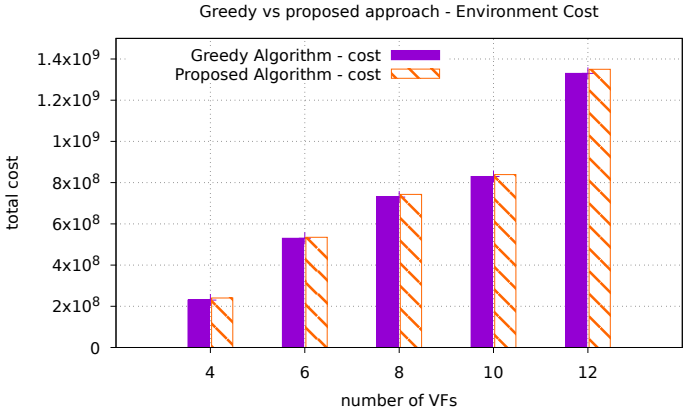

Fig. 5. Comparison between the optimum solution (greedy algorithm) and the proposed algorithm - Environment cost.

appear fluctuations in their main performance metrics, like available CPU and RAM. A surveillance service, in order to maintain the QoS standards, adequate resources are required through its lifecycle. The proposed VFC model is more prune to the fluctuations on the performance indicators compared to the $Monolithic$ approach, as it depends not only from one edge device but from a set of edge devices, where if one of the hosting devices fail (overload, battery drain, network disconnection), the whole service collapses.

Aiming to address this issue, a *"failure alert"* methodology has been designed and developed, based on a well established Recursive Neural Network, the Long-Short Term Memory (LSTM). By *failure* we consider the overloading of an edge node at such level that the assigned VF can no

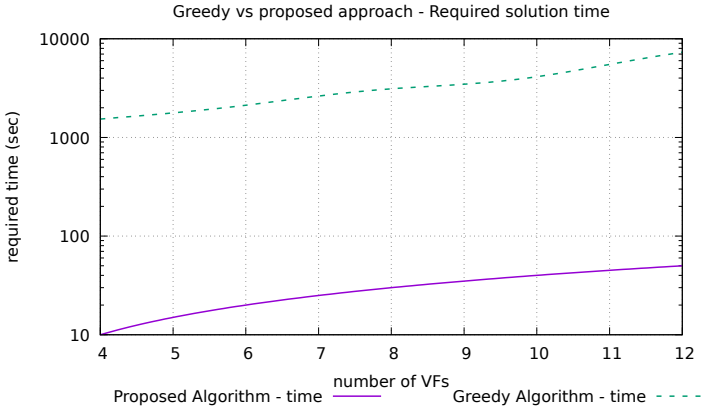

Fig. 6. Comparison between the optimum solution (greedy algorithm) and the proposed algorithm - Required solution time.

longer be executed properly, in terms of assigned mips.

A variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by [34]. It combines the forget and input gates into a single "update gate" while it also merges the cell state and hidden state compared to the classic LSTM cell. The core memory cell of the utilized network is presented in Fig. 7 and governed by (eq.15) - (eq. 18).

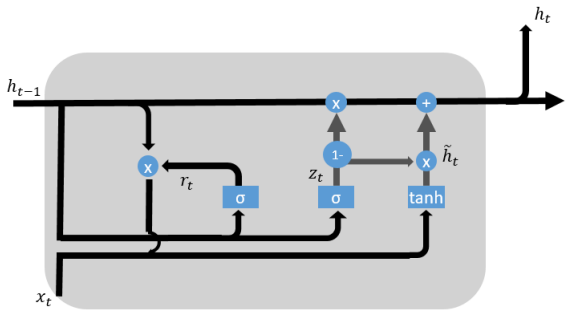

Fig. 7. LSTM memory cell.

$$z_t = \sigma(W_z) \cdot [h_{t-1}, xt] \quad (15)$$

$$r_t = \sigma(W_r) \cdot [h_{t-1}, xt] \quad (16)$$

$$\widetilde{h_t} = \tanh(W \cdot [r_t * ht - 1, x_t]) \quad (17)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \widetilde{h_t} \quad (18)$$

The model learns long term dependencies on the performance metrics of the edge devices. More specifically, two LSTM models have been established, one for predicting CPU usage and one for predicting RAM usage. The training datasets have been created using the benchmark edge environment and by mimicking artificial fluctuations on the edge devices. Details on the process are provided in Section 4.3. When the inference of the model predicts high CPU and / or RAM utilization, it informs the VFO for the specific VFC that is possible to face a failure within the specific time-frame. At this point, VFO recalculates the optimal placement of the VFC and resets the VFC.

The proposed QoS monitoring model aims to increase the robustness of the VFC model when the edge devices have low capacity on CPU and RAM resources compared with the relative demands of the VFs. Thus, even when considering advanced edge servers, with considerably high capacities on CPU and RAM, the proposed QoS model can benefit the VFC model in cases when (i) more demanding VFs are required, like deep learning models and (ii) edge devices host multiple tasks (aside VFs), thus requiring excessive resources.

3.4 Caching Mechanism

Surveillance as a service is one of the most promising models for delivering surveillance analytics to the end users. According to this model, a user can choose a camera feed and request for specific analytics. It is important to consider that the VFC model enables the sharing of the results among different services, in case two (or more) services share the same(s) VFs. For instance, a municipality offers video analytics services on live video streams from the busiest shopping streets of a region. A user can request a service named "Count women" from a specific video camera for two weeks, as she / he is interested on opening a beauty salon while another user request a service named "Detect abandoned items" from the same video stream. While the two services seems to have nothing in common, they share a subset of common VFs, *image-enhancement* and *light-equalization*.

For this, a caching mechanism has been designed and developed, according to which when a node executes a VF for data related with frame k, it stores the results in a stack for a specific time-frame. In case another video analytics service request from the same VF to process data related to an already processed frame, it retrieves the results and forwards them to the next VF of the VFC, without recalculations.

4 System Implementation

Aiming to evaluate the VFC model described in Section 3, both simulation and prototype platforms have been developed, where all the necessary functionalities have been deployed to support AI real-time video analytics of surveillance services. More specifically, three discrete tools have been formulated: (i) a simulation platform, (ii) an edge benchmark environment and (iii) a cloud-based surveillance service.

4.1 Simulation Platform

The simulation platform has been developed under Python 3.6. All the entities described in Table 1 have been modeled as distinct processes. Linux commands *cpulimit* and *ulimit* have been utilized to mock specific computational capacities for each 'edge device'. In the experiments described in the Section 5, each video analytic service has been modeled as a set of n VFs, where n is a random integer $\in [3, 7]$. Each VF could be either a *light* VF, a *moderate* VF or a *heavy* VF, with relative computational characteristics each. Finally, each VF may by identical with another VF with a probability of 15%, enabling the caching mechanism described in the following sections.

Withing the simulation platform, three different *Setups* of a surveillance service have been implemented.

- **Setup I**: The surveillance service has been implemented under a monolithic approach. This means that each deployed surveillance service has been hosted in one edge device (details in 4.4.3).
- **Setup II**: The surveillance service has been implemented under a simplistic distributed method, where each surveillance service were deployed to multiple edge devices (details in 4.4.4).
- **Setup III**: The proposed VFC model.

All of the above setups have been tested under different service demand probability distributions. By service demand probability distribution, we refer to the probability a user requests a service at a specific time-point t_d. More specifically, various Poisson distributions have been used. For the Poisson distributions, three different λ parameters have been used, aiming to mimic low, normal and high user

demand rates, according to [35], [36] and [37]. For the same Setup, the cumulative number of the requested services was the same. All the different scenarios are presented in Figure 8.

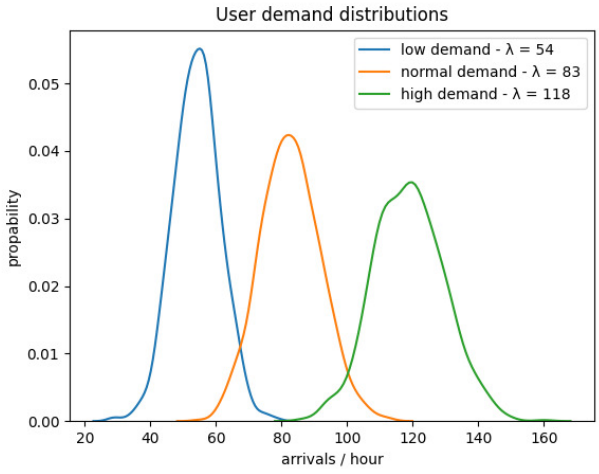

Fig. 8. Probability distributions for user demand.

