Communication-Aware DNN Pruning

Tong Jian, Debashri Roy, Batool Salehi, Nasim Soltani, Kaushik Chowdhury, Stratis Ioannidis
Electrical and Computer Engineering Department, Northeastern University, Boston, MA, USA

Abstract—We propose a Communication-aware Pruning (CaP) algorithm, a novel distributed inference framework for distributing DNN computations across a physical network. Departing from conventional pruning methods, CaP takes the physical network topology into consideration and produces DNNs that are communication-aware, designed for both accurate and fast execution over such a distributed deployment. Our experiments on CIFAR-10 and CIFAR-100, two deep learning benchmark datasets, show that CaP beats state of the art competitors by up to 4% w.r.t. accuracy on benchmarks. On experiments over real-world scenarios, it simultaneously reduces total execution time by 27%–68% at negligible performance decrease (less than 1%).

Index Terms—Distributed Inference, Model Pruning

I. INTRODUCTION

With billions of pervasively deployed and connected IoT devices, combined sensing and fusion can be used towards massively distributed and diverse inference tasks. As a motivating scenario, illustrated in Fig. 1 consider a smart city [1], [2], involving a ubiquitous deployment of networked sensors (cameras, LiDARs, acoustic sensors, wireless receivers etc.) at different locations across a large urban area. Data collected from these sensors can be used to perform a massive inference task, such as, e.g., the detection of an adverse event (a fire or an explosion) across the city, the assessment of environmental conditions such as temperature or pollution, the monitoring of traffic and detection of congestion, etc.

Deep Neural Networks (DNNs) have shown tremendous success at performing inference by fusing diverse, multi-modal inputs [3]–[6]. However, performing inference via DNNs at a such a massive scale lies at the frontier of current methods. The naïve approach, namely, sending collected data at a central location (e.g., the cloud, or a server belonging to a municipal authority), and executing the DNN there, does not scale; challenges arise when (a) raw data is large (e.g., video, LiDAR), (b) sensors are numerous and geographically dispersed, and (c) inference needs to be (near) real-time, and latency is a concern. In recent years, edge computing [7]–[9] has been proposed as a solution, bringing computation closer to the sensors, ideally reducing latency, accelerating inference, distributing computation, and addressing privacy concerns. However, distributing DNN inference over connected yet resource-constrained IoT edge devices, without a reduction of predictive accuracy, is a challenge. Existing network partitioning approaches for distributed inference are tailored to a fixed design of device connections, without considering their physical network topology: literature on partitioning a DNN across edge devices typically assumes all-to-all connections [10]–[13], or no connections until the last layer [14]–[17]; the former results in high latency in a networked environment, while the latter yields significant performance degradation compared to centralized computation.

In this work, we assume that, to reduce latency and accelerate computation, inference over a DNN is distributed across the network fabric, involving both edge and intermediate nodes of an arbitrary physical network. As illustrated in Figure 1, the DNN model can be partitioned into sub-networks, each of which is mapped to the physical network topology; our goal is to reduce latency while maintaining predictive performance.

Fig. 1: Motivation of our CaP framework. We consider a smart city as a motivating scenario, involving a ubiquitous deployment of networked sensors at different locations across a large urban area. To reduce latency and accelerate computation, inference over a DNN is distributed across the network fabric. Using CaP, the DNN model can be partitioned into sub-networks, each of which is mapped to the physical network topology; our goal is to reduce latency while maintaining predictive performance.

Our main contributions are as follows:
1) We propose a **Communication-aware Pruning (CaP)** algorithm, a novel deep learning framework for distributed inference. Our method simultaneously trains-and-prunes a DNN in a communication-aware fashion, using (a) a carefully-designed penalty capturing communication costs, and (b) additional constraints enforcing sparsity; these are jointly optimized using the Alternating Directions-Method of Multipliers [18], a technique often used in DNN pruning [19], [20].

2) We compare CaP with state-of-the-art (SOTA) network partitioning methods over parallel topology on two benchmarks, CIFAR-10 and CIFAR-100, and show that CaP has improved accuracy, beating SOTA methods by up to 4% with 8 devices. Our ablation study shows that CaP balances accuracy and communication costs favorably, in an adaptive, tunable fashion.

3) We illustrate the impact and the effectiveness of CaP via two real-life case studies, Radar Detection for spectrum sharing in the CBRS band and Multimodal Beamforming. In Radar Detection, we consider a scenario in which a spectrum access system wishes to detect the presence of a primary (radar) user through distributed inference over environmental sensing capability stations. In Multimodal Beamforming, inference over diverse sensors (GPS, LiDAR, and video) is used to determine the optimal beam of a wireless station. We thoroughly explore the end-to-end computation and communication cost of CaP in these settings. We show that CaP reduces the total execution time by 68% and 27% while incurring a negligible accuracy prediction performance drop, 0.75% and 0.24%, for radar detection and beamforming, respectively. This reduction enables real-time inference. We pledge to release our code and data to the community.

