Exploring Dew Computing for distributed inference using real testbeds

April 26, 2024

Abstract

AI is becoming ubiquous at the network edge and drives higher demand for computing resources. Is Dew computing a competitive resource provider for realtime image processing scenarios? On one hand, recent advancements on on-device AI processing, such that present in high-end commercial intelligent cameras, still have adependency on Cloud resources to complete complex AI tasks. . On the other hand, the performance achieved using commodity hardware in the form of SBCs and smartphone clusters as complement or replacement to Cloud resources in certain processing scenarios has not been extensible studied. Given the popularity gained by deep neural networks such as CNN for solving image classification problems, the developement of libraries like tensorflow lite allows practitioners to perform inferences with resource limited hardware, we elaborate a first-cut performance comparison with data derived from experiments on real test beds. First, we classify dew computing scenarios based on a multidimensional analysis of stream workloads with dimensions including data volume generated, data volume time distribution and processing time required per data unit.

1 Introduction

Deep learning (DL) application to perform on-site tasks is expanding exponentially. In the healthcare domain, it is applied for predicting, detecting, diagnosing, and prognosing different diseases [15, 11]. In food production, particularly, animal farming [3] these tasks include animal behavior recognition, growth evaluation, individual identification, among others. Associated to the smart city domain [21, 8], tasks include intelligent transportation, waste management, crime prediction, public infrastructure and green spaces maintenance. Many of these tasks involves the classification and/or object recognition using images as input. The application of CNNs (Convolutional Neural Networks) has become popular to perform these tasks due to the training procedure requires minimal human intervention compared to other machine learning techniques. In contrast, training a deep neural network model requires big datasets and powerful hardware. However, in these days, data gathering, leveraged by mobile and IoT devices, as well as computing power, provided by Cloud Computing platforms, allow to think in that such training process is facilitated 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 is deployed in a remote computing infrastructure that executes the model evaluation for input that in some cases must traverse the Internet. The latency incur until having the inference result has been been identified a limitation for time-sensitive applications. To overcome this issue and others such as unavailability of Internet connection, network overloading, privacy protection, among others, edge-based architectures has been proposed [23, 16].

Despite differences highlighted in the literature when referring to edge-based architectures [23], some of these simply terminologic, others describing substantial differences in the way their constituent components are defined interoperates with components of other computing layers [19], there are points in common which are minimizing data transportation and increase computing infrastructure usage close to where computing needs and data originates. Heterogeneity and energy

contrained are common features of computing infrastructure operating in cyber-physical and IoT systems. For these reasons, cooperation is a key concept of Dew Computing which is one of these edge-based architectures that endow low processing power and consumer devices with the role of main computing resource providers.

As formally described in [4], Dew computing has two modes of operation, local mode and global mode. In local mode, services are provided by Dew Servers with resources that can be accessed within the scope of a local area network. By contrast, the global operation mode involves reaching Cloud servers to mitigate limitations of resource constrained local nodes. Rendered services can be storage or computing capabilities. In this work, we focus on evaluating collaboration among consumer devices to perform cloud-independant computing services for real-time execution of machine vision tasks using CNN. We evaluate this with a series of experiments that comprise real smartphones and SBCs that exclusively relying on its local operation mode capabilities, collaborate in performing inferences using real CNN models deployed in targeted instances over data streams of different characteristics, i.e., workloads.

2 Related works

In this section we differentiate and objectively discuss efforts in the field of distributed inference research considering mobile devices as main computing resource provider. Such differentiation allow the reader to rapidly contextualize the contribution of this work.

2.1 Distributed Intelligence at the edge

When referring to distributed intelligence at the edge, it comes into mind the existence of tasks related to some sort of input collection and AI code execution in which intervene a fleet of resource limited nodes operating close to where the intelligence service is required and performing most or all the burden that such services imply by mostly relying on their computing and communicating capabilities. Such definition comprises concepts like federated learning and distributed inference, among others, that worth to differentiate because each can be associated to challenges with varying characteristics from the perspective of distributed systems.

2.1.1 Federated Learning

Federated Learning (FL) consist in a decentralized training with data locally stored on participant devices and emphasizes on conserving participants data privacy. With a centralized coordination, the training sessions are coordinated by a central server that selects participants, schedule training tasks, receives model updates from participants and applies model updates to a global AI model and pushes the global model to participants again for local training. This procedure is repeated until the minimization of a loss function converges or the desired accuracy is achieved [13]. When coordination is desentralized, every node communicated through a P2P network, plays the roles of server and client, i.e., train with local data and apply updates to the model it receives from its neighbors.

FL can be also classified in the dimension of data partitioning [14] which refers to how training data is distributed over the samples and feature spaces. It can be Horizontal Federated Learning (HFL) when all participants contribute to training with local data in the same feature space but different sample space. Conversely, Vertical Federated Learning (VFL) is called when all participant contribute to training with the same sample space but over different feature spaces. In the third place, Federated Transfer Learning (FTL) is when participants contribute with different sample and feature spaces.

