# Over-the-Air Split Machine Learning in Wireless MIMO Systems

Yuzhi Yang, Zhaoyang Zhang, Yuqing Tian,
Zhaohui Yang, Chongwen Huang, Caijun Zhong, and Kai-Kit Wong

**Abstract**

In split machine learning (ML), different partitions of a neural network (NN) are executed by different computing nodes, requiring a large amount of communication among them. On the other hand, over-the-air computing (OAC) can efficiently implement all or part of the computation at the same time of communication. In this paper, we propose to deploy split ML in a wireless multiple-input multiple-output (MIMO) communication system by exploiting the MIMO-based OAC capability. In particular, we show that the inter-layer connection in a NN of any size can be mathematically decomposed into a set of linear precoding and combining transformations over the MIMO channels. Therefore, the precoding matrix at the transmitter and the combining matrix at the receiver of each MIMO link, as well as the channel matrix itself, can jointly serve as a fully connected layer of the NN. In such a split ML system, the precoding and combining matrices are regarded as the parameters to be trained, while the MIMO channel matrix is regarded as unknown (implicit) parameters. By exploiting the channel reciprocity between the transmitter and the receiver and properly casting the backward propagation (BP) process through the MIMO channel, the explicit channel estimation process is eliminated, thus greatly saving the system costs and/or further improving its overall efficiency. Finally, we extend the proposed scheme to the widely used convolutional neural networks and demonstrate its effectiveness under both the static and quasi-static memory channel conditions with comprehensive simulations.

This work was supported in part by National Key R&D Program of China under Grant 2020YFB1807101 and 2018YFB1801104, and National Natural Science Foundation of China under Grant U20A20158, 61725104 and 61631003.

Y. Yang, Z. Zhang (Corresponding Author), Y. Tian, C. Zhong and C. Huang are with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310007, China, and with the International Joint Innovation Center, Zhejiang University, Haining 314400, China, and also with Zhejiang Provincial Key Lab of Information Processing, Communication and Networking (IPCAN), Hangzhou 310007, China. (e-mails: {yuzhi_yang, ning_ming, tianyq, caijunzhong, chongwenhuang}@zju.edu.cn)

Z. Yang and K. Wong are with the Department of Electronic and Electrical Engineering, University College London, WC1E 6BT London, UK. (emails: {zhaohui.yang, kai-kit.wong}@ucl.ac.uk)

**Index Terms**

Over-the-air computing (OAC), multiple-input multiple-output (MIMO), split machine learning, neural network

## I. Introduction

### A. Motivation

In future sixth-generation (6G) wireless communication systems, human-like intelligence will be brought everywhere in networking systems [1]. The rapid development of artificial intelligence leads to booming mobile machine learning (ML) applications and requires vast data transmission. Split ML is a common method to distribute a NN to several devices to ease the computation burden on each device. Each device calculates the intermediate results in a split ML system and transmits it to the next device in order. Each device passes the intermediate gradient in backward order when casting backward propagation. The system can be applied in cloud networks, net-of-things, or even the deep learning-based joint source-channel coding (JSCC) problem [2]. However, split ML requires frequent information exchange among edge devices, which increases the communication burden for wireless communications. For the specific structure of the split ML system, we can reduce the communication overhead cost by coupling communication and computation.

To realize dual-functional communication and computation, over-the-air computation (OAC) has been proposed as an alternative to the traditional digital communication structures [3]. By using the characteristics of wireless channels, OAC enables wireless communication systems to compute some large-scale but simple calculation tasks [4]. Through OAC, a float number can be transmitted by an analog symbol instead of several binary symbols, sharply reducing the number of symbols. Previous work [5] shows that if a group of devices transmits analog modulated signals simultaneously, the receiver directly obtains an aggregated signal, which can be directly applied to the typical federated learning (FL) network. However, traditional OAC works only consider the weighted sum of multiple edge users' messages and do not suit the split ML system directly.

To fit the structure of split ML, OAC can also be applied in the general wireless communication systems besides the time-synchronized multiple access systems. In the widely-applied multiple-input multiple-output (MIMO) systems, a group of antennas on the transmitter send different signals simultaneously, and another group of antennas on the receiver collect the transmitted signal. On the receiver side, each antenna can individually receive the aggregated signal from multiple antennas on the transmitter. Due to different channel gains, each antenna on the receiver side can perform individual computation tasks. In particular, the MIMO channel can be viewed as a multiplication-and-addition procedure on the transmitted analog signals, which is widely used in NNs.

Due to MIMO channels' characteristics, there is an intricate interplay between MIMO OAC and NN. A MIMO channel can provide *full connection* between the inputs and the outputs and can be viewed as a *weighted sum calculator*. Unlike fully connected layers in NNs, which can be viewed as controlled weighted sums, the weights in MIMO systems are determined by the channel matrices, which is uncontrollable in reality. To control the weight in the equivalent system, we can introduce the beamforming and combining procedure to our system. Both beamforming and combining are controllable linear transformations on the signal and are necessary parts of MIMO systems. With the help of them, *we can control the parameters* in the equivalent weighted sum of the overall system. Simply implementing the MIMO OAC in split ML introduces two fundamental issues: the forward and backward channels are different and may not be accurately known, and the analog transmission in OAC results in unavoidable noise.

To deal with the uncertainty issue of the MIMO channel, we find that the mathematical operation is similar to the forward-backward propagation of a NN and channel reciprocity of a wireless communication network (see Section II-B for details). Thus, deploying a NN through MIMO OAC can lead to correct gradients even without any priori knowledge about the MIMO channel according to reciprocity and quasi stability of the channel. Moreover, in most NNs, we cannot accurately interpret the intermediate results, but the systems can work well after proper training. The unexplainably in NNs inspires us that casting a deterministic linear transformation, i. e., a MIMO channel, on the intermediate results of a NN through the whole training and execution period may not intensively deteriorate the performance.

The other issue about the noise in wireless communication is not fatal in NNs. From the perspective of information theory, both digital and analog communication can be optimal in wireless communications such as sensor networks [6]. However, the digital communication system can reduce transmission errors by employing error detecting and correcting codes, whereas transmission errors can only be restricted but never eliminated in analog communication. The unavoidable noise in analog communication has strongly restricted its development. However, it is found that noise is tolerable and sometimes even becomes a training trick in NNs [7]. Since the devices transmit intermediate results of the NN in split ML systems, the advantage of progressive error-free does not exist in such applications. Based on the above finding, Jankowski et al. [2] also show similar results that analog communication performs better than digital communication in the deep learning-based JSCC problem, which is a particular case of split ML.

## *B. Related Works*

In communication systems, OAC is usually used in multiple access systems to compute the weighted sum or some easy mathematical operations such as geometric mean, polynomial, and Euclidean norm [4]. Hence most OAC works are mainly used in FL, where the weighted sum is widely deployed. For example, the authors in [5] optimize the number of simultaneous accesses, which may improve the efficiency of FL. Besides, Zhu et al. apply broadband analog aggregation to improve OAC in FL with multiple bands [8], while Shao et al. consider FL with misaligned OAC [9].

OAC for multiple access systems still has significant drawbacks. Firstly, OAC requires strict synchronization among all transmitters, which is hard to realize. Moreover, OAC does not support backward communication, which is rarely considered in previous works as far as we know. Furthermore, the above works [4], [5], [8], [9] do not consider the MIMO system, which is widely used in practical scenarios. The multiple antennas of the transmitter in a MIMO system can be viewed as a group of transmitters in the multiple access scheme, which overcomes the drawbacks above. The authors of [10] apply MIMO OAC to multimodal sensing. However, in [10], the scenario is still a multiple access system where all channels are MIMO channels, and the task of OAC is still the weighted sum, which can be regarded as a direct expansion of previous designs in [4], [5], [8], [9]. Besides, the implementation of [10] is strictly limited to FL applications, which is not suitable for split ML.

OAC have also provided an alternative to traditional NNs by realizing parts of NNs with acoustic [11], optical [12], and RF [13] signals. The authors in [11]–[13] use the characteristics of the target systems similar to NNs, and employ the target systems as part of a NN. OAC systems calculate aggregated results of multiple inputs from different transmitters or time slots by moderating the environments or some parts of the system. Recently, in wireless communications, Sanchez et al. also realize NNs with the help of multiple paths and reconfigurable intelligent surfaces [14]. Their system transmits the intermediate output of NNs via time sequential signals and uses the delay of multiple paths to realize one-dimensional convolution. However, in this paper, we use MIMO channels to realize fully connected layers through multiplexed signals.

To realize NNs over MIMO channels, we utilize the structure of split ML. Split ML is a method where multiple computation nodes cooperatively execute an ML application. In such a system, the ML model is split into multiple parts allocated to different computation nodes. Each node executes a part of the ML model in order and transmits the intermediate results to the next one. When training the NN, the nodes also execute backward propagation in backward order. Most of the existing works [15]–[17] on split ML focus on how to distribute the model on the nodes in a way to minimize the total communication and computation delay. There are also some other works bringing up specialized NN structures for split ML [18]–[20]. Recent work also tries to deploy a proper NN architecture on a given communication network [21]. Their framework utilizes the neural architecture search method to meet latency and accurate requirements. However, the above works [15]–[21] all consider ideal communication among the computing nodes, which ignores the communication scheme design.

## *C. Contributions*

The main contributions of this paper are summarized as follows:

- A split ML framework is proposed for wireless MIMO networks by exploiting the MIMO's OAC capability, which not only enables high-throughput and efficient wireless transmission, but also reduces the overall computation load by synergistically incorporating the split ML process with the wireless transmission procedure rather than just taking it as a bit pipe.
- By exploiting the reciprocity of MIMO channel in the forward and backward propagation procedures in the proposed framework, we find it unnecessary to conduct explicit channel estimation as otherwise indispensable in conventional communication systems, thus further improving the overall communication and computation efficiency.
- We also provide some design rules for the proposed system so as to apply it to a fully connected layer of any size in a FCN or a convolutional layer of any size in a CNN. Simulation results show that the proposed scheme is efficient under both static and slowly-varying memory channel conditions.

