Distributed Edge Inference: an Experimental Study on Multiview Detection

Gianluca Mittone
gianluca.mittone@unito.it
University of Turin
Turin, Piedmont, Italy

Giulio Malenza
giulio.malenza@unito.it
University of Turin
Turin, Piedmont, Italy

Marco Aldinucci
marco.aldinucci@unito.it
University of Turin
Turin, Piedmont, Italy

Robert Birke
robert.birke@unito.it
University of Turin
Turin, Piedmont, Italy

ABSTRACT

Computing is evolving rapidly to cater to the increasing demand for sophisticated services, and Cloud computing lays a solid foundation for flexible on-demand provisioning. However, as the size of applications grows, the centralised client-server approach used by Cloud computing increasingly limits the applications' scalability. To achieve ultra-scalability, cloud/edge/fog computing converges into the compute continuum, completely decentralising the infrastructure to encompass universal, pervasive resources. The compute continuum makes devising applications benefitting from this complex environment a challenging research problem. We put the opportunities the compute continuum offers to the test through a real-world multi-view detection model (MvDet) implemented with the FastFL C/C++ high-performance edge inference framework. Computational performance is discussed considering many experimental scenarios, encompassing different edge computational capabilities and network bandwidths. We obtain up to 1.92x speedup in inference time over a centralised solution using the same devices.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Distributed computing methodologies**; **Distributed artificial intelligence**; *Scene understanding*; • **Networks** → *Network experimentation.*

KEYWORDS

Edge Inference, Edge Computing, Computing Continuum, Computational Performance, Network Performance

1 INTRODUCTION

Recent machine learning techniques advancements lead to models reaching (and even surpassing) human capabilities on many tasks, fuelling a proliferation of AI-based applications. These applications mainly leverage the flexible computing and network resources cloud computing provides to achieve their scalability goals [34]. However, as the number of users and the complexity of deep neural network (DNN) models grows, the centralised approach taken by cloud computing is quickly reaching its limit. Especially AI-based applications with high resource costs per user request, such as transformer-based large language models, e.g. ChatGPT [27], and generative diffusion models, e.g. DallE [26], put increasing strains on the cloud datacenters. New approaches try to alleviate this scalability bottleneck by gradually decentralising the infrastructure via "walking-size" cloud-connected server rooms pervasively distributed in the environment (edge computing) and the resources along the network paths (fog computing) up to a fully decentralised infrastructure (compute continuum) [16]. The complexity stemming from such an environment's sheer size and heterogeneity makes designing applications able to take full benefits extremely challenging.

Traditionally, most "smart" applications leverage the classic client-server model where the user interacts with an app or a web API to send requests processed by (possibly clustered) servers in the cloud. On the one hand, the cloud can offer performance unreachable by smaller devices available in server rooms or even more at the edge. On the other hand, an increasing number of existing edge devices, like Raspberry Pi[23] and most smart home devices [37], still offer enough computing power to handle inference of deep models, either unmodified or distilled [11]. However, most edge inference-related art considers either the single-device scenario, trying to optimise the execution of one or multiple DNNs on constrained devices, or the client-server scenario, which offloads heavy computations to the cloud. Few broaden the view to the possibilities offered by more decentralised infrastructures. We aim to fill this gap via an initial case study applied to a multiview detection system.

In this work, we propose a real-world implementation of a multiview detection system based on the MVDet state-of-the-art detection model [14]. We port the model to C++ using the libtorch [24] and opencv [6] libraries and integrate it into a framework for distributed inference. We experiment with it under different computational and networking hypotheses, comparing the traditional client-server approach to a distributed approach, which splits the model execution across multiple devices to test the real benefits of a decentralised infrastructure. Our contributions are:

- an open-source implementation of a real-world edge-based system for multiview detection. While our implementation is based on the MVdet model, the methodology can be applied to different models;
- extensive experimentation of the proposed system under different computational and network conditions;
- a critical discussion on the potentiality of decentralised infrastructure.

The paper is structured as follows: Section 1 briefly introduces the research context, Section 2 introduces the basics for understanding the subsequent sections, Section 3 details the multiview detection use-case, the implementation methodology and the research choices, Section 4 discusses in detail the experimental contribution of this work, and Section 5 wraps up the exposition, summarising the findings and future research directions.

