Cloud Versus Edge Deployment Strategies of Real-Time Face Recognition Inference

Anis Koubaa, Adel Ammar, Anas Kanhouch, and Yasser AlHabashi

***Abstract*—Choosing the appropriate deployment strategy for any Deep Learning (DL) project in a production environment has always been the most challenging problem for industrial practitioners. There are several conflicting constraints and controversial approaches when it comes to deployment. Among these problems, the deployment on cloud versus the deployment on edge represents a common dilemma. In a nutshell, each approach provides benefits where the other would have limitations. This paper presents a real-world case study on deploying a face recognition application using MTCNN detector and FaceNet recognizer. We report the challenges faced to decide on the best deployment strategy. We propose three inference architectures for the deployment, including cloud-based, edge-based, and hybrid. Furthermore, we evaluate the performance of face recognition inference on different cloud-based and edge-based GPU platforms. We consider different models of Jetson boards for the edge (Nano, TX2, Xavier NX, Xavier AGX) and various GPUs for the cloud (GTX 1080, RTX 2080Ti, RTX 2070, and RTX 8000). We also investigate the effect of deep learning model optimization using TensorRT and TFLite compared to a standard Tensorflow GPU model, and the effect of input resolution. We provide a benchmarking study for all these devices in terms of frames per second, execution times, energy and memory usages. After conducting a total of 294 experiments, the results demonstrate that the TensorRT optimization provides the fastest execution on all cloud and edge devices, at the expense of significantly larger energy consumption (up to +40% and +35% for edge and cloud devices, respectively, compared to Tensorflow). Whereas TFLite is the most efficient framework in terms of memory and power consumption, while providing significantly less (-4% to -62%) processing acceleration than TensorRT. *Practitioners Note:* The study reported in this paper presents the real-challenges that we faced during our development and deployment of a face-recognition application both on the edge and on the cloud, and the solutions we have developed to solve these problems. The code, results, and interactive analytic dashboards of this paper will be put public upon publication.**

***Index Terms*—Cloud inference, computation offloading, edge inference, face recognition, FaceNet, jetson boards, production environment.**

Manuscript received October 30, 2020; revised December 20, 2020; accepted January 20, 2021. Date of publication February 8, 2021; date of current version January 11, 2022. This research was funded by Prince Sultan University, with funding reference SEED-2020-05. The authors would like to thank Prince Sultan University for supporting this research under the grant number SEED-CCIS-2020-05. Recommended for acceptance by Dr. Shiping Wen. *(Corresponding author: Adel Ammar.)*

Anis Koubaa is with the Department of Computer Science, and the Robotics, and Internet-of-Things Lab, Prince Sultan University, Riyadh 12435, Saudi Arabia (e-mail: akoubaa@psu.edu.sa).

Adel Ammar, Anas Kanhouch, and Yasser AlHabashi are with the ROITU LAB, Prince Sultan University, Riyadh 11586 Saudi Arabia (e-mail: aammar@psu.edu.sa; Akanhouch@psu.edu.sa; yalhabashi@psu.edu.sa).

Digital Object Identifier 10.1109/TNSE.2021.3055835

I. Introduction

REAL-TIME video analytics is an exponentially evolving field with the emergence of deep learning applications and the proliferation of IoT devices. With the increasing evolution of Artificial Intelligence, the deep learning market is valued in 2017 at USD 2.28 Billion. It is expected to be eight times higher at the end of 2023 with a Compound annual growth rate (CAGR) of 41.7%, according to Market and Market report [1]. The amount of devices generating data is increasing tremendously, and according to CISCO, there are 50 billion IoT devices connected to the Internet. Among all IoT devices, the use of cameras for surveillance and monitoring is estimated to receive a market share of 45.5 Billion in 2020 and is expected to rise to 74.6 billion by 2025, according to Market and Market report [1]. It is also reported that there are around 1 Billion cameras deployed globally by 2021, which generates an incredible amount of sensor data to process in different video analytics applications areas such as access control (e.g., airports, buildings) [2], assets management and operation [3], traffic monitoring and engineering [4], retail analytics [5], logistics [6], and much more. This infers that the combination of deep learning applications, particularly those related to computer vision and real-time video analytics, is expected to generate a massive amount of data. Consequently, it poses several challenges in terms of processing, communication, and storage, considering the scale of the data and the real-time requirements of these applications. Therefore the deployment of such applications in the real-world poses severe challenges due to the conflict between the constraints on the devices or network and the requirements of these applications [7].

A. Cloud vs Edge

There has been a lot of dilemma and controversial opinions regarding the deployment strategies of deep learning applications, which can be roughly categorized into two folds, namely: (1.) cloud-based deployment [8], [9], where all computations are offloaded to and performed in the cloud, (2.) edge-based deployment, where the computation is performed closer to the edge device, and only the meta-data is sent to the cloud [10], [11]. While the cloud provides abundant resources in terms of storage, processing, and energy, this approach cannot scale well with the increasing number of camera devices; particularly, video streaming and processing is quite resource-greedy. On the other hand, the processing of edge devices provides better flexibility, but it suffers from a much lower

2327-4697 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

TABLE I
SUMMARY OF RELATED WORKS WITH DIFFERENT COMPUTATION STRATEGIES

Strategy	Reference	Application	Models	Device	Main Idea	Results
Single-Edge computation	Taylor et al., 2018 [10]	Image classification	MobileNet v1 [30] ResNet v1 & v2 [31] Inception v2 [32]	Jetson TX2	Adaptive scheme to select the best DL model to use for each input	7.52% improvement in testing accuracy, and 1.8x reduction in inference time compared to the best single DL model.
	Alzantot et al., 2017 [13]	Image classification	Inception [32]	Nexus 5x Nexus 6	Accelerate the execution of DL networks on commodity Android devices using RenderScript	Up to 3x speedup of matrix multiplications with RenderScript compared to original Tensorflow
Distributed computation across edge devices	Lane et al., 2016 [23]	Image classification Character recognition Speech recognition Audio classification	AlexNet [33] SVHN [34] 2-layer custom NN 2-layer custom NN	Qualcomm Snapdragon 800 Nvidia Tegra K1	Resource control and scaling algorithms that subdivide DL networks into smaller blocks to be executed on different edge devices	From 76% to 93% memory reduction, with less than 5% loss in accuracy.
	Huang et al., 2017 [11]	Character recognition	CNN / LSTM (Non specified)	DL cluster: 3 desktops (Intel i7-6850K CPU Dual GeForce GTX 1080Ti) Edge server: Intel Core i7-3770 @ 3.4 GHz quad core 16 GB 1333 MHz DDR3 RAM Client: Google Nexus 9 Android Tablet	Performing PCA dimension reduction on the edge server to minimize network traffic and running time	Up to 3x speedup and 20x traffic reduction with less than 10% loss in accuracy
	Minh et al., 2020 [18]	Object detection	SSD [35] Faster RCNN [36] Mask RCNN [37]	Edge device: Raspberry Pi 4 Quad core Cortex-A72 CPU @ 1.5GHz 4GB RAM LPDDR4	Light-weight framework for remotely deploying DL as a service on edge devices	Faster RCNN shows the highest latency, CPU and RAM consumption
Partial offloading	Xu et al., 2019 [21]	Digit recognition Image classification Activity recognition Document classification Emotion recognition Speech recognition	MobileNet [30] GoogLeNet [38] LSTM-HAR [39] DeepSense [40] TextRNN [41] DeepEar [42] WaveNet [43]	COTS smartphones and smartwatches	Context-aware offloading of DL tasks from wearable to hand-held devices to improve performance and reduce energy footprint	Execution speedup: 5.08x and 23.0x Energy saving: 53.5% and 85.5% compared to wearable-only and handheld-only computations
	Ran et al., 2018 [19]	Object detection	YOLO [44] for Android Tensorflow	Backend server: Intel processor @ 2.7 GHz 8 GB of RAM GeForce GTX970 (4GB of RAM) Edge device: Samsung Galaxy S7	Determine an optimal offloading strategy based on video quality, network usage and condition, battery constraints, model accuracy, and latency	Higher accuracy than baseline approaches, at 15 FPS, while adapting to the network condition
	Drolia et al., 2017 [20]	Image classification	LSH [45]	Cloud server: 24 CPU cores @ 2.0 GHz 32 GB RAM Edge server: 8 CPU cores @ 2.2 GHz 8 GB RAM Mobile device: 4 CPU cores @ 1.5 GHz 2 GB RAM	Prefetching and caching parts of the trained classifiers onto the devices to reduce the need to offload images to the cloud	Pros: Up to 5x latency reduction Improved accuracy Cons: 5% higher power consumption Does not apply to DL
	Yu et al., 2020 [46]	Non-specified	Non-specified	Edge server: 16 cores Edge network: Rayleigh-fading environment with 256 subcarriers	Using Deep Imitation Learning to minimize the fine-grained computation offloading cost in time-varying network environments	Better than the state of the art, in terms of offloading decision accuracy (64.7%), cost reduction (23.17%), and execution time.
Total offloading	Liu et al., 2018 [9]	Face verification Object detection Activity detection	FaceNet [47] DetectNet [48] SSD [35]	Edge server: Intel i7-6700 CPU @ 3.40GHz Nvidia GeForce GTX 1060 6GB 24GB system RAM	An API that manages edge nodes and services, and provides high-level abstraction of DL frameworks	Provides up to 40% inference speedup compared to original Caffe framework
	Liu et al., 2019 [8]	Object detection Human keypoint detection	Faster RCNN [36] Mask R-CNN [37]	Edge device: Jetson TX2 Cloud device: Intel i7-6850K CPU Nvidia Titan XP GPU	Decoupling the rendering pipeline from the offloading pipeline, and using an object tracking method to maintain detection accuracy	20.2% - 34.8% increase in accuracy, 27.0% - 38.2% decrease in false positives, for the object detection, and human keypoint detection, respectively. 2.24ms latency for object tracking. Consumes only 15% of CPU resources, and 13% of GPU resources.
Comparative studies	Koubaa et al., 2020 [12]	Object detection	Yolov4-tiny [49]	Cloud servers: 1) CPU:Intel i7-8700K @ 3.7 GHz GPU: GTX 1080 (8GB) RAM: 64 GB 2) CPU:Intel i9-9900K @ 3.7 GHzGPU: RTX 2080 Ti (11GB) RAM: 64 GB Edge servers: Jetson Nano (4 GB RAM) Raspberry Pi 4 (4 GB RAM) End devices: 4G-connected custom drones with on-board Raspberry PI 3	Computation offloading architecture for Internet-connected drones. Comparing cloud vs edge computation, in terms of energy, bandwidth, and latency	Cloud computation allows a higher inference speed, despite a larger latency
	Our work	Face Recognition	FaceNet [47] with MTCNN [50]	4 different cloud servers (Table 3) 5 different edge devices (Table 4)	Comparing three deployment strategies (cloud-based, edge-based, and hybrid). Investigating the effect of DL model optimization using TensorRT and TFLite compared to standard Tensorflow.	TensorRT provides the fastest execution on all cloud and edge devices, at the expense of significantly larger energy consumption

performance of these devices in terms of processing. Several papers have investigated the edge-versus-cloud dilemma [8]–[13] and are summarized in Table I.

B. The Rise of Edge

In recent years, edge computing emerged as an increasing trend compared to cloud computing for several reasons [14]. First, with the explosion of the number of devices, the cloud solution becomes unsustainable to support the increasing demands in processing and storage demands. Besides, it usually leads to increased latencies due to communication, particularly in real-time video analytics. Second, the edge devices' capabilities have been growing tremendously over the last three years, and there are several available options in the market nowadays. Initially, Raspberry Pi was considered a de

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

facto standard as an edge device for IoT applications for several years. It was still a limited solution for video analytics, as it does not have sophisticated graphics processing units (GPU). More recently, NVIDIA has come up with different platforms having various capabilities and tradeoffs between cost and performance known as the Jetson boards. Jetson TX2 was the triggering platform for these AI-enabled edge devices. Later, several other platforms were proposed, including the Jetson Nano, Jetson NX, and lately, the Jeston AGX board (see a summary in Table IV).

