# A distributed neural network architecture for dynamic sensor selection with application to bandwidth-constrained body-sensor networks

Thomas Strypsteen and Alexander Bertrand, *Senior Member, IEEE*

***Abstract*— We propose a dynamic sensor selection approach for deep neural networks (DNNs), which is able to derive an optimal sensor subset selection for each specific input sample instead of a fixed selection for the entire dataset. This dynamic selection is jointly learned with the task model in an end-to-end way, using the Gumbel-Softmax trick to allow the discrete decisions to be learned through standard backpropagation. We then show how we can use this dynamic selection to increase the lifetime of a wireless sensor network (WSN) by imposing constraints on how often each node is allowed to transmit. We further improve performance by including a dynamic spatial filter that makes the task-DNN more robust against the fact that it now needs to be able to handle a multitude of possible node subsets. Finally, we explain how the selection of the optimal channels can be distributed across the different nodes in a WSN. We validate this method on a use case in the context of body-sensor networks, where we use real electroencephalography (EEG) sensor data to emulate an EEG sensor network for motor execution decoding. For this use case, we demonstrate that the distributed algorithm -with only a small amount of cooperation between the nodes- achieves a performance close to the upper bound defined by a fully centralized dynamic selection (maximum absolute decrease of 4% in accuracy). Furthermore, we observe that our dynamic sensor selection framework can achieve large reductions in transmission energy with a limited cost to the task accuracy, validating it as a practical tool for increasing the lifetime of body-sensor networks.**

***Index Terms*— Distributed deep neural networks, EEG channel selection, Sensor Selection, Wireless sensor networks**

## I. Introduction

Wearable, physiological sensors are increasingly being used for ambulant health monitoring, thanks to technological advances like miniaturization of microprocessors and energy-efficient batteries. In many applications though, multiple modalities or multiple channels of one data type will have to be measured on different devices on different locations on the body. This has led to the creation of body-sensor networks (BSNs), where the sensor nodes communicate over a wireless network to share their data and solve a given task together [1]. One example of such a BSN is a neuro-sensor network that monitors the brain with with electroencephalography (EEG) sensors, a so-called Wireless EEG Sensor Network (WESN) [2], [3]. EEG is a widely used, noninvasive way to record time series of electrical brain activity. These contain useful information for a variety of tasks such as epileptic seizure detection [4], sleep stage analysis [5] and brain-computer interfacing (BCI) [6]. However, EEG typically requires measurements at multiple locations on the scalp - referred to as 'EEG channels' - for two reasons. Firstly, the information of interest for a specific task may be concentrated in different area's of the brain. Secondly, EEG signals typically suffer from an extremely low signal-to-noise ratio (SNR), which needs to be improved by leveraging the spatio-temporal correlations between different EEG channels with multi-channel signal processing techniques [2]. These channels are traditionally jointly measured by a bulky, cumbersome

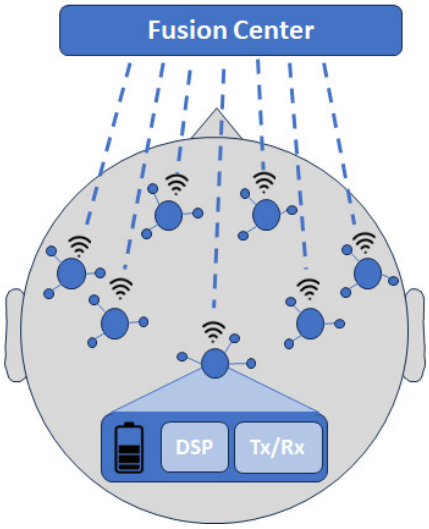



Fig. 1: Conceptual diagram of a wireless EEG sensor network (WESN). Each node measures local EEG, consisting of one or more EEG channels. In addition, each node can perform some local processing, before transmitting and aggregating all node data in a fusion center for joint inference. Each node is powered by its own local battery.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 802895). The authors also acknowledge the financial support of the FWO (Research Foundation Flanders) for project G.0A49.18N, and the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

T. Strypsteen and A. Bertrand are with KU Leuven, Department of Electrical Engineering (ESAT), STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics and with Leuven.AI - KU Leuven institute for AI, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium (e-mail: thomas.strypsteen@kuleuven.be, alexander.bertrand@kuleuven.be).

EEG cap. In contrast, a WESN employs multiple lightweight mini-EEG sensor devices to locally record one or a few EEG channels from their respective scalp areas, pre-process the data, and wirelessly transmit it to other nodes or a central fusion center for joint inference. This concept is schematically illustrated in Figure 1. The absence of wires between the different nodes reduces electromagnetic interference and wire artifacts, which are a notorious disturbance in EEG recordings, while at the same time resulting in a discreet and flexible deployment [2].

Ensuring maximal battery lifetime is a crucial consideration in the design of these WESNs or other types of BSNs [7]. The energy bottleneck will typically be found in the wireless transmission of the data between the sensors and/or a fusion center [2]. This not only motivates energy-efficient hardware design, but also a shift in the algorithmic design of the models running on these sensor nodes. Instead of optimizing the model only for accuracy, the amount of data that needs to be transmitted when the model is used in the context of a BSN becomes an important design factor.

In this paper, we focus on reducing this data transmission by teaching the nodes a policy where they only transmit their data when the contribution of this specific node towards the inference task would be very informative for the current input sample, while upholding a given bandwidth constraint. Such a constraint could be, e.g., that each node can on average only transmit at most 50% of its collected sensor data. To this end, we propose a *dynamic channel selection* methodology. For each block of collected samples across the nodes of the BSN, a distributed dynamic channel selector computes an input-dependent, optimal subset of channels, represented by a binary mask across the channels, as illustrated in Fig. 2. Inference is then performed on the masked input by a deep neural network (DNN) at a fusion center which collects the data transmitted by the sensors. The selector and the inference model are trained jointly in an end-to-end manner, with the discrete parameters involved in the selection process being made trainable through the Gumbel-Softmax trick [8], [9]. How often each channel is selected is limited by a per-channel sparsity loss on the computed masks.

The application of this dynamic channel selection in a sensor network however, comes with two design constraints that is absent in traditional dynamic feature selection paradigms. Firstly, since the lifetime of the BSN will ultimately be determined by the *critical node*, i.e., the node with the highest transmission load, it is crucial that the the dynamic selection is *balanced*, meaning that on average, all sensor nodes are required to transmit their data equally as often. Secondly, to avoid the high transmission costs associated with centralizing all the sensor data, the selection algorithms must operate in a distributed way.

In addition, the usage of this dynamic channel selection means that the inference model will be presented with different channel subsets for different inputs, as if channels were randomly missing. We show that applying dynamic spatial filtering (DSF) [10] to the masked input to re-weight the channels helps the inference model become more robust against the missing of channels and improves performance.

To validate our proposed architecture, we focus on a specific use case in the area of brain-computer interfaces, where a WESN needs to solve a motor execution decoding task. We note that, while our evaluation use case is focused on brain signals, our proposed methodology is generic and can be applied to other kinds of BSNs or more broadly, to any kind of wireless sensor network (WSN). Our Pytorch [11] implementation is available at Github[1].

The main contributions of this paper are:

- We propose an end-to-end learnable dynamic sensor or channel selection method that selects, for each window of a multi-channel input, an optimal subset of channels to use for inference, given a certain selection budget. This dynamic selection is learned jointly with the task DNN model and the Gumbel-Softmax trick is used to enable backpropagation for the discrete decisions involved.
- We demonstrate how this methodology can be used to reduce the transmission load in a wireless sensor network, thus increasing its battery lifetime. We do this by moving from centralized to distributed channel selection and enforcing per-node constraints to ensure a proper balancing of the transmission load. In addition, we present a use case where the method can improve the robustness of the classifier to noise bursts.

The paper is organized as follows. In section II we go over previous work in static and dynamic feature selection. Section III formally presents our problem statement and dynamic channel selection methodology. In section IV we provide an overview of the used dataset and how it was used to emulate a WESN environment and provide more details on the used model architecture and training strategy for this specific experiment. Our experimental results are then presented in section V and we end with some conclusions in section VI.

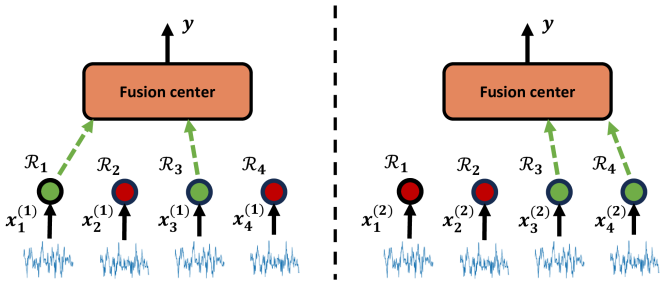



Fig. 2: Schematic illustration of dynamic sensor selection in a WSN. Sensor data is measured at nodes $m$ and transmitted to a fusion center for joint inference at a rate $\mathcal{R}_m$. For each input window $X^{(i)}$, a subset of the available nodes is dynamically selected to employ for inference, reducing the average amount of data to be transmitted.