Irrespective of the the FL classification, local training is performed with data and computing resources of participants devices. Training tasks are expected interfere the least with user's experience and causing the least to none monetary costs to the device owner, e.g., due to use of expensive and metered networks. For this reason, local training is scheduled to be performed when

Table 1: Intelligence at the edge: Overview

the device is charging and connected to WiFi [13]. These execution conditions, in sum to the fact that computing power differs between participants due heterogeneous hardware specs, and result updates might suffer from delays due to intermittent connection availability, training tasks fit into the category of delay tolerant batch processing tasks more than latency-sensitive real time tasks.

2.1.2 Distributed inference

Make inferences is the task of predicting or classifying input data automatically by means of a trained model. In edge computing aims at not only preserve data privacy by avoiding the transport and storage of data in Cloud servers but to perform inferences fast enought to support decisions in realtime or near realtime. However, inferences are resource intensive tasks for edge devices with limited computing capabilities and/or that operate with batteries. Problems associated with the execution of complex models in these devices relate to, on one hand, models size which commonly exceeds the memory available in edge devices. Compress models, prunning [10, 9], quantization [5, 7] and knowledge distillation are among the most commonly applied techniques for mitigating the model size problem.

On the other hand, there is also an issue directly related to the constrained resources of edge devices for executing the amount of computing operations these models require to make inferences. In this regard, Split Computing (SC) aims at mitigate the need for resources by distributing the computational cost among several devices. The technique consist in splitting model layers vertically and assign groups of them to different devices. In this way, the computation of an inference is achieved by all devices in kind of pipeline. For instance, with a split operation that dissects a model into head and tail groups of layers, a device assigned with the head layers receive raw input and produce intermediate outputs for activating the tail layers that are executed by other device which will produce the final inference result. The split operation can be without or with DNN arquitecture modification. In the first case, split operation relies on natural bottlenecks inside the model, that is when intermediate layers whose output tensor size is smaller than the input. This is the case of DNN architectures like AlexNet, VGG and DenseNet. For other architectures, with different anatomy like ResNet, Inception-V3, Faster R-CNN and Mask R-CNN, the application of SC requires from the injection of artificial bottlenecks, e.g., through encoder layers. This case is refered as SC with DNN modification. The objective is to minimize the data size transferred from edge devices to edge servers and maintain as lightweight as possible the model part executed by the edge device. Early Exit (EE) is other strategy for speeding up inferences. In few words, EE consist in preparing a model to have multiple "exits" or intermediate results. For a giving input, the model is able to produce outputs at different times with different accuracy. The computation of an inference ends when the provided output meets a desired accuracy level threshold, i.e., an acceptable result can be an intermediate, not necessary the final, result. In turn, the final result calculation can be aborted giving the possiblity to shorten the time in which a result is obtained. A comprehensive and detailed analysis of different techniques for SC and EE can be found in [12]. Despite many works made source code available for validity and results reproduction, the study highlight as an important future challenge the need for standarized experimetation settings and make them more practical since wireless communication, computation enviroment, models and datasets are of paramount importance to asses the convenience of applying different SC and EE techniques.

In [1], the concept of Hierarchical Inference (HI) is presented where collaboration between resource constrained edge devices and edge or cloud servers is driven by the complexity of data samples used as input for making inferences with different DL models versions a tiny version locally executed by the edge device and a large version running at edge or cloud servers-. There is a decision-making module that decides whether to perform a data offloading and solve the inference remotely or solve the inference locally with the tiny version. Authors warn about some

considerations for effectively apply HI including that the energy cost of computing an inference locally should be less than transmiting a data sample, the accuracy of the tiny model should be reasonable for simple data and the fraction of simple data should be greater than that of complex data samples. Last but not least, it should be clearly defined how to differentiate simple and complex data samples.

A common denominator of all abovementioned techniques is distributed inference considering collaboration between edge devices and edge servers or even cloud servers. Few works study the collaboration between edge devices.

2.2 Tools and mechanisms for enabling distributed mobile computing

For exploiting computing capabilities from edge nodes such as smartphones and SBCs, it is required a series of tools and mechanisms, most of which are presented as middleware services in high power distributed computing platforms such as computer clusters and datacenters. In this section we describe efforts in this regard.

Job Scheduling and Load Balancing: [Hirsch][Locke][Sahh]
Incentive mechanisms:
Nodes Discovery: [17]
Work logic update and deployment:
Testing platform: [Cmateos][Toloza]

3 System model & assumptions

We envision an autonomic Dew Computing scheme for dynamic load execution where input adquisition, filtering, processing are associated to AI tasks performed with components residing within the boundary of a wire(less) local area network and with independency of Cloud and/or Fog services. Generically, we can describe the Dew Computing scheme with components that play at least one of three roles named data acquirer role, data concentrator role and intelligence provider role. Components with the data acquirer role relay raw data sensed to a component with the role of data concentrator whose main duty is preprocessing raw data -buffer, filter, denoise, segment it-. Moreover, data concentrator components feed intelligence providers with the necessary preprocesed data to allow the latter to apply intelligence via AI algorithms, e.g, a deep learning model, to convert the preprocessed data it into valuable information for the purpose the Dew Computing scheme operates for.