The remainder of this paper is structured as follows. We review related work in Section II. In Section IV, we present our method. Section V includes our experiments on both benchmarks and real-life case studies; we conclude in Section VI.

### II. Related Work

Partitioning a workflow, represented as a directed acyclic graph, and mapping it to a distributed computation infrastructure is a classic distributed computing topic [21]–[23]. Recently, several works have proposed algorithms specifically for the mapping of DNN computations over IoT devices, partitioned either horizontally [24], [25] or vertically [10]–[13]. Our approach can be combined with such methods (either classic and edge-specific) and is orthogonal, as it assumes that the mapping of the DNN to devices is a priori given.

Several works assume that the DNN is fixed (e.g., [10], [12], [24], [25]) and split across communicating devices in a manner that minimizes end-to-end execution. In contrast, weight-based partitioning methods aim to distribute a DNN over parallel, non-communicating workers. The disjoint sub-networks of the DNN are typically generated-and, as in our case, retrained-from a dense network via compression techniques, including low-rank factorization [26], knowledge distillation [27], [28] and pruning [18], [29]–[31]. For example, NoNN [15] compresses a large teacher into a set of parallel students; each student mimics one part of the teacher’s output layer through knowledge distillation, an approach also followed by ParaDis [17]. RePurpose [16] rearranges and partitions the neurons layer-wise so that connections among sub-networks are minimized through $\ell_0$-regularized pruning. SplitNet [14] splits the neural network in disjoint groups by regularizing both the class-to-group and feature-to-group assignments over weights. We depart in (a) considering a DNN execution over a networked topology, leading to a hierarchical partitioning and execution of the DNN’s dataflow, and, crucially (b) we allow for cross-communication across devices. We thus introduce a penalty tailored to the physical network topology; in practice, cross-communication yields significant accuracy advantages over (full) partitioning schemes (see Tables I and II in Sec. V).

### III. Problem Formulation

We assume that inference via a multi-layered DNN is to be performed over data collected by distributed networked sensors. Sensors are connected to each other as well as to a central inference point (e.g., the cloud or a central server) through a communication network. To accelerate computations, reduce communication, and leverage computational resources within the network, DNN computations are distributed across the network; this is illustrated in Fig. 2. As a result, communication costs accrue during DNN execution, affecting total execution time. Our goal is to account for this during training, producing a DNN that is communication-aware, designed for both fast and accurate execution over such a distributed deployment.

**Neural Network.** Formally, we are given an $L$-layer DNN $f_\theta : \mathbb{R}^{d_0} \to \mathbb{R}^{d_m}$ parameterized by $\theta \in \mathbb{R}^M$, that performs joint inference (i.e., fusion) over data collected by $K$ sensors. Data by each sensor $k \in \{1, \ldots, K\}$ is represented by a vector $x^k \in \mathbb{R}^{d_k}$, and the DNN operates on the concatenation of these inputs, i.e., on $x = [x^k]_{k=1}^K \in \mathbb{R}^{d_a}$, where $d_a = \sum_{k=1}^K d_k$. 

![Diagram](image-url)
We also represent the DNN by a directed acyclic graph $G_{DNN}(V_{DNN}, E_{DNN})$, comprising neurons $V_{DNN}$ and their respective neural connections $E_{DNN} \subseteq V_{DNN} \times V_{DNN}$. Note that each neural connection $e \in E_{DNN}$ is associated with a weight/parameter in $\theta$; we denote by $\theta(e)$ the coefficient of $\theta$ that corresponds to connection $e \in E_{DNN}$. We use subscript $t \in \{0, 1, \ldots, L\}$ to indicate layers, where $t = 0$ is the input ($x$). Then, the neurons, the neural connections, and the layer $l$ can be represented as $V_{DNN}^l \subseteq V_{DNN}$, $E_{DNN}^l = E_{DNN} \cap (V_{DNN}^l \times V_{DNN}^l)$, and $\theta_l(e) = \{\theta(e) : e \in E_{DNN}^l\}$, respectively.

Communication Network. We assume that, in the physical world, sensors are connected by a communication network. We represent this physical network topology by an undirected graph $G_{net}(V_{net}, E_{net})$, comprising network devices $V_{net}$ and communication links $E_{net} \subseteq V_{net} \times V_{net}$. Each communication link $e \in E_{net}$ is associated with a cost $c(e)$, capturing the cost of unit transmission (e.g., delay per bit) over link $e$.

Distributed Inference. The computation of the DNN is distributed across the nodes of the communication network. In particular, there exists a function $M : V_{DNN} \to V_{net}$ mapping neurons to devices on which the neurons are computed (see Fig. 2). For example, input neurons could map to edge devices, attached to the sensors, and all output neurons could map to a single (terminal) device, which computes the final inference outcome. W.l.o.g.2 we assume that direct communication between two devices that are assigned connected neurons is feasible; formally, the $M$ satisfies the following property:

**Property 1.** For all $(n, n') \in E_{DNN}$, if $M(n) \neq M(n')$, then $(M(n), M(n')) \in E_{net}$.

Communication Costs. To enable computation at inference time, each device stores all parameters/weights associated with neurons it needs to compute. Formally, device $v \in V_{net}$ stores $\theta(e)$ for all $e = (n', n) \in E_{DNN}$ s.t. $M(n) = v$.

