Exploring Smartphone-Based Edge Computing for AI Inference using Real Testbeds

March 7, 2025

Abstract

Gradually, AI is becoming ubiquitous at the network edge. The wide availability of pre-trained models and AI execution frameworks contribute to such ubiquity. Particularly, deep learning (DL) models such as convolutional neural networks (CNN) are extensively used in computer vision (CV) for performing object recognition and image classification tasks. Solutions in various domains ranging from security surveillance to e-health exploit computer vision tasks. In many cases, such solutions must deliver realtime inferences; some progress has been made in this respect. Some approaches show a strong dependency on Cloud resources as complements to the computing power that resource limited edge nodes can offer. Other solutions distribute workload horizontally, i.e., by harnessing the power of several edge nodes within the same LAN or WLAN. Many of these efforts experiment with real settings comprising SBC-like (Single Board Computers) edge nodes only such as NVidia Jetson Nano and Raspberry Pi; few of these consider nomadic hardware such as smartphones as edge nodes. Given the ubiquity of smartphones worldwide and the unlimited scenarios where clusters of these nodes can be exploited for providing substantial computing power, in this work, we shed some light in answering the following question: Is smartphone-based edge computing a competitive resource provider for realtime AI inferences? We evaluate the potential of smartphone clusters for making realtime inferences associated to CV tasks based on intuitive image/video partitioning schemes. In our experiments we use three pre-trained DL models that present different computing requirements and eight heterogeneous edge nodes including five low/mid-end smartphones and three SBCs. We compared the performance achieved using workloads from three image stream processing scenarios with different characteristics. Experiments were run with the help of a tool set designed for reproducing battery-driven edge computing tests. We compared latency and energy efficiency achieved by using either several smartphone clusters testbeds or SBCs only. Additionally, for battery-driven settings, we include metrics to measure how workload execution impact smartphone battery levels. As per the computing capability shown in our experiments, we conclude that edge computing based on smartphone clusters can help in providing valuable computing resources to enable the expansion of AI in application scenarios requiring realtime responses.

1 Introduction

The application of deep learning (DL) to perform on-site tasks is rapidly expanding. In the healthcare domain, DL is used for predicting, detecting, diagnosing, and prognosing different diseases [35, 31]. In food production, particularly, animal farming [8] these tasks include animal behavior recognition, growth evaluation, and individual identification. In the smart city domain [45, 28], tasks include intelligent transportation, waste management, crime prediction, public infrastructure and green spaces maintenance. Many of these tasks involve object classification and recognition using images as input. The application of CNNs (Convolutional Neural Networks) has become popular to perform these tasks due to the training procedure requiring minimal human intervention compared to other machine learning techniques. However, training a deep neural network model requires big datasets and powerful hardware. Nowadays, data gathering leveraged by mobile and IoT devices, and computing power offered by cloud computing platforms, greatly speeds up such training process compared to a couple of decades ago. Cloud computing is considered not only in the training phase of DL models but also for making inferences. It implies that a trained model deployed in a remote computing infrastructure is evaluated with locally-generated input data, and produces output results that must be transferred back traversing long-latency Internet uplinks and downlinks.

Distributed inference type	Associated works
Vertical	[29, 26, 25, 13, 41]
Horizontal	[17, 23, 40, 50, 24]

Table 1: Approaches for distributed inference

This latency has been been identified a limitation for time-sensitive applications. To overcome this issue and others such as unavailability of the Internet connection, network overloading and privacy protection, edge-based architectures have been proposed [49, 36].

There are differences highlighted in the literature when referring to edge-based architectures [49], some of these being simply terminological, and some others describing substantial differences in the way their constituent components are defined and interoperate with components from other computing layers [38]. Points in common, however, are minimizing data transportation and bringing computing resources close to where computing needs and data originate. Heterogeneity and energy constraints are common features of computing infrastructures operating in cyber-physical and IoT systems. For these reasons, cooperation is a key concept of *Dew Computing* [18], which is among these edge-based architectures and endows low processing power and consumer devices with the role of main computing resource providers.

In this work, we focus on evaluating collaboration among consumer devices to perform cloud-agnostic computing services for realtime execution of computer vision tasks using DL. The contributions of this work can be summarized as follows:

- We applied an intuitive data partition scheme to create and distribute CV tasks among clusters of smartphones for performing real-time inferences over data streams workloads with different characteristics.
- We evaluated and compared using real testbeds the performance and energy consumption of smartphone clusters against typical edge nodes including SBCs, explaining some implications of the results obtained.
- The evaluated scenarios serve as a guide and example on how to use previously published tools especially designed to facilitate the experimenation with real testbeds comprising smartphone clusters.

This work is organized as follows. Section 2 presents and classifies recent efforts backed by real testbeds results to speed up inferences at the edge using different approaches for distributed execution. Section 3 describes the mobile distributed computing model and infrastructure-related architectural assumptions we made in our exploration tests, as well as details concerning DL models used and data stream workloads. In Section 3.1 we explain details about node setups and metrics, results obtained in our empirical evaluation and a discussion. Lastly, Section 6 presents the conclusions and future research directions.

2 Related Works

In this section we analyze different relevant works in line with the broad concept of *distributed inference*, which is present in recent research related to AI at the network edge, and we define and explore in the rest of the Section. In the context of AI, an inference means asking a model to compute a result based on input data (e.g., given an image, list the objects detected). All mentioned works include experimentation with real testbeds.

When computing infrastructures of at least two nodes perform inference tasks, it is said that distributed inference is applied. The scheme under which such a distribution is given can be vertical [29, 13, 41] or horizontal [17, 23, 40, 50, 24]. Under the vertical scheme, collaborating nodes typically belong to different computing layers of the edge architecture. These layers are organized in a hierarchical structure where nodes in a layer render services to those in the layer below using services provided by nodes in the layer above. Nodes at the base of such hierarchy commonly have much less powerful computing capabilities and use the services of layers above to complete computationally intensive tasks. On the other side, nodes at the top layer present more powerful computing capabilities, like Fog or Cloud resources. However, reaching

such nodes incur in long-latency communication when transferring task input/output data. For this reason, its usage should be carefully analyzed especially for applications with delay-sensitive time constrains. A classical example of this type of collaboration is given when a resource-limited user device leverages the services of an Edge server or Edge layer to partially o completely compute the result of inference tasks. Since the Edge layer can render computing services to a variable number of users in a locality, depending on the burden at the time of receiving tasks, the Edge layer can have enough resources to complete or partially perform a task. In the later case, the Edge layer would eventually leverage the services of a remote cloud datacenter to fullfill the request while meeting potential task deadlines. Like in [13, 29], offloading decisions are evaluated considering communication throughput between computing layers. Besides, a task requiring the execution of a DL model is split into several parts. In this splitting process authors propose to consider insights of data anatomy and computing requirements of layers characterizing different DL models. Partitions of model's graph are dynamically assigned and executed between a device and remote servers. Other works adopting this distribution scheme are the ones by [25, 26, 41]. Several works evidence that vertical collaboration is feasible to allow resource-limited devices to execute compute-intensive tasks, such as CV tasks using large DL models. However, a major drawback relates to the long-latency communication dependency when offloading tasks to upper resource layers that provide the required computing capability to complete the tasks.

Another inference distribution scheme is when nodes within the same layer collaborate in the execution of compute intensive tasks. This scheme is known as horizontal collaboration. A key difference with vertical collaboration is that participating nodes and data rarely occurs outside the scope of a LAN or WLAN. Even though this form of collaboration governs the internal communication of nodes within the same layer, in this paper, we put special focus on the horizontal communication of resource-constrained nodes including edge and user device nodes. In the literature, we have identified several works proposing distributed inference under this form of collaboration [17, 23, 40, 50, 24] that we further analyzed taking into account several dimensions. Namely, the type of parallelism proposed, i.e., whether it is by intervening the anatomy of a DL model, partitioning the data input or adopting a hybrid approach by combining the previous techniques. Other dimension of analysis relates to the methodology used to evaluate the proposal. We selected works whose methodology involves setting up real testbeds and describe the hardware used and the way resource heterogeneity is present in the experiments. Moreover, providing that QoS is a concept that relates to adapting to dynamic changes in the workload and/or available resources, we indicate the support of these works in this respect. Finally, we contextualize the strategy of each work for resource management mentioning the main motivation behind the proposal.

Works can be also classified according to how workload execution is parallelized. We found three approaches of parallelism named model partitioning, data partitioning, and hybrid. Within the former group, layers composing a DL model are deployed into different devices so that outputs computed by a device are the inputs for another one. With this type of parallelism, all participating devices are involved in computing inferences for all portions of the input. By contrast, when using data partitioning paralellism, edge nodes run the whole model over portions of the input. A third type, which we call hybrid, parallelizes workload by combining the data and model partitioning schemes. In all cases, results for the whole input are obtained by joining partial results calculated by all participating devices.

When making several devices to participate in inferences, two questions related to devices emerge: Does the approach consider executing inferences on heterogeneous devices? If so, what does the approach do to deal with the performance differences from executing the same workload on devices with different computing capabilities in order to assure certain QoS levels?