Apart from improving hardware platforms, big deep learning companies such as NVIDIA and Google have been working on optimizing inference models. Models trained with deep learning frameworks, such as Tensorflow and PyTorch, are not designed to execute on low-power devices, which hinders their performance and renders them unpractical. Therefore, different types of optimization frameworks were proposed, among them Tensorflow Lite (TFLite) from Google and TensorRT from NVIDIA are now the leading solutions. TFLite was mainly developed for iOS and Android mobile devices (i.e., phones, tablets), whereas TensorRT was designed for more hardware-oriented optimization targeted for NVIDIA GPUs. These hardware acceleration techniques have also contributed to the development of edge computing approaches for real-time video analytics.

C. Contributions

This paper focuses on the real-time face recognition application using deep learning models and its deployment on the cloud versus the edge. Face recognition is among the most popular applications in real-time video analytics, and its execution on edge share the challenges above. The face recognition application has two deep learning models in the pipeline, including a face detection module and a face identification (or recognition module), which results in more demanding computing requirements in terms of processing, memory usage, and energy. Through a series of 294 methodical experiments on nine different cloud or edge GPU devices, we evaluate the impact of platforms, framework implementations, input resolution, and operating systems. Besides, we provide a detailed comparative analysis for all GPU platforms in terms of execution times (detection, recognition, and total), energy, and memory usage.

The rest of this paper is organized as follows. Section II provides a comprehensive overview of previous works related to different deployment strategies of video analytics on the cloud and edge. Section III presents a detailed background on face recognition systems and their characteristics. In Section IV, we present the non-functional requirements for the deployment of face recognition systems and the different deployment strategies. Section V describes the experimental methodology conducted in this paper to evaluate cloud and edge devices' performance for real-time face recognition systems. Section VI presents the experimental results and discusses the performance of edge versus cloud deployments. Finally, we conclude the paper in Section VII and outline future works.

II. Related Works

The ability of mobile devices to offload intensive computations toward a cloudlet was first demonstrated by Noble *et al.* [15], in 1997, for a speech recognition application. However, over the past few years, with the emergence of deep learning and edge servers with ever-expanding capabilities, there has been a hugely increased number of works that explored various computation offloading techniques for data-intensive applications from edge devices to edge servers or to the cloud. Hu *et al.* [16], in 2016, have proven edge computing's ability to provide a substantial gain in latency, bandwidth, energy consumption, and scalability compared to a cloud-only solution and predicted the ubiquity of edge computing infrastructures. More recently, Zhang *et al.* [17] exposed multiple research challenges and opportunities for leveraging deep learning capabilities in edge computing and predicted that this domain would bring the biggest challenges over the next decade.

Deep learning applications in edge computing cover a wide range of areas including object detection [8], [9], [12], [18], [19], image classification [10], [13], [20], human pose estimation [8], activity recognition [9], [21], brain-computer interface [22], character recognition [11], [23], speech recognition [21], [23], autonomous driving [24], and document classification [21]. Most of these works were mainly concerned with speeding up inference performance on edge devices, without significant accuracy degradation. The closest to the present study is the work of Koubaa *et al.* [12], who designed a computation offloading system architecture for 4G-connected drones, and compared cloud computation to edge computation in terms of energy, bandwidth, and communication latency, for an object detection application. However, this work considered a basic object detection application using Yolo as opposed to this paper, where we address a face recognition system (including detection and recognition modules). Furthermore, in [12], the authors evaluated a limited number of platforms and did not investigate the effect of model optimization using TFLite and TensorRT, as we proposed in this paper

Wang *et al.* [14], and Chen and Ran [25] presented theoretical surveys that compared several edge computation and communication modes used in the literature (integral or partial offloading, and vertically or horizontally distributed inter-edge collaboration) and their reported performance. Nevertheless, the compared works used disparate edge/cloud devices and network types, besides applying deep learning models of diverse complexity, making any quantitative performance comparison inconsistent.

On the other hand, Mittal [26] presented a survey on the architectural and algorithmic optimizations for accelerating NN applications on NVIDIA's Jetson platforms (TK1, TX1, and TX2) in various areas (medical, farming, robotics, image processing, speech recognition, autonomous driving, traffic surveillance, and drone navigation). The author also reviewed research papers that compared Jetson boards and similar low-cost, low-power platforms. Nevertheless, he did not consider the recent Jetson Xavier, which was released in March 2019.

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

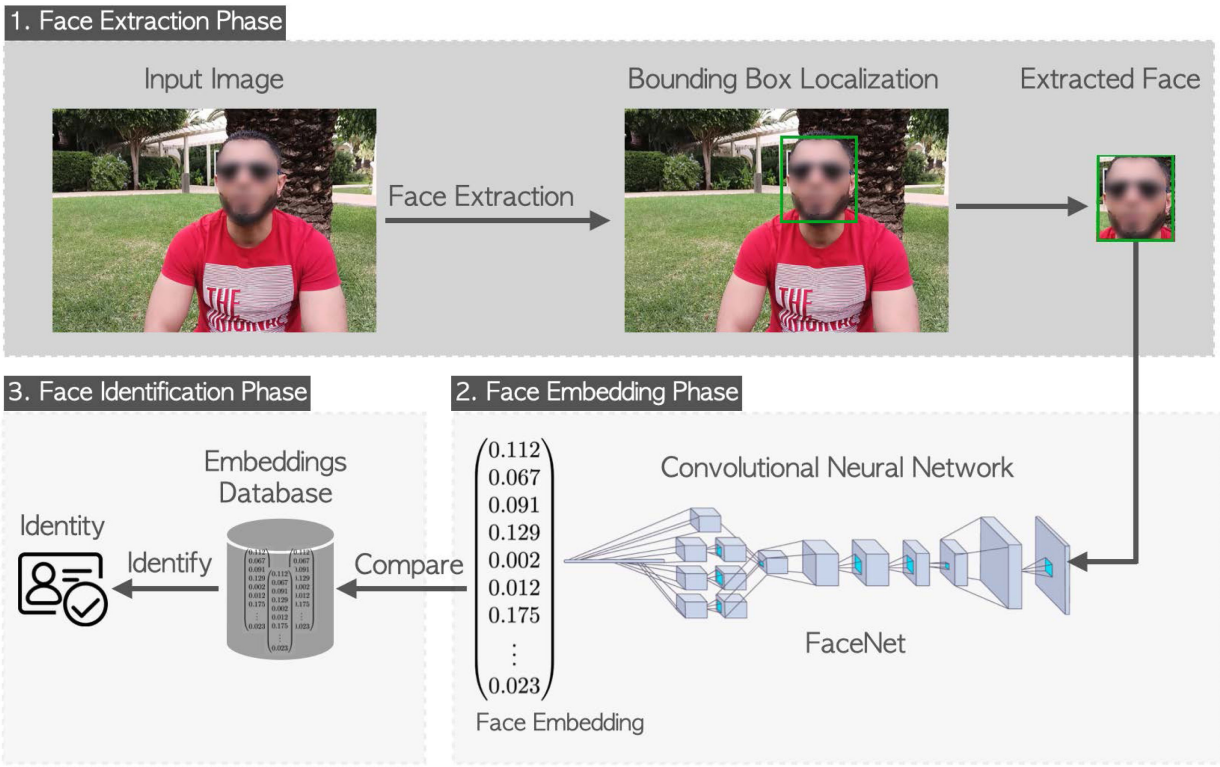

Fig. 1. Face Recognition System Workflow.

Some recent works addressed the problem of training DL models on end devices before transferring them to edge servers or the cloud. In this scope, the concept of Federated Learning (FL) allows edge devices to be collaboratively trained, then to send only model updates rather than raw data. These updates are then aggregated in the server, which addresses the concerns of security and privacy. Wang *et al.* [27] implemented such a solution for face recognition using secure sparse representation, whereas Lim *et al.* [28] presented a comprehensive survey on the use of this approach in mobile edge networks in different areas. However, in this paper, we are only interested in DL models' inference phase since training abilities on edge devices are still limited due to their constrained resources.

Table I summarizes the main related works that investigated various deployment strategies from end devices to edge or cloud servers, specifies their deployment context, and reports their achieved results. Our work's added value is the focus on the practical aspects of deploying a face recognition application. It compares several deployment strategies by assessing their efficiency using various types of cloud GPUs (Table III) and edge devices (Table IV). To the best of our knowledge, only Liu *et al.* [9] investigated the performance of DL face detection and verification on edge devices. Still, they focused especially on the deployment framework and tested face detection only as part of a prototype to demonstrate their proposed framework's efficiency. Kosta *et al.* [29] had a similar objective. They tested a face detection algorithm, among other scenarios, to evaluate their code offloading technique. Still, it was before the advent of deep learning and the release of NVIDIA embedded computing boards.

III. Background on Face Recognition

A. Overview

Face Identification Systems are typically of two types:

- *Face Verification:* the input is an image of a face and the name or ID of the person to identify. A face verification system's output is whether or not the input image corresponds to the claimed identity.
- *Face Recognition:* the input of a face recognition system is an image of a face and a database of faces for N persons. The system's output is the identity (name or ID) of the most likely person in the database that corresponds to the input face.

In this paper, we focus on face recognition systems. They have been increasingly popular with the emergence of deep learning algorithms. A face recognition inference system comprises two major operations (Fig. 1):

- *Face Detection:* the first step (Phase 1 in Fig. 1) consists in finding and extracting the bounding boxes of one or multiple faces in an input image. The earliest solution to this problem is the Viola-Jones object/face detection approach [51], which dates back from 2001. The Viola-Jones algorithm is based on hand-crafted Haar-feature-based cascade classifiers. Nowadays, The state-of-the-art approach for face extractions is Multitask Cascaded Convolutional Networks (i.e., MTCNN) [50] released in 2016, and based on convolutional neural networks. In this paper, we use MTCNN as a face detector.
- *Face Identification:* This second step (Phase 2 and Phase 3 in Fig. 1) takes as input the list of extracted

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

faces and processes them through a face recognizer to transform the raw image into a one-column vector representation, called *face embedding*. The face embedding is compared to other face embeddings in a database to finally find the person's identity. There are several approaches proposed in the literature for face identification, including DeepFace [52] from Facebook (2014), VGGFace [53] and VGGFace 2.0 [54] from University of Oxford (2015 and 2017 respectively), FaceNet from Google (2015) [47], OpenFace [55] and OpenFace2 [56] from University of Pittsburgh (2016 and 2018 respectively). In this paper, we use FaceNet [47] as a face recognizer.

In what follows, we will denote the face identification step as face recognition.

B. Face Embedding

The critical phase in any face recognition system is transforming a raw image of a face into a one-column vector representation called *face embedding*, presented as Phase 2 in Fig. 1. This operation is performed through a convolutional neural network that takes as input a square image of a pre-defined size and outputs the face embedding. The typical face embedding size is a power of 2, where 128 and 512 are the two most common sizes of the face embeddings. FaceNet, proposed by Google in 2015, represents a state-of-the-art face recognition algorithm that generates face 128 and 512 face embeddings. It is a convolutional neural network trained on a large dataset of faces to generate embeddings of faces.