[1]https://github.com/AlexanderBertrandLab/Dynamic-Channel-Selection

**Note on terminology:** Throughout this paper, we will always use the term 'channel selection' to refer to a selection of channels from a multi-channel input signal. Sensor selection or node selection could be viewed as a special case of channel selection. In the case of single-channel sensors, sensor or channel selection refer to the same thing. However, in the case of multi-channel sensors, sensor selection refers to the problem of selecting pre-defined *groups* of channels rather than individual channels, where each group corresponds to a sensor. For the sake of an easy exposition, but without loss of generality, we will assume single-channel sensors throughout this paper. Sometimes, we will refer to sensors as 'nodes' for consistency in terminology with the WSN literature.

## II. Related work

### A. Static feature selection

The goal of feature selection is to find an optimal subset of an available set of features that maximizes the performance of a classification or regression model on a given task. A host of literature exists that solves this problem in a *static* way, i.e., the optimal subset is determined for a certain dataset as a whole and the same selection is then applied to all input samples. Filter-based approaches rank the available features by a criterion like mutual information (MI) with the target labels and select the $K$ highest scoring features [12]. Wrapper-based approaches use methods like greedy backward selection to efficiently explore the space of possible feature subsets, train the model on these candidate subsets and finally select the one that performs the best [13]. Embedded approaches jointly learn the subset and the task model in an end-to-end way, by performing $L_1$ regularization on the input weights [14] or learning the discrete parameters of the feature selection using continuous relaxations [15], [16]. In this paper, we employ this approach of continuous relaxations to perform *dynamic* channel selection instead.

### B. Dynamic feature selection

In *dynamic*, or *instance-wise* feature selection, the aim is to find an optimal subset of features for each individual input sample. One area where this approach has been highly relevant is field of explainable machine learning, where the goal is to indicate which features contributed most to the model output. For instance, L2X [17] trains an explainer model that maximizes the mutual information between the feature subset of size $K$ of a given sample and the class distribution yielded by a trained task model. This line of work however, is mainly interested in finding the most relevant features for an already trained model, not in optimizing the performance of the model on reduced feature sets.

Another relevant area is the field of active feature acquisition [18], [19]. In this setting, obtaining features is associated with a certain cost. The goal is then to obtain maximal model performance with a minimal amount of features, without being able access all the features of a given input sample from the start. This typically results in an iterative procedure where, based on the current feature subset, the optimal feature to extend the set with is estimated, until sufficient confidence in the model prediction is reached or the budget is saturated. In our WSN setting in contrast, we do have access to all the features to perform subset selection, but we are not allowed to centralize all of them by transmitting them over the wireless link. In addition, a balanced selection across the features is not really a concern for traditional feature active feature acquisition, as there is no real disadvantage to a certain feature being present in every dynamic selection instance. In the WSN context though, a single node having to continuously transmit its data would mean the lifetime of the WSN has not been improved at all.

The most similar approach to ours in terms of methods is taken by Verelst et al. in the field of computer vision [20]. The aim of their work is to decrease the computation time and energy of a CNN by learning input-dependent binary masks that are applied to the feature maps of an image at each layer. That layer then only performs convolutions on the pixels that are not masked out. A sparsity loss on these masks then forces the network to adhere to a certain computational budget, with backpropagation for the discrete masks being enabled through the Gumbel-Softmax trick. We will employ a similar strategy to learn binary masks for our dynamic selection, albeit in a distributed architecture and with the goal of reducing the data transmission over the wireless links, as detailed in the next section.

## III. Proposed Method

In this section, we describe our dynamic channel selection methodology. Without loss of generality, we assume a classification task, although all methods can be easily extended to a regression task. For the sake of an easy exposition, we will initially assume a centralized architecture where all the channels are available to make a decision on the selection. In a WSN context, this implies that the channel selection is performed *after* transmitting all the channels to the fusion center, in which case there are no bandwidth savings. Nevertheless, this setting is still relevant to make the network robust against non-stationary noise influences and/or to reduce the computational complexity at the fusion center. Later on, in Section III-E, we will explain how the channel selection can be performed at the level of the sensor nodes, such that non-selected channels do not have to be transmitted at all.

### A. Problem statement

Let $\mathcal{D} = \{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \ldots, (X^{(N)}, y^{(N)})\}$ be a dataset of $N$ samples of a multi-channel signal $X^{(i)}$ with class labels $y^{(i)}$. Each $X^{(i)} \in \mathbb{R}^{M \times L}$ contains $M$ channels and a window of $L$ consecutive time steps. We are also given a DNN model $f_\theta$ that is used to perform inference on these samples. Our goal is to learn a dynamic selection function that, for each separate input sample, determines this sample's optimal subset of channels to be presented to the inference model, while adhering to certain budget constraints on the amount of channels we are allowed to use on average. This selection of channels is based on a score vector $\boldsymbol{\alpha} \in \mathbb{R}^M$

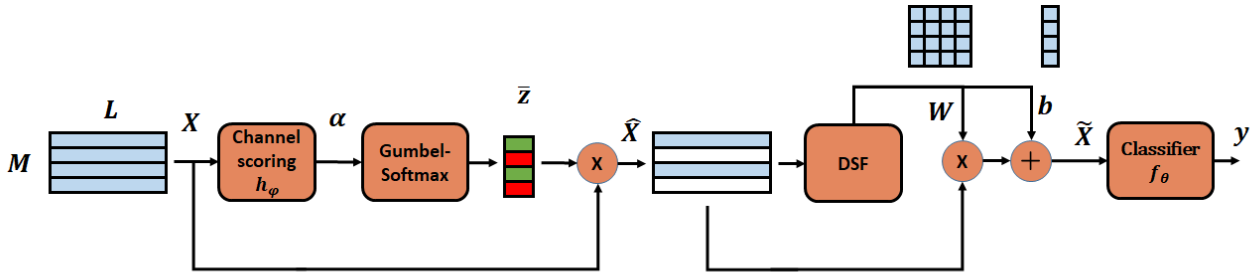



Fig. 3: Overview of the dynamic channel selection. An $L$-sample window of an M-channel signal is passed to a channel scoring module, yielding unnormalized channel scores $\boldsymbol{\alpha}$. These are then converted in discrete selections by the Gumbel-Softmax module and applied to the input as a binary mask $\bar{\mathbf{z}}$, dropping a number of channels. This masked input $\hat{X}$ is then fed to the DSF module which re-weights the received channels with an attention mechanism, computing a weight matrix $W$ and bias $b$ that are multiplied with and added to the masked input: $\tilde{X} = W\hat{X} + b$ [10]. Finally, the original classifier is then applied to the resulting signal to obtain a prediction $y$. The entire pipeline is jointly learned in an end-to-end manner.

that is computed by a channel scoring function $h_\varphi(X)$ for each input $X$. To go from a continuous score to a discrete selection in a way that still allows for end-to-end learning through backpropagation, we make use of a Gumbel-Softmax module $G$, which converts the score vector $\boldsymbol{\alpha}$ to a binary mask $\bar{\mathbf{z}} \in \{0,1\}^M$ to be applied to the input. Formally, this means learning parameters $\theta$ of the task model $f_\theta$ and the parameters $\varphi$ of a selection model $s_\varphi = G \circ h_\varphi : \mathbb{R}^{M\times L} \mapsto \{0,1\}^{M\times 1}; X \mapsto \bar{\mathbf{z}}$ such that

$$\begin{aligned}\varphi^*, \theta^* =& \underset{\varphi,\theta}{\text{argmin}}\ \mathcal{L}_{CE}(f_\theta(X \odot \bar{\mathbf{z}}\mathbf{1}_L^\top), y) + \lambda\mathcal{L}_S(\bar{\mathbf{z}}) \\ =& \underset{\varphi,\theta}{\text{argmin}}\ \mathcal{L}_{CE}(f_\theta(X \odot s_\varphi(X)\mathbf{1}_L^\top), y) + \lambda\mathcal{L}_S(s_\varphi(X))\end{aligned} \tag{1}$$

with $\mathcal{L}_{CE}(p, y)$ the cross-entropy loss between the predicted label $p$ and the ground truth $y$, $\odot$ an element-wise product, $\mathbf{1}_L^\top$ the row vector of dimension $L$ containing only ones, $\mathcal{L}_S$ a cost function that enforces sparsity in the learned masks and $\lambda$ a hyperparameter to balance the two losses. A schematic overview of our method is presented in Fig. 3. We will now delve deeper into the design of each of the modules involved.

### B. Learning discrete decisions with Gumbel-Softmax

To enable the network to learn discrete decisions while still keeping the entire network end-to-end learnable we make use of the Gumbel-Softmax trick [8], [9]. Take a discrete random variable, drawn from a categorical distribution with $K$ classes and class probabilities $\pi_1, ...\pi_K$, represented as a one-hot vector $\bar{\mathbf{y}} \in \{0,1\}^K$, with the index of the one indicating the class $\bar{\mathbf{y}}$ belongs to. Discrete samples from this distribution can then be drawn with the Gumbel-Max trick [8]:

$$\bar{\mathbf{y}} = \text{one_hot}(\underset{k}{\text{argmax}}\ (\log \pi_k + g_k)) \tag{2}$$

with $g_k$ independent and identically distributed (i.i.d.) samples from the Gumbel distribution [21] and one_hot$(i)$ the operator that generates a one-hot $K \times 1$ vector where the one is placed at position $i$. The Gumbel-Softmax is then a continuous, differentiable relaxation of this discrete sampling procedure, approximating the discrete one-hot vectors $\bar{\mathbf{y}}$ with continuous vectors $\mathbf{y}$ whose elements sum to one instead by replacing the argmax with a softmax. For the k-th element $y_k$, this results in [8]:

$$y_k = \frac{\exp((\log\ \pi_k + g_k)/\tau)}{\sum_{j=1}^K \exp((\log\ \pi_j + g_j)/\tau)} \tag{3}$$

with $\tau$ the temperature of this continuous relaxation. Lowering the temperature causes the softmax to more closely resemble an argmax, thus causing the continuous $\mathbf{y}$ to be a closer approximation of the discrete $\bar{\mathbf{y}}$. It will however, also cause the relaxation to become less smooth and increase the variance of the gradients.