4.2 Edge environment implementation

The implemented edge network comprise 6 Raspberry PI 3 (model B+) devices, with a Quad Core 64bit CPU @ 1.2GHz and 1GB RAM and 2 Raspberry PI 4 devices with a Quad core Cortex-A72 (ARM v8) 64-bit CPU @ 1.5GHz with 4GB RAM, running Raspbian OS. The feed from the camera has been mocked as video file from the VIRAT dataset [38].

Two video analytics services have been deployed on the edge environment. *Service A* and *Service B*, require gender and age classification respectively. For the main inference model, the pre-trained deep learning models proposed in [39] have been used. Both *Service A* and *Service B* decompose to 4 VFs. *Service A* includes $VF_1()$, $VF_2()$, $VF_3()$ and $VF_4()$, while *Service B* includes $VF_1()$, $VF_2()$, $VF_5()$ and $VF_6()$.

Details on the VFs are presented in Table 3.

- $VF_1()$: Frame acquisition and image enhancement (histogram equalization and Multi-scale retinex on low light conditions,
- $VF_2()$: Blob calculation for a specific frame,
- $VF_3()$: Convolutional Neural Network (MobileNet v2) pass and probability matrix acquisition,
- $VF_4()$: Coordinates calculation for the detected objects and non maxima suppression for overlapping objects,
- $VF_5()$: Convolutional Neural Network (gender CNN networks) pass and probability matrix acquisition,
- $VF_6()$: Results reporting.

The model's basic parameters are presented in Table 3.

The values have been selected after performing a set of experiments for different workloads. A non-linear model for the cost function has been chosen. VFs have been implemented using Python3, utilizing the multiprocessing library.

TABLE 3
Model's parameters.

Parameter	Value
n	8
W	$98 \pm 1.7 Mbps$
cpu_{load}	$[10^3, 5 \times 10^3, 10^6, 10^2]$ instructions/frame
$output$	$[10^5, 3 \times 10^4, 10^7, 10^4]$ bytes
$c_k(l)$	$\frac{l^2+0.8}{1000}$
$r(l)$	$\frac{20l+9}{m_k}$ sec
m_1	10^4 instructions/sec (PI 3)
m_2	10^7 instructions/sec (PI 4)

4.3 LSTM models

As discussed in Section 3.3, a QoS monitoring and failure avoidance mechanism has been developed, in order to enable the hosting of demanding surveillance services throughout their life-cycle. The basic concept is based on the idea of monitoring the basic performance indicators of an edge device and trying to predict the value of these indicators within a specific time-window in the future. For this prediction, two LSTM models have been utilized, one for each performance indicator. While the possibility of using a single LSTM model has been explored, by combining RAM and CPU utilization in a singe metric, the experimental results have shown that it is more appropriate to deploy two models, one for each metric.

The challenging part of this approach is to acquire a suitable dataset for successfully training the LSTM models. As no suitable dataset came to our knowledge, the edge environment described in 4.2 has been used in order to produce the suitable datasets.

An agent hosted in the VFO device constantly collecting data regarding the CPU and RAM utilization for each device which is part of a VFC. A software which can mimic overload demand on the edge devices (stress tool) has been installed on each edge device, under a random distribution on the required resources. At the same time, VFO monitors the QoS (processed frames / sec) for each one of the deployed services (Fig. 9).

The training dataset has been created after 248 hours of monitoring and collecting data from the eight (8) devices of the edge environment, with an interval of five (5) secs. This process has resulted to a set of time series $t_{cr} = c_i, r_i$, one for each VFC service applied on the edge environment.

The models have been implemented by taking 100 neurons in the LSTM layer. The utilized loss function is Mean Squared Error. Train and test errors are presented in Fig. 10 and Fig. 11.

4.4 Comparison setups

4.4.1 *Baseline VF allocation algorithm*

The first comparison study refers to the quantification of the improvement on the VFC placement algorithm in an edge environment upon the deployment of a VFs set. To achieve this, a simple placement algorithm (*Algorithm 2*) has been considered, in order to assess the influence of the proposed placement algorithm. The rationale of the baseline algorithm is based on an initial random placement of the VFs of a VFC on the edge nodes. After the constitution

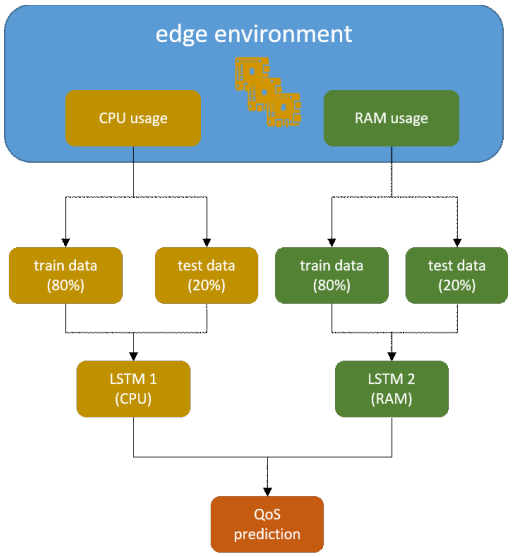

Fig. 9. LSTM dataset creation.

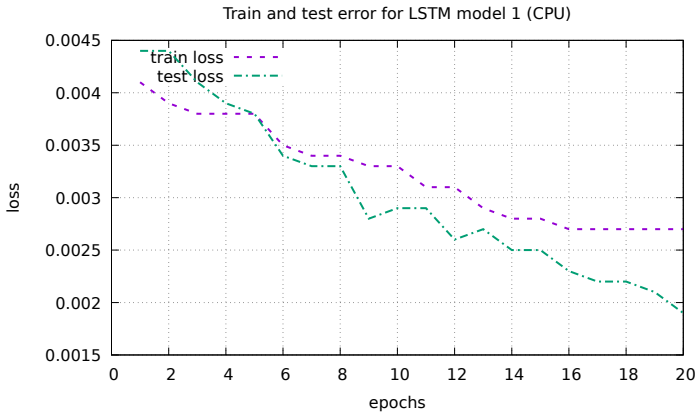

Fig. 10. Train and test error for LSTM model 1 (CPU).

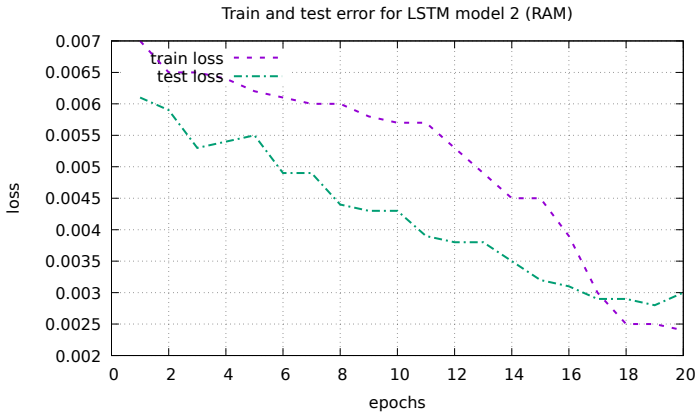

Fig. 11. Train and test error for LSTM model 2 (RAM).

of the chain, the processed fps are measured and compared with the desired QoS. In case the QoS requirements aren't met, the baseline algorithm detects the edge node with the lowest throughput and creates a replica of the relative VF, which is again randomly placed on an edge node. The last step iterates until the QoS requirements are met, or their are no more resources to commit.

Algorithm 2: Baseline placement algorithm.

Input: fps, VF_i, K_j, ϵ
x_{ij} = random-placement(VF_i, K_j);
$fps_{current} = measure_{fps}()$;
if $|fps - fps_{current}| < \epsilon$ **then**
  return x_{ij}
else
  while $(|fps - fps_{current}| > \epsilon)$ **do**
    sleep(1);
    $K_{low}, VF_h = getLowestThroughput(K_j)$
    $addReplicate(VF_h)$
    $fps_{current} = measure_{fps}()$;
  end
end

4.4.2 *Cloud service*

Aiming to compare the proposed architecture with the current common practice of deploying a surveillance service, a cloud surveillance service has been deployed. The characteristics of the utilized Virtual Machine were CPU: Intel Xeon E5 v3 @2.8GHz and RAM: 32GB. The IaaS of the Google Platform ©was chosen to host the services.

The services deployed on the Cloud are identical (*Service A* and *Service B*), in terms of implementation, with the services deployed on the edge environment.

4.4.3 $Monolithic$ *model*

The simplest approach for deploying a video surveillance service on the edge would be to host the entire service on a single edge node using a monolithic model. Despite the simplicity of this approach, several advantages can be found, like the easy deployment and the straightforward management of the service. Yet, the computational capacity of the edge nodes is expected to limit the QoS.