2 BACKGROUND

Edge inference is a research field undergoing strong experimental efforts due to the convergence of many phenomena. The abundance of pervasive computational devices [20] offers lots of spare computational power exactly where data are harvested. Many approaches exploit this latent computing power to process data timely, especially for medical purposes [12, 25]. However, as ML models became increasingly complex and computationally power-consuming, edge devices adapted like in a co-evolution schema [33]. Thus, AI-specialised edge devices have flourished, like TPUs [28], NPUs [31], FPGAs [10], and many others based on innovative computing paradigms, like in-memory computing [15]. All this hardware-related research effort is devoted to increment the efficiency of AI-related computations, trying to mitigate as much as possible the natural computing and battery life limitation of such devices, but many real-world still incorporate only CPUs as compute due to cost, thermal and power constraints [4]. On the other hand, ML practitioners devised new algorithms to exploit edge opportunities. Some efforts, like model compression [29], pruning [36], and distillation [19], focus on reducing as much as possible the model size to improve both its memory and energy impact. Conversely, model partitioning aims at dividing models into manageable chunks to optimise scheduling on single devices [8] or to offload parts of computation across multiple computing devices according to the capabilities of each one [5, 22, 35], improving inference latency. Still, another strategy is to completely avoid using DNNs in favour of simpler, less computationally intensive models such as classical, non-deep ML models. Despite, the feasibility of this approach in an edge environment [17], not all tasks are suited to these models.

ML-based detection systems can be modelled in many ways, but reliance on DNNs is one constant. Single-view detection [13, 21, 38, 39] addresses the case in which just a single image of a scenario is available for inference at a time. Such methods can be anchor-based or not and can handle occlusion issues by exploiting techniques such as part-recognition, non-maximal suppression, and repulsion loss. Additional information exploited by the inference can be domain-specific, like the detection of heads and feet when searching for people, or can be obtained through the use of particular image acquisition tools, such as RGB-D cameras and LIDAR detectors, to obtain single images correlated with more spacial information. Multiview detection exploits multiple sources of image acquisition to produce a single inference [9, 14, 30] to overcome occlusion problems. The principal research focus in this scenario is aggregating the information retrieved from multiple data sources. Popular approaches target combining multiple single-view detection, aggregating the features extracted from each image, and applying geometrical transformation to the camera outputs.

3 METHODOLOGY

The MVDet is a state-of-the-art multiview detector that identifies persons standing and moving across an open public area. The main idea is to use a trained base model, e.g. ResNet18 or VGG11, to extract the features from each camera frame and transform it via a homography, i.e. perspective warp, projecting it to the bird eye view of the area. The results are then fed into an aggregation model, which detects the position of all the persons in the area.

Bringing the MVDet model to a real-world edge environment requires methodological and implementational choices. Starting from the original MVDet code, we identify two different computational stages: a parallelisable one that comprehends the frame acquisition, feature extraction, and perspective warping, and, conversely, a sequential one that is the multiview aggregation. Since the system is synchronous at the frame level, the multiview aggregation is data-dependant from the previous processing steps: the aggregation can not start until all the camera frames from the current time step have been acquired and processed. Given this computational scenario, we propose two different edge implementations of the MVDet system: a more distributed one and a more centralised one. The distributed implementation is depicted in Figure 1. It takes full advantage of the model partitioning technique, allocating part of the ML models on the edge devices while trying to parallelise the execution as much as possible by assigning all the parallelisable code to a different camera. Conversely, the centralised implementation takes full advantage of model offloading, allocating all the computational burden on the server while requiring the Camera to acquire the frames. A more technical discussion on the differences between these two approaches is presented in Section 4.1.