In [17], authors propose AutoDiCE, a tool for partitioning, deploying, and distributedly executing a large CNN model on multiple resource-constrained edge devices. The platform specification and mapping specification allow practitioners to indicate which submodel runs on which edge devices. Communication between sub-models and resources usage is implemented via the well-known parallel programming standards MPI and OpenMP. Although the tool was designed to support horizontally and vertically distributed inferences, it was tested using the former scheme, using an infrastructure comprising homogeneous edge nodes communicated via a gigabit Ethernet switches. DeColla [23] utilizes Deep Reinforcement Learning (DRL)-based adaptive allocation for horizontal collaborative inference. All DL layers are present in all collaborating IoT devices. Neurons calculation of each layer occurs in parallel, coordinated via a message queue and orchestrated by DeColla engine that operates in a requester IoT device. Number of IoT devices,

Parallelism type	Experiments		Online adaptive QoS support?	Main motivation	Work
	Resource heterogeneity type	Type			
Model Partitioning	Homogeneous nodes; varying resource type and quantity –CPU Cores and GPU–	Real hardware: LAN with up to eight edge devices (Nvidia Jetson Xavier NX)	No (once generated and deployed, mappings between sub-model and edge nodes are fixed at runtime)	Automate large CNN model partitioning, deployment and execution on multiple resource-constrained edge devices	[17]
	Homogeneus nodes; varying CPU usage and network bandwidth	Real Hardware: LAN with three IoT devices (Raspberry Pi 4)	Yes (uses a Deep Reinforcement Learning-based adaptive allocator and a fault tolerant mechanism implemented using a message queue)	Accelerate performance with an intra-layer parallelism instead of layers sequential execution as in traditional hierarchical inference	[23]
Data partitioning	Heterogeneous nodes (IoT devices)	Real Hardware: LAN with four IoT devices (2x Odroid XU4, Jetson Nano, Rapsberry Pi 4)	Yes (accuracy-performance-heterogeneity tradeoffs exploitation)	Exploit the concept of approximate computing to ensure performance and accuracy constraints considering heterogeneity of the edge cluster and nodes availability	[40]
	Heterogeneous nodes (Smartphones and IoT devices)	Real Hardware: WLAN of different smartphones clusters size. IoT devices operating in isolated mode	Yes (pull-based scheme using a shared workload queue)	Compare inference delivery capabilities and energy efficiency of IoT devices versus clusters of smartphones	This work
Hybrid	Homogeneous nodes; varying number of available devices	Real Hardware: WLAN with up to six IoT devices (Raspberry Pi 3)	Yes (work stealing)	Allow large DL models to execute in resource limited devices achieving low memory footprint and execution latency	[50]
	Heterogeneous nodes (smartphones)	ion: with data profiled from hardware running real workloads Real Hardware: some real experiments	Yes (uses Deep Reinforcement Learning)	Reduce Mobile Edge Computing server's computing pressure	[24]

Table 2: Approaches for horizontal distributed inference at the Edge

available computing resources, and the network state are evaluated online upon each neuron assignment.

In [40] the most appropriate version among different pruned models is selected online via an adaptive workload distribution to meet the required application specific accuracy and performance level. Such dynamic selection is performed using a base knowledge of accuracy-performance tradeoffs profiled from heterogeneous edge nodes, together with a runtime monitoring system for resource management. In [50] the DeepThings framework for executing large CNN models in resource-constrained devices is proposed. The framework adopts a hybrid parallelization approach due to it combines model and data partitioning schemes. On one side, as an offline step the framework fused convolutional and pooling layers to create independent and distributable execution units, which leads to reduced memory footprint and allow resource limited devices to contribute with the computation of partial results that are combined to produce the final result of an inference task. Besides, workload represented by a single frame is divided into overlapping and distributable regions that are computed collaboratively by a group of edge devices. With this scheme, inference capabilities differences of participating devices are handled with two strategies. One of these is static and represented by the fact that memory requirements of fused layers executed by a device adapts to its available memory resources. The other strategy is that data regions are dynamically distributed according to devices computing capabilities by following a work stealing scheme. Another work adopting hybrid paralelization is eDDNN [24], which leverages cross-platform Web technologies and WebRTC protocol to allow for distributed inference among end devices. The collaboration is mediated by an edge server that keeps track of nodes and tasks status information. Tasks, determined by images input size, and the DL model to be used, are divided into sub-images and sub-models by a decision maker component that coordinates different pieces execution among end devices using a shared dependency table and realtime nodes information. Edge data duplication is present at the step of dividing an image into sub-images. When distributing a convolutional operation by dividing the input and the model, duplication is required for ensuring results correctness.

In the discussed works, horizontal distributed inference considering the collaboration of battery-operated end-user devices, such as smartphones, combined with data partitioning schemes are scarce, meaning that the benefits and limitations of performing realtime CV tasks –and possibly other AI tasks as well– are yet unknown. Our work sheds light on providing empirical evidence for quantifying the computing power of smartphone clusters. Existing approaches exploiting data partitioning schemes add redundant data to the original input for performing distributed inference. We propose a simple yet effective data partitioning scheme for performing realtime CV tasks, which is evaluated under varying settings of real smartphones cooperating in WLAN clusters. Since performance may vary according to how workload is distributed, we provide insights on the performance achieved and energy-related measurements using different state-of-the-art online heuristics and smartphone clusters. Moreover, performance comparisons of different <smartphone clusters, load balancing heuristics> combinations were compared against the performance achieved by different edge nodes commonly used in IoT applications. All experiments performed in this work demonstrate the viability of constructing a platform for in-vivo experiments whose main components has been previously published, validated and made publicly available for use [34, 47, 42].

3 System Model and Assumptions

We envision a system that opportunistically scavenges computing cycles of nomadic hardware, such as smartphones, to complement – or eventually replace – the computing cycles delivered by low power edge nodes. Particularly, we shed light on the computing capability of different smartphone clusters and load balancing heuristics for processing streams at the Edge. In this work, we consider streams of images analyzed using CV tasks in realtime or near realtime by harnessing deep neural networks. Definition of appropriate incentives for smartphone users – tasks executors –, privacy preservation and security mechanisms for tasks submitters as for tasks executors, are outside the scope of the current work [30, 21, 46].

We assume a master-worker architecture operating within the scope of WLAN, with a master node officiating as tasks submitter and worker nodes contributing with computing resources, i.e., playing the role of tasks executors. A master node is assigned the roles of data concentrator and tasks coordinator. Playing the first role implies that the node exposes end points for receiving raw data and preprocessed data. On one side, raw data transferred to a master node is captured by devices with sensing capabilities, e.g.,

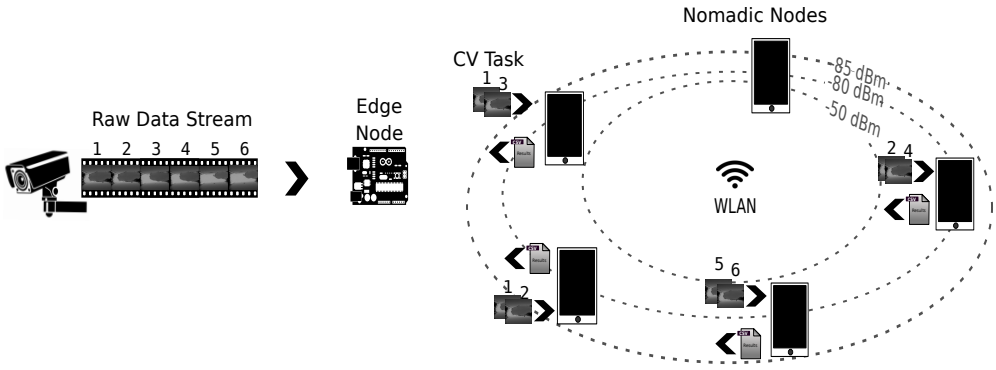


Figure 1: Main nodes schematic overview

RGB, IR or depth images taken with different camera devices. On the other side, pre-processed data is obtained by applying a series of computational operations to decode, synthesize and/or represent raw data in a format that can be more efficiently stored, retrieved or analyzed. Moreover, playing the role of tasks coordinator means that the node runs logic to decide how to distribute the load involved in preprocessing raw data by available worker nodes to meet the expected QoS. Finally, worker nodes opportunistically connected to the WLAN register through a master node service to contribute with computing resources and wait for tasks assigned by the master node. As tasks completion occurs at the worker nodes, results are sent back to the master node.

With this architecture, we aim at measuring the feasibility of using different smartphones groups cooperating under the same WLAN for performing CV tasks, and compare the performance achieved with that of SBC-like edge nodes commonly used for servicing IoT and edge applications.

3.1 Empirical Evaluation

To conduct this study we employed different hardware as edge nodes, whose characteristics are shown in Table 3.

All nodes have multicore processors and all nodes but the Nvidia Jetson Nano have 802.11 WiFi radio, with a theoretically bandwidth capacity between hundreds of Mbps (norm n) to a few Gbps (norm ac). In fact, SBC nodes WiFi was not used because, in our experiments, these nodes play all the roles – tasks submitter, tasks coordinator and tasks executor – simultaneously and no data transfers were involved. Edge nodes main memory ranges from 2 to 8 GB. In our setups, SBC operate plugged to the electricity power grid, while smartphones operate using their battery as main power source. Additionally we included an estimated per unit acquisition cost in USD. When searching for cost-effective configurations, the cost can be used as a deciding factor in situations where several nodes combinations achieved the desired performance.

Section 3.2 elaborates on the deep learning models characteristics considered in the experiments. In section 3.3 we describe three study cases belonging to different application domains, how image streams are produced, and task characteristics. Finally, Section 4.1 presents the comparison results based on the designed benchmarks, which we explain in the following sections.

3.2 DL Models Benchmarking

Benchmarking is a relevant task to approximate nodes performance in solving a specific task and this procedure was done for nodes in Table 3, where task refers to make inferences using different Tensorflow DL models. Tensorflow is a popular library for creating, training, evaluating and executing deep learning models. The Tensorflow project [5] offers a benchmarking tool to collect statistical information of a `.tflite` model, one of the file formats for created models. The tool measures the time a model takes to produce an inference, memory consumed, among other information using randomly-generated synthetic model input. It is worth mentioning that all inferences times reported in this paper are on the basis of using CPU support. Even though today there is specialized hardware –sometimes referred as AI hardware accelerators – to run

Edge Node Type	Edge Node Name	Short name	Processor	RAM	Conectivity	Battery Capacity (mAh)	Avg. Price (USD)
SBC	Raspberry Pi 4	RPi4	Quad-core 1.5 GHz ARM Cortex-A72	4 GB	Wi-Fi 802.11 ac	N/A	55
	Nvidia Jetson Nano	NvJN	Quad-core ARM Cortex-A57 MPCore	4 GB	Gigabit Ethernet, M.2 Key E	N/A	99
	Gigabyte Brix GB-BRi5H-8250	GBx	Quad core i5-8250U	8 GB	Wi-Fi 802.11 ac	N/A	500
Smartphone	Samsung A02	SamA02	Quad-core 1.5 GHz Cortex-A53	2 GB	Wi-Fi 802.11 b/g/n	4900	116
	Motorola Moto G6	MotG6	Octa-core 1.8 GHz Cortex-A53	3 GB	Wi-Fi 802.11 a/b/g/n	3000	160
	Samsung A30	SamA30	Octa-core (2x1.8 GHz Cortex-A73 & 6x1.6 GHz Cortex-A53)	3 GB	Wi-Fi 802.11 a/b/g/n/ac	3900	170
	Xiaomi Redmi Note 7	XRN7	Octa-core (4x2.2 GHz Kryo 260 Gold & 4x1.8 GHz Kryo 260 Silver)	4 GB	Wi-Fi 802.11 a/b/g/n/ac	4000	200
	Motorola Moto G9 Play	MotG9	Octa-core (4x2.0 GHz Kryo 260 Gold & 4x1.8 GHz Kryo 260 Silver)	4 GB	Wi-Fi 802.11 a/b/g/n/ac	5000	200

Table 3: Edge computing nodes: hardware features

AI logic faster than with general purpose hardware like CPUs, there are some practical limitations including library support for different platforms and proprietary chips embedded only in high-end smartphones of certain brands, which still prevent the massive use of mobile devices for distributed computing as we envision.

