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 enviromental 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.

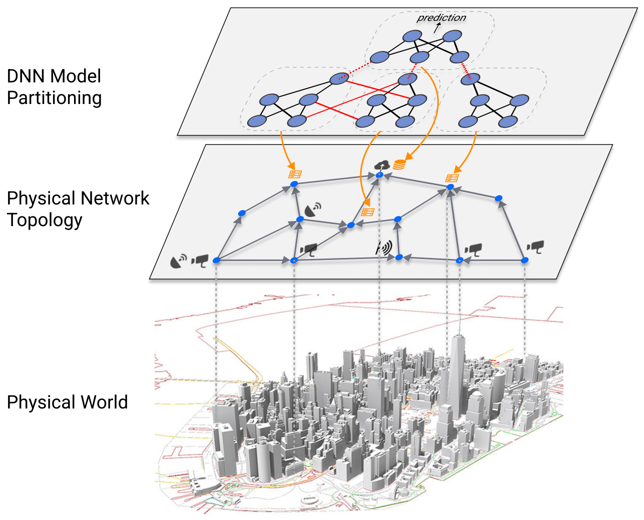


Fig. 1: Motivation of our CaP framework. We consider a smart city as a motivating scenario, involving a ubiquitous deployment of networked sensors at different locations across a large urban area. To reduce latency and accelerate computation, inference over a DNN is distributed across the network fabric. 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.

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 is partitioned into sub-networks, each of which is mapped to the physical network topology and executed on either the edge or an intermediate network device (e.g., a router). As inference computations occur in a distributed fashion, communication needs to occur between devices tasked with performing partial computations of the DNN's layers. Our goal is to *take this into account while training the neural network*. In particular, we introduce the notion of *communication-aware* DNN pruning: cognisant of the physical network topology over which the DNN is mapped, we simultaneously train-and-prune the network so that (a) neural connections are pruned in a way that overall communication costs are reduced, while (b) the resulting pruned DNN's accuracy remains as high as that of a fully dense network.

Our main contributions are as follows:

1) We propose a **C**ommunication-**a**ware **P**runing (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 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 ℓ_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).

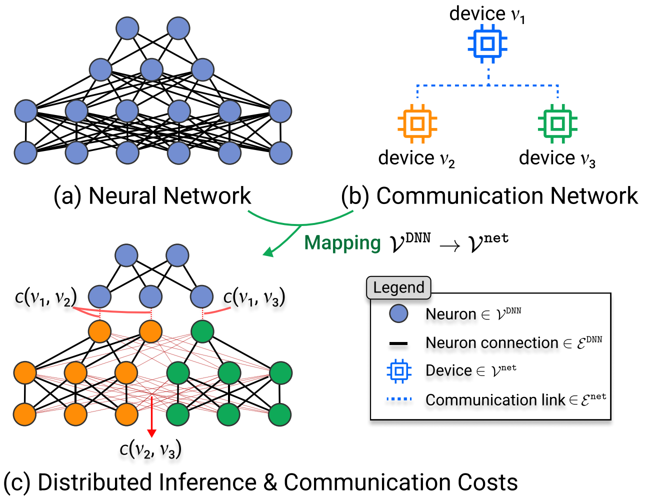

device v_1
device v_2
device v_3
(a) Neural Network
(b) Communication Network
Mapping $\mathcal{V}^{\text{DNN}} \to \mathcal{V}^{\text{net}}$
$c(v_1, v_2)$
$c(v_1, v_3)$
$c(v_2, v_3)$
Legend
Neuron $\in \mathcal{V}^{\text{DNN}}$
Neuron connection $\in \mathcal{E}^{\text{DNN}}$
Device $\in \mathcal{V}^{\text{net}}$
Communication link $\in \mathcal{E}^{\text{net}}$
(c) Distributed Inference & Communication Costs

Fig. 2: An illustration of the communication-aware pruning problem.

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 $f_{\boldsymbol{\theta}} : \mathbb{R}^{d^{\text{in}}} \to \mathbb{R}^{d^{\text{out}}}$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^M$, 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 $\mathbf{x}^k \in \mathbb{R}^{d^k}$, and the DNN operates on the concatenation of these inputs, i.e., on $\mathbf{x} = [\mathbf{x}^k]_{k=1}^K \in \mathbb{R}^{d^{\text{in}}}$, where $d^{\text{in}} = \sum_{k=1}^K d^k$.

