PhyDNNs: Bringing Deep Neural Networks to the Physical Layer

Mohammad Abdi*, Khandaker Foysal Haque*, Francesca Meneghello†, Jonathan Ashdown‡, and Francesco Restuccia*

*Northeastern University, United States, †University of Padova, Italy, ‡Air Force Research Laboratory, United States

IEEE INFOCOM 2025 - IEEE Conference on Computer Communications | 979-8-3315-4305-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/INFOCOM55648.2025.11044671

***Abstract*—Emerging applications require mobile devices to continuously execute complex deep neural networks (DNNs). While mobile edge computing (MEC) may reduce the computation burden of mobile devices, it exhibits excessive latency as it relies on encapsulating and decapsulating frames through the network protocol stack. To address this issue, we propose PhyDNNs, an approach where DNNs *are modified* to operate directly at the physical layer (PHY), thus significantly decreasing latency, energy consumption, and network overhead. Conversely from recent work in Joint Source and Channel Coding (JSCC), PhyDNNs *adapt already trained DNNs to work at the PHY*. To this end, we developed a novel information-theoretical framework to fine-tune PhyDNNs based on the trade-off between communication efficiency and task performance. We have prototyped PhyDNNs with an experimental testbed using a Jetson Orin Nano as the mobile device and two USRP software-defined radios (SDRs) for wireless communication. We evaluated PhyDNNs performance considering various channel conditions, DNN models, and datasets. We also tested PhyDNNs on the Colosseum network emulator considering two different propagation scenarios. Experimental results show that PhyDNNs can reduce the end-to-end inference latency, amount of transmitted data, and power consumption by up to 48×, 1385×, and 13× while keeping the accuracy within 7% of the state-of-the-art approaches. Moreover, we show that PhyDNNs experience 4.3 times less latency than the most recent JSCC method while incurring in only 1.79% performance loss. For replicability, we shared the source code for the PhyDNNs implementation.**

***Index Terms*—Semantic Communication, Edge Computing, Task-Oriented Communication, Distributed Neural Networks, Device-Edge Collaborative Inference.**

I. Introduction and Motivation

Emerging mobile applications require mobile devices to continuously perform complex computer vision (CV) tasks using deep neural networks (DNNs). For example, object detection [1] and semantic segmentation [2] are needed to include physical movements into digital avatars in virtual reality (VR) headsets, as well as to superimpose virtual objects in augmented reality (AR) smart glasses [3].

Existing issues. While mobile edge computing (MEC) can decrease the computational load on mobile devices [4], there are still several fundamental limitations that prevent the use of MEC strategies for AR/VR applications. The first key issue is that the end-to-end edge offloading latency – which includes the round-trip device-edge network latency plus the DNN execution time – has to be extremely low to avoid VR-related motion sickness. Specifically, it has been shown that CV tasks have to be executed with end-to-end inference latency at least equal to the frame rate, which is 8 milliseconds at 120 Hz [5]. *However, existing wireless networks are not able to consistently deliver such required latency*. For example, measurements from Suer et al. have shown that Wi-Fi 6 (IEEE 802.11ax) can exhibit up to 60 ms latency – an order of magnitude greater than what is required in AR/VR systems [6]. Moreover, in [7], the authors unveil the limitations of fifth-generation (5G) in fulfilling AR/VR requirements: considering a system bandwidth of 100 MHz, 5G new radio can serve only a single user per base station while guaranteeing the above-mentioned latency requirement. The second critical issue is that *continuously offloading high-resolution images/video frames may saturate the network capacity*. For example, 100 mobile devices connected to the same 5G cellular network forwarding 8K frames at 120 Hz would create more than 8 Gbps of traffic [8], whereas 5G has a maximum capacity of 1 Gbps in the mid-bands [9]. Therefore, how to achieve AR/VR-level latency while preserving DNN performance is still an open problem in mobile edge computing.

Approved for Public Release; Distribution Unlimited; AFRL-2024-4883.

Limitations of existing work. Distributed inference approaches – discussed in detail in Section II – try to reduce transmission latency by dividing the DNN into *head* and *tail* DNNs, executed by the mobile device and the edge server, respectively. The head DNN is trained to produce compressed *latent features* carrying only the task-relevant information, which are then transferred to the edge server using Wi-Fi or cellular networks and given as input to the tail DNN, which produces the task result (see upper part of Fig. 1).