Moreover, in our study, the notion of *realtime* is context-dependent. Depending on the application a constant delay of a few seconds might satisfy the notion of realtime. For this reason and with the aim of evaluating smartphone based settings in different application scenarios, i.e., with different realtime requirements, we used three DL models that present quite dissimilar inference times. It is not part of the evaluation core to test DL models in their most efficient version, i.e., we do not aim to test and improve models performance per se. Conversely, our objective is to compare the performance achieved with edge devices and smartphone based opportunistic clusters.

All models used in the tests are pre-trained object recognition models that take images as input and produce text as output, where text refers to bounding boxes, classes and confidence levels of recognized objects, or categories of a unique object depending on the purpose the model was trained for. To test different mobile distributed computing scenarios, we intentionally selected models that, when used to make inferences, present dissimilar computing times. In this sense, one of the models is YoloV4 [6] used to perform realtime object detection. Figure 3b shows average inference times (in milliseconds) for different nodes when performing object detection using a YoloV4-tiny model, which is the version that can be run on mobile devices using Tensorflow. We see, for instance, that a smartphone Xiaomi RN7 makes inferences in around 248 milliseconds, i.e., is able to process up to 4 FPS, doubling the capability of a RaspberryPi4 – but it is slower than a Gigabyte Brix which reaches nearly 10 FPS. Another of the benchmarked models encapsulate expert knowledge of what is known as Body Condition Score (BCS) [7] and was proposed to score dairy cows to differentiate healthy and non-healthy cows. The model uses SqueezeNet as base neural network. Figure 2b shows the BCS average inference time for different edge nodes. For instance, the Xiaomi RN7 smartphone completes an inference in around 184 milliseconds, i.e. it is able to deliver barely more than 5 FPS, while a Gigabyte Brix node is around 15 FPS. The third model used is based on an EfficientNetB4 and was trained to recognize progression of diabetes using foot images as input [9]. Figure 4b shows the inferences times obtained by different edge nodes. By comparing the fastest smartphone with the fastest SBC it can be noted that Xiaomi RN7 completes an inference in around 1470 milliseconds while the Gigabyte Brix does the same in approximately 559 milliseconds. When comparing performance of DL models, notice that for most of the edge nodes benchmarks, inference times differ up to one order of magnitude. As said, the Xiaomi RN7 completes inferences in 184, 248 and 1470 milliseconds for the BCS model, the YoloV4 model and the diabetes foot model respectively.

3.3 Workload Characterization

In this section we describe the workloads associated to the machine vision tasks that were performed over stream-like inputs. Indeed, machine vision tasks such as object detection and classification using CNNs are commonly applied in diverse domains. To complete the experimental setting, we consider video frame streams of different length and distinct frame rates, which serve as input to the CNN models described above and in conjunction both elements shape heterogeneous workloads to the mobile distributed computing setups under evaluation.

Without losing generality, we associate each model-stream pair different domain/usage contexts. These domains are ”a dairy farm application scenario called “Body Condition Score” (BCS), a smart city application scenario that we call “Sense while travel (SWT), and a human healthcare monitoring application scenario called “Disease Monitoring and Early Diagnosis” (DMED). Next we describe the dynamics of each scenario that give sense to the stream-like input assumed:

- Body Condition Score (BCS): in a dairy farm, outfitted cows – i.e., cows that are either overweighted or underweighted – tend to produce less milk and might breed less frequently than those properly weighted. Identifying such animals so as to give them a proper treatment is crucial to maintain the cow roundup fully productive. To detect such animals, a CNN model is used, which takes depth images taken from the top with a camera strategically positioned as cows walk to a milking parlor [7]. The image capturing procedure is performed within a time window that does not exceed 10 seconds.

During such time, providing that animals naturally do not state quite under the camera and the capture should contain a specific part of the cow for the model to produce accurate results, a high percent of captured images, say 50%, is dropped, i.e., tagged and filtered, as these images do not represent useful input for the CNN model. Moreover, during the transition of one animal to the next under the lens of camera, which can take 10 seconds in average, it is reasonable to stop capturing images. To make the body condition score calculus feasible, reliable, and energy efficient, we should count with no less than a hundred of frames ready to serve as input for the CNN model, from each cow. All these constraints configure a stream processing scenario where depth images can be produced at a rate of 15 FPS during 10 seconds, followed by other 10 seconds without image captures. We stated that an atomic CV task would be represented by the burden of applying the CNN model to fifteen consecutive frames, meaning that a CV task is created every second which gives a total of ten CV tasks created for a cow. Considering that the dataset used to recreate this scenario includes 1 740 images, the resulting workload stream comprises 116 CV tasks given within a time window of 3 minutes and 34 seconds.

- Sense while Travel (SWT): a city can take advantage of vehicles such as urban line buses to collect information of relevant street events while traveling, e.g., for statistical purposes, crime detection, and infrastructure maintenance planning. Images captured with a bus front camera might feed a YOLO model, which instead gives a plain text representation of objects present in an image and the accuracy percentage of each detected object class as output. An energy efficient way to achieve this without having to process a large amount of frames with redundant information, is to cleverly select a sample rate accordingly with the expected moving change of detection target. We assume this sample rate in 2 FPS and evaluate the system performance for a stream duration of 30 minutes. These parameters means sensing data for 15 km assuming a vehicle traveling the road at 30 km/h. A CV task would be represented by the execution of the CNN model to two consecutive filtered frames, meaning that a CV task is created every second. Providing that the dataset used to recreate this scenario includes 3 600 images, the resulting workload stream comprises 1800 CV tasks given within a time window of 30 minutes.

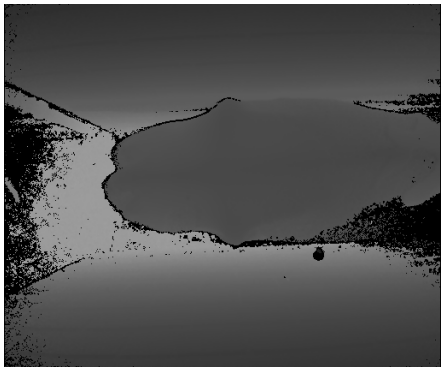
- Disease Monitoring and Early Diagnosis (DMED): We envision a remote monitoring mobile health scenario where a CNN model encapsulates expert knowledge on different pathologies to allow people, specially older people, receiving intensive care services, and being measured to analyze different health parameters as a way to prevent the appearance of some pathology or monitoring a pre-existent illnesses. We assume an scenario where a caregiver or nurse rendering services in an elderly residence is assigned with the task of taking pictures of all patients foot tissue resulting in a data stream of 10 FPS during 2 seconds, with pauses of 7 seconds without capturing images. In this case, a CV task executes the model described in [9] to 2 consecutive frames, meaning that during image capturing time, five CV tasks are created per second. Providing that the dataset used to recreate this scenario includes 600 images, the resulting workload stream comprises 300 CV tasks given within a time window of 8 minutes and 55 seconds.

Figures 2c and 2a show a timeline representation of the data input stream shape, i.e., the kilobytes of data transferred per second, and an example of the input for the CNN used in the Body Condition Score scenario. Similar information can be found in Figures 3c, 3a and Figures 4c, 4a for SWT and DMED scenarios respectively. In all cases, we depict the first four minutes of the three image streams, and we report per-image inference times.

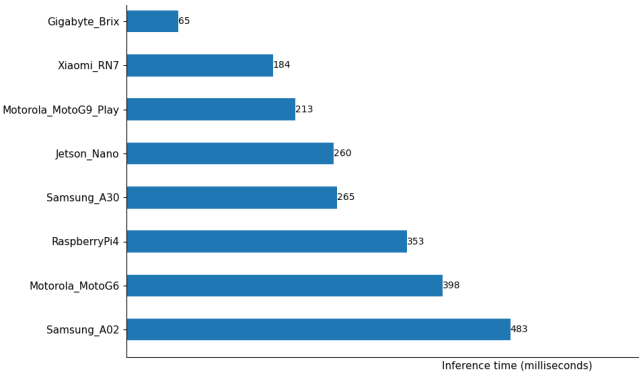
Irrespective of the application domain, these example application scenarios present streams with different characteristics not only by the dynamics of input generation but also the computing capability required to perform inferences over the input. In addition, it is assumed that inference results should be available at the master node with minimun latency, i.e., as close as possible to realtime.

