# Edge-Only Inference: Load Balancing vs Split Computing from a Latency Perspective

Mateus S. Gilbert[a,b], Nicolas Peugnet[a], Marcello L. R. de Campos[c], Miguel Elias M. Campista[b], Brigitte Kervella[a,d], Thi-Mai-Trang Nguyen[a,e], Marcelo Dias de Amorim[a], Anne Fladenmuller[a]

[a] *Sorbonne Université, CNRS, LIP6, F-75005, Paris, France*
[b] *Grupo de Teleinformática e Automação – GTA, PEE/COPPE-DEL/POLI, Universidade Federal do Rio de Janeiro – UFRJ, Rio de Janeiro, RJ, Brazil*
[c] *Laboratório de Sinais, Multimídia e Telecomunicações – SMT, PEE/COPPE-DEL/POLI, Universidade Federal do Rio de Janeiro – UFRJ, Rio de Janeiro, RJ, Brazil*
[d] *Université Picardie Jules Verne, UPJV, F-80000, Amiens, France*
[e] *Université Sorbonne Paris Nord, L2TI, F-93430, Villetaneuse, France*

**Abstract**

The deployment of deep neural networks (DNN) at the edge is critical for time-sensitive applications, such as emergency response scenarios, where a disconnected wireless mesh network (WMN) is deployed in a disaster zone. However, as the size and complexity of DNNs increase, executing these models on resource-constrained edge devices becomes challenging. We observe that as the memory footprint of a model approaches the available capacity of an edge server, the system is forced into early memory swapping, leading to a severe degradation in execution performance. Split computing (SC) has emerged as a promising solution for edge-only DNN inference to bypass these hardware limitations. However, the ad hoc deployment and multi-hop characteristics of WMNs in emergency contexts can introduce significant challenges for SC deployment. To this end, we evaluate four different DNN models in a real testbed using different split criteria to precisely measure the delay of the SC inference pipeline. This granular breakdown of processing and network transmission times demonstrates that the communication cost is the dominant bottleneck, regardless of the chosen split criterion. From these empirical measurements, we deduce that the viability of SC in wireless edge networks depends primarily on minimizing the size of transmitted data transfers, such as through feature map compression and quantization, rather than solely optimizing model partitioning.

*Keywords:* resource-constrained devices, wireless mesh networks, deep neural networks, split computing, computer vision

## 1. Introduction

Recent advances in machine learning are enabling the development of artificial intelligence-based systems to assist first responders and emergency workers in disaster environments [1]. In particular, the deep neural network (DNN)'s prowess in computer vision [2], natural language processing [3], and big-data processing is driving the emergence of smart, automated services for crisis management, including automated damage assessment and real-time situational awareness [1, 4].

Deploying artificial intelligence services in disaster scenarios presents significant challenges. First responders typically rely on resource-constrained devices unable to execute state-of-the-art DNNs locally [1, 5, 6]. Furthermore, the core network infrastructure is often degraded or unavailable during crisis scenarios [1, 7–9], which often limits the use of cloud-based implementations, despite the cloud offering the necessary computational power [6]. Combining these constraints with the fact that emergency services typically require low-latency processing to ensure rapid emergency responses [1], it is necessary to develop solutions that deploy DNNs at the edge or optimize them for local, resource-constrained execution.

### 1.1. Reducing DNN Implementation Cost

A significant effort is devoted to designing new models and implementation strategies to reduce DNN requirements, including lowering memory footprint and computational cost [10, 11]. Most state-of-the-art DNNs are overparametrized for their intended tasks [12], meaning that they could still perform well with fewer parameters. Techniques such as parameter pruning and quantization reduce complexity by removing insignificant parameters or lowering representational precision, at the cost of small performance losses in DNNs [12]. However, emergency services require reliable decisions by machine learning agents [1], which can limit the usage of these techniques. Quantization-

*Email addresses:* gilbert@gta.ufrj.br (Mateus S. Gilbert), nicolas.peugnet@lip6.fr (Nicolas Peugnet), campos@smt.ufrj.br (Marcello L. R. de Campos), miguel@gta.ufrj.br (Miguel Elias M. Campista), brigitte.kervella@lip6.fr (Brigitte Kervella), thimaitrang.nguyen@univ-paris13.fr (Thi-Mai-Trang Nguyen), marcelo.amorim@lip6.fr (Marcelo Dias de Amorim), anne.fladenmuller@lip6.fr (Anne Fladenmuller)

aware training and parameter tuning can mitigate performance loss but add extra steps to the DNN deployment process [12], limiting their adoption on rapid, off-the-shelf deployments required during some emergency scenarios.

Alternatively, and orthogonal to parameter pruning and quantization techniques, split computing (SC) exploits the multi-stage computational nature of DNNs to distribute layers across multiple devices [13–15]. By partitioning a DNN for a given task into disjoint sets of layers and placing them on different devices, SC enables cooperation during DNN computation, thereby minimizing the computational burden and memory footprint on resource-constrained devices. While SC has shown promise in edge and mobile networks with single access points [16, 17], its application in multi-hop wireless mesh network (WMN) (common in emergency scenarios) introduces challenges, such as increased communication latency due to multi-hop links [18, 19].

### *1.2. How and When to Use Split Computing in Emergency Scenarios*

Implementing SC introduces a critical trade-off between communication and computation latencies. On the network side, both the number of DNN splits and the topological distance between consecutive blocks generate communication overhead. On the computational side, assigning multiple layers or the entire DNN to a single resource-constrained device risks overloading it at peak request rates, as these edge devices lack the processing power of conventional servers. Focusing on a cloud-isolated WMN of resource-constrained devices, this work investigates *the specific conditions under which SC effectively minimizes overall inference latency.* Ultimately, we aim to provide concrete deployment guidelines driven by resource capacity (e.g., available memory) and request generation rates to enable plug-and-play deployment of DNN-based edge services, guiding decisions on whether to adopt SC and how to deploy it efficiently.

### *1.3. Contributions*

We implement an SC inference pipeline in a multi-hop WMN and compare it with a load-balancing baseline that uses the same devices, each hosting a copy of the complete DNN. We perform experiments with different memory availability, client request generation rates, and transmission policies, which yield the following contributions:

- We observe that the multi-hop architecture of WMNs is the primary source of delay in the inference pipeline, leading us to conclude that load-balancing approaches that do not partition a DNN are generally preferable to SC in time-constrained scenarios.
- We demonstrate that SC outperforms traditional load balancing under intensive processing workloads. This finding allows us to clearly delineate the specific conditions under which SC should be deployed to minimize inference latency.
- We provide a comprehensive analysis of latency sources within the SC pipeline to better characterize system bottlenecks.
- Drawing on our experimental results, we establish practical recommendations to inform and guide future integration of SC within WMN architectures.

## 2. Background on Split Computing

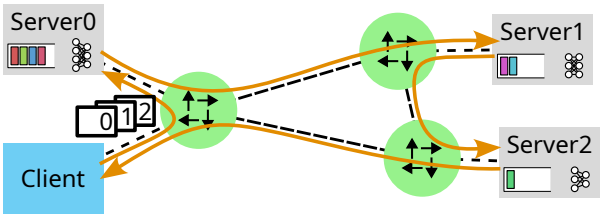
Split Computing (SC) has emerged as a promising distributed inference technique for resource-constrained environments. Fundamentally, it consists of dividing a pretrained DNN at specific split points.

### *2.1. Edge-Cloud Split Computing*

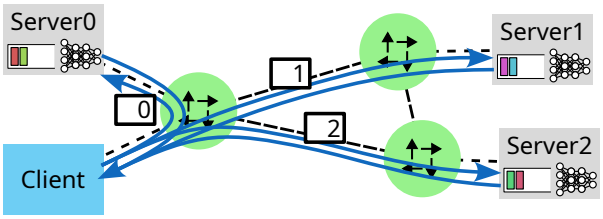
SC traditionally relies on a collaborative edge-cloud architecture, where powerful cloud nodes support edge devices in executing DNN-based services [13]. In this configuration, a DNN is split into two disjoint sets of layers, with the initial DNN layers executed on resource-constrained edge devices and the remaining, more computationally intensive layers offloaded to the cloud. This approach seeks a balance between reducing local implementation costs and managing the communication overhead of data exchange between the edge and the cloud. If minimizing inference latency is the main goal and cloud connectivity is reliable and fast, it is often best to store a small set of layers at the edge and rely more on the cloud, given that resource-constrained edge devices typically perform DNN computations more slowly than cloud servers. Performing several DNN operations at the edge can slow down the inference process to the point of negating any advantage of doing a split [20]. In these cases, the first DNN layers can function as a resource-efficient feature extractor for intermediate DNN layers [21, 22], stored in the cloud. Alternatively, the layers stored in the resource-constrained device can function as a lightweight DNN by adding auxiliary output layers (referred to as early exits) that can perform the inference attempt on easy-to-predict data, thus minimizing the inference latency and communication cost associated with unnecessary transmissions to the cloud [14]. Yet, miscalibrated and overconfident early exits in such contexts can lead to significant performance degradations [23].

### *2.2. Edge-Only Split Computing*

Disaster scenarios can cause partial or complete failure of cellular network infrastructure, preventing reliable access to remote cloud computing resources [1, 24], limiting the adoption of traditional SC. These scenarios require rapid, ad hoc deployment of essential services, often on resource-constrained devices. WMN can provide a self-adaptive wireless implementation for connecting edge devices, adjusting on the fly to connection failures to keep relevant services running [24]. Given these characteristics, an alternative SC approach partitions the DNN into disjoint




(a) SC: the DNN is split among the three servers, and each image is processed sequentially by them.




(b) RR: all the servers have the full DNN, each image is sent to a different server.

Figure 1: Two methods for sharing the inference load in a WMN. Grey nodes represent resource-constrained devices that contain the DNN sections, the blue node is the client of the DNN-based service, and green nodes are routers in the WMN.

blocks of layers that can be distributed across the WMN, enabling fully localized edge deployment. The objective of this deployment is to create an inference pipeline that parallelizes the processing of multiple data instances by having each resource-constrained device concurrently execute its designated DNN computation on different inputs.

In this direction, Tuli *et al.* proposed an SC deployment for mobile networks [16]. Through simulation, in which Raspberry Pi (RPi)4*B*-like nodes form a star topology with a centralized broker, they show SC capabilities for minimizing inference latency at the edge and their capacity to adapt to mobile network dynamics. In another work, Li *et al.* implement SC inference pipeline using NVIDIA Jetson Xavier, a GPU-capable edge device, in a star topology with a single access point that provides a circular (ring) overlay topology in which DNN sections exchange intermediate results [17]. The proposed pipeline outperformed a load-balancing approach, which evenly distributes inference requests among devices hosting copies of the same DNN, when the DNN application receives a large number of inference requests. Although promising, the modest network topologies in most multi-split SC make it unclear whether SC suits multi-hop deployments, such as those found in a typical WMN. In particular, the success of such deployments requires low-latency wireless links between adjacent nodes in the DNN-inference pipeline to enable rapid exchange of feature maps (i.e., intermediate DNN data). Additionally, emergency scenarios may constrain node placement and network design, further complicating the provision of low-latency feature map exchange.