We also represent the DNN by a directed acyclic graph $\mathcal{G}^{\texttt{DNN}}(\mathcal{V}^{\texttt{DNN}}, \mathcal{E}^{\texttt{DNN}})$, comprising *neurons* $\mathcal{V}^{\texttt{DNN}}$ and their respective *neural connections* $\mathcal{E}^{\texttt{DNN}} \subset \mathcal{V}^{\texttt{DNN}} \times \mathcal{V}^{\texttt{DNN}}$. Note that each neural connection $e \in \mathcal{E}^{\texttt{DNN}}$ is associated with a weight/parameter in $\boldsymbol{\theta}$; we denote by $\theta(e)$ the coefficient of $\boldsymbol{\theta}$ that corresponds to connection $e \in \mathcal{E}^{\texttt{DNN}}$.[1] We use subscript $l \in \{0, 1, \ldots, L\}$ to indicate layers, where $l = 0$ is the input ($\mathbf{x}$). Then, the neurons, the neural connections, and the coefficients of layer l can be represented as $\mathcal{V}_l^{\texttt{DNN}} \subset \mathcal{V}^{\texttt{DNN}}$, $\mathcal{E}_l^{\texttt{DNN}} = \mathcal{E}^{\texttt{DNN}} \cap (\mathcal{V}_{l-1}^{\texttt{DNN}} \times \mathcal{V}_l^{\texttt{DNN}})$, and $\theta_l(e) = \{\theta(e) : e \in \mathcal{E}_l^{\texttt{DNN}}\}$, 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 $\mathcal{G}^{\texttt{net}}(\mathcal{V}^{\texttt{net}}, \mathcal{E}^{\texttt{net}})$, comprising network *devices* $\mathcal{V}^{\texttt{net}}$ and *communication links* $\mathcal{E}^{\texttt{net}} \subset \mathcal{V}^{\texttt{net}} \times \mathcal{V}^{\texttt{net}}$.[2] Each communication link $e \in \mathcal{E}^{\texttt{net}}$ 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 $\mathcal{M} : \mathcal{V}^{\texttt{DNN}} \to \mathcal{V}^{\texttt{net}}$ 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,[3] we assume that direct communication between two devices that are assigned connected neurons is feasible; formally, the $\mathcal{M}$ satisfies the following property:

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

Communication Costs. To enable computation at inference time, each device stores all parameters/weights associated with neurons it needs to compute. Formally, device $v \in \mathcal{V}^{\texttt{net}}$ stores

$$\theta(e) \text{ for all } e = (n', n) \in \mathcal{E}^{\texttt{DNN}} \text{ s.t. } \mathcal{M}(n) = v.$$

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 $e = (n, n') \in \mathcal{E}^{\texttt{DNN}}$ is given by:

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

Throughout our analysis, we assume that (a) the mapping $\mathcal{M}$ is predetermined (e.g., by algorithms such as [13], [16], [32]), and (b) devices are computationally homogeneous, 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 $\boldsymbol{\theta}$ 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 $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, comprising samples with features $\mathbf{x}_i \in \mathbb{R}^{d^{\text{in}}}$ and a corresponding labels $y_i \in \mathbb{R}^{d^{\text{out}}}$.[4] Let $\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i)$ be the aggregate loss on the dataset, where ℓ is an appropriate loss (e.g., max-entropy, ℓ_2-squared, etc.).

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

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

where P is a communication-aware regularization penalty, and $\lambda > 0$ is a regularization parameter controling 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 slowest transmission, this is:

$$P_{\max}(C, \boldsymbol{\theta}) = \textstyle\sum_{l=1}^L \max_{e \in \mathcal{E}_l^{\texttt{DNN}}} \mathbb{1}_{([\boldsymbol{\theta}(e)] \neq \mathbf{0})} \cdot C(e). \tag{3}$$

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

$$P_{\text{count}}(C, \boldsymbol{\theta}) = \textstyle\sum_{e \in \mathcal{E}^{\texttt{DNN}}} \mathbb{1}_{([\boldsymbol{\theta}(e)] \neq \mathbf{0})} \cdot C(e). \tag{4}$$

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 $\mathbb{1}_{([\boldsymbol{\theta}(e)] \neq \mathbf{0})}$ is highly non-differentiable, which can lead to sharp oscillations and lack of convergence when using either $P_{\max}$ or P_{count}. We also observe this experimentally (see Figure 4 in Section V-C).

A. The Communication-aware Pruning (CaP) Algorithm

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

$$\begin{aligned} &\underset{\boldsymbol{\theta}}{\text{Minimize:}} && \mathcal{L}(\boldsymbol{\theta}) + \lambda \textstyle\sum_{e \in \mathcal{E}^{\texttt{DNN}}} |\theta(e)| \cdot C(e) \\ &\text{subject to} && \boldsymbol{\theta}_l(e) \in S_l(\alpha_l), \quad l = 1, \cdots, L, \end{aligned} \tag{5}$$

where set S_l, parameterized by α_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,

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

[2]To 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*.

[3]If 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.

[4]For categorical classification, we assume that labels are one-hot encoded.