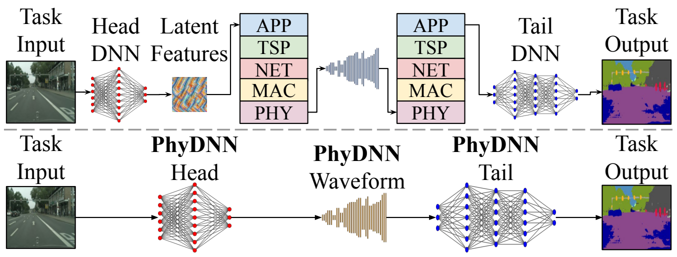


Fig. 1: (top) Distributed Inference; (bottom) Physical-Layer DNNs (in short, PhyDNNs).

The key issue is that the DNNs operate at the application layer (APP), which requires encapsulating and decapsulating the latent features through the transport (TSP), network (NET), medium access control (MAC) and physical layer (PHY). However, distributed inference is usually implemented in networks where mobile devices have direct links to the edge server, making higher-level networking functionalities unnecessary. Hence, as shown in Section IV-C, the full-stack approach causes excessive communication overhead and excessive latency (i.e., 251.15 ms for a batch of 64 inputs).

Proposed approach. To address these fundamental limitations, we propose *Physical-Layer DNNs* (PhyDNNs), a new approach to MEC where a pre-trained DNN is *modified* so that it can be executed directly *at the waveform level*. More in detail, as shown at the bottom of Fig. 1, the output of the head DNN and the input of the tail DNN are encoded PHY waveforms. By bringing the DNN to the physical layer, PhyDNNs avoid the overhead required to encapsulate and decapsulate the latent representations through the protocol stack. **In other words, PhyDNNs implement a task-oriented semantic communication link between the transmitter and the receiver**. Importantly, PhyDNNs adapt an already-trained DNNs to produce PHY waveforms. This makes PhyDNNs fundamentally different from prior work in Joint Source and Channel Coding (JSCC), which is discussed in detail in Section II. In short, JSCC approaches *design custom DNNs and train them from scratch* to jointly learn to perform the task and channel coding. This is not feasible when considering modern DNNs that are trained on large datasets. In addition, as discussed in details in Section IV-C, the most recent JSCC work [10] incurs in latency and energy consumption that are respectively 4.3 and 7.3 times higher than PhyDNNs.

Summary of Novel Contributions:

• We propose PhyDNNs, a new approach to MEC where **an already-trained DNN is fine-tuned to operate directly at the PHY**, thus learning to perform channel coding on top of its main inference task. As such, PhyDNNs achieve a significant decrease in latency by avoiding the encapsulation/decapsulation through the network protocol stack. In stark opposition with JSCC approaches, the DNN *is not* specifically designed for PHY operation. By integrating the channel model into the fine-tuning process, the modified DNN robustifies the transmitted waveform against the corresponding distortion;

• We develop a new training procedure that **optimizes the trade-off between communication efficiency and final task performance**, formulated using the Information Bottleneck (IB) theory [11]. To overcome the computational intractability of mutual information terms in such formulation, an upper bound is derived using variational inference approximation. During training, we leverage the sparsity induced in the embedding layer to further compress the latent features. This way, PhyDNNs learn to transmit less data while guaranteeing high task performance;

• We have prototyped PhyDNNs using USRP software-defined radios (SDRs) and extensively evaluated their performance in a wide variety of wireless propagation environments against other MEC approaches. To benchmark the proposed approach, we have considered three DNNs (ResNet20, ResNet56, ResNet110) pre-trained on the CIFAR-10 and CIFAR-100 datasets. Apart from task performance, we reported the end-to-end inference latency, communication overhead, and power consumption of PhyDNNs by implementing the head DNN on a Jetson Nano Orin. PhyDNNs have also been evaluated on the Colosseum network emulator and through simulations. The state-of-the-art distributed inference approaches [12], [13] are used as comparison baselines;

• The experimental results show that PhyDNNs are able to decrease the total inference time, amount of transmitted data, and energy consumption of the mobile device by up to 48×, 1385×, and 13× while experiencing only up to 7% performance loss with respect to [13]. We also show that our approach achieves 4.3 times less latency than the most recent JSCC method while incurring in only 1.79% performance loss. **To the best of our knowledge, this work is the first to demonstrate the feasibility of task-oriented semantic communications through DNNs operating at the waveform level using a real-world experimental testbed.**