Although it uses a standard convolutional neural network, the training of the face recognition system, such as FaceNet, is different from training standard object classifiers. In the context of object classification, the network is trained to differentiate between different types of objects, where each instance of an object has its label (e.g., cat vs. not cat). However, this training approach is not appropriate for the face recognition system; the number of persons that we may need to recognize is variable, whereas object classifiers are based on a static number of objects. The use of a *Softmax* function at the output of face recognition is not consistent because the number of persons to recognize may increase or decrease dynamically, and retraining the classifier for every change is not appropriate. The alternative to training a face recognizer is to use *Siamese Networks* [57] for one-shot image recognition. They represent twin neural networks with the same weight and are trained in tandem on two different input vectors to compute comparable output vectors. In other words, one-shot learning uses a limited training dataset to learn a similarity function that quantifies how different two given images are. Siamese Networks are used to train a network to generate an encoding of an object such as similar objects (or faces) with an encoding vector with a *close distance* to each other. In contrast, different objects (or faces) would have embedding very far from each other. The distance is evaluated and optimized using two types of loss functions used to train such networks:

- *Triplet Loss:* the loss is calculated with respect to a reference image, called anchor, that is compared with an image of the same class (positive image) and also compared with an image of a different class (negative image). The triplet loss is optimized by reducing the distance between the anchor and its positive example and increasing the distance between the anchor and its negative example. The Triplet loss aims at guaranteeing this inequality for all triplet pairs of images:

$$\|f(A) - f(P)\|^2 + \alpha \leqslant \|f(A) - f(N)\|^2 \tag{1}$$

  where $f(x)$ is the embedding function to learn, such that the distance between the embedding of the anchor A and its positive counterpart P is smaller than the distance between the embedding of the anchor A and its negative counterpart N. The α increment is used to avoid the special case $f(Image) = 0$, which compromises the loss function and the training. As such, the Triplet loss function $l(A, P, N)$ for an Anchor A, a Positive image P, and a Negative image N is expressed as:

$$l(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha) \tag{2}$$

- *Similarity Loss:* the loss is calculated with respect to two images trained on two similar networks in tandem to learn about the embedding, such as if these two images are of the same class, the output will be *one*, and if the two images are of different classes, the output will be *zero*. The training is reduced and similar to a logistic regression classification considering that the output is binary. The similarity loss function $l(A, B)$ for two images A and B is expressed as:

$$l(A, B) = \sum -y\, log(\hat{y}) - (1 - y) log(1 - \hat{y}) \tag{3}$$

  Where y is the embedding of image A, and $\hat{y}$ is the embedding of image B, such that $\hat{y} = \sigma(\sum_K^{128} (w_k * |f(x_k^i) - f(x_k^j)| + b))$ is the output vector from the logistic regression function of the neural network.

Upon training a neural network either with a Triplet loss or Similarity loss, the resulting network will take an image of a face as input and generate an embedding in the output. Once an embedding is generated, it will be compared against the embeddings of known faces stored in a database (Phase 3 in Fig. 1).

There are mainly three ways to make this comparison and identify the person (Phase 3 in Figure): 1):

- *Classifier:* a classifier is trained with the input being the embedding and the output being the identity of the person corresponding to this embedding. This approach's limitation is that the neural network has to be trained for any new person added or removed from the database. Possible classifiers are k-nearest neighbor

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

algorithm (KNN), Support Vector Machine (SVM), and a fully-connected neural network.
- *L2-Distance:* the Euclidean distance is measured between the two embedding vectors. The identity is assigned to the embedding with the lowest distance.
- *Voting:* Assuming that the database contains more than one image of the same face, the voting strategy consists of finding the max number of embedding that is closer to the face's embedding to recognize.

IV. Deployment Architecture in Production

The deployment of deep learning projects in production environments has always been a very challenging task. The main reason is that there are several constraints and non-functional requirements that must be satisfied, much beyond the need for high accuracy, which is the main objective during the training phase and the evaluation of deep learning models. During the deep learning model development phase, the main focus is to achieve the highest accuracy possible without much attention to some environment-specific constraints. However, when the model has to be deployed, these initially ignored constraints give rise to several problems for the model's operation in its target environment. These challenges also hold when deploying a face recognition application.

In this section, we will provide an overview of the constraints of deployment, in addition to the non-functional requirements of deploying a face recognition application. We will then present the possible deployment architectures to address the non-functional requirements while coping with the environments' constraints.

A. Non-Functional Requirements

In the industrial-level deployment of a face recognition model, several non-functional requirements must be satisfied, which are enumerated as follow:

- Real-Time: in a face recognition application, it is required to process the frames in real-time, including face detection and face recognition tasks. Real-time performance is typically measured by the number of frames per second (FPS). However, achieving a high FPS rate represents a real challenge when the deployment is on edge devices, considering the limitation of computation resources, which induces high processing delays. In a cloud deployment, the real-time issue is more related to communication delays rather than processing delays.
- Energy-Efficiency: the main objective is to reduce the energy consumption of the face recognition applications in particular when deployed on edge devices. These devices can be battery-powered and operate in areas without sufficient power supplies in some cases. Even when being mains-powered, reducing energy consumption is essential to keep the operational cost as low as possible, particularly with the increasing cost of the energy supplies.
- Scalability: In real-world deployment use cases, it is common to use multiple cameras as data sources, which brings an additional challenge. On the one hand, processing every camera stream on a standalone computing device (typically an edge device) would be costly. On the other hand, combining the processing of multiple streams on the same computing device (edge or cloud device) would put more load and burden on the machine. It may compromise the quality-of-service (more delays, higher energy consumption). Therefore, it is crucial to consider optimized deployment architecture to cope with the scalability requirement.
- Security: In deep learning applications in general, particularly in face recognition, the data carries out confidential information unveiling people's identities or assets. Therefore, it is crucial to implement security mechanisms to prevent possible attacks, such as eavesdropping, man-in-the-middle, jamming, and other traditional threats in communication networks. Furthermore, adversarial machine learning approaches represent a significant threat as they attempt to leverage deep learning models' vulnerabilities to compromise the output prediction's correctness. Adversarial machine learning is a very active research area attracting several research works [58]–[61].
- Privacy: The use of artificial intelligence for surveillance and people's behavior monitoring represents a dilemma in what concerns privacy-preserving, particularly for face recognition. It exposes the personal information about people being monitored, putting their privacy at risk. Several recent research works attempted to develop privacy-preserving deep learning models [62]–[65]. It is essential that this collected personal information is only accessible to authorized users and does not violate the monitored people's privacy and their assets, particularly in face recognition applications.

The satisfaction of all these conflicting non-functional requirements represents the main objective of deploying face recognition in a production environment. It raises several questions on which deployment architecture can achieve the best trade-off among the non-functional requirements. In the next section, we present three possible deployment solutions, and we discuss their advantages and limitations.

B. Deployment Architecture

The objective of designing a deployment architecture is to find the best strategy to hookup the sensor devices (i.e., camera) with the computing devices (edge/cloud) while satisfying the non-functional requirements. In what follows, we mainly focus on real-time and energy-efficiency requirements. We aim at achieving the highest frame per second rate possible while keeping energy consumption at its lowest level. Considering a typical face recognition, it has two primary operations that require extensive computation (as illustrated in Fig. 1) that may increase processing delays and induce higher energy

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

TABLE II
DEPLOYMENT STRATEGY CHARACTERISTICS

Characteristics	Cloud Deployment	Edge Deployment	Hybrid Deployment
Communication Delays	High	Low	Low
Processing Delays	Low	High	Medium
Energy Consumption	High on Cloud, Low on Edge	High on Edge, Low on Cloud	Balanced between Edge and Cloud
Scalability	Limited Scalability	High Scalability	High Scalability
Security and Privacy	Low Privacy-Preserving	High Privacy-Preserving	Medium Privacy-Preserving

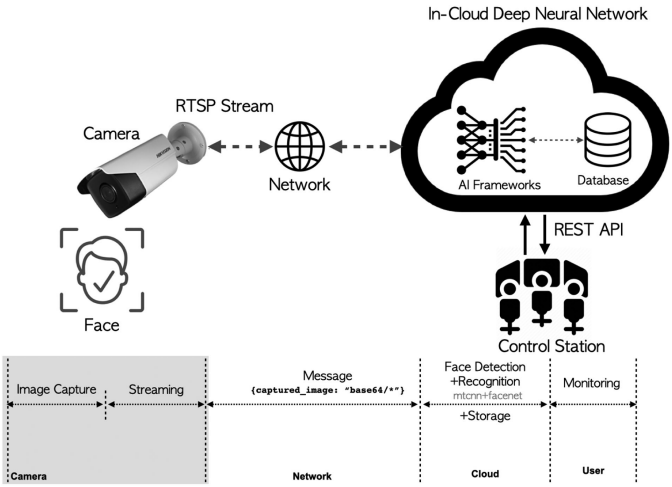

Fig. 2. Face Recognition Cloud Deployment.

consumption: (1) face detection using the MTCNN algorithm, (2) the face recognition using FaceNet.

The questions that we address in this section to design a deployment architecture are where the face recognition computing tasks should be implemented? Should it be fully in the cloud? or entirely on edge? or a combination of both? In what follows, we present the different possible strategies and discuss their advantages and drawbacks.

Table II presents a summary of the different deployment strategies' characteristics.

1) Cloud-Based Deployment: The cloud-based deployment is illustrated in Fig. 2. It is often referred to as *computation offloading* [12] in the literature, as computation-intensive tasks are offloaded from the end-device to the cloud for remote processing. The camera is connected to a cloud server through an RTSP stream over the Internet or a local network. The camera's frame rate is constrained by the bandwidth available over the communication channel with the server. With the emergence of 5 G networks, the communication delay and jitter tend to decrease. However, it remains a major component in the end-to-end delay of the whole application, particularly for video streams with Ultra High-Definition (UHD) quality or higher. The jitter also induces an additional issue because of the best-effort nature of the Internet. There are no guarantees that all frames will encompass the same delay.The cloud server receives the video stream at a specific FPS rate through the RTSP interface and redirects it to the deep learning processing module. The latter executes both the face detection algorithm using MTCNN and the FaceNet face recognizer in real-time and stores the result into a monitoring and visualization database. One challenge is that the number of frames processed per second may be lower than the ingress video stream's frame rate. In this case, the frame loss might result in dropping them due to the unavailability of processing resources. However, usually, the GPUs of cloud servers have high processing performance (e.g., Nvidia V100, RTX 8000, RTX 2080Ti), which allows them to achieve high FPS after deep learning inference. The cloud-based GPU devices used for the experiments are presented in Table III. Note that for the RTX 2080Ti GPU, we have conducted the experiments on two machines having the same hardware characteristics (CPU, GPU, RAM) but running on different operating system versions (Ubuntu 16.04 and Ubuntu 18.04). In the performance evaluation section, we will discuss the inference speed on GPU cloud resources and the impact of the operating system.

Regarding energy-consumption at the edge level, only the camera consumes energy for streaming the video to the cloud over the Internet. There is no internal processing at the edge level that would induce more energy dissipation. Communication is the primary source of power drainage. At the cloud level, the energy consumption will be the highest due to (1) communication by receiving the video stream continuously from the camera (2) deep learning inference, as it will process every frame to detect faces using MTCNN and recognize them using FaceNet. Indeed, this approach will put a high load on the cloud server, and as a consequence, will become less scalable. In fact, as the number of cameras increases, a single cloud's resources might quickly turn out to be insufficient to handle a large number of cameras, for example, at the scale of a city.

As for Security and Privacy, with cloud deployment, it is required to stream the entire image frame from the camera to the cloud, which exposes the whole captured data to possibly unauthorized access or threats. Besides, if the frames are processed and stored in a public cloud, all the personal information will be available to the cloud service provider, representing a security and privacy gap.