The remainder of the paper is organized as follows. We first introduce the proposed system in Section II. We then mathematically provide some principles and propose a training algorithm

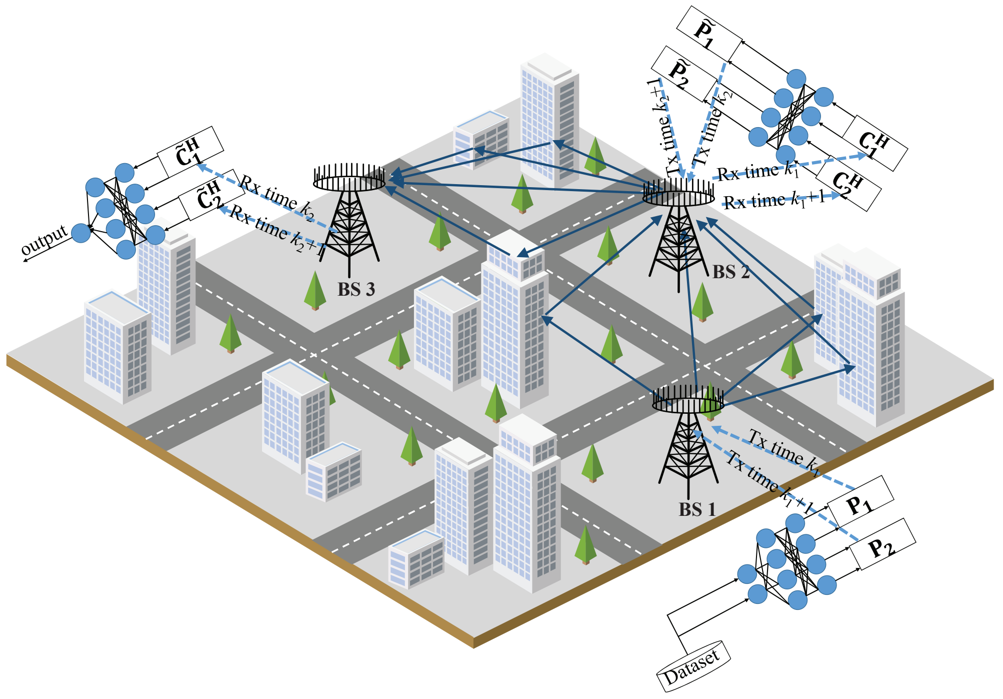

**Fig. 1:** The considered split ML system based on MIMO-based OAC scheme.

in Section III. The proposed system is extended to convolutional NNs in IV. Numerical results are provided in Section V. Section VI concludes the paper and provides future directions.

## II. System Model

In this section, we first briefly introduce the considered system model and then propose a MIMO OAC-based approach to accelerate the communication in split ML.

### A. MIMO OAC Model

Consider a MIMO OAC-based split ML system with multiple base stations, as shown in Fig. 1. Each base station is equipped with multiple antennas, and the MIMO channels among base stations are quasi-stable. A deep NN is split into several small NN, each of which is deployed on one specific base station. The base stations progress the forward computation and backward propagation of the NN. For simplicity, unless otherwise stated, we consider the considered split ML system with only one splitting point since we can decompose split ML systems with multiple splitting points to a set of single split points. To simplify the description, we use “transmitter” and “receiver” to refer to the transmitter and receiver with $N_t$ and $N_r$ antennas in the forward transmission.

Assume that the channel between the transmitter and the receiver is a quasi-stable MIMO channel with reciprocity, i. e., the forward channel is $\mathbf{H} \in \mathcal{C}^{N_r \times N_t}$, and the backward channel is $\mathbf{H}^T$. We do not make any further priori hypotheses of the channel as the split ML applications may be deployed in different scenarios. When designing the system, we assume that the rank of channel $\mathbf{H}$ is known to be $r$, which determines the maximum amount of dataflows transmitted simultaneously under $\mathbf{H}$. We note that although knowing the accurate value of $r$ is impossible without channel estimation, we can get $r$ roughly from the wireless communication scenario. Some small singular values, which are below a threshold, can be regarded as zeros in the aspect of engineering. Hence, the rank $r$ can be underestimated in some cases. In designing the system, we assume that $r$ is accurately known, and we show the cases where there exists a mismatch numerically in the simulations. Besides, to better realize OAC, we may apply other techniques such as using reconfigurable intelligent surfaces to obtain a channel with a higher rank or applying the orthogonal frequency division multiplexing technique to realize multiple transmissions simultaneously. However, we do not consider those techniques since they do not affect the principles we use in the paper and can be easily combined with the proposed system.

## *B. OAC for Split Machine Learning*

We consider a two-node split ML system, where the original NN is split into two parts, i. e., the former and latter parts, which are respectively deployed on the transmitter and the receiver. A fully connected layer between the transmitter and the receiver is realized through the OAC technique. The transmitter can be regarded as a federation of several synchronized single-antenna transmitters in MIMO systems. Hence OAC still works similarly with multiple access systems. We will later discuss how to extend the proposed system to convolutional layers in Section IV-A. In particular, a fully connected layer in a NN can be represented by

$$\boldsymbol{x}_{\text{next}} = \phi(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}), \tag{1}$$

where $\boldsymbol{x} \in \mathcal{C}^{N_i * 1}$ is the input of the considered layer, $\mathbf{W} \in \mathcal{C}^{N_o * N_i}$ and $\boldsymbol{b} \in \mathcal{C}^{N_o * 1}$ are the network parameters, $\phi(\cdot)$ is the element-wise activation function, and $\boldsymbol{x}_{\text{next}}$ is the output.

Considering the procedure of OAC, the transmitted and received signals may be amplified. Rather than using amplified parameters, we can adopt a batch normalization (BN) layer to remain the other parameters in a reasonable range and accelerate training [22]. A BN layer casts an element-wise linear transformation to the forward propagated data and normalizes

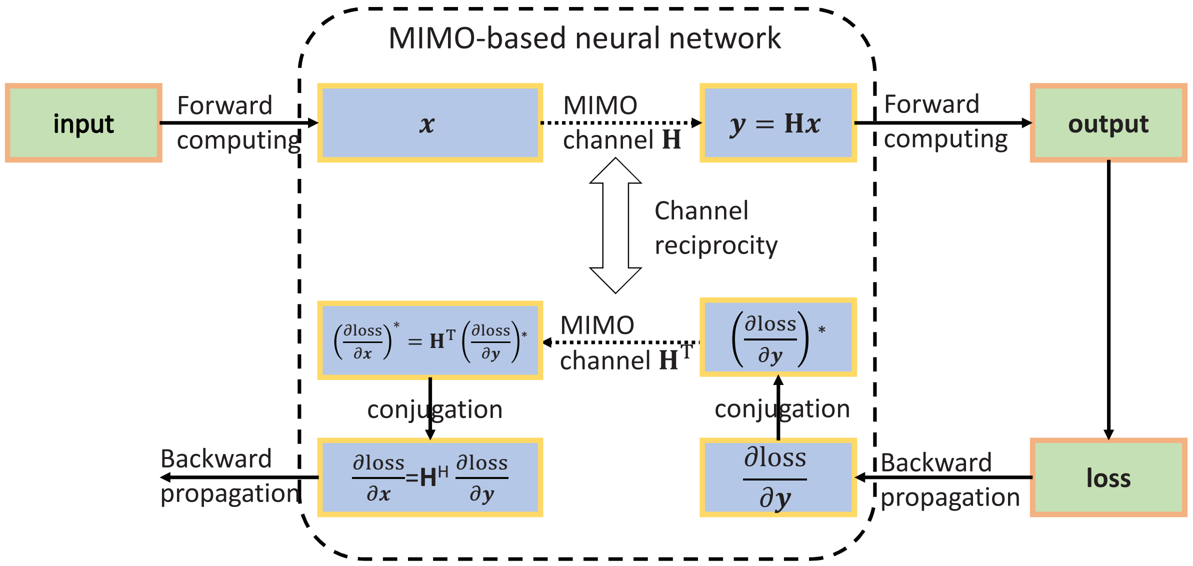
MIMO-based neural network
input
Forward computing
$x$
MIMO channel $\mathbf{H}$
$y = \mathbf{H}x$
Forward computing
output
Channel reciprocity
$\left(\frac{\partial \text{loss}}{\partial x}\right)^* = \mathbf{H}^T \left(\frac{\partial \text{loss}}{\partial y}\right)^*$
MIMO channel $\mathbf{H}^T$
$\left(\frac{\partial \text{loss}}{\partial y}\right)^*$
conjugation
conjugation
Backward propagation
$\frac{\partial \text{loss}}{\partial x} = \mathbf{H}^H \frac{\partial \text{loss}}{\partial y}$
$\frac{\partial \text{loss}}{\partial y}$
Backward propagation
loss

**Fig. 2:** The relationship between channel reciprocity and backward propagation

them to a predetermined mean and variation. The BN layer normalizes the intermediate output calculated by OAC to a reasonable range and can counteract the widely existing amplifications in communication systems, making it suitable for the considered system. With the BN layer, the output becomes

$$\boldsymbol{x}_{\text{next}} = \phi(\text{BN}(\boldsymbol{y} + \boldsymbol{b})) = \phi(\text{BN}(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})), \tag{2}$$

where $\boldsymbol{y} = \mathbf{W}\boldsymbol{x}$ is calculated through OAC and $\boldsymbol{x}_{\text{next}}$ is calculated on the receiver by the received $\boldsymbol{y}$. In the remaining parts of this paper, we only consider the OAC part of the system, i. e., how to calculate $\boldsymbol{y}$ from $\boldsymbol{x}$, since other parts have been well studied in traditional NNs. Below, we discuss the mathematical similarity between the channel reciprocity and the forward-backward propagation in NNs, which is also shown in Fig. 2.

*1) Forward Computation:* For each intermediate result, we conduct $K$ times of OAC for each forward computing. In the $k$-th OAC, the precoding and combining matrices are denoted by $\mathbf{P}_k \in \mathcal{C}^{r*N_i}$ and $\mathbf{C}_k \in \mathcal{C}^{r*N_o}$, respectively. Hence, the $k$-th OAC can be described as

$$\begin{aligned} \boldsymbol{x}_{t,k} &= \mathbf{P}_k \boldsymbol{x}, & (3) \\ \boldsymbol{y}_{r,k} &= \mathbf{H}\boldsymbol{x}_{t,k} + \boldsymbol{n}_k, & (4) \\ \boldsymbol{y}_k &= \mathbf{C}_k^H \boldsymbol{y}_{r,k}, & (5) \end{aligned}$$

corresponding to the precoding, transmitting, and combining procedures, respectively, where $\boldsymbol{n}_k \sim \mathcal{CN}(0, \sigma^2\mathbf{I})$ is the Gaussian white noise vector with variance of $\sigma^2$ of the forward transmission. The receiver then combines all the $K$ received signals and the result of OAC becomes:

$$\boldsymbol{y} = \sum_{k=1}^{K} \boldsymbol{y}_k = \sum_{k=1}^{K} \mathbf{C}_k^H \mathbf{H} \mathbf{P}_k \boldsymbol{x} + \boldsymbol{n}_k. \tag{6}$$

*2) Backward Propagation:* Here, we consider the gradient backward propagation of our system. Let the gradient of $\boldsymbol{y}$ be $\boldsymbol{g}_y \triangleq \partial \text{loss}/\partial \boldsymbol{y}$. According to (3)-(5) and the principle of gradient backward propagation, we can easily obtain that:

$$\boldsymbol{g}_{C_k} = \boldsymbol{y}_{t,k} \boldsymbol{g}_y^H, \tag{7}$$

$$\boldsymbol{g}_{x_{t,k}} = \mathbf{H}^H \mathbf{C}_k \boldsymbol{g}_y, \tag{8}$$

$$\boldsymbol{g}_{P_k} = \boldsymbol{x} \boldsymbol{g}_{x_{t,k}}^H, \tag{9}$$

$$\boldsymbol{g}_x = \sum_{k=1}^{K} \mathbf{P}_k^H \boldsymbol{g}_{x_{t,k}}. \tag{10}$$

All above equations can be easily calculated by either the receiver or the transmitter except for (8). Equation (8) can be calculated with noise through OAC, where the receiver acts as a transmitter in OAC and transmits $\boldsymbol{g}_y^*$ with beamforming matrix $\mathbf{C}_k^*$. For the backward propagation, the received signal at the transmitter becomes:

$$\tilde{\boldsymbol{g}}_{x_{t,k}} = \mathbf{H}^T \mathbf{C}_k^* \boldsymbol{g}_y^* + \tilde{\boldsymbol{n}}_k = \boldsymbol{g}_{x_{t,k}}^* + \tilde{\boldsymbol{n}}_k, \tag{11}$$

where $\tilde{\boldsymbol{n}}_k \sim \mathcal{CN}(0, \sigma^2\mathbf{I})$ is the Gaussian white noise vector of the backward channel. Equation (11) reveals that channel reciprocity ensures that the backward OAC is identical to backward propagation in NNs if we transmit the conjugation of the gradient and ignore the noise.