II. Related Work

Distributed Inference. A sizable body of prior work focuses on finding suitable partitioning points for a DNN architecture based on the relative processing power of the mobile device and edge server [14], [15]. Since in most DNNs the representation size increases in the first layers for feature extraction purposes [16], compression must be done at the splitting point [13]. Previous work achieves this goal using a process called "bottleneck injection" [17]–[19]. In such a process, feature compression is achieved within the backbone model without adding a separate compression block. The authors in [20] use simple end task loss such as cross-entropy loss for an image classification task to retrain the bottleneck-injected model. To compensate for the reduction in DNN representational capacity, more recent work utilizes Knowledge Distillation (KD), which can partially recover the accuracy drop [21]. In [22], the authors adopt a two-stage training strategy using different loss terms to recover from the drop in task performance. While these methods mostly take heuristic approaches to reduce the width of the splitting layer, most recent work uses the IB theory to formulate a rate-distortion minimization problem. Matsubara et al. [23] learn a prior distribution for the intermediate representation, and minimize its entropy during training. An entropy coding scheme is then used to reduce the amount of data to be sent depending on its entropy. Overall, the key limitation of current distributed inference systems is that even though they achieve relatively high compression rates, the entire wireless protocol stack is used to transfer the latent features to the edge server. As shown in Section IV-C when using [22], this results in the communication delay taking up the major part (88.93 %) of the total inference latency, increasing it to 251.15 ms for a batch of 64 images. Instead, our approach can reduce the end-to-end delay to 42.11 ms with only 7% drop in performance.

Joint Source and Channel Coding. Recently, learning-based JSCC was proposed for image [24], [25] and text transmission [26]. However, these methods originally aimed to reconstruct the input message in their output mainly using autoencoders (AEs) with a noise-injection layer to simulate the channel. Some recent work has proposed task-oriented JSCC [27] by replacing the AE output with inference result. However, most of the existing JSCC schemes assume the channel noise is directly summed up with each element of the floating-point encoded tensor [28], [29], which makes them incompatible with digital communication systems, which involve modulation and demodulation.

Recently, Xie et al. [10] addressed this issue by proposing a task-oriented JSCC model with discrete output, called DT-JSCC. However, the work in [10] presents three key issues that make it inapplicable in real-world wireless systems. First, in line with all JSCC approaches, it requires to design and train a custom-tailored DNN from scratch, thus not benefiting from the state-of-the-art designs. **On the contrary, our proposed PhyDNNs do not require the DNN to be specifically designed to work at the physical layer**. Conversely, we take DNNs designed and trained for a particular task at the application layer and bring them to the physical layer by only fine-tuning their parameters. Second, DT-JSCC does not account for the resource constraints of mobile devices. Specifically, the encoder consists of 4.36 million parameters (fixed) in DT-JSCC, which is more than 100 times larger than the PhyDNNs head, i.e., 42.5 thousand parameters. As such, its execution on the mobile device takes around $7\times$ more time and consumes about $8\times$ more energy than the PhyDNNs head. Third, DT-JSCC focuses on maximizing the redundancy in the latent space, which leads to more communication overhead. Moreover, [10] uses a fixed-size quantizer, which results in the transmission of a fixed amount of data irrespective of the optimized entropy level. In stark opposition, **PhyDNNs optimize the latent dimension based on the accuracy-communication efficiency trade-off**. Specifically, we show in Section IV-C that to perform image classification on a batch of 64 images, DT-JSCC transmits 41.7 kilobytes, while PhyDNNs require only 16 KB. Overall, the end-to-end latency, accounting for both computing and communication, is 182.53 ms for DT-JSCC while it is 42.11 ms for PhyDNNs. Moreover, all the prior work on JSCC has only been tested through simulations. **To the best of our knowledge, this is the first work to demonstrate the concept of physical-layer DNNs with real-world testbed experiments.**

III. The PhyDNNs Framework

The main idea behind PhyDNNs is that a pre-trained DNN can be implemented at the physical layer and distributed between different nodes by adding an embedding layer plus a modulator block at the transmitter (e.g., mobile device), and a demodulator combined with an embedding lookup layer at the receiver (e.g., edge server). Hence, the PhyDNNs framework consists of the following steps: (i) the pre-trained DNN is divided into two parts, namely head $\mathcal{H}$ and tail $\mathcal{T}$, based on the relative processing power of the mobile device and edge server through, e.g., [14]; (ii) at the mobile device, the embedding layer and the modulator map the head DNN output (floating-point tensor) into a modulated waveform that is transmitted through the wireless channel; (iii) at the edge server, the demodulator and the embedding lookup layer map the received waveform back into a floating-point tensor that is processed by the tail DNN to compute the task output. The whole framework is trained in an end-to-end fashion adding a non-trainable layer to model the wireless channel. In this way,