The specific model has been implemented and deployed to the edge environment, mainly for identifying the lowest threshold for the QoS and edge environment cost. For the implementation of the model, the same VFs have been used as the ones in the VFC model.

4.4.4 *Simple Frame Distribution model*

A second benchmark model has been implemented, also for comparison purposes. The Simple Frame Distribution model ($SFDM$) extends the $Monolithic$ model by deploying the same service on several edge nodes and by distribution the incoming frames to the nodes. This model requires an new agent which handles the packaging and distribution of the frames under a proprietary protocol and a second agent which sinks and synchronize all responses and inform the end user about the final result.

The implementation and deployment process of the $SFDM$ model is simpler than the VFC model. Yet, this approach nullifies benefits of the differential algorithms, like background subtraction [40]. Nonetheless, the $SFDM$ is considered, aiming to evaluate the processing capacity of the VFC model and its cost over the edge environment.

4.4.5 Apache Spark framework

Apache Spark© [40] is a general-purpose distribution system, which can utilize the processing capacity of a cluster to perform complex computations. There are three different ways in which Apache Spark can be used for distributing computational tasks: (i) Standalone Mode, in which Spark and HDFS (Hadoop Distributed File System) directly communicate with each other and optionally MapReduce can submit jobs in the same cluster; (ii) Hadoop Yarn according to which Spark executes over a Hadoop container manager distributed across the cluster and (iii) Spark in MapReduce, where Spark can execute its own jobs along with the one submitted by MapReduce.

For benchmark purposes, the Standalone deployment mode of the Apache Spark has been selected, in which both HDFS and Apache Spark are the part of the cluster. The master node, which acts as a server, also hosts the streaming framework of Kafka [41], which is used to collect the input of the camera and distributed efficiently on the cluster. The Spark cluster deployed on the same edge devices as the ones used to test the VFC model. It's configuration parameters have been set according to [42].

The acquirement of the data was performed by a single service which hosted the Apache Kafka © framework, outside the edge environment, as described in [43].

4.4.6 Kubernetes framework

Kubernetes [44] is an open-source container orchestration tool, which quickly after its introduction, became the de facto standard for managing large container deployments. Kubernetes support by default orchestration and autoscaling of containerized services, based on the users' demand. Aiming to evaluate the performance of the autoscaling capacity of the proposed VFC model, a Kubernetes cluster has been build using the Raspberry Pi cluster as worker nodes, based on the blueprint proposed in [45].

The testbed comprise a PC (Intel x64 architecture) serving as Kubernetes master node and the 8 Raspberry Pi devices described in Section 4.2. *Service A* has been deployed on the cluster and a user simulated the demand alterations by changing the requested fps processed. Total edge environment cost and number of deployed VFs have been considered when comparing the two approaches.

5 Experimental Setups and Results

A set of experiments have been contacted, aiming to report the performance of the proposed VFC model. One can categorize the experimental work in three parts:

- Experiments set out to assess the scalability and sustainability of the VFC model.
- Experiments set out to assess the contribution and the benefits of the two add-on services on the VFC model (caching and QoS mechanisms).
- Experiments set out to compare the VFC model with alternative frameworks which can host surveillance services.

The tools described in Section 4 have been used in order to evaluate the VFC model under specific metrics. Namely, the QoS of the deployed services, the total cost of the edge environment and the scaling capacity of the VFC model have been considered. A set of variants (Setups) have been used, aiming to examine the aforementioned metrics. More specifically, (as briefly mentioned in Section 4.1) the experimental setups described in Tables 4, 5, 6 and 7 have been implemented on the edge environment detailed in 4.2.

Edge network data collection tools

For implementing the proposed model on a real case scenario, it is necessary to calculate the required data and calculate the metrics presented in Table 1. More specifically, the parameters of the cost function, along with the bandwidth of the links, need to be probed. To achieve this, the following tools have been utilized, under a *Linux/Ubuntu 20.04* environment.

- **Cost function.** For collecting the cost function's parameters (i.e. CPU and RAM utilization), the ***htop*** *(manpages.ubuntu.com/manpages/focal/en/man1/htop.1.html)* library has been utilized.
- **Links bandwidth and utilization**. For supporting the VFC model, the bandwidth among two edge nodes, as well as the utilization of a network link, are required. A star network architecture was selected for the experiments, according to which each edge node is connected to a router R (TP-Link©WiFi Router AX 1500 Archer). The link between two nodes n_i, n_j is actually comprise two links ($n_i \leftrightarrow R$ and $R \leftrightarrow n_j$). For each connection, the QoS option of the router R has been utilized, reassuring that each node n_i will have at least w_i bandwidth (upstream and downstream). As each router supports up to e edge devices, $w_i = \frac{W_R}{e}$, where W_R is the router's bandwidth capacity. For probing the actual bandwidth between two nodes, the ***iperf*** *(manpages.ubuntu.com/manpages/xenial/man1/iperf.1.html)* library has been utilized. The measurements are performed for all possible pairs n_i and n_j which are connected to the same router, and the links are considered symmetrical. Thus, the required number of measurements are $num_{link-measurements} = \frac{e(e-1)}{2}$. For probing the real time throughput of a link, the ***iftop*** *(manpages.ubuntu.com/manpages/focal/man8/iftop.8.html)* library has been utilized. Finally, router to router link delays are not considered in this model, as they are assumed wired links (edge backbone network) with a much greater bandwidth compared to the other communication links and on dedicated network interfaces. For the experiments reported in the following sections, the links were wireless, thus $W_R = 1201Mbps$ and $e = 8$ edge nodes.

A python script on each edge node collects the output of *htop* and *iftop*, process the required fields and informs the

VFO about the measurements every t secs. Time interval t depends on how dynamic the edge environment is. For a highly dynamic environment, smaller values of t are required. For the reported results in the following sections, $t = 60$ secs.

5.1 Assessing the scalability of the VFC model

The first metric under consideration is the scalability of the proposed model under parameters like edge devices number and user services demand. To examine these parameters, the simulation platform detailed in Section 4.1 is utilized.

Fig. 12 and Fig. 13 present the experimental results of the simulation environment for $Monolithic$ model, SFD model and for the VFC model. All models refer to single service implementations. More specifically, Fig. 13 presents the number of required edge devices in order to support a specific number of services, while Fig. 12 presents the total cost of the edge environment as the number of requested services increase. The characteristics of the simulated services are given in Table 2. It is obvious that the VFC model enables the support of a specific number of services with fewer edge devices and with substantially lower total cost, as the service demand scales.

An additional aspect to be considered is the affect of the edge devices computational capacity to the benefits of the proposed VFC model. More specifically, the experiments reported in Fig. 12 and Fig. 13 refer to edge devices with low computational capacity (cpu_{load} is greater than m_k). As edge devices are becoming more and more powerful, it is interesting to assess then performance of the VFC under a scenario according to which m_k is greater than cpu_{load}. For this, a set of simulations have been performed, again with the parameters of 2, but with $m_k = 10^7\mathcal{N}(20, 0.9)$. The relative results are presented in Fig. 14 and Fig. 15. As one can notice, *Monolithic* and *SFD* model outperform VFC model for a small number of services. Yet, as the number of services increase, VFC model presents better metrics, in terms of both edge environment cost and required number of edge devices. The reason for this behavior is the distribution of the required processing power over the edge network, allowing the VFC model to better explore the available resources, as the different VFs can be hosted more efficiently by the edge devices, when compared to both *Monolithic* and *SFD* models.

Next, the scaling of the VFC model in a temporal simulation scenario is assessed. For this, simulations for the $Monolithic$ model and the VFC model have been implemented under a 24 hour scenario, according to which users request surveillance services under different probability distributions (Fig. 8). Five parameters have been examined. Namely, the percentage of served services over total requested services (Fig. 18), the percentage of rejected services (due to lack of resources) over total requested services (Fig. 17), the total edge environment utilization (Fig. 16), the total edge environment cost (Fig. 19) and the edge network traffic (Fig. 20) have been calculated, as the number of the edge devices scales up. It is important to mention that low demand scenarios appear to produce more traffic and higher environment cost due to the fact that fewer services are rejected during these setups.

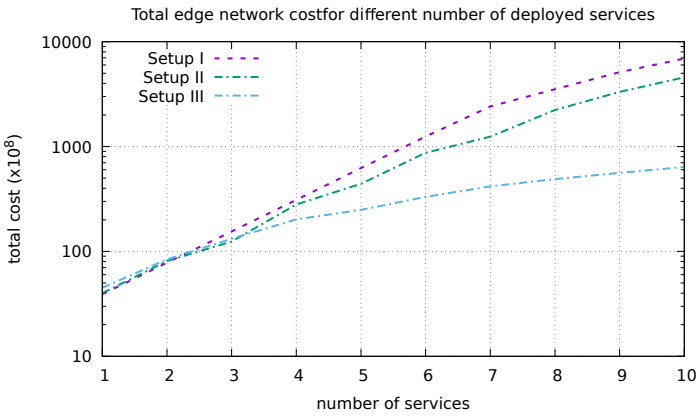

Fig. 12. Total network cost (simulation environment).