4 Experiments Setup and Metrics

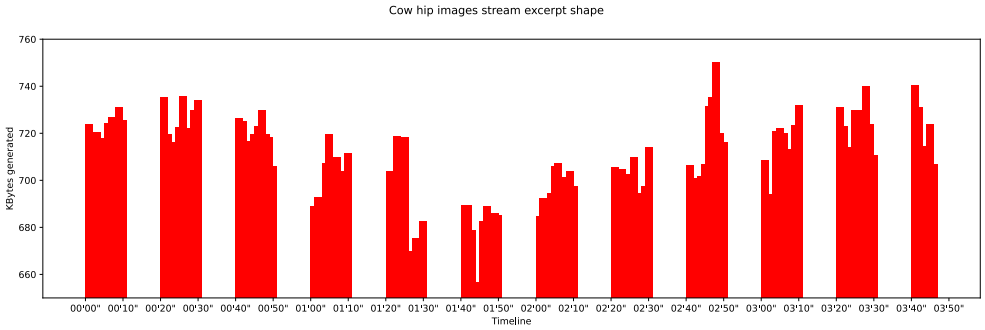
For setting up and executing tests we employed a tool set that combines several open source programs [34, 47] and hardware [42] modules. The tool set allow us to automate tasks involved in experimenting with



(a) Cow image example

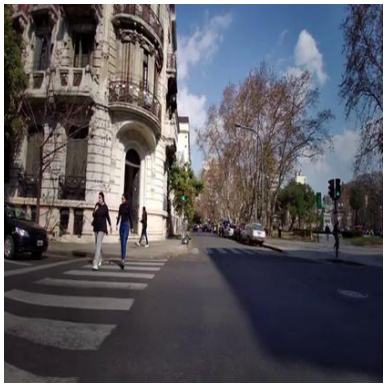


(b) Cow BCS CNN model benchmarking

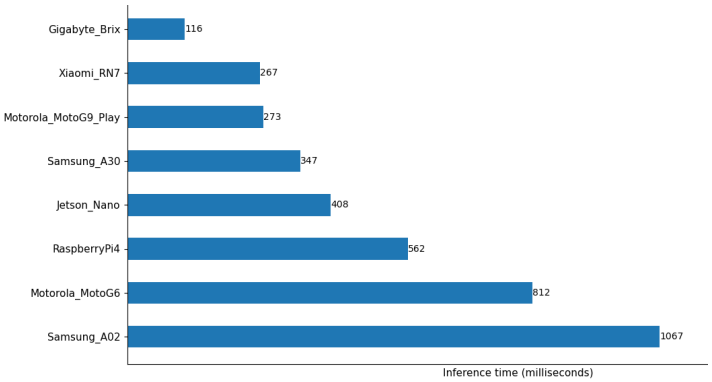


(c) Cow images stream

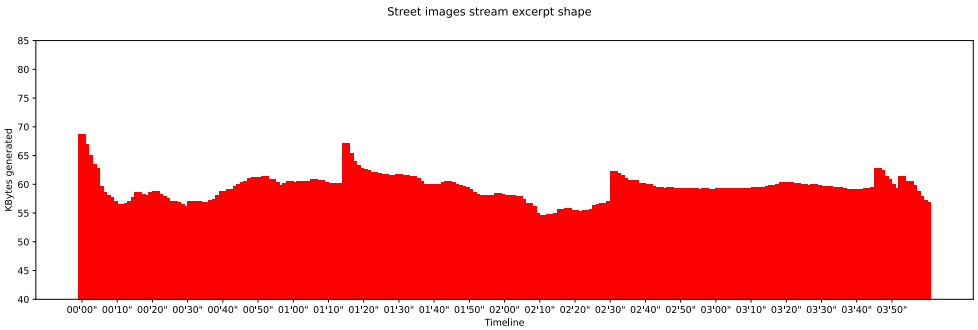
Figure 2: BCS - workload characterization



(a) street image example



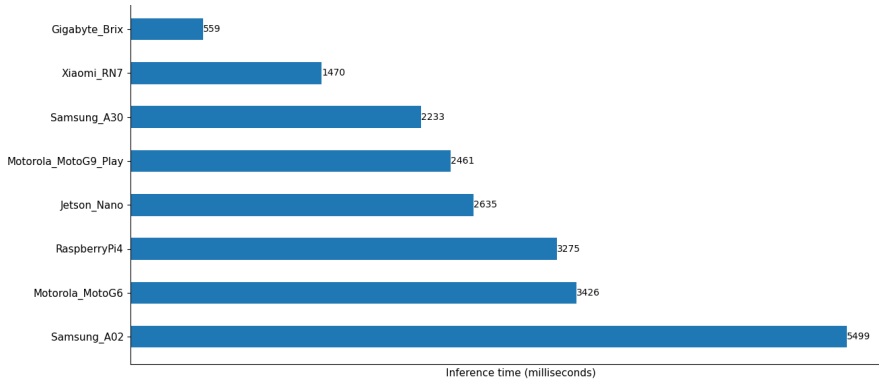
(b) YoloV4 tiny CNN model benchmarking



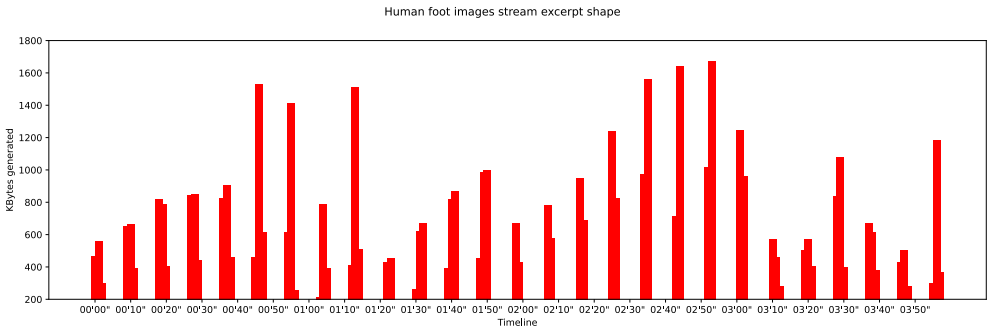
(c) Street images stream

Figure 3: SWT - workload characterization

(a) Foot image example



(b) Foot image CNN model benchmarking



(c) Foot images stream

Figure 4: DMED - workload characterization

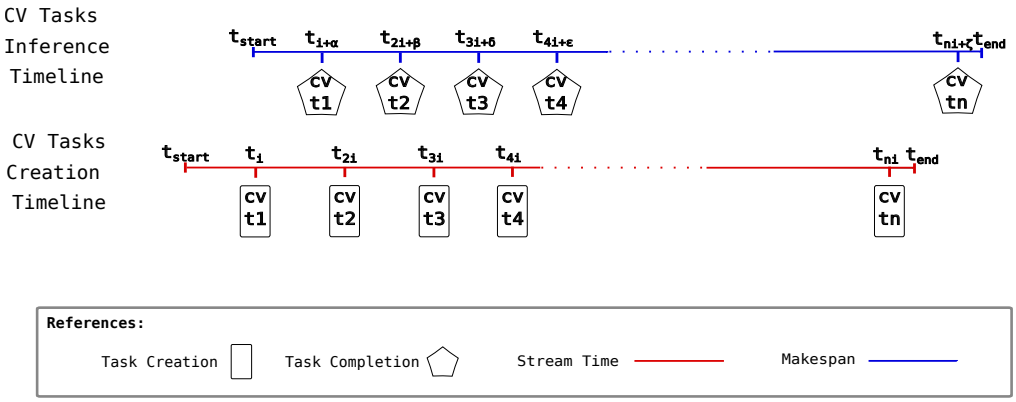


Figure 5: Accumulated inference latency: visual explanation

smartphones and real workloads including cluster formation, devices preparation (battery charge/discharge actions), deep learning models deployment, stream dynamics reproduction, and results collection and summarization.

As discussed earlier in the paper, workloads in edge computing and IoT scenarios are commonly computed with medium-to-high-end edge nodes such as SBCs. In the benchmarks we compare performance of several SBC-based and smartphone-based setups to complete the workloads described in Section 3.3. The "Edge Setup" column in Tables 5,6 and 7 differentiates both setups. Considering that smartphone-based setups comprise clusters of smartphones, Table 4 expands on the model and battery information of nodes integrating each cluster. All clusters include a single instance of each smartphone model. Smartphone brands and models are indicated using a short name that can be found in Table 3. The order in which these short names appear in a row corresponds to the order in which smartphones' initial battery values appear. The latter were randomly chosen. Cluster heterogeneity arises from the combination of smartphones quantity, models and initial battery levels. Cluster maximum energy and Cluster initial energy, both expressed in Joules, as well as, Cluster initial battery expressed in %, all derived from individual smartphones data, complement the cluster information.

One of the metrics we report is accumulated inference latency, a measure of how long the makespan deviates from the stream time. The stream time is the time window where CV tasks are created, while makespan is the time the inference system employs in completing all the created CV tasks. Figure 5 graphically represents how these entities coexist in time. Inference completion of a task naturally occurs in time after the CV task creation. While tasks creation could occur by following a quite controlled and periodically process, tasks inference process does not. This is what $\alpha, \beta, \gamma, \delta, \varepsilon, \zeta$ represent in the CV tasks inference timeline associated to each CV task completion event, i.e. different completion times that could be given as consequence but not limited to variations in network latency, CNN data input content and nodes execution capability. When subtracting stream time to makespan, the result is the accumulated inference latency. The nearest this difference to zero, the better the setup capability is to respond with proper inferences time compared to the time taken to create all CV tasks. A derived metric which gives a notion of the average time employed by the setup to complete a CV task after a new CV task is available, could be obtained dividing the accumulated inference latency by the number of CV tasks created. We call this latter inference latency.

Another metric we report is total energy consumption. To measure energy consumption we followed two approaches depending on the type of setup. For SBC-based setups we employ a power monitor [3] that uses a toroid connected as shown in Figure 6, which allow us to collect current and voltage readings from an SBC in a non-invasive way while running workloads. The power monitor is connected to the 220V AC power and it has a socket after a toroid where the SBC power source is connected. Current and Voltage readings are transmitted in realtime via a USB port to a laptop. The power monitor gives two current and voltage values per second. Then, for a workload scenario whose inference time is around 3 minutes and 50 seconds, the reported energy consumption is obtained by averaging 460 samples.

For smartphone-based setups the described approach for registering energy consumption cannot be

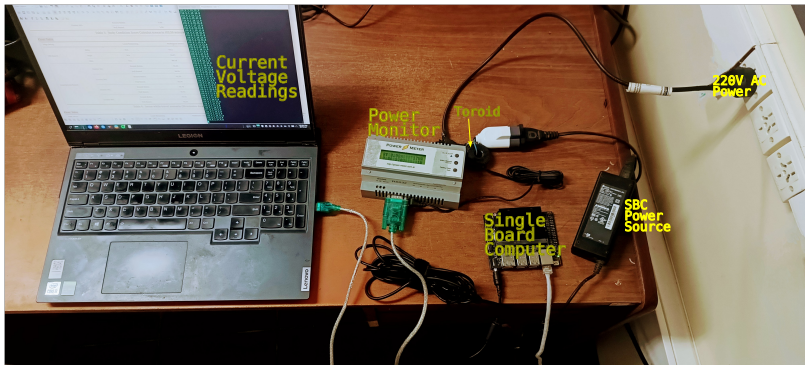


Figure 6: SBC power monitoring connection scheme

used, because smartphone batteries are employed as the power source. We, thus, followed a different approachby registering smartphones battery levels at the beginning and at the end of a workload scenario. Then, we used battery manufacturer information – such as battery mAh and operating voltag e– of each smartphone to approximate the total energy used during the workload scenario execution (WSE), expressed in Joules:

$$WSEJoules = \sum_{i=0}^{\#smartphones} (battLevelDrops^i * battMAh^i * 0.01) * battVoltage^i * 3.6 \quad (1)$$

where $\#smartphones$ corresponds to the amount of smartphones integrating the cluster, $battLevelDrops^i$is the amount of battery level drops registered for the smartphone i used for executing the assigned workload, which is in turn calculated as:

$$battLevelDrops = batt_{endWSE} - batt_{startWSE} \quad (2)$$

where $batt_{startWSE}$ and $batt_{endWSE}$ are the battery level of smartphone i at the start and end of the workload scenario execution respectively, $battMah^i$ and $battVoltage^i$ are milli-amperes hour and operating voltage information respectively, provided by the battery manufacturer of smartphone i. Formula 1 emerges from combining a rule of three that relates battery percentage usage with the unit conversion formula from mAh to Joules which is $mAh * voltage * 3.6 = JoulesOfenergy$.