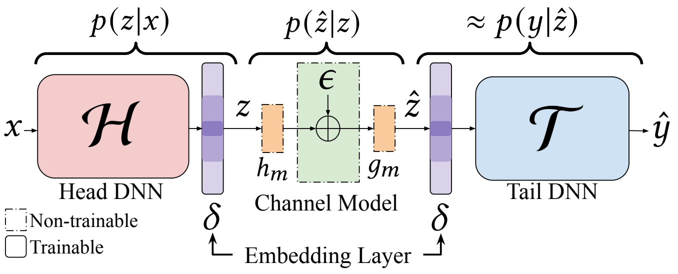


Fig. 2: PhyDNN Probabilistic System Model.

the DNN is fine-tuned to learn waveforms which are robust to channel distortions. Note that when the pre-trained weights are not available, the backbone DNN can also be trained from scratch using the same approach without additional steps.

A. Probabilistic System Model and Assumptions

Henceforth, we adopt the following notation. Random variables and their realizations are indicated by upper-case and lower-case letters, respectively. $E\{X\}$ and $H(X)$ denote the statistical expectation and entropy of the variable X. The mutual information between X and Z is shown by $I(X;Z)$, while $H(Z|X)$ represents the conditional entropy of Z given X. We use $D_{KL}(p(x)||q(x))$ to denote the Kullback–Leibler (KL) divergence between $p(x)$ and $q(x)$ probability distributions. The circularly-symmetric complex Gaussian distribution with mean μ and covariance matrix $\mathbf{\Sigma}$ is shown by $\mathcal{N}_{\mathcal{C}}(\mu, \mathbf{\Sigma})$.

The probabilistic model for designing and training the PhyDNNs distributed system is depicted in Fig. 2. The DNN task input at the mobile device is denoted as x having a corresponding target y. Such data is assumed to be generated from a joint distribution $p(x, y)$. Let z be the N_s discrete latent features containing only the necessary information to predict y using x. The extraction of such features (z) from x is modeled by sampling from a categorical conditional distribution $p(z|x)$. The head DNN combined with the embedding layer provide the parameters of such distribution. To reduce training computational complexity, we adopt the mean-field assumption [30] that the conditional distributions for different dimensions of z are independent. Therefore, $p(z|x)$ can be written as

$$p(z|x) = \prod_{i=1}^{N_s} p(z_i|x) = \prod_{i=1}^{N_s} \mathcal{CAT}(f_i^{\phi}(x)). \qquad (1)$$

where f^{ϕ} is the function defined by the head plus the embedding with adjustable parameters ϕ. More in detail, the embedding layer projects the activation at the output of the head DNN to a categorical distribution over the latent features using a learnable dictionary δ of M embeddings, where M is the order of the adopted modulation scheme. Each element of z, sampled from $p(z|x)$, is a M-ary symbol, which is then modulated following the digital modulation scheme h_m.

The obtained modulated signal is then sent to the edge server through the wireless channel. In our formulation, we consider an Additive White Gaussian Noise (AWGN) model for the channel. Despite this choice, PhyDNNs can be readily extended to any arbitrary channel model based on which PhyDNNs will learn to perform Forward Error Correction

(FEC) implicitly. The additive noise vector ϵ elements are sampled from a zero-mean complex Gaussian distribution with variance σ^2, i.e., $\epsilon \sim \mathcal{N}_{\mathcal{C}}(\mathbf{0}, \sigma^2\mathbf{I})$. Considering the limited power of the transmitter on the mobile device, we constrain the peak power of the modulated signal to be P, resulting in a Signal-to-Noise-Ratio (SNR) value of $10\log_{10}\frac{P}{\sigma^2}$.

At the receiver side (edge server), the distorted signal is demodulated using g_m. The output of the demodulator is indicated by $\hat{z}$. The digital modulator h_m, channel, and demodulator g_m are modeled as non-trainable yet differentiable layers in the end-to-end training of PhyDNNs. The equivalent conditional distribution modeling these layers is $p(\hat{z}|z)$. The corrupted latent $\hat{z}$ is then projected onto floating-point feature maps through an embedding lookup layer which uses the same learned embedding dictionary δ as the one on the mobile device. The recovered feature maps are fed to the tail DNN $\mathcal{T}$ that computes the task result $\hat{y}$. The combination of the embedding lookup layer and the tail DNN defines the function $g^{\theta}(\hat{z})$ where θ is the set of trainable parameters. From a probabilistic viewpoint, the inference at the receiver side is modeled as sampling from the conditional distribution $p(y|\hat{z})$.