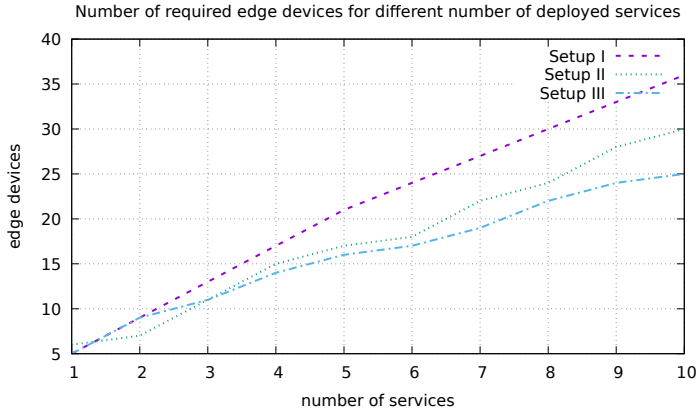

Fig. 13. Total edge devices (simulation environment).

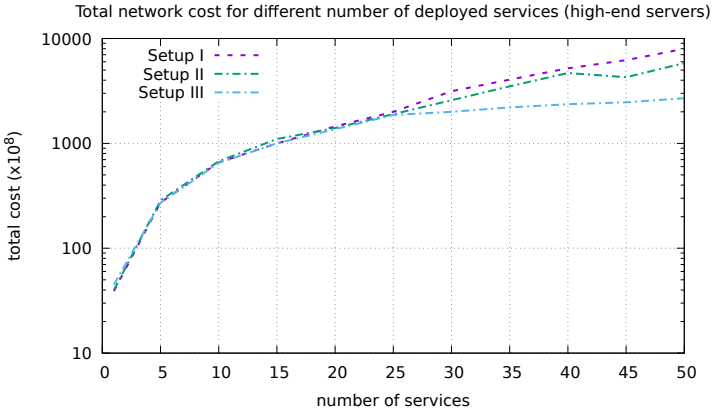

Fig. 14. Total network cost (simulation environment - high-end devices).

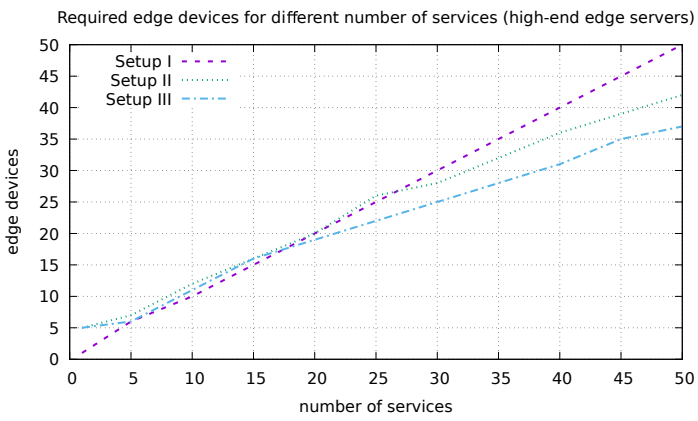

Fig. 15. Total edge devices (simulation environment - high-end devices).

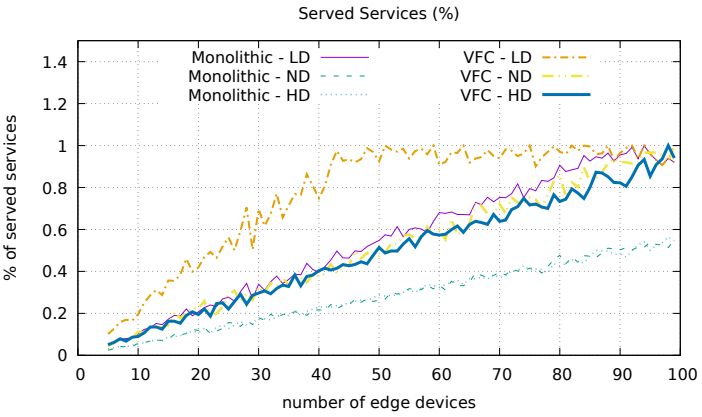

Fig. 18. Percentage of served services on different user demands (LD - Low Demand, ND - Normal Demand, HD - High Demand) and number of edge devices.

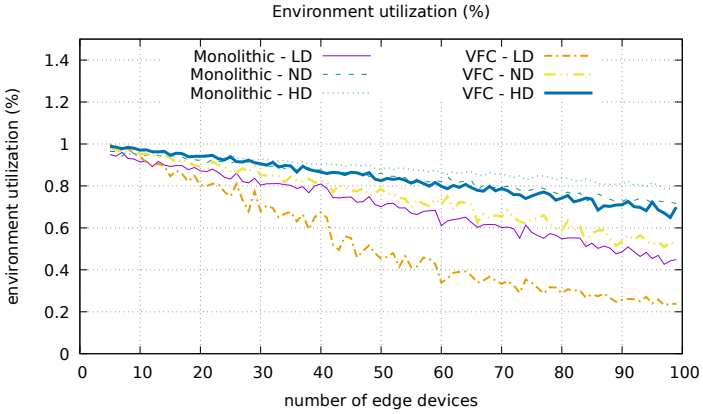

Fig. 16. Edge environment utilization on different user demands (LD - Low Demand, ND - Normal Demand, HD - High Demand) and number of edge devices.

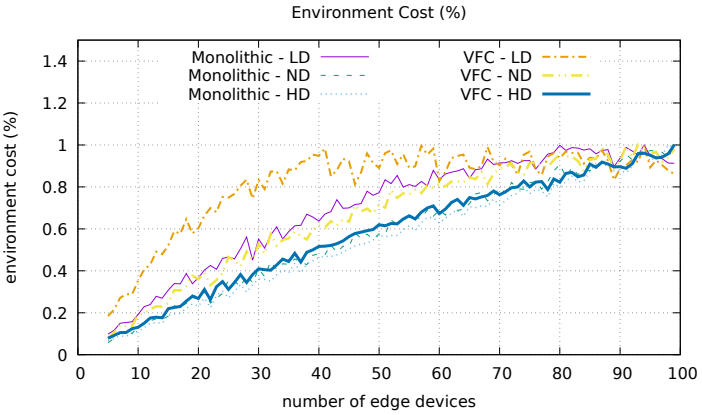

Fig. 19. Edge environment cost on different user demands (LD - Low Demand, ND - Normal Demand, HD - High Demand) and number of edge devices.

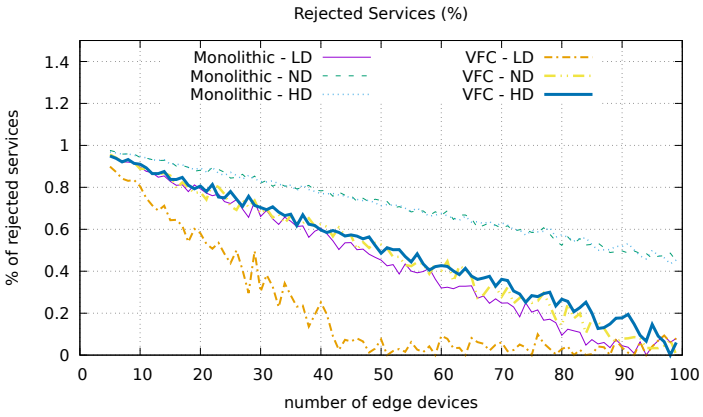

Fig. 17. Percentage of rejected services on different user demands (LD - Low Demand, ND - Normal Demand, HD - High Demand) and number of edge devices.