Due to the targeted scenario, i.e. edge devices, we stumbled upon the fact that no Distributed ML (DML) framework currently available on the market is designed to be energy-efficient and computationally lightweight. The commercial software is Python-based, requiring computational and memory capabilities that not every edge device can afford. Furthermore, popular DML software is being developed keeping into account mostly user-friendliness and additional security features, such as Homomorphic Encryption and Secure Multiparty Computation, rather than computational performance. While this strategy comes in handy when dealing with powerful computing devices, an edge-oriented DML framework should care about the amount of resources it consumes (computation, memory, energy), trying to be as efficient as possible. Also, most commercial DML software are designed to handle in a very solid way only one communication topology, the master-worker one. Any other communication structure would require heavy software modifications, resulting in a not-as-intended use of the frameworks. The tree-based inference structure we envision is thus not implementable straightforwardly in current DML software. Due to these two motivations, efficiency and communication topology malleability, we use an experimental DML framework named FastFederatedLearning (FastFL) [18]. FastFL offers a high-performance C/C++ implementation wrapped with a user-friendly Python interface, allowing user-friendliness and computational performance. The communication backend is handled by the FastFlow

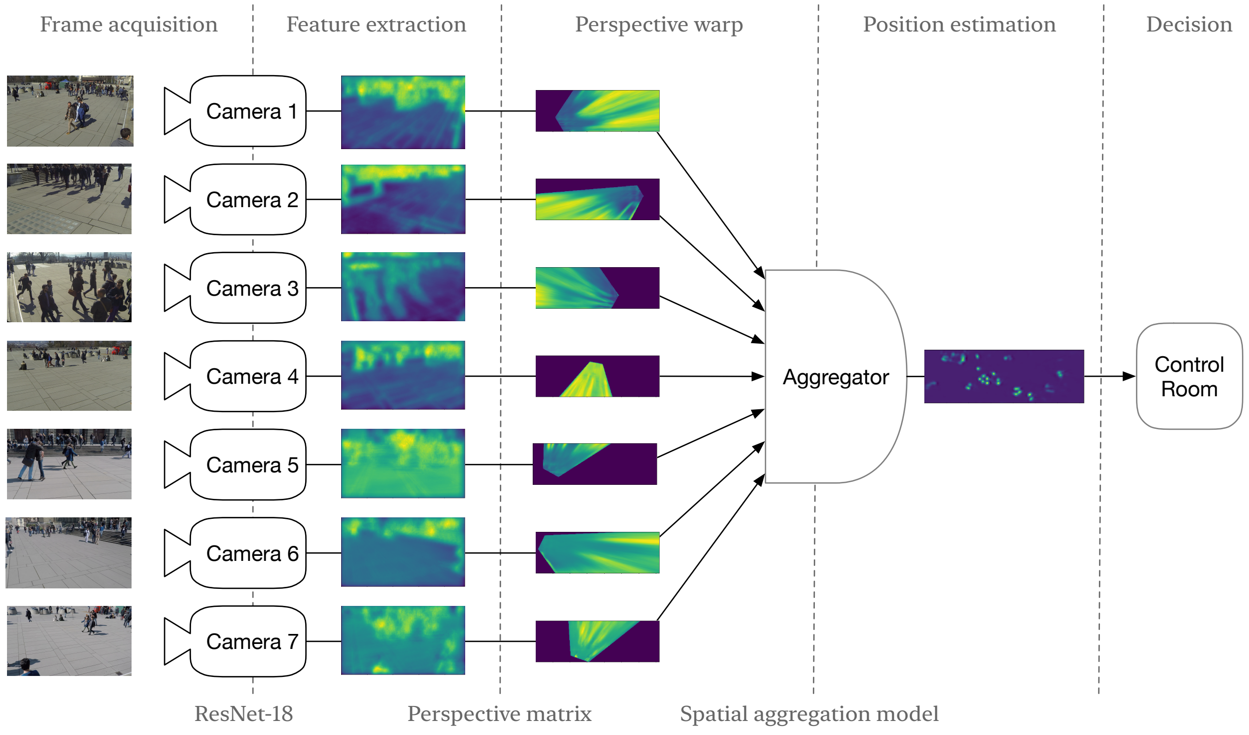

Figure 1: Distributed implementation of MVDet with FastFL. The workflow starts with each camera acquiring the current time step frame; the frame features are then extracted by a ResNet18 and warped according to the camera perspective matrix directly on the edge; then all the warped feature maps are collected by the aggregator, which aggregates them via the spatial aggregation model, producing the position estimation map; this last result is then sent to the control room for operational decision. This process is repeated iteratively.