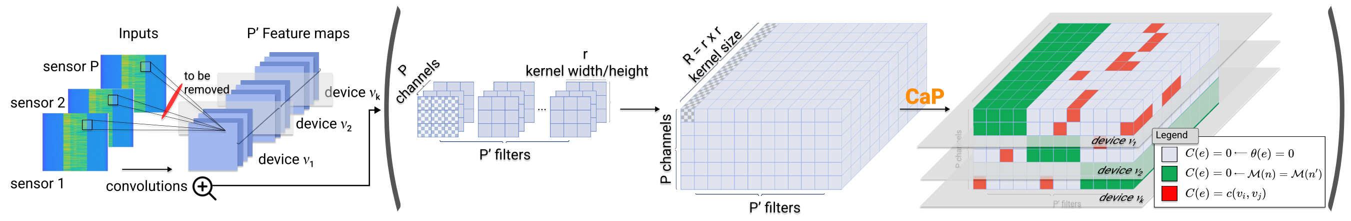


Fig. 3: An illustration of our CaP framework over convolution layers. Data coming from different sensors is treated as a different channel in the overall CNN. We assume that sensor inputs partitioned across k devices form separate input channels. Feature maps produced at each layer output (which become channels in the higher convolution layer) are partitioned thusly. As a result, each convolution layer, represented by a three-dimensional tensor $\boldsymbol{\theta}_l \in \mathbb{R}^{P_l \times P'_l \times R_l}$, can be partitioned into blocks along filter and channel dimensions. To better visualize the partitioning, we color each entry of tensor by white for zeros, green for non-zeros within the same device, and red for parameters used in cross-device communications. To perform efficient inference, we encourage kernels in off-diagonal blocks (i.e., those account for cross-device communications) to be pruned.

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 $\theta(e)$ 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 $\mathcal{L}(\boldsymbol{\theta})$ to attain good predictive accuracy. In contrast, parameters $\theta(e)$ 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:

$$S_l = \{\boldsymbol{\theta}_l \mid \|\boldsymbol{\theta}_l\|_0 \le \alpha_l\}, \tag{6}$$

where $\|\cdot\|_0$ is the size of $\boldsymbol{\theta}_l$'s support: i.e., the number of non-zero l-layer parameters $\boldsymbol{\theta}_l$ is at most α_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 $\boldsymbol{\theta}_l$ can be considered as a 2D convolution and represented by a three-dimensional tensor $\boldsymbol{\theta}_l = [\boldsymbol{\theta}_l]_{i,j,r}, i \in P_l, j \in P'_l, r \in R_l$, 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 along filter and channel dimensions (see Fig. 3); we represent the per-channel partition via $\mathcal{V}_l^{\text{DNN}} = \bigcup_{j=1}^{P} [\mathcal{V}_l^{\text{DNN}}]_j$. Then, $\mathcal{M}$ satisfies the following property:

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

Given this assumption, we define set S_l as:

$$S^l = \left\{\boldsymbol{\theta}^l \mid \left(\sum_{i=1}^{P_l} \sum_{j=1}^{P'_l} \mathbb{1}_{([\boldsymbol{\theta}^l]_{i,j,:} = \mathbf{0})}\right) \le \alpha^l\right\}, \tag{7}$$

In other words, this structured constraint enforces that the number of non-zero kernels on the l-th layer does not exceed α_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 $\Pi_{S^l}(\cdot)$ to the constraint sets S^l; 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 $f_{\boldsymbol{\theta}}$, communication function $c$, sparsity parameter $\alpha_l$, learning rate $\beta$, proximal parameters $\{\rho_l\}_{l=1}^{L}$.
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.
    SGD step:
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \nabla \Big( \mathcal{L}(\boldsymbol{\theta}) + \lambda \sum_{e \in \mathcal{E}^{\mathtt{DNN}}} |\boldsymbol{\theta}(e)| \cdot C(e) + \sum_{l=1}^{L} \frac{\rho^l}{2} \|\boldsymbol{\theta}^l - \boldsymbol{\theta}'^l + \boldsymbol{u}^l\|_F^2 \Big)$.
    Projection step:
    $\boldsymbol{\theta}'^l \leftarrow \Pi_{S^l}\left(\boldsymbol{\theta}^l + \boldsymbol{u}^l\right)$, for $l = 1, \ldots, L$.
    Dual variable update step:
    $\boldsymbol{u} \leftarrow \boldsymbol{u} + \boldsymbol{\theta} - \boldsymbol{\theta}'$
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 tradoffs 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[5] [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 $\mathcal{M}$ 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 α_ℓ 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.

[5]SplitNet is designed only for CIFAR-100 and a partition over 8 devices.