In an attempt to establish an association of physical components and roles, we could say that data acquirer role is attributed to physical devices such as CMOS sensors, microphones, thermometers, vital signs sensors, in general, devices used to make observations or capture data in any format (images, text, sound). Data concentrator, by contrast, can be associated to devices with storaging capability, but also with minimum computing capability that enable it to run data preprocessing functions including data type convertion, formatting, frame extraction, data slicing, and also middleware tasks such as job creation, job distribution and result collection. Finally, the intelligence provider role is mapped to devices with enough computing capability to apply AI algorithms to the received preprocessed data. Considering that commodity hardware in the form of edge and end-user devices are first-class citizen in Dew Computing, intelligent provider role can be directly mapped to Single Board Computers (SBC), smartwatches, and the so popular smartphones. In this sense, a smart camera that tags frames to help humans to detect security threats and actuate in consequence, or the "Ok Google" words recognition by a smartphone that indicates the device to listen for voice commands from the user, are materializations of Dew Computing schemes where all roles are played by a single device. However, in this work, we propose an evaluation of Dew Computing power for performing tasks that exceeds the computing capability of a single resource-constrained device.

In a master-worker architecture, nodes with the role of Data concentrator would be also associated to master node responsabilities while worker nodes would be nodes contributing with

Edge Node Type	Edge Node Model	Processor	RAM	Conectivity	Avg. Price (U$D)
SBC	Raspberry Pi 4	Quad-core 1.5 GHz ARM Cortex-A72	4 GB LPDDR4 SDRAM	Wi-Fi 802.11ac	55
	Jetson Nano	Quad-core ARM Cortex-A57 MPCore	4 GB 64-bit LPDDR4	Gigabit Ethernet, M.2 Key E	99
	Gigabyte Brix GB-BRi5H-8250	Intel Quad core i5-8250U	8 GB SO-DIMM DDR4	IEEE 802.11ac, Dual Band Wi-Fi	500
Smartphone	Samsung A02	Quad-core 1.5 GHz Cortex-A53	2 GB	Wi-Fi 802.11 b/g/n	116
	Motorola Moto G6	Octa-core 1.8 GHz Cortex-A53	3 GB	Wi-Fi 802.11 a/b/g/n	160
	Samsung A30	Octa-core (2x1.8 GHz Cortex-A73 & 6x1.6 GHz Cortex-A53)	3 GB	Wi-Fi 802.11 a/b/g/n/ac	170
	Xiaomi Redmi Note 7	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	200
	Motorola Moto G9 Play	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	200

Table 2: Edge computing candidate nodes

computing resources for providing intelligence support.

4 Empirical evaluation

We benchmarked the energy efficiency and computing capability of resource contrained edge nodes formed by groups of smartphones cooperating under the same wireless LAN and coordinated by a proxy that we compared with the performance achieved through edge nodes of type SBC commonly used for servicing IoT and edge applications.

Table 2 shows hardware details of resource contrained edge nodes used in the benchmark tests. Single board computers (SBC) are assumed to operate plugged to the electric power grid, i.e., potentially with infinity energy, while smartphones are assumed to exclusively use their batteries as main power source. All nodes have multicore processors (SBC with quad-cores and most of smartphones with octa-cores), and 802.11 WiFi radio that theoretically is able to reach bandwidth between hundreds of Mbps (norm n) to few Gbps (norm ac). Edge nodes main memory ranges from 2 to 8 GBs. As additional information, it is also included an estimated per unit adquisition cost in american dollars. When searching for cost-effective Dew configurations, the cost can be used as indicator to untie situations where several nodes achieved the desired throughput.

As computing workload, we consider dynamic jobs (stream processing), concretely images streams that can be associated to machine vision tasks. We assumed latency-sensitive application scenarios of different domains that require real time response where the latter derived from inferences made with pre-trained deep learning models.

For setting up and executing benchmark tests, comprising tasks such as cluster formation, charge/discharge devices battery, deep learning models deployment, stream dynamics repetition, resuts collection and sumarization, we employed a tool set that combines open source software [6, 22] and hardware [20] modules designed and developed by our team that allow us to perform and

repeat in-vitro evaluations, i.e., using real mobile devices, to compute real jobs with full control over physical devices and experimental variables such as battery level, devices externally injected and user load, wifi signal strenght, among others.