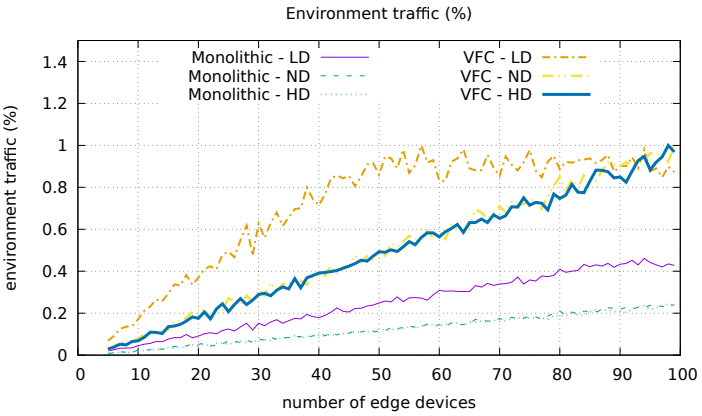

Fig. 20. Edge environment network traffic on different user demands (LD - Low Demand, ND - Normal Demand, HD - High Demand) and number of edge devices

Combining the results of aforementioned figures, one can notice that the VFC model can achieve higher percentages of served services, under all three of the different user demand distribution probabilities, while maintaining lower levels of total environment cost. Additionally, VFC model can achieve higher edge devices utilization while reducing the percentage of the reject services must faster as the network scales, always compared with the $Monolithic$ model.

On top of the aforementioned studies regarding the scalability of the proposed VFC model, the required time for deploying a service on an edge network with n devices is analyzed. Without loss of generality, lets assume an edge network with wireless devices, based on WiFi 802.11g connections (S Mbps actual bandwidth). Additionally, let j be the number of VFs to be deployed using c GB containers each. The deployment of a service following the VFC model comprises four steps:

- **Initial probe of edge devices available resources.** If a kb is the size of the probing message, then it would require approximately $\frac{a\times 8}{S\times 10^3}$ secs for the j devices to transmit the data.
- **Placement problem solving.** Based on [30], APOPT solver requires polynomial time to solve a mixed-integer problem with one non-linear equation.
- VFs **deployment.** For each one of the chosen edge devices, it would require $f \times \frac{c\times 8\times 10^3}{S}$, where f is a coefficient which expresses the delay which will be caused by reaching the limit of the output bandwidth of the VFO node.
- **Monitoring phase.** Every $\mathcal{T}$ secs, the j selected edge devices, hosting a VF each, informing the VFO about their available resources, enabling the QoS monitoring service to function. This phase requires $j\times a$ kb of data to be transferred in the network every 30 secs, with each transmission to require $\frac{a\times 8}{S\times 10^3}$ secs.

Based on the described network times, as well as the times acquired for solving the placement problem (using an Intel i7 2.8GHz (8core) on 8GB of RAM), the results presented in Fig. 21 has been obtained. The values of the variables were: $j = n/2$, $c = 1.2GB$, $a = 2kb$, $S = 100Mbps$, $\mathcal{T} = 30sec$. The interval $\mathcal{T}$ depends on how stable the edge environment is (higher values will correspond to more stable environments.)

One can notice that even when using 1000 edge devices and $500VFs$, in order to deploy a service, the required time to select the optimal nodes is kept relatively small, enabling the efficient scaling up of the model.

5.2 Assessing the add-on services on the VFC model

As described in Section 4.2, a prototype edge environment with eight low capacity edge devices has been implemented. Using this environment, a set of experiments (scenarios) have been implemented, as presented in Table 4 for a single service and in Table 5 for multiple services.

In order to support *Service A*, VFO deployed 1 instance of $VF1$, 2 of $VF2$, 4 of $VF3$ and 1 instance of $VF4$. For *Service B*, the resulted instances were 1, 3, 3 and 1 for $VF1$, $VF2$, $VF5$ and $VF6$ respectively. The calculated $VFCs$ are in accordance with the VF characteristics, as the most demanding VFs ($VF3$ and $VF5$) participated in the chains with the largest number of instances.

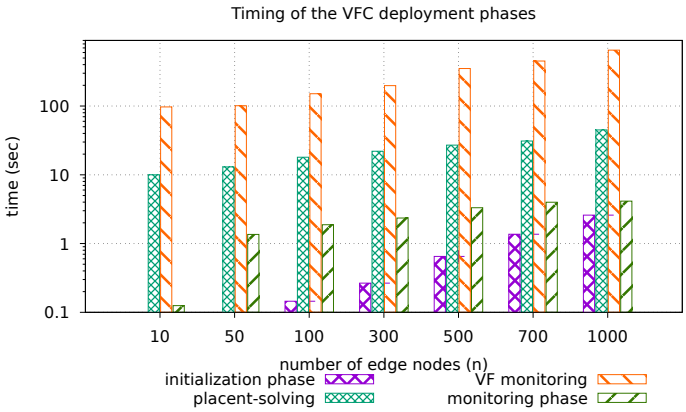

Fig. 21. Timing of the different VFC phases.

TABLE 4
Evaluation of VFC services - single service (*Service A* (gender classification)).

Scenario	Description
S11	VFC model.
S12	VFC model + QoS mechanism.

5.2.1 QoS monitoring mechanism

This set of experiments aim at assessing the performance of the QoS mechanism based on the learning service build on the LSTM models described in Section 3.3. For this, scenarios described in Tables 4 and 5 have been utilized. Each scenario has been implemented in the edge benchmark environment for a real case scenario of 12 hours of continuous execution of the surveillance services.

The results of this set of experiments are reported in Fig. 22 and in Fig. 23. QoS monitoring mechanism, which is supported by the two LSTM models described in Section 4.3, plays a crucial role in the stabilization of the QoS of the executed services. This applies to both single-service and multi-service scenarios, as presented in both figures. Applying the QoS mechanism decreases the variation of the processed frames by more than 59.24%, resulting to a more stable surveillance service.

5.2.2 Caching mechanism

As described in Section 3.4, the VFC model enables caching the processed data and avoiding recalculations when two of more services require the processing of the same frames. A set of experiments has been conducted, trying to reveal the benefits of the VFC approach. *Services A* and *B* can share $VF1()$ and $VF2()$.

Fig. 23 presents the results on the evaluation of the caching mechanism. The different scenarios were configured in order to support the QoS constraints of *Services A* ($fps = 12$) and *B* ($fps = 10$). Caching mechanism reduces the environment cost by 32.3% from the VFC, while improving the the QoS by 2.93% and the edge environment cost by $9.8\%1$ compared to scenario S21.

TABLE 5
Evaluation of VFC services - multiple services (*Services A* (gender classification) and *B* (age classification) deployed on $VFCs$).

Scenario	Description
S21	VFC model.
S22	VFC model + cashing mechanism.
S23	VFC model + cashing + QoS mechanism.

Caching mechanism reduces the total environment cost by 30% while maintaining the QoS of the services.

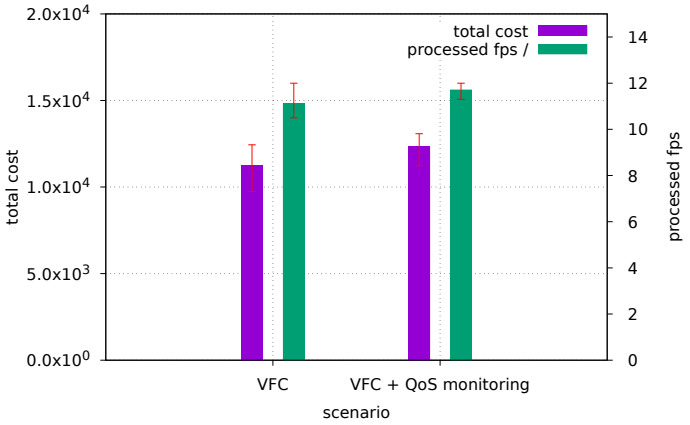

Fig. 22. Total cost and processed fps for (a) VFC model, (b) VFC model with QoS model enabled. Single service mode.

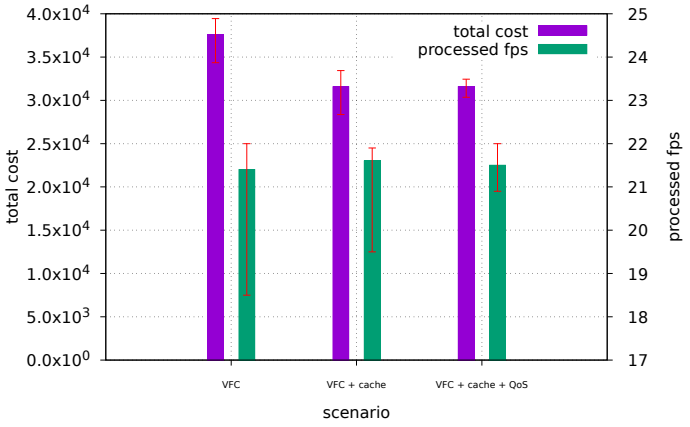

Fig. 23. Total cost and processed fps for (a) VFC model, (b) VFC model with caching enabled and (c) VFC model with caching and QoS model enabled. Multiservice mode.

5.3 Assessing the performance of the VFC model

This set of scenarios aim to provide evidence about the performance of the VFC model against alternative approaches.