Crucially, connected neurons that are mapped to distinct devices may incur communication costs at inference time. In particular, the communication cost of a neural connection $e = (n, n') \in E_{DNN}$ is given by:

$$C(e) = \begin{cases} 0, & \text{if } M(n) = M(n') \text{ or } \theta(e) = 0, \\ c(M(n), M(n')), & \text{otherwise.} \end{cases} \quad (1)$$

Throughout our analysis, we assume that (a) the mapping $M$ is predetermined (e.g., by algorithms such as [13], [16], [32]), and (b) devices are computationally homogenous, so that neuron computations take the same time, irrespective of the device computing them. Under these assumptions, our goal is to determine the DNN parameter weights $\theta$ so that (a) communication costs are minimized, while simultaneously (b) the accuracy of the DNN is not adversely impacted.

Communication-aware Pruning Problem. We assume that, at training time, we are given access to a dataset $D = \{(x_i, y_i)\}_{i=1}^N$, comprising samples with features $x_i \in \mathbb{R}^{d_x}$ and a corresponding labels $y_i \in \mathbb{R}^{d_y}$. Let $L(\theta) = \sum_{i=1}^N \ell(y_i, f_{\theta}(x_i))$ be the aggregate loss on the dataset, where $\ell$ is an appropriate loss (e.g., max-entropy, $l_2$-squared, etc.).

We wish to train DNN parameters $\theta$ as follows:

$$\text{Minimize: } L(\theta) + \lambda P(C, \theta), \quad (2)$$

where $P$ is a communication-aware regularization penalty, and $\lambda > 0$ is a regularization parameter controlling the tradeoff between accuracy and communication costs. Intuitively, by Eq. (1), we wish to design $P$ so that it encourages sparsity (i.e., setting $\theta(e) = 0$) along neural connections that are likely to incur large communication costs.

IV. COMMUNICATION-AWARE PRUNING

Ideally, we would like to set the weights in such a way so that the total delay due to communication is as small as possible. Assuming that transmissions per layer are parallel and thereby dominated by a lowest transmission, this is:

$$P_{\text{max}}(C, \theta) = \sum_{l=1}^L \max_{e \in E_{net}^l} 1(|\theta(e)| \neq 0) \cdot C(e). \quad (3)$$

Assuming that transmissions per layer contend with each other and happen sequentially, the penalty becomes:

$$P_{\text{count}}(C, \theta) = \sum_{e \in E_{net}} 1(|\theta(e)| \neq 0) \cdot C(e). \quad (4)$$

Intermediate penalties can capture partial contention between transmissions (e.g., when they share a communication link, or when links interfere); these can be expressed as a maximum across sums of costs, each sum being over contending subsets of transmissions. Nevertheless, this spectrum of penalties present significant implementation challenges during training. The main reason is that $1(|\theta(e)| \neq 0)$ is highly non-differentiable, which can lead to sharp oscillations and lack of convergence when using either $P_{\text{max}}$ or $P_{\text{count}}$. We also observe this experimentally (see Fig. 4 in Section V.C).

A. The Communication-aware Pruning (CaP) Algorithm

To address this, we use a much smoother version of penalty $P_{\text{count}}$, which also has additional benefits w.r.t. accuracy/sparsity tradeoff attained. In particular, our CaP algorithm solves the following optimization problem:

$$\text{Minimize: } L(\theta) + \lambda \sum_{e \in E_{net}} |\theta(e)| \cdot C(e) \quad \text{subject to } \theta_l(e) \in S_l(\alpha_l), \quad l = 1, \ldots, L, \quad (5)$$

where set $S_l$, parameterized by $\alpha_l$, enforces sparsity to layer $l$. We describe these sets in more detail below.

Intuitively, the CaP objective measures the importance of a weight by its contribution to predictive power as well as its communication cost; it then minimizes less-important weights.

1Mapping $e \to \theta(e)$ need not be 1-1: in convolutional layers, several neural connections map to the same coefficient.

2To avoid confusion, we refer to DNN nodes and edges as neurons and neural connections, respectively, and communication network nodes and edges as devices and communication links.

3If the communication network is connected, we can represent it as a clique with costs capturing end-to-end path delays, so that Property 1 holds.

4For categorical classification, we assume that labels are one-hot encoded.
while enforcing that the learnt weights are subject to the sparsity constraints. Our use of the absolute value in the penalty of (5) has the following additional advantage: For parameters θ(e) whose neurons are mapped to the same device, we have by Eq. (1) that C(e) = 0; hence, these parameters are effectively unconstrained, and can grow as determined by the loss L(θ) to attain good predictive accuracy. In contrast, parameters θ(e) for which C(e) is non-zero are forced by the penalty to have small norm, especially when C(e) is high. Thus, pruning them as the algorithm enforces sparsity constraints S_l does not have a significant effect in accuracy. Penalties P_count or P_max do not exhibit this property.