C/C++ high-performance parallel programming library [2], allowing the user to specify custom communication topologies, working both in shared and distributed memory.

The application architecture is based on several building blocks, i.e. specialisations of the `ff_node` class, the primary logical unit exposed by FastFlow, connected in a tree-like structure. Building blocks can be executed in a distributed manner and can communicate by exchanging messages through different backends, like TCP and MPI [32]. At the leaves of the tree, we place multiple `ff_monode_t`, named *Camera*, each representing a different camera in the system. These building blocks are executed in parallel independently. For the distributed implementation, a new frame is read at each time step, its features extracted by a pre-trained base model, i.e. ResNet18 in our case, and warped according to a perspective transformation to consider the camera view angle. Conversely, for the centralised implementation, the Camera just acquires the current time step frame. Each *Camera* sends the result to the next level of the tree. This level consists of a `ff_minode_t` called *Aggregator*, which takes multiple inputs and returns a single output. In the distributed implementation, results collected from all children *Camera* are aggregated by the spatial aggregation model into the final position estimation map. Conversely, for the centralised implementation, the *Aggregator* node also takes over the feature extraction and perspective warping for each received camera frame.

Finally, the position estimation map is sent to the tree's root, i.e. `ControlRoom` node, where real-time decisions can be taken in a real-world deployment. Figure 1 summarises the distributed implementation architecture. This procedure is synchronously repeated using an additional *Sync* node connected to all *Camera* (not shown in the figure for simplicity) used to trigger the next frame until all frames are processed. In case of multiple open areas, the application supports multiple subtrees, one for each area, each consisting of an *Aggregator* with multiple *Camera* as children.

4 EXPERIMENTAL RESULTS

4.1 Setup

Dataset: We use the Wildtrack multiview dataset [7]. It comprises 7 static cameras capturing views of a public open area with dense groups of pedestrians standing and walking. The dataset provides each camera with the accurate position and view angle and 400 time-synchronised full-HD frames.

Testbed: All experiments were run on the HPC4AI cloud computing facility [3] exploiting 10 virtual machines, each one equipped with 8 64-bit vCPUs mapping to dedicated cores of an Intel Xeon Gold-6230 @2.10GHz processor (Skylake, IBRS), 16GB RAM, 1 Gb/s interconnection network, and running Ubuntu 22.04 as operating