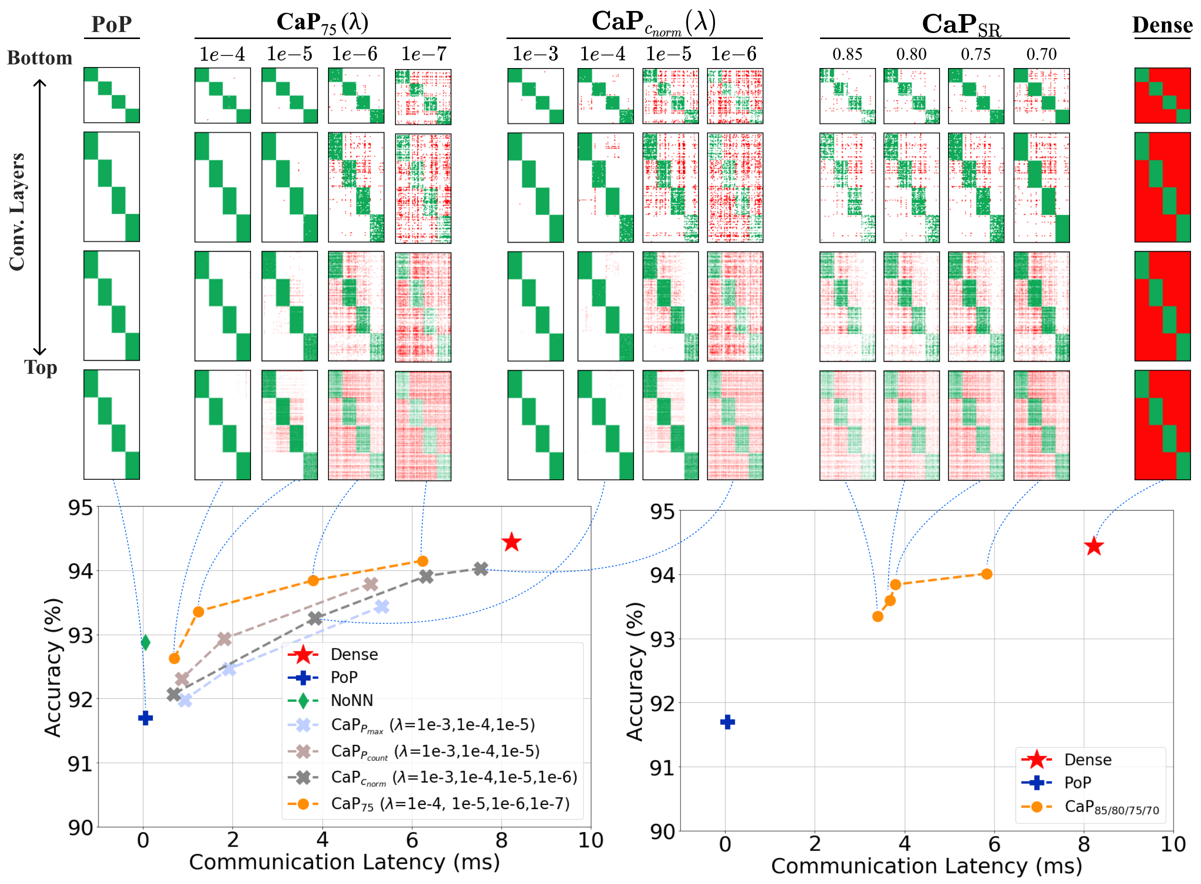
TABLE I: Performance on CIFAR-10 via ResNet-18. For all methods, we report test accuracy (in %), # params and inference FLOPs for both overall and largest partition; the latter dominates parallel computation. CaP$_{\{\cdot\}}$ represents CaP with a sparsity ratio given by $\cdot$, set to match the sparsity of competitors (full partition).

Methods	# Devices	Overall # Params $\times 10^6$ (↓)	Overall FLOPs $\times 10^9$ (↓)	Largest partition # Params $\times 10^6$ (↓)	Largest partition FLOPs $\times 10^9$ (↓)	Accuracy % (↑)
Dense	-	11.17	1.11	-	-	94.44
NoNN	2	5.71	0.60	2.89	0.30	94.10
CaP$_{50}$		5.60	0.56	2.79	0.28	**94.50**
NoNN	4	2.95	0.32	0.75	0.08	92.88
CaP$_{75}$		2.81	0.28	0.70	0.07	**93.98**
NoNN	8	1.60	0.20	0.21	0.03	90.26
CaP$_{87.5}$		1.41	0.14	0.18	0.02	**92.35**

TABLE II: Performance of CIFAR-100 via WRN-28-10. As in Table I we report test accuracy (in %), and # params and inference FLOPs for both overall and largest partition, and CaP$_{\{\cdot\}}$ represents CaP with a sparsity ratio given by $\cdot$, set to match the sparsity a full partition.

Methods	# Devices	Overall # Params $\times 10^6$ (↓)	Overall FLOPs $\times 10^9$ (↓)	Largest partition # Params $\times 10^6$ (↓)	Largest partition FLOPs $\times 10^9$ (↓)	Accuracy % (↑)
Dense	-	36.54	10.50	-	-	80.17
NoNN	2	18.32	5.35	9.16	2.65	78.24
CaP$_{50}$		18.30	5.25	9.15	2.62	**80.11**
NoNN	4	9.24	2.78	2.33	0.72	76.19
CaP$_{75}$		9.19	2.63	2.30	0.68	**79.95**
SplitNet	-	4.12	-	-	-	75.20
NoNN	8	4.72	1.44	0.59	0.20	75.83
CaP$_{87.5}$		4.64	1.32	0.58	0.16	**79.80**

Evaluation Metrics. We report the test accuracy of the final trained models. We also evaluate the number of parameters (non-zero weights), inference FLOPs, and communication latency to demonstrate the efficiency of CaP. We compute latency as follows. We measure the longest transmission required per layer assuming that only transmissions over the same point-to-point 5G-NR communication link contend; we also account for the neurons sent from devices to the cloud for final prediction, each having a distinct 5G-NR link. We assume that each neuron is represented by a 32-bit floating point, and thus estimate latency (in seconds) as $\mathrm{N} \times 4/1024^2/63.59$, where N is the number of neurons per link, and 63.59MBps is the throughput using the 5G-NR standard [40], [41]; we estimate the latter assuming 256-QAM modulation over 100 MHz bandwidth, and 132 symbols over 0.5ms slots.