We now turn our attention to the definition of constraint sets S_l. As is typical in neural network pruning [18], [20], [33], there can be tailored to different layer types (e.g., convolution and fully-connected layers). Though these sets are non-convex in general, as is standard practice [18], [20] to solve Prob. (5) via ADMM; we describe how to do this in Sec. IV-B.

Fully-Connected (FC) Layers. For FC layers, we set:

\[ S_l = \{ \theta_l \mid \|\theta_l\|_0 \leq \alpha_l \}, \]

where \( \| \cdot \|_0 \) is the size of \( \theta_l \)'s support: i.e., the number of non-zero \( l \)-layer parameters \( \theta_l \) is at most \( \alpha_l \).

Convolution Layers. We can use a similar sparsity constraint as (6) for convolution layers. Nevertheless, practical considerations make it more reasonable to follow a so-called structured pruning [29], [34] approach for convolutional layers.

The main reason to do so is the following standard practice in fusion literature: data coming for multiple sensors (not necessary of the same dimension) is often expressed via different input channels in a CNN (see, e.g., [6], [35] and Fig. 3). As sensors are distinct, channels must be distributed across sensor devices through mapping \( \mathcal{M} \). We make the additional assumption/design choice that this separation is repeated across layers: we assume that feature maps produced at each layer output (which become channels in the higher convolutional layer) are also partitioned by \( \mathcal{M} \).

To state this formally, note that \( \theta_l \) can be considered as a 2D convolution and represented by a three-dimensional tensor \( \theta_l = [\theta_l]_{i,j,r}, i \in P_l, j \in P'_l, r \in R_l \), with each dimension \( P_l, P'_l, R_l \in \mathbb{N} \) representing the number of channels, filters, kernel size (i.e., kernel width \( \times \) kernel height), respectively. Hence, each layer can be (semantically) partitioned into blocks across layer and channel dimensions (see Fig. 3); we represent the per-channel partition via \( \mathcal{Y}^{\text{DNN}}_l = \bigcup_{j=1}^{P'_l} \mathcal{Y}^{\text{DNN}}_{l,j} \). Then, \( \mathcal{M} \) satisfies the following property:

**Property 2.** For channel \( i, j \), if \( n \in [\mathcal{Y}^{\text{DNN}}_l]_i \) and \( n' \in [\mathcal{Y}^{\text{DNN}}_l]_i \), then \( \mathcal{M}(n) = \mathcal{M}(n') \).

Given this assumption, we define set \( S_l \) as:

\[ S_l^d = \{ \theta_l \mid \sum_{i=1}^{P_l} \sum_{j=1}^{P'_l} \mathbb{1}_{[\theta_l]_{i,j,:,=0}} \leq \alpha_l \} \]

In other words, this structured constraint enforces that the number of non-zero kernels on the \( l \)-th layer does not exceed \( \alpha_l \). This is exactly tailored to maps \( \mathcal{M} \) that satisfy Property 2 and produces block structures of un-pruned/dense parameters per device, along with sparse parameters to be communicated across devices, as needed to attain good accuracy (see Fig. 3).

### B. Solving CaP via ADMM

Both (6) and (7) are combinatorial constraints; to deal with this, we follow an ADMM-based pruning strategy [18], [20]. In short, ADMM is a primal-dual algorithm designed for constrained optimization problems with decoupled objectives. Through the definition of an augmented Lagrangian, the algorithm alternates between two primal steps that can be solved efficiently and separately. The first subproblem optimizes a communication-aware objective augmented with a proximal quadratic penalty; this is an unconstrained optimization solved by classic SGD, occurring in a continuous domain. The second subproblem is solved by performing Euclidean projections \( \Pi_{\mathcal{M}_l}(\cdot) \) to the constraint sets \( S_l^d \); even though the latter are not convex, projections for sets given by Eq. (6) and Eq. (7) can be computed in polynomial time (see, e.g., [18]). Subsequent proximal optimization steps force the solution to be closer to these (sparse) projections, constructing a final sparse solution in a continuous fashion; the overall CaP framework is summarized in Algorithm [1].
Algorithm 1 CaP Framework

Input: input samples \( \{(x_i, y_i)\}_{i=1}^{N} \), a model \( f_\theta \), communication function \( c \), sparsity parameter \( \alpha_l \), learning rate \( \beta \), proximal parameters \( \{p_l\}_{l=1}^{L} \).