Our goal is to model a learnable, binary random variable $\bar{z}_m$ for each channel $m$, which is 1 when the channel is selected and 0 otherwise. In the case of such a binary random variable $\bar{z}_m$, with $P(\bar{z}_m = 1) = \pi_1$, it can be shown [20] that by setting $K = 2$ in Eq. 3, the Gumbel-Softmax trick can be simplified to

$$\begin{aligned} y_1 &= \sigma\left(\frac{\log \pi_1 + g_1 - g_2}{\tau}\right) \\ y_2 &= 1 - y_1 \end{aligned} \tag{4}$$

with $\sigma(\cdot)$ the sigmoid function. A continuous relaxation $z_m$ of the binary random variable $\bar{z}_m$ can then be obtained by taking $z_m = y_1$ We can use this binary Gumbel-Softmax trick to transform unnormalized, learnable channel scores $\boldsymbol{\alpha} \in \mathbb{R}^M$ yielded by a network $h_\varphi(X)$ into continuous, differentiable approximations $\mathbf{z} = [z_1, ..., z_M]^\top$ of the discrete $\bar{\mathbf{z}} \in \{0,1\}^M$. There are a number of ways this continuous relaxation can be used to obtain approximating gradients for the discrete $\bar{\mathbf{z}}$, but we will follow the Straight-Through estimator approach [8], [20]. This means that we will sample discrete decisions from our binary distribution in the forward pass [20]:

$$\begin{aligned}\bar{z}_m &= \left\lfloor \sigma\left(\frac{\alpha_m + g_1 - g_2}{\tau}\right)\right\rceil \\ &= \begin{cases} 1, & \text{if } z_m = \sigma\left(\frac{\alpha_m + g_1 - g_2}{\tau}\right) > 0.5 \\ 0, & \text{otherwise} \end{cases}\end{aligned} \tag{5}$$

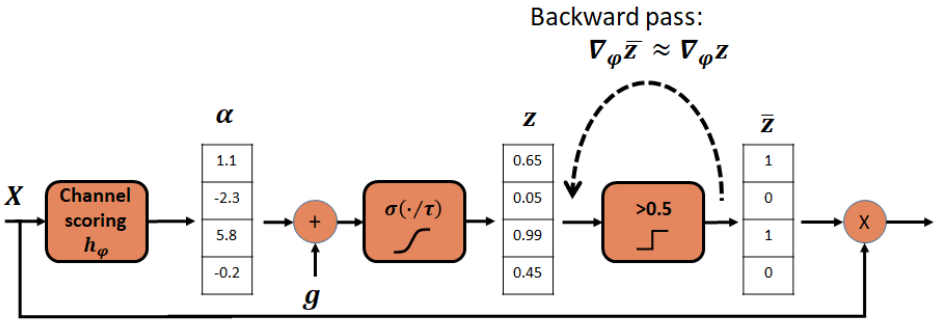



Fig. 4: Illustration of the Gumbel-Softmax trick. During training, hard decisions are sampled by perturbing the channel scores $\boldsymbol{\alpha}$ with Gumbel noise, passing this through a sigmoid to obtain soft probabilities and applying a thresholding operator. Backpropagation through the thresholding operation is enabled by using a Straigth-Through estimator, treating the threshold in the backward pass as an identity function, i.e. $\frac{\partial \bar{\mathbf{z}}}{\partial \mathbf{z}} \approx 1$ and thus $\nabla_\varphi \bar{\mathbf{z}} \approx \nabla_\varphi \mathbf{z}$

with $\lfloor \cdot \rceil$ the rounding operator, resulting in a binary distribution where $P(\bar{z}_m = 1) = \sigma(\alpha_m)$ (replacing $\pi_1$ in Eq. 4). To enable backpropagation through the discrete rounding operator, we use gradients from the continuous relaxation in the backward pass, which implies the approximation

$$\nabla_\varphi \bar{\mathbf{z}} \approx \nabla_\varphi \mathbf{z}. \tag{6}$$

This scheme allows for hard decisions to be used during training and learned through end-to-end backpropagation. This process is schematically illustrated in Fig. 4. At inference time, Gumbel noise is no longer added to the score vector, resulting in the network no longer sampling from binary distributions, but behaving in a deterministic manner instead, i.e. $\bar{z}_m = 1$ if $\sigma(\alpha_m) > 0.5$.

### C. Enforcing sparsity

We assume that each channel is measured on a different node of a wireless sensor network, whose nodes are able to communicate with each other over bandwidth-constrained links. To reduce the communication load of these nodes, we now impose constraints on how often these nodes are allowed to transmit their data., i.e., yield a 1 in their computed binary mask. Let the relative rate $\mathcal{R}_m$ be the percentage of input samples for which the binary transmission decision for node $m$ is set to 1, i.e.,

$$\mathcal{R}_m = \mathbb{E}_X[\bar{z}_m]. \tag{7}$$

We consider two types of constraints, corresponding to an *unbalanced* and a *balanced* distribution of the rate across the nodes. If we do not consider balancing the load, we simply expect the average rate to be below a certain target $T \in [0, 1]$:

$$\mathcal{R}_{avg} = \frac{1}{M} \sum_{m=1}^{M} \mathcal{R}_m \leq T. \tag{8}$$

In a scenario where we require a balanced load however, the rate of every individual node must be below the target $T$ instead of simply the average. In other words, the rate of the critical node, i.e., the node with the highest transmission load, must be lower than $T \in [0, 1]$:

$$\mathcal{R}_{max} = \max_m \mathcal{R}_m \leq T \tag{9}$$

To enforce these constraints during training, we apply one of the following sparsity losses:

$$\begin{aligned} \mathcal{L}_{S,unbalanced} &= \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}_{S,m} \\ \mathcal{L}_{S,balanced} &= \max_m \mathcal{L}_{S,m} \end{aligned} \tag{10}$$

where

$$\mathcal{L}_{S,m} = \max \left( \frac{1}{B} \sum_{b=1}^{B} \sigma \left( \frac{\alpha_m^{(b)}}{\tau_0} \right) - T, 0 \right)^2 \tag{11}$$

with $B$ the batch size, $\alpha_m^{(b)}$ the score for node $m$ for the b'th input sample of the batch and $\tau_0$ a temperature constant we set at $0.1$. This sparsity loss replaces the expected value in the constraints of Eqs. 8 and 9 with a batch average and the discrete node decisions $\bar{z}_m$ with the continuous approximation $\sigma\left(\frac{\alpha_m^{(b)}}{\tau_0}\right)$. The fact that this approximation is computed through a sigmoid with a low temperature, without the addition of Gumbel noise means we more closely approximate the behaviour of the selection layer at inference time than we would if we directly penalized the hard decisions $\bar{\mathbf{z}}$. This is important to ensure that if the sparsity constraints are met at training time, they will also be met at inference time. For instance, if the network ensures that for all $X$, $\sigma(\alpha_m) = 0.51$, then due to the addition of Gumbel noise in the computation of $\bar{z}_m$, the network will sample $\bar{z}_m = 1$ and $\bar{z}_m = 0$ an about equal amount of times during training. At inference time however, when no noise is added, the network will always yield $\bar{z}_m = 1$ since $\sigma(\alpha_m) > 0.5$, surely violating the constraints. Also important to note is that using Eq. 10 requires training with a large enough batch size such that the batch average of Eq. 10 is a meaningful estimate of the expected value used in Eqs. 8 and 9.

### D. Dealing with different channel subsets

Performing dynamic channel selection means the classification network $f_\theta$ will see a different subset of active channels depending on the current input sample. Stated in another way, our network needs to be able to deal with missing inputs. This

can cause problems in the learning of the network weights, as the network needs to be able to extract relevant information when a channel is selected, but not cause interference when the channel is not selected and the corresponding input only contains zeros. Ideally, we would employ a number of separate classification networks, each optimized for a specific channel subset. In practice however, this would require training and storage of $2^M$ networks, which quickly becomes infeasible. Thus, the question arises how we can make a single network be able to cope as efficiently as possible when multiple input sets are possible. We tackled this issue by extending our network with the Dynamic Spatial Filtering (DSF) proposed by Banville et al. [10]. The idea of DSF is to re-weight the $M$ input channels using an attention layer. In this setting, new (virtual) channels are formed by applying a spatial filter to all input channels, i.e., making linear combinations of the channels, with the weights being computed from the spatial covariance matrix of the current input window. This re-weighting decreases the impact of missing channels on the network activations and has been shown to make a network more robust against noisy or missing channels.