B. PhyDNNs Design

Next, we provide the implementation details of the embedding layers and the modulator and demodulator blocks, together with the channel model used during PhyDNNs training.

Embedding and Embedding Lookup Layer. Fig. 3 shows an overview of the embedding layer design. Denoting the deterministic head output as $\mathcal{H}(x)$, the value $f_i^{\phi}(x)$ in Eq. (1) is obtained by projecting each slice $\mathcal{H}_i(x)$ of the head output feature maps onto different embeddings as follows:

$$f_i^{\phi}(x) = \text{softmax}(\delta^T \cdot \mathcal{H}_i(x)), \tag{2}$$

where $\delta = [e_1, e_2, \cdots, e_M]$ is the embedding dictionary consisting of M *learnable* embeddings. From Eq. (1) and Eq. (2), each element of the embedded vector is obtained as the inner product similarity between a slice and different embeddings, converted into probabilities by the softmax function.

The embedding lookup layer at the edge server transforms the received symbols back into floating-point feature maps. Therefore, the embedding dictionary needs to be shared between the mobile device and edge server and is a part of both parameter sets ϕ and θ. Such parameters are jointly trained in an end-to-end fashion through backpropagation.

A key challenge is that the main inference task performed by the DNN and the channel coding task are different in nature, thus interfering with each other. To mitigate this while jointly tuning the embedding layer along with the already-trained DNN, we first cluster the head DNN output slices by feeding the whole training dataset to the head DNN. The cluster centroids are then taken to be the initial embeddings. Additionally, the original DNN weights are frozen at the beginning of training and, when unfrozen after a few epochs, they are trained using a lower learning rate compared to the embedding parameters. To further prevent the distributed DNN from forgetting its main task, a loss term is also added to force

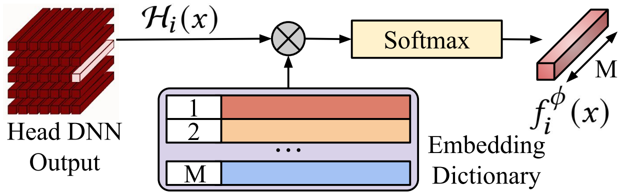


Fig. 3: PhyDNN embedding layer design.

the DNN excluding the added embedding layers to mimic the output of the original pre-trained DNN (Section III-C).

Modulator, Demodulator and Channel Model. For the sake of generality, in this paper we used Phase-shift Keying (PSK) modulation and AWGN channel model. However, PhyDNNs can easily extend to more advanced modulation schemes and realistic channel models. PhyDNNs assign the embeddings to different constellation points based on their relative distances. Specifically, closer embeddings are associated with adjacent points in the constellation to improve the modulation performance. The modulator, channel model, and demodulator are included as non-trainable layers during the end-to-end training so that channel-robust features are learned. Intuitively, PhyDNNs learn to perform task-oriented FEC on corrupted representations to get the same task results. *Notice that, except for the synchronization sequence for packet detection, PhyDNNs introduce no additional communication protocol overhead.*

C. PhyDNNs Loss Function Formulation and Training

The loss function for PhyDNNs training is obtained by accounting for the trade-off between the communication efficiency at the mobile device and the task inference performance at the edge server. Intuitively, by compressing the latent features to lower the communication load and latency, the feature maps that the edge server receives would be less informative thus resulting in degraded task performance. On the other hand, if better performance is required, more symbols should be transmitted by the mobile device to ensure robustness against channel distortions. From an information-theoretic viewpoint, PhyDNNs' goal is to learn a discrete latent Z which is *maximally compressive* about X while being *maximally predictive* about Y. Such compression and prediction abilities translate into communication efficiency and end-task performance in our problem. Hence, by considering the IB principle [11], the PhyDNNs loss function is written as

$$\begin{aligned}\mathcal{L}_{IB} &= -I(\hat{Z};Y) + \beta \cdot I(X;\hat{Z}) \\ &= \mathbb{E}_{p(x,y)}\Big\{\underbrace{\mathbb{E}_{p(\hat{z}|x)}\left[-\log p(y|\hat{z})\right]}_{\text{Task Performance}} \\ &\quad + \beta \cdot \underbrace{D_{KL}\big(p(\hat{z}|x)||p(\hat{z})\big)}_{\text{Communication Efficiency}}\Big\} - H(Y),\end{aligned} \tag{3}$$

where $0 \leq \beta \leq 1$ is the weight controlling the efficiency-performance trade-off. By minimizing this loss during training PhyDNNs, we can find the optimal Z which captures only the minimal sufficient statistics of X for expressing Y. Note that $H(Y)$ can be removed from the formulation since it is a constant value during minimization.