Output: parameter of model \( \theta \)

Initialization: initialize \( \theta \) by training on samples.

while \( \theta \) has not converged do

Sample a mini-batch of size \( m \) from input samples.

\[
\theta \leftarrow \theta - \beta \nabla \left( \mathcal{L}(\theta) + \lambda \sum_{e \in \text{comm}} |\theta(e)| \cdot C(e) + \sum_{l=1}^{L} \frac{\alpha_l}{2} \| \theta^l - \theta^l + u^l \|_F^2 \right).
\]

Projection step:

\[
\theta^l_i \leftarrow \Pi_{S^l_i} \left( \theta^l_i + u^l_i \right), \text{ for } l = 1, \ldots, L.
\]

Dual variable update step:

\[
u \leftarrow u + \theta - \theta^l.
\]
end

C. Batch Normalization

Batch normalization (BN) \[36\] normalizes features in each layer and introduces learnable parameters w.r.t. the mean and standard deviation per layer. This may introduce communication across devices; to avoid this we execute separate BNs per device, i.e., we introduce and train separate batch-normalizing parameters of neurons mapped to each device.

V. EXPERIMENTS

In our experiments, (a) we show that our proposed CaP outperforms the current state-of-the-art (SOTA) partitioning algorithms in terms of accuracy on benchmark datasets; (b) we explore tradeoffs of different communication-aware penalties w.r.t. accuracy and communication costs; and (c) we illustrate the effectiveness of CaP via two real-life case studies, Radar Detection and Multimodal Beamforming.

A. Experiment Setting for Benchmarks

We first evaluate CaP on two benchmarks, CIFAR-10 and CIFAR-100 \[37\]. We compare to two SOTA weight-based partitioning algorithms: SplitNet \[14\] and NoNN \[15\], that split the DNN across devices, allowing no cross-communication.

Experiment Details. We assume \( M \) devices and map all layers across \( M - 1 \) worker devices, except the final output layer, which is deployed on the remaining (central) device. We vary the number of worker devices (2, 4, 8). Our map \( M \) is such that consecutive neurons (along the channel dimension) are split across workers equally.

We use the ResNet-18 \[38\] architecture for CIFAR-10, and WideResNet-28-10 \[39\] for CIFAR-100. We use the same dense model for all methods. We run CaP using SGD with initial learning rate 0.01, momentum 0.9 and weight decay \( 10^{-4} \), and set ADMM and fine-tuning epochs to 300 and 100, respectively. For SplitNet and NoNN, we use code and hyperparameters as provided by their respective authors.

CaP Parameters. We assume that worker devices communicate simultaneously via 5G-NR links, and set the CaP cost per communication link \( e \) as \( c(e) = 1 \). We control \( \alpha_e \) to set the sparsity ratio, i.e., the ratio of pruned/zero parameters vs. dense DNN size. We use the same sparsity ratio for all layers, and set it to match the sparsity of competitors (i.e., full partition); this yields sparsity ratio (0.50, 0.75, 0.875) for the (2, 4, 8)-worker settings, respectively.

Evaluation Metrics. We report test accuracy (in %), and # params and inference FLOPs for both overall and largest partition; and CaP\(\{\} \) represents CaP with a sparsity ratio given by \( \cdot \), set to match the sparsity of competitors (full partition).

<table>
<thead>
<tr>
<th>Methods / Devices</th>
<th>Overall</th>
<th>Largest partition</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Params</td>
<td>FLOPs</td>
<td># Params</td>
</tr>
<tr>
<td>Dense</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NoNN</td>
<td>18.32</td>
<td>5.35</td>
<td>9.16</td>
</tr>
<tr>
<td>CaP_{50}</td>
<td>18.30</td>
<td>5.25</td>
<td>9.15</td>
</tr>
<tr>
<td>NoNN</td>
<td>9.24</td>
<td>2.78</td>
<td>2.33</td>
</tr>
<tr>
<td>CaP_{25}</td>
<td>9.19</td>
<td>2.63</td>
<td>2.30</td>
</tr>
<tr>
<td>SplitNet</td>
<td>4.12</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NoNN</td>
<td>4.72</td>
<td>1.44</td>
<td>0.59</td>
</tr>
<tr>
<td>CaP_{2.5}</td>
<td>4.64</td>
<td>1.32</td>
<td>0.58</td>
</tr>
</tbody>
</table>
B. Comparison with Partitioning Methods

Tables I and II report performance on CIFAR-10 and CIFAR-100, comparing CaP with SplitNet and NoNN. From both tables, we can clearly tell that CaP significantly outperforms both SplitNet and NoNN in terms of predictive power (measured in accuracy) while achieving similar or even better computation efficiency (measured in FLOPs). Increasing partitions (and decreasing parameters per partition), leads to more pronounced difference between CaP and competitors w.r.t. accuracy. Notably, on CIFAR-10, CaP achieves accuracy +0.40%, +1.10%, +2.09% higher than NoNN with 2, 4, and 8 devices, respectively. This is even more pronounced when moving to the (harder) dataset CIFAR-100: CaP maintains high accuracy with only 0.37% drop (with 8 partitions) from the dense model, while the accuracy achieved by SplitNet and NoNN drops by 4.97% and 4.34%, respectively. CaP successfully improves accuracy precisely by introducing cross-partition communication; we discuss the intrinsic trade-off between accuracy and communication next.