### E. From centralized to distributed

The channel scoring function $h_\varphi$ in Eq. 1 currently still uses all M input channels to make a decision. However, an important aspect to be taken into account is the distributed nature of WSN platforms, where different channels are recorded on different physical devices. In this setting, we want to reduce the transmission load of these devices by only selecting and thus transmitting the signal of a node when its information is relevant for the current sample. However this will only actually be beneficial when we are able to perform the selection *without centralizing the data of the different sensors* in the first place. Thus, this requires the channel selection module to perform inference in a *distributed* way. To investigate the consequences of the transition from centralized to distributed inference, we will consider three different cases corresponding to different constraints on our dynamic channel scoring function $h_\varphi(X)$:

- *Centralized*: The selection is derived from the joint information of all channels, i.e., $\boldsymbol{\alpha} = h_\varphi(X)$. This setting serves as a theoretical upper bound for the following two practical settings.
- *Distributed*: Each node has to decide whether to transmit solely on its own data, i.e., $\alpha_m = h_{\varphi,m}(\mathbf{x}_m)$ where $x_m$ denotes the $m$-th row of $X$. This allows channel selection to be performed without requiring any data centralization.
- *Distributed-Feedback*: Each node computes a short vector $\boldsymbol{\beta}_m = h_{\varphi,m}(\mathbf{x}_m) \in \mathbb{R}^C$ with $C << L$, that is transmitted to the fusion center. At the fusion center, the $\boldsymbol{\beta}_m$ of all nodes are combined into the stacked vector $\boldsymbol{\beta}$ to determine a final scoring vector $\boldsymbol{\alpha} = g_\phi(\boldsymbol{\beta})$. The discrete selection $\bar{\mathbf{z}}$ resulting from this scoring vector is then returned to the nodes to inform them which of them should transmit. The size of these vectors $\boldsymbol{\beta}_m$ should be small compared to the length $L$ of the window to be transmitted to minimize the overhead cost of the selection. In order to reduce the trainable parameters, one can decide to make the different $h_{\varphi,m}$ models copies of each other, with shared weights for all layers, except the final layer having its own set of parameters for each channel $m$.

These three settings are illustrated in Fig. 5.

### F. Training strategy

Successfully training a model with masking units typically hinges on a good initialization. Since a sparsity loss is much easier to minimize than the training objective - by simply driving the weights of the binary masks to zero - the network can quickly collapse into a state where barely any units are executed [20], [22]. Once this has happened, it is very hard for the network to learn task-relevant information that could pull it out of this state. To avoid this, we adopt a step-wise training strategy that learns one module at a time.

1) Initialize the weights of the classifier $f_\theta$ with the weights of the original M-channel network trained without any dynamic selection.
2) Add the *centralized* dynamic selection layer and train it while fine-tuning (i.e., training at a lower learning rate) the classifier.
3) Add the DSF module and train it while fine-tuning the the dynamic selection and classifier.
4) Transform the centralized dynamic selection layer in a distributed dynamic selection layer and fine-tune the whole model (see below).

To go from a centralized to a distributed architecture, we employ a 2-step transfer learning approach. First, we employ the centralized channel scoring function $h_\varphi$ as a teacher model and try to ensure that the outputs $\boldsymbol{\alpha}_{distr}$ of the student model - the distributed channel scoring function - match the outputs $\boldsymbol{\alpha}_{centr}$ of the teacher by minimizing the following loss:

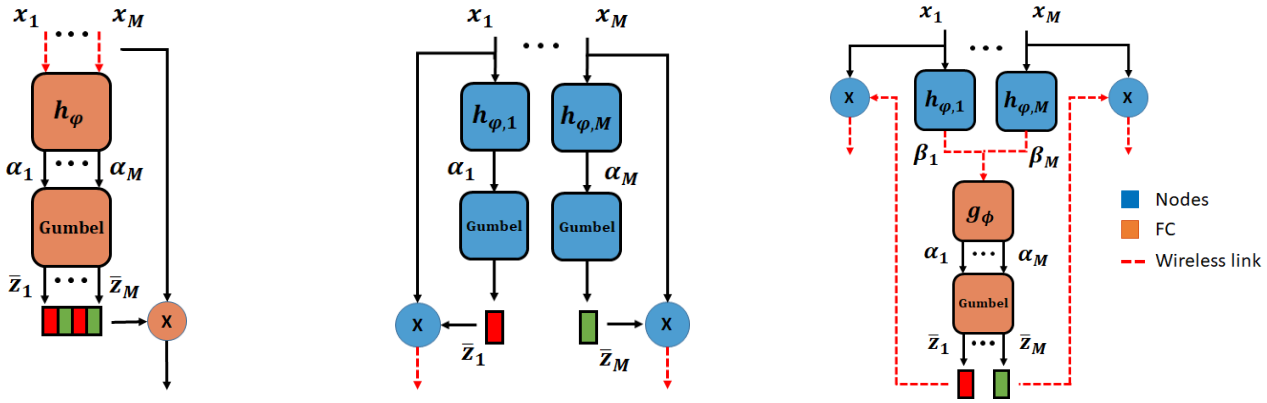
$$\mathcal{L}(h_{\varphi,distr}) = \mathcal{L}_{BCE}(\sigma(\boldsymbol{\alpha}_{distr}), \lfloor \sigma(\boldsymbol{\alpha}_{centr}) \rceil) \qquad (12)$$

with $\mathcal{L}_{BCE}$ the binary cross-entropy loss. By minimizing this loss, we do not necessarily ensure that the channel scores $\boldsymbol{\alpha}$ are exactly alike, but rather that the discrete outputs at inference time will be similar, which is what we ultimately want. In the final step, we use the newly learned distributed channel selection layer, initialize the DSF module and the classifier with the corresponding weights of the centralized model and fine-tune the whole network in an end-to-end fashion.

## IV. Application to Wireless EEG Sensor Networks

### A. Data set

In the field of BCI, the motor execution paradigm is used to decode body movement from the corresponding neural signals in the motorsensory areas of the brain. The High Gamma Dataset [23] contains EEG training data from about 1000 trials of executed movements following a visual cue for each of the 14 subjects. From this, a validation set comprising 200 trials per subject, is extracted for early stopping and model selection. Evaluation is performed on an additional, held-out test set of about 180 trials per subject. The dataset includes 4 classes




(a) Centralized (b) Distributed (c) Distributed-Feedback

Fig. 5: Different settings for the channel scoring module in a sensor network. Blue blocks indicate modules on the sensor nodes, orange modules on the fusion center and red dotted lines communications between the two. (a) The centralized upper bound employs all joint information of all channels to compute the optimal selection for a given input segment, but would require all data to be centralized. Since the transmission load of centralizing the data would outweigh any gains of the dynamic selection, this setting only serves as a theoretical performance benchmark without practical application. (b) The distributed setting does not allow for communication between the nodes (via the fusion center) and makes each selection only dependent on the local data of the corresponding node, eliminating the need for data centralization. (c) The distributed-feedback setting allows for a small amount of communication between the nodes to make a better, joint decision compared to the fully distributed setting.

of movements - left hand, right hand, feet, and rest - with the trials being balanced across these classes. As in [23] we only use the 44 channels that cover the motor cortex, which are preprocessed by resampling at 250 Hz, highpass filtering above 4 Hz, standardizing the per-channel mean and variance to 0 and 1 respectively, and extracting a window of 4.5 seconds for each trial. This pre-processing is adopted from [23] and described in full detail there.

### B. WESN node emulation and selection

In a WESN, each EEG sensor node has its own local set of electrodes and is galvanically separated from the other sensors. As a result, we cannot, we cannot measure the potential between a given electrode and a distant reference (e.g. the mastoid or Cz electrode) , as we would in traditional EEG caps [13]. Instead, we can only record the local potential between two nearby electrodes belonging to the same sensor device. Due to the local measurements with short inter-electrode distances, the expected spatio-temporal correlation across the signals of different nodes is lower than traditional scalp EEG, yet a joint multi-channel processing has been shown to be advantageous [2], [3], [13].

To emulate this setting using a standard cap-EEG recording, we follow the method proposed in [13], which considers each pair of electrodes within a certain maximum distance as a candidate electrode pair or node. By subtracting one channel from the other, we can remove the common far-distance reference and obtain a signal that emulates the local potential of the node. Applying this method with a distance threshold of 3 cm to our dataset, we obtain a set of 286 candidate electrode pairs or nodes, which have an average inter-electrode distance of 1.98 cm and a standard deviation of 0.59 cm.

Given that our WESN will consist of a limited number of mini-EEG devices, we first need to select the $M$ most informative sensor nodes from the pool of 286 candidate nodes. To achieve this, we adopt the static channel selection method described in [16], which enables us to learn the $M$ optimal nodes for a given task and neural network by jointly training the network and a selection layer. Note that this is a fixed selection for the entire data set, not for each sample separately. We train this selection layer, along with the centralized network ($f_\theta$ in Fig. 3 we will use for classification (see Section IV-C), using data from all subjects in the dataset, which results in a subject-independent set of $M$ mini-EEG nodes that are best suited for solving the motor execution task. We do this for 3 different values of $M$, corresponding to a small WESN ($M = 4$ nodes), a medium-size WESN ($M = 8$ nodes), and a high-density WESN ($M = 16$ nodes).