The metrics we used to compare both Dew Computing approaches are: processing delay (w.r.t stream duration) and total energy consumption. Additionally for tests involving resource provision with a cluster of mobile devices, we report fairness to give insights of intra-cluster energy utilization.

Section 4.1 elaborates on the deep learning models characteristics considered in experiments. In section 4.2 we describe characteristics of dynamic jobs associated to different domains and study cases. Finally, Section 4.3 presents the benchmark results.

4.1 DNN models benchmarking

Benchmarking is a relevant task to approximate nodes performance in solving a specific task and this is what we did for nodes in Table 2, where task refers to make inferences using different tensorflow DNN models. Tensorflow project[1] offers a benchmarking tool to collect statistical information of a `.tflite` model. The tool measures the time a model takes to produce an inference, memory consumed among other information using synthetic input randomly generated. It worth mention that all inferences times reported in this paper are on the basis of using CPU support. Even though today there is especialized hardware, sometimes referred as AI hardware accelerators to run AI logic faster than with general purpose hardware like CPU, there are some practical limitations including library support for different platforms and propietary chips embeed only in high-end smartphones of certain brands that ralentless the massively use of Dew Computing as we envision.

The notion of real time is context dependent and depending on the application an ever-present time delay of a few seconds satisfies the notion of real time. For this reason and with the aim of evaluating Dew Computing capabilities for potential different application scenarios, i.e., with different real-time notions, we used three DNN models that between them present quite disimil inferences times. It not part of the evaluation core to test DNN models in their most efficient version, i.e., we do not aim to test and improve models performance per se, although in a real implementation it is relevant to considered. Conversely, our objective is to compare the execution capabilities of edge devices and distributed systems composed by collaborating end user devices.

All models used in benchmark tests 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. One of the models is the widely known, publicly available YoloV4[2] used to perform object detection. Figure 1b shows average inference times (in miliseconds) for different nodes when performing object detection using a YoloV4-tiny model ported to Tensorflow lite. 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 which doubles the capability of a RaspberryPi4, but is slower than a Gigabyte Brix which reaches nearly 10 FPS. Other of the benchmarked models used in experiments was proposed to make milk cows scoring to differentiate healthy and not healthy cow groups. The model encapsulate expert knowledge on the determination of what is known as Body Condition Score (BCS) [2]. Figure 2b shows the BCS average inference time that different edge nodes achieve. For instance, the Xiaomi RN7 smartphone completes an inference in around 184 milliseconds, i.e. is able to deliver barely more than 5 FPS, while a Gigabyte Brix node is around 15 FPS. The third model used was trained to recognize diabetes grade using foot images as input. Figure 3b shows the inferences times obtain by different edge nodes. By comparing the fastest smartphone with fastest SBC it can be noted that Xiaomi RN7 completes an inference in around 1470 milliseconds while a Gigabyte Brix does the same in approximately 559 milliseconds. When comparing performance of DNN models, notice that for most of the edge nodes benchmarks, inference times differ in at least one order

[1] `https://www.tensorflow.org/lite/performance/measurement`
[2] https://github.com/hunglc007/tensorflow-yolov4-tflite

of magnitude. As said, a Xiaomi RN7 completes inferences in 248, 184 and 1470 milliseconds using yolov4 model, BCS model and diabetes foot model respectively. These differences were intentionally searched to test Dew Computing capabilities in heterogeneous computing scenarios. Such heterogeneity resides in models anatomy. Table 3 shows layers count discriminated by type and total number of layers each model presents.

4.2 Workload characterization

Based on the fact that deep learning models of section 4.1 use images as input, we consider workloads that can be associated to machine vision tasks and depend on a stream-like input. Machine vision tasks such as object detection and classification using deep learning are commonly applied in diverse domains. However, due to imperfections on the training data, input data capaturing devices, problem specific constrains like tracking or identify moving subjects, etc., make machine vision systems to produce outputs based on multiple frames more than on a single shot frame which is used to disambiguate erroneous inferences and improve output accuracy [18]. To complete the experimental setting of our dew computing scenario, we consider video frames streams printing different workload to the execution system.

Without loosing generality, we associate each model-stream pair a different domains/usage contexts where Dew Computing could be applied. These domains are: a smart city application scenario that we call "Sense while Travel", a dairy farm application scenario called "Body Condition Score" and a health monitoring application scenario called "E-health early diagnosis". Next we describe the workload dynamics of each scenario:

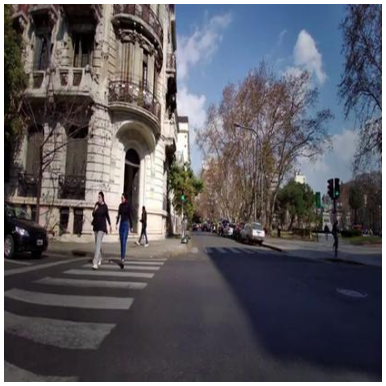
- Sense while Travel (SWT): a city can take advantage of mobile agents such as urban line buses to collect information of relevant street events while traveling, e.g., for statistical purposes. Images captured with a bus front camera might feed a YOLO v4 model which, as output, gives a plain text representation of objects class found in an image and the accuracy percentage of each detected object class. An energy eficient 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. Without loosing generality, we state this sample rate in 2 FPS and evaluate the system for a stream of 30 minutes lenght. These parameters equates approximately to have 15 km of sensed data when the stream is produced by a vehicle travelling at 30 km/h.