## 3. Split Computing in Wireless Mesh Networks

If a DNN is bigger than the available memory on a resource-constrained edge device, we can use SC to split the model between similar devices, which can then perform the DNN inference collaboratively. However, if the DNN fits within the available memory, we have two options. Either we can split the model across these resource-constrained devices to implement an SC inference pipeline, or we can deploy copies of the full DNN on these devices and share the load between them. In particular, we can think of the former strategy as a type of partial offloading scheme where the inference task is split into sub-tasks that are sent to different computing nodes, whereas the latter is a type of binary offloading scheme [25]. However, hereinafter we will drop these terms and use SC and load balancing instead for simplicity.

### *3.1. DNN Edge Deployment*

We depict a an SC implementation in a cloud-isolated edge environment deployed over a WMN in Figure 1a. In this architecture, distinct subsets of DNN layers are distributed across separate edge servers interconnected by routers. `Server0` starts the DNN inference process, followed by `Server1`, and then by `Server2`. Similarly, Figure 1b presents a load-balancing approach in which we place copies of the same DNN on the servers. We adopt the round-robin (RR) scheduling algorithm where the $i$-th server handles requests $(i, (i+N), (i+2N), \ldots)$, where $N$ is the number of servers. Although load-balancing approaches include more sophisticated schemes, with some accounting for task priority [26], we opt for this scheduling algorithm as a simple baseline to evaluate the SC inference pipeline. In both setups, the total inference latency, which we refer to as end-to-end (E2E) latency, is the time from the client's inference request to the client's receipt of the corresponding inference result. For a single image, the E2E latency corresponds to the time spent in the SC pipeline in the former case, and to the time it takes for the assigned server to respond to the client's inference request in the latter.

Considering $n$ devices, and $T_{d_i}$, $T_{q_i}$, and $T_{c_i}$ device $i$'s transmission, queue, and DNN computation times, respectively, we can sketch the E2E latency of the SC pipeline as the sum of latencies across all pipeline stages

$$T_{sc} = \sum_{i \in [n]} \left( T_{d_i} + T_{q_i} + T_{c_i} \right),$$

and the RR's under low load conditions (away from queue saturation) as

$$T_{rr} = \frac{1}{n} \sum_{i \in [n]} \left( T_{d_i} + T_{q_i} + T_{c_i} \right),$$

i.e., the average across all $n$ devices, expecting the queuing time to decrease to approximately a fraction $\frac{1}{n}$ of what it would be if device $i$ operated alone, as it receives this fraction of the incoming requests. $T_{d_i}$ is determined by the ratio between data size (raw image for the first device in the SC pipeline and the devices in the RR, feature map

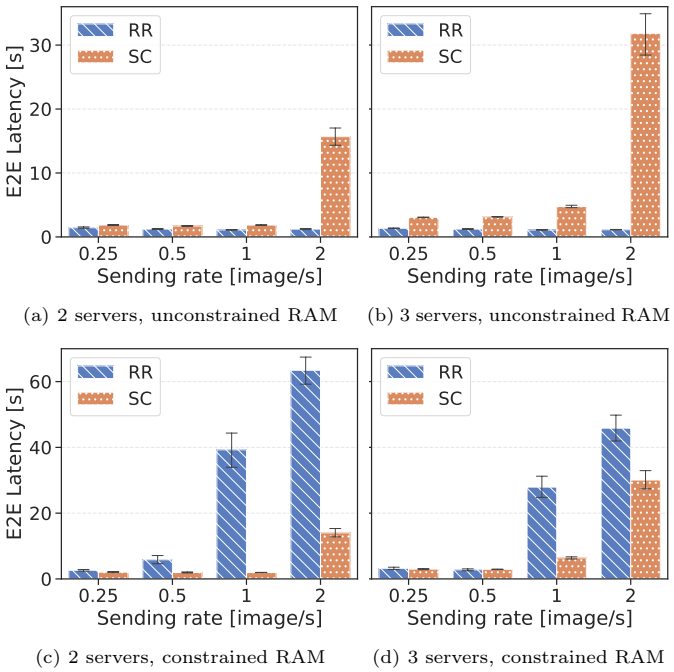
size in the case of the other SC-recruited devices) and the network bandwidth, whereas $T_{q_i}$ depends on the amount of images (feature maps) waiting to be processed at device $i$. Considering the device's peak processing capacity to be $C_i$ (in floating-point operation (FLOP)/s), $W_{\text{DNN}_i}$ denote a DNN's (or a set of DNN layers) workload (in FLOPs), and $\eta_{\text{DNN}}$ to be a correction factor that accounts for other hardware and DNN architectural factors that impact the true inference performance, $T_{c_i} = \frac{W_{\text{DNN}_i}}{\eta_{\text{DNN}} C_i}$. Specifically for the upcoming experiments, prior hardware benchmarking studies[1] report a $C_i \approx 9.69$ GFLOP/s, and we estimated (on preliminary experiments) $\eta_{\text{DNN}} \approx 0.57$ for the residual network (ResNet) family and $\eta_{\text{DNN}} \approx 0.34$ for the swin transformer family when using 256 RGB images, obtaining an inference latency estimation error below 5%.

The success of SC over RR leans on a few conditions. First is that $T_{c_i}$ of the full DNN is significantly large and/or the inference request arrival rate is such that it overwhelms the devices in the RR deployment in a manner that the $T_{q_i}$ increases considerably. Additionally, it requires $T_{d_i}$ in the SC pipeline to be small or negligible. In particular, the communication latency incurred by exchanging feature maps is a key contributor to total inference latency in SC. While in most SC works the number of hops between two adjacent DNN layers is limited to at most two hops [16, 17], this number can be larger in typical WMNs. SC implementation in WMNs aims to reduce service time for resource-constrained devices, but it may incur high communication latency. Therefore, except when the full DNN cannot fit on these edge devices, deploying the entire model on the same devices used in the SC pipeline and sharing the inference load among them can be a more effective strategy.

### *3.2. Inference Latency Analysis*

Figure 2 presents a comparison of the average E2E latency between the SC pipeline and the RR-based load balancing approaches. In both cases we use residual network (ResNet)50 [27] (an image classification model provided by `torchvision` [28]) across two- and three-server RPi4B topologies (in the former, we remove `Server2` from Figure 1). We evaluate these two inference methods under resource-limited (constrained RAM) and optimal (unconstrained RAM) conditions. In particular, the RPi4B has 4 GiB of RAM. The client generates requests at regular intervals, with an interoccurrence rate ranging from 0.25 to 2 images/s. Averages for each sending rate are computed over 100 images.

As illustrated in Figures 2a and 2b, processing a 265×256 RGB image with the DNN application takes under 1.5 seconds on an RPi4B when RAM is not constrained. Consequently, the RR-based load-balancing setup successfully manages the tested transmission rates. In contrast, Figure 2a demonstrates that the two-server SC pipeline yields




(a) 2 servers, unconstrained RAM (b) 3 servers, unconstrained RAM

(c) 2 servers, constrained RAM (d) 3 servers, constrained RAM

Figure 2: E2E classification latency (seconds) with ResNet50 deployed using SC and RR configurations, using 100 images from a client that generates requests at a regular interval. In the top subplots, the DNN application has access to the full 4 GiB of memory on the RPi4B, whereas in the bottom subplots, the available memory is limited to 145 MB. The error bars show the 95% confidence intervals.

latencies between 1.7 and 1.9 seconds for rates up to 1 image/s, spiking to 15 seconds at 2 images/s. Adding a third server to the SC pipeline further degrades performance, pushing E2E latencies to 3–5 seconds at lower rates ($\leq 1$ image/s) and up to approximately 30 seconds at 2 images/s. These results indicate that the transmission of feature maps across servers in the SC is non-negligible. Furthermore, the sharp rise in latency between the 1 and 2 images/s sending rates indicates that rapid, successive inference requests can overwhelm the first server, turning it into a bottleneck for the SC pipeline. This latency degradation may also be exacerbated by the increased network traffic from transmitting feature maps between devices within the SC architecture.

While Figures 2a and 2b might initially suggest that SC may not be optimal for edge-native execution under unconstrained conditions, Figures 2c and 2d present a scenario (where RAMs are constrained to 145 MiB) for which SC is preferable to the RR approach. If the nodes provide other services or perform other computations in parallel with the DNN service, memory availability can decrease to the point that it hinders the DNN service. The E2E latency differences between the RAM-constrained and unconstrained scenarios are less than 2 s when employing the SC pipeline, with E2E latency remaining virtually unchanged for rates equal to and below 0.5 images/s. Conversely, the RR setup experienced increases of at least 1 s, reaching an E2E latency of 45 s with 3 servers at a sending rate of 2 images/s,

[1] `https://web.eece.maine.edu/~vweaver/group/green_machines.html`

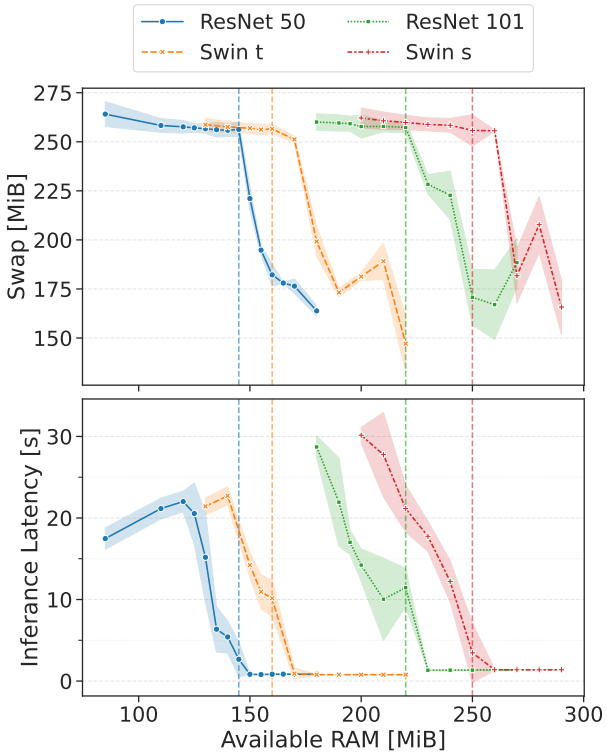



Figure 3: Impact of swap usage on DNN inference latency executed on an RPi4B across different memory availability. The dashed lines show an inflection boundary where inference latency remains stable until swap usage surpasses 250 $MiB$.

and above 60 s with 2 servers at the same rate. These preliminary results suggest a memory-availability threshold at which splitting a DNN across resource-constrained devices becomes more advantageous, despite the communication issues inherent to SC deployment in WMN. This observation suggests that SC is preferable when deployed nodes experience dynamic changes in memory availability.

## 4. DNN Inference Under Memory Pressure

As discussed in Section 3, SC deployments in WMN are advantageous when the DNN size is close to the RAM size. If the device has enough memory to fit the DNN comfortably, it is more sensible to deploy copies of the same model in different devices and share the load between them instead of splitting the DNN among them, because the first setup does not need to transmit feature maps through the WMN. However, if the device experiences memory pressure, the performance of the load-balancing deployment degrades significantly, thereby mitigating the latency issues observed in feature map transmission in the SC setup. Given this result, we investigate in greater detail the memory usage by the DNN-service.

We evaluate the performance of the DNN service related to its memory footprint by running the complete model on a single server, isolated in a Docker container. We distinguish between RAM, the high-speed physical memory used for active processing, and swap, a disk-based partition used as virtual memory when physical resources are saturated. We use Linux's *Control Group* feature [29] to artificially limit and vary the size of the RAM, then measure the performance of the DNN service across different RAM-size configurations.