Additionally, since a workload scenario execution using smartphone-based setups might cause different battery percentage usage on each node, we report Jain's fairness index [27] as a metric that summarizes intra-cluster energy utilization. In other words, the metric is used to measure the disparity of energy pulled by the system from resource provider nodes. The metric takes values from 0 to 1. The nearest the index to value of one the more balanced the energy utilization among cluster nodes is. The Jain's index inspired formula we applied is:

$$Fairness = \frac{\left[\sum_{i=0}^{\#smartphones} battLevelDrops^i\right]^2}{\#smartphones * \left(\sum_{i=0}^{\#smartphones} [(battLevelDrops^i)]^2\right)} \quad (3)$$

Since all smartphone-based setups comprise clusters of at least two heterogeneous smartphones, it was necessary to indicate the load balancing strategy used to distribute load between these. For this reason, all smartphone-based setups are indicated with Round Robin or Pull-Based as value in the load balancing column. Round Robin is a classic load balancing strategy that assigns an incoming task to the next available node using a rotating scheme, ensuring workload is evenly distributed among all participating devices. By contrast, with a Pull-Based approach tasks are pulled by nodes from a shared queue based on their availability. In our configurations, cluster nodes were configured to pull at most one task per request. In turn, a node is able to make a request when the result of the previously pulled task was sent back to the Edge master, i.e. the task is marked by the system as completed.

Cluster ID	Cluster size	Smartphone models	Smartphones initial battery %	Cluster initial energy (Joules)	Cluster max. energy (Joules)[1]	Cluster initial battery %[2]
Cl202	2	XRN7, MotG9	87, 75	99532.80	123840	80.37
Cl203	2	XRN7, MotG9	27, 54	51904.80	123840	41.91
Cl205	2	MotG9, SamA30	54, 30	53152.20	122454	43.41
Cl301	3	SamA30, XRN7, MotG9	27, 45, 60	80582.58	177894	45.30
Cl302	3	SamA30, SamA02, MotG9	27, 45, 60	86195.88	190368	45.28
Cl304	3	MotG6, XRN7, MotG9	77, 25, 53	81712.80	164880	49.55
Cl305	3	MotG9, MotG6, SamA30	54, 92, 25	88206.30	163494	53.95
Cl408	4	MotG6, SamA30, XRN7, MotG9	30, 27, 45, 60	92894.58	218934	42.43
Cl409	4	MotG6, SamA30, XRN7, MotG9	88, 80, 38, 49	133941.6	218934	61.18
Cl411	4	MotG6, SamA30, XRN7, MotG9	25, 52, 44, 38	88753.68	218934	40.54

Table 4: Details of the smartphone-based setups

4.1 Benchmark Results

4.1.1 Performance Analysis

Figure 7 shows the performance achieved by all setups for all scenarios. Blue bars correspond to SBC-based setups while orange ones to smartphone-based setups. By comparing all setups, it can be clearly noted that the best performance, i.e. the lowest accumulated inference latency value in all tested scenarios is achieved with the Gigabyte Brix setup. The close-to-zero value indicates that inferences performed with this setup is near to real-time, i.e. inference results associated to all CV tasks are obtained within the stream time. Concretely, in the BCS scenario, the accumulated latency in a stream time of 3' 50" was 3 seconds. In the SWT scenario, the accumulated latency in a stream time of 30' was 1.2 seconds while in the DMED scenario with a stream time of 8'55", the value was 3.6 seconds. With the Raspberry Pi4 and Nvidia Jetson Nano setups the obtained accumulated inference latency was significantly greater, and in most cases, the inference latency was the stream time multiplied by a factor of two to four – for detailed values please refer to tables in appendix 6 – . The Nvidia Jetson nano setups performed better than the Raspberry Pi4 setups in all cases. However, the workload always exceeded the inference capability. With these setups, reducing workload would be necessary to achieve near real-time performance. Some alternatives to reduce workload are creating more lightweight CV tasks by dropping frames and/or offloading some CV tasks to other Edge nodes. These alternatives could come, in turn, with some QoS degradation due to increased network latency.

When comparing smartphone-based setups for the BCS scenario, we empirically confirm what other studies found [22, 24]: by incrementing the number of participating smartphones, performance improves, i.e, lower accumulated inference latency is obtained. This is also true for the SWT and DMED scenarios. However, not only the number of smartphones impact the setup inference service capability, but also the smartphones computing characteristics. Clusters 301 and 302, for instance, are of the same size. Both clusters are composed by the Samsung A30 and the Motorola Moto G9 play. However, the Samsung A02 smartphone present in Cluster 302 is not present in Cluster 301, which has the Xiaomi Redmi Note 7 instead. If we compare the inference times associated to the latter mentioned smartphone models and reported in Figures 2b,3b and 4b, we note that the computing capability of the Samsung A02 is in between 2.62 and 3.99 times slower than that of the Xiaomi Redmi Note 7. The effect of such difference is observed in the performance values registered for the SWT scenario, where both clusters are tested. While the accumulated inference latency with Cluster 301 was between $\backsim$16 and $\backsim$31 seconds, with Cluster 302 it was between $\backsim$131 and $\backsim$678 seconds.

[1] To calculate this value we used a formula derived from formula 1 assuming each smartphone is fully charged.

[2] The value was calculated using formula 1, replacing batteryDrops by battery level of each smartphone at the begining of a workload test.

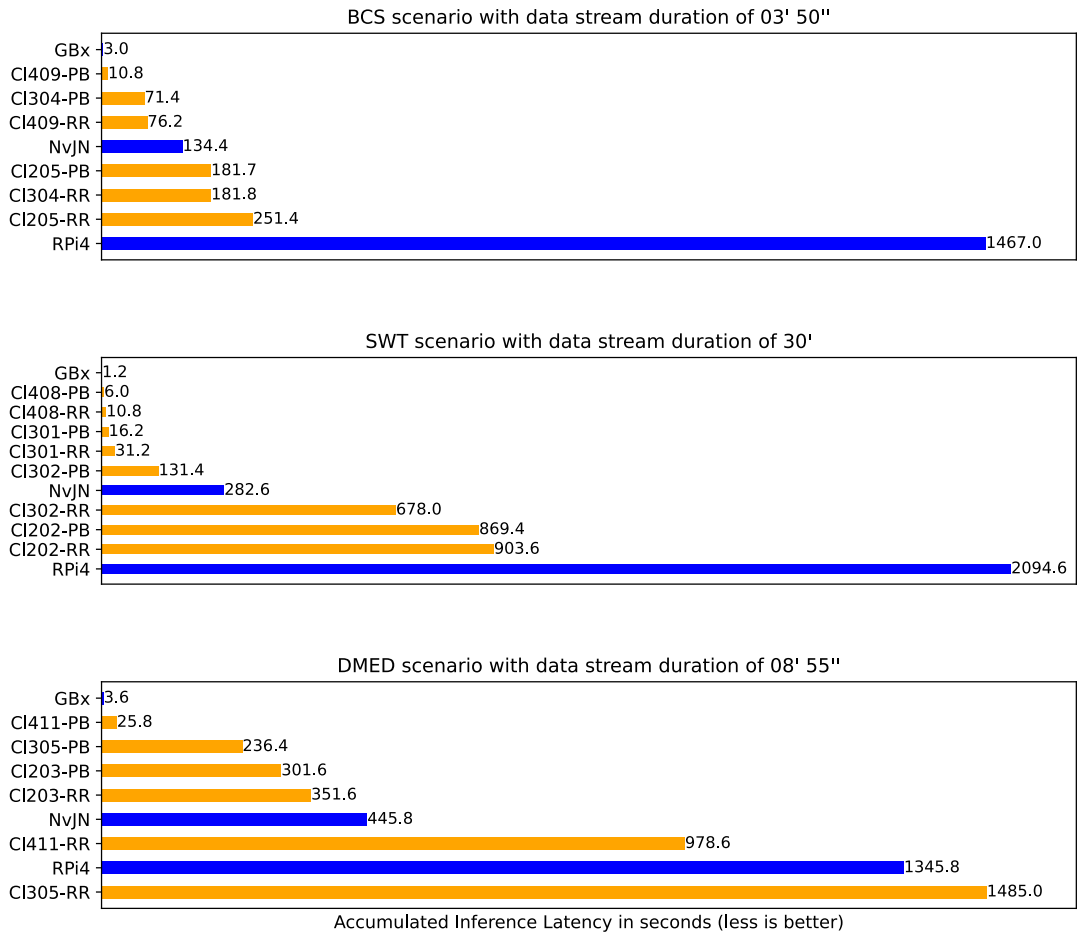


Figure 7: Accumulated Inference Latency for the three scenarios. For the cluster setups, we indicate the load balancing technique used (PB=pull based; RR=round robin). Load balancing is not used in SBC-based setups.