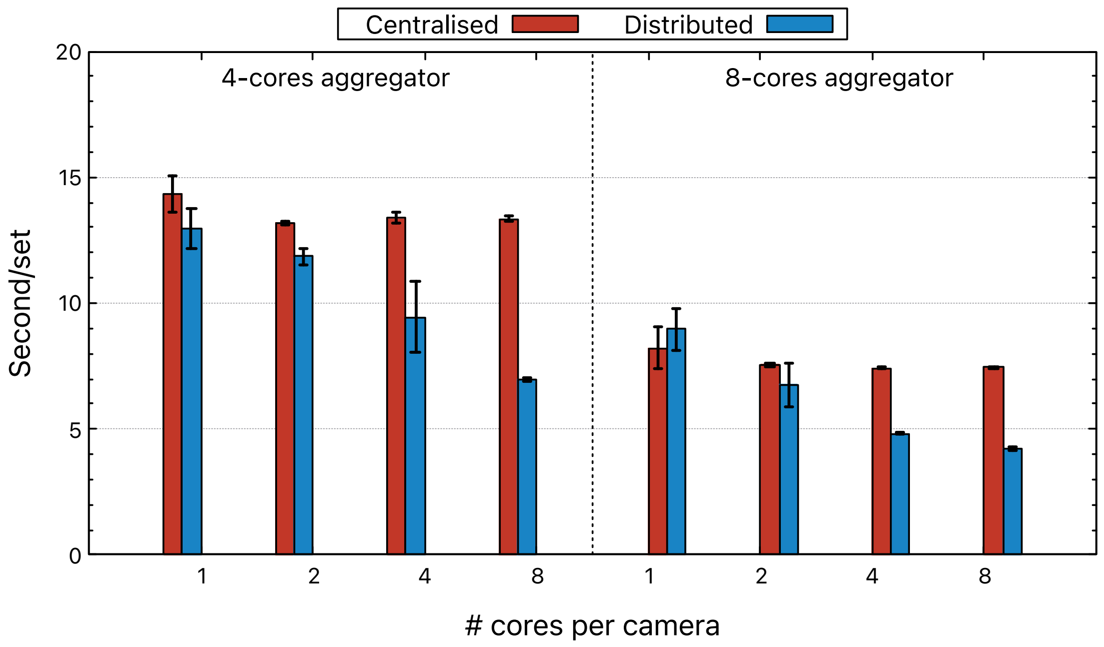

Figure 2: Computational performance of the proposed systems with varying computational power available. This plot compares the computational performance of the distributed MVDet system, measured as seconds taken to process each frame +/- the 95% confidence interval over 5 runs, in all possible combinations of computational power assigned to the server (4, 8 cores) and cameras (1, 2, 4, 8 cores).

system. We chose to use CPU-only nodes to better model the capabilities of edge devices, which often forgo GPUs for cost, energy, and thermal constraints. In contrast, the single-core performance of modern SoC can be in line with that of a vCPU. Each system component (7 Camera, 1 Sync, 1 Aggregator and 1 ControlRoom node) is deployed on a different virtual machine.

Performance metric: We choose the time needed to process one complete set of camera frames and produce the estimated positions as a performance metric. Experiments are replicated 5 times, and we report the mean plus 95% confidence interval (CI).

Deployment scenarios: We tested the proposed system under various conditions to emulate different deployment scenarios. We created two versions of the multiview detection application. The first is a centralised version, which acts as a baseline representing the traditional client-server approach where each Camera (client) sends the acquired frame to the Aggregator (server). In this case, the Aggregator node performs all the model computation, whilst the Camera nodes limit themselves to acquiring the next frame and sending it to the Aggregator. This implementation implies almost no computational power needed from the Camera but places a heavy computational burden on the Aggregator instead; furthermore, the communications are lighter, since the acquired frames are smaller tensors, size $[3x1920x1080]$, i.e. 6.1 MB, than the extracted feature tensors, size $[512, 360, 120]$, i.e. 21.6 MB. The second is the distributed version, where we split the detection model according to Figure 1 leveraging the computing power of each Camera for feature extraction and perspective warping, while leaving to the Aggregator spatial aggregation and final position estimation.

Resource modulation: We test the proposed multiview detection system under different combinations of computational power assigned to the critical software components, i.e. Camera and Aggregator, thus simulating how different edge devices with varying performance impact the system. We modulate the computing power by varying the number of cores available to the different components using taskset and providing hints to the number of threads to spawn to libtorch via the `MKL_NUM_THREADS` and `OMP_NUM_THREADS` environment variables. Specifically, we tested the system assigning to each Camera 1, 2, 4, or 8 cores, and to the Aggregator 4 or 8 cores, for 16 different computational power configurations. We also emulate different network conditions by limiting the bandwidth available to the different nodes to study how it affects the performance of the proposed system. Indeed, edge devices typically need to rely on slower networks, e.g. cellular connections. Kollaps is a decentralised container-based network emulator, exploiting Docker Swarm (or Kubernetes) to deploy and run distributed computations with specific network topology made of bridges and links with specific upload/download bandwidth, latency, and jitter characteristics enforced using the `traffic control` capability of the Linux kernel. In the following experiments, we study the impact of varying upload/download bandwidths of the Camera and Aggregator. We test the system with 10/10 Mbps per Camera and 100/100 Mbps for the Aggregator, representing a low bandwidth scenario. We then increase the Camera bandwidth to 25/284 Mb/s, which represents

Table 1: Computational performance of the proposed systems with varying computational power and network bandwidth. The first two scenarios adopt a bandwidth comparable with current 5G performance in Italian cities, while the third simulates a resource-constrained edge deployment.