2) Edge-Based Deployment: The edge-based deployment is illustrated in Fig. 3. It is usually known as on-board processing or local processing because the extensive computation tasks are executed close to the data sources. In this deployment, the edge device is composed of the sensor, which is the camera in our case, and an embedded device (e.g., GPU Computing Device, such as a Jetson Board) attached to the camera through a proper channel (e.g., RTSP, USB, or Serial). In this case, the communication bandwidth between the camera and embedded device is high since a typical USB3 bandwidth reaches 5 Gbps, and a typical Ethernet connection reaches

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

TABLE III
CHARACTERISTICS OF THE CLOUD-BASED GPU DEVICES USED FOR THE EXPERIMENTS

Cloud-based GPUs	GTX 1080	RTX 2070	RTX 8000	RTX 2080 Ti (1)	RTX 2080 Ti (2)
GPU Architecture	Pascal	Turing™	Turing™	Turing™	Turing™
CUDA Parallel-Processing Cores	2560	2304	4608	4,352	4,352
NVIDIA Tensor Cores	N/A	288	576	544	544
NVIDIA RT Cores	N/A	36	72	46	46
GPU Memory	8 GB GDDR5X	8 GB GDDR6	48 GB	11 GB GDDR6	11 GB GDDR6
FP32 Performance (TFLOPS)	8.873	9.062	16.3	13.45	13.45
Max Power Consumption	184 Watts	215W	260W-295W	280W	280W
Operating System	Ubuntu 18.04	Ubuntu 16.04	Ubuntu 18.04	Ubuntu 18.04	Ubuntu 16.04
CUDA version	10.2	10.2	10.0	10.2	10.0
CuDNN version	7.6.5	7.6.5	7.6.5	7.6.5	7.6.3
Tensorflow version	1.15.0	1.14.0	1.14.0	1.14.0	1.14.0
TFLite version	tflite-runtime 2.1.0.post1	Tensorflow subpackage (tf.lite)	tflite-runtime 2.1.0.post1	tflite-runtime 2.1.0.post1	Tensorflow subpackage (tf.lite)
TensorRT version	7.0.0.11	7.0.0.11	7.0.0.11	7.0.0.11	7.0.0.11

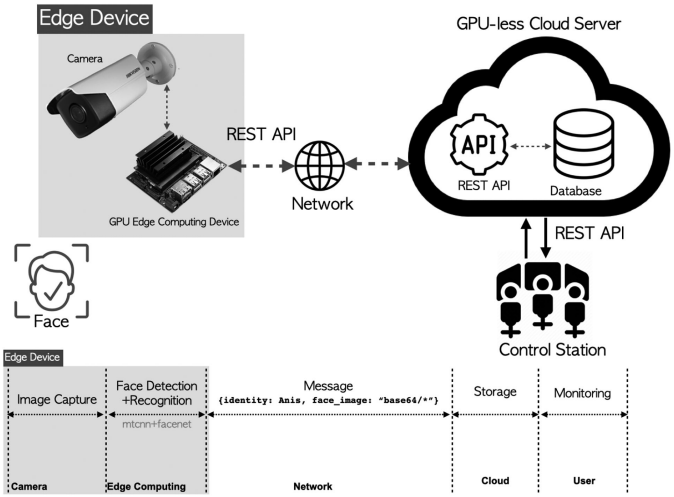

Fig. 3. Face Recognition Edge Deployment.

10 Gbps. Consequently, video streaming from the camera to the computing device has no communication bottleneck (no delay and jitter issues). Still, it will be constrained by the computing and storage performance of the embedded device.

The edge embedded device receives the camera frames at a certain FPS rate and processes them in real-time locally. The edge GPU device performs the two operations for face recognition, including face detection using MTCNN and face recognition using FaceNet. The biggest challenge in edge deployment is whether the edge embedded device's capabilities are sufficient to process the incoming frames in real-time with a decent FPS rate. The answer depends on two factors: *The embedded device's specification.* : There are various types of embedded devices available in the market with different performance and costs. The characteristics of the edge devices used for the experiments are presented in Table IV. For example, A Jetson Nano has much less computing and storage resources (4-core CPU, 128 CUDA cores, and 4 GB Memory) compared to Jetson Xavier AGX (8-core CPU, 512 CUDA cores, and 32 GB Memory), and consequently consumes much less power. In the case of edge deployment, the performance of a face recognition application heavily depends on the used platform. In the experimental evaluation section, we will compare in detail these edge devices. *The deep learning framework.* : the inference performance heavily depends on the deep learning framework used for inference. Once a model is trained with a particular framework, for example with Tensorflow, it is used for inference on the same framework. However, the performance might be an issue when running on Tensorflow in an edge device since the model contains several additional layers needed for training and not required for inference. Furthermore, the trained models usually have 64-bit precision (FP64), which makes them more computationally extensive during inference. Considering the dramatic need for high-speed inference, several optimization frameworks were proposed that perform different optimization procedures to speed-up the inference time. The most important inference frameworks are TensorRT from NVIDIA and TFLite from Google. TensorRT performs various optimization types specifically for NVIDIA GPUs and Jetson boards, which are heavily dependent on the target device. It calibrates weights and activation functions' precision to be reduced to FP32, FP16, and even INT8, without much accuracy loss. This aims at maximizing the throughput (i.e., the number of images processed by the inference model) by quantization. It also performs layer and tensor fusion, which optimizes the memory and bandwidth usages of the GPU by fusing different nodes. Besides, it ensures kernel auto-tuning by selecting the best layers based on the target platform, dynamic tensor memory by minimizing the memory footprint, and finally multi-stream execution to provide a scalable design to process multiple streams in parallel. TensorRT drastically accelerates the inference performance of deep learning models, specifically on NVIDIA GPUs and Jetson Boards. On the other hand, TFLite (i.e., Tensorflow Lite) is a model optimization toolkit whose purpose is also to reduce the complexity of deep learning models and speed-up the inference time. One prominent feature of TFLite as compared to TensorRT is that the optimization is

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
CHARACTERISTICS OF THE EDGE-BASED DEVICES USED FOR THE EXPERIMENTS

Edge-based GPUs	Jetson Nano	Jetson TX2	Jetson Xavier NX	Jetson Xavier AGX
CUDA Parallel-Processing Cores	4-core ARM A57 @ 1.43 GHz	4-core ARM A57 @ 2 GHz	6-core NVIDIA Carmel ARM 64-bit @ 1.4 GHz	8-core ARM Carmel 64-Bit CPU @ 2.26 GHz
GPU	128 CUDA cores Maxwell @ 921 MHz	256 CUDA cores Maxwell @ 1.3 MHz	384 CUDA cores and 48-TENSOR cores NVIDIA Volta @ 1.1 GHz	512-core Pascal with 64 Tensor Cores @ 1.37 MHz
Memory	4 GB LPDDR4, 25 GB/s	8 GB 128-bit LPDDR4, 58 GB/s	8 GB 128-bit LPDDR4x, 51.2GB/s	32 GB 256-bit LPDDR4, 137 GB/s
Storage	MicoSD	32 GB eMMC 5.1	32 GB eMMC 5.1	32 GB eMMC 5.1
Power	5W-10W	7.5W-15 W	10-15W	10W-30W
Jetpack version	4.4	4.4	4.4	4.4
Operating System	NVIDIA L4T (based on Ubuntu 18.04)	NVIDIA L4T (based on Ubuntu 18.04)	NVIDIA L4T (based on Ubuntu 18.04)	NVIDIA L4T (based on Ubuntu 18.04)
CUDA version	10.2	10.2	10.2	10.2
CuDNN version	8.0	8.0	8.0	8.0
Tensorflow version	1.15	1.15	1.15	1.15
TFLite version	tflite-runtime 2.1.0	tflite-runtime 2.1.0	tflite-runtime 2.1.0	tflite-runtime 2.1.0
TensorRT version	7.1.3	7.1.3	7.1.3	7.1.3

platform-independent; this means that the same optimized models run the same on different GPU platforms. Furthermore, TFLite was more designed to optimize the inferencing on mobile devices such as smartphones for both iOS and Android mobile operating systems. The main features of Tensorflow Lite are (*i.*) size reduction as it reduces the size of the model, so it has less storage size and lower memory usage, (*ii.*) latency reduction, by reducing the inference time mainly through quantization which helps to reduce the complexity of calculation during inference, with only minimal loss of accuracy (due to precision degradation). The optimization in TFLite is based on quantization, clustering, and pruning. Quantization deals with reducing the model's precision from FP64 to lower resolutions (FP32, FP16, INT8). *Clustering* consists in grouping the weights of the layers of the trained model into clusters, then the centroid of the weight of each cluster are shared, which reduces the complexity of a model by diminishing the number of its unique weights. Finally, *pruning* remove the less relevant parameters in the model that have only a minor impact on the prediction results. This helps to reduce the complexity of the model at the expense of some loss of accuracy.

Regarding energy-consumption with full edge deployment, it saves communication energy because there is no longer a video streaming towards the cloud, but at the expense of increased energy consumption due to local processing for the face detection (MTCNN) and face recognition (FaceNet) at the edge. The communication energy consumed will be minimal as only metadata extracted from the images after the inference will be sent to the cloud for local storage and visualization. We will also evaluate the impact of edge computation of the face recognition and detection loop at the edge on the throughput in terms of frames per second.

In terms of scalability, this solution is more scalable than the cloud-based deployment because an edge device, such as NVIDIA Xavier AGX, can process from 1 to 16 camera feeds and only send the metadata to the cloud for storage. In this way, the cloud can manage a much higher number of cameras as most of the heavy computation will be performed at the edge, resulting in better load balancing in a large-scale deployment.

As for Security and Privacy, the full edge deployment is more secure and better preserves privacy because it is possible to control the data to send to the cloud and filter sensitive data in advance.

3) Hybrid Deployment: The hybrid deployment is illustrated in Fig. 4.

It is called hybrid because part of the computation is executed on the edge, and the other part is performed on the cloud. In this deployment, the edge device is attached to the camera, similar to edge-based architecture, and will be used for image capturing, besides face detection and tracking. The face recognition part is not executed on edge, but it is deferred to be executed later in the cloud. This approach's benefit is (*i.*) first, to reduce the computing load on the edge device by only executing face detection on-board using MTCNN and extracting a cropped image of only the detected face or faces. The size of the extracted face(s) will be much smaller than the whole image's size. (*ii.*) Consequently, sending the extracted faces to the cloud for storage and post-processing with a face recognizer (FaceNet) will be much faster due to the small size of the extracted faces and consumes much less bandwidth and

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

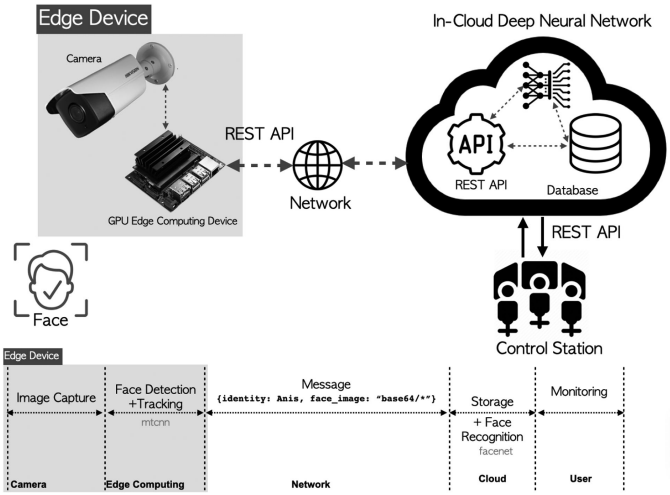

Fig. 4. Face Recognition Hybrid Deployment.

lower energy due to communication. The hybrid approach brings the advantage of alleviating the communication bottleneck and the storage requirements compared to the cloud approach and reducing energy consumption compared to the edge approach.

The hybrid deployment is useful when the edge device has minimal computation and storage capabilities and cannot efficiently handle both the detection and the recognition to be executed simultaneously on-board. It provides a trade-off between the two extreme deployment cases to be entirely on edge or fully on the cloud, in terms of energy consumption, real-time, and security and privacy.

In terms of energy, the edge device consumes power for frame capturing and MTCNN detector, and the transmission of the detected faces' images to the cloud. Using the tracking features with the face detector allows the assign a unique ID to the same faces, and this prevents from streaming the same face multiple times as only new detected faces need to be forwarded to the cloud. It leads to significantly reducing the energy consumption and bandwidth due to communication to the best possible. The scalability of the hybrid approach is similar to the edge approach. It is higher than the cloud scalability as the more massive processing of the video stream is performed at the edge and only extracted faces are processed in the cloud. The advantage is that these extracted faces can be executed for face recognition either online (i.e., in real-time) or offline, in case we would like to perform the FaceNet inference on a large batch size for maximum efficiency.

Regarding security and privacy, it has an intermediate level between the edge and the cloud deployment since it is required to share the faces' data through the network, which may expose this data to threats over the Internet.

V. Methodology

A. Integration of Open-Source Implementations

This section describes the face recognition application's deployment, using MTCNN as a face detector and FaceNet as a face identifier (Fig. 1).