## III. Theoretical Analysis and Algorithm Design

In this section, we first provide theoretical findings for split ML based on fully connected layers over wireless networks. We then formulate the optimization problem and propose the training algorithm for the proposed system.

### *A. Theoretical Findings in the Proposed System*

Here, we only consider the matrix multiplication in the fully connected layer, i. e., $\boldsymbol{y} = \mathbf{W}\boldsymbol{x}$. To ensure that the split ML system can work well, we must guarantee that the OAC system can be equivalent to any parameter $\mathbf{W}$ of the fully connected layer. Hence, we have the following feasible condition for noiseless channels.

To realize the function of a fully connected layer, condition (12) should have at least a feasible solution $(\mathbf{P}_k, \mathbf{C}_k), k = 1, \cdots, K$ for any channel $\mathbf{H} \in \mathcal{C}^{N_r * N_t}$ with rank $r$ and parameter $\mathbf{W} \in \mathcal{C}^{N_o * N_i}$, where $N_i$ and $N_o$ are the dimensions of the input and the output, respectively.

$$\mathbf{W} = \sum_{k=1}^{K} \mathbf{C}_k^H \mathbf{H} \mathbf{P}_k. \tag{12}$$

From feasible condition (12), we can obtain the following lemma.

*Lemma* 1. If the rank of $\mathbf{G} \triangleq (\mathbf{C}_1^H \mathbf{H}, \cdots, \mathbf{C}_K^H \mathbf{H})$ is at least $N_o$ and $\mathbf{P}_k$ can take any values from $\mathcal{C}^{N_t \times N_i}$, then feasible condition (12) always holds. Symmetrically, if the rank of $\mathbf{G}' \triangleq (\mathbf{P}_1^H \mathbf{H}^H, \cdots, \mathbf{P}_K^H \mathbf{H}^H)^H$ is at least $N_i$ and $\mathbf{C}_k$ can take any values from $\mathcal{C}^{N_r \times N_o}$, then feasible condition (12) always holds.

*Proof.* We first prove the first part. Since $\mathbf{G} \triangleq (\mathbf{C}_1^H \mathbf{H}, \cdots, \mathbf{C}_K^H \mathbf{H})$, we have $\sum_{k=1}^{K} \mathbf{C}_k^H \mathbf{H} \mathbf{P}_k = \mathbf{G}(\mathbf{P}_1^H, \cdots, \mathbf{P}_K^H)^H$. Due to the fact that the rank of $\mathbf{G}$ is at least $N_o$, we can easily observe that for arbitrary $\mathbf{W} \in \mathcal{C}^{N_o \times N_i}$, let $\mathbf{P}_i, k = 1, \cdots, K$ be the submatrix consisting the $(kr - k + 1)$-th to the $(kr)$-th rows of $\mathbf{G}^{-1}\mathbf{W}$. Then, $\mathbf{W} = \sum_{k=1}^{K} \mathbf{C}_k^H \mathbf{H} \mathbf{P}_k$, which completes the first part of Lemma 1. The second part of Lemma 1 is symmetric to the first part, hence Lemma 1 is proved. □

Based on Lemma 1, we further obtain the requirement for the time $K$ of OAC and the rank $r$ of $\mathbf{H}$ that can satisfy(12).

*Theorem* 1. Under the condition that $\mathbf{P}_k$ and $\mathbf{C}_k$ can take any values from $\mathcal{C}^{N_t \times N_i}$ and $\mathcal{C}^{N_r \times N_o}$ respectively, feasible condition (12) holds if and only if $Kr \geq \min\{N_i, N_o\}$.

*Proof.* We first provide the necessary condition. Since rank$(\mathbf{C}_k^H \mathbf{H} \mathbf{P}_k) \leq$ rank$(\mathbf{H}) = r$, we have

$$\text{rank}\left(\sum_{k=1}^{K} \mathbf{C}_k^H \mathbf{H} \mathbf{P}_k\right) \leq Kr. \tag{13}$$

Hence if $Kr < \min\{N_i, N_o\}$, for any $\mathbf{W}$ with rank $\min\{N_i, N_o\}$, feasible condition (12) does not hold.

Then, we provide the sufficient condition. We know rank$(\mathbf{I}_K \otimes \mathbf{H}) = Kr$, where $\otimes$ denotes the Kronecker product and $\mathbf{I}_K$ denotes the $K \times K$ identity matrix. If $N_i \geq N_o$, we can easily know that there exist some $\mathbf{C}_k, k = 1 \cdots, K$, such that rank$(\mathbf{G}) = Kr > N_o$, where $\mathbf{G}$ is defined in Lemma 1. From the first part of Lemma 1, feasible condition (12) holds. Similarly, if $N_i < N_o$, we can still find some $\mathbf{P}_k, k = 1 \cdots, K$, which satisfy the condition of the second part of Lemma 1, and hence feasible condition (12) also holds.

As a result, Theorem 1 is proved. $\square$

Lemma 1 and Theorem 1 give the sufficient and necessary conditions to feasible condition (12). Theorem 1 shows that to satisfy feasible condition (12), we should at least conduct $\min\{N_i, N_o\}/r$ times of transmissions. The proof to Theorem 1 also guides us that we must follow the conditions in Lemma 1 to reach the bound. We will later show how to design the system with minimum transmission times following the conditions above.

In practical communication systems, there exist noise in both forward and backward communications, and the maximum transmitting powers for both transmissions are also limited. However, the influence of the noise on the forward and backward channels are not the same. In the forward channel, the noise directly takes part in the forward computation and results in worse performance on the task. However, the noise only affects the gradient calculation in the backward channel. The well-known stochastic gradient descent (SGD) [23] algorithm uses gradients on random parts of the total dataset instead of the entire dataset and still works well since the expectation of the stochastic gradient equals the actual gradient. Since the expectation of the gradient added with zero-mean noise still equals to the real gradient, the noise in the backward channel may not destroy the learning performance.

Besides, we note that all NNs consist of only two components, i.e., linear layers and nonlinear activation functions. Since the activation functions are always unary, the backward propagation procedures of both types of components are linear. Hence the backward propagation from the output of the NN to any parameter is linear. The only nonlinear part when training NNs may lie in the loss function at the output. Hence if the noise at the backward receiving antenna is Gaussian, as shown in Fig. 3, compared with the corresponding centralized NN, the gradient for any parameter should be either noiseless or with Gaussian noise.

Inspired by SGD, we know that the white noise in the backward channel does not have obvious significance in convergence, which can also be theoretically shown in the following Theorem 2. Before introducing Theorem 2, we define the following notations. Denote $h(\boldsymbol{x}; \boldsymbol{\theta})$ as the model, where $\boldsymbol{x}$ represents the input and $\boldsymbol{\theta}$ stands for the parameters. With the SGD algorithm, the loss of the $t$-th batch can be written by

$$f_t(\boldsymbol{\theta}) \triangleq \sum_{b=1}^{B} \ell\left(h\left(\mathbf{x}_{b,t}; \boldsymbol{\theta}\right), y_{b,t}\right), \tag{14}$$

where $B$ represents the batch size, $\ell(\cdot,\cdot)$ is the loss function, $\mathbf{x}_{b,t}$ and $y_{b,t}$ represent the $b$-th input data and the corresponding label of the $t$-th batch, respectively. Now, it is ready to provide the following theorem about convergence with noise in the backward channel.

*Theorem* 2. If each loss function $f_t(\boldsymbol{\theta})$ is convex, all the parameters and the gradients are bounded, i. e. $\|f_t(\boldsymbol{\theta}) - f_t(\boldsymbol{\theta}')\| \le D$ and $\|\boldsymbol{g}_t\| \le G$ for any $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$, where $\boldsymbol{g}_t = \nabla f_t(\boldsymbol{\theta})$, the SGD algorithm converges in $\mathcal{O}(\sigma^2 T^{-1/2})$ even with noise $\boldsymbol{n}$. However, the coefficient in the convergence time increases linearly with noise power.

*Proof.* Please refer to Appendix A. □

Theorem 2 indicates that the introducing of Gaussian white noise in the backward propagation will not affect the order in convergence rate of SGD. Besides, the noise power only linearly changes the coefficient of the convergence rate.

Another problem comes from the amplification. In wireless communication systems, due to the effect of noise and limited budget of the transmitting power, we must apply amplification for both forward and backward transmissions to match the transmit power, which may have an influence on the performance of NNs. Hence, in the following, we discuss how to realize arbitrary amplification in the forward computation and backward propagation without scarifying the performance of NNs significantly.

In the forward computation, we can apply a simple power normalization before transmitting, i. e., transmit $\mathbf{P}_k\boldsymbol{x}/A$ instead of $\mathbf{P}_k\boldsymbol{x}$, where $A = \sqrt{\mathbb{E}(\boldsymbol{x}^H\mathbf{P}_k^H\mathbf{P}_k\boldsymbol{x})}$, as shown in Fig. 3. Since the normalization layer amplifies all the transmitted signals and there is a BN layer at the receiver to offset the effect of power amplification, it will not significantly affect the NN's result.

In the backward propagation, normalizing the power of the gradient of the backward propagation will only cause the gradient of the subsequent calculation to be amplified on the same scale. To overcome this, we can use optimizers insensitive to the magnitude of the gradient, such as the Adam optimizer [24].