### C. Model architecture

As mentioned above, the neural network architecture we employ for classification ($f_\theta$ in Fig. 3) is the MSFBCNN proposed in [24], which was designed specifically for a motor execution task. Inspired by the Filterbank-CSP approach of [25], this model computes log-power features by applying a number of temporal filters in parallel, aggregating these with spatial filters, and then applying squaring and average pooling over time. These features are then classified by a single linear layer. While the details of this network are not relevant for this study, we provide a summary of this network in table format in Appendix A for completeness.

For our channel scoring module $h_\varphi$ in the centralized setting, we will employ a similar, smaller version of this architecture, with the final linear layer being adapted to output a vector of dimension $M \times 1$ (see Appendix B for details). In the two distributed settings, the different node scoring models $h_{\varphi,m}$ are copies of each other, with shared weights for all layers, except the final layer having its own set of parameters

for each node/channel $m$. The network architecture of these $h_{\varphi,m}$ too is simply a single-input, smaller version of the MSFBCNN, but where the last fully-connected layer outputs the scalar $\alpha_m$ in the distributed setting and the node summary $\boldsymbol{\beta}_m \in \mathbb{R}^{C\times 1}$ in the distributed-feedback setting (see Appendix C). The dimension of these node summaries $\boldsymbol{\beta}_m$ is chosen to be $C = 10$, ensuring the overhead of its transmission is negligible compared to the transmission of the full window of $L = 1125$ time samples.The module $g_\phi(\boldsymbol{\beta})$ aggregating the node summaries in the fusion center is a simple 2-layer multilayer perceptron (MLP) with a hidden dimension of 50 and ReLU nonlinearity. The DSF module also consists of a 2-layer MLP with hidden dimension 50 and ReLU nonlinearity, which is applied to the vectorized sample covariance matrix $\frac{1}{L}\hat{X}\hat{X}^\top$ of the masked input sample and which produces a weight matrix $W \in \mathbb{R}^{M\times M}$ and a bias $b \in \mathbb{R}^{M\times 1}$ which are used to compute a re-weighted output $\tilde{X} = W\hat{X} + b$.

Finally, for training, we follow the procedure described in section III-F, using the Adam optimizer [26] with a learning rate of $10^{-3}$ when a module is trained for the first time and $10^{-4}$ when it is fine-tuned during subsequent steps. A batchsize of 64 is employed and training lasts for 100 epochs with early stopping activated when the validation loss does not decrease for 10 epochs. The hyperparameter $\lambda$, controlling the penalization of the sparsity loss was set to 10 for this application and a fixed temperature $\tau = 1$ was used for the Gumbel-Softmax module.

### D. A note on computational complexity

Before moving on to the experimental results, there are a few important points to consider on computational complexity. While our proposed approach for dynamic channel selection allows for a reduction in the energy required for wireless transmission, it comes at the cost of the introduction of local computations on the sensor nodes in the form of the channel scoring module. These local computations have two important consequences for the WSN. They introduce a computational energy overhead and cause additional latency. To combat both these issues, it is crucial that the channel scoring model is as small as possible. In our experiments, each node's channel scoring model is an extremely small model containing only 7670 parameters and only requiring about 0.08 MFLOPS (floating point operations per second) to operate, which will cause only a negligible overhead in terms of computational energy and latency. For more complex problems, model compression techniques allowing the channel scoring module to operate at lower bit precisions could also still be applied if necessary.

## V. Experimental Results

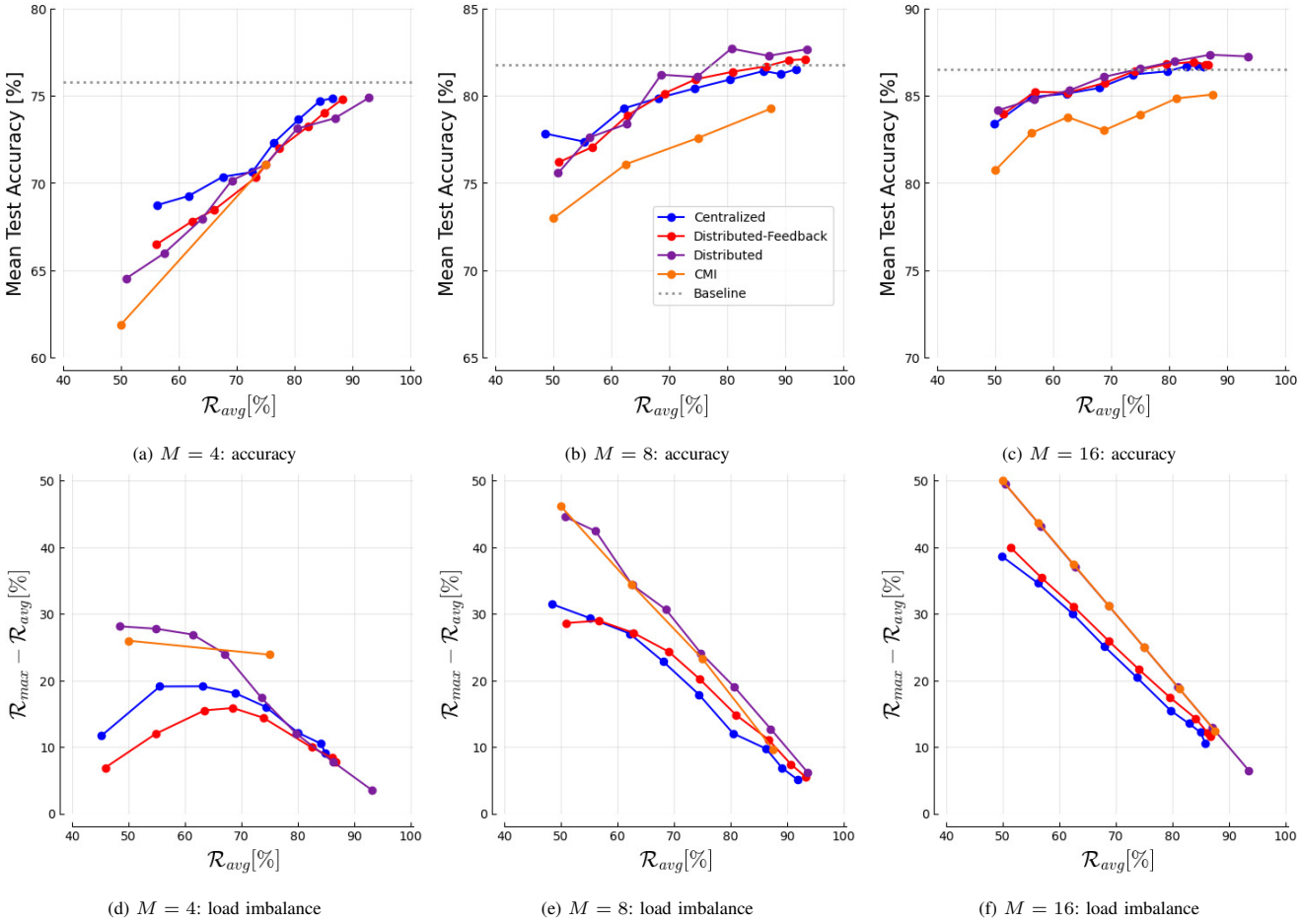
### A. Evaluation procedure

We will now proceed to validate our proposed dynamic channel selection method and how well it is able to trade transmission rate for accuracy in the motor execution WESN described above. We will employ $\mathcal{R}_{max} - \mathcal{R}_{avg}$, the difference between the rate of the node with the highest transmission rate and the average rate across the nodes, as a metric for the imbalance of the node transmission rates. The closer this metric comes to zero, the more similar the different transmission rates are to each other and the more balanced the selection is. We will first consider the scenario where the rate is allowed to be unbalanced across the nodes and then proceed to the more challenging case where the rate must also be balanced across the nodes, i.e., every individual node needs to meet the bandwidth constraints.

In both the balanced and unbalanced scenario's, we compare our distributed and distributed-feedback approaches with a theoretical, centralized upper bound, where all data can be jointly analyzed to perform the selection. In addition, we will compare the performance of our method with a state-of-the-art dynamic feature selection method: the Greedy Conditional Mutual Information (CMI) approach of Covert et al. [19]. The goal of greedy CMI is to find the optimal $K$ out of $M$ features on an instance-wise basis. It does this by learning an iterative policy network where, starting from an empty subset of features, the feature that would be the most informative in conjunction with the features already acquired is selected and added to the subset. This process is repeated until the given, desired amount of features $K$ (a fixed value set a-priori) is reached. The major advantage of this method is that it is able to estimate the value of adding an additional feature, without having access to this feature in the first place. In our WSN setting, this means that it does not require a distributed implementation on the sensor nodes, as it can be fully operated on the fusion center, requesting data from the sensor nodes based on the data it has previously already acquired. However, by construction, the greedy CMI method will select exactly $K$ channels for every input instance. As a consequence, the average rate $\mathcal{R}_{avg}$ achieved in this way will be equal to the ratio $\frac{K}{M}$. Since $K$ and $M$ are integers, this also means that we can only explore a discrete number of target rates. Applying this method to the unbalanced selection scenario thus only requires setting the appropriate value for $K$ given the target $\mathcal{R}_{avg}$ . To extend the greedy CMI method to also be able to deliver balanced selection policies, we apply the balanced sparsity loss of Eq. 10 to the decisions yielded by the policy network. This leads to a new version of CMI, which we will henceforth refer to as 'CMI-balanced' throughout the remainder of this paper.