(a) Ablation study and explorations over λ (b) Explorations over sparsity ratio (SR)

Fig. 4: We perform an ablation study over the (a) balancing parameter λ and (b) sparsity ratio (SR) on CIFAR-10 by ResNet-18 under a 4-device setting. In particular, we set SR=0.75 for all variants in (a) and set λ=$1e{-}6$ when exploring SR in (b). For reference purposes, we also include (a) a dense model, (b) a fully partitioned model PoP, and (c) NoNN. We show the trade-off between accuracy and communication latency. To visualize the partitioning, we further show the learned tensors on selected layers, from bottom (i.e., nearer to the inputs) to top, and color the weight tensor kernel-wise, resulting in a colored matrix with a size of $P_l' \times P_l$: each entry indicates the absolute sum over a kernel, colored by white for zeros, green for non-zeros within the same device, and red for parameters used in cross-device communications.

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$_{P_{\max}}$, using penalty defined in Equation (3); (b) CaP$_{P_{\text{count}}}$, using penalty defined in Equation (4); and (c) CaP$_{c_{\text{norm}}}$, where $c_{\text{norm}}(e) = \frac{1}{\text{N}_e}$, where N_e is the numper of parameters per link, and (c) the dense model distributed across the 4 devices.

Impact of λ. Fig. 4(a) shows the accuracy/latency tradeoff when varying λ: decreasing λ to $1e{-}7$, increases latency but improves accuracy (94.15%), close to the dense model (94.44%); when increasing λ, the communication cost gets rapidly suppressed yet with accuracy drop. In particular, increasing λ to ∞, 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 λ=$1e{-}6$ attains a favorable trade-off between accuracy

TABLE III: Dataset, architecture and experiment setting summary.

Term	Parameter	Information ESC	Information FLASH
Dataset	# Classes	2	34
	# Train samples	4,000	28,636
	# Test samples	1,000	3,287
Architecture	# Conv. layers	5	-
	# FC layers	3	4
	# Params in conv. layers	60,384	-
	# Params in FC layers	5,439,488	4,509,376
Experiments	# ADMM epochs	150	
	# Finetune epochs	100	
	Learning rate	0.001	
	Optimizer	SGD	

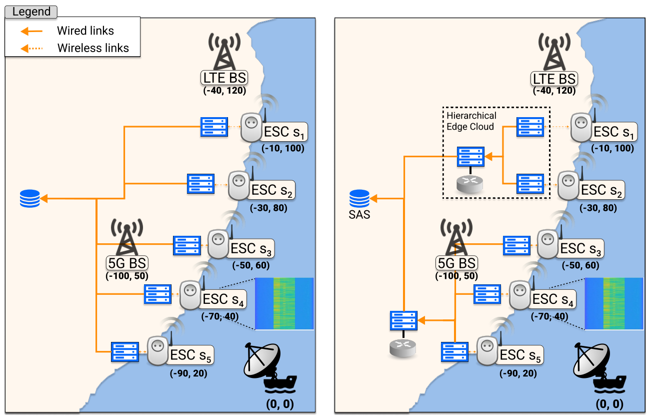
(93.98%) and communication latency (3.78ms).

Impact of Sparsity Ratio. In Fig. 4(b), we observe that for this 4-device setting, 0.75 sparsity ratio is the best choice: higher sparsity ratios (i.e., 0.80 or 0.85) lead to sparser kernels, decreasing predictive power; lower sparsity ratios (i.e., 0.70) increase communication burden, without significant improvement to predictions.

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_{abs}, the smoother penalty used for CaP defined in Eq. (5), is more effective than both $P_{\max}$ and P_{count}. Specifically, CaP$_{75}$(λ=1e−5) achieves 93.36% accuracy, which is 0.84% and 1.10% higher than CaP$_{P_{\max}}$ and CaP$_{P_{\text{count}}}$, while reducing latency by +32% and +35%, respectively. We also observe that, compared to CaP$_{P_{\max}}$, CaP$_{P_{\text{count}}}$ 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$_{c_{\text{norm}}}$ requires 1.5×–2× more latency. This is expected, as CaP$_{c_{\text{norm}}}$ assumes cost *equally across links*, without considering the corresponding feature map size. This is evident in the colored tensors as well: CaP$_{c_{\text{norm}}}$(λ=1e−5) prunes off-block diagonal kernels uniformly across layers. On the contrary, CaP$_{75}$(λ=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



(a) Star Topology (b) Tree Topology

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.