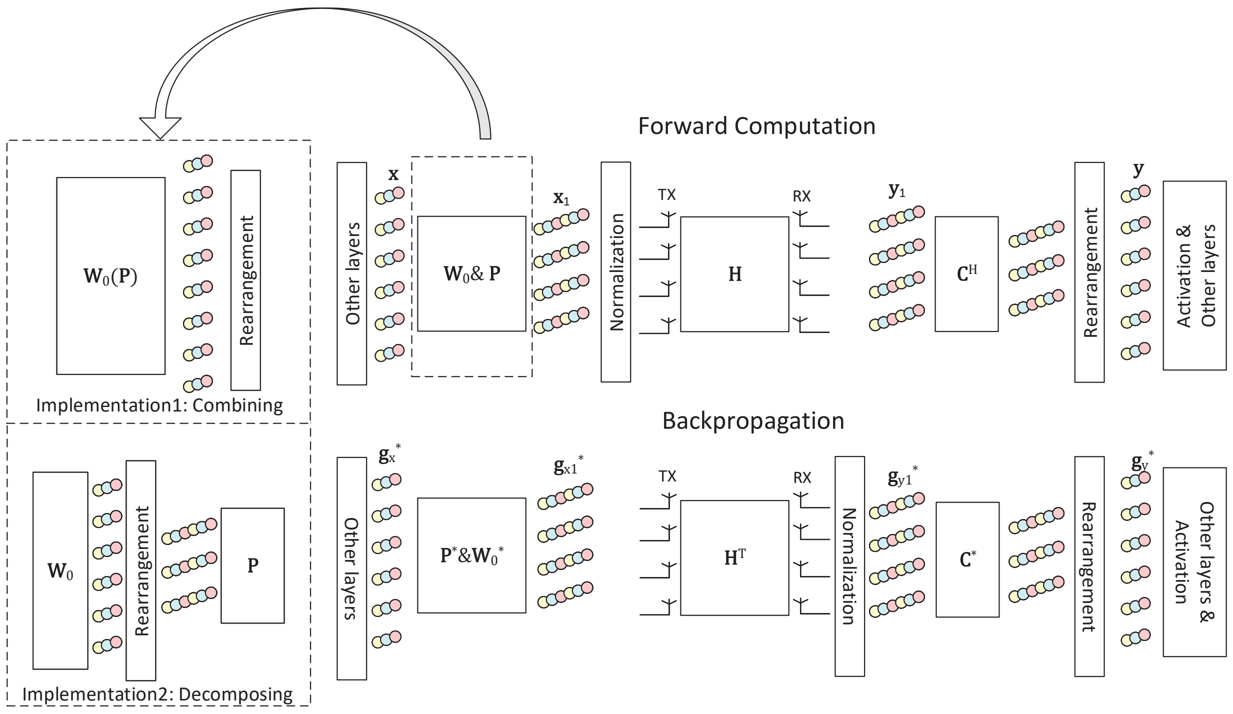

**Fig. 3:** Two transmitter parameterized implements of the proposed MIMO OAC-based split ML system. In this illustration, batch size $B = 3$, the rank of the channel is $r = 3$, the input and output sizes are $N_i = N_o = 6$, and the numbers of antennas at the transmitter and the receiver are $N_t = N_r = 4$.

## *B. The Implementations for Fully Connected NNs*

Since the channel does not change over time, the optimum beamforming matrix should be the same among the transmissions. Hence we can always construct a receiving combining matrix $\mathbf{C} \in \mathcal{C}^{N_r \times r}$ such that matrix $\mathbf{H}^H\mathbf{C}$ is of rank $r$. As a result, the receiver can simultaneously decode $r$ independent data streams, i.e,

$$\mathbf{C}_k = (\mathbf{0}_{N_r \times (k-1)r}, \mathbf{C}, \mathbf{0}_{N_r \times (K-k)r}), k = 1, \cdots, K. \tag{15}$$

According to Lemma 1 and Theorem 1, the $\mathbf{C}_k$ above can satisfy feasible condition (12) with the least number of transmissions $K$ when $N_i \geq N_o$. The procedure above is symmetric between $\mathbf{P}_k$ and $\mathbf{C}_k$ that we can also use the same precoding matrix $\mathbf{P} \in \mathcal{C}^{N_t \times r}$ such that $\mathbf{HP}$ is of rank $r$, and $\mathbf{P}_k = (\mathbf{0}_{N_t \times (k-1)r}, \mathbf{P}, \mathbf{0}_{N_t \times (K-k)r})$ for $k = 1, \cdots, K$ to satisfy feasible condition (12) with the least $K$ when $N_i \leq N_o$. We refer to the former ones as receiver parameterized designs, and the later ones as transmitter parameterized designs, respectively. Based on the symmetry, we can only discuss the receiver parameterized designs in this section.

According to Lemma 1, a straightforward implementation is to directly set all the $\mathbf{P}_k$s to be trainable parameters as shown in Fig. 3. However, the straightforward implementation does not take advantage of the fact that the structure of the precoding beamforming matrices should also be determined by the channel $\mathbf{H}$ to a certain degree. It is more desirable to separate the precoding matrices from the other parameters, i. e.,

$$\mathbf{P}_k = \mathbf{W}_0(\mathbf{0}_{N_r \times (k-1)r}, \mathbf{P}, \mathbf{0}_{N_r \times (K-k)r}), k = 1, \cdots, K, \tag{16}$$

where $\mathbf{P}$ plays a similar role in the precoding matrices comparing to $\mathbf{C}$ in the receiving combining matrices, and $\mathbf{W}_0$ is corresponding to the remaining NN parameters. The system is also illustrated in Fig. 3. In this paper, as shown in Fig. 3, we refer to the former implementation as the combined design and the latter as the separated design, respectively. The detailed comparisons among the implementations are given in Section IV-B.

## C. Problem Formulation

We first analyze the receiver parameterized and separated design for full-connected layers. From equations (6), (15), and (16), the sum of the received signals of the forward transmissions $\boldsymbol{y}$ can be represented by

$$\boldsymbol{y} = \begin{pmatrix} \mathbf{C}^H \mathbf{H} \mathbf{P} \bar{\boldsymbol{x}}_1 + \mathbf{C}^H \boldsymbol{n}_1 \\ \vdots \\ \mathbf{C}^H \mathbf{H} \mathbf{P} \bar{\boldsymbol{x}}_K + \mathbf{C}^H \boldsymbol{n}_K \end{pmatrix}, \tag{17}$$

$$\begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_K \end{pmatrix} = \mathbf{W}_0 \boldsymbol{x}/A, \tag{18}$$

where $A$ denotes the power normalization parameter, i. e., $A = \mathbb{E}\|\mathbf{W}_{0,((k-1)r:kr-1,:)}\boldsymbol{x}\|_2$. Hence the signal-noise ratio (SNR) of the $(k-1)r+t$-th element of $\boldsymbol{y}$ can be represented as

$$\text{SNR}_{f,(k-1)r+t} = \mathbb{E}_{\boldsymbol{x}}(\|\mathbf{C}^H_{(t,:)} \mathbf{H} \mathbf{P} \boldsymbol{s}_k\|_2) N_r / P_n, \tag{19}$$

where $\mathbf{C}^H_{(t,:)}$ denotes the $t$-th row of $\mathbf{C}^H$ and $P_n$ is the total power of noise.

Symmetrically, the SNR of the backward transmission can be written by

$$\text{SNR}_{b,(k-1)r+t} = \mathbb{E}_{\boldsymbol{x}}(\|\mathbf{P}^T_{(t,:)} \mathbf{H}^T \mathbf{C}^* \boldsymbol{g}_{y,k}\|_2) N_t / (A' P_n), \tag{20}$$

where $A'$ is the power normalization parameter of backward communication.

Then, we briefly show the relationship among different implementations of our system. The transmitter parameterized designs are symmetric to the corresponding receiver parameterized designs, and the only difference is that the precoding matrices or the combining matrices are more complicated for combined designs. From the definition of SNR and the structure, $SNR_f$ and $SNR_b$ can be obtained similarly following the procedure above.

Naturally, the goal of MIMO communication is to maximize the SNR of the worst used equivalent subchannel, i. e.,

$$\arg\max_{\mathbf{P},\mathbf{C}} \min\{\min_{k,t} \text{SNR}_{f,(k-1)r+t}, \min_{k,t} \text{SNR}_{b,(k-1)r+t}\}. \tag{21}$$

In the optimal solution to (21), each row of $\mathbf{P}$ and $\mathbf{C}$ is the singular vector corresponding to the largest singular value of channel matrix $\mathbf{H}$, i. e., both transmitter and receiver allocate full power on the optimal beamforming vector. We remark that our system is a joint design of communication and split ML. Hence we should consider the constraints of split ML, i. e., the conditions of Lemma 1 and Theorem 1 hold. Based on this fact, we show how to translate problem (21) to the loss of NN in the following.

## *D. Training Method of the Proposed System*

In this section, we introduce a method to translate (21) to the loss of NN. As explained above, simply considering the goal of (21) leads to a obviously unreasonable solution. However, it gives us inspiration that in the optimal system, only some of the best subchannels should be used. Based on this inspiration, we can find conditions for suboptimal solutions of (21) that

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H, \tag{22}$$

$$\mathbf{P}\mathbf{V}_{:,-r:} = \mathbf{0}, \tag{23}$$

$$\mathbf{C}\mathbf{U}_{:,-r:} = \mathbf{0}, \tag{24}$$

where (22) is the SVD decomposition of $\mathbf{H}$, and $\mathbf{V}_{:,-r:}$ denotes the last $r$ columns of $\mathbf{V}$, i. e., the singular vectors corresponding to the smallest $r$ singular values.