We have developed a modular face recognition application that can be executed on any platform (cloud and edge). The application can be easily configured to run any version of MTCNN (Tensorflow and TensorRT) and any version of FaceNet (Tensorflow, TensorRT, and TFLite). The user needs to set the experiments' characteristics, including the platform type, the deep learning framework, the data source, and some other parameters. The experiments will be automatically executed based on the selected configuration.

We have integrated different open-source implementations in our face recognition library. For MTCNN, we used the Keras/Tensorflow implementation of Iván de Paz Centeno [66]. This MTCNN face detector implementation uses as a reference the implementation of MTCNN from David Sandberg (FaceNet's MTCNN) in FaceNet. It is based on the paper from Zhang [50]. For TensorRT implementation of MTCNN, we have used the implementation of JK Jung available in this link [67]. We have customized and adapted the code of MTCNN TensorRT implementation to our face recognition application to conduct the experiments on the different platforms. For FaceNet, we used the Keras model of David Sandberg's implementation of FaceNet [68]. For the conversion of the FaceNet model to TensorRT, we have developed a script that converts from Tensorflow models to TensorRT using ONNX [69]. To convert Facenet to TFLite, we have used the following converter [70].

At the end of each experiment, we save in a CSV file all the data collected after processing each frame for further analysis and performance evaluation. The objective is to compare their performance in terms of the non-functional requirements (inference speed, energy consumption, communication latency, scalability, security, and privacy) in order to determine the optimal trade-off.

Code, CSV file of the results (+600 K records), analytical dashboards will all be made open-source and public.

B. Experimental Settings

We considered multiple settings for each strategy:

- Cloud Servers: Five GPU servers with different computing capabilities were tested for cloud computation. Their hardware and software characteristics are specified in Table III. To measure the impact of software, we have used two RTX2080Ti machines having the same hardware characteristics but different operating systems (Ubuntu 16.04 and Ubuntu 18.04), CUDA, CuDNN, and TFLite versions.
- Edge Devices: Four edge computing embedded boards with different capabilities and power consumption were tested for the edge-based and hybrid strategies. Their hardware and software characteristics are specified in Table IV.
- Deep Learning Frameworks: For each deployment strategy, we ran the face recognition (FaceNet) inference model using (1) the original Tensorflow model, (2) after a platform-specific optimization using TensorRT, (3) and after optimizing the pre-trained model using TFLite. Besides, we run the MTCNN face detector

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

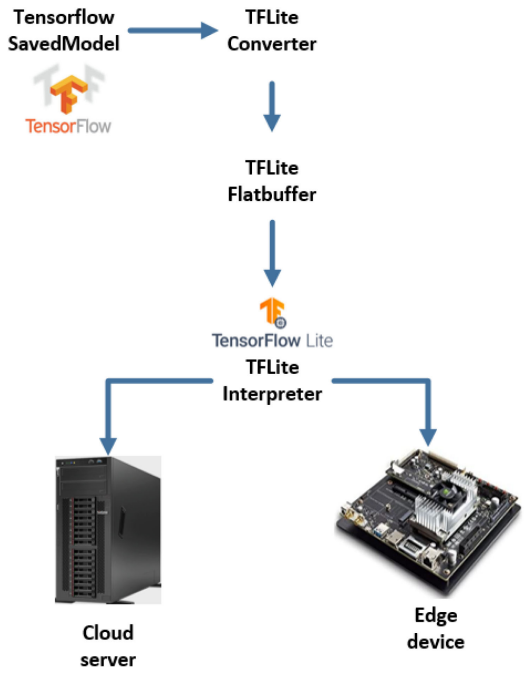

Fig. 5. TFLite Optimization.

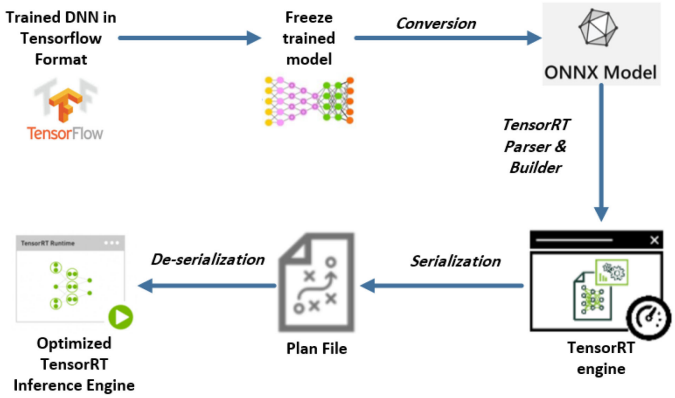

Fig. 6. TensorRT optimization.

using the two first frameworks, as we did not manage to obtain a stable version of MTCNN running on TFLite. As explained in section IV-B2, TensorRT optimizations are platform-dependent, contrary to TFLite. Figs. 5 and 6 depict the optimization process for TFLite and TensorRT respectively. Nonetheless, some edge devices cannot run the original Tensorflow model of MTCNN or FaceNet due to their limited processing capabilities. Table VI enumerates the list of experiments per platform and framework and summarizes the main results that will be discussed in Section VI.

C. Input Data

As input, we used two recorded videos with three different resolutions each (480, HD, and Full HD) and different FPS values, as reported in Table V. The input resolution directly impacts the processing speed, as we will see in Section VI-C.

TABLE V
CHARACTERISTICS OF THE INPUT VIDEOS

	Duration	FPS	Resolution	Width	Height
Video 1	2mn10s	20	480	720	480
			HD	1280	720
			FHD	1920	1080
Video 2	1mn18s	30	480	720	480
			HD	1280	720
		60	FHD	1920	1080

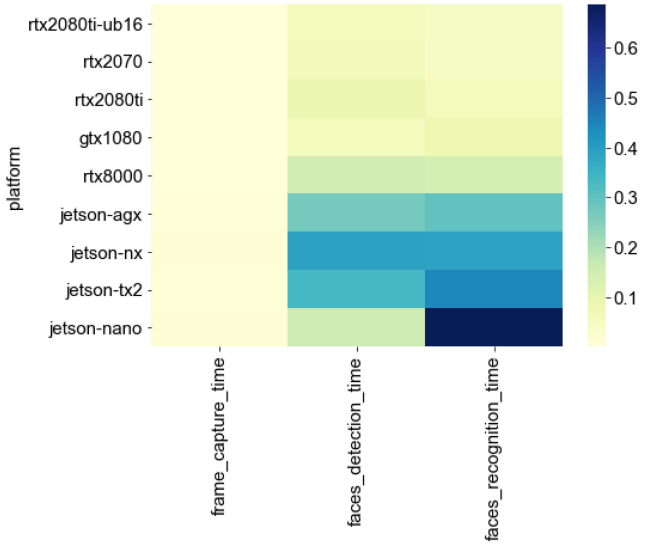

Fig. 7. Heat map of the processing time (in seconds) for each phase (frame capture, face detection, and face recognition) on each platform.

VI. EXPERIMENTAL STUDY AND EVALUATION

A. Impact of the Platform

Fig. 7 depicts the heat map of the processing time of the inference phase's successive stages on each platform. While the frame reading time using OpenCV is negligible on all devices (from 0.6% to 1% of the total time), the detection and recognition times show a large variation between platforms, with a clear contrast between cloud and edge devices. Moreover, the relative contribution of detection and recognition to the total time is approximately equivalent (ranging from 41% to 58%), except for Jetson-Nano, since the TF implementation of MTCNN could not be run on it. Consequently, the detection time on Jetson-Nano only accounts for the faster TensorRT implementation. Note that, for a given platform and implementation framework, the (total) recognition time presented here depends on the number of detected faces obtained in the previous stage. In contrast, the detection time only depends on the input resolution.

More precisely, Table VI shows the face detection and face recognition time for each framework and platform. When analyzing the results per platform, it appears clearly that cloud

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
MAIN RESULTS FOR EACH PLATFORM AND FRAMEWORK, AVERAGED OVER THE SIX INPUT VIDEOS

Platform	Detector..	Face Recognition Mo..	Avg. Faces Recognition Time (s)	Avg. Face Detection Time (s)	Avg. FPS	Avg. Num Of Faces
jetson-nano	TRT	TRT	0.37	0.16	2.59	3.30
		TF	0.84	0.15	1.68	3.30
		TFLite	0.86	0.17	1.61	3.30
jetson-nx	TF	TRT	0.18	0.68	1.48	2.37
		TF	0.45	0.66	1.14	2.37
		TFLite	0.47	0.69	1.13	2.37
	TRT	TRT	0.11	0.09	6.37	3.07
		TF	0.55	0.14	2.53	3.07
		TFLite	0.60	0.11	2.39	3.07
jetson-tx2	TF	TF	0.40	0.68	1.18	2.37
		TFLite	0.49	0.68	1.10	2.37
	TRT	TRT	0.14	0.11	5.14	3.15
		TF	0.56	0.10	2.49	3.30
		TFLite	0.64	0.12	2.12	3.15
jetson-agx	TF	TRT	0.28	0.46	1.75	2.37
		TFLite	0.27	0.47	1.67	2.37
		TF	0.35	0.48	1.53	2.37
	TRT	TRT	0.11	0.08	6.89	3.08
		TFLite	0.34	0.08	3.59	3.08
		TF	0.45	0.08	3.29	3.08
gtx1080	TF	TF	0.07	0.12	6.49	2.40
		TFLite	0.09	0.13	6.27	2.40
	TRT	TRT	0.03	0.02	25.05	3.31
		TF	0.10	0.02	13.16	3.31
		TFLite	0.13	0.02	11.38	3.31
rtx8000	TF	TRT	0.05	0.26	4.10	2.40
		TF	0.12	0.27	3.24	2.40
		TFLite	0.20	0.26	2.82	2.40
	TRT	TRT	0.04	0.03	16.22	3.31
		TF	0.16	0.04	8.20	3.31
		TFLite	0.27	0.03	6.13	3.31
rtx2070	TF	TRT	0.03	0.12	9.11	2.40
		TFLite	0.05	0.11	8.22	2.40
		TF	0.06	0.11	7.47	2.40
	TRT	TRT	0.02	0.02	29.32	3.20
		TFLite	0.06	0.02	18.12	3.20
		TF	0.08	0.02	16.29	3.20
RTX 2080ti (Ubuntu 18.04)	TF	TRT	0.02	0.16	7.79	2.40
		TFLite	0.08	0.16	5.84	2.40
		TF	0.07	0.17	5.67	2.40
	TRT	TRT	0.02	0.01	34.68	3.31
		TF	0.10	0.02	15.35	3.31
		TFLite	0.10	0.01	15.23	3.31
RTX 2080ti (Ubuntu 16.04)	TF	TRT	0.02	0.11	10.40	2.40
		TFLite	0.05	0.11	8.59	2.40
		TF	0.06	0.11	7.78	2.40
	TRT	TRT	0.02	0.02	31.20	3.31
		TFLite	0.07	0.02	18.48	3.31
		TF	0.08	0.02	17.17	3.31