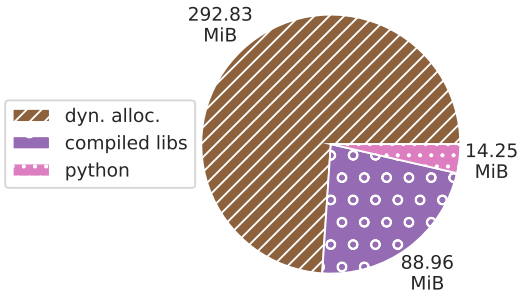
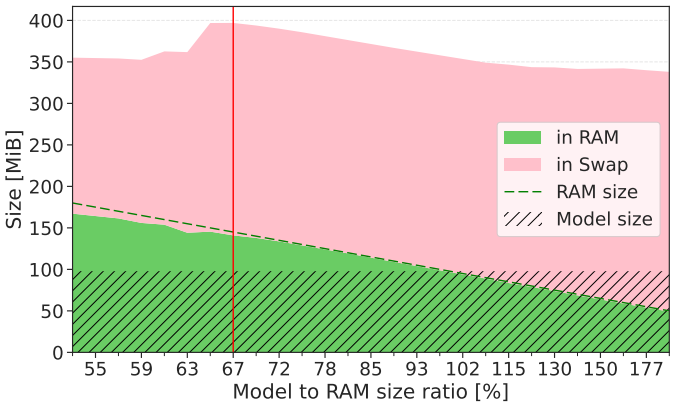


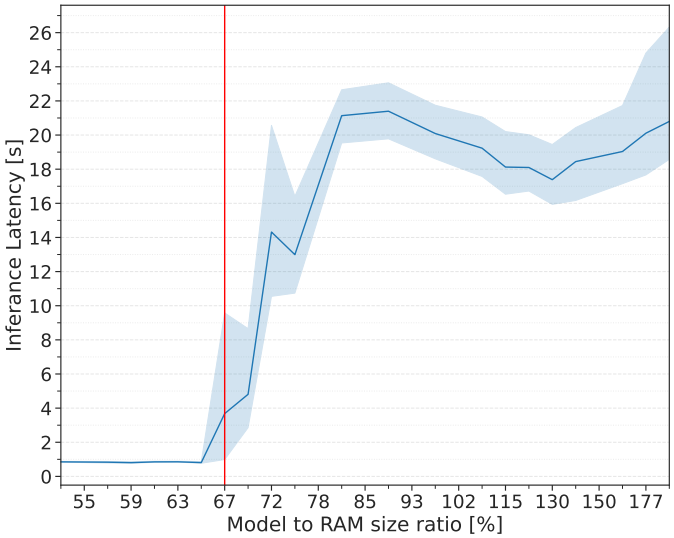

Figure 4: Content of the memory allocated by the DNN service using ResNet50. "dyn. alloc." is the memory dynamically allocated by the program with `malloc()` [30].

Figure 3 shows the impact of swap usage on the DNN inference latency using different DNNs. We can see that inference latency remains virtually stable until swap usage surpasses 250 MiB. Specifically, it seems there is an inflection point (indicated by the dashed lines) beyond which DNN inference latency becomes unstable and starts to degrade sharply. For instance, ResNet50, which achieved one of the mildest degradations when reaching this threshold, shows an approximately 2.5× increase; on the opposite end, Swin $t$ [31] observes a 10× increase from the start. Focusing on the former, ResNet50 occupies about 98 MiB of memory space; however, the DNN service requires more memory to operate. As seen in Figure 4, Python code and compiled libraries account for slightly more than a quarter of the total service memory. The remainder corresponds to memory dynamically allocated by the program, including the model itself, the Python libraries required by the DNN service, and additional memory allocated by Python to run the service.

Figure 5 presents in greater depth the RAM-swap data split for the execution of a ResNet50 on an RPi4B that receives a new image every 2 seconds. Specifically, we depict RAM availability as a function of model size, where a high percentage indicates a tight memory cap (represented by the green dashed line) relative to the fixed model size (hatched area). As we reduce the amount of RAM, the system must offload a significant portion of runtime data to swap space. For instance, Figure 5a shows a noticeable increase in swap size (pink area) when the ratio between model and RAM sizes goes from 63% to 67%, then remains constant from 67% onwards. We can see that 67%, although a significant portion of the DNN service is moved to swap, the amount of memory in RAM is slightly below the available capacity. Figure 5b shows that this model-to-RAM ratio coincides with when inference latency increases, suggesting a wasteful memory allocation by the DNN library. Most DNN libraries are not memory-efficient, performing




(a) Memory usage of the DNN service.




(b) Inference latency of the DNN service; the colored region shows the $5^{th}$ and $95^{th}$ percentiles.

Figure 5: Inference latency and memory usage of ResNet50 in a RPi4B. Although the device has sufficient memory and resources to host the DNN, the service time increases as available RAM diminishes, due to increased swap usage. Inference latency obtained for processing 256×256 RGB images, with 40 images sent at the rate of 1 image every 2 s, and memory limits enforced by restricting the container hosting the DNN service.

redundant memory copies that incur significant delays and memory usage [32]. Until the ratio reaches 67%, some inactive data that could be moved to swap may still remain in RAM. After this point, data relevant to the DNN service must be allocated in swap, forcing more active swapping to free RAM and retrieve data from swap, which increases the DNN service's latency. Specifically for the ResNet50, it suggests a transition state around 67% where the system shifts from regular operation in which load balancing is sufficient (and preferable) to alleviate the pressure imposed by high arrival rates, to a degraded regime in which SC can be the best option.

Combining the results in Figures 3 and 5, it becomes evident that the primary motivation for adopting SC is to reduce the memory footprint. When a single device lacks adequate capacity for a DNN, SC enables resource-constrained devices to execute inference collaboratively. Furthermore, even when a device theoretically possesses enough memory to host the model, dynamic memory constraints can force the system to rely heavily on swap space, significantly degrading response times, an issue that tends to worsen with larger DNNs. Under such conditions, incurring the communication overhead inherent to the SC framework is a justified trade-off. To optimize this approach, we will next analyze the specific sources of latency within the SC architecture and explore strategies to mitigate them.

## 5. Experimental Setup

We run an image classification task in an isolated WMN, where resource-constrained devices recruited for DNN inference are associated with different routers. We use the ILSVRC2012 ImageNet dataset [33], and DNNs provided by `Torchvision` [28]. We compare SC against a load balancing between resource-constrained devices, similar to the preliminary experiment performed in Section 3. Following the initial results discussed in Sections 3 and 4, we conduct extensive experiments by varying the inference-request arrival rate and DNN layer allocation per node to delineate the operational boundaries under which SC deployment is advantageous.



Figure 6: Picture of our WMN implementation. The YOI devices (in black) are wireless routers, and the RPi4B (in red and white) are devices recruited for the DNN service.

### 5.1. Network and Node Configuration

We build a WMN using `Green Communications`'s *Your Own Internet of Edges (YOI)* [34] as routers. YOIs are low-power edge devices that use two Wi-Fi 4 interfaces (2.4 GHz and 5 GHz) to build a wireless backhaul network and serve end-user devices as access points. In our setup, we use the 5 GHz band to build the backhaul mesh network and the 2.4 GHz band to deploy access points for resource-constrained devices recruited for the DNN inference process. We opted for this setup because traffic between routers is heavier than traffic between the RPis and their corresponding access points. With this configuration, we measure an

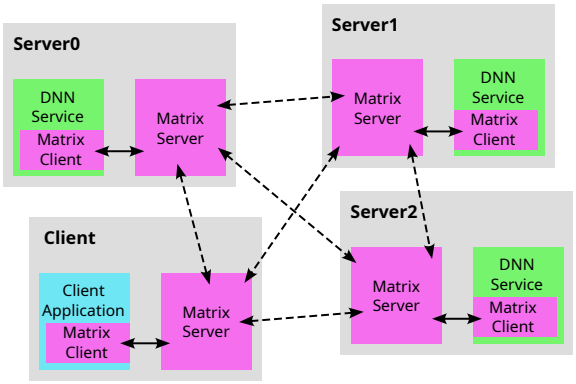



Figure 7: Diagram illustrating how the DNN and communication applications interact with one another. The gray boxes represent physical machines, and the colored squares are containers.

average bit rate of 38.1 Mb/s between a server node and its corresponding access point, 19.3 Mb/s from one server to another without competing traffic, and 16.9 Mb/s with competing traffic (from the other servers).

We deploy the DNNs on RPi4B. As shown in the green boxes of Figure 7, we deploy a DNN or a section of it in a Docker container. The RPis exchange data using a modified version of the Matrix protocol [35], discussed in more detail in Section 6. In a nutshell, Matrix is a decentralized publish-subscribe (PubSub) communication protocol for persistent data exchange, designed for instant messaging, voice-over-IP, and Internet of things (IoT) applications [35]. As shown in Figure 7, the communication service is also isolated in a Docker container, with no resource limitations (other than those of the device on which it is deployed). In the SC pipeline, a node hosting a section of the DNN sends a message (referred to as an event) in the Matrix room (equivalent to a PubSub topic) to indicate that it has finished its computation and includes a link to retrieve its feature maps. Then, the Matrix server on the node holding the next DNN section downloads the data into an in-memory byte queue to perform the next processing step, minimizing disk I/O and associated read/write overhead.

### *5.2. Deep Neural Networks and Split Configurations*

We select four DNNs trained for image classification from the `Torchvision` library to conduct our analysis: ResNet50 and ResNet101 [27], and Swin Transformers versions $t$ and $s$ [31]. In both pairs of models, the first version is more compact than the second, with fewer layers or other DNN structures. Table 1 brings estimates of the inference-related metrics of these three models when processing 256×256 RGB images, with the model accuracy being the one on the ILSVRC2012 ImageNet's *val* dataset [33]. The inference latency shown in Table 1 is an average of 100 repetitions excluding queue time, whereas FLOPs presents an estimate of the amount of computations performed by each DNN to produce a single inference [36].

We split the DNN by first grouping its layers into blocks, each containing exactly one residual connection. Residual

Table 1: Model size and inference characteristics of the DNNs selected for the upcoming experiments. Accuracy, inference latency, model size, and FLOPs computed for 256×256 RGB images.

| Model | Acc. [%] | Inf. Lat. [s] | Size [MiB] | FLOPs [G] |
|---|---|---|---|---|
| ResNet 50 | 80.3 | 1.0 | 97.7 | 5.4 |
| ResNet 101 | 81.7 | 1.8 | 170.3 | 10.3 |
| Swin $t$ | 81.1 | 1.2 | 107.9 | 3.9 |
| Swin $s$ | 82.8 | 2.2 | 186.7 | 7.5 |

connections are bypass links that skip a set of DNN layers [27]. We opt for this layer pre-grouping to minimize the amount of data transmitted. A split at a layer that is bypassed by a residual connection requires the transmission of two tensors: the input of the residual connection and the output of the mentioned layer. We adopt static SC configurations using three criteria: (1) split the number of DNN blocks equally among devices, (2) split DNN layers to minimize the memory footprint of the biggest DNN section, (3) split DNN layers to minimize the number of FLOPs adopting a similar approach as the previous criterion, and (4) minimizing the computation latency. All methods, except the one that splits layers based on the number of blocks, use the dynamic programming algorithm proposed by Skiena to solve the *ordered partition problem* [37]. In particular, the split criterion that aims to minimize computational latency is device-specific, as it requires knowledge of each block's processing time, which limits its use in mesh scenarios. We did not attempt to optimize the splitting algorithm because the paper focuses on providing guidelines for future SC deployments. In particular, early experiments indicated that, aside from the number of splits, split-configuration optimization has little impact on latency because the communication overhead arising from the multi-hop nature of our WMN outweighed any potential gains.