Variational Upper Bound Reformulation. The conditional distribution $p(\hat{z}|x)$ in Eq. (3) is obtained as $p(\hat{z}|x) = \int p(z|x)p(\hat{z}|z)dz$. Hence, given $p(\hat{z}|x)$ and the joint data distribution $p(x,y)$, the distributions $p(\hat{z})$ and $p(y|\hat{z})$ in Eq. (3) are fully determined by the following integrals:

$$p(\hat{z}) = \iint p(x,y)p(\hat{z}|x)dxdy, \tag{4}$$

$$p(y|\hat{z}) = \int \frac{p(x,y)p(\hat{z}|x)}{p(\hat{z})}dx. \tag{5}$$

However, computing such integrals for high dimensional data with arbitrary distributions is computationally demanding. To overcome this issue, we leverage the Variational Information Bottleneck (VIB) theory [31] by defining two variational distributions $q(\hat{z})$ and $q(y|\hat{z})$ to approximate $p(\hat{z})$ and $p(y|\hat{z})$. Specifically, we take the stochastic DNN consisting of the embedding dictionary lookup layer and tail DNN $\mathcal{T}$ with parameters θ – defining the function $g^{\theta}(\hat{z})$ and deployed on the edge server – to be the variational approximation of $p(y|\hat{z})$. The type of distribution to be approximated depends on the task. For example, for a classification task, $q(y|\hat{z})$ is a categorical distribution, i.e., $q(y|\hat{z}) = \mathcal{CAT}(g^{\theta}(\hat{z}))$. Using this approximation, the variational upper bound of the task performance – the first term in Eq. (3) – is obtained as

$$\begin{aligned}&\mathbb{E}_{p(x,y)}\Big\{\mathbb{E}_{p(\hat{z}|x)}\big[-\log p(y|\hat{z})\big]\Big\} = \\ &\mathbb{E}_{p(x,y)}\Big\{\mathbb{E}_{p(\hat{z}|x)}\big[-\log q(y|\hat{z})\big]\Big\} - \underbrace{\mathbb{E}_{p(\hat{z})}\Big\{\mathbb{E}_{p(y|\hat{z})}\big[\log\frac{p(y|\hat{z})}{q(y|\hat{z})}\big]\Big\}}_{D_{KL}\left(p(y|\hat{z})||q(y|\hat{z})\right)\geq 0} \\ &\qquad \leq \mathbb{E}_{p(x,y)}\Big\{\mathbb{E}_{p(\hat{z}|x)}\big[-\log q(y|\hat{z})\big]\Big\}.\end{aligned}$$

Similarly, we compute the upper bound for the communication efficiency – the second term in Eq. (3) – as

$$\begin{aligned}&D_{KL}\big(p(\hat{z}|x)||p(\hat{z})\big) = \\ &\qquad D_{KL}\big(p(\hat{z}|x)||q(\hat{z})\big) - \underbrace{D_{KL}\big(p(\hat{z})||q(\hat{z})\big)}_{\geq 0} \\ &\qquad\qquad \leq D_{KL}\big(p(\hat{z}|x)||q(\hat{z})\big).\end{aligned}$$

Since minimizing this upper bound minimizes the KL divergence between $p(\hat{z}|x)$ and $q(\hat{z})$, a certain variational prior $q(\hat{z})$ can be used to induce that probability distribution on the latent representation. Therefore, to sparsify the latent features during training, we replace this second upper bound with $\mathbb{E}_{p(z|x)}\big[H(z|x)\big]$. This operation is equivalent to reducing the conditional entropy of latent which is similar in concept to minimizing the rate in existing work [32]–[34].