The conditions mainly come from Lemma 1 that $r$ is the minimum value that ensures the conditions of Lemma 1 and the conditions above could be satisfied simultaneously. Under the conditions above, the columns of the matrices $\mathbf{P}_k$ and $\mathbf{C}_k$ are constrained to a linear space of dimension $r$, respectively. According to the basic linear algebra theory, there exist $\mathbf{P}_k$s and $\mathbf{C}_k$s

**Algorithm 1: Training Split ML via OAC**

```
1  {Forward Computation}
2  for $n = 1$ to $N$ do
3      if $n = 1$ then
4          Sample a batch of input data $\boldsymbol{x}$
5          $\boldsymbol{y}_0 \leftarrow \boldsymbol{x}$
6      else
7          Receive signal $\boldsymbol{y}$ by antennas from multiple transmissions
8          Calculate the covariance matrix: $\mathbf{R}_{f,n-1} \leftarrow \mathbb{E}(\boldsymbol{y}\boldsymbol{y}^H)$
9          Update the time averaged covariance matrix: $\bar{\mathbf{R}}_{f,n-1} \leftarrow \alpha\bar{\mathbf{R}}_{f,n-1} + (1-\alpha)\mathbf{R}_{f,n-1}$
10         Obtain $\boldsymbol{y}_{n-1}$ with the combining matrices as shown in Fig. 3
11     Do local forward computation with $\boldsymbol{y}_{n-1}$, and outputs $\boldsymbol{x}_n$ for transmitting
12     if $n = N$ then
13         Output $\boldsymbol{x}_n$ as the final prediction
14     else
15         Transmit $\mathrm{s_n}$ with the precoding matrices as shown in Fig. 3
16 Calculate the loss $\ell$ of the main task.
17 {Backward Propagation}
18 for $n = N$ to $1$ do
19     if $n \neq N$ then
20         Receive signal $\boldsymbol{g}_s$ by antennas from multiple transmissions
21         Calculate the covariance matrix: $\mathbf{R}_{b,n} \leftarrow \mathbb{E}(\boldsymbol{g}_x\boldsymbol{g}_x^H)$
22         Update the time averaged covariance matrix: $\bar{\mathbf{R}}_{b,n} \leftarrow \alpha\bar{\mathbf{R}}_{b,n} + (1-\alpha)\mathbf{R}_{b,n}$
23         Cast SVD decomposition on $\bar{\mathbf{R}}_{b,n}$: $\bar{\mathbf{R}}_{b,n} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$
24         Add the goal of communication: $\boldsymbol{g}_x \leftarrow \boldsymbol{g}_x + (1/2B^2)\partial\|\boldsymbol{x}_n^H\mathbf{V}(:, N_t - r + 1 : N_t)\|_2^2/\partial\boldsymbol{x}$
25         Obtain $\boldsymbol{g}_{x,n}$ with the precoding matrices as shown in Fig. 3
26         Employ backward propagation from $\boldsymbol{g}_{x,n}$, and obtain $\boldsymbol{g}_{y,n-1}$
27     else
28         Employ backward propagation from $\ell$, and obtain $\boldsymbol{g}_{y,n-1}$
29     if $n \neq 1$ then
30         SVD decomposition on $\bar{\mathbf{R}}_{f,n-1}$: $\bar{\mathbf{R}}_{f,n-1} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$
31         Transmit $\mathrm{g_y}$ with the combining matrices as shown in Fig. 3
32         Add the goal of communication: $\boldsymbol{g}_C \leftarrow \boldsymbol{g}_C + \partial\|\mathbf{C}\mathbf{U}(:, N_r - r + 1 : N_r)\|_2^2/\partial\mathbf{C}$
33 Each node updates local parameters based on the calculated gradients.
```

that satisfy both constraints. We note that in the final trained NNs, the ranks $\mathbf{P}_k$ and $\mathbf{C}_k$ may not always be $r$. However, they are brought by the sparsity of NN parameters [25], which does not mean a mistake in our design.

In real systems, we cannot directly obtain $\mathbf{H}$ directly. However, we can still realize the constraints with the help of self-adaptive beamforming. In simple self-adaptive beamforming algorithms, we use the covariance matrix of the received signal to estimate the singular vectors of the channel $\mathbf{H}$, and hence obtain the optimal beamforming vector. Similarly, we can use the singular vectors of the covariance matrix of the received signal to estimate the singular vectors and then compute the loss by (23) and (24). Combining (23), (24), and the training algorithm of centralized NNs, we obtain the detailed training algorithm for a batch as shown in Alg. 1.

## IV. Generalization an Comparison

In this section, we generalize the proposed MIMO OAC-based split ML system to convolutional NNs. We also analyze and compare different implementations for the proposed system.

### A. Extension to Convolutional NNs

For the considered convolutional layer, the size of the input and the output images are $N_{wi} \times N_{hi}$ and $N_{wo} \times N_{ho}$ with $N_{ci}$ and $N_{co}$ channels, respectively. The size of the convolutional kernel is $N_k \times N_k$.

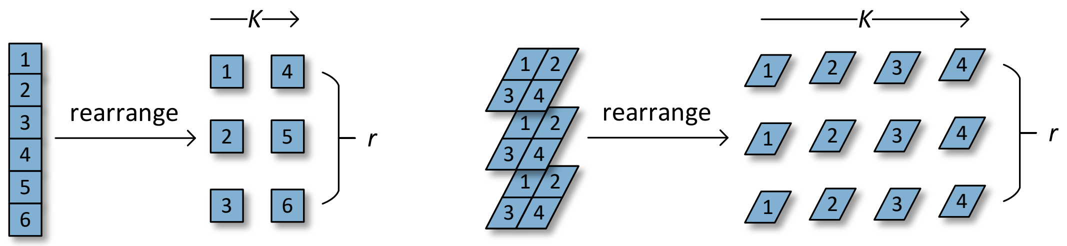

**Fig. 4:** Illustration of the rearrangement in Fig. 3 for both full-connected (left) and convolutional (right) layers.

Here, we show how to extend the implementations to convolutional layers. The calculation in a convolutional layer can be described as:

$$\begin{pmatrix} \mathbf{Y}_1 \\ \vdots \\ \mathbf{Y}_{N_{ci}} \end{pmatrix} = \begin{pmatrix} \mathbf{K}_1 & \cdots & \mathbf{K}_1 \\ \vdots & \ddots & \vdots \\ \mathbf{K}_{N_{co}} & \cdots & \mathbf{K}_{N_{co}} \end{pmatrix} * \begin{pmatrix} \mathbf{X}_1 \\ \vdots \\ \mathbf{X}_{N_{ci}} \end{pmatrix}, \tag{25}$$

where $\mathbf{X}_n$, $\mathbf{Y}_n$, $\mathbf{K}_n$ represent the $n$-th input channel, the $n$-th output channel, and the $n$-th convolutional kernel, respectively. The symbol $*$ represents convolution operation, and its calculation rules between matrices are the same as matrix multiplication, except that convolution replaces multiplication.

Since the convolution operation can be seen as a particular multiplication with specific rules, it is still a linear operation. Hence the associative law still holds. Introducing an arbitrary matrix $\mathbf{W}$, we have

$$\begin{aligned}&\left((\mathbf{W}\otimes\mathbf{I}_{N_k\times N_k})\begin{pmatrix}\mathbf{K}_1 & \cdots & \mathbf{K}_1\\ \vdots & \ddots & \vdots\\ \mathbf{K}_{N_{co}} & \cdots & \mathbf{K}_{N_{co}}\end{pmatrix}\right)*\begin{pmatrix}\mathbf{X}_1\\ \vdots\\ \mathbf{X}_{N_{ci}}\end{pmatrix}\\ =&(\mathbf{W}\otimes\mathbf{I}_{N_{ho}\times N_{ho}})\left(\begin{pmatrix}\mathbf{K}_1 & \cdots & \mathbf{K}_1\\ \vdots & \ddots & \vdots\\ \mathbf{K}_{N_{co}} & \cdots & \mathbf{K}_{N_{co}}\end{pmatrix}*\begin{pmatrix}\mathbf{X}_1\\ \vdots\\ \mathbf{X}_{N_{ci}}\end{pmatrix}\right).\end{aligned} \tag{26}$$

Equation (26) shows that multiplying a matrix $(\mathbf{W}\otimes\mathbf{I}_{N_k\times N_k})$ after convolution is equivalent to multiplying the matrix on the kernels or applying another convolutional layer with $1\times 1$ kernels, i. e.,

$$\begin{pmatrix}\mathbf{K}'_1 & \cdots & \mathbf{K}'_1\\ \vdots & \ddots & \vdots\\ \mathbf{K}'_{N_{co}} & \cdots & \mathbf{K}'_{N_{co}}\end{pmatrix}=(\mathbf{W}\otimes\mathbf{I}_{N_k\times N_k})\begin{pmatrix}\mathbf{K}_1 & \cdots & \mathbf{K}_1\\ \vdots & \ddots & \vdots\\ \mathbf{K}_{N_{co}} & \cdots & \mathbf{K}_{N_{co}}\end{pmatrix}. \tag{27}$$

Hence, for convolution layers, we only need to replace the rearranging method in Fig. 3. The difference between the rearranging methods of full-connected and convolutional layers is shown in Fig. 4. We note that, in Fig. 4, only the intermediate output corresponding an input data is shown, whereas a batch of outputs is transmitted each time in practice.

## *B. Comparisons Among the Implementations*

In this section, we compare all the proposed implementations in terms of the amounts of parameters, calculation, and communication. The amount of parameters is described by the number of complex parameters, and that of communication is described by the number of forward transmissions per batch. The amount of calculation is described by the number of complex multiply-accumulate operations (MACs). A MAC operation is to calculate $a \leftarrow a + b \times c$ by

**TABLE I:** The amounts of parameters, computation, and communication of all designs per batch of data. ("FC" is for fully connected layers, and "Conv" is for convolutional layers. $B$ represents the batch size.)

| | Implementation | Parameters | MACs | Transmissions |
|---|---|---|---|---|
| FC | Transmitter parameterized, combined design | $N_i N_o N_t / r + N_r r$ | $BN_o(N_r + N_i N_t / r)$ | $BN_o / r$ |
| | Transmitter parameterized, separated design | $N_i N_o + (N_t + N_r) r$ | $BN_o(N_i + N_t + N_r)$ | $BN_o / r$ |
| | Receiver parameterized, combined design | $N_i N_o N_r / r + N_t r$ | $BN_i(N_t + N_o N_r / r)$ | $BN_i / r$ |
| | Receiver parameterized, separated design | $N_i N_o + (N_t + N_r) r$ | $BN_i(N_o + N_t + N_r)$ | $BN_i / r$ |
| Conv | Transmitter parameterized, combined design | $N_{co} N_k^2 N_t / r + N_r r$ | $BN_{ci} N_{co} N_{wo} N_{ho} N_k^2 N_t / r$ $+BN_{co} N_{wo} N_{ho} N_r$ | $BN_{co} N_{wo} N_{ho} / r$ |
| | Transmitter parameterized, separated design | $N_{co} N_k^2 + (N_t + N_r) r$ | $BN_{ci} N_{co} N_{wo} N_{ho} N_k^2$ $+BN_{co} N_{wo} N_{ho}(N_t + N_r)$ | $BN_{co} N_{wo} N_{ho} / r$ |
| | Receiver parameterized, combined design | $N_{co} N_k^2 N_r / r + N_t r$ | $BN_{ci} N_{co} N_{wo} N_{ho} N_k^2 N_r / r$ $+BN_{ci} N_{wi} N_{hi} N_t$ | $BN_{ci} N_{wi} N_{hi} / r$ |
| | Receiver parameterized, separated design | $N_{co} N_k^2 + (N_t + N_r) r$ | $BN_{ci} N_{co} N_{wo} N_{ho} N_k^2$ $+BN_{ci} N_{wi} N_{hi}(N_t + N_r)$ | $BN_{ci} N_{wi} N_{hi} / r$ |

a multiplier–accumulator unit, which is widely used in hardware. We show the comparison in Table I.