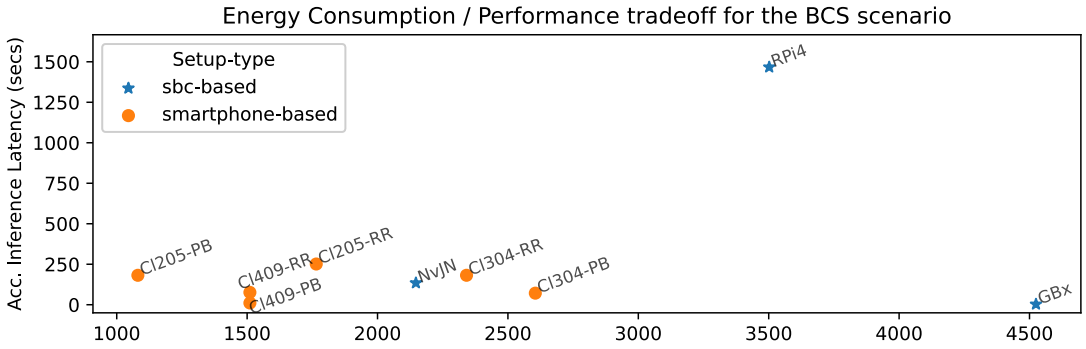
Another observation when comparing smartphone-based setups is that in general pull-based load balancing scheme achieves lower accumulated inference latency than round robin. This is because using pull-based allow the cluster to balance the workload in accordance to the availability and computing capability of worker nodes rather than treat all nodes with the same capability as round robin does. This advantage of pull-based over round robin diminishes in clusters where nodes present quite similar computing capability, which is the case of Cluster 202 in the SWT scenario, or even when computing resources are apparently abundant enough to satisfy the workload requirements, which seems to be the case of Cluster 408. In general, the most competitive accumulated inference latency comparison to that obtained by the Gigabyte Brix setup was always achieved with a pull-based load balancing scheme. For example, in the BCS scenario, the difference between Cluster 409 and Gigabyte Brix setups was less than 8 seconds. The third best performance was also achieved with a pull-based setup. The Cl304-PB, a size 3 cluster, achieves quite similar performance to that of a size cluster 4 combined with round robin, i.e. Cl409-RR. A similar analysis is valid in the SWT and DMED scenarios, with Clusters 408 and 411 respectively. Pull-based load balancing scheme achieves considerable better performance than round robin when clusters nodes present heterogeneous computing capabilities or when available computing capabilities are tight to cope with realtime tasks requirements.

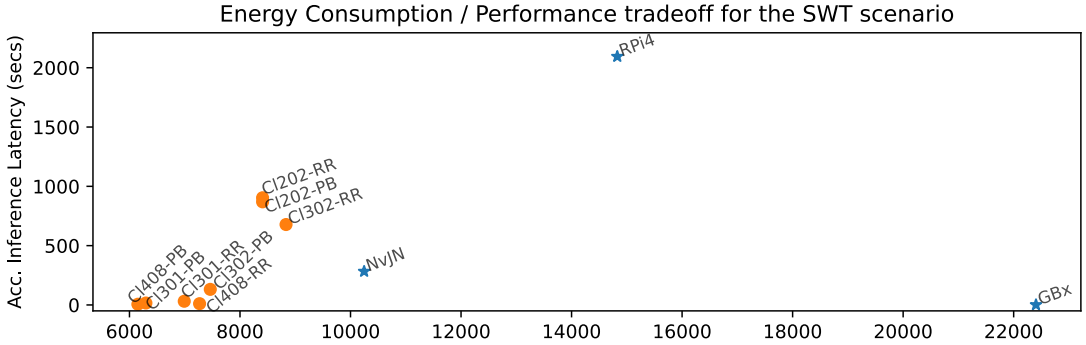
4.1.2 Energy Consumption-Performance Tradeoff Analysis

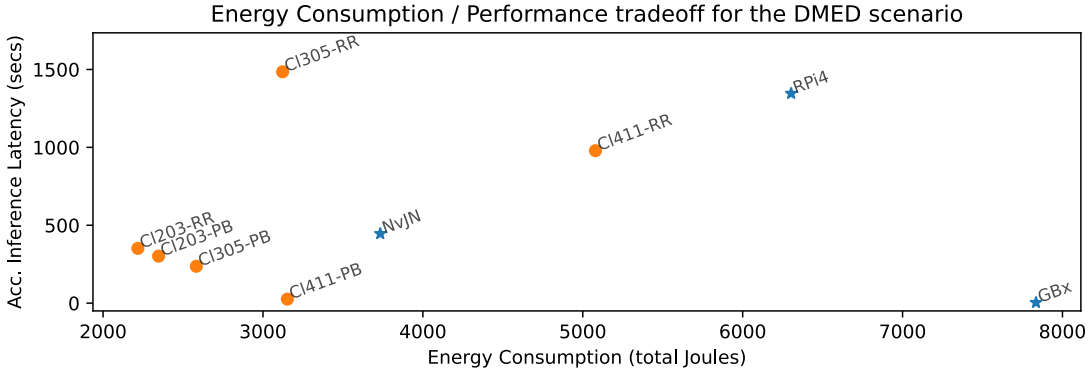
In view of the high energy consumption that AI applications already demand and expected further increases, studies in this regard [19, 39, 14] and initiatives such as the Low-Power Recognition Challenge [15] encourage the improvement of not only DLs model's performance but also the energy its execution and training consume. In this sense, we study the tradeoff between performance and energy consumed in all evaluated scenarios. Figure 8 reports the energy consumption and accumulated inference latency tradeoff between different setups in each tested scenario. The best setups are those that achieve the minimum accumulated inference latency using the least energy, i.e., those positioned close to the origin. As can be observed, in all scenarios the best balance is always achieved by orange dots with a label starting with Cl which corresponds to smartphone-based setups. It can be noted, for instance, that the accumulated inference latency of GBx setups are the lowest in BCS, SWT and DMED scenarios, however, these are also associated to the highest values of energy utilization. Moreover, the slowest setups are not necessary associated to the most convenient ones in terms of energy consumption. Interestingly, smartphone-based setups, particularly four-node clusters using pull-based as load balancing scheme offer a good tradeoff in all scenarios, i.e. competitive performance with the least energy consumption. These are the cases of Cl409-PB, Cl408-PB and Cl411-PB for BCS, SWT and DMED scenarios respectively. In other words, for the BCS scenario, when it is acceptable that all CV tasks results were calculated with a delay of 7.8 seconds w.r.t to the fastest setup, the Cluster 409 setup with pull-based scheme can achieve the same objective by utilizing $\backsim$1510 Joules, i.e. almost three times less energy than the Gigabyte Brix setup. Similar relations can be found for the SWT and DMED scenarios.

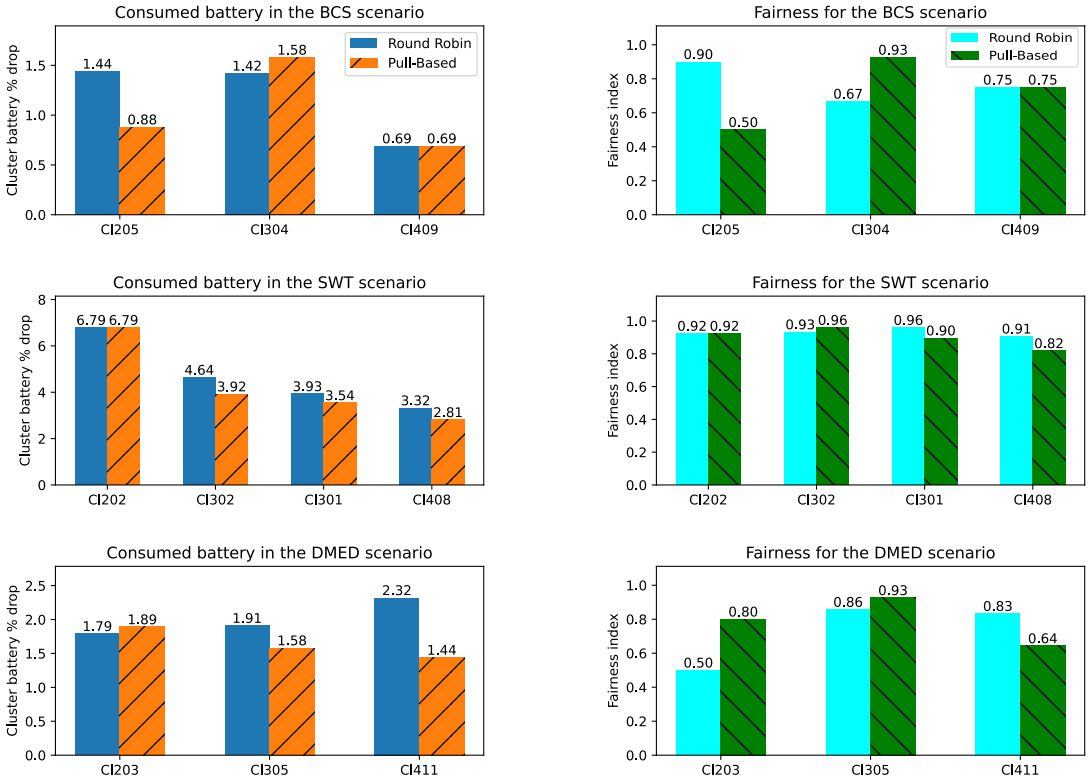
4.1.3 Battery Utilization Analysis

For smartphone-based setups, it is relevant to show the impact of CV tasks computation on clusters energy availability. Provided that clusters are composed of at least two nodes, cluster battery can be seen as the aggregation of nodes battery. To represent the cluster current battery level with a single positive integer value, we aggregate the Joules each smartphone contributes to the global cluster energy. By assuming that a cluster with 100% global battery level is one where batteries of all integrating smartphones are fully charged, using Equation 1 it is possible to calculate the global battery level expressed in percentage of a cluster for different smartphones battery levels. Then, Figure 9a shows global battery level drops for all smartphone-based setups evaluated in all scenarios. Since in all cases smartphone-based setups run workloads while unplugged from the electricity grid, i.e. with batteries in discharging mode, the battery drops were calculated as the difference between the initial and the final global battery level of a cluster before and after the workload execution respectively. In Figure 9a it can be noted, for instance, that CV tasks in the SWT scenario causes the highest global battery drops compared to BCS and DMED, which is mainly due to the length of the workload execution -30' compared to $\backsim$4' and $\backsim$9' in BCS and DMED


Energy Consumption / Performance tradeoff for the BCS scenario
Setup-type
sbc-based
smartphone-based
Acc. Inference Latency (secs)
RPi4
Cl205-PB
Cl409-RR
Cl205-RR
Cl409-PB
NvJN
Cl304-RR
Cl304-PB
GBx


Energy Consumption / Performance tradeoff for the SWT scenario
Acc. Inference Latency (secs)
RPi4
Cl202-RR
Cl202-PB
Cl302-RR
Cl408-PB
Cl301-PB
Cl301-RR
Cl302-PB
Cl408-RR
NvJN
GBx


Energy Consumption / Performance tradeoff for the DMED scenario
Acc. Inference Latency (secs)
Cl305-RR
RPi4
Cl411-RR
Cl203-RR
Cl203-PB
Cl305-PB
NvJN
Cl411-PB
GBx
Energy Consumption (total Joules)



(a) Global battery impact

(b) Fairness index

Figure 9: CV tasks impact on clusters global energy availability and Fairness index

respectively.