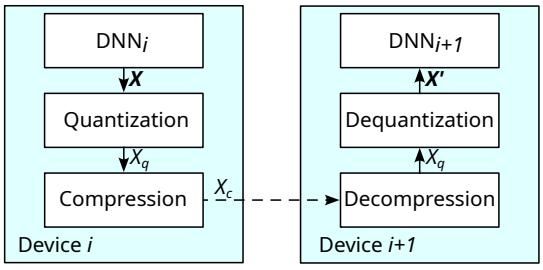



Figure 8: Block diagram representing the end-to-end quantization-compression pipeline. Compression/Decompression blocks employ the `Zstandard` compression [38]. Thus, $X = X'$ if the quantization compression is suppressed.

### *5.3. Further Specifications*

We employ data compression and quantization in some SC configurations to reduce data size without retraining the DNN [39]. Figure 8 shows a diagram of the end-to-end quantization-compression pipeline. We apply a linear quantization from float32 to uint8 ($X_q$ in Figure 8), followed

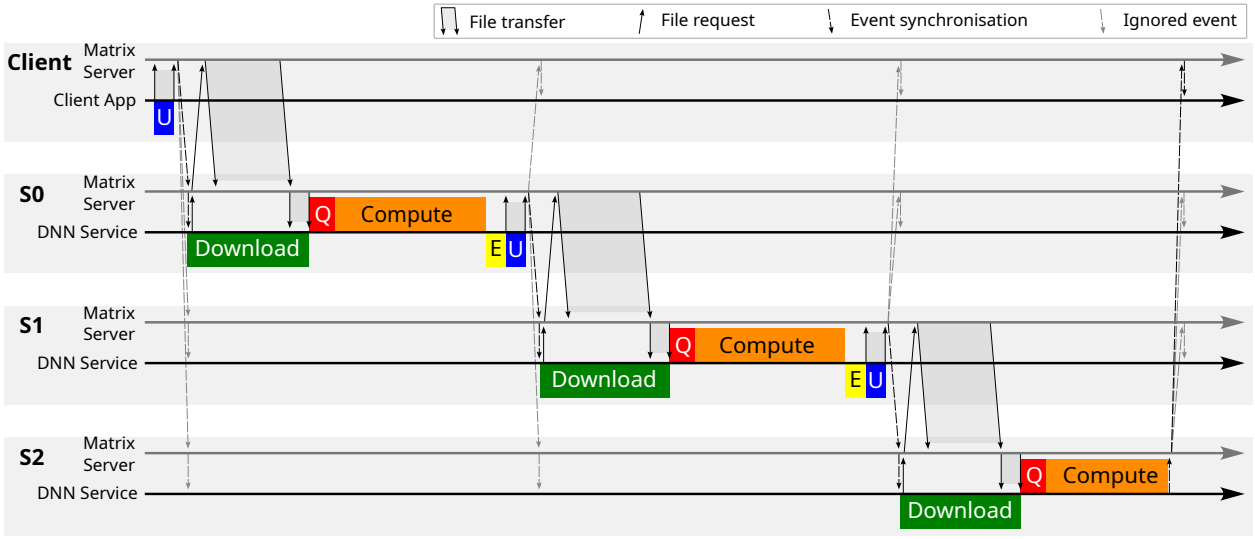



Figure 9: Chronogram of the inference pipeline for a single image in an SC deployment with three nodes. In our implementation, there is a single Matrix room shared by all servers, so each event sent to this room (represented by the arrows) is replicated on all participating servers. In this setup, each node downloads an image or the previous feature maps (in green), downloaded data are placed in an in-memory queue (in red), processed by the node's DNN section (in orange), and encoded for transmission (in yellow), and uploaded to the node's Matrix server (in blue).

by lossless compression of the quantized data ($X_c$). In implementations where we do not adopt the quantization procedure, $X' = X$. Specifically, we employ the `Zstandard` compressor because it has low computational latency, supporting real-time operation [38]. As will be seen later, these data reduction techniques can significantly reduce transmission latency between nodes; however, they can introduce accuracy degradations to the DNN service, which might not be tolerated by some emergency applications. We discuss this topic in more depth at the end of the paper.

We measure the time spent with each operation in the inference pipeline described in Section 3 to produce an in-depth analysis of the sources of latency in each step of the SC process. For such, as shown in Figure 9 for an SC that involves three nodes, we compute: (1) the full latency from client upload to download of the generated inference; (2) upload (in blue) and download (in green) delays from each device recruited in the SC pipeline; (3) the delay associated with the in-device DNN computation, including DNN-related computations (in orange) and data encoding procedure (linear quantization and lossless compression) for transmission (in yellow); and (4) the queuing delays (in red). In particular, the former is computed by measuring the total E2E latency from the client's upload of a new image to the client's receipt of the corresponding inference result. This includes the sum of the other measured delays together with the communication overhead of the Matrix protocol (represented in Figure 9 by the arrows).

## 6. Low-Latency Matrix to Ensure Service Resilience

This study was conducted as part of a larger project that deploys highly dynamic WMNs in emergency scenarios. In this project, disconnections and reconnections can happen frequently, leading to network partitions. We want

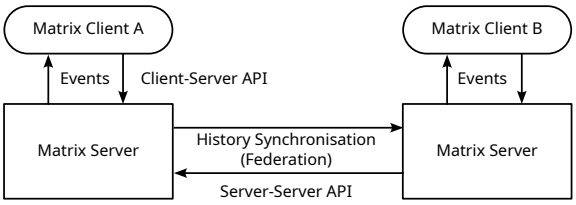



Figure 10: Basic principle of the Matrix protocol.

participants in the same partition to remain able to communicate with one another, even when isolated from the rest of the network. For this reason, we cannot use a centralized PubSub protocol like message queuing telemetry transport (MQTT). Thus, in this project, we adopt the Matrix protocol [35] as the communication layer between nodes because it gracefully handles disconnections.

The Matrix protocol is a decentralized publish-subscribe communication framework comprising client and server entities. As shown in Figure 10, its architecture is defined by two primary interfaces: the client-server API, which lets clients connect to their designated "home server" to exchange "events" within "rooms", and the server-server API, which enables federation. Through federation, servers synchronize room states, enabling clients to participate in rooms hosted beyond their home server. Furthermore, Matrix employs a robust state-resolution mechanism; when a server identifies missing prior events, it retrieves them from its peers. Because each server retains a comprehensive history of its respective rooms, it can effectively route messages on behalf of other nodes. This decentralized replication ensures network-wide event propagation and resilience, even in the absence of a fully connected network topology.

Traditionally, the Matrix protocol handles disconnections on long timescales that cannot accommodate WMN

dynamics. Matrix servers have timeouts and retries on the order of minutes, which cannot handle disconnections and reconnections that occur within a few seconds in WMN due to node mobility and varying link conditions. To better adapt Matrix to highly dynamic networks, we customized server implementations to react to network changes, using information gathered by OLSR [40], the underlying routing protocol. This allows them to transmit events to other servers as soon as they are available again.

Our customized Matrix server is based on Conduit[2], a lightweight implementation of the Matrix specification designed for low system requirements, and is thus well-suited for the RPi4B. A new endpoint has been added to its API, allowing us to force a retry when sending messages to specific destinations. An additional process monitors the network graph retrieved from Green Communication's router API [41] and queries this new endpoint when a destination becomes available.

## 7. Results

As seen previously, SC's advantage over a load-balancing strategy that replicates the same DNN across resource-constrained devices lies in its ability to minimize the memory footprint of the DNN-based service. Even in a resource-constrained scenario with sufficient memory to fit the DNN, inference latency increases significantly when the DNN-based service experiences memory pressure, which can occur when the edge device performs multiple tasks in parallel. If the device has no significant memory constraints, load-balancing approaches are preferable to SC, because transmitting feature maps over the WMN introduces significant delays that increase the SC-inference pipeline. To this end, we investigate in greater depth the sources of delay and possible solutions to mitigate the gap between SC and load-balancing approaches in scenarios without memory pressure. Table 2 provides a recap of the key experimental parameters discussed in the previous sections, along with additional parameters. Unless stated otherwise, all experiments share this setup.

Before examining the sources of delay of an SC pipeline in a WMN, we compare the impact of various DNN splitting strategies on end-to-end (E2E) delay across four different models. Specifically, we evaluate layer distributions across two or three servers (using one or two split points) driven by four distinct criteria: FLOPs balancing, memory footprint, block count, and processing latency. These criteria lead to distinct layer distributions. For instance, prioritizing a minimal memory footprint typically assigns more layers to the first RPi, as deeper layers tend to contain denser DNN computing units [27, 31]. Conversely, optimizing for FLOPs yields a distribution closely resembling an equal-block split, differing by only a few blocks. As illustrated in Figure 11, these structural differences do not yield consistent performance improvements, since each model favors a different split criterion. Interestingly, the criterion aimed at minimizing on-device computation latency actually resulted in the highest E2E inference latency in two-split SC configurations for three of the four models evaluated. This counterintuitive result suggests that data transmission bottlenecks or in-device queueing significantly impact the overall E2E inference time. Consequently, improving performance requires a more detailed examination of the primary sources of delay.

Table 2: Experiment Setup

| Parameter | Value |
|---|---|
| SC/RR Servers | RPi4B |
| Mesh routers | *Green Communications'* YOI |
| Router-to-Router bit rate[3] | 312 Mb/s |
| Server-to-Router bit rate[3] | 38.1 Mb/s |
| Server-to-Server bit rate[3] | 19.3 Mb/s |
| Dataset | ILSVRC2012 ImageNet val |
| Images | $256 \times 256$ RGB |
| Container Memory Limit | 512 MiB |
| Communication Protocol | Matrix conduit |
| Repetitions | 20 |
| Images per Repetition | 100 |
| Inference Request Generation | Poisson process |
| Average rate | 0.5 images/s |

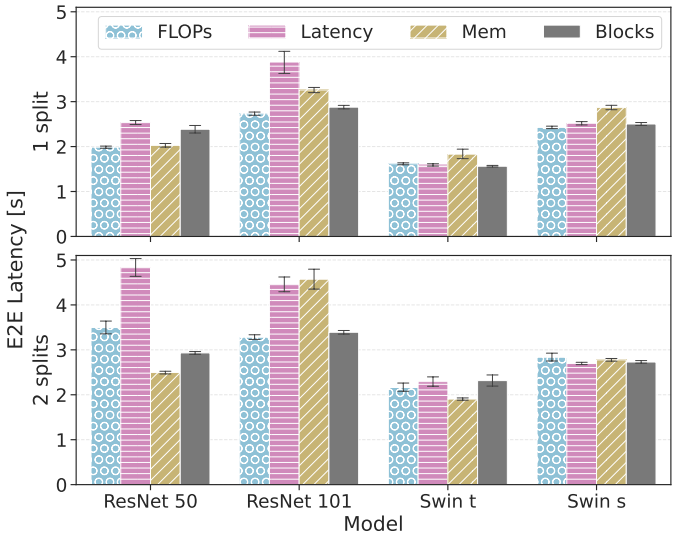



Figure 11: E2E latency using different splitting criteria. The error bars show the 95% confidence intervals.