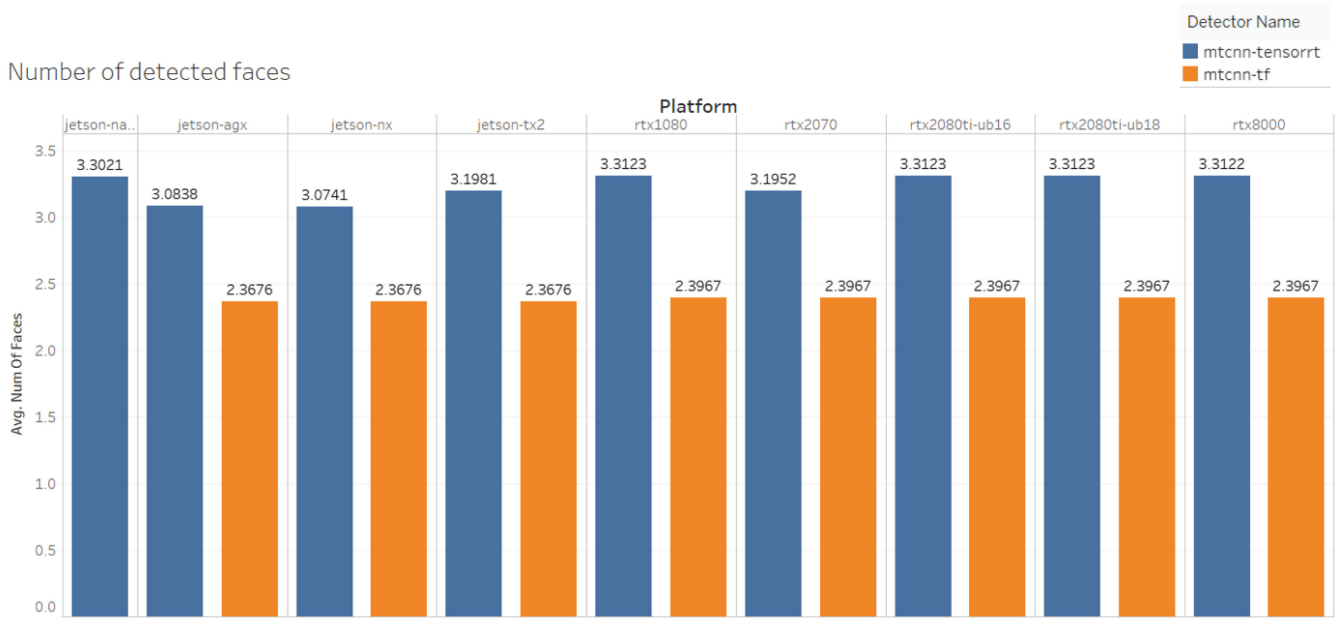

Fig. 8. Average number of detected faces per platform and detector framework.

devices show a largely reduced execution time (2x to 7.7x on average), as expected. RTX2080Ti running on Ubuntu 16.04 (Named RTX208Ti-ub16 on the figures) is consistently the fastest, followed by RTX2070 (also running on Ubuntu 16.04). Then, GTX1080 and RTX2080Ti (both running on Ubuntu 18.04) yield close performance, while RTX8000 is the slowest cloud GPU in all configurations. In fact, the main advantage of the RTX8000 is its large GPU memory (48 GB), but it is of little benefit when running the face recognition application in the inference phase where the batch size is 1.

As for the edge devices, Jetson Xavier AGX is consistently the fastest device, followed by Jetson NX, Jetson TX2. The

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

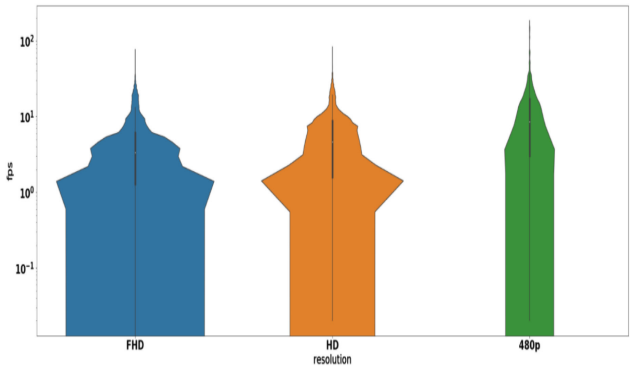

Fig. 9. Violin plot of the processing speed (in Frames per Second) for each input video resolution, with all other configurations combined.

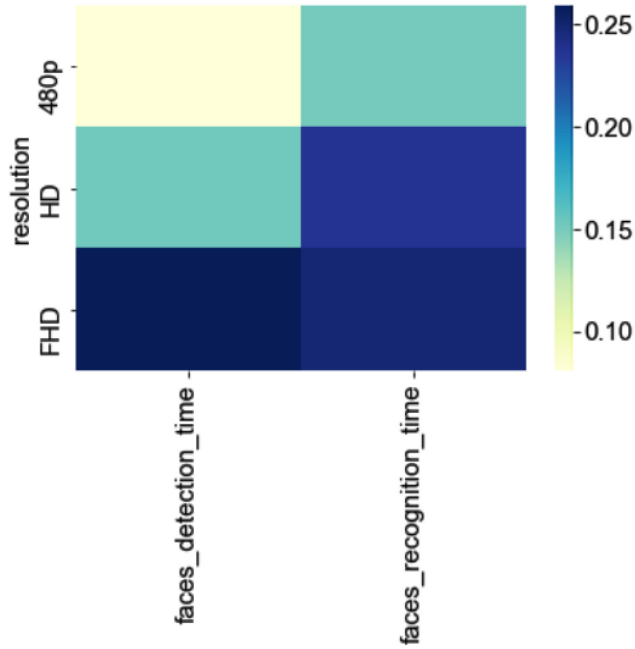

Fig. 10. Face detection and recognition time for each input video resolution, with all other configurations combined.

slowest is Jetson Nano, which cannot run the Tensorflow implementation of MTCNN due to its limited computing capabilities.

Fig. 8 reports the average number of detected faces for each platform and each detector framework. This number is almost the same across all platforms when using Tensorflow implementation, with edge devices showing only 1% lower average. Whereas it varies up to 7%, especially on edge devices, when using the platform-dependent TensorRT optimization.

B. Impact of the Tensorflow Optimization

On all tested platforms, the MTCNN detector is substantially accelerated (6x to 11x) on TensorRT compared to Tensorflow. As for FaceNet time execution, the conversion from TF to TFLite shows contrasting performance depending on the platform. In fact, while the recognition time is decreased by 24%, 18%, and 17% on Jetson Xavier AGX, RTX2070, and RTX2080Ti (running on Ubuntu 16.04) respectively, it is almost unchanged on Jetson Nano, Jetson NX, and RTX2080Ti (running on Ubuntu 18.04). It is even increased by 17%, 21%, and 66% on Jetson TX2, GTX1080, and RTX8000, respectively. This highlights the fact that TFLite applies a generic optimization that is especially aimed at certain mobile devices does not take into account the specific characteristics of each platform architecture. By contrast, The platform-dependent TensorRT optimization of Facenet consistently and significantly accelerates its execution on all tested platforms by 51% to 76% compared to the Tensorflow implementation.

The significant difference in the number of detected faces between the Tensorflow and TensorRT implementations of MTCNN (Fig. 8) is mainly due to the appearance of false positives due to the degradation in accuracy when converting to TensorRT.

C. Impact of the Resolution

Both MTCNN and Facenet are based on convolutional neural networks that are trained once and for all. At the inference phase, a forward pass of the CNN amounts to a series of convolutions and matrix multiplications. The CNN architecture is fixed except for the input layer, which depends on the input image size. The number of convolutions to be applied to the image is proportional to the number of pixels (width*height) represented by the image resolution. Fig. 9 depicts the density of the processing speed values (FPS) for each input video resolution, with all other configurations combined. Passing from 480p to HD resolution yields an important decrease in speed (2.4x on average), while the decrease is less pronounced when passing from HD to FHD (1.2x).

By breaking down this total processing speed, Fig. 10 represents the face detection and recognition times for each input video resolution and each detector framework separately. We notice that the increase is linear when passing from 480p (0.3 MPix) to HD (0.9 MPix), then to FHD (2 MPix), with a multiplying factor of 1.9 each time, except for the TensorRT implementation that only decelerates by 10% when passing from HD to FHD. This highlights the ability of TensorRT optimization to process high-resolution inputs efficiently.

On the other hand, the recognition time is multiplied by 1.5 to 1.6 when passing from 480p to HD resolution for all three implementations (TF, TFLite, TensorRT). By contrast, increasing the input resolution from HD to FHD costs only a raise between 4% and 6% in recognition time for all three implementations.

Fig. 11 shows that the input resolution also has a noticeable effect on the number of detected faces. This number increases by 35% to 83% when moving from 480p to HD resolution, whereas it rises only marginally (3% to 4%) when moving from HD to FHD. This accentuates further the increase in recognition time as a function of the input resolution.

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

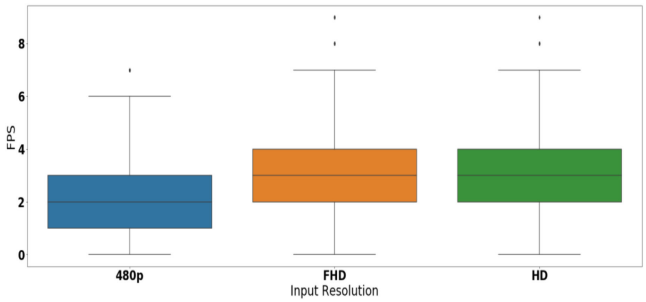

Fig. 11. Box plot of the number of detected faces for each video input resolution, with all other configurations combined.

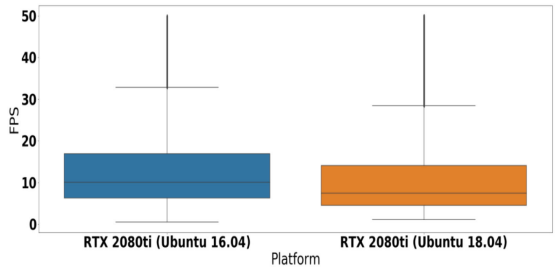

Fig. 12. Box plot comparing the FPS performance (truncated at a maximum of 50 FPS) on two cloud machines having identical hardware characteristics but running different operating system versions.

D. Impact of the Operating System

Fig. 12 compares the FPS performance on two cloud machines having identical hardware characteristics but running different operating system versions. It shows that Ubuntu 16.04 allows for faster execution on average. If we examine more closely, we notice that the TF implementation of the MTCNN detector is 53% slower on Ubuntu 18, while its TensorRT implementation is 5% faster. As for FaceNet, its TF and TFLite implementations run 24% and 56% (respectively) slower on Ubuntu 18, while its TensorRT implementation runs 8% faster. This disparity is likely due to the difference in GPU memory management between the two operating systems. In fact, we were obliged to fix a GPU memory consumption limit of 5 GB for Tensorflow on the two machines running on Ubuntu 16. Otherwise, the execution crashes. Whereas, the GPU memory consumption on the other machines running on Ubuntu 18 was much lower, so that we did not need to fix a limit for it. This difference is clearly shown in Fig. 14, where we observe that it is much more pronounced for the Tensorflow implementations of MTCNN and FaceNet.

E. Memory Consumption

Fig. 13 depicts the ratio between used and total GPU memory (see Tables III and IV), broken down by detection and recognition frameworks. It appears, as expected, that the Tensorflow implementations consume much more memory, particularly on cloud devices. On average, MTCNN-TF consumes 29% more memory than MTCNN-TRT on edge devices and 120% more on cloud devices. Besides, FaceNet-TF consumes 26% and 15% more memory on edge devices than Facenet-TFLite and Facenet-TRT, respectively, whereas these numbers rise to 80% and 67% on cloud devices. The TensorRT implementation of FaceNet runs faster than the TFLite implementations (as seen in section VI-B), at the expense of larger memory consumption (9% on average on edge devices, and 38% on cloud devices).

F. Power Consumption

Fig. 15 presents the GPU power consumption on each platform when running the face recognition application. The power consumption of edge devices ranges from 0 to 3.5 W, with similar profiles, but with a larger variance on Jetson TX2. For cloud GPUs, it ranges from 13 W to 147 W, with one extreme measure at 238 W, and with an important variability between machines.

When breaking down these results by detection and recognition frameworks (Fig. 16), it appears that for both edge and cloud devices, the TensorRT implementations of MTCNN and FaceNet consume the most energy. In contrast, the TFLite implementation of FaceNet consumes the least. Nevertheless, the most energy-efficient combination is MTCNN-TRT/FaceNet-TFLite, except on RTX2080Ti running on Ubuntu 18.04, for which the TF/TF combination is unexpectedly the most energy-efficient.