C. A Comprehensive Understanding of CaP

In Fig. 4 we compare several CaP variants on CIFAR-10 with ResNet-18, under 4 devices. We show both the trade-off between accuracy and communication latency, as well as the learned tensors on selected layers. We also compare CaP to the following variants: (a) CaP$_\text{max}$, using penalty defined in Equation (3); (b) CaP$_\text{count}$, using penalty defined in Equation (4); and (c) the dense model distributed across the 4 devices.

Impact of $\lambda$. Fig. 4(a) shows the accuracy/latency tradeoff when varying $\lambda$: decreasing $\lambda$ to 1e$-7$, increases latency but improves accuracy (94.15%), close to the dense model (94.44%); when increasing $\lambda$, the communication cost gets rapidly suppressed yet with accuracy drop. In particular, increasing $\lambda$ to infinity, which we refer to as Partition-only-Pruning (PoP), leads to disjoint partitions, thus is immune to communication but drops accuracy significantly (91.70%). Among all, setting $\lambda$=1e$-6$ attains a favorable trade-off between accuracy.
observe that, compared to CaP, reducing latency by +32% and +35%, respectively. We also observe that, to achieve similar accuracy as CaP, the smoother penalty used for CaP defined in Eq. (5), is more effective than both $P_{\text{max}}$ and $P_{\text{count}}$. Specifically, $\text{CaP}_{75}(\lambda=1e-5)$ achieves 93.36% accuracy, which is 0.84% and 1.10% higher than CaP$_{\text{max}}$ and CaP$_{\text{count}}$, while reducing latency by +32% and +35%, respectively. We also observe that, compared to CaP$_{\text{max}}$, CaP$_{\text{count}}$ achieves better performance w.r.t. both accuracy and latency reduction; this is because the max term exacerbates non-differentiability. We also observe that, to achieve similar accuracy as CaP, CaP$_{\text{count}}$ requires $1.5 \times -2 \times$ more latency. This is expected, as CaP$_{\text{count}}$ assumes cost equally across links, without considering the corresponding feature map size. This is evident in the colored tensors as well: CaP$_{\text{count}}$($\lambda=1e-5$) prunes off-block diagonal kernels uniformly across layers. On the contrary, CaP$_{75}$($\lambda=1e-6$) more heavily prunes layers nearer to inputs, thus achieving similar accuracy at lower latency.

Ablation Study/Comparison to Variants. Fig. 4(a) indicates that CaP attains a much more favorable trade-off between accuracy and communication cost than other variants. We observe that, $P_{\text{abs}}$, the smoother penalty used for CaP defined in Eq. 3, is more effective than both $P_{\text{max}}$ and $P_{\text{count}}$. Specifically, CaP$_{75}$($\lambda=1e-5$) achieves 93.36% accuracy, which is 0.84% and 1.10% higher than CaP$_{\text{max}}$ and CaP$_{\text{count}}$, while reducing latency by +32% and +35%, respectively. We also observe that, compared to CaP$_{\text{max}}$, CaP$_{\text{count}}$ achieves better performance w.r.t. both accuracy and latency reduction; this is because the max term exacerbates non-differentiability. We also observe that, to achieve similar accuracy as CaP, CaP$_{\text{count}}$ requires $1.5 \times -2 \times$ more latency. This is expected, as CaP$_{\text{count}}$ assumes cost equally across links, without considering the corresponding feature map size. This is evident in the colored tensors as well: CaP$_{\text{count}}$($\lambda=1e-5$) prunes off-block diagonal kernels uniformly across layers. On the contrary, CaP$_{75}$($\lambda=1e-6$) more heavily prunes layers nearer to inputs, thus achieving similar accuracy at lower latency.

D. Case Study: Radar Detection

The Citizen Broadband Radio Service (CBRS) band [42] is a 150 MHz band in the mid-range frequencies between 3.55 and 3.70 GHz, that is originally reserved for Federal users such as naval radars, in the US. The Federal Communication Commission (FCC) has recently opened this band to secondary users including cellular service providers, private networks, and IoT applications, as the conventional bands of these users become more and more crowded everyday. The usage of this band by secondary users is allowed provided that the activities of the primary (Federal) users are protected [43].

In order to realize if a radar activity exists in the band, a central entity called Spectrum Access System (SAS) maintains frequent messaging with Environmental Sensing Capability stations (ESCs) that have spectrum sensing abilities and are deployed near coastal regions where radar pulses are received. In our proposed CBRS spectrum sensing scheme, we assume a network of geographically distributed ESCs connected to a SAS, whose purpose is to detect the presence of a primary user (i.e., radar transmission) using ESC-sensed signals [42].