### 7.1. Identifying the Main Delay Sources

For each of the four evaluated models, Figure 12 presents delay breakdown for each of the four considered models. The figure illustrates the total accumulated delays across the different nodes of the split, categorized into: model inference time, queueing delays, the overhead of the Matrix messaging protocol, data upload time to the Matrix server, and data transfer times between the servers. Figure 12

[2] `https://gitlab.lip6.fr/ie6/conduit`
[3] Bitrates measured with no other competing traffic.

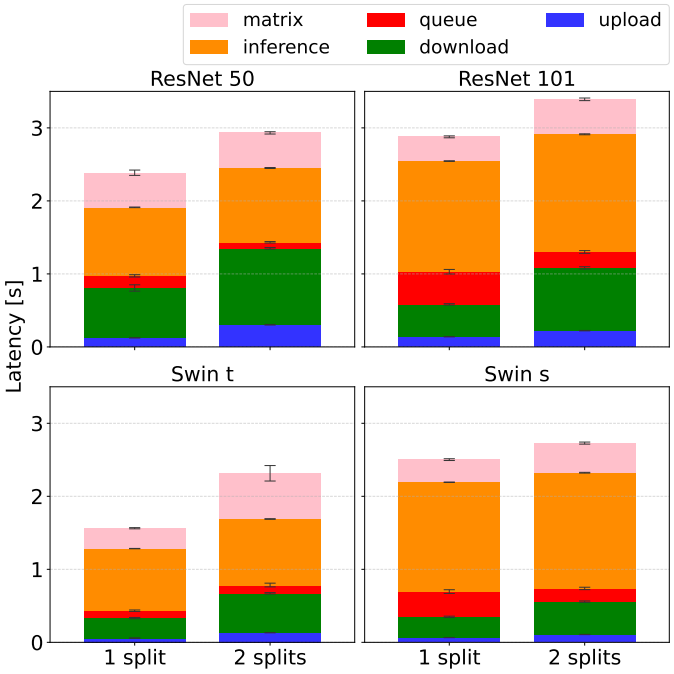



Figure 12: Total latency of each task in the SC inference pipeline, in deployments involving a single split (top) and two splits (bottom). We compute the Matrix protocol's overhead by subtracting the sum of the measured individual latencies from the E2E latency. Results obtained using a SC split criterion that splits DNN blocks equally among nodes.

shows in more detail the sources of delay of the deployments using the "block" criterion in Figure 11. Focusing solely on the inference time (in orange), which reflects the time spent with DNN computations, we observe minimal difference between adopting a configuration with one or two splits. Excluding ResNet101, which observed an increase of 120 ms when transitioning from 1 to 2 splits, all models observed discrepancies inferior to 100 ms. As anticipated, delays due to intermediate data exchange (upload and download) increase with the number of splits, nearly doubling across all models analyzed. For instance, Swin *s* exhibited the smallest discrepancies. Indeed, upload latency increases from 65 ms to 105 ms with an extra split, and download latency increases from 264 ms to 418 ms. However, we observe that the queuing delay almost halves as the number of splits increases. With ResNet101, we see a decrease from 360 ms with a single split to 184 ms with an extra split. This occurs because placing fewer layers on each node reduces the node's service time, making the SC pipeline more resilient to closely spaced arrivals. In particular, we observe that these latency changes cancel out with the ResNet101, making the E2E latency of the two configurations very similar. Finally, Figure 12 shows that the Matrix protocol's overhead affects E2E latency, increasing with the addition of an extra node. Still, the added delay is significantly lower than the time required to download feature maps.

### *7.2. Minimizing Data Transmission*

Figure 13 shows in finer detail the latency distribution among nodes and procedures when using ResNet50. If we exclude the time spent on data transmission between nodes, we see that `Server0` has the highest inference and queue times among all nodes across all configurations, except for the one that minimizes in-node computation latency. This is because this node handles higher-dimensional data. DNNs designed for image classification tend to employ severe dimensionality reduction procedures [42], which can reduce data size as it moves towards the output layer. This is evident when we focus on the criterion that minimizes in-node computation latency during the transmission of raw feature maps. In this configuration, `Server0` takes longer to transfer data to its Matrix server (upload) and to transmit data to `Server1` (download) because it forwards feature maps from earlier layers than its counterparts do.

Figures 13 and 14 indicate that the main direction for improving SC inference latency in WMN is minimizing the size of the feature maps that are transmitted between servers. Particularly for ResNet models, compressing feature maps before transmission reduced latency by almost 250 ms across all configurations, and by more than 500 ms when quantization was employed. These changes lead to a latency gain of more than 1 s with ResNet50 and almost 500 ms with ResNet101. Data generated by ResNet models tends to be sparse, allowing us to significantly reduce the amount of data transmitted between nodes, and thereby reduce the time spent on these transmissions. If, with ResNet models, we were able to reduce download and upload latency with compression alone, we were not able to replicate these improvements with Swin models. As seen in Figure 14, there is no latency improvement when we only compress feature maps. These DNNs produce feature maps with lower sparsity and thus are less compressible. Only after employing quantization were we able to reduce the download latency, although less effectively than with the other set of DNNs. Specifically, we observe reductions of at least 200 ms in download time. This leads to a marginal E2E latency improvement of at least 100 ms with Swin *s* configurations, and almost 500 ms with all Swin *t* configurations excluding the criterion that minimizes memory footprint.

### *7.3. SC Optimized for Communication vs RR*

Figure 15 presents a comparison between the SC optimized for communication, which includes compression and linear quantization of feature maps, against the SC and RR configurations seen in Section 3 when the RPi4*B* are unconstrained. Using the four models from the previous experiment, we see that optimizing the split DNN for communication can minimize the latency differences between SC and RR. Although the modified SC implementation remains slower than the RR approach, the modifications reduce the difference to less than 1 s when the average image sending rate is at most 1 image/s. And at 2 images/s, ResNet50 with the modified SC deployment reduced E2E

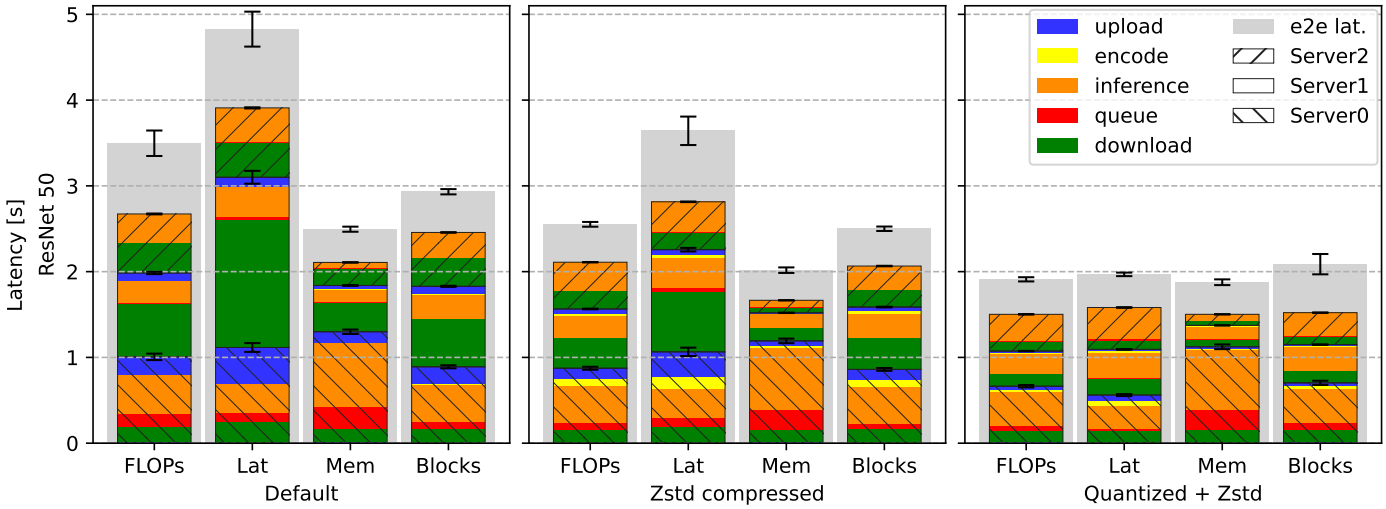



Figure 13: Latency measurements of each task in a SC configuration with two splits. The client generates inference requests according to a Poisson process with a rate of 0.5 image/s. The experiment was repeated 20 times, with 100 images transmitted per run.

latency by 70% compared to the original deployment, while the other models achieved a 40% reduction. However, as discussed previously, reducing latency via quantization can degrade DNN accuracy. Still, if the emergency application tolerates a small degradation in accuracy, the resulting SC implementation minimizes the impact of the delay inherent to WMN, thereby retaining the advantage of a DNN implementation that is more robust to dynamic and extreme memory pressure.

## 8. Additional Deployment Considerations

In the previous experiments, images are transmitted in regular intervals or as a stationary Poisson process. Although it enables better analysis of the sources of latency in the SC pipeline, this choice might not fully reflect real emergency scenarios. In particular, we need to ensure operability under bursty states in which multiple requests arrive in a short period of time. Additionally, a key takeaway from the previous section is that a key factor for SC deployment is reducing the amount of data transmitted between DNN partitions. Finally, power consumption is an important concern in edge deployments, particularly when employing battery-powered devices. To this end, we perform additional experiments to draw additional deployment guidelines focusing on each scenario. Specifically, in this section, all results use 3-node configurations (2-split SC and RR with 3 devices).

### *8.1. SC under bursty traffic*

We model the inference request generation as a two-state Markov-modulated Poisson process (MMPP) [43] that alternates between a low-rate state in which images are transmitted at 0.2 Hz mean rate, and a high-rate state that varies from 1 Hz to 4 Hz. Specifically, we set the transition rate from the low to high state to be equal to 0.02 and 0.2 on the opposite transition, yielding a mean sojourn time of 5 s in the high-rate state and 50 s in the low-rate state [44]. Figure 16 shows that the SC configurations are more sensitive to sudden increases in inference requests than the RR-based deployment, particularly the configurations that adopted the memory-based split criterion. Recalling the results from the previous section, the memory-based splits generated deployments where the first node experienced the highest computational latency. This suggests that requests piled up in the first RPi, thus slowing down the processing of the full SC pipeline. Remarkably, the configurations that adopt feature map compression deviated less from the RR configurations than the default deployment, suggesting that compressing intermediate results can also make the SC pipeline more robust to sudden increases in inference requests. For instance, if we take ResNet101 with a memory-based split as an example, in the default case the E2E latency reached a mean value of approximately 40 s when the bursty state had 4 Hz; the modified version is just above 10 s, approximately double the E2E latency obtained by the RR configuration. Still, when we compare the SC deployments with the load balancing approach, we see that the latter remains the preferable deployment strategy.