G. Optimization Using DeepStream

As a further optimization, we tested the NVIDIA DeepStream SDK [71] which is built upon the open source GStreamer [72] plugins, and provides a scalable cross-platform framework that can be deployed on edge devices. DeepStream is optimized for NVIDIA GPUs, ensures optimum memory management, allows secure bi-directional messaging between the edge and the cloud, and is especially tailored for data streaming. We integrated the TensorRT-optimized FaceNet model and a pre-built, optimized face detector (based on ResNet10 [31] from NVIDIA Transfer Learning Toolkit [73]) into the DeepStream pipeline. Then we ran them on video 1 in FHD (1920x1080) resolution (see Table V), on two different edge devices (Jetson Xavier AGX and Jetson NX). The batch size was fixed at 16 frames. Table VII shows the results in terms of frames per second (FPS) and average number of detected faces. Compared to our original TensorRT implementation of MTCNN and FaceNet on the same video and resolution, the DeepStream implementation yields an acceleration of 3.7x and 2.4x on AGX and NX, respectively, despite a slightly higher average number of detected faces (around 6% higher). This confirms the efficiency of the aforementioned DeepStream optimizations.

VII. Conclusion

This paper presented a real-world case study on deploying a real-time face recognition inference application using

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

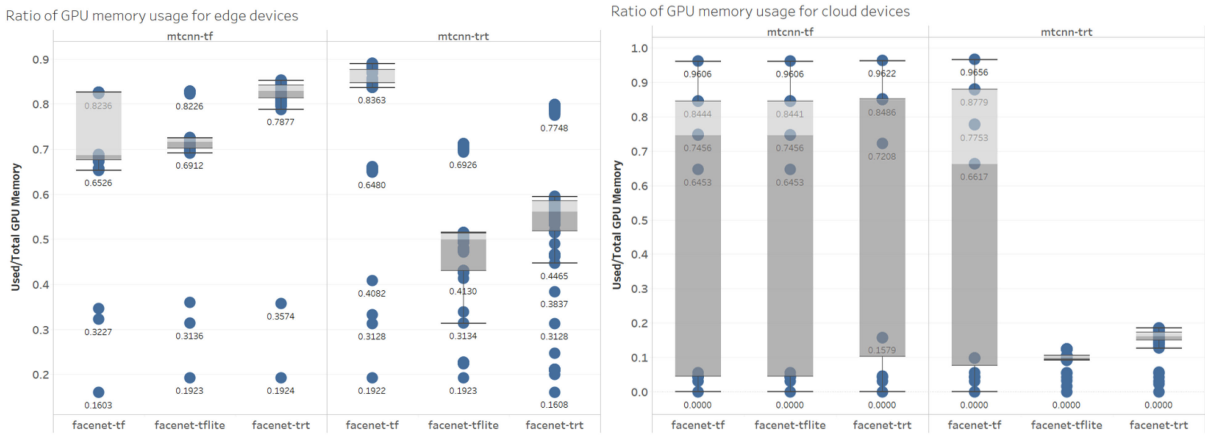

Fig. 13. Box plot of the ratio of used GPU memory, broken down by detection and recognition frameworks, for edge (left) and cloud (right) devices.

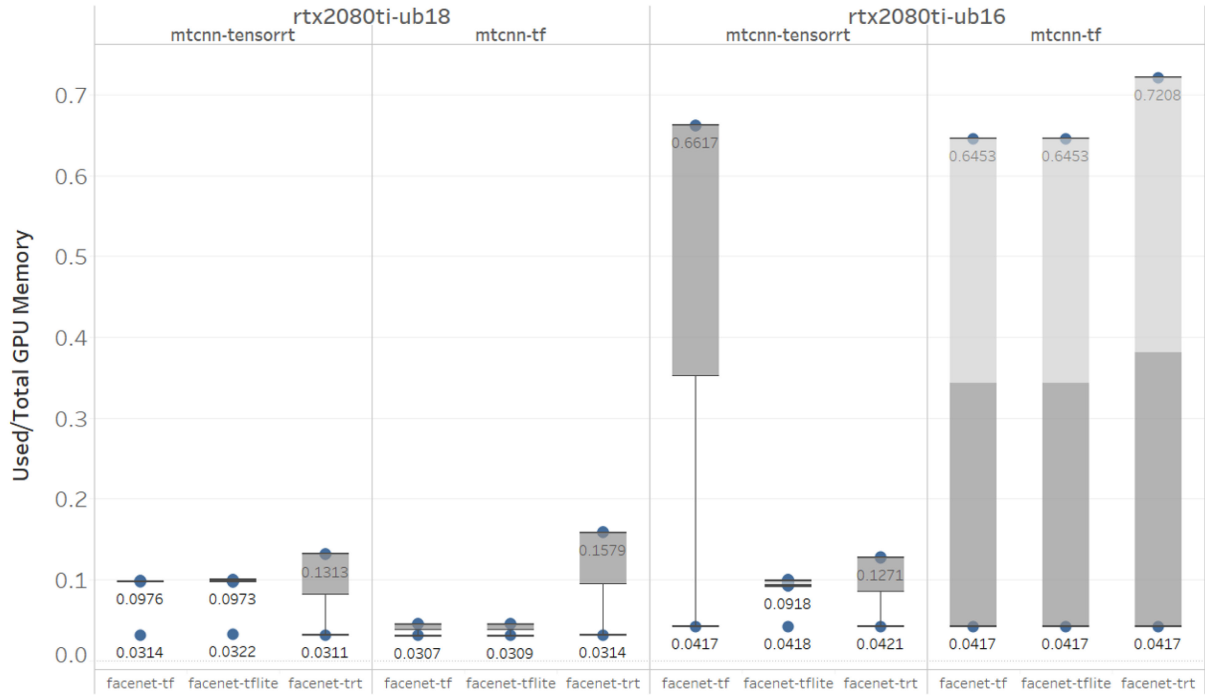

Fig. 14. Ratio of GPU memory usage for two identical machines (RTX2080Ti) running different operating systems (Ubuntu 16.04 and Ubuntu 18.04), for each detection and recognition implementation.

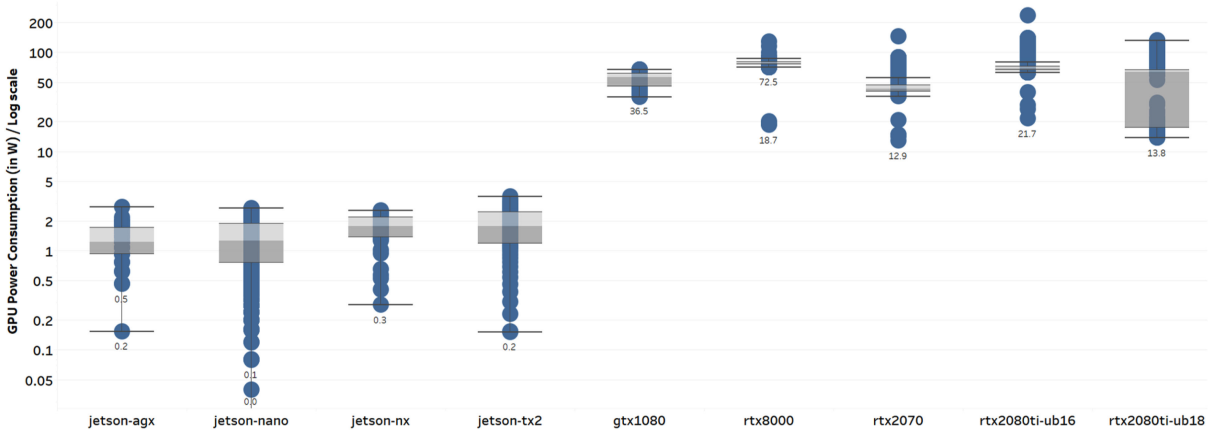

Fig. 15. Box plot of the power consumption per platform, in logarithmic scale.

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

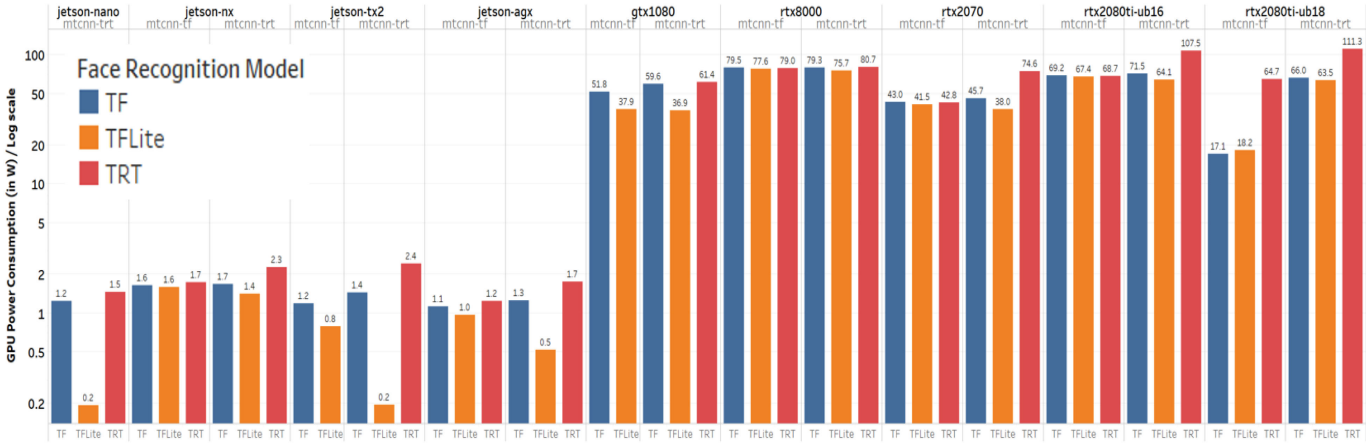

Fig. 16. Average power consumption of each platform in logarithmic scale, broken down by detection and recognition frameworks.

TABLE VII
PERFORMANCE OF THE FACE RECOGNITION APPLICATION USING DEEPSTREAM FRAMEWORK ON TWO DIFFERENT EDGE DEVICES

Platform	FPS	Average number of faces
Jetson Xavier AGX	20.00	3.40
Jetson NX	12.11	3.41

MTCNN detector and FaceNet recognizer. We evaluated the performance of different cloud-based and edge-based GPU platforms when running (1) the standard Tensorflow implementation, (2) TensorRT platform-dependent optimization, and (3) TFLite platform-independent optimization. We provided a comparative analysis of nine edge or cloud devices in terms of execution times, energy, and memory consumption. Through a series of 294 experiments, the results demonstrate that the TensorRT optimization consistently provides the fastest execution on all cloud and edge devices, at the cost of around 40% larger energy consumption. Whereas FaceNet-TFLite is the most efficient implementation in terms of memory and power consumption, at the expense of significantly less (up to -62%) processing speed (FPS) than TensorRT. Additional figures summarizing the performance of the face recognition inference application have been made available on: www.riotu-lab.org/face.

In future work, we intend to conduct a similar study on the training phase of deep learning algorithms and examine the impact of framework optimizations on the detection and recognition accuracy.

REFERENCES

[1] "MarketsandMarkets Research," 2020. Accessed: Feb. 11, 2021. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/deep-learning-market-107369271.html

[2] O. Amosov, S. Amosova, S. Zhiganov, Y. S. Ivanov, and F. Pashchenko, "Computational method for recognizing situations and objects in the frames of a continuous video stream using deep neural networks for access control systems," *J. Comput. Syst. Sci. Int.*, vol. 59, no. 5, pp. 712–727, 2020.

[3] M. Cherrington, Z. J. Lu, Q. Xu, D. Airehrour, S. Madanian, and A. Dyrkacz, "Deep learning decision support for sustainable asset management," in *Proc. Adv. Asset Manage. Condition Monit.*, Springer, 2020, pp. 537–547.

[4] M. Peppa, D. Bell, T. Komar, and W. Xiao, "Urban traffic flow analysis based on deep learning car detection from CCTV image series." *Int. Archives Photogrammetry Remote Sens. Spatial Inf. Sci.*, vol. 42, no. 4, pp. 499–506, 2018.