Another observation that emerges from Figure 9a is that when comparing cluster battery level drops between clusters w.r.t. the same scenario, it appears to decrease as cluster size increases. In fact, as more smartphones integrate a cluster the cluster battery also increases and one percentage of such cluster battery is represented by a larger number in Joules. Something similar happens when two clusters are of the same size but one has smartphones whose batteries have more energy capacity than the other, which is the case of Cl301 and Cl302. However, a cluster with a bigger battery does not imply that it will complete tasks in a more energy-efficient way. As an example, refer to the "Max. Cluster Energy" column of Table 4 to appreciate that Cl302 has slightly larger cluster battery capacity than Cl301, because the Xiaomi RN7 with a battery capacity of 4000 mAh in Cl301 was swapped by the Samsung A02 with a battery capacity of 4900 mAh in Cl302. However, according to Figure 3b, the latter smartphone is almost four times slower to perform a task from the SWT scenario than the former, meaning that Cl302 will spent more energy than Cl301 in completing the same amount of tasks. This explains why despite having a larger cluster battery, the battery drop of Cl302 is larger than that of Cl301 in SWT scenario shown in Figure 9a. Moreover, it is relevant to point out that, in general, the pull-based load balancing scheme achieves equal or less global battery drops than round robin (approximately 0.5%).

Finally, we analyze fairness values derived from applying the round robin and the pull-based load balancing schemes. The index serves as a hint on how the edge cluster utilizes energy from all participating smartphones. The more the value approximates to one, the more balanced the battery utilization is. Figure 9b reveals that in some clusters, the pull-based scheme achieves slightly higher fairness values than that of round robin, while in other cases, the results are the opposite. There are cases where even both schemes achieve equal fairness values. Since there is not a clear pattern, we can conclude that there is not a clear

winner load balancing scheme as measured by the fairness index.

5 Discussion

A few previous works have also focused on proving the strengths of edge computing for reducing end-to-end latency and allowing the execution of data intensive IoT and delay sensitive applications [48, 29]. In this work, we explored how edge computing capabilities can be augmented with smartphone clusters under different workload scenarios for real-time AI inferences involving image streams. The competitive performance and reduced energy consumption that smartphone clusters showed, along with the pervasiveness of such devices worldwide evidence their suitability for being considered a powerful harness to complement in-the-field, already present edge settings, and/or to fully support opportunistic distributed inference in a sustainable form. However, there are several questions still open, demanding for further research. One of these relates to whether current software stacks including middlewares for distributed computing [43, 44] and frameworks for AI execution [16] in low-powered devices are prepared to integrate smartphone clusters as a new type of edge node by attending diverse aspects related to computational resource provision with devices that present non-dedicated resources, i.e. with user applications that have high priority on the usage of resources, unstable communication due to device's owner physical position changes, and resource scavenging time limited by batteries operation time.

Another open question concerns incentives that make a device's onwer to offer unused computing mobile resources. In the literature, it has been identified at least two schemes. One of them is associated to altruistic attitudes of devices owners, in some sense, similar to what is promoted in volunteer computing platforms with unused computing cycles of desktop computers for scientific projects. Only that, in this case, targeting daily normal situations that required in-the-field inference capability either for reducing cost of data transmission -e.g. chronic disease health monitoring-, complement safety-guard mechanisms in risky situations -e.g., alerting about obstacles in the road while driving-, providing realtime response to improve user experience - e.g., augmented reality applications in tourism-related activities- among others. By contrast, the other incentive scheme we visualize is based on a opportunistic inference as a service approach that benefits device's owner with some revenue in the form of credits [11] or reputation [10], in exchange of completing tasks that require in-the-field inference capability, for example, as a complement to surveillance cameras for crime prevention in public spaces. The energy consumption caused by using efficient hardware such as that embedded in smartphones w.r.t using other edge nodes can be used as baseline for establishing economically and environmentally sustainable inference mechanisms.

In this regard, another aspect worth to be mentioned that is intimately related to AI and computing infrastructures concerns energy consumption and hence, indirectly, CO2 emissions. In its World Energy Outlook 2024 [2, 1], the well-known International Energy Agency (IEA), projects that energy consumption in datacenters is set to expand in the future, in part due to digitalization of the economy but also as a consequence of the rapid advance of AI technologies and the outbreak of AI startups. In fact, figures from 2022 also reported by the IEA -i.e. even some years before the AI boom we have been experiencing since then- result in datacenters consuming 1-2% of the world's electricity, which is backed up by estimates from similar studies [33, 32]. This positions energy-efficient approaches to edge computing as a crucial path to cap these estimates while providing viable computing settings to continue helping bringing AI to the masses. Then, edge computing research should not only merely quantify energy savings compared to traditional computing approaches but also develop mechanisms to measure the impact on CO2 emissions at the software level such as [37]. Quantification must also consider current mobile hardware manufacturing practices, for example promoting replaceable (sealed) batteries, which encourage users to upgrade to a new device when battery performance declines, rather than simply replacing the battery. Metrics such as the Software Carbon Intensity (SCI) index [4] might serve as a pertinent reference since it captures the environmental impact of software in terms of CO2 emissions considering carbon intensity of the energy used to run software and the embedded emissions due to the carbon footprint to build the hardware on which the software runs.

6 Conclusions and Future Works

In this work, we explored the computing capabilities of mobile distributed computing setups for real-time inferences by experimenting with heterogeneous AI workloads, built upon real convolutional neural network (CNN) models and images data sets that were used to emulate computer vision (CV) tasks generated in a stream-like fashion. We run and compare the performance achieved by several setups built with commodity hardware including single board computers (SBCs) and smartphone clusters that could be found or exploited in practical Edge computing and IoT application scenarios.

We conclude that tiny clusters composed by two to four low to middle-end smartphones using a pull-based load balancing scheme offer competitive performance compared to an SBC equipped with a reasonable CPU (Intel core i5 8th generation) in computing realtime inferences of CV tasks. This suggests that to satisfy certain delay sensitive application scenarios, the exploitation of already-in-place smartphones clusters can be considered before the investment and deployment of SBCs. According to Table 3 the necessary investment for supporting the evaluated scenarios using a powerful SBC like the Gigabyte Brix is only 30% cheaper than with a cluster-based setup of four low to mid-end smartphones. This is when not considering that smartphones can be already present in the application context and the computing capability hired to device owners. In any case, energy consumption measurements show that the tradeoff including performance and energy consumption favors smartphone clusters over SBCs. The former can deliver competitive performance with a reduction of around tree times the energy consumption spent by a powerful SBC such as the Gigabyte Brix.

Several improvement opportunities lend itself to future work. In line with developing new load balancing heuristics, and based on the results from preliminary runs, we plan to improve previously published heuristics that use MFLOPs to rank nodes computing capabilities [20]. An alternative solution is to use information derived from the Tensorflow benchmarking tool [5]. In spite of differences observed in inference times obtained via the Tensorflow benchmarking tool and results collected in experiments using real input, relative benchmarked positions between edge nodes are similar, meaning that it is feasible to use such tool to build node rankings that serve as input for the load balancing component to distribute tasks based on an alternative nodes performance score to the classic MFLOPs.

Another future direction is to consider other forms of workload distributions, i.e. that exploit the structure of deep learning (DL) arquitectures, for example, split computing, early exiting techniques, and model quantization [12] to better exploit CPU capabilities of mobile platforms while performing network inferences.

Acknowledgments

This work was funded by CONICET via grants PIBAA-28720210101298CO and PIP-11220210100138CO.

References

[1] Iea electricity 2024. `https://www.iea.org/reports/electricity-2024`. [Online; accessed 7-March-2025].

[2] Iea reports. `https://www.iea.org/reports/world-energy-outlook-2024`. [Online; accessed 7-March-2025].

[3] Power meter website. `https://www.powermeter.com.ar/home`. [Online; accessed 2-February-2025].

[4] Software carbon intensity (sci). `https://sci.greensoftware.foundation/`. [Online; accessed 7-March-2025].

[5] Tensorflow lite benchmarking tool. `https://www.tensorflow.org/lite/performance/measurement`. [Online; accessed 2-February-2025].

[6] Yolov4 project. https://github.com/hunglc007/tensorflow-yolov4-tflite. [Online; accessed 2-February-2025].

[7] Juan Rodríguez Alvarez, Mauricio Arroqui, Pablo Mangudo, Juan Toloza, Daniel Jatip, Juan M Rodríguez, Alfredo Teyseyre, Carlos Sanz, Alejandro Zunino, Claudio Machado, et al. Body condition estimation on cows from depth images using convolutional neural networks. *Computers and electronics in agriculture*, 155:12–22, 2018.

[8] Jun Bao and Qiuju Xie. Artificial intelligence in animal farming: A systematic literature review. *Journal of Cleaner Production*, 331:129956, 2022.

[9] Michel Bergoeing, Andres Neyem, Paulina Jofré, Camila Hernández, Juan Chacón, Richard Morales, and Matías Giddings. Exploring the potential of an ai-integrated cloud-based mhealth platform for enhanced type 2 diabetes mellitus management. In *International Conference on Ubiquitous Computing and Ambient Intelligence*, pages 100–111. Springer, 2023.

[10] Dimitris Chatzopoulos, Mahdieh Ahmadi, Sokol Kosta, and Pan Hui. Openrp: A reputation middleware for opportunistic crowd computing. *IEEE Communications Magazine*, 54(7):115–121, 2016.

[11] Dimitris Chatzopoulos, Mahdieh Ahmadi, Sokol Kosta, and Pan Hui. Flopcoin: A cryptocurrency for computation offloading. *IEEE transactions on Mobile Computing*, 17(5):1062–1075, 2017.

[12] Yanjiao Chen, Baolin Zheng, Zihan Zhang, Qian Wang, Chao Shen, and Qian Zhang. Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions. *ACM Computing Surveys (CSUR)*, 53(4):1–37, 2020.