### *8.2. SC compression and the latency-accuracy trade-off*

One of the main findings of the previous section is that communication overhead has a greater impact on the SC pipeline than split-point optimization in our setup. However, these techniques can introduce accuracy degradations, which can be particularly problematic in emergency scenarios. Moreover, the encoding-decoding latencies can impact the inference pipeline's E2E latency. To illustrate this, Figure 17 compares the compression scheme adopted in the previous experiments (zstandard with linear quantization) against the same compressor combined with two layer-wise sparsification mechanisms, and neural network coding (NNCodec) [45]. The first sparsification consists of masking (zeroing out) the least-active feature map regions,

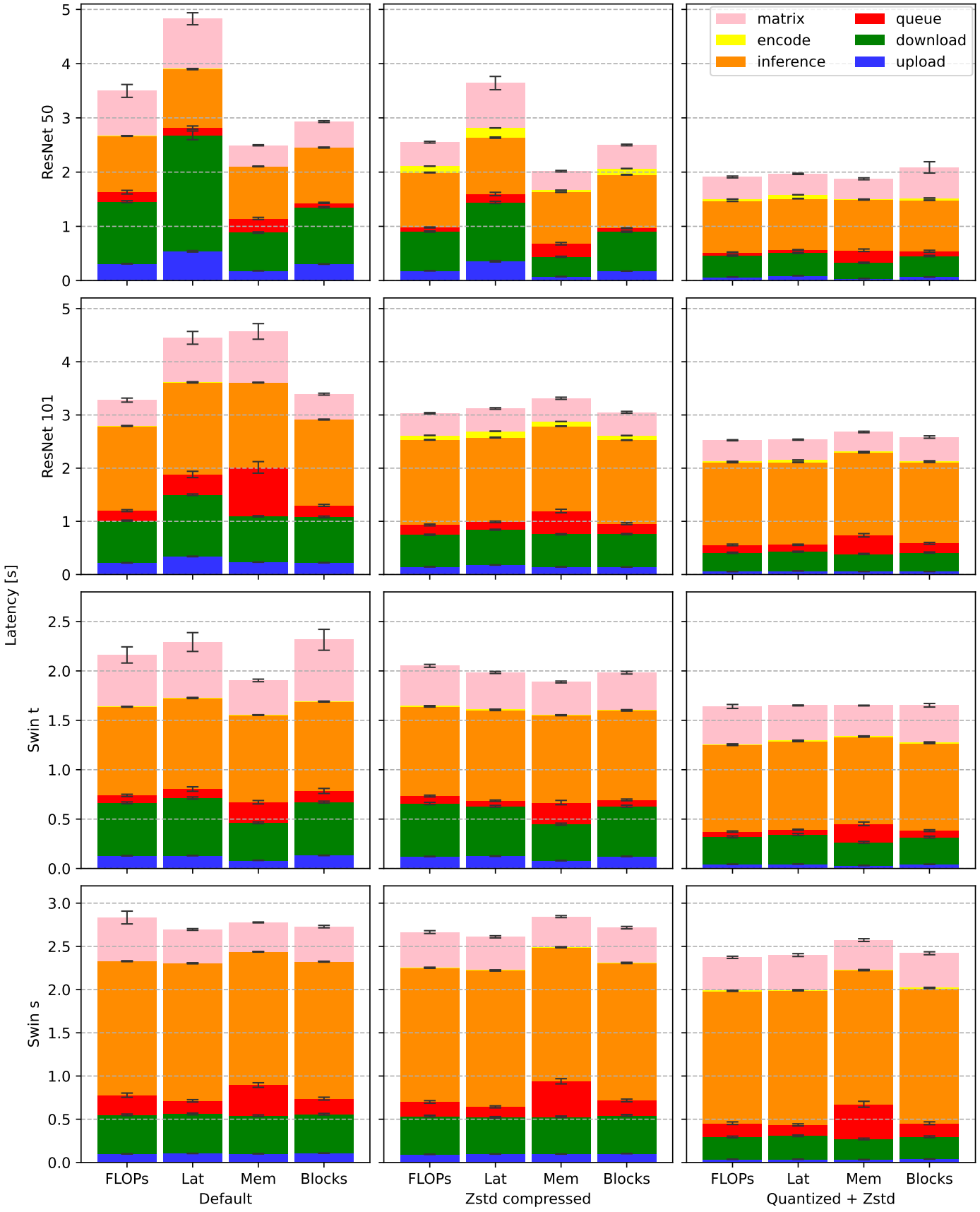



Figure 14: Total latency of each task in a two-split configuration using different split criteria and compression schemes.

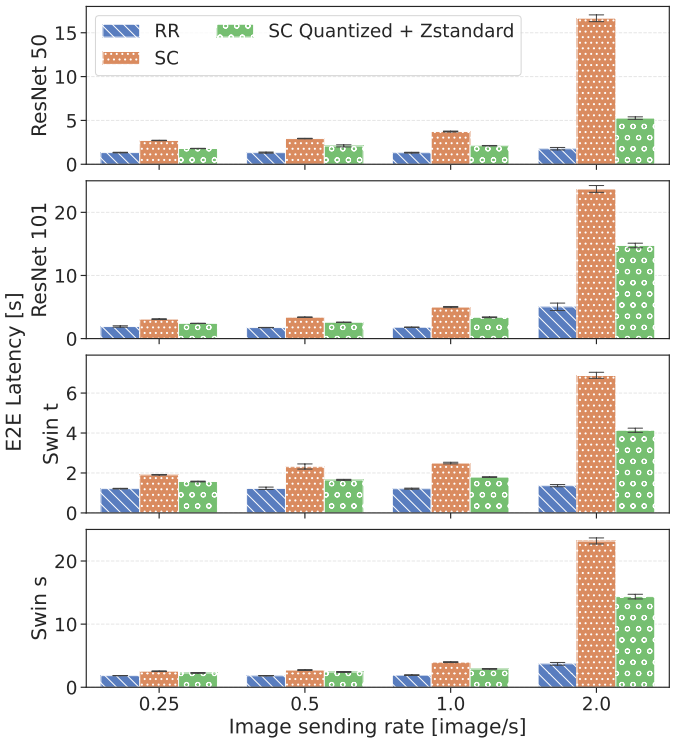



Figure 15: E2E comparison between RR and SC using three nodes. The error bars show the 95% confidence intervals.

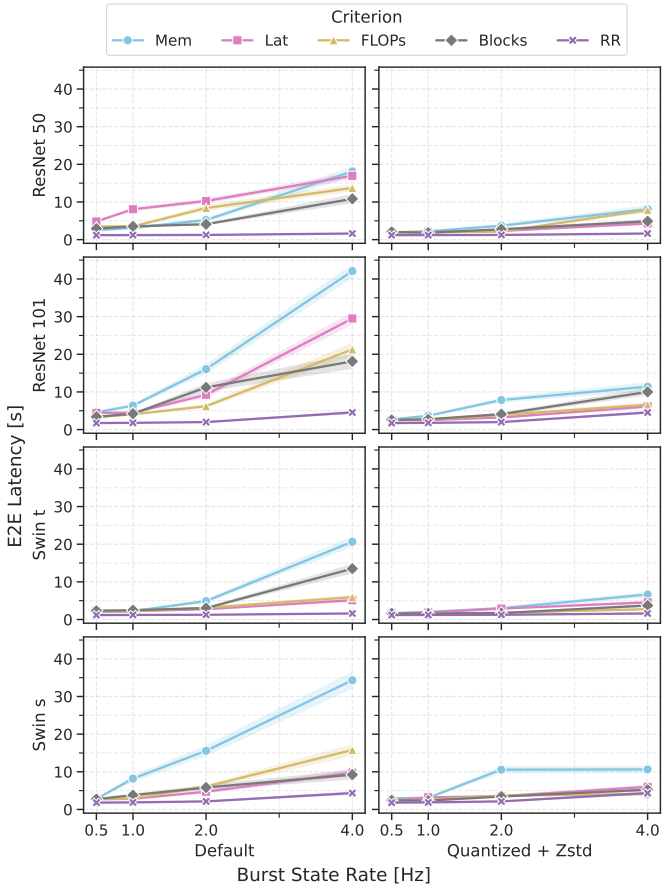



Figure 16: E2E latency when image transmission is modeled as an MMPP. SC deployments adopting feature map compression seem more resilient to bursty arrivals, exhibiting a smaller latency increase than the configurations that transmit raw feature maps. Although the RR deployments do not employ any encoding operation, we show them in both sets of plots as a baseline.

measured on a calibration dataset, to achieve the target sparsity ratio. The second technique generates a more refined mask using layer-wise relevance propagation (LRP), which accounts for the impact of each feature map element on subsequent DNN layers [46]. Finally, the last one is a codec that implements the *ISO/IEC 15938-17 (Neural Network Coding)* [45, 47], a standard for the compressed representation of neural network parameters. All schemes conform to the plug-and-play requirements of our deployment scenario and do not require retraining of DNNs, except for the offline calibration phase for mask generation in the sparsification schemes.

Unlike the previous experiments, we run each configuration twice, using 500 different images in each run. Sparsification masks were generated using 500 images, different from the ones used in the experiments. Figure 17 shows the compression-related metrics and their impact on accuracy and E2E. NNCodec achieves the best results in terms of accuracy, even outperforming the default configuration by 0.6 percentage points, achieving compression ratios of 7.96× and 10.28× in the first and last splits, respectively. This significantly reduced download latency; however, the encoding time of the compressor outweighed the transmission advantages. Alternatively, the sparsification schemes achieved the lowest data sizes and transmission latency with negligible encoding latency, reducing E2E latency by 100 ms compared to the previous encoding setup. Still, these latency gains come with a significant accuracy degradation: the simplest scheme drops more than 28 percentage points, whereas the LRP drops almost 9 percentage points. This issue tends to get worse with larger sparsity ratios. Sparsification-aware training, such as the one presented in [46], can minimize this impact, at the cost of extra steps in the deployment procedure. Still, these results illustrate that there is always a trade-off between transmission efficiency, retaining DNN performance, and having a fast encoding procedure that does not slow down the SC pipeline.

### *8.3. DNN Service Power Consumption*

Although this paper focuses on inference latency, power consumption is a critical, non-negligible metric in disaster-emergency scenarios, especially when edge devices are battery-powered. We performed a power-consumption profiling of the RPis employed in our testbed under each SC configuration from Section 7. We measured the fundamental electrical metrics (voltage, current, power and energy) using Joulescope's JS220+ Precision Energy Analyser[3].