- Body Condition Score (BCS): in a dairy farm, outfitted cows, i.e., cows that are either overweighted or underweighted tend to produce less milk than those properly weighted. Identify such animals to give them a proper treatment is crucial to maintain the cow roundup fully productive. To identify such animals a DNN model is used which takes depth images taken from top of a cow with a camera strategically positioned [2]. The image capturing procedure should be performed within a time window that not exceceds 10 seconds. During such time, providing that animals naturally don't state quite under the camera and the capture should contain a specific part of the cow, a high percent of captured images, e.g. a 50%, is expected to be have to be tagged and filtered as not useful input for the DNN 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 proper to stop capturing images. To make the body condition score calculus feasible, realiable, and energy effienct it is proper to count with no less than a hundred of frames ready to serve as input for the DNN model, from each cow. All these constraints configure a stream processing scenario where depth images can be produced at a sample rate of 15 FPS during 10 seconds, followed by other 10 seconds where none images are captured.

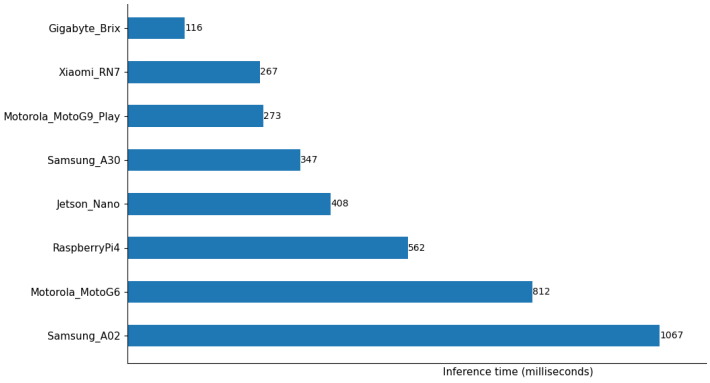
- E-health early diagnosis (EED): periodically, older people in a nursing home subject themselves to control different health parameters as a way to prevent the appearance of some pathology or monitoring a pre-existence illnees. We envision an advanced e-health scenario

Operator	YoloV4	BCS	Diabetes
CONV_2D	21	26	128
MAX_POOL_2D	3	3	0
CONCATENATION	17	8	0
RELU	0	8	0
MEAN	0	1	33
SOFTMAX	0	1	0
LOGISTIC	0	0	128
MUL	24	0	130
DEPTHWISE_CONV_2D	0	0	32
PAD	2	0	5
ADD	6	0	25
SUB	6	0	1
RESHAPE	18	0	32
STRIDED_SLICE	0	0	32
PACK	0	0	32
FULLY_CONNECTED	0	0	1
SHAPE	0	0	32
LEAKY_RELU	19	0	0
SPLIT	3	0	0
LOGISTIC	12	0	0
RESIZE_BILINEAR	1	0	0
SPLIT_V	2	0	0
STRIDED_SLICE	12	0	0
EXP	6	0	0
Total layers number	152	47	610