### B. Unbalanced dynamic selection

Firstly, we will analyze the rate-accuracy trade-off obtained by our proposed dynamic channel selection method and investigate the impact of going from a theoretical, centralized approach to a practical, distributed one in an unbalanced load scenario. To do this, we train our model while applying the unbalanced sparsity loss of Eq. 10 for a given target rate $T$, indicating the average percentage of input samples for which the data of each node should be used and transmitted. We train our model for a range of target rates and for networks consisting of 4, 8 and 16 nodes, each time averaging the




(a) $M = 4$: accuracy (b) $M = 8$: accuracy (c) $M = 16$: accuracy

(d) $M = 4$: load imbalance (e) $M = 8$: load imbalance (f) $M = 16$: load imbalance

Fig. 6: Rate-accuracy trade-off for the proposed dynamic channel selection method for networks of 4,8 and 16 nodes ((a)-(c)) and corresponding load imbalance ((d)-(f)) for the unbalanced transmission load scenario. Mean test accuracies are plotted against the average node transmission rate. Each data point is an average of 5 runs for a given maximal target rate $T$ (see Eqs. 8 and 10). Baseline performance indicates accuracy without dynamic selection involved, i.e. each node transmits at a rate $\mathcal{R}$ of 100%. Dynamic selection consistently outperforms greedy CMI, since it is able to base its decision on all channels jointly, while CMI only sees the channel once it has already added it to the selection. Both methods however, will force one or a few nodes to transmit all the time, leading to a large discrepancy between the critical node and the node average.

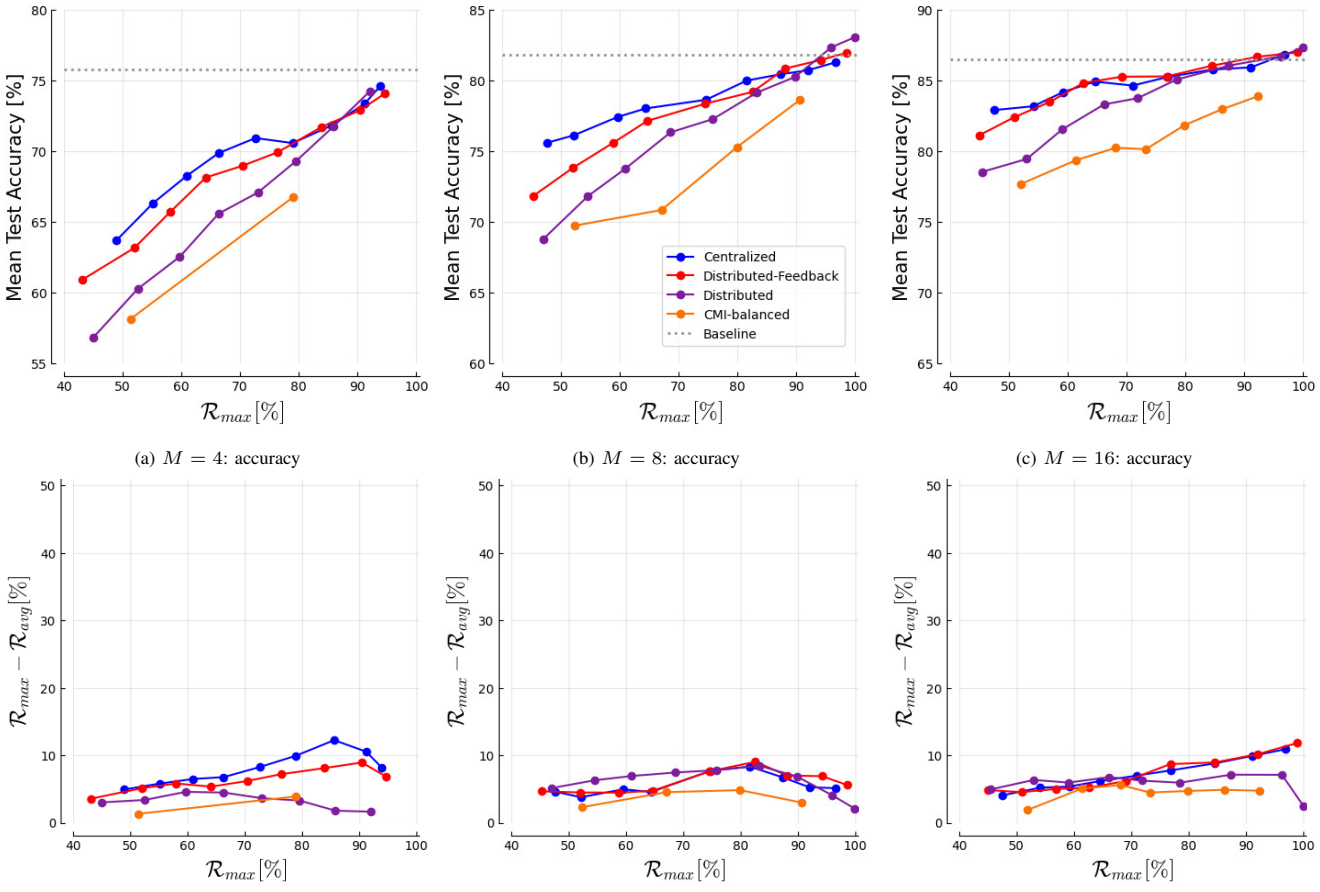
results over 5 runs.

Fig. 6 shows the resulting rate-accuracy tradeoffs and corresponding load imbalance. Firstly, we can observe that moving from a centralized to a distributed network does not affect the performance of the model all that much. Secondly, we can see that the proposed dynamic selection method consistently outperforms the greedy CMI method. One explanation for this is that our proposed method is able to analyze all available channels jointly to decide upon the selection, while the greedy CMI method can only rely on previously selected channels to decide which channel to add next. Also, our proposed method has the advantage that it has more freedom in its selection: it is able to assign a variable amount of nodes for each specific input sample, while the CMI must always assign $K$ nodes. In addition, note that we can only evaluate the CMI performance at a discrete number of target rates, since this is determined by the amount of iterations $K$ the policy network performs.

When comparing the networks with a different amount of nodes, it can be observed that the more nodes are being used, the smaller the relative performance losses are when moving from the starting rate of 100% to a rate of 50% (for instance, the 16-node network only loses 5% accuracy, while the 4-node network loses about 10%). Since there will be a higher amount of redundancy between the 16 nodes than between the 4 nodes, it makes sense that dropping channels in the former has less of an impact on the accuracy than in the latter. Finally, it can be observed that the selections yielded by both methods are highly unbalanced. Without explicit constraints on the per-node rates, some nodes are required to send their data for every single input sample, meaning the lifetime of the WSN is not improved at all by these solutions, as some nodes will run out of energy much quicker than all the others.

### C. Balanced dynamic selection

We will now proceed to investigate the performance of our model when enforcing a fair, balanced load across the nodes by applying the balanced sparsity penalty of Eq. 10 during training. In this scenario, we are interested in the trade-off between the accuracy and the rate of the critical node




(a) $M = 4$: accuracy (b) $M = 8$: accuracy (c) $M = 16$: accuracy

(d) $M = 4$: load imbalance (e) $M = 8$: load imbalance (f) $M = 16$: load imbalance

Fig. 7: Rate-accuracy trade-off for the proposed dynamic channel selection method for networks of 4,8 and 16 nodes ((a)-(c)) and corresponding load imbalance ((d)-(f)) for the balanced transmission load scenario. Mean test accuracies are plotted against the percentage of samples for which the critical node of the network needs to transmit, i.e. the node with the highest percentage of transmission. Each data point is an average of 5 runs for a given maximal target rate $T$ (see Eqs. 9 and 10). Baseline performance indicates accuracy without dynamic selection involved, i.e. each node transmits at a rate $\mathcal{R}$ of 100%. The distribution of the rates between the nodes is far more balanced in this case, with the rate difference between the critical node and the node average rarely exceeding 10%. The more stringent constraints on the balanced load scenario requires more coordination between the nodes to reach an optimal solution, causing a gap to emerge between the distributed implementation and the centralized upper bound. However, sharing a small amount of information between the nodes in the distributed-feedback setting largely overcomes this and performs about as well as the centralized upper bound.