From Table I, since there is always $r \leq \min\{N_r, N_t\}$ and $r$ may be far smaller than $\min\{N_r, N_t\}$ if the channel is sparse, we can find that for most cases, although the separated designs require one more time of matrix multiplication, it uses fewer parameters and costs less in computation. Whether the transmitter parameterized designs or the receiver parameterized designs are better depends on the relationship between the input and output sizes. If the input size is larger than the output size, the receiver parameterized designs are better, and vice versa.

## V. Numerical Results

In this section, we numerically evaluate the performance of the proposed system, and provide numerical results based on the well-known CIFAR-10 image classification dataset.

### A. Experimental Settings

*1) Communication System Settings:* We consider simple MIMO communication settings, where the channel is composed of $N_p$ different paths, the gain, transmission direction angle

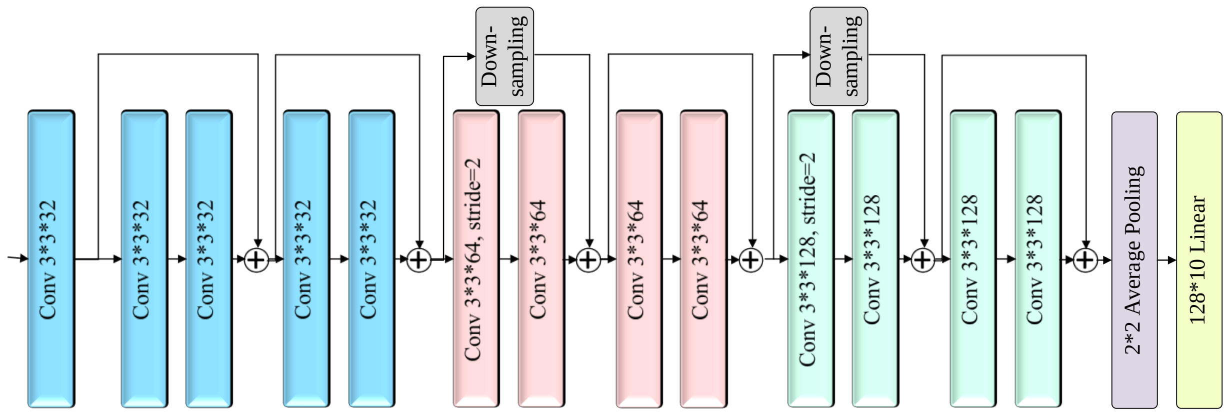

**Fig. 5:** The NN structure used for numerical results.

and arrival direction angle of each path are independently and randomly generated by uniform distribution, i. e.,

$$\mathbf{H} = \sum_{n=1}^{N_p} a_n (1, e^{j\theta_n}, \cdots, e^{j(N_r-1)\theta_n})^H (1, e^{j\phi_n}, \cdots, e^{j(N_t-1)\phi_n}), \tag{28}$$

where $\theta_n \sim U(-\pi, \pi)$, $\phi_n \sim U(-\pi, \pi)$, $|a_n| \sim U(0.5, 1.5)$, and angle$(a_n) \sim U(-\pi, \pi)$ for each $n$.

The total transmitting power of each transmission is normalized to 1, the noise power of each receiving antenna is the same, and the SNR is defined by

$$\text{SNR} = \|\mathbf{H}\|_2 N_r / P_n, \tag{29}$$

where $P_n$ is the total noise power.

We consider the following settings in this section. Firstly, we consider a 2-node complex channel setting, where $N_t = N_r = 16$ and $N_p = 20$ such that the channel is with full rank. We also consider a 3-node sparse channel setting, where $N_t = N_r = 16$ and $N_p = 4$ for both channels. Considering the massive MIMO settings in reality, we consider a massive MIMO setting where there are 3 nodes, and for both channels, $N_t = N_r = 64$ and $N_p = 8$. Finally, we consider a 3-node moving setting where $N_t = N_r = 16$ and $N_p = 4$ for both channels. To ensure fairness, the channels of all settings are pre-generated, and remain the same in all experiments.

*2) NN Settings:* We use the 16-layer complex ResNet to evaluate the performance of our proposed system and algorithms in this section. The NN structure follows that proposed in [26], and is shown in Fig. 5. In Fig. 5, "Conv$3*3*N_c$, stride=2" refers to a convolutional layer with

$N_c$ kernels with size $3 * 3$ and stride 2, and stride is 1 if not written. “Downsampling” refers to cast downsampling such that the outputs of the shortcut connection and the convolutional layers have the same size, which is realized by convolutional layer with $1 * 1$ kernels. “Average pooling” and “Linear” stand for the corresponding layer with the following parameters. Each layer except the pooling layer is followed by a BN layer. For 2-node settings, the NN is split at the 9th layer, and for 3-node settings, the NN is split at the 5th and 13th layer. We use batch size 64 and Adam optimizer [24] with learning rate 0.005 to train the NN, and set $\alpha = 0.99$. Since the output sizes of the convolutional layers are no more than the corresponding input sizes, we use the receiver parameterized designs. According to Table I, we choose separated design for $r < 16$, and combined design for $r = 16$ for the minimum computation and storage cost.

## B. Performance Comparison

We use the straightforward algorithms below to verify the proposed system and algorithm. The first is the centralized algorithm, where the NN is trained in a centralized manner. We also use traditional split ML and MIMO-based split ML, where the communication in split ML is conduct by traditional MIMO. We note that both split ML systems are identical to the centralized algorithm in performance since in such systems, communication is just a tube. The last comparison algorithm is called the ideal case where the channel is assumed to be perfectly known. Hence, from (22)-(24), we can use the following precoding and combining matrix:

$$\mathbf{P}^H = \mathbf{V}_{:,:N_r-r}, \tag{30}$$

$$\mathbf{C}^H = \mathbf{U}_{:,:N_r-r}, \tag{31}$$

where the matrices are calculated by (22). When training NNs, the matrices $\mathbf{P}$ and $\mathbf{C}$ is determined by the channel, and is not trainable in the whole progress.

The costs of all considered algorithms are compared in Table II. In Table II, the smallest value in each column is marked as bold font. We assume that 16 QAM modulation is used in digital systems. Hence a 64-bit complex float number is represented by 16 digital symbols or an analog symbol. Moreover, digital communication systems require channel coding and retransmissions according to channel quality. Thus, analog communication requires no more than 1/16 times of transmission comparing to digital communication. In Table II, $C_{\text{NN}}$, $C_{\text{Trans}}$, and $C_{\text{CE}}$ are defined as the total cost of calculating the NN (in terms of MACs), transmission (in terms of communication rounds), and channel estimation (in terms of times) in traditional

**TABLE II:** Cost comparisons among different algorithms.

| | Computation | Transmission | Channel Estimation |
|---|---|---|---|
| Traditional Split ML | $\boldsymbol{C}_{\text{NN}}$ | $C_{\text{Trans}}$ | $C_{\text{CE}}$ |
| MIMO-Based Split ML | $C_{\text{NN}} + C_{\text{MIMO}}$ | $C_{\text{Trans}}/r$ | $C_{\text{CE}}/r$ |
| Ideal Case | $C_{\text{NN}} + C_{\text{MIMO}}$ | **at most $\boldsymbol{C}_{\textbf{Trans}}/(\mathbf{16}\boldsymbol{r})$** | at most $C_{\text{CE}}/(16r)$ |
| Proposed | $C_{\text{NN}} + C_{\text{MIMO}}$ | **at most $\boldsymbol{C}_{\textbf{Trans}}/(\mathbf{16}\boldsymbol{r})$** | **0** |

split ML, respectively. $C_{\text{MIMO}}$ represents the total native computation cost by the MIMO system (precoding and combining, in terms of MACs) in traditional ML. According to the calculation in Table I, we always have $C_{\text{MIMO}} << C_{\text{NN}}$. We emphasize that our MIMO-based OAC design greatly reduces the transmission cost compared to traditional split ML without OAC. It also saves the system costs and/or improve its overall efficiency by eliminating explicit channel estimation when comparing with the ideal case. According to Table II, the proposed algorithm yields the lowest cost, which reflects high communication and transmission efficiency of the considered MIMO-based OAC structure.

## C. Results Under Stable Complex Channel

In this section, we consider the 2-node complex channel setting. In the setting, the channel is almost of full rank since there are 20 independent randomly generated paths. We suppose the pre-estimation of the rank $r$ to be 4, 8, 12, or 16 to verify the performance of the algorithms.

Fig. 6 shows the convergence trend of various algorithms. In this figure, we consider high quality channel, where SNR = 35 dB and $r = 16$. In Fig. 6, both the ideal case and the proposed algorithm do not show significant difference in terms of convergence rate, but with a slight performance loss, which verifies our analysis that the proposed system implementation is equivalent to the centralized ML system if there is almost no noise.

In Figs. 7 to 9, we present the effect of SNR with different pre-estimation $r$. We can find that the trends of all the curves are similar that the accuracy increases as SNR increasing and obtain almost centralized performance when the channel is of high quality, which is intuitive.

For the same method with different $r$s, although the pre-estimation $r = 16$ is the most accurate, smaller $r$ lead to better performance. This is because when using smaller $r$, smaller data size is transmitted per transmission and fewer subchannels is used, both equivalent to higher SNR.

We note that when $r = 16$, all possible subchannels are used and hence the optimization for communication does not valid. For other values of $r$, we can find that the proposed algorithm performs better with low SNR whereas the ideal case performs better with high SNR. This is mainly because when the channel is well, the best strategy is to simply transmit all the outputs to the receiver, hence the ideal case performs better. However, when the channel becomes poor, the best strategy becomes to stress on transmitting only a part of the outputs to obtain a higher quality. This strategy can make sense mainly because the output of NNs is usually sparse [25], and the performance loss caused by the reduce of width is smaller than that by the increase of the noise on the intermediate output.