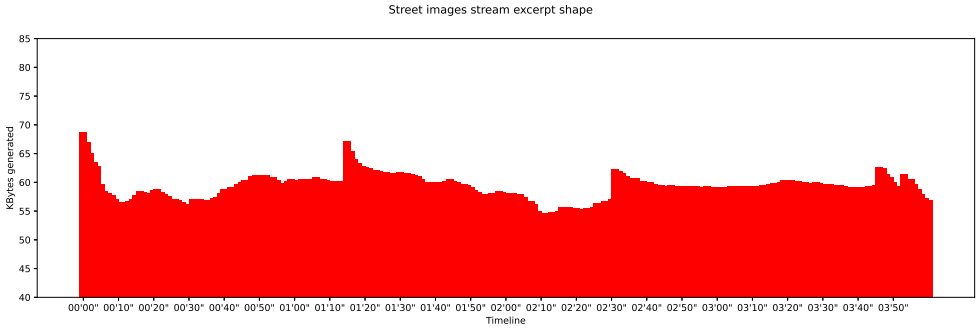
Table 3: Models outline



(a) street image example



(b) YoloV4 tiny DNN model benchmarking



(c) Street images stream

Figure 1: SWT - workload characterization

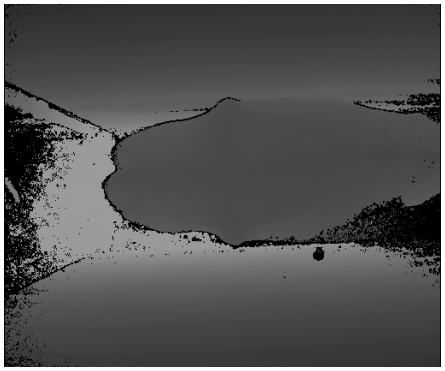
where a DNN model encapsulates medicine expert knowledge to allow, e.g. diabetes diagnosis using foot images., Imaging an encounter of a nurse in which as part of the healthcare service consist in taking pictures in burst of all patients feet resulting in a stream of images with the a shape of 10 FPS during 2 seconds followed by 7 seconds without images capturing.

The dynamics described above are Figures 2c and 2a on one side, and figures 1c and 1a on the other side, show a timeline representation of the data kilobytes produced by a stream and an example of the DNN input image for Sense While Travel and Body Condition Score Calculus scenarios respectively.

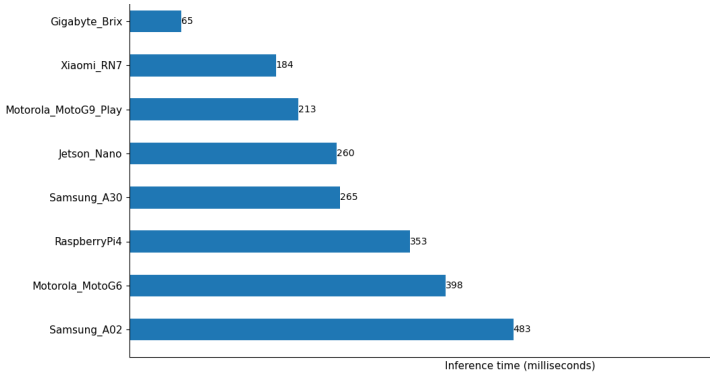
Irrespective of the application domain, the three examples represent streams with different characteristics not only by the dynamics of input image generation but also the computing power required to perform inferences. In addition, it is assumed that inferences results should be available at the master node with minimun delay, i.e., as close as possible to a real time detection, which is a characteristic of computing services provided at edge.

4.3 Benchmark results

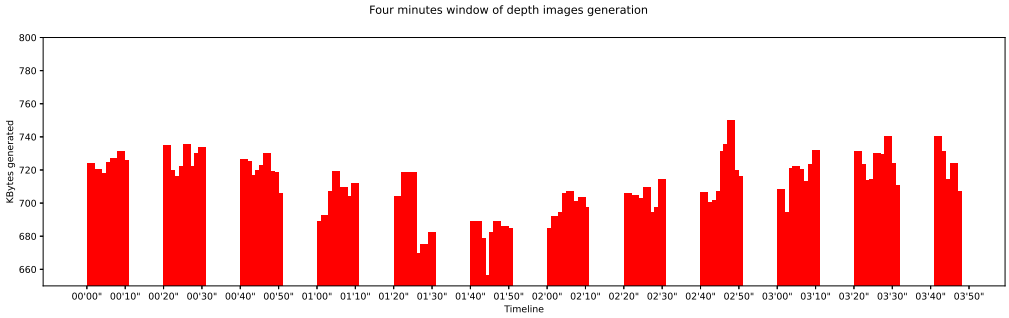
In all scenarios, the best makespan was obtained when using a pull-based load balancing scheme. With such scheme, as soon as a worker node finishes a work unit and the corresponding result is received by the master, the worker node is able to pull another work units from a common queue. The work completion delivering rate adapts to the job completion rate of participating nodes.