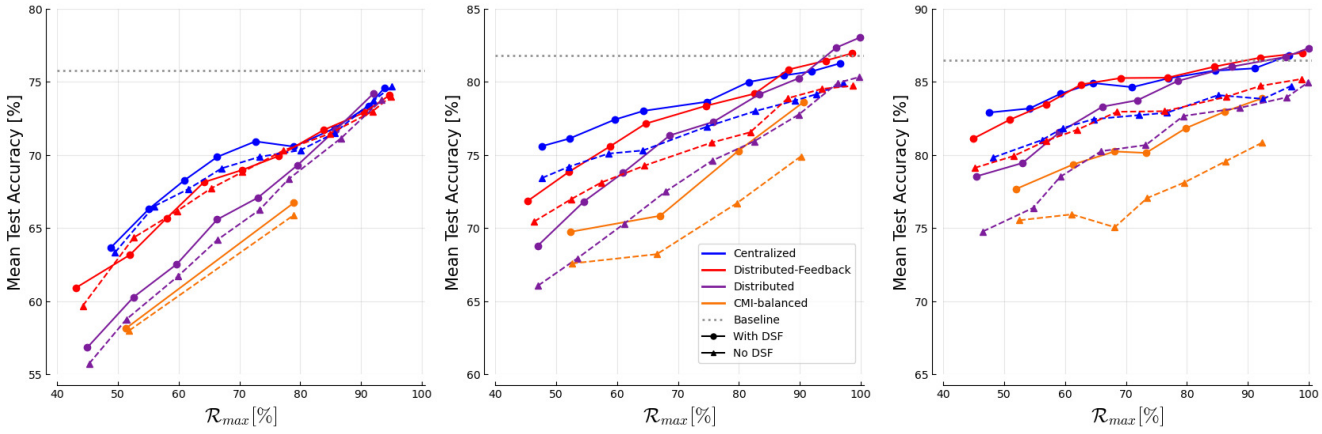
$\mathcal{R}_{max}$ as opposed to the previously employed average $\mathcal{R}_{avg}$, as illustrated in Fig. 7. It can be observed that the balanced sparsity loss enables a far more balanced distribution of the rates, as the rate difference between the critical node and the node average rarely exceeds 10%. The more stringent constraints in this scenario are also reflected in a slight decrease in accuracy for similar rate levels compared to Fig. 6. Important to note is that the distributed network starts to show a noticeable performance gap with the centralized upper bound in this scenario. This can be attributed to the fact that, under these stronger constraints, the network needs to make more intelligent decisions to ensure that the rate of every node stays below the threshold. Being able to coordinate the different node decisions (as in the centralized model) thus offers more opportunities for better performance than when the nodes need to make independent decisions (as in the distributed model). This gap can be overcome by allowing only a very limited amount of communication, as evidenced by the distributed-feedback model which performs very similarly to the centralized upper model.

### D. Impact of dynamic spatial filtering

Next, we analyze the importance of the presence of the DSF module by comparing it with the networks where it has not been added. The results are illustrated in Fig. 8. As mentioned in section III-D, the main purpose of the DSF module is to increase the capability of the classifier to cope with the different channel subsets it is presented with. In this small, 4-node network, the amount of channel subsets $2^M$ is still manageable and the DSF thus offers only marginal improvement. When moving to 8- and 16-node networks, the amount of subsets quickly grows and the performance gains afforded by DSF become more and more salient.

### E. Impact of noisy environments

Up until now, we have discussed situations where we perform a trade-off between the amount of channels we reject and the accuracy of the classifier. In some cases however, working with only a subset of the channels can actually be beneficial for the accuracy as well. In environments where sudden noise bursts can occur, these unexpected inputs, even when limited




(a) $M = 4$ nodes (b) $M = 8$ nodes (c) $M = 16$ nodes

Fig. 8: Rate-accuracy trade-off for the proposed balanced dynamic channel selection method for networks of 4,8 and 16 nodes, with and without inclusion of the DSF module. Baseline performance indicates accuracy without dynamic selection involved, i.e. each node transmits at a rate $\mathcal{R}$ of 100%. While the impact of DSF on the 4-node network is limited, it becomes more important as the size of the network increases and the amount of node subsets the classifier must be capable of processing increases. For 8- and 16-node networks, DSF delivers a consistent performance gain across all settings.

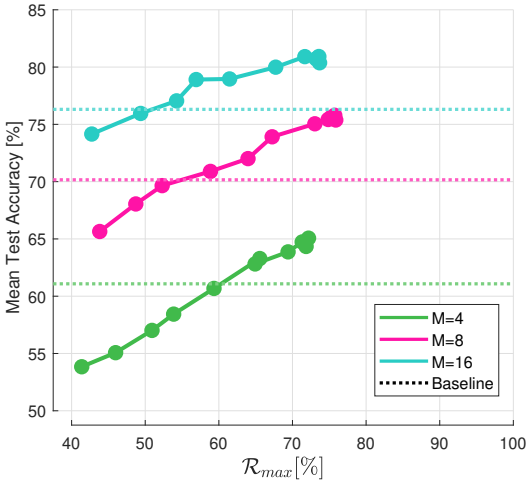



Fig. 9: Rate-accuracy trade-off for the proposed balanced dynamic channel selection method in the distributed-feedback setting for networks of 4, 8 and 16 nodes in a simulation of a noisy environment where each channel has a 25% probability to be replaced by Gaussian noise. When the model is trained with a target rate above this threshold, it automatically rejects the transmission of the noisy and the resulting rate is capped at 75%. Rejecting these noisy channels yields higher accuracies than the baseline network which accepts the noisy channels as input, demonstrating the network is now more robust against this noise.

to a single channel, can heavily disturb the activations of the entire neural network and lead to misclassifications. To make it easier for the network weights to be robust against these noise bursts, it can be beneficial to detect when these happen and zero the corresponding input instead. To test this hypothesis, we repeated our previous experiments with the dynamic selection method, but in this case, each channel of each input window had a 25% chance to be replaced by Gaussian noise with zero mean and standard deviation uniformly sampled between 0 and 3 instead, leaving no more relevant information on this channel. Fig. 9 compares the performance of the distributed-feedback dynamic selection with a baseline network, which directly takes the perturbed data as input. The noise is added during both training and testing to enable a fair comparison, i.e., the network without dynamic channel selection can in principle learn how to cope with these noise bursts. Firstly, it can be observed that the dynamic selection never transmits more information than absolutely necessary: only 75% of the channels actually contain information, so the resulting rate is automatically capped around 75%. Secondly, the automatic rejection of noisy channels does indeed lead to an increased accuracy compared to the baseline accepting this noise as input. A probable reason is that it will be easier for the classifier to find weights that process normal inputs normally and minimize the impact on the activations of disturbances when these disturbances are zero inputs rather than noise bursts.

## VI. Conclusion and Future Outlook

We have proposed a dynamic channel or sensor selection method in order to reduce the communication cost and improve the battery lifetime of WSNs. For each input window, the method selects the optimal subset of sensors to be used by a neural network classifier, while optimizing a trade-off between the amount of channels selected and the accuracy of the given task. The dynamic selection and the classifier are jointly trained in an end-to-end way through backpropagation. The dynamic selection module consists of three major parts: a channel scoring function assigning a relevance score to each channel, a binary Gumbel-Softmax trick converting these scores to discrete decisions and the dynamic spatial filtering module of Banville et al. [10] to make the classifier more robust against the resulting absence of channels. The main advantage of this dynamic selection is that it can incorporate all available node data in its selection, while being computed in a *distributed* way, requiring minimal communication overhead between the nodes.

We have demonstrated the use of this method to perform a trade-off between the transmission rate of the nodes in an

emulated wireless EEG sensor network and the accuracy of a motor execution task. More importantly, we have shown how our approach can elegantly ensure a fair, balanced distribution of the transmission load across the nodes, an aspect that is not present in traditional dynamic feature selection approaches. Additionally, we have presented a use case where the dynamic selection can even improve the accuracy of the model, by automatically rejecting inputs that might harm performance, such as heavy bursts of noise.

In all of the cases above though, there is no point on the accuracy-rate tradeoff curve that can be considered dominant in a Pareto-optimal sense. In other words, improvements in either the model accuracy or transmission load imply a certain drop in the other. The optimal operating point in real-world scenario's will thus depend on the specific constraints of the application. Given that a maximal transmission load can be tolerated due to energy constraints (e.g., related to the battery lifetime of the sensors), the network can be trained with an accordingly set $\mathcal{R}_{max}$ to optimize the accuracy under this constraint. If, conversely, a minimum accuracy must be maintained, but the aim is to minimize the transmission energy required to make this happen, a range of possible $\mathcal{R}_{max}$ can be explored to find the minimal amount of transmission that satisfies the a-priori imposed accuracy constraints.

Though we have focused on the application use case of wireless EEG sensor networks, our methodology is generic and can be applied to sensor networks with any kind of modalities. An important next step is the application of our approach to real-time, streaming applications. In the context of this work, the data windows that needed to be transmitted were independent of each other. In many applications however, temporal correlations will exist between subsequent windows. Thus, it stands to reason that the selection at the current time step can be further improved by re-using data windows or decisions from previous time steps, as in [27]. In future work, we will explore such real-time extensions and applications of this method in other distributed platforms than WESNs.


[1] P. K. Donta, I. Murturi, V. Casamayor Pujol, B. Sedlak, and S. Dustdar, "Exploring the potential of distributed computing continuum systems," *Computers*, vol. 12, no. 10, p. 198, 2023.

[2] A. Bertrand, "Distributed signal processing for wireless EEG sensor networks," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 23, no. 6, pp. 923–935, 2015.

[3] T. Strypsteen and A. Bertrand, "Bandwidth-efficient distributed neural network architectures with application to neuro-sensor networks," *IEEE Journal of Biomedical and Health Informatics*, vol. 27, no. 2, pp. 933–943, 2022.

[4] A. H. Ansari, P. J. Cherian, A. Caicedo, G. Naulaers, M. De Vos, and S. Van Huffel, "Neonatal seizure detection using deep convolutional neural networks," *International journal of neural systems*, vol. 29, no. 04, p. 1850011, 2019.