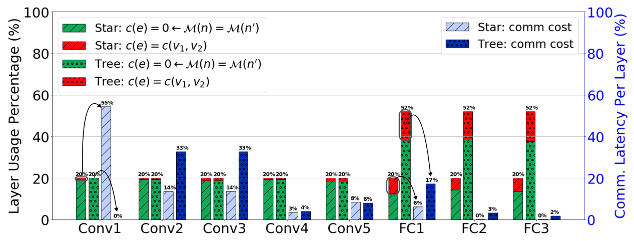


Fig. 6: Examination of learned weights on a per-layer basis. We show the layer usage (i.e., percentage of non-zeros per layer, measured in %) by the left y-axis, and the percentage of communication latency contributed per layer by the right y-axis. We also use green for non-zeros within the same device, and red for non-zeros used in cross-device communications, the latter of which accounts for the communication latency.

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 μs pulse width, 20 pulses per burst, and pulse repetition rate of 1010 per seconds.

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.

Methods	λ	Sparsity	**Star Topology**				**Tree Topology**			
			AUC % (↑)	Comp.	Time Comm. ms (↓)	Total	AUC % (↑)	Comp.	Time Comm. ms (↓)	Total
Dense$_C$	-	-	99.76	25.43	0.10	25.53	99.76	25.43	0.19	25.62
Dense$_D$	-	-	99.76	25.43	0.44	25.87	99.76	25.43	0.91	26.34
PoP	-	0.80	98.95	7.64	0.00	7.64	99.01	7.97	0.00	7.97
CaP$_{80}$	1	0.80	99.56	8.53	0.09	8.58	99.64	8.66	0.07	8.70
	1e+3		99.52	8.54	0.07	8.55	99.52	8.66	0.05	8.68
CaP$_{85}$	1	0.85	99.48	8.32	0.04	8.33	99.63	8.44	0.04	8.45
	1e+3		99.44	8.33	0.03	8.34	99.52	8.44	0.03	8.45

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 $B/10$Gbps, where B is the data transmission size.

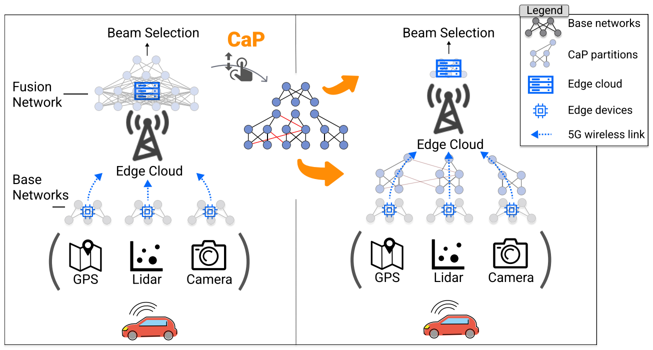
Neural Network Topology. We design a fusion network as described in Table III. It contains 5 convolution, 3 FC and 1 final output layers. Each sensor contributes 3 channels (RGB) per spectrogram, resulting in inputs with the size of $(15, 320, 266)$ (15 channels in total). For the star topology, we set $c(e) = 1$, 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_l/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$_{80}$(λ=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



(a) State-of-the-art (b) CaP Framework

Fig. 7: A schematic of the deployment of the multimodal beamforming case study using CaP framework.

TABLE V: Performance of CaP on FLASH dataset. We examine two sparsity levels SR$= 0.67$ and SR$= 0.75$ with a series of λ, ranging from $1e{-}4$ to $1e{+}2$. We also report Dense$_C$, Dense$_D$ and PoP for reference. Among all variants, CaP$_{67}$(λ=$1e+2$) 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%).

Methods	λ	Sparsity	Accuracy % (↑)	Comp.	Time Comm. ms (↓)	Total
Dense$_C$	-	-	90.31	9.20	2.59	11.79
Dense$_D$	-	-	90.31	9.20	29.92	39.92
PoP	-	0.67	83.18	7.88	0.01	7.89
CaP$_{67}$	1e−3	0.67	90.62	7.89	4.69	12.58
	1e−1		89.78	7.91	4.16	12.07
	1e+2		89.56	7.77	1.51	9.28
CaP$_{75}$	1e−3	0.75	90.09	7.69	3.40	11.09
	1e−1		88.78	7.67	1.38	9.07
	1e+2		87.38	7.71	0.55	8.26

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[6] 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 ∼32K 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 λ, ranging from $1e{-}4$ to $1e{+}2$. We also report Dense$_C$, Dense$_D$, and PoP for reference. We make the following observations. First, CaP reduces computation cost largely, by up to 16.6% comparing to Dense$_D$. Second, CaP can maintain a favorable trade-off between accuracy and communication cost by tuning λ. Among all variants, CaP$_{67}$($1e{+}2$) performs the best: comparing to Dense$_C$, it reduces communication cost by 41.7%, resulting in a total of 27% reduction, under only negligible accuracy loss (less than 1%); CaP$_{67}$(1e$-$3), by paying less than 1ms latency per round, even improves accuracy of the dense model. Third, increasing the sparsity level reduces communication costs effectively but harms accuracy more than adjusting λ. Finally, not surprisingly, different from Fig. 4, CaP prunes kernels roughly equally across layers. This is because FlashNet contains only fully-connected layers, resulting in the same (i.e., 1) size for each feature map.