(a) Cow image example



(b) Cow BCS DNN model benchmarking



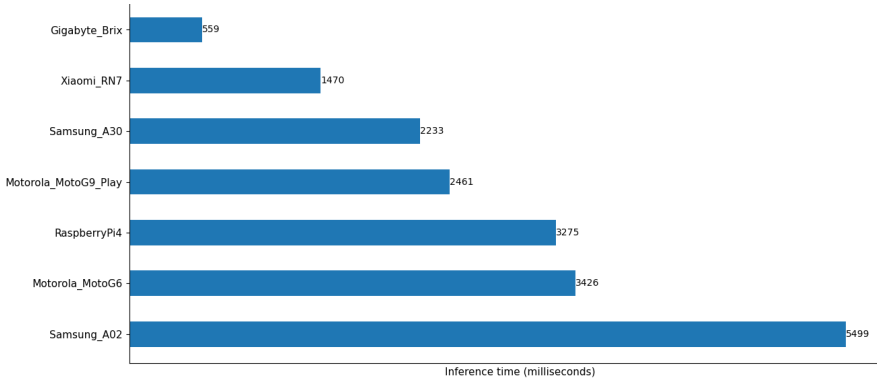
(c) Cow images stream

Figure 2: BCS - workload characterization

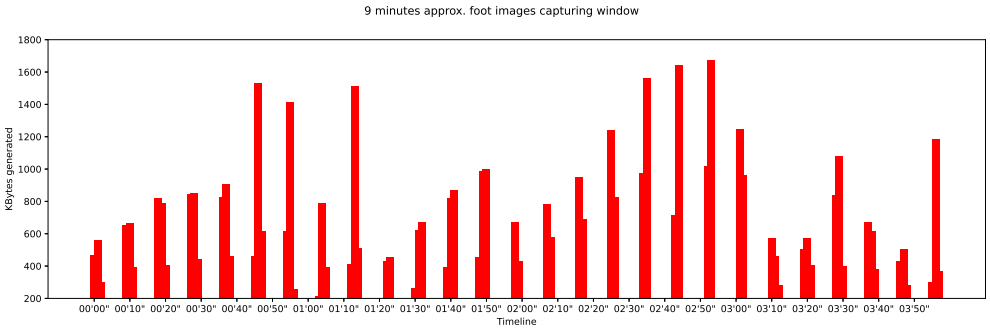
Edge Node	Max. Batt. (mAh)	Init. Energy (%)	Load Balancing	Makespan (mins)	Jain's Fairness Index	Consumed Energy (%)
Raspberry pi 4	∞	N/A	N/A	63.2	N/A	N/A
Jetson Nano	∞	N/A	N/A	34.27	N/A	N/A
Gigabyte Brix	∞	N/A	N/A	**30.14**	N/A	N/A
Cluster 202	9000	80.33	Round Robin	45.28	0.924	6.77
			Pull Based	44.66	0.924	6.77
Cluster 302	13800	45.35	Round Robin	41.53	0.933	4.63
			Pull Based	32.34	**0.96**	3.92
Cluster 301	12900	45.37	Round Robin	30.76	**0.96**	3.92
			Pull Based	30.42	0.896	3.53
Cluster 408	15900	42.47	Round Robin	30.41	**0.907**	3.31
			Pull Based	30.24	0.818	2.80

Table 4: Sense while travel scenario (30 minutes stream)

(a) Foot image example



(b) Foot image DNN model benchmarking



(c) Foot images stream

Figure 3: EED - workload characterization

Edge Node	Max. Energy (mAh)	Init. Energy (%)	Load Balancing	Makespan (mins)	Jain's Fairness Index	Consumed Energy (%)
Raspberry pi 4	∞	N/A	N/A	26.92	N/A	N/A
Jetson Nano	∞	N/A	N/A		N/A	N/A
Gigabyte Brix	∞	N/A	N/A	**3.9**	N/A	N/A
Cluster 304	12000	49.4	Round Robin	6.81	0.667	volver a correr
			Pull Based	4.96	0.926	1.58
Cluster 409	15900	61.2	Round Robin	5.06	0.75	0.68
			Pull Based	3.96	0.75	0.68

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

Edge Node	Max. Energy (mAh)	Init. Energy (%)	Load Balancing	Makespan (mins)	Jain's Fairness Index	Consumed Energy (%)
Raspberry pi 4	∞	N/A	N/A	27.72	N/A	N/A
Jetson Nano	∞	N/A	N/A		N/A	N/A
Gigabyte Brix	∞	N/A	N/A	**8.99**	N/A	N/A
Cluster 305	11900	54.1	Round Robin	33.67	0.857	1.91
			Pull Based	12.84	0.926	1.58
Cluster 411	15900	40.5	Round Robin	25.24	0.833	2.3
			Pull Based	9.34	0.643	1.4

Table 6: E-health early detection scenario (08:55 minutes stream)

5 Conclusions and Future works

In this work we explored the capabilities of Dew Computing for performing real-time inferences using three deep learning models and stream-like workloads. Puntually, we tested and compare the performance obtained by several Dew Computing settings built with commodity hardware including SBCs and smartphone clusters. Some of the conclusions that we arrive at are:

- Makespan attained with tiny clusters composed by 3-4 middle-range smartphones using a pull-based execution scheme is competitive compared to the computing capability of an SBC equiped with an intel core i5 8th generation in delivering near real-time inferences. It suggests that to satisfy certain delay sensitive application scenarios, the exploitation of already-in-place smartphones clusters should be analyzed and considered before the investment and deployment of SBCs.

- A pull-based scheme shows better makespan than the classical Round Robin for intra-cluster load balancing load. However, in some scenarios with apparently plentyfull computing resources to satify the computing demand, e.g., cluster 301 y cluster 408 with SWT scenario, Round Robin behaved competitive and also achieves better fairness score than a pull-based scheme. In other words, due to resources abundance, the addition of more resources seems to be not translated into a makespan reduction but into the improvement score of other metric, which reveal that there is more room for further exploration of load balancing schemes to target the best balance between different metrics, e.g., makespan, fairness, global remaining energy, parallelism level, etc.