ESC Dataset Description. To evaluate our distributed sensing/computing scheme for the CBRS band, we assume a network of 5 ESCs geographically located in a coastal area (see Fig. 7). One LTE and one 5G base station are placed close to the coastal region, within fixed distances from each ESC sensor. We monitor each 10 MHz channel in the CBRS band independently and create spectrograms with 10 MHz frequency content (sampling rate) every 20 ms, at each ESC. Each snapshot of the channel has a combination of 5 MHz non-overlapping LTE and/or 5G signals in the Time Division Duplexing (TDD) mode, which is a requirement of cellular activity in the CBRS band [43]. For the radar signal, we select a burst of radar pulses of Type 1 with 0.5 $\mu$s pulse width, 20 pulses per burst, and pulse repetition rate of 1010 per seconds.

Fig. 5: A schematic of the deployment of CBRS case study using CaP framework. All distance units are in km. We only focus on the processing part of the deployment. During the deployment either of (1) Star or (2) Tree topology are used. We provide the comparative analysis between the two topologies in Table IV.

![Fig. 5](image-url)

**Fig. 6:** Examination of learned weights on a per-layer basis. We show the layer usage (i.e., percentage of non-zeros per layer, measured in %) by the left y-axis, and the percentage of communication latency contributed per layer by the right y-axis. We also use green for non-zeros within the same device, and red for non-zeros used in cross-device communications, the latter of which accounts for the communication latency.
TABLE IV: Performance of CaP on ESC dataset over both star and tree topologies. For reference purposes, we include the performance attained by (a) Dense$_C$, that sends all raw data to SAS and executes the dense model there, and (b) Dense$_D$, that distributes the dense model onto the communication network. Overall, CaP reduces the total execution time by up to 68% on both topologies with negligible AUC drop (i.e., less than 0.3%). Moreover, CaP is able to perform real-time inference: spectrograms are created every 20 ms, and CaP requires <9ms, thus can achieve real-time safely and efficiently. In contrast, Dense$_C$ and Dense$_D$ (~25ms) cannot perform inference within this deadline.

<table>
<thead>
<tr>
<th>Methods</th>
<th>$\lambda$</th>
<th>Sparsity</th>
<th>Star Topology</th>
<th>Tree Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>AUC % (↑)</td>
<td>Comp. Time (↑)</td>
</tr>
<tr>
<td>Dense$_C$</td>
<td>-</td>
<td>-</td>
<td>99.76</td>
<td>25.43</td>
</tr>
<tr>
<td>Dense$_D$</td>
<td>-</td>
<td>-</td>
<td>99.76</td>
<td>25.43</td>
</tr>
<tr>
<td>PoP</td>
<td>-</td>
<td>0.80</td>
<td>98.95</td>
<td>7.64</td>
</tr>
<tr>
<td>CaP$_{80}$</td>
<td>1</td>
<td>0.80</td>
<td>99.56</td>
<td>8.53</td>
</tr>
<tr>
<td>CaP$_{85}$</td>
<td>1</td>
<td>0.85</td>
<td>99.48</td>
<td>8.32</td>
</tr>
</tbody>
</table>

This radar type is currently used for naval radars and might or might not overlap with the cellular activity in the CBRS band. Each ESC perceives the snapshots of the radar, 5G, and LTE signals with signal strength and phase determined by the distance from these sources. Therefore, through the 5 ESCs, the SAS has access to 5 different snapshots for each 20 ms. To create the complete dataset, we simulate 500 spectrograms in MATLAB for each ESC, that sum up to 2500 total spectrograms in the complete dataset. The goal is to decide whether there is a radar pulse appearing in each sampled 20 ms of the recorded data.

Backbone Network. The 5 ESCs connect to 5 edge cloud servers using Sub-6 GHz 5G wireless channels (Frequency Range 1 for 5G-NR standard). The edge cloud servers are connected to the SAS via a 5G backhaul network using optical fiber wired connections of 10 Gbps bandwidth. We study both a (a) star and (b) tree network topology as, shown in Fig. 7. As before, we set the latency of the wireless link considering the throughput of 63.59MBps of the 5G-NR standard (see Sec. V-A), and the latency of wired transmissions to be equal to $B$/10Gbps, where $B$ is the data transmission size.

Neural Network Topology. We design a fusion network as described in Table III. It contains 5 convolution, 3 FC and 1 final output layers. Each sensor contributes 3 channels (RGB) per spectrogram, resulting in inputs of the size of (15,320,266) (15 channels in total). For the star topology, we set $c(e) = 1$, and assume all convolution and FC layers are equally partitioned across edge devices, with the final output layer computed at the SAS. For the tree topology, we set $c(e) = \#hops between devices and consider a mapping in which (a) convolution layers are partitioned across the first tree layer, (b) all FC layers are partitioned across the second tree layer, and (c) the final output layer is mapped to the SAS.