[3] `https://download.joulescope.com/products/JS220/JS220-K002/users_guide/Joulescope%20JS220%20Plus%20User%27s%20Guide%20v1_1.pdf`

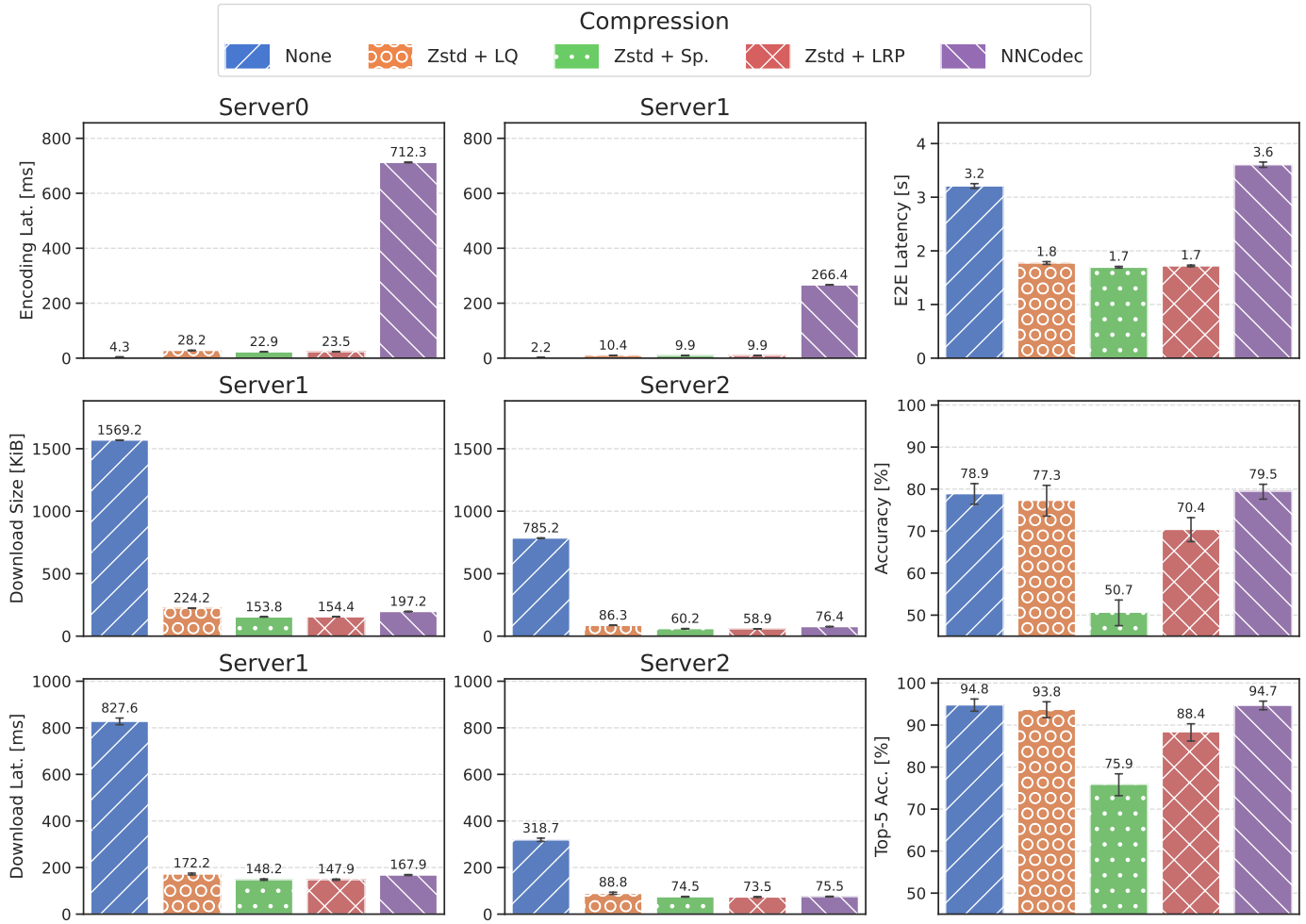



Figure 17: Comparison between different compression methods using a ResNet50 partitioned across three nodes. Leftmost plots show transmission-related latencies and data sizes, whereas the rightmost plots show the E2E latencies and accuracy results of the SC application. Results obtained for the block-based split configuration, with sparsification schemes and NNCodec using a sparsity rate of 0.5.

Table 3 shows mean over 6 repetitions of an experiment transmitting 100 images transmitted at a regular interval of one image being transmitted every 2 s (200 s per run). We omit voltage, as it remains approximately equal to 5.2 V in all configurations.

First, our measurements show that running the DNN and communication services on an otherwise idle RPi adds about 100 mW over the bare baseline when the servers are not actively processing inference requests. When processing inference requests, we observe a power consumption increase of 0.77–1.72 W across all servers, accounting for an additional 154–343 J over the same 200 s interval. When comparing the RR deployment and the SC configurations, we observe that the RR deployment has the lowest mean energy consumption per server because it receives a portion of the inference requests (a third, in this case), distributes computations more evenly, and does not transmit feature maps between the nodes. In particular, the last two factors make power consumption across SC servers more uneven, as it depends on the number of layers stored on each server and the size of the intermediate data it must transmit.

Next, SC configurations that use compression techniques (particularly those that perform quantization) show lower power consumption than their uncompressed counterparts. This highlights another issue with transmitting raw feature maps: high-dimensional data can cause significant power consumption. Finally, the results with the memory-based split, which produced the most unbalanced layer allocation, show an interesting direction for SC. Server1 and Server2, which received fewer layers than Server0, consume less power than the RR servers. This suggests that SC configurations with more splits that reduce the number of layers stored in each node can lower individual-node power consumption. Still, as shown previously, more splits increase inference latency, making feature map minimization vital to the success of SC deployments.

## 9. Conclusion

This paper investigates the deployment of DNNs in resource-constrained edge networks. We observed that as the memory footprint of these models approaches the avail-

Table 3: Mean results of the fundamental electrical metrics of each configuration, with the (mean) energy consumption throughout the 200 s.

| Configuration | Node | Current (A) | Power (W) | Energy (J) |
|---|---|---|---|---|
| RPi4B Baseline | – | $0.301 \pm 0.003$ | $1.577 \pm 0.017$ | $315.47 \pm 3.46$ |
| Idle Server | – | $0.322 \pm 0.004$ | $1.685 \pm 0.020$ | $337.02 \pm 3.94$ |
| RR | Server0 | $0.490 \pm 0.003$ | $2.554 \pm 0.017$ | $510.87 \pm 3.42$ |
| | Server1 | $0.481 \pm 0.003$ | $2.509 \pm 0.013$ | $501.87 \pm 2.60$ |
| | Server2 | $0.491 \pm 0.001$ | $2.563 \pm 0.008$ | $512.58 \pm 1.53$ |
| | Sever Mean | $0.487 \pm 0.005$ | $2.542 \pm 0.027$ | $508.44 \pm 5.43$ |
| Blocks Split | Server0 | $0.592 \pm 0.005$ | $3.087 \pm 0.024$ | $617.36 \pm 4.77$ |
| | Server1 | $0.543 \pm 0.003$ | $2.829 \pm 0.016$ | $565.82 \pm 3.10$ |
| | Server2 | $0.517 \pm 0.004$ | $2.697 \pm 0.022$ | $539.49 \pm 4.39$ |
| | Sever Mean | $0.551 \pm 0.032$ | $2.871 \pm 0.168$ | $574.22 \pm 33.51$ |
| Blocks Split Zstd | Server0 | $0.579 \pm 0.003$ | $3.016 \pm 0.015$ | $603.10 \pm 2.99$ |
| | Server1 | $0.518 \pm 0.007$ | $2.702 \pm 0.038$ | $540.32 \pm 7.58$ |
| | Server2 | $0.500 \pm 0.002$ | $2.610 \pm 0.012$ | $522.01 \pm 2.37$ |
| | Sever Mean | $0.532 \pm 0.035$ | $2.776 \pm 0.180$ | $555.14 \pm 36.03$ |
| Blocks Split Quantized + Zstd | Server0 | $0.540 \pm 0.001$ | $2.813 \pm 0.005$ | $562.67 \pm 0.95$ |
| | Server1 | $0.491 \pm 0.001$ | $2.561 \pm 0.004$ | $512.12 \pm 0.86$ |
| | Server2 | $0.501 \pm 0.001$ | $2.615 \pm 0.004$ | $523.04 \pm 0.84$ |
| | Sever Mean | $0.511 \pm 0.022$ | $2.663 \pm 0.112$ | $532.61 \pm 22.36$ |
| FLOPs Split | Server0 | $0.591 \pm 0.005$ | $3.081 \pm 0.024$ | $616.18 \pm 4.87$ |
| | Server1 | $0.538 \pm 0.001$ | $2.804 \pm 0.004$ | $560.89 \pm 0.89$ |
| | Server2 | $0.514 \pm 0.002$ | $2.679 \pm 0.009$ | $535.71 \pm 1.83$ |
| | Sever Mean | $0.548 \pm 0.034$ | $2.855 \pm 0.174$ | $570.93 \pm 34.70$ |
| FLOPs Split Zstd | Server0 | $0.581 \pm 0.001$ | $3.026 \pm 0.007$ | $605.10 \pm 1.40$ |
| | Server1 | $0.513 \pm 0.002$ | $2.678 \pm 0.013$ | $535.66 \pm 2.52$ |
| | Server2 | $0.518 \pm 0.004$ | $2.702 \pm 0.023$ | $540.31 \pm 4.57$ |
| | Sever Mean | $0.537 \pm 0.032$ | $2.802 \pm 0.164$ | $560.36 \pm 32.75$ |
| FLOPs Split Quantized + Zstd | Server0 | $0.539 \pm 0.001$ | $2.809 \pm 0.005$ | $561.80 \pm 1.06$ |
| | Server1 | $0.479 \pm 0.002$ | $2.502 \pm 0.009$ | $500.35 \pm 1.81$ |
| | Server2 | $0.507 \pm 0.003$ | $2.645 \pm 0.017$ | $528.98 \pm 3.36$ |
| | Sever Mean | $0.508 \pm 0.025$ | $2.652 \pm 0.130$ | $530.38 \pm 25.92$ |
| Latency Split | Server0 | $0.620 \pm 0.003$ | $3.229 \pm 0.015$ | $645.85 \pm 3.07$ |
| | Server1 | $0.580 \pm 0.009$ | $3.024 \pm 0.046$ | $604.85 \pm 9.24$ |
| | Server2 | $0.532 \pm 0.004$ | $2.776 \pm 0.020$ | $555.18 \pm 3.99$ |
| | Sever Mean | $0.578 \pm 0.037$ | $3.010 \pm 0.193$ | $601.96 \pm 38.57$ |
| Latency Split Zstd | Server0 | $0.593 \pm 0.002$ | $3.092 \pm 0.011$ | $618.44 \pm 2.18$ |
| | Server1 | $0.542 \pm 0.002$ | $2.825 \pm 0.009$ | $564.91 \pm 1.80$ |
| | Server2 | $0.531 \pm 0.002$ | $2.766 \pm 0.011$ | $553.24 \pm 2.12$ |
| | Sever Mean | $0.555 \pm 0.028$ | $2.894 \pm 0.146$ | $578.87 \pm 29.27$ |
| Latency Split Quantized + Zstd | Server0 | $0.523 \pm 0.000$ | $2.728 \pm 0.002$ | $545.59 \pm 0.48$ |
| | Server1 | $0.497 \pm 0.001$ | $2.594 \pm 0.005$ | $518.82 \pm 0.90$ |
| | Server2 | $0.527 \pm 0.003$ | $2.748 \pm 0.016$ | $549.64 \pm 3.22$ |
| | Sever Mean | $0.516 \pm 0.014$ | $2.690 \pm 0.071$ | $538.02 \pm 14.19$ |
| Memory Split | Server0 | $0.632 \pm 0.010$ | $3.292 \pm 0.054$ | $658.33 \pm 10.76$ |
| | Server1 | $0.483 \pm 0.004$ | $2.520 \pm 0.022$ | $504.00 \pm 4.36$ |
| | Server2 | $0.451 \pm 0.002$ | $2.354 \pm 0.012$ | $470.79 \pm 2.34$ |
| | Sever Mean | $0.522 \pm 0.082$ | $2.722 \pm 0.422$ | $544.38 \pm 84.32$ |
| Memory Split Zstd | Server0 | $0.630 \pm 0.003$ | $3.282 \pm 0.016$ | $656.40 \pm 3.26$ |
| | Server1 | $0.464 \pm 0.001$ | $2.422 \pm 0.005$ | $484.38 \pm 1.01$ |
| | Server2 | $0.449 \pm 0.001$ | $2.345 \pm 0.008$ | $469.06 \pm 1.56$ |
| | Sever Mean | $0.514 \pm 0.085$ | $2.683 \pm 0.437$ | $536.61 \pm 87.42$ |
| Memory Split Quantized + Zstd | Server0 | $0.599 \pm 0.003$ | $3.122 \pm 0.015$ | $624.38 \pm 3.01$ |
| | Server1 | $0.452 \pm 0.005$ | $2.361 \pm 0.025$ | $470.29 \pm 7.27$ |
| | Server2 | $0.450 \pm 0.001$ | $2.349 \pm 0.003$ | $469.78 \pm 0.54$ |
| | Sever Mean | $0.500 \pm 0.072$ | $2.611 \pm 0.372$ | $521.49 \pm 74.99$ |