VI. Conclusions

In this paper, we propose CaP, a framework to perform model communication-aware pruning 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. Experiments on benchmarks demonstrate that CaP achieves significant improvement w.r.t accuracy over SOTA competitors, by up to 4% with 8 edge devices. Experiments on real-life applications show that CaP reduces the total execution time by 27% and 68% while incurring a negligible prediction performance drop (less than 1%), for radar detection and beamforming, respectively.

Our CaP framework can be extended in several ways. First, CaP can be used as a basis for scaling inference to a much larger number of sensors. Second, exploring more complicated physical network structures and types of connections, investigating robustness of CaP against a dynamic network, as well applying CaP to more real-life applications are interesting future directions. Finally, an important additional direction to explore is the joint optimization of both partitioning/mapping neurons to devices and pruning, combining CaP with existing partitioning methods such as [13], [16], [32]. Our code is publicly available. [7]

Acknowledgment

The authors gratefully acknowledge the funding from the US National Science Foundation (grants CCF-1937500 and CNS-2112471).

[6]Retrieved from https://www.rfdatafactory.com/datasets.

[7]https://github.com/neu-spiral/CaP

References

[1] A. Gohar and G. Nencioni, "The role of 5g technologies in a smart city: The case for intelligent transportation system," *Sustainability*, vol. 13, no. 9, 2021.

[2] S. K. Rao and R. Prasad, "Impact of 5g technologies on smart city implementation," *Wireless Personal Communications*, vol. 100, pp. 161–176, 2018.

[3] K. Kim, C. Lee, D. Pae, and M. Lim, "Sensor fusion for vehicle tracking with camera and radar sensor," in *ICCAS*, 2017, pp. 1075–1077.

[4] H. Jha, V. Lodhi, and D. Chakravarty, "Object detection and identification using vision and radar data fusion system for ground-based navigation," in *SPIN*, 2019, pp. 590–593.

[5] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Gläser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE T-ITS*, vol. 22, no. 3, pp. 1341–1360, 2021.

[6] D. Roy, Y. Li, T. Jian, P. Tian, K. Roy Chowdhury, and S. Ioannidis, "Multi-modality sensing and data fusion for multi-vehicle detection," *IEEE Transactions on Multimedia*, 2022.

[7] T. Jian, Y. Gong, Z. Zhan, R. Shi, N. Soltani, Z. Wang, J. G. Dy, K. R. Chowdhury, Y. Wang, and S. Ioannidis, "Radio frequency fingerprinting on the edge," *IEEE Transactions on Mobile Computing*, 2021.

[8] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE IoT Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[9] S. Singh, R. Sulthana, T. Shewale, V. Chamola, A. Benslimane, and B. Sikdar, "Machine-learning-assisted security and privacy provisioning for edge computing: A survey," *IEEE IoT Journal*, vol. 9, no. 1, pp. 236–260, 2022.

[10] J. Mao, X. Chen, K. W. Nixon, C. D. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *DATE*, 2017, pp. 1396–1401.

[11] J. Mao, Z. Yang, W. Wen, C. Wu, L. Song, K. W. Nixon, X. Chen, H. Li, and Y. Chen, "Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns," in *ICCAD*, 2017, pp. 751–756.

[12] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE T-CADICS*, vol. 37, no. 11, pp. 2348–2359, 2018.

[13] R. Stahl, A. Hoffman, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Deeperthings: Fully distributed CNN inference on resource-constrained edge devices," *IJPR*, vol. 49, no. 4, pp. 600–624, 2021.

[14] J. Kim, Y. Park, G. Kim, and S. J. Hwang, "Splitnet: Learning to semantically split deep networks for parameter reduction and model parallelization," in *ICML*, D. Precup and Y. W. Teh, Eds., vol. 70, 2017, pp. 1866–1874.

[15] K. Bhardwaj, C. Lin, A. L. Sartor, and R. Marculescu, "Memory- and communication-aware model compression for distributed deep learning inference on iot," *ACM T-ECS*, vol. 18, no. 5, pp. 1–22, 2019.

[16] A. Abdi, S. Rashidi, F. Fekri, and T. Krishna, "Restructuring, pruning, and adjustment of deep models for parallel distributed inference," *arXiv preprint arXiv:2008.08289*, 2020.

[17] A. Ozerov, A. Lambert, and S. K. Kumaraswamy, "Paradis: Parallelly distributable slimmable neural networks," *arXiv preprint arXiv:2110.02724*, 2021.