More specifically, section 5.3.1 presents the quantification of the improvement VFC placement algorithm, when compared with the baseline placement algorithm. Section 5.3.2 describes the results on the comparison of the VFC model with a cloud-based surveillance service while section 5.3.3 includes the relative results on the comparison of the model with other distribution schemes.

5.3.1 *Baseline placement algorithm compared to VFC model*

The first set of experiments concern the evaluation of the VFC placement algorithm. For this, the simulation environment has been utilized, aiming to compare the two approaches as the edge environment scales up.

For each simulation scenario, a specific number of $VFCs$ has been considered and the simulation was executed 10 times. For each VFC, the number of VFs was randomly selected from the distribution $\lfloor \mathcal{N}(3, 8) \rfloor$. The relevant results (average values for the 10 fold executions) are presented in Fig. 24 and Fig. 25.

More specifically, one can observe that the number of the required VFs (including replicas) are reduced at approximately 95.32% (average value) when using the VFC placement algorithm, when compared with the baseline placement algorithm. Additionally, the total edge environment cost is reduced by 68.22% with the use of the VFC placement model.

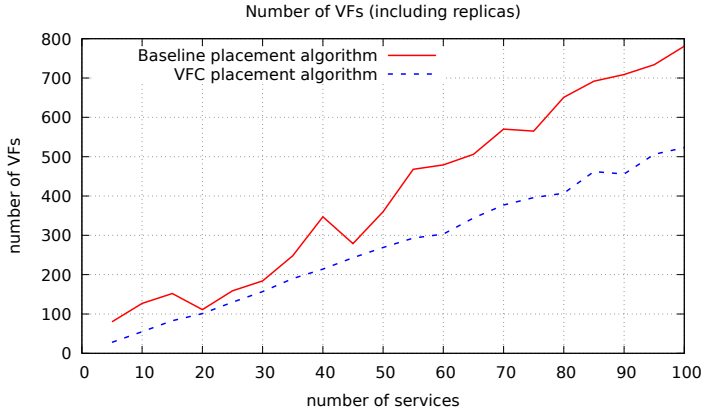

Fig. 24. Baseline vs. VFC placement algorithms - number of VFs.

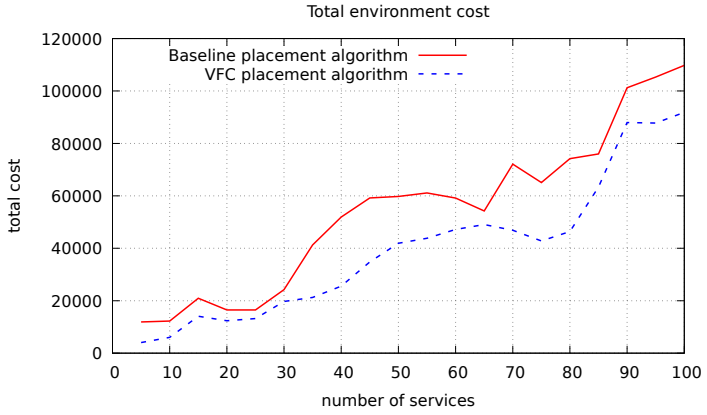

Fig. 25. Baseline vs. VFC placement algorithms - total environment cost.

5.3.2 *Cloud infrastructure compared to VFC model*

The next set of experiments aimed at evaluating the performance of the proposed VFC model against a surveillance service deployed on a cloud infrastructure. For this, *Service*

A has been deployed to the cloud environment described in Section 4.4.2 and specific performance metrics have been compared against the VFC model deployed on the edge environment detailed in Section 4.2.

The first parameter under consideration is the performance of the cloud environment under different network communication channels and on different video resolutions. These experiments, which results are reported in Fig. 27, aim to assess the importance of different broadband communication technologies to the real time QoS (processed fps) of the deployed service.

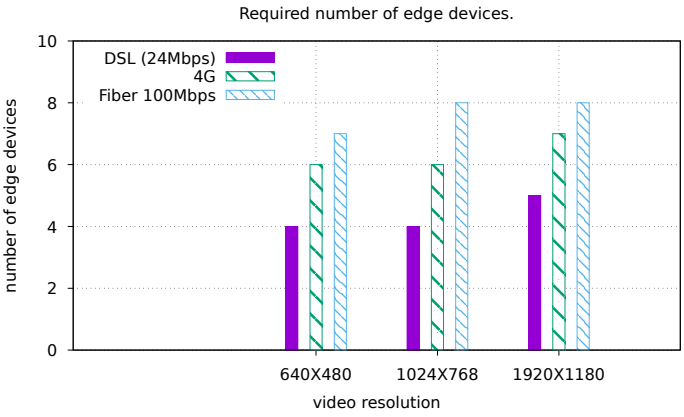

Fig. 26. Required number of edge devices

It is obvious that the network link between the surveillance camera and the cloud infrastructure plays an important role to the QoS of the service. Next, the second parameter under consideration are the number of edge devices required to meet the same QoS as the one observed on the cloud infrastructure. This parameter has been calculated for the different video resolutions. The results, which are presented in Fig. 26, indicates that with a relative small number of edge devices, VFC model can meet the performance of a cloud service.

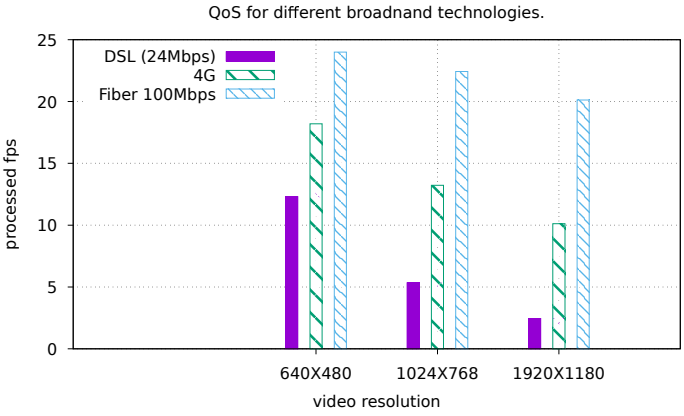

Fig. 27. QoS for different broadband technologies

5.3.3 Comparison with similar distribution frameworks

The experiments for this purpose are summarized in Tables 6 and 7. More specifically, the VFC model has been tested against $Monolithic$ and $SFDM$ models, as well as with the Apache Spark © both on single service and on mutli-service scenarios. The relative results are presented in Fig. 28 and in Fig. 29 respectively. Both for the single service and the multi-service scenarios, VFC model achieved higher QoS compared with the $Monolithic$ approach (+120.5% for the single service and +133.3% for the multi-service scenario) and with the $SFDM$ model (+22.5% for the single service and +10.8% for the multi-service scenario). At the same time, the VFC model reduced the operational cost by 43.5% on average compared with the aforementioned models.

As far as the comparison with the Apache Spark © framework is concerned, the VFC model has achieved on average the same QoS by reducing the operational cost of the edge environment by approximately 103.3% on the single service scenario and by 90.7% on the multi-service scenario. Additionally, while the average value of the achieved QoS by the VFC model is slightly improved with the one produced by the Apache Spark ©, the variation of the QoS throughout the service is reduced by almost 50.3% in both single service and multi-service scenarios.

TABLE 6
Comparison of VFC model - single service (*Service A* (gender classification) against other frameworks).

Scenario	Description
S31	Monolithic model.
S32	SFD model.
S33	Spark framework.
S34	VFC model + QoS mechanism.

TABLE 7
Comparison of VFC model - multiple services (*Services A* (gender classification) and *B* (age classification) against other frameworks).

Scenario	Description
S31	Monolithic model.
S32	SFD model.
S33	Spark framework.
S34	VFC model + cashing + QoS mechanism.

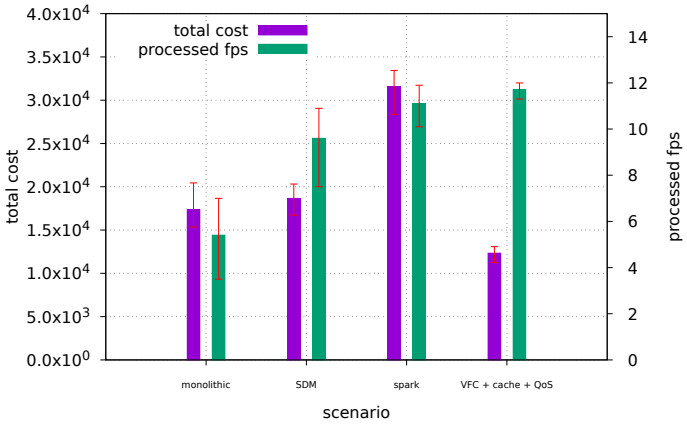

Fig. 28. Total cost and processed fps for (a) Monolithic model, (b) SDM, (c) Apache Spark and (d) VFC model with with caching and QoS model enabled. Single service mode.