[5] N. T. Karim, S. Jain, J. Moonrinta, M. N. Dailey, and M. Ekpanyapong, "Customer and target individual face analysis for retail analytics," in *Proc. Int. Workshop Adv. Image Technol.*, 2018, pp. 1–4.

[6] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu, "Deep learning for smart manufacturing: Methods and applications," *J. Manuf. Syst.*, vol. 48, pp. 144–156, 2018.

[7] B. Varghesea *et al.*, "A survey on edge benchmarking," 2020, *arXiv:2004.11725v1*.

[8] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, 2019, pp. 1–16.

[9] P. Liu, B. Qi, and S. Banerjee, "EdgeEye: An edge service framework for real-time intelligent video analytics," in *Proc. 1st Int. Workshop Edge Syst., Anal. Netw.*, 2018, pp. 1–6.

[10] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," *ACM SIGPLAN Notices*, vol. 53, no. 6, pp. 31–43, 2018.

[11] Y. Huang, X. Ma, X. Fan, J. Liu, and W. Gong, "When deep learning meets edge computing," in *Proc. IEEE 25th Int. Conf. Netw. Protoc.*, 2017, pp. 1–2.

[12] A. Koubaa, A. Ammar, M. Alahdab, A. Kanhouch, and A. T. Azar, "Deepbrain: Experimental evaluation of cloud-based computation offloading and edge computing in the internet-of-drones for deep learning applications," *Sensors*, vol. 20, no. 18, 2020, Art. no. 5240.

[13] M. Alzantot, Y. Wang, Z. Ren, and M. B. Srivastava, "RStensorflow: GPU enabled tensorflow for deep learning on commodity android devices," in *Proc. 1st Int. Workshop Deep Learn. Mobile Syst. Appl.*, 2017, pp. 7–12.

[14] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tut.*, vol. 22, no. 2, pp. 869–904, Apr.-Jun. 2020.

[15] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," *ACM SIGOPS Oper. Syst. Rev.*, vol. 31, no. 5, pp. 276–287, 1997.

[16] W. Hu *et al.*, "Quantifying the impact of edge computing on mobile applications," in *Proc. 7th ACM SIGOPS Asia-Pacific Workshop Syst.*, 2016, pp. 1–8.

[17] M. Zhang *et al.*, "Deep learning in the era of edge computing: Challenges and opportunities," *Fog Comput.: Theory Pract.*, pp. 67–78, 2020.

[18] K.-H. Le Minh, K.-H. Le, and Q. Le-Trung, "DLASE: A light-weight framework supporting deep learning for edge devices," in *Proc. 4th Int. Conf. Recent Adv. Signal Process., Telecommun. & Comput.*, 2020, pp. 103–108.

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

[19] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2018, pp. 1421–1429.

[20] U. Drolia, K. Guo, and P. Narasimhan, "Precog: Prefetching for image recognition applications at the edge," in *Proc. 2nd ACM/IEEE Sympos. Edge Comput.*, 2017, pp. 1–13.

[21] M. Xu, F. Qian, M. Zhu, F. Huang, S. Pushp, and X. Liu, "Deepwear: Adaptive local offloading for on-wearable deep learning," *IEEE Trans. Mobile Comput.*, vol. 19, no. 2, pp. 314–330, Feb. 2020.

[22] X. Wang, M. Hersche, B. Tömekce, B. Kaya, M. Magno, and L. Benini, "An accurate EEGnet-based motor-imagery brain-computer interface for low-power edge computing," 2020, *arXiv:2004.00077*.

[23] N. D. Lane *et al.*, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2016, pp. 1–12.

[24] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7635–7647, Oct. 2019.

[25] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[26] S. Mittal, "A survey on optimized implementation of deep learning models on the NVIDIA jetson platform," *J. Syst. Archit.*, vol. 97, pp. 428–442, 2019.

[27] Y. Wang, T. Nakachi, and H. Ishihara, "Edge and cloud-aided secure sparse representation for face recognition," in *Proc. 27th Eur. Signal Process. Conf.*, 2019, pp. 1–5.

[28] W. Y. B. Lim *et al.*, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Sur. Tut.*, vol. 22, no. 3, pp. 2031–2063, Jul./Sep. 2020.

[29] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE Infocom.*, 2012, pp. 945–953.

[30] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[34] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Conf. Neural Inf. Proc. Syst. Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, pp. 1–9.

[35] W. Liu *et al.*, "SSD: Single Shot Multibox Detector," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 21–37.

[36] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[37] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2961–2969.

[38] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[39] G. Chevalier, "LSTMS for human activity recognition," 2016. Accessed: Feb. 11, 2021. [Online]. Available: https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition

[40] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "Deepsense: A unified deep learning framework for time-series mobile sensing data processing," in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 351–360.

[41] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning," 2016, *arXiv:1605.05101*.

[42] N. D. Lane, P. Georgiev, and L. Qendro, "DeepEar: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2015, pp. 283–294.

[43] A. v. d. Oord *et al.*, "Wavenet: A generative model for raw audio," 2016, *arXiv:1609.03499*.

[44] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7263–7271.

[45] A. Dasgupta, R. Kumar, and T. Sarlós, "Fast locality-sensitive hashing," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2011, pp. 1073–1081.

[46] S. Yu, X. Chen, L. Yang, D. Wu, M. Bennis, and J. Zhang, "Intelligent edge: Leveraging deep imitation learning for mobile edge computation offloading," *IEEE Wireless Commun.*, vol. 27, no. 1, pp. 92–99, Feb. 2020.

[47] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 815–823.

[48] NVIDIA, "Detectnet: Deep neural network for object detection indigits," 2016. Accessed: Feb. 11, 2021. [Online]. Available: https://developer.nvidia.com/blog/detectnet-deep-neural-network-object-detection-digits

[49] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.

[50] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Process. Lett.*, vol. 23, no. 10, pp. 1499–1503, Oct. 2016.

[51] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, pp. I–I, 2001.

[52] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 1701–1708.

[53] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deepface recognition," in *Proc. Brit. Mach. Vis. Conf.*, M. W. J. Xianghua Xie and G. K. L. Tam, Eds. BMVA Press, pp. 41.1–41.12, Sep. 2015. [Online]. Available: https://dx.doi.org/10.5244/C.29.41

[54] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "VGGface2: A dataset for recognising faces across pose and age," in *Proc. 13th IEEE Int. Conf. Autom. Face Gesture Recognit.*, 2018, pp. 67–74.

[55] T. Baltrušaitis, P. Robinson, and L. Morency, "Openface: An open source facial behavior analysis toolkit," in *Proc. IEEE Winter Conf. Appl. Comput. Vis.*, 2016, pp. 1–10.

[56] T. Baltrusaitis, A. Zadeh, Y. C. Lim, and L. Morency, "Openface 2.0: Facial behavior analysis toolkit," in *Proc. 13th IEEE Int. Conf. Autom. Face Gesture Recognit.*, 2018, pp. 59–66.

[57] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *Proc. Int. Conf. Mach. Learn. Deep Learn. Workshop*, vol. 2, 2015, pp. 1–8.

[58] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.

[59] Q. Liu, P. Li, W. Zhao, W. Cai, S. Yu, and V. C. M. Leung, "A survey on security threats and defensive techniques of machine learning: A data driven view," *IEEE Access*, vol. 6, pp. 12 103–12 117, 2018.

[60] M. Xue, C. Yuan, H. Wu, Y. Zhang, and W. Liu, "Machine learning security: Threats, countermeasures, and evaluations," *IEEE Access*, vol. 8, pp. 74 720–74 742, 2020.

[61] A. Qayyum, M. Usama, J. Qadir, and A. Al-Fuqaha, "Securing connected autonomous vehicles: Challenges posed by adversarial machine learning and the way forward," *IEEE Commun. Surveys Tut.*, vol. 22, no. 2, pp. 998–1026, Apr.–Jun. 2020.

[62] L. Lyu *et al.*, "Towards fair and privacy-preserving federated deep models," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 11, pp. 2524–2541, Nov. 2020.

[63] X. Zhang, X. Chen, J. K. Liu, and Y. Xiang, "Deeppar and deepdpa: Privacy preserving and asynchronous deep learning for industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 16, no. 3, pp. 2081–2090, Mar. 2020.

[64] O. Kwabena, Z. Qin, T. Zhuang, and Z. Qin, "Mscryptonet: Multi-scheme privacy-preserving deep learning in cloud computing," *IEEE Access*, vol. 7, pp. 29 344–29 354, 2019.

[65] L. Zhao, Q. Wang, Q. Zou, Y. Zhang, and Y. Chen, "Privacy-preserving collaborative deep learning with unreliable participants," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 1486–1500, 2020.

[66] "Face recognition using tensorflow implementation by David Sandberg," 2020. Accessed: Feb. 11, 2021. [Online]. Available: https://github.com/ipazc/mtcnn

[67] "Convert FaceNet From Keras to TFLite," 2020. Accessed: Feb. 11, 2021. [Online]. Available: https://github.com/jkjung-avt/tensorrt_demos

[68] "Face recognition using tensorflow implementation by David Sandberg," 2020. Accessed: Feb. 11, 2021. [Online]. Available: https://github.com/davidsandberg/facenet

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.

[69] "RIOTU Lab, Conversion from tensorflow to TRT using ONNX," 2020. Accessed: Feb. 11, 2021. [Online]. Available: https://github.com/riotu-lab/tf2trt_with_onnx
[70] "Convert FaceNet from keras to TFLite," 2020. Accessed: Feb. 11, 2021. [Online]. Available: https://colab.research.google.com/drive/1ts-KFwU0eBs9jaXiM1U2GytEI8j7Rbh6?usp=sharing
[71] "NVIDIA DeepStream SDK," 2020. Accessed: Feb. 11, 2021. [Online]. Available: https://developer.nvidia.com/deepstream-sdk
[72] S. D. Burks and J. M. Doe, "Gstreamer as a framework for image processing applications in image fusion," in *Proc. SPIE Multisensor, Multisource Inf. Fusion: Architectures, Algorithms, Appl.*, vol. 8064. 2011, p. 80640M, doi: 10.1117/12.884797.
[73] "NVIDIA transfer learning toolkit," 2020. Accessed: Feb. 11, 2021. [Online]. Available: https://developer.nvidia.com/transfer-learning-toolkit

Anis Koubaa is currently the Director with the Research and Initiatives Center and the Leader with the Robotics and Internet of Things Lab, Prince Sultan University, Riyadh, Saudi Arabia. He is currently a Full Professor of computer science and has been working in several R&D projects on data science and deep learning, including face recognition, vehicle identification, and object classification and detection. He is an ACM Distinguished Speaker and a Senior Fellow of the Higher Education Academy of the U.K. He presented several training programs on data science, Python programming, Tableau for data science, deep learning, and several other technologies.

Adel Ammar received the Engineering degree from the Ecole des Mines de Nancy, Nancy, France, in 2003, the master's degree from the University of Versailles St-Quentin-en-Yvelines, Versailles, France, in 2005, and the Ph.D. degree in applied data processing, from the Université Paul Sabatier, Toulouse, France, in 2008. He is currently an Associate Professor with the College of Computer Science and Information Systems, and a Researcher with the Robotics and Internet of Things Laboratory, Prince Sultan University, Riyadh, Saudi Arabia. His research interests include machine learning, deep learning, pattern recognition, and image processing.

Anas Kanhouch received the B.S. degree in software engineering in 2019. He is currently a Research Engineer with the Robotics and Internet of Things Lab, Prince Sultan University, Riyadh, Saudi Arabia. He has more than three years of experience in web development and has been working on several R&D projects on deep learning including face recognition and vehicle identification.

Yasser Alhabashi received the bachelor of science degree in computer science from Prince Sultan University, Riyadh, Saudi Arabia, in 2017. He is currently a Research Engineer with Robotics and Internet of Things Lab, Prince Sultan University. He has been working on several R&D projects on data science and deep learning, including face recognition, and vehicle identification.

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on March 22,2025 at 14:09:48 UTC from IEEE Xplore. Restrictions apply.