[18] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *ECCV*, 2018, pp. 184–199.

[19] T. Li, B. Wu, Y. Yang, Y. Fan, Y. Zhang, and W. Liu, "Compressing convolutional neural networks via factorized convolutional filters," in *CVPR*, 2019, pp. 3977–3986.

[20] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *ASPLOS*, 2019.

[21] M. Maheswaran and H. Siegel, "A dynamic matching and scheduling algorithm for heterogeneous computing systems," in *HCW*, 1998.

[22] Y. Xu, K. Li, L. He, and T. K. Truong, "A dag scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization," *Journal of Parallel and Distributed Computing*, vol. 73, no. 9, pp. 1306–1322, 2013.

[23] E. Ilavarasan and P. Thambidurai, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environments," *Journal of Computer sciences*, vol. 3, no. 2, pp. 94–103, 2007.

[24] W. He, S. Guo, S. Guo, X. Qiu, and F. Qi, "Joint dnn partition deployment and resource allocation for delay-sensitive deep learning inference in iot," *IEEE IoT Journal*, vol. 7, no. 10, pp. 9241–9254, 2020.

[25] A. Parthasarathy and B. Krishnamachari, "DEFER: distributed edge inference for deep neural networks," in *COMSNETS*, 2022, pp. 749–753.

[26] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li, "Coordinating filters for faster deep neural networks," in *ICCV*, 2017, pp. 658–666.

[27] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[28] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," in *Advances in neural information processing systems*, 2017, pp. 742–751.

[29] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017, pp. 1389–1397.

[30] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient ConvNets," in *ICLR*, 2017.

[31] T. Zhang, K. Zhang, S. Ye, J. Li, J. Tang, W. Wen, X. Lin, M. Fardad, and Y. Wang, "Adam-admm: A unified, systematic framework of structured weight pruning for dnns," *arXiv preprint arXiv:1807.11091*, 2018.

[32] X. Hou, Y. Guan, T. Han, and N. Zhang, "Distredge: Speeding up convolutional neural network inference on distributed edge devices," in *2022 IEEE IPDPS*, 2022, pp. 1097–1107.

[33] S. Ye, T. Zhang, K. Zhang, J. Li, K. Xu, Y. Yang, F. Yu, J. Tang, M. Fardad, S. Liu, X. Chen, X. Lin, and Y. Wang, "Progressive weight pruning of deep neural networks using admm," *arXiv preprint arXiv:1810.07378*, 2018.

[34] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "Autoslim: An automatic dnn structured pruning framework for ultra-high compression rates," *arXiv preprint arXiv:1907.03141*, 2019.

[35] B. Salehi, J. Gu, D. Roy, and K. Chowdhury, "Flash: Federated learning for automated selection of high-band mmwave sectors," in *IEEE INFOCOM*, 2022, pp. 1719–1728.

[36] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, vol. 37, 2015, pp. 448–456.

[37] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[39] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *MBVC*, 2016.

[40] N. Varsier, L.-A. Dufrène, M. Dumay, Q. Lampin, and J. Schwoerer, "A 5G new radio for balanced and mixed iot use cases: Challenges and key enablers in fr1 band," *IEEE Communications Magazine*, vol. 59, no. 4, pp. 82–87, 2021.

[41] S.-Y. Lien, S.-L. Shieh, Y. Huang, B. Su, Y.-L. Hsu, and H.-Y. Wei, "5G new radio: Waveform, frame structure, multiple access, and initial access," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 64–71, 2017.

[42] "FCC Releases Rules for Innovative Spectrum Sharing in 3.5 GHz Band," https://docs.fcc.gov/public/attachments/FCC-15-47A1.pdf, accessed: May 2022.

[43] "Requirements for Commercial Operation in the U.S. 3550-3700 MHz Citizens Broadband Radio Service Band," https://winnf.memberclicks.net/assets/CBRS/WINNF-TS-0112.pdf, accessed: May 2022.

[44] N. González-Prelcic, A. Ali, V. Va, and R. W. Heath, "Millimeter-Wave Communication with Out-of-Band Information," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 140–146, 2017.

[45] B. Salehihikouei, G. Reus-Muns, D. Roy, Z. Wang, T. Jian, J. Dy, S. Ioannidis, and K. Chowdhury, "Deep learning on multimodal sensor data at the wireless edge for vehicular network," *IEEE Transactions on Vehicular Technology*, 2022.

[46] D. Steinmetzer, D. Wegemer, M. Schulz, J. Widmer, and M. Hollick, "Compressive Millimeter-Wave Sector Selection in Off-the-Shelf IEEE 802.11ad Devices," *CoNEXT*, 2017.

[47] A. Klautau, N. González-Prelcic, and R. W. Heath, "LIDAR Data for Deep Learning-Based mmWave Beam-Selection," *IEEE Wireless Communications Letters*, vol. 8, no. 3, pp. 909–912, 2019.