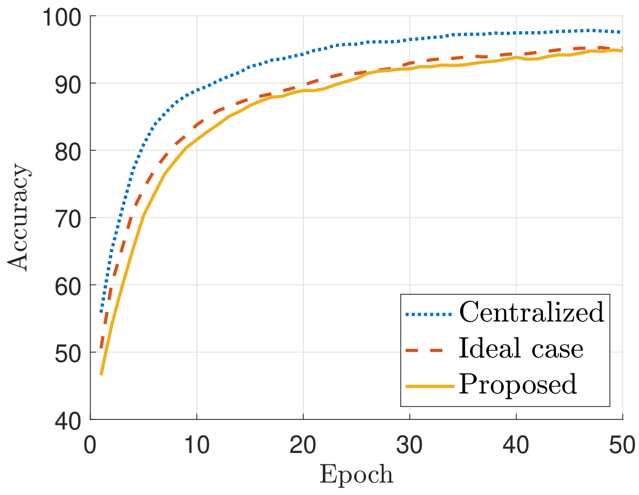

**Fig. 6:** The convergence curve of different algorithms.

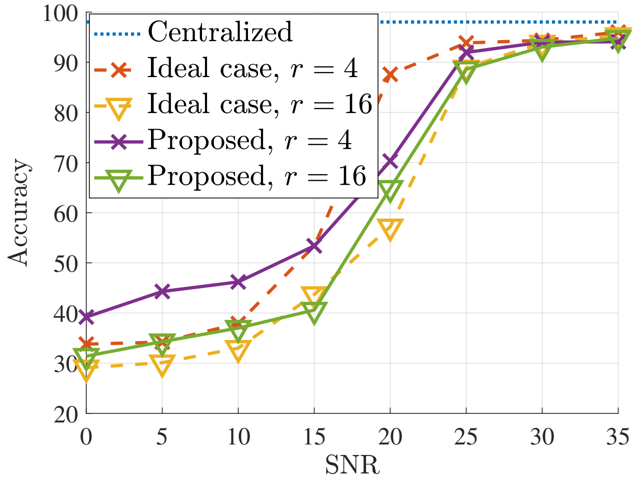

**Fig. 7:** The performance of different algorithms with different $r$s under stable complex channel.

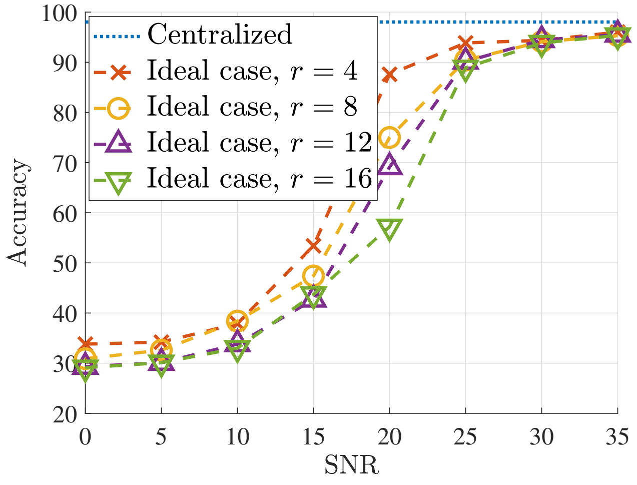

**Fig. 8:** The performance the ideal case with different $r$s under stable complex channel.

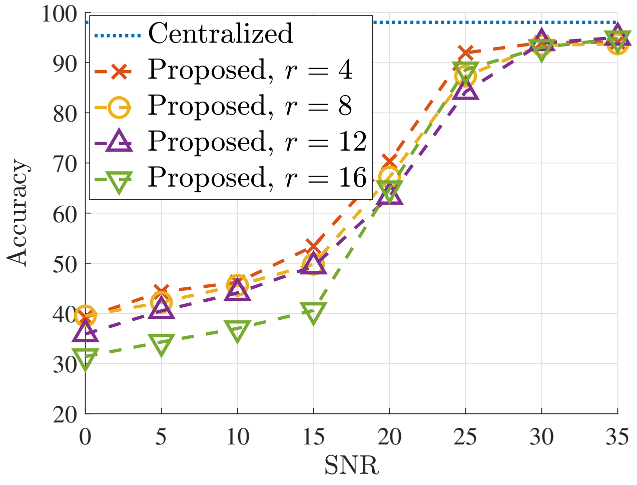

**Fig. 9:** The performance the proposed algorithm with different $r$s under stable complex channel.

## *D. Results Under Stable Sparse Channels*

In this section, we consider the 3-node sparse channel setting. In the setting, the channel is of rank 4. We suppose the pre-estimation of the rank $r$ to be 4 or 8 to verify the performance of the algorithms. In Fig. 10, we show the performance of different algorithms and different pre-estimations of $r$. The results and conclusions are similar with those of complex channel, however, it is counterintuitive that the performance when $r = 8$ does not decline sharply although it does not satisfy Theorem 1. This phenomenon is mainly caused by the sparsity of NNs [25]. All the preconditions are derived by the suppose that the parameters $\mathbf{W}$ can take any values in $\mathcal{C}^{N_i \times N_o}$, whereas $\mathbf{W}$ is usually sparse in reality, which may explain the small performance loss.

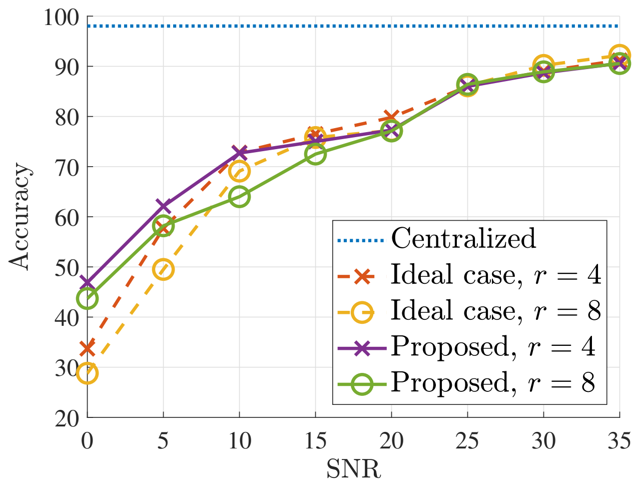

**Fig. 10:** The performance of different algorithms with different $r$s under stable sparse channels.

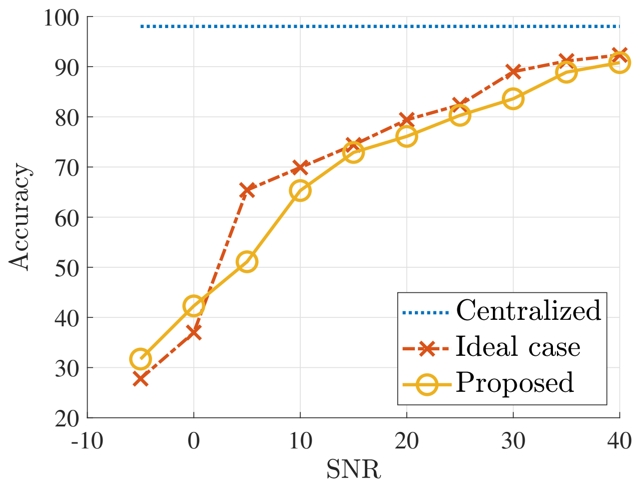

**Fig. 11:** The performance of different algorithms under stable massive MIMO channels.

## *E. Results Under Stable Massive MIMO Channels*

We show the results under stable massive MIMO channels with $r = 8$ in Fig. 11, which does not show apparent discrimination comparing to Fig. 10, showing the effectiveness of our proposed system in massive MIMO scenarios. However, in Fig. 11, the proposed algorithm cannot outperform the ideal case, which is mainly caused by that the proposed algorithm brings much error when the dimension becomes larger.

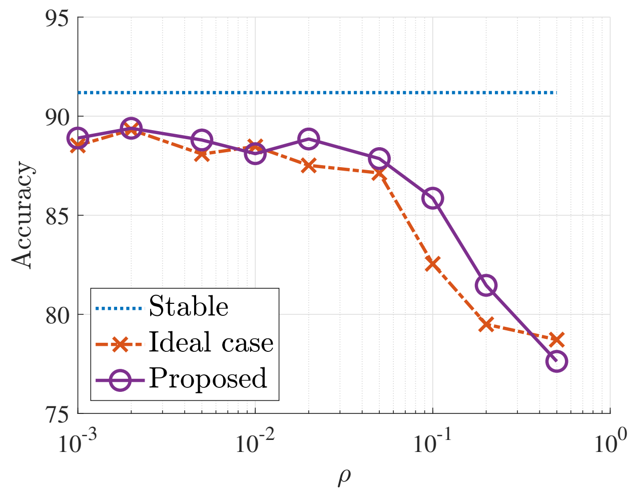

**Fig. 12:** The performance of different algorithms under moving MIMO channels.

## *F. Results Under Time-Varying Channels*

We show the results under dynamic channel settings here. In this setting, both the channel matrices vary according to the following memory model:

$$a_{n,t+1} = (1-\rho)a_{n,t} + \rho\hat{a}_n, \tag{32}$$

$$\theta_{n,t+1} = (1-\rho)\theta_{n,t} + \rho\hat{\theta}_n, \tag{33}$$

$$\phi_{n,t+1} = (1-\rho)\phi_{n,t} + \rho\hat{\phi}_n, \tag{34}$$

where $\mathbf{a}_n$, $\theta_n$, and $\phi_n$ are the value of the corresponding values in (28) at $t$-th time slot, respectively. $\hat{a}_n$, $\hat{\theta}_n$, and $\hat{\phi}_n$ are generated randomly following the way in (28), and $\rho$ indicates the changing rate. The channel $\mathbf{H}_t$ at each time slot is also generated by (28). In Fig. 12, we can find that both the ideal case and the proposed algorithm perform well even under moving channels. In the ideal case, both the precoding and the combining matrices suddenly changes when the channel changes, while the proposed algorithm can learn to adapt the change of channel. Hence, the proposed algorithm outperforms the ideal case with smooth channel change, while performs poor when the channel changes fiercely because the changing can not be learnt well.

## VI. Conclusions and Future Directions

In this paper, we proposed a MIMO-based OAC system for implementing split ML over wireless networks, which significantly improves communication efficiency. In the proposed system, the precoder and combiner design, together with the implicit MIMO channel matrix, contribute to a trainable layer, which can be natively integrated into a NN. We theoretically found the inherit equivalent procedure between the reciprocity of MIMO channels and the training process of NNs. Based on this finding, explicit channel estimation process is eliminated in our system, which can improve the system efficiency. Besides, we provided basic principles of implementing the proposed system such that the proposed structure can be mathematically equivalent to any fully connected or convolutional layer in NNs. Numerical results show that the proposed system has a similar converge rate with slight performance degradation under high-quality channels compared to the centralized algorithm. We also observed that our algorithm could work well under various channels and is robust to slowly varying channels.

In this paper, we considered channel reciprocity and the extension to natural channels without strict reciprocity remains to be a future direction. The proposed MIMO OAC-based split ML structure can play an essential role in native distributed learning problems for wireless communications such as distributed sensing, channel estimation and prediction, location estimation and prediction, distributed network optimization, interference coordination, etc.