Aggregator cores	Camera bandwidth (up/down)	Aggregator bandwidth (up/down)	Centralised (s/set)	Distributed (s/set)
4 cores	25/284 Mb/s	1/1 Gb/s	15.38	49.68
8 cores	25/284 Mb/s	1/1 Gb/s	11.98	46.89
8 cores	10/10 Mb/s	100/100 Mb/s	14.22	108.79

the average upload/download speeds achieved by the best 5G network in Italy in 2022 [1], and consequently increase the network bandwidth of the aggregator to 1/1 Gb/s to cope with the faster aggregated bandwidth from the Cameras. The latter configuration is tested assigning 4 or 8 cores to the Aggregator.

4.2 Results

Figure 2 presents the results across the 16 computational power configurations. One can notice how the centralised implementation performance is basically unaltered by the amount of computing power given to the Camera, while doubling the number of cores given to the Aggregator almost halves the amount of time required to process a single frame, i.e. from 13.57 s to 7.66 s on average. Conversely, the distributed approach is more sensible to the computing power given to both Camera and Aggregator, steadily increasing its computational performance when the computing resources given to the system (Camera and Aggregator nodes) increase from 12.97 s/set using a total of $7x1 + 4 = 11$ vCPUs to 4.23 s/set using a total of $7x8 + 8 = 64$ vCPUs.

By comparing the two systems, it is clear that the distributed implementation obtains globally lower computational times, achieving better computational performance with respect to the centralised approach thanks to exploiting the computational power spread over the computing continuum. The higher the computing power of the Camera, the more significant the performance gap, i.e. up to 1.92x faster with 4 cores per Aggregator and 8 cores per Camera. However, also the network plays a role. Indeed, in the 1 (8) cores per Camera (Aggregator), the centralised approach beats the decentralised approach by a small margin due to the increased communication time to transmit the 3.5x larger feature maps instead of the plain captured frames.

Looking at Table 1, it is possible to observe how both implemented systems behave under different network bandwidth conditions. To better simulate a low-power edge system, each camera node is allocated 2 cores. As can be seen, the distributed implementation particularly suffers from bandwidth limits. As explained before, the feature maps are way heavier than the plain frames; hence, bandwidth limits have a particular impact on the system performance, regardless of the amount of computing power allocated to each component. This behaviour can be noted especially in the last row of Table 1, in which the centralised system is 7.65x times faster than the distributed one. In a 5G network, which has higher bandwidths, this speedup decreases to 3.23x-3.91x, a reduction of 42.22%-51.11% with respect to the previous scenario. Instead, when the network is not the bottleneck, the distributed approach is the best in almost all scenarios, as shown in Figure 2. Please note that here we ported the MVDet model as is without any optimisation for communication not to alter the model performance. While possibly having a detrimental effect on the model performance, the communication cost could be lowered by compressing the feature maps or retraining the model with feature maps having a lower number of channels.

This comparison clearly shows how the environment can influence the real-world deployment of an edge inference system. Computational power and network bandwidth allocated to each computing continuum element are crucial, but there is no one-fits-all strategy to implement such systems. Workload distribution across the continuum can often become disadvantageous if it leverages critical resources that the centralised counterpart, instead, is not so reliant on. Nevertheless, the same distributed deployment can efficiently exploit all the available resources in an ideal scenario, thus outperforming the more centralised approach. Since environmental conditions change over time, future edge inference systems should consider this variability and try to accommodate and adapt to it, especially if they claim to be flexible, reliable, and efficient.

5 CONCLUSIONS

This research work presented a real-world, edge-inference multi-view detection system implemented with the FastFL framework. We showed how to move from an offline ML model to a high-performance, distributed inference system, measuring how different computing and network conditions can affect a real-world implementation. We showed how the environmental setting can affect the real-world performance of the computation implemented according to two different strategies: the centralisation of the computation on more powerful devices and the distribution of the computation to exploit as much edge computing power as possible.

We demonstrated that no strategy is inherently better than the other; however, they offer different properties that can offer better performance in different scenarios, pushing for the need for dynamic, adaptable edge inference systems. Implementing more sophisticated ML techniques in the proposed software, such as model distillation, quantisation and compression, and the exploitation of specialised hardware, such as TPUs or NPUs, could compensate for some of the disadvantages of the distributed system and are thus worth investigating as future work.