Putting everything together, the variational upper bound of the loss function in Eq. (3) can be written as follows:

$$\begin{aligned}\mathcal{L}_{VIB} = \mathbb{E}_{p(x,y)}\Big\{&\mathbb{E}_{p(\hat{z}|x)}\big[-\log q(y|\hat{z})\big] \\ &+ \beta\cdot\mathbb{E}_{p(z|x)}\big[H(z|x)\big]\Big\}.\end{aligned} \tag{6}$$

Using the mean-field assumption presented in Eq. (1), the conditional entropy term $H(z|x)$ can be decomposed and computed as $H(z|x) = \sum_{i=1}^{N_s} H(z_i|x)$.

Algorithm 1: Training PhyDNNs

```
   Data: $N_e$ (number of epochs),
1  while epoch $e = 1$ to $N_e$ do
2      Sample a mini-batch of data $(x_i, y_i)_{i=1}^{N_{mb}}$;
3      while $k = 1$ to $N_{mb}$ do
4          while $i = 1$ to $N_s$ do
5              Compute conditional distribution
                 $p(z_{k,i}|x_k) = \mathcal{CAT}(f_i^{\phi}(x_k))$;
6              Sample $N_{mc}$ realizations of latent symbol
                 $\{z_{k,j,i}\}_{j=1}^{N_{mc}} \sim p(z_{k,i}|x_k)$ using Gumbel trick;
7              Compute conditional entropy $H(z_{k,j,i}|x_k)$;
8              Compute the second term in Eq. (7);
9              Sample realizations of channel noise
                 $\{\epsilon_{k,j,i}\}_{j=1}^{N_{mc}} \sim \mathcal{N}(0,\sigma^2)$;
10             Estimate corrupted symbol $\hat{z}_{k,j,i}$;
11         Concatenate $[\hat{z}_{k,j,i}]_{i=1}^{N_s}$ to obtain $\hat{z}_{k,j}$;
12         Obtain $\hat{y}_k$ by feeding to DNN;
13         Compute the first term in Eq. (7);
14     Accumulate the total loss in Eq. (8);
15     Update parameters $\phi$ and $\theta$ through backpropagation;
16     Increase $\beta$ in Eq. (7);
17     Remove embeddings that are least used.
```

Computing the Loss Function. An unbiased estimate of the loss function in Eq. (6) can be obtained using Monte Carlo sampling and used for training PhyDNNs using the gradient descent algorithm, as summarized in Algorithm 1 and explained below. First, by feeding x to the head DNN, the parameters of the categorical distribution over each latent feature z_i are obtained at the output of the embedding layer (line 5 of Algorithm 1). Then, a realization of z is sampled from these distributions and fed to the modulation layer h_m. Since such sampling is not differentiable, we use the Gumbel-softmax trick [35] to make the probabilistic model differentiable and trainable using loss backpropagation. Using this trick, given a mini-batch of N_{mb} data points $\{(x_k, y_k)\}_{k=1}^{N_{mb}}$ (line 2), by sampling N_{mc} realizations of z (line 6) and N_{mc} realizations of the physical channel noise ϵ (line 9) for each pair (x_k, y_k), we have the following Monte Carlo estimation of $\mathcal{L}_{VIB}$ in Eq. (6) (line 13)

$$\begin{aligned}\tilde{\mathcal{L}}_{VIB} = &\frac{1}{N_{mb}}\sum_{k=1}^{N_{mb}}\Big\{\frac{1}{N_{mc}}\sum_{j=1}^{N_{mc}}\big[-\log q(y_k|\hat{z}_{k,j}) \\ &+ \beta\cdot\sum_{i=1}^{N_s} H(z_{k,j,i}|x_k)\big]\Big\},\end{aligned} \tag{7}$$

where $z_{k,j,i} \sim \mathcal{CAT}(f_i^{\phi}(x_k))$, $\epsilon_{k,j,i} \sim \mathcal{N}(0,\sigma^2)$, and $\hat{z}_{k,j,i} = g_m\big(h_m(z_{k,j,i}) + \epsilon_{k,j,i}\big)$.