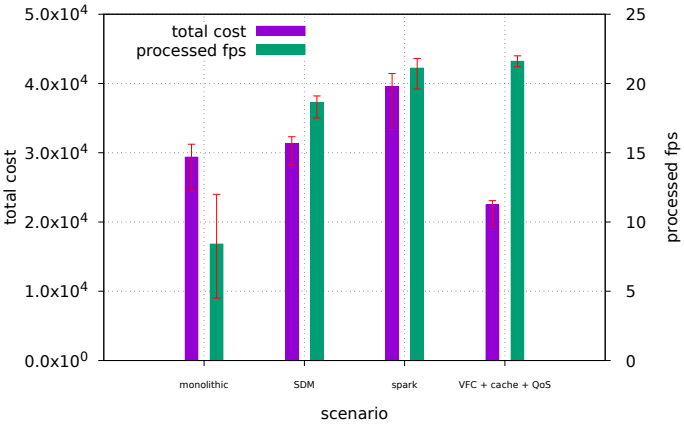

Fig. 29. Total cost and processed fps for (a) Monolithic model, (b) SDM, (c) Apache Spark and (d) VFC model with with caching and QoS model enabled. Multi-service mode.

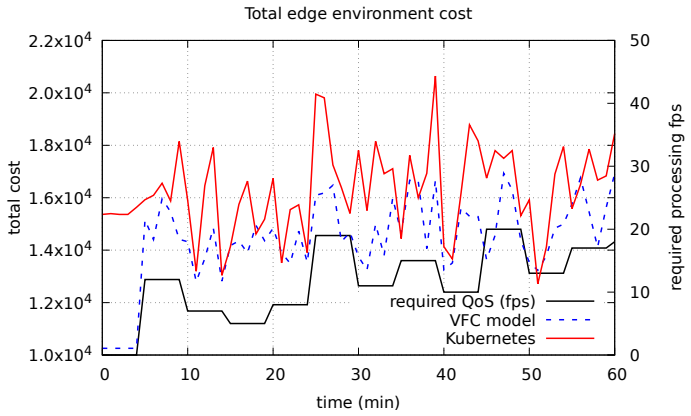

Fig. 31. VFC vs. Kubernetes - Total environment cost.

5.3.4 VFC Autoscaling capacity evaluation

A set of experiments for assessing the VFC model's capacity to autoscale the deployment of the VFs depending on the users' QoS demand has been conducted. According to the experimental scenario, *Service A* has been deployed on both VFC and Kubernetes frameworks, as described in Section 4.4.6. The simulation has run for 60 minutes on each framework. Within the simulation time, the required QoS (requested fps processed) was randomly changed every 5 minutes, within the range $[5, 20]$. During the first 5 minutes, the requested *fps* was set to 0, aiming to assess the zero-demand footprint of the solutions under comparison. The total edge environment cost, as well as total number of deployed containerized VFs have been probed and the relative results are presented in Figures 30 and 31.

From the reported results we can observe that the VFC model can follow the demand changes more efficiently compared with Kubernetes, even if the improvement is small. As far as the total edge network cost is concerned, VFC presented a reduction of 12.54% compared to Kubernetes, averaging the cost over the 60 minutes experiment.

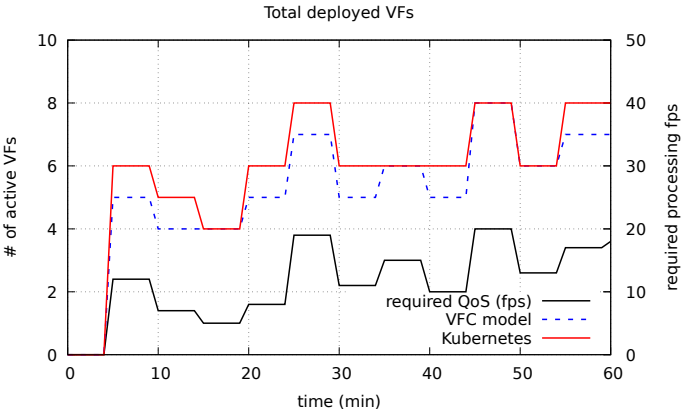

Fig. 30. VFC model vs. Kubernetes - deployed VFs.

6 Discussion

Enabling edge computing to support heavy load computational tasks is crucial for integrating IoT environments with machine learning and artificial intelligence services. Towards this direction, a novel approach is proposed, which introduces the decoupling of the services to independent micro-services, all integrated under a VFC model.

The proposed VFC model is introduced by incorporating a decoupling scheme on the main VF allocation process. It is shown that this particular NP hard problem can be solved in a viable time-frame, even for edge computing low level devices.

Aiming to assess the scalability and the performance of the proposed VFC model, a comparative study has been performed, both on a simulator and on a real-case benchmark edge environment. As far as the scalability and expandability of the model is concerned, the simulation results have revealed that the VFC model can be deployed on a high number of edge devices, maintaining each advantages as far as the total cost and the edge devices utilization are concerned. Additionally, the proportions of the served services and the rejected services over the total requested services under different user demand rates show that the VFC model can operate effectively on large-scale scenarios.

As far as the comparison of the VFC model with alternative distribution approaches is concerned, a set of experiments has taken place. The experiments can be categorized into three main categories. Namely, the first category involve the comparison of the VFC model with baseline approaches, like the $Monolithic$ service approach and the $SFDM$ model. The scope of these experiments is to set a performance borderline, aiming to assess the improvement level of the VFC model. The relevant results have shown a substantial improvement over the aforementioned simplistic approaches.

The other two categories of experiments involve more pragmatic alternative approaches. Namely, a comparison study with a surveillance service deployed on a Cloud infrastructure has been held. The results have shown that with a relative small number of edge devices, the VFC model can have the same performance metrics as the Cloud service, under different technologies of broadband connections. Finally, a comparison study has been performed with Spark,

a generic distribution framework. Both due to the extensive footprint of Spark on the low-level edge devices and due to the achieved QoS, the VFC model has outperformed Spark in all performed scenarios, especially on stabilization of the achieved QoS.

On top of the VFC model, two add-on services have been designed, developed and deployed, aiming to boost its performance. More specifically, a caching mechanism has been introduced, reducing the operational cost of the edge environment on multi service usage scenarios. Finally, QoS monitoring service based on a deep learning framework attempts to predict possible VFs failures and inform the VFO to take the appropriate actions.

The aforementioned results support the conclusion that the proposed VFC model can act as an efficient and scalable solution for supporting *heavy* streaming applications on the edge. It's comparison with both cloud solutions and generalized distribution frameworks, like Apache Spark and Kubernetes reveals that VFC can be considered a new, novel approach for edge-deployment architectures. While several alternative frameworks have been proposed, VFC bundles all the key features such frameworks should explore, like optimization of VF placement, auto-scaling and QoS monitoring.

7 Conclusions and Future Work

Edge computing is expected to be an important part of the AI industry during the next few years. Its advantages lie not only on the proximity with the processing data, but also on the data protection and safety issues, which are debatable on the Cloud computing paradigm. This paper proposes a novel concept for enabling real-time AI applications on an Edge network. Our proposal is based on the $VFCs$ which are used to distribute an AI application across the edge network on an scalable fashion. After providing a mathematical model for our system, we report the results of a real-case scenario, where the system has been implemented and tested in various conditions. A caching mechanism is also described, which extents even further the capacity of our system. The experiments have provided evidence that such this approach can be used to undertake heavy-load AI applications and handle them in real-time. Regarding the next step of our work, we plan to extent our model in order to be able to handle node failures, by adding a migration mechanism to our architecture.

The proposed model has been applied on a video analytic service, which belong in the family of streaming applications, in terms of data generation. The nature of the streaming applications matches the characteristics of the proposed VFC model, which partially explains the really good performance of the model against other distribution approaches. Thus, while we anticipate that our model would be outperformed by generalized distribution schemes (i.e. Apache Spark ©) on non-streaming applications, streaming applications from other domains, like sentiment analysis on data coming from social media are expected to have similar performance benefits as the tested surveillance service. Deploying such applications on the VFC model is part of our future plans.

References

[1] Jain et al., "Performance characterization of dnn training using tensorflow and pytorch on modern clusters," in *Proc. of the 2019 CLUSTER*. 2019, IEEE.

[2] He, D. et al., "A Survey to Predict the Trend of AI-able Server Evolution in the Cloud," *Special Section on Emerging Trends, Issues and Challenges in energy - efficient Cloud Computing*, vol. 6, Feb. 2018.