able capacity of edge servers, the overhead introduced by memory swapping results in substantial performance degradation. Specifically, our preliminary experiments show a significant inference latency increase when swap memory usage plateaus at a specific value, which is around 250 MiB in the RPi4B devices analysed. While our study focuses on relatively small DNNs deployed at the edge, the ongoing trend toward larger model sizes implies that these deployment challenges will only become more widespread. To address this, our real-world testbed experiments confirm that SC emerges as a promising alternative. However, our experiments reveal that the communication cost is the dominant bottleneck, regardless of the model or the specific split criterion chosen. This suggests that we should prioritize minimizing the size of feature maps exchanged throughout the SC pipeline rather than focusing solely on optimizing where to split the model. A detailed analysis of processing times shows that applying feature map compression and quantization prior to transmission is critical to mitigating communication costs and making SC viable. Ultimately, we argue that future optimization efforts in SC should pivot toward strategies that minimize these network exchanges without sacrificing accuracy, rather than focusing solely on split-point optimization.

## Acknowledgments

This work was supported by Bpifrance IE6 - France 2030 "Telecom 5G", CNPq (310234/2025-5, 407304/2025-8, 408255/2023-4, and 405940/2022-0), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – Funding Code 001 and project numbers 88887.954253/2024-00, 88887.987121/2024-00, and 88881.987122/2024-01, COFECUB project number Ma1074/25, FAPERJ (E-26/200.438/2026 and E-26/210.778/2025), FAPESP (2023/00673-7 and 2023/00811-0), Fundação de Desenvolvimento da Pesquisa (Fundep) Rota 2030, Stellantis and Mobway.

## References

[1] C. Kyrkou, P. Kolios, T. Theocharides, M. Polycarpou, Machine learning for emergency management: A survey and future outlook, Proceedings of the IEEE 111 (1) (2022) 19–41.

[2] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, M. Parmar, A review of convolutional neural networks in computer vision, Artificial Intelligence Review 57 (4) (2024) 99.

[3] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, D. Roth, Recent advances in natural language processing via large pre-trained language models: A survey, ACM Computing Surveys 56 (2) (2023) 1–40.

[4] V. Chamola, V. Hassija, S. Gupta, A. Goyal, M. Guizani, B. Sikdar, Disaster and pandemic management using machine learning: a survey, IEEE Internet of Things Journal 8 (21) (2020) 16047–16071.

[5] Y. Wang, Z. Su, N. Zhang, D. Fang, Disaster relief wireless networks: Challenges and solutions, IEEE Wireless Communications 28 (5) (2021) 148–155.

[6] Y. Wang, C. Yang, S. Lan, L. Zhu, Y. Zhang, End-edge-cloud collaborative computing for deep learning: A comprehensive survey, IEEE Communications Surveys & Tutorials (2024).

[7] A. Carreras-Coch, J. Navarro, C. Sans, A. Zaballos, Communication technologies in emergency situations, Electronics 11 (7) (2022) 1155.

[8] J. Bravo-Arrabal, C. S. Alvarez-Merino, J. J. Fernández-Lozano, J. A. Gómez-Ruiz, A. Mandow, R. Barco, A. Garcia-Cerezo, Field report on experimental comparison of a wifi mesh network against commercial 5g in an underground disaster environment, in: 2023 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), IEEE, 2023, pp. 186–192.

[9] M. Sheng, X. Chen, J. Liu, J. Li, T. Q. Quek, Towards disaster-resistant cellular communication networks based on network capacity scalability, IEEE Transactions on Wireless Communications (2025).

[10] H. Han, J. Siebert, Tinyml: A systematic review and synthesis of existing research, in: 2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), IEEE, 2022, pp. 269–274.

[11] M. S. Gilbert, M. L. R. de Campos, M. E. M. Campista, Asymmetric Autoencoders: An NN alternative for resource-constrained devices in IoT networks, Ad Hoc Networks (2024) 103412.

[12] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, K. Keutzer, A survey of quantization methods for efficient neural network inference, in: Low-power computer vision, Chapman and Hall/CRC, 2022, pp. 291–326.

[13] Y. Matsubara, M. Levorato, F. Restuccia, Split computing and early exiting for deep learning applications: Survey and research challenges, ACM Computing Surveys 55 (5) (2022) 1–30.

[14] S. Scardapane, M. Scarpiniti, E. Baccarelli, A. Uncini, Why should we add early exits to neural networks?, Cognitive Computation 12 (5) (2020) 954–966.

[15] Z. Liao, W. Hu, J. Huang, J. Wang, Joint multi-user dnn partitioning and task offloading in mobile edge computing, Ad Hoc Networks 144 (2023) 103156.

[16] S. Tuli, G. Casale, N. R. Jennings, Splitplace: Ai augmented splitting and placement of large-scale neural networks in mobile edge environments, IEEE Transactions on Mobile Computing 22 (9) (2022) 5539–5554.

[17] P. Li, E. Koyuncu, H. Seferoglu, Adaptive and resilient model-distributed inference in edge computing systems, IEEE Open Journal of the Communications Society 4 (2023) 1263–1273.

[18] I. F. Akyildiz, X. Wang, W. Wang, Wireless mesh networks: a survey, Computer networks 47 (4) (2005) 445–487.

[19] H. Li, X. Hao, J. Wen, F. Liu, Y. Zhang, Multi-objective deployment of uavs for multi-hop fanet: Uav-assisted emergency vehicular network, Drones 8 (6) (2024) 262.

[20] A. Furutanpey, P. Raith, S. Dustdar, Frankensplit: Efficient neural feature compression with shallow variational bottleneck injection for mobile edge computing, IEEE Transactions on Mobile Computing 23 (12) (2024) 10770–10786.

[21] Y. Matsubara, D. Callegaro, S. Singh, M. Levorato, F. Restuccia, Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing, in: 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), IEEE, 2022, pp. 337–346.

[22] Y. Matsubara, R. Yang, M. Levorato, S. Mandt, Sc2 benchmark: Supervised compression for split computing, arXiv preprint arXiv:2203.08875 (2022).

[23] H. Rahmath P, V. Srivastava, K. Chaurasia, R. G. Pacheco, R. S. Couto, Early-exit deep neural network-a comprehensive survey, ACM Computing Surveys 57 (3) (2024) 1–37.

[24] A. Yarali, B. Ahsant, S. Rahman, Wireless mesh networking: A key solution for emergency & rural applications, in: 2009 Second International Conference on Advances in Mesh Networks, IEEE, 2009, pp. 143–149.

[25] H. Hao, C. Xu, W. Zhang, S. Yang, G.-M. Muntean, Task-driven priority-aware computation offloading using deep reinforcement learning, IEEE Transactions on Wireless Communications 24 (10) (2025) 8114–8128.

[26] H. Hao, C. Xu, W. Zhang, X. Chen, S. Yang, G.-M. Muntean, Reliability-aware optimization of task offloading for uav-assisted edge computing, IEEE Transactions on Computers (2025).

[27] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[28] T. maintainers, contributors, Torchvision: Pytorch's computer vision library, `https://github.com/pytorch/vision` (2016).

[29] T. Heo, Control Group v2, `https://docs.kernel.org/admin-guide/cgroup-v2.html`, accessed: 2026-01-19 (2015).

[30] M. Kerrisk, malloc(3) — Linux manual page (Sep. 2025).
URL `https://man7.org/linux/man-pages/man3/malloc.3.html`

[31] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, B. Guo, Swin transformer: Hierarchical vision transformer using shifted windows, in: Proceedings of the IEEE/CVF international conference on computer vision, 2021, pp. 10012–10022.

[32] K. Wang, J. Cao, Z. Zhou, Z. Li, Swapnet: Efficient swapping for dnn inference on edge ai devices beyond the memory budget, IEEE Transactions on Mobile Computing 23 (9) (2024) 8935–8950.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE conference on computer vision and pattern recognition, Ieee, 2009, pp. 248–255.

[34] Green Communications, YOI datasheet, https://www.green-communications.fr/wp-content/uploads/2024/09/YOIRouter.pdf, accessed: 2026-03-10 (2024).

[35] Matrix.org Foundation C.I.C., Matrix specification, https://spec.matrix.org, accessed: 2025-07-25; version v1.14 (2025).

[36] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, N. D. Lane, Spinn: synergistic progressive inference of neural networks over device and cloud, in: Proceedings of the 26th annual international conference on mobile computing and networking, 2020, pp. 1–15.

[37] S. S. Skiena, Dynamic Programming, Springer International Publishing, Cham, 2020, pp. 307–353. doi:10.1007/978-3-030-54256-6_10.
URL https://doi.org/10.1007/978-3-030-54256-6_10

[38] Y. Collet, M. Kucherawy, Zstandard compression and the application/zstd media type, Tech. rep. (2018).

[39] D. Carra, G. Neglia, Dnn split computing: Quantization and run-length coding are enough, in: GLOBECOM 2023-2023 IEEE Global Communications Conference, IEEE, 2023, pp. 7357–7362.

[40] T. H. Clausen, P. Jacquet, Optimized Link State Routing Protocol (OLSR), RFC 3626 (Oct. 2003). doi:10.17487/RFC3626.
URL https://www.rfc-editor.org/info/rfc3626

[41] Green Communications, GreenSoft: Software suite to create an internet of edge on customer hardware, https://www.green-communications.fr/wp-content/uploads/2025/07/GreenSoft.pdf, accessed: 2026-03-10 (2025).

[42] L.-C. Chen, G. Papandreou, F. Schroff, H. Adam, Rethinking atrous convolution for semantic image segmentation (2017). arXiv:1706.05587.

[43] W. Fischer, K. Meier-Hellstern, The markov-modulated poisson process (mmpp) cookbook, Performance evaluation 18 (2) (1993) 149–171.

[44] H. Heffes, D. Lucantoni, A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance, IEEE Journal on selected areas in communications 4 (6) (1986) 856–868.

[45] D. Becking, P. Haase, H. Kirchhoffer, K. Müller, W. Samek, D. Marpe, NNCodec: An open source software implementation of the neural network coding ISO/IEC standard, in: ICML 2023 Workshop Neural Compression: From Information Theory to Applications, 2023.
URL https://openreview.net/forum?id=5VgMDKUgX0

[46] M. A. Höfler, D. Becking, K. Müller, D. Marpe, W. Samek, Relevance-guided activation sparsification for bandwidth-efficient collaborative inference, in: 2025 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), IEEE, 2025, pp. 3104–3113.

[47] ISO/IEC, Information technology — multimedia content description interface — part 17: Compression of neural networks for multimedia content description and analysis (2024).
URL https://www.iso.org/standard/85545.html