Evaluation Metrics. We report the inference performance in terms of AUC, as well as computation, communication, and total execution time. To precisely estimate the computation time (e.g., the time required to execute the largest partition per stage and the final output layer), we evaluate the dedicated architectures on a NVIDIA Jetson TX2 platform, which is an industrial-graded GPU aided computer designed for GPU computations. We construct architectures which have exactly same number of channels/weight width (e.g., $P/K$) as the largest partition of CaP, but maintain the original number of filters/weight height (e.g., $P/l$). We then cut the intermediate feature maps along channel/weight width to eliminate the dimension mismatch introduced by this construction. We also load the sparsified weights from the largest partition. We measure communication time assuming transmissions over the same (wired or wireless) link contend, using bandwidths as reported above.

Results. Table IV shows the performance of CaP over both star and tree topologies. For reference purposes, we include the performance attained by (a) Dense$_C$, that sends all raw data to SAS and executes the dense model there, (b) Dense$_D$, that distributes the dense model onto the communication network (without pruning), and (c) PoP, introduced in Section V-C that allows no cross-communication. Overall, CaP reduces the total execution time by up to 68% on both topologies with negligible AUC drop (i.e., less than 0.3%); even for the extreme PoP case that producing disjoint partitions, the AUC only drops less than 1%. In fact, CaP is able to perform real-time inference: spectrograms are created every 20 ms, and CaP requires <9ms, thus can achieve real-time safely and efficiently. In contrast, Dense$_C$ and Dense$_D$ (~25ms) cannot perform inference within this deadline.

We also observe that CaP over tree topology costs more than the star topology, especially for computation cost. The reason is that the second tree layer contains two devices only, thus each device is responsible for more computations than the setting of star topology with five devices. Surprisingly, communication latency is reduced more effectively over tree topology: comparing to Dense$_D$, CaP$_{80}(\lambda=1)$ reduces communication latency by 95.6% and 88.6% over tree and star topology, respectively.

To explore how CaP behaves on both topologies further, we examine the learned weights on a per-layer basis. Fig. 6 shows the layer usage as well as the percentage of communication cost contributed per layer. Compared to the star topology, CaP over tree topology preserves more weights on FC layers due to the two-device mapping, increasing execution time but improving AUC, as shown in Table IV. We also observe that convolution layers contribute larger costs than FC layers, thus
being pruned significantly, indicating the effectiveness of CaP.

E. Case Study: Multimodal Beamforming

**Dataset Description.** We also validate CaP on a publicly available real-world dataset\(^4\) of multimodal beamforming [44], [45] in a vehicle to infrastructure (V2I) scenario, as depicted in Fig. 7. The FLASH dataset includes synchronized sensor data from on-board Global Positioning System (GPS), a GoPro Hero 4 camera, and a Velodyne Light Detection and Ranging (LiDAR) sensors, along with the RF ground truth including the received signal strength indicator (RSSI) of all beams recorded by Talon AD7200 routers [46]. The onboard GPS records the latitude and longitude of the vehicle as it passes in front of the static base station. The camera is faced toward the base station and records RGB samples with shape (3, 90, 160). The LiDAR data is released as a quantized representation of the environment with unique indicators that depict the location of the transmitter, receiver, and obstacles [47]. The overall dataset spans four different categories of experiment setups with \(~32\)K samples.

**Experiment Details.** Following [35], we leverage three convolution-based networks serving as base models for GPS, Image, and LiDAR modalities, generating embeddings with the size of \((64, 64), (256, 64), (512, 64),\) respectively. The fusion model, described in Table III contains 4 FC and 1 final output layers and takes inputs as the concatenated embedding of each modality along the first dimension, resulting in an input size of \((832, 64)\). We consider 3 devices (one per modality) in a dense topology, assuming a 5G communication channel between them (as above).

**Results.** Table V shows the performance of CaP on two sparsity levels SR=0.67 and SR=0.75 with a series of \(\lambda\), ranging from \(1e^-4\) to \(1e^+2\). We also report Dense\(_C\), Dense\(_D\), and PoP for reference. Among all variants, CaP\(_D\)(\(\lambda=1e^+2\)) performs the best: comparing to Dense\(_C\), it reduces communication latency by 41.7\%, resulting in a total of 27\% reduction, under only negligible accuracy loss (less than 1\%).

![Networks Fusion Image, and LiDAR modalities, generating embeddings with convolution-based networks serving as base models for GPS, Image, and LiDAR modalities, generating embeddings with](https://www.rfdatafactory.com/datasets)

![Networks Fusion Image, and LiDAR modalities, generating embeddings with convolution-based networks serving as base models for GPS, Image, and LiDAR modalities, generating embeddings with](https://github.com/neu-spiral/CaP)

![Networks Fusion Image, and LiDAR modalities, generating embeddings with convolution-based networks serving as base models for GPS, Image, and LiDAR modalities, generating embeddings with](https://www.rfdatafactory.com/datasets)

![Networks Fusion Image, and LiDAR modalities, generating embeddings with convolution-based networks serving as base models for GPS, Image, and LiDAR modalities, generating embeddings with](https://github.com/neu-spiral/CaP)