ACKNOWLEDGMENTS

This work receives EuroHPC-JU funding under grant no. 101034126, with support from the Horizon2020 programme (the European PILOT). This work is also supported by the Spoke "FutureHPC &

BigData" of the ICSC – Centro Nazionale di Ricerca in "High Performance Computing, Big Data and Quantum Computing", funded by European Union – NextGenerationEU.

REFERENCES

[1] [n. d.]. OpenSignal Italy - Mobile Network Experience Report - November 22. https://www.opensignal.com/reports/2022/11/italy/mobile-network-experience. Accessed: 2023-09-21.

[2] Marco Aldinucci, Marco Danelutto, Peter Kilpatrick, et al. 2017. *Fastflow: High-Level and Efficient Streaming on Multicore.* Wiley-Blackwell, 261–280.

[3] Marco Aldinucci, Sergio Rabellino, Marco Pironti, et al. 2018. HPC4AI: an AI-on-demand federated platform endeavour. In *Proceedings of the 15th ACM International Conference on Computing Frontiers.* Association for Computing Machinery, Ischia, Italy, 279–286.

[4] Faisal Alsakran, Gueltoum Bendiab, Stavros Shiaeles, et al. 2019. Intrusion Detection Systems for Smart Home IoT Devices: Experimental Comparison Study. In *International Symposium Security in Computing and Communications (SSCC) (Communications in Computer and Information Science, Vol. 1208).* Springer, 87–98.

[5] Amine Benelallam, Massimo Tisi, Jesús Sánchez Cuadrado, et al. 2016. Efficient model partitioning for distributed model transformations. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering.* 226–238.

[6] Gary Bradski, Adrian Kaehler, et al. 2000. OpenCV. *Dr. Dobb's journal of software tools* 3, 2 (2000).

[7] Tatjana Chavdarova, Pierre Baqué, Andrii Maksai, et al. 2018. WILDTRACK: A Multi-camera HD Dataset for Dense Unscripted Pedestrian Detection. *IEEE Conference On Computer Vision And Pattern Recognition (CVPR)*, 5030–5039.

[8] Bart Cox, Robert Birke, and Lydia Y. Chen. 2022. Memory-aware and context-aware multi-DNN inference on the edge. *Pervasive Mob. Comput.* 83 (2022), 101594.

[9] Francois Fleuret, Jerome Berclaz, Richard Lengagne, et al. 2007. Multicamera people tracking with a probabilistic occupancy map. *IEEE transactions on pattern analysis and machine intelligence* 30, 2 (2007), 267–282.

[10] Tiago Gomes, Sandro Pinto, Adriano Tavares, et al. 2015. Towards an FPGA-based edge device for the Internet of Things. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA).* IEEE, 1–4.

[11] Jianping Gou, Baosheng Yu, Stephen J Maybank, et al. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129 (2021), 1789–1819.

[12] Luca Greco, Gennaro Percannella, Pierluigi Ritrovato, et al. 2020. Trends in IoT based solutions for health care: Moving AI to the edge. *Pattern recognition letters* 135 (2020), 346–353.

[13] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. 2017. Learning non-maximum suppression. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 4507–4515.

[14] Yunzhong Hou, Liang Zheng, and Stephen Gould. 2020. Multiview Detection with Feature Perspective Transformation. In *Computer Vision – ECCV 2020*, Vol. 12352. Springer International Publishing, Cham, 1–18.

[15] Tzu-Hsiang Hsu, Yen-Cheng Chiu, Wei-Chen Wei, et al. 2019. Ai edge devices using computing-in-memory and processing-in-sensor: from system to device. In *2019 IEEE International Electron Devices Meeting (IEDM).* IEEE, 22–5.

[16] Dragi Kimovski, Roland Mathá, Josef Hammer, et al. 2021. Cloud, Fog, or Edge: Where to Compute? *IEEE Internet Computing* 25, 4 (2021), 30–36.

[17] Gianluca Mittone, Walter Riviera, Iacopo Colonnelli, et al. 2023. Model-Agnostic Federated Learning. In *Euro-Par 2023: Parallel Processing.* Springer, Limassol, Cyprus, 383–396.