[5] O. De Wel, M. Lavanga, A. C. Dorado, K. Jansen, A. Dereymaeker, G. Naulaers, and S. Van Huffel, "Complexity analysis of neonatal EEG using multiscale entropy: applications in brain maturation and sleep stage classification," *Entropy*, vol. 19, no. 10, p. 516, 2017.

[6] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance, "EEGNet: a compact convolutional neural network for EEG-based brain–computer interfaces," *Journal of neural engineering*, vol. 15, no. 5, p. 056013, 2018.

[7] D. P. Kumar, T. Amgoth, and C. S. R. Annavarapu, "Machine learning algorithms for wireless sensor networks: A survey," *Information Fusion*, vol. 49, pp. 1–25, 2019.

[8] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[9] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[10] H. Banville, S. U. Wood, C. Aimone, D.-A. Engemann, and A. Gramfort, "Robust learning from corrupted EEG with dynamic spatial filtering," *NeuroImage*, vol. 251, p. 118994, 2022.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[12] T. Lan, D. Erdogmus, A. Adami, M. Pavel, and S. Mathan, "Salient EEG channel selection in brain computer interfaces by mutual information maximization," in *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*. IEEE, 2006, pp. 7064–7067.

[13] A. M. Narayanan and A. Bertrand, "Analysis of miniaturization effects and channel selection strategies for EEG sensor networks with application to auditory attention detection," *IEEE Transactions on Biomedical Engineering*, vol. 67, no. 1, pp. 234–244, 2019.

[14] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini, "Group sparse regularization for deep neural networks," *Neurocomputing*, vol. 241, pp. 81–89, 2017.

[15] A. Abid, M. F. Balin, and J. Zou, "Concrete autoencoders for differentiable feature selection and reconstruction," *arXiv preprint arXiv:1901.09346*, 2019.

[16] T. Strypsteen and A. Bertrand, "End-to-end learnable EEG channel selection for deep neural networks with gumbel-softmax," *Journal of Neural Engineering*, vol. 18, no. 4, p. 0460a9, 2021.

[17] J. Chen, L. Song, M. Wainwright, and M. Jordan, "Learning to explain: An information-theoretic perspective on model interpretation," in *International Conference on Machine Learning*. PMLR, 2018, pp. 883–892.

[18] Y. Li and J. Oliva, "Active feature acquisition with generative surrogate models," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6450–6459.

[19] I. Covert, W. Qiu, M. Lu, N. Kim, N. White, and S.-I. Lee, "Learning to maximize mutual information for dynamic feature selection," *arXiv preprint arXiv:2301.00557*, 2023.

[20] T. Verelst and T. Tuytelaars, "Dynamic convolutions: Exploiting spatial sparsity for faster inference," in *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, 2020, pp. 2320–2329.

[21] E. J. Gumbel, *Statistical theory of extreme values and some practical applications: a series of lectures*. US Government Printing Office, 1948, vol. 33.

[22] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1039–1048.

[23] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball, "Deep learning with convolutional neural networks for EEG decoding and visualization," *Human brain mapping*, vol. 38, no. 11, pp. 5391–5420, 2017.

[24] H. Wu, F. Li, Y. Li, B. Fu, G. Shi, M. Dong, and Y. Niu, "A parallel multiscale filter bank convolutional neural networks for motor imagery EEG classification," *Frontiers in Neuroscience*, vol. 13, p. 1275, 2019.

[25] K. K. Ang, Z. Y. Chin, H. Zhang, and C. Guan, "Filter bank common spatial pattern (fbcsp) in brain-computer interface," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 2390–2397.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] P. K. Donta, S. N. Srirama, T. Amgoth, and C. S. R. Annavarapu, "icocoa: Intelligent congestion control algorithm for coap using deep reinforcement learning," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 3, pp. 2951–2966, 2023.

## APPENDIX I
## MSFBCNN ARCHITECTURE

| Layer | # Filters | Kernel | Stride | # Params | Output | Activation | Padding |
|---|---|---|---|---|---|---|---|
| Input | | | | | $(M, L)$ | | |
| Reshape | | | | | $(1, L, M)$ | | |
| Timeconv1 | $F_T$ | $(64, 1)$ | $(1, 1)$ | $64F_T$ | $(F_T, L, M)$ | Linear | Same |
| Timeconv2 | $F_T$ | $(40, 1)$ | $(1, 1)$ | $40F_T$ | $(F_T, L, M)$ | Linear | Same |
| Timeconv3 | $F_T$ | $(26, 1)$ | $(1, 1)$ | $26F_T$ | $(F_T, L, M)$ | Linear | Same |
| Timeconv4 | $F_T$ | $(16, 1)$ | $(1, 1)$ | $16F_T$ | $(F_T, L, M)$ | Linear | Same |
| Concatenate | | | | | $(4F_T, L, M)$ | | |
| BatchNorm | | | | $2F_T$ | $(4F_T, L, M)$ | | |
| Spatialconv | $F_S$ | $(1, M)$ | $(1, 1)$ | $4MF_TF_S$ | $(F_S, L, 1)$ | Linear | Valid |
| BatchNorm | | | | $2F_S$ | $(F_S, L, 1)$ | | |
| Non-linear | | | | | $(F_S, L, 1)$ | Square | |
| AveragePool | | $(75, 1)$ | $(15, 1)$ | | $(F_S, L/15, 1)$ | | Valid |
| Non-linear | | | | | $(F_S, L/15, 1)$ | Log | |
| Dropout | | | | | $(F_S, L/15, 1)$ | | |
| Dense | $N_C$ | $F_T(L/15, 1)$ | $(1, 1)$ | $F_S(L/15)N_C$ | $N_C$ | Linear | None |

TABLE I: Architecture of the MSFBCNN classifier $f_\theta$ used for motor execution classification. This table is cited from [24]. In our experiments, $M$ is the amount of nodes in the sensor networks, window length $L = 1125$, $F_T = 10$, $F_S = 10$ and the number of classes $N_C = 4$. The total amount of parameters is 6100, 7700 and 10900 for $M = 4$, $M = 8$ and $M = 16$ respectively.

## APPENDIX II
## CENTRALIZED CHANNEL SCORING MODULE ARCHITECTURE

| Layer | # Filters | Kernel | Stride | # Params | Output | Activation | Padding |
|---|---|---|---|---|---|---|---|
| Input | | | | | $(M, L)$ | | |
| Reshape | | | | | $(1, L, M)$ | | |
| Timeconv | $F_T$ | $(16, 1)$ | $(1, 1)$ | $16F_T$ | $(F_T, L, M)$ | Linear | Same |
| BatchNorm | | | | $2F_T$ | $(F_T, L, M)$ | | |
| Spatialconv | $F_S$ | $(1, M)$ | $(1, 1)$ | $MF_TF_S$ | $(F_S, L, 1)$ | Linear | Valid |
| BatchNorm | | | | $2F_S$ | $(F_S, L, 1)$ | | |
| Non-linear | | | | | $(F_S, L, 1)$ | Square | |
| AveragePool | | $(75, 1)$ | $(15, 1)$ | | $(F_S, L/15, 1)$ | | Valid |
| Non-linear | | | | | $(F_S, L/15, 1)$ | Log | |
| Dropout | | | | | $(F_S, L/15, 1)$ | | |
| Dense | $M$ | $F_T(L/15, 1)$ | $(1, 1)$ | $F_S(L/15)M$ | $M$ | Linear | None |

TABLE II: Architecture of the centralized channel scoring module $h_\varphi$, based on the MSFBCNN architecture. In our experiments, $M$ is the amount of nodes in the sensor network, window length $L = 1125$, $F_T = 10$, and $F_S = 10$. The total amount of parameters is 3600, 4000 and 4800 for $M = 4$, $M = 8$ and $M = 16$ respectively.

## APPENDIX III
## DISTRIBUTED CHANNEL SCORING MODULE ARCHITECTURE

| Layer | # Filters | Kernel | Stride | # Params | Output | Activation | Padding |
|---|---|---|---|---|---|---|---|
| Input | | | | | $(1, L)$ | | |
| Reshape | | | | | $(1, L, 1)$ | | |
| Timeconv | $F_T$ | $(16, 1)$ | $(1, 1)$ | $16F_T$ | $(F_T, L, 1)$ | Linear | Same |
| Non-linear | | | | | $(F_T, L, 1)$ | Square | |
| AveragePool | | $(75, 1)$ | $(15, 1)$ | | $(F_S, L/15, 1)$ | | Valid |
| Non-linear | | | | | $(F_S, L/15, 1)$ | Log | |
| Dense | $C$ | $F_T(L/15, 1)$ | $(1, 1)$ | $F_T(L/15)C$ | $C$ | Linear | None |

TABLE III: Architecture of the distributed channel scoring modules $h_{\varphi,m}$, based on the MSFBCNN architecture. Each node $m$ contains one of these, with the output fused in the fusion center with an MLP (see Section IV-C). In our experiments, window length $L = 1125$ and $F_T = 10$. In the distributed setting, each output is simply the binary decision, i.e. $C = 1$. In the distributed-feedback setting, the output is the summary $\boldsymbol{\beta}_m$, i.e., $C = 10$. The total amount of parameters is 911 in the distributed setting and 7670 in the distributed-feedback setting.