## APPENDIX A
## PROOF OF THEOREM 2

*Proof.* To prove Theorem 2, we introduce the regret to analyze the performance of the learning algorithm [27]. In particular, the regret is defined as

$$R(T) = \sum_{t=1}^{T} f_t\left(\boldsymbol{\theta}^{(t)}\right) - \sum_{t=1}^{T} f_t(\boldsymbol{\theta}^*), \tag{35}$$

where $\boldsymbol{\theta}^*$ is the optimal parameter set. Since function $f_t(\boldsymbol{\theta})$ is convex, we have

$$f_t(\boldsymbol{\theta}^{(t)}) - f_t(\boldsymbol{\theta}^*) \leq \langle \boldsymbol{g}_t, \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* \rangle, \tag{36}$$

which shows that

$$R(T) \leq \sum_{t=1}^{T} \langle \boldsymbol{g}_t, \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* \rangle. \tag{37}$$

According the update procedure of SGD with noise, we have

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t(\boldsymbol{g}_t + \boldsymbol{\Lambda}_t \boldsymbol{n}_t), \tag{38}$$

where $\boldsymbol{\Lambda}_t$ is a non-negative diagnose matrix, each element of which is no more than $\xi$, and $\boldsymbol{n}_t \sim \mathcal{N}(\mathbf{0}^{N*1}, \sigma \boldsymbol{I}^{N*N})$ is the noise. Hence, we have

$$\begin{aligned}
&\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2^2 \\
=&\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* - \eta_t(\boldsymbol{g}_t + \boldsymbol{\Lambda_t n}_t)\|_2^2 \\
=&\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 - 2\eta_t\langle \boldsymbol{g}_t, \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\rangle + \eta_t^2\|\boldsymbol{g}_t\|_2^2 \\
&+ \eta_t^2|\boldsymbol{\Lambda}_t|^2\|\boldsymbol{n}_t\|_2^2 - 2\eta_t\langle \boldsymbol{\Lambda}_t\boldsymbol{n}_t, \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\rangle + 2\eta_t\langle \boldsymbol{\Lambda_t n}_t, \boldsymbol{g}_t\rangle.
\end{aligned} \tag{39}$$

According to (39), we can obtain

$$\begin{aligned}
\langle \boldsymbol{g}_t, \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\rangle =&\frac{1}{2\eta_t}(\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 - \|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2^2) \\
&+ \frac{\eta_t}{2}|\boldsymbol{\Lambda}_t|^2\|\boldsymbol{n}_t\|_2^2 + \frac{\eta_t}{2}\|\boldsymbol{g}_t\|_2^2 \\
&- \langle \boldsymbol{\Lambda}_t\boldsymbol{n}_t, \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\rangle + \eta_t\langle \boldsymbol{\Lambda_t n_t}, \boldsymbol{g_t}\rangle \\
\le&\frac{1}{2\eta_t}(\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 - \|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2^2) \\
&+ \frac{\eta_t\xi}{2}\|\boldsymbol{n}_t\|_2^2 + \frac{\eta_t G^2}{2} - \langle \boldsymbol{\Lambda}_t\boldsymbol{n}_t, \boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\rangle,
\end{aligned} \tag{40}$$

where the last inequality comes from the fact that $\|\boldsymbol{g}_t\|_2^2 \le G$. Since there are always many parameters in a neural network, i. e., $N$ is very large, we can approximately get $\|\boldsymbol{n}_t\|_2^2 \approx N\sigma^2$ based on the law of large numbers. Substituting (40) to (37), we have

$$\begin{aligned}
R(T) \le&\sum_{t=1}^{T}\frac{1}{2\eta_t}(\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 - \|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2^2) \\
&+ \frac{\xi\|\boldsymbol{n}_t\|_2^2}{2}\sum_{t=1}^{T}\eta_t - \sum_{t=1}^{T}\langle \boldsymbol{\Lambda}_t\boldsymbol{n}_t, \boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\rangle.
\end{aligned} \tag{41}$$

Due to the fact that the parameters are bounded and learning rate is decreasing over time, we have

$$\begin{aligned}
&\sum_{t=1}^{T}\frac{1}{2\eta_t}(\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 - \|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2^2) \\
=&\frac{1}{2\eta_1}\|\boldsymbol{\theta}^{(1)} - \boldsymbol{\theta}^*\|_2^2 - \frac{1}{2\eta_T}\|\boldsymbol{\theta}^{(T+1)} - \boldsymbol{\theta}^*\|_2^2 \\
&+ \sum_{t=2}^{T}(\frac{1}{2\eta_t} - \frac{1}{2\eta_{t-1}})\|\boldsymbol{\theta}^{(1)} - \boldsymbol{\theta}^*\|_2^2 \\
\le&\frac{1}{2\eta_1}D^2 + \sum_{t=2}^{T}(\frac{1}{2\eta_t} - \frac{1}{2\eta_{t-1}})D^2 \\
=&\frac{D^2}{2\eta_T}.
\end{aligned} \tag{42}$$

Since $\boldsymbol{n}_t$ follows a standard Gaussian distribution, we can obtain expectation $\mathbb{E}\{1/T\sum_{t=1}^{T}\langle\boldsymbol{\Lambda}_t\boldsymbol{n}_t,\boldsymbol{\theta}^{(t)}-\boldsymbol{\theta}^*\rangle\}=0$ and standard deviation $\sigma\{1/T\sum_{t=1}^{T}\langle\boldsymbol{\Lambda}_t\boldsymbol{n}_t,\boldsymbol{\theta}^{(t)}-\boldsymbol{\theta}^*\rangle\}=\mathcal{O}(\sigma T^{-1/2})$.

When we choose $\eta_t=C/t^{1/2}$, we have

$$\begin{aligned} R(T)/T \leq & \frac{D^2}{2CT^{1/2}}+\frac{C(\xi\|\boldsymbol{n}_t\|_2^2+G^2)}{2T}\sum_{t=1}^{T}t^{-1/2} \\ & -\frac{1}{T}\sum_{t=1}^{T}\langle\boldsymbol{\Lambda}_t\boldsymbol{n}_t,\boldsymbol{\theta}^{(t)}-\boldsymbol{\theta}^*\rangle. \end{aligned} \tag{43}$$

In the right hand of (43), the first item is of $\mathcal{O}(T^{-1/2})$, the second item is approximately by $\mathcal{O}(\sigma^2T^{-1/2})$, and the third item is a random item with zero expectation and standard deviation of $\mathcal{O}(\sigma T^{-1/2})$. Hence, the convergence rate of the algorithm is $\mathcal{O}(\sigma^2T^{-1/2})$. Comparing (43) with the regret of SGD provided in [27] that

$$R(T)/T \leq \frac{D^2}{2CT^{1/2}}+\frac{CG^2}{T}\sum_{t=1}^{T}t^{-1/2}, \tag{44}$$

we can easily observe that noise power has a linear effect on the convergence rate of SGD with noise. Theorem 2 is proved. □

## References

[1] Y. Xiao, G. Shi, Y. Li, *et al.*, "Toward self-learning edge intelligence in 6G," *IEEE Communications Magazine*, vol. 58, no. 12, pp. 34–40, Dec. 2020.

[2] M. Jankowski, D. Gündüz, and K. Mikolajczyk, "Wireless image retrieval at the edge," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 89–100, Jan. 2021.

[3] S. Dörner, S. Cammerer, J. Hoydis, *et al.*, "Deep learning based communication over the air," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 132–143, Feb. 2018.

[4] M. Goldenbaum and S. Stanczak, "Robust analog function computation via wireless multiple-access channels," *IEEE Transactions on Communications*, vol. 61, no. 9, pp. 3863–3877, Sep. 2013.

[5] K. Yang, T. Jiang, Y. Shi, *et al.*, "Federated learning via over-the-air computation," *IEEE Transactions on Wireless Communications*, vol. 19, no. 3, pp. 2022–2035, Mar. 2020.

[6] M. Gastpar, "Uncoded transmission is exactly optimal for a simple gaussian "sensor" network," *IEEE Transactions on Information Theory*, vol. 54, no. 11, pp. 5247–5251, Nov. 2008.

[7] T. S. Nazaré, G. B. P. da Costa, W. A. Contato, *et al.*, "Deep convolutional neural networks and noisy images," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications (CIARP)*, Madrid, Spain, Nov. 2018, pp. 416–424.

[8] G. Zhu, Y. Wang, and K. Huang, "Broadband analog aggregation for low-latency federated edge learning," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 491–506, Jan. 2020.

[9] Y. Shao, D. Gündüz, and S. C. Liew, “Federated edge learning with misaligned over-the-air computation,” *arXiv preprint arXiv:2102.13604*, Feb. 2021.

[10] G. Zhu and K. Huang, “MIMO over-the-air computation for high-mobility multimodal sensing,” *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6089–6103, Aug. 2019.

[11] T. W. Hughes, I. A. D. Williamson, M. Minkov, *et al.*, “Wave physics as an analog recurrent neural network,” *Science advances*, vol. 5, no. 12, eaay6946, Dec. 2019.

[12] X. Sui, Q. Wu, J. Liu, *et al.*, “A review of optical neural networks,” *IEEE Access*, vol. 8, pp. 70 773–70 783, 2020.

[13] K. V. M. Ahmet M. Elbir, “A survey of deep learning architectures for intelligent reflecting surfaces,” *arXiv preprint arXiv:2009.02540*, Sep. 2020.

[14] S. G. Sanchez, G. R. Muns, C. Bocanegra, *et al.*, “AirNN: Neural networks with over-the-air convolution via reconfigurable intelligent surfaces,” *arXiv preprint arXiv:2202.03399*, Feb. 2022.

[15] Y. Kang, J. Hauswald, C. Gao, *et al.*, “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge,” in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, vol. 45, Xi’an, China, Apr. 2017, pp. 615–629.

[16] C. Hu, W. Bao, D. Wang, *et al.*, “Dynamic adaptive dnn surgery for inference acceleration on the edge,” in *IEEE Conference on Computer Communications (INFOCOM)*, Paris, France, 2019, pp. 1423–1431.

[17] S. Wang, X. Zhang, H. Uchiyama, *et al.*, “Hivemind: Towards cellular native machine learning model splitting,” *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 2, pp. 626–640, Feb. 2022.

[18] A. E. Eshratifar, A. Esmaili, and M. Pedram, “Bottlenet: A deep learning architecture for intelligent mobile cloud computing services,” in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Lausanne, Switzerland, Jul. 2019, pp. 1–6.

[19] J. Shao and J. Zhang, “Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems,” in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, Dublin, Ireland, Jun. 2020, pp. 1–6.

[20] H. Li, C. Hu, J. Jiang, *et al.*, “JALAD: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution,” in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, Singapore, Dec. 2018, pp. 671–678.

[21] Y. Tian, Z. Zhang, Z. Yang, *et al.*, “JMSNAS: Joint model split and neural architecture search for learning over mobile edge networks,” in *International Communications Conference (ICC) Workshop*, 2022 (accepted to appear).

[22] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, Feb. 2015.

[23] H. E. Robbins, “A stochastic approximation method,” *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.

[24] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations(ICLR)*, San Diego, CA, USA, May 2015.

[25] T. Hoefler, D. Alistarh, T. Ben-Nun, *et al.*, “Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks,” *arXiv preprint arXiv:2102.00554*, Feb. 2021.

[26] C. Trabelsi, O. Bilaniuk, Y. Zhang, *et al.*, “Deep complex networks,” in *International Conference on Learning Representations(ICLR)*, Vancouver, BC, Canada, May 2018.

[27] M. Zinkevich, “Online convex programming and generalized infinitesimal gradient ascent,” in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML)*, Washington, DC, USA, 2003, pp. 928–935.