- Despite differences observed in inference times obtained via tensorflow benchmarking tool and results collected in experiments using real input, when cross comparing model performance, relative positions between edge nodes are almost the same, meaning that it is feasible to derive a model agnostic nodes ranking to help a load balancing component to decide how to distribute jobs based on nodes performance.

As future works we visualize several points of improvements. In line with developing new load balancing heuristics, and based on results of preliminary runs we plan to improve previously published heuristics that use MFLOPs to rank nodes computing capability. An alternative is to use information derived from Tensorflow benchmarking tool. Other future direction is to consider other forms of workload distributions, i.e., that exploit internals of DNN arquitectures, for example, split computing, early exiting techniques, model quantization; other forms of workload execution, e.g, using flags that reduce the inference time, e.g, make inferences over a batch of images, or accelerators such as the XNNPACK support to better exploit CPU capabilities of mobile platforms while performing network inferences.

Other improvements to the distributed execution system prototype made up of end user devices are in the line of refinining aspects such as:

- Workers full exploitation: as CPU profiles of experiments run reveal, CPU utilization is not full.

Even though flags, such as XNNPACK, ncnn, are presented as accelerators to better exploit CPU capabilities, the technology is applied only to tensorflow framework. A library agnostic solution that consider not only better exploitation of CPU but also maximum CPU usage configured by end users would signify another relevant step towards a middleware that make Dew Computing a reality.

- Improve workers exploitation by considering model anatomy and workers capability: for prediction and/or classification tasks using deep learning, one or several models can be utilized. Each model migh differ in their number of layers. The model, in turn, can be splitted into different groups of layers where each group can be assigned to nodes with different computing and/or transferring capabilities in order to improve the inference time. Similar point of improvements can be thought for the application of ensemble learning techniques where the final output to a prediction or classification task is built by averaging partial outputs of different weak models. In this sense, and considering that weak models differ in their anatomy and output quality, heavy models can be deployed in high capable devices, while lighter models on low capable devices. How to rate a device as high or low capable is challenging due to the, device connection unstability, heterogeneous hardware, owner usage, i.e., the high dynamic nature of device clusters.

- Input/output transfer latency: workers request input images to a master which, in turn, pull them from an HDD. Such decision was made as part of as experimental platform. However, migrate the input provisioning in the master node to a more on-the-fly scheme that uses main memory instead of HDD to buffering images might considerable reduce latency in the input provisioning. Naturally, trade offs between main memory amount and reduced latency will emerge.

- Wireless medium optimization: wireless medium usage can be optimized since images and results are currently transported through a two hop wireless network, i.e., a router mediates the communication between master and workers nodes. It means that bytes are wirelessly transferred from the master to the router and from the router to the workers in the case of inputs, or in the other way around for outputs. Whenever possible, such data transferring scheme could be sinplified and wireless communication reduced to a single hop. One posibility is to perform the master role and the sensing task within the same physical node or in separate nodes but communicated through a high speed wired connection.

References

[1] Ghina Al-Atat, Andrea Fresa, Adarsh Prasad Behera, Vishnu Narayanan Moothedath, James Gross, and Jaya Prakash Champati. The case for hierarchical deep learning inference at the network edge. In *Proceedings of the 1st International Workshop on Networked AI Systems*, pages 1–6, 2023.

[2] 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.

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

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

[5] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[6] Matías Hirsch, Cristian Mateos, Alejandro Zunino, and Juan Toloza. A platform for automating battery-driven batch benchmarking and profiling of android-based mobile devices. *Simulation Modelling Practice and Theory*, 109:102266, 2021.

[7] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[8] 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.

[9] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2016.

[10] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2018.

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

[12] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys*, 55(5):1–30, 2022.

[13] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[14] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, and H Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658, 2021.

[15] 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.

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

[17] Ahmed Salem, Theodoros Salonidis, Nirmit Desai, and Tamer Nadeem. Kinaara: Distributed discovery and allocation of mobile edge resources. In *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 153–161. IEEE, 2017.

[18] Harisu Abdullahi Shehu, Will Browne, and Hedwig Eisenbarth. Emotion categorization from video-frame images using a novel sequential voting technique. In George Bebis, Zhaozheng Yin, Edward Kim, Jan Bender, Kartic Subr, Bum Chul Kwon, Jian Zhao, Denis Kalkofen, and George Baciu, editors, *Advances in Visual Computing*, pages 618–632, Cham, 2020. Springer International Publishing.

[19] 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.

[20] 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.

[21] 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.

[22] Virginia Yannibelli, Matías Hirsch, Juan Toloza, Tim A Majchrzak, Alejandro Zunino, and Cristian Mateos. Speeding up smartphone-based dew computing: In vivo experiments setup via an evolutionary algorithm. *Sensors*, 23(3):1388, 2023.

[23] 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.