[13] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing*, 20(2):565–576, 2019.

[14] Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88, 2019.

[15] Kent Gauen, Rohit Rangan, Anup Mohan, Yung-Hsiang Lu, Wei Liu, and Alexander C Berg. Low-power image recognition challenge. In *2017 22nd Asia and South pacific design automation conference (ASP-DAC)*, pages 99–104. IEEE, 2017.

[16] Qianyu Guo, Sen Chen, Xiaofei Xie, Lei Ma, Qiang Hu, Hongtao Liu, Yang Liu, Jianjun Zhao, and Xiaohong Li. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 810–822. IEEE, 2019.

[17] Xiaotian Guo, Andy D Pimentel, and Todor Stefanov. Automated exploration and implementation of distributed cnn inference at the edge. *IEEE Internet of Things Journal*, 10(7):5843–5858, 2023.

[18] Marjan Gusev and Yingwei Wang. Formal description of dew computing. In *Proceedings of The 3rd International Workshop on Dew Computing*, pages 8–13, 2018.

[19] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 21(248):1–43, 2020.

[20] Matías Hirsch, Cristian Mateos, Alejandro Zunino, Tim A Majchrzak, Tor-Morten Grønli, and Hermann Kaindl. A task execution scheme for dew computing with state-of-the-art smartphones. *Electronics*, 10(16):2006, 2021.

[21] Yichen Hou, Sahil Garg, Lin Hui, Dushantha Nalin K Jayakody, Rui Jin, and M Shamim Hossain. A data security enhanced access control mechanism in mobile edge computing. *IEEE Access*, 8:136119–136130, 2020.

[22] Zhiming Hu, Ahmad Bisher Tarakji, Vishal Raheja, Caleb Phillips, Teng Wang, and Iqbal Mohomed. Deephome: Distributed inference with heterogeneous devices in the edge. In *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications*, pages 13–18, 2019.

[23] Yakun Huang, Xiuquan Qiao, Schahram Dustdar, Jianwei Zhang, and Jiulin Li. Toward decentralized and collaborative deep learning inference for intelligent iot devices. *IEEE Network*, 36(1):59–68, 2022.

[24] Yakun Huang, Xiuquan Qiao, Wenhai Lai, Schahram Dustdar, Jianwei Zhang, and Jiulin Li. Enabling dnn acceleration with data and model parallelization over ubiquitous end devices. *IEEE Internet of Things Journal*, 9(16):15053–15065, 2021.

[25] Yakun Huang, Xiuquan Qiao, Pei Ren, Ling Liu, Calton Pu, Schahram Dustdar, and Junliang Chen. A lightweight collaborative deep neural network for the mobile web in edge cloud. *IEEE Transactions on Mobile Computing*, 21(7):2289–2305, 2020.

[26] Yakun Huang, Xiuquan Qiao, Jian Tang, Pei Ren, Ling Liu, Calton Pu, and Junliang Chen. Deepadapter: A collaborative deep learning framework for the mobile web using context-aware network pruning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 834–843. IEEE, 2020.

[27] Rajendra K Jain, Dah-Ming W Chiu, and William R Hawe. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, DEC*, 1984.

[28] Abdul Rehman Javed, Waqas Ahmed, Sharnil Pandya, Praveen Kumar Reddy Maddikunta, Mamoun Alazab, and Thippa Reddy Gadekallu. A survey of explainable artificial intelligence for smart cities. *Electronics*, 12(4):1020, 2023.

[29] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

[30] Gang Li and Jun Cai. An online incentive mechanism for collaborative task offloading in mobile edge computing. *IEEE Transactions on Wireless Communications*, 19(1):624–636, 2019.

[31] Palak Mahajan, Shahadat Uddin, Farshid Hajati, and Mohammad Ali Moni. Ensemble learning for disease prediction: A review. *Healthcare*, 11(12), 2023.

[32] Jens Malmodin, Nina Lövehagen, Pernilla Bergmark, and Dag Lundén. Ict sector electricity consumption and greenhouse gas emissions–2020 outcome. *Telecommunications Policy*, 48(3):102701, 2024.

[33] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, 2020.

[34] Cristian Mateos, Matías Hirsch, Juan Manuel Toloza, and Alejandro Zunino. Livedewstream: A stream processing platform for running in-lab distributed deep learning inferences on smartphone clusters at the edge. *SoftwareX*, 20:101268, 2022.

[35] Duc-Khanh Nguyen, Chung-Hsien Lan, and Chien-Lung Chan. Deep ensemble learning approaches in healthcare to enhance the prediction and diagnosing performance: The workflows, deployments, and surveys on the statistical, image-based, and sequential datasets. *International Journal of Environmental Research and Public Health*, 18(20), 2021.

[36] Partha Pratim Ray. Minimizing dependency on internetwork: Is dew computing a solution? *Transactions on Emerging Telecommunications Technologies*, 30(1):e3496, 2019.

[37] Andreas Schmidt, Gregory Stock, Robin Ohs, Luis Gerhorst, Benedict Herzog, and Timo Hönig. carbond: An operating-system daemon for carbon awareness. *ACM SIGENERGY Energy Informatics Review*, 4(3):52–57, 2024.

[38] Zorislav Šojat and Karolj Skala. The rainbow through the lens of dew. In *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 1916–1921. IEEE, 2020.

[39] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13693–13696, 2020.

[40] Zain Taufique, Antonio Miele, Pasi Liljeberg, and Anil Kanduri. Adaptive workload distribution for accuracy-aware dnn inference on collaborative edge platforms. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 109–114. IEEE, 2024.

[41] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, pages 328–339. IEEE, 2017.

[42] Juan Manuel Toloza, Matías Hirsch, Cristian Mateos, and Alejandro Zunino. Motrol: A hardware-software device for batch benchmarking and profiling of in-lab mobile device clusters. *HardwareX*, 12:e00340, 2022.

[43] Daniel R Torres, Cristian Martín, Bartolomé Rubio, and Manuel Díaz. An open source framework based on kafka-ml for distributed dnn inference over the cloud-to-things continuum. *Journal of Systems Architecture*, 118:102214, 2021.

[44] Shreshth Tuli, Nipam Basumatary, and Rajkumar Buyya. Edgelens: Deep learning based object detection in integrated iot, fog and cloud computing environments. In *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, pages 496–502. IEEE, 2019.

[45] Zaib Ullah, Fadi Al-Turjman, Leonardo Mostarda, and Roberto Gagliardi. Applications of artificial intelligence and machine learning in smart cities. *Computer Communications*, 154:313–323, 2020.

[46] Xiaolong Xu, Yuan Xue, Lianyong Qi, Yuan Yuan, Xuyun Zhang, Tariq Umer, and Shaohua Wan. An edge computing-enabled computation offloading method with privacy preservation for internet of connected vehicles. *Future Generation Computer Systems*, 96:89–100, 2019.

[47] Virginia Yannibelli, Matías Hirsch, Juan Toloza, Tim A Majchrzak, Tor-Morten Grønli, Alejandro Zunino, and Cristian Mateos. Bagess: A software module based on a genetic algorithm to sequentially order load-balancing evaluation scenarios over smartphone-based clusters at the edge. *IEEE Access*, 2024.

[48] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pages 1–13, 2017.

[49] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[50] Zhuoran Zhao, Kamyar Mirzazad Barijough, and Andreas Gerstlauer. Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2348–2359, 2018.

Appendix A

Tables 5, 6 and 7 contain the data reported in all graphs presented in Section 4.1 for BCS, SWT and DMED scenarios.

Edge Setup		Load Balancing	Acc. Inference Latency (secs)	Jain's Fairness Index	Cluster Consumed Battery (%)	Consumed Energy (Joules)
SBC-based	RPi4	N/A	1467	N/A	N/A	3501.97
	NvJN	N/A	134.4	N/A	N/A	2147.13
	GBx	N/A	**3.0**	N/A	N/A	4524.82
Smartphone-based	Cluster 205	Round Robin	251.4	0.9	1.44	1765.08
		Pull Based	181.7	0.5	0.88	1081.08
	Cluster 304	Round Robin	181.8	0.667	1.42	2341.29
		Pull Based	71.4	0.926	1.58	2605.10
	Cluster 409	Round Robin	76.2	0.750	0.69	1510.64
		Pull Based	10.8	0.750	0.69	1510.64

Table 5: Body Condition Score scenario (03:50 minutes data stream)

Edge Setup		Load Balancing	Acc. Inference Latency (secs)	Jain's Fairness Index	Cluster Consumed Battery (%)	Consumed Energy (Joules)
SBC-based	RPi4	N/A	2094.6	N/A	N/A	14827.11
	NvJN	N/A	282.6	N/A	N/A	10246.13
	GBx	N/A	**1.2**	N/A	N/A	22403.73
Smartphone-based	Cluster 202	Round Robin	903.6	0.924	6.79	8408.74
		Pull Based	869.4	0.924	6.79	8408.74
	Cluster 302	Round Robin	678.0	0.933	4.64	8833.07
		Pull Based	131.4	0.960	3.92	7462.43
	Cluster 301	Round Robin	31.2	0.960	3.93	6991.23
		Pull Based	16.2	0.896	3.54	6297.45
	Cluster 408	Round Robin	10.8	0.907	3.32	7268.61
		Pull Based	6.0	0.818	2.81	6152.04

Table 6: Sense while travel scenario (30 minutes data stream)

Edge Setup		Load Balancing	Acc. Inference Latency (secs)	Jain's Fairness Index	Cluster Consumed Battery (%)	Consumed Energy (Joules)
SBC-based	RPi4	N/A	1345.8	N/A	N/A	6302.99
	NvJN	N/A	445.8	N/A	N/A	3733.47
	GBx	N/A	**3.6**	N/A	N/A	7834.28
Smartphone-based	Cluster 203	Round Robin	351.56	0.5	1.79	2217.60
		Pull Based	301.56	0.8	1.89	2347.20
	Cluster 305	Round Robin	1485.0	0.857	1.91	3122.73
		Pull Based	236.4	0.926	1.58	2583.20
	Cluster 411	Round Robin	978.6	0.833	2.32	5079.27
		Pull Based	25.8	0.643	1.44	3152.65

Table 7: Disease Monitoring and Early Diagnosis scenario (08:55 minutes data stream)