[3] Sun et al., "Vu: Edge computing-enabled video usefulness detection and its application in large-scale video surveillance systems," *IEEE Internet of Things Journal*, 2019.

[4] Alnoman A. et al., "Emerging edge computing technologies for distributed iot systems," *IEEE Network*, vol. 16, no. 6, pp. 140–147, 2019.

[5] Carvalho G. et al., "Edge computing: current trends, research challenges and future directions," *Computing*, 2021.

[6] D Aishwarya and RI Minu, "Edge computing based surveillance framework for real time activity recognition," *ICT Express*, vol. 7, no. 2, pp. 182–186, 2021.

[7] Vassilis Tsakanikas and Tasos Dagiuklas, "Enabling real-time ai edge video analytics," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

[8] Rachad Atat, Lingjia Liu, Jinsong Wu, Guangyu Li, Chunxuan Ye, and Yi Yang, "Big data meet cyber-physical systems: A panoramic survey," *IEEE Access*, vol. 6, pp. 73603–73636, 2018.

[9] Rob Kitchin, "The real-time city? Big data and smart urbanism.," *GeoJournal*, vol. 79, no. 1, pp. 1–14, Feb. 2014.

[10] W. Zhou et al., "A System Architecture to Aggregate Video Surveillance Data in Smart Cities," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–7.

[11] Li, Jianhua et al., "Virtual Fog: A Virtualization Enabled Fog Computing Framework for Internet of Things," *IEEE Internet of Things Journal*, vol. 5, no. 1, Feb. 2018.

[12] Chen et al., "Dynamic Urban Surveillance Video Stream Processing Using Fog Computing," in *2016 IEEE Second International Conference on Multimedia Big Data (BigMM)*, 2016, pp. 105–112.

[13] Dautov et al., "Metropolitan intelligent surveillance systems for urban areas by harnessing IoT and edge computing paradigms," *Softw Pract Expe.*, vol. 48, no. 1, pp. 1475–1492, 2018.

[14] Sun Mao, Shunfan He, and Jinsong Wu, "Joint uav position optimization and resource scheduling in space-air-ground integrated networks with mixed cloud-edge computing," *IEEE Systems Journal*, vol. 15, no. 3, pp. 3992–4002, 2020.

[15] Songlin Chen, Hong Wen, Jinsong Wu, Wenxin Lei, Wenjing Hou, Wenjie Liu, Aidong Xu, and Yixin Jiang, "Internet of things based smart grids supported by intelligent edge computing," *IEEE access*, vol. 7, pp. 74089–74102, 2019.

[16] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2019.

[17] Ning Chen and Yu Chen, "Smart City Surveillance at the Network Edge in the Era of IoT: Opportunities and Challenges," in *Smart Cities*, pp. 153–176. Apr. 2018.

[18] Hwejoo et al., "A data streaming performance evaluation using resource constrained edge device," in *2017 ICTC*, Jeju, South Korea, 2017, IEEE.

[19] S. Khan, "A review on the application of deep learning in system health management," *Mechanical Systems and Signal Processing*, vol. 107, pp. 241–265, 2018.

[20] M. Ranzato, "Video (language) modeling: A baseline for generative models of natural videos," *arXiv:1412.6604*, 2014.

[21] W. Rawat, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Computation*, vol. 29, no. 9, pp. 1–10, 2017.

[22] Sharma P. et al., "Era of deep neural networks: A review," in *8th International Conference on Computing, Communication and Networking Technologies*, 2017, pp. 1–5.

[23] Gallagher J. et al., "A reconfigurable continuous time recurrent neural network for evolvable hardware applications," in *2005 IEEE Congress on Evolutionary Computation*, 2005.

[24] Neil D. et al., "Phased lstm: Accelerating recurrent network training for long or event-based sequences," *Advances in neural information processing systems*, vol. 16, pp. 3882–3890, 2016.

[25] J. Halpern and C. Pignataro, "RFC 7665 - Service Function Chaining (SFC) Architecture," Oct. 2015.

[26] C. Besse and B. Chaib-draa, "An Efficient Model for Dynamic and Constrained Resource Allocation Problems," in *2nd COPLAS '07*, 2007.

[27] Ahuja, Ravindra et al., *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, 1993.
[28] Laborie, P. et al., "IBM ILOG CP optimizer for scheduling," *Constraints*, vol. 23, no. 2, pp. 210–250, 2018.
[29] Aderaldo et al., "Kubow: an architecture-based self-adaptation service for cloud native applications," in *Proc. of the 13th ECSA*, Paris, France, 2019, vol. 2, pp. 42–45, ACM.
[30] Hedengren et al., "Apopt: Minlp solver for differential and algebraic systems with benchmark testing," in *Proceedings of INFORMS National Meeting*, 2012.
[31] Amir HajiRassouliha, Andrew J. Taberner, Martyn P. Nash, and Poul M.F. Nielsen, "Suitability of recent hardware accelerators (dsps, fpgas, and gpus) for computer vision and image processing algorithms," *Signal Processing: Image Communication*, vol. 68, pp. 101–119, 2018.
[32] Yuexian Zou, Guangyi Shi, Yufeng Jin, and Yali Zheng, "Extraocular image processing for retinal prosthesis based on dsp," in *2009 4th IEEE International Conference on Nano/Micro Engineered and Molecular Systems*, 2009, pp. 563–566.
[33] Beal et al., "Gekko optimization suite," *Processes*, vol. 6, no. 8, 2018.
[34] Cho, K. et al., "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, , no. 1, 2014.
[35] M. Shamim Hossain, "Qos-aware service composition for video surveillance," in *2011 IEEE International Conference on Multimedia and Expo*, 2011, pp. 1–5.
[36] Biao Song, Yuan Tian, and Bingyin Zhou, "Design and evaluation of remote video surveillance system on private cloud," in *2014 International Symposium on Biometrics and Security Technologies (ISBAST)*, 2014, pp. 256–262.
[37] C. P. Nwokolo and H. C. Inyiama, "Quality of service evaluation in on-demand cloud-based video surveillance," in *2017 IEEE 3rd International Conference on Electro-Technology for National Development (NIGERCON)*, 2017, pp. 532–537.
[38] Sangmin et al., "A Large-scale Benchmark Dataset for Event Recognition in Surveillance Video," in *Proceedings of IEEE Comptuer Vision and Pattern Recognition (CVPR), 2011*. 2011, IEEE.
[39] Google AI Blog, "MobileNetV2: The Next Generation of On-Device Computer Vision Networks," 2018.
[40] M. Piccardi, "Background subtraction techniques: a review," in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, 2004, pp. 3099–3104.
[41] Khochare et al., "Distributed video analytics across edge and cloud using echo," in *International Conference on Service-Oriented Computing (ICSOC) Demo*, 2017.
[42] Nasir et al., "Fog computing enabled cost-effective distributed summarization of surveillance videos for smart cities," *Journal of Parallel and Distributed Computing*, vol. 126, pp. 161–170, 2019.
[43] Ichinose P. et al., "A study of a video analysis framework using kafka and spark streaming," in *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 2396–2401.
[44] Brendan Burns, Joe Beda, Kelsey Hightower, and Lachlan Evenson, *Kubernetes: up and running*, " O'Reilly Media, Inc.", 2022.
[45] Paridhika Kayal, "Kubernetes: Towards deployment of distributed iot applications in fog computing," in *Companion of the ACM/SPEC International Conference on Performance Engineering*, 2020, pp. 32–33.

Vassilios Tsakanikas received his Diploma Degree in Electrical and Computer Engineering from the National Technical University of Athens in 2005 and his M.Sc. in Computer Science from Athens University of Economics and Business in 2007. His research interests are signal processing, computer vision, artificial intelligence and edge computing. Vassilios is a Ph.D. student at the London South Bank University and member of the SuITE research group, a member of the Technical Chamber of Greece and a Student IEEE member.

Tasos Dagiuklas received the Engineering Degree from the University of Patras-Greece in 1989, the M.Sc. from the University of Manchester, U.K., in 1991, and the Ph.D. degree from the University of Essex-U.K. in 1995, all in Electrical Engineering. He is a leading researcher and expert in the fields of Internet and multimedia technologies for smart cities, ambient assisted living, healthcare, and smart agriculture. He has been a principle investigator, a co-investigator, a project and technical manager, a coordinator, and a focal person of over 20 internationally R&D and capacity training projects with total funding of approximately £.5.0m from different international organizations. He is currently the Leader of the SuITE Research Group, London South Bank University, where he also acts as the Head of the Division in Computer Science. His research interests include smart internet technologies, media optimization across heterogeneous networks, QoE, virtual reality, augmented reality, and cloud infrastructures and services.