[18] Gianluca Mittone, Nicolò Tonci, Robert Birke, et al. 2023. Experimenting with Emerging RISC-V Systems for Decentralised Machine Learning. In *20th ACM International Conference on Computing Frontiers (CF '23).* ACM, Bologna, Italy, 73–83.

[19] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, et al. 2019. Online model distillation for efficient video inference. In *Proceedings of the IEEE/CVF International conference on computer vision.* 3573–3582.

[20] Arun Narayanan, Arthur Sousa De Sena, Daniel Gutierrez-Rojas, et al. 2020. Key Advances in Pervasive Edge Computing for Industrial Internet of Things in 5G and Beyond. *IEEE Access* 8 (2020), 206734–206754.

[21] Wanli Ouyang, Xingyu Zeng, and Xiaogang Wang. 2015. Partial occlusion handling in pedestrian detection with a deep model. *IEEE Transactions on Circuits and Systems for Video Technology* 26, 11 (2015), 2123–2137.

[22] Roberto G Pacheco, Rodrigo S Couto, and Osvaldo Simeone. 2021. Calibration-aided edge inference offloading via adaptive model partitioning of deep neural networks. In *ICC 2021-IEEE International Conference on Communications.* IEEE, 1–6.

[23] Claus Pahl, Sven Helmer, Lorenzo Miori, et al. 2016. A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW).* 117–124.

[24] Adam Paszke, Sam Gross, Francisco Massa, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 8024–8035.

[25] Amir M Rahmani, Tuan Nguyen Gia, Behailu Negash, et al. 2018. Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach. *Future Generation Computer Systems* 78 (2018), 641–658.

[26] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, et al. 2022. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125* 1, 2 (2022), 3.

[27] Partha Pratim Ray. 2023. ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems* 3 (2023), 121–154.

[28] Jonah Sengupta, Rajkumar Kubendran, Emre Neftci, et al. 2020. High-speed, real-time, spike-based object tracking and path prediction on google edge TPU. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS).* IEEE, 134–135.

[29] Suhail Mohmad Shah and Vincent KN Lau. 2021. Model compression for communication efficient federated learning. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[30] Hang Su, Subhransu Maji, Evangelos Kalogerakis, et al. 2015. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision.* 945–953.

[31] Tianxiang Tan and Guohong Cao. 2020. FastVA: Deep learning video analytics through edge processing and NPU in mobile. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications.* IEEE, 1947–1956.

[32] Nicolò Tonci, Massimo Torquati, Gabriele Mencagli, et al. 2022. Distributed-Memory FastFlow Building Blocks. *Int. J. Parallel Program.* 51, 1 (2022), 1–21.

[33] Xiaowei Xu, Yukun Ding, Sharon Xiaobo Hu, et al. 2018. Scaling for edge inference of deep neural networks. *Nature Electronics* 1, 4 (2018), 216–222.

[34] Yang You, Zhao Zhang, Cho-Jui Hsieh, et al. 2019. Fast deep neural network training on distributed systems and cloud TPUs. *IEEE Transactions on Parallel and Distributed Systems* 30, 11 (2019), 2449–2462.

[35] Minchen Yu, Zhifeng Jiang, Hok Chun Ng, et al. 2021. Gillis: Serving large neural networks in serverless functions with automatic model partitioning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS).* IEEE, 138–148.

[36] Ruichi Yu, Ang Li, Chun-Fu Chen, et al. 2018. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 9194–9203.

[37] Shaojun Zhang, Wei Li, Yongwei Wu, et al. 2018. Enabling Edge Intelligence for Activity Recognition in Smart Homes. In *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS).* 228–236.

[38] Shifeng Zhang, Longyin Wen, Xiao Bian, et al. 2018. Occlusion-aware R-CNN: Detecting pedestrians in a crowd. In *Proceedings of the European conference on computer vision (ECCV).* 637–653.

[39] Menglong Zhu, Konstantinos G Derpanis, Yinfei Yang, et al. 2014. Single image 3D object detection and pose estimation for grasping. In *2014 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 3936–3943.