Learning the embedding using the loss function in Eq. (7) is challenging as it interferes with the pre-trained DNN weights. To prevent the pre-trained DNN from forgetting its main inference task, we add another loss term calculated by excluding the two embedding layers plus the channel layer from the PhyDNNs computation graph. From a high-level perspective, the two embedding layers together with the channel noise injection can be viewed as a variational encoder-decoder pair. In theory, one can have more than one set of such layers added to the backbone DNN in cases where the DNN needs to be divided into several parts to be distributed against more than two devices. Therefore, each set of variational encoder-decoder apart from its main output passes the input through the output. This way, the DNN has an auxiliary output corresponding to the original architecture (consisting of the head and tail DNNs only) but using the updated weights in each training iteration. Adding this binding loss term, the final loss function $\mathcal{L}_{tot}$ writes as:

$$\mathcal{L}_{tot} = \tilde{\mathcal{L}}_{VIB} + \mathcal{L}_{bind}, \quad \text{where} \tag{8}$$

$$\mathcal{L}_{bind} = \frac{1}{N_{mb}} \sum_{i=1}^{N_{mb}} -\log \mathcal{T}_{y_i}\big(\mathcal{H}(x_i)\big). \tag{9}$$

Latent Feature Compression. While minimizing the second term in Eq. (7) during training PhyDNNs reduces the latent conditional entropy, it does not reduce the amount of transmitted data by itself. To exploit such minimized entropy, existing work in neural image compression uses entropy coding [33], [34], [36] to generate variable-length codes with the expected length approaching the entropy. However, entropy coding is not differentiable and cannot be utilized in our distributed inference system. As such, we propose to compress the embedding layer as a substitute to entropy coding the latent representation. Intuitively, reducing the entropy of the latent features leads to a gradual sharpening of their distribution. On the other hand, embeddings are intended to capture the principal components of latent features. Therefore, entropy reduction in itself makes it easier for the embeddings to represent the latent distribution. Based on this, our approach is to gradually increase β in Eq. (7) during training to reduce the latent entropy progressively (line 16 of Algorithm 1). As a result, not only the embedding error is minimized but also the latent distribution can be learned using fewer embeddings. We then leverage such induced sparsity in the embedding space to remove the least-used embeddings iteratively resulting in a compressed embedding layer (line 17). On the other hand, progressive growth in sparsity enables gradual embedding dictionary compression avoiding the irreversible damage to the DNN performance that could happen when performing an abrupt embedding compression. By fine-tuning for a few epochs after each removal step, PhyDNNs trade off task performance for shorter waveforms.

IV. Experimental Evaluation

We tested the proposed PhyDNNs approach through extensive evaluations with (i) the experimental testbed described in Section IV-A deployed in two different environments and considering both line of sight (LoS) and Non-Line-of-Sight (NLoS) transmissions; (ii) the Colosseum wireless network emulator [37] considering the urban scenario with a path loss of 40 dB and 80 dB; and (iii) end-to-end simulations to evaluate the robustness of PhyDNNs performance considering a wider range of SNR levels, i.e., SNR $\in [4, 20]$ dB.

We use image classification for benchmarking, which is one of the widely considered CV tasks, utilizing the publicly-available CIFAR-10 and CIFAR-100 datasets. We considered ResNet20, ResNet56, and ResNet110 [38] as the backbone DNN models, which are brought to the physical layer by PhyDNNs. These architectures have progressively higher inference accuracy at the cost of more computation and memory. We first assess the PhyDNNs performance by analyzing several relevant metrics. Next, we demonstrate the robustness of PhyDNNs task inference to channel variations through both experiments and simulations. Finally, we explore the inherent trade-off between the channel-encoded latent feature size and inference performance.

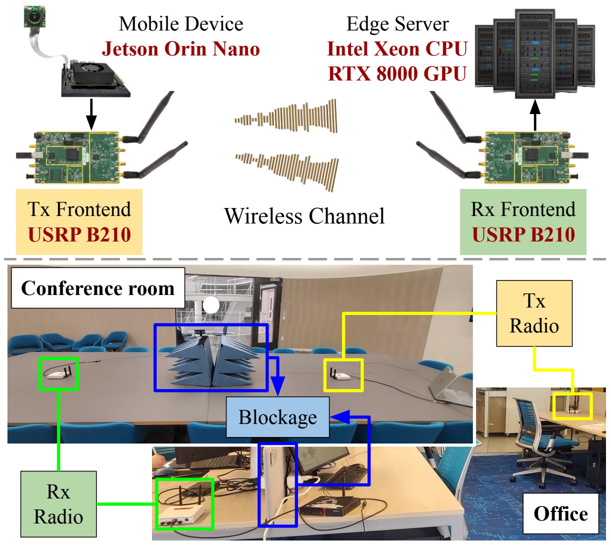


Fig. 4: Experimental setup for performance evaluation in the conference room and office. The radio-absorbing cones and the desk divider are used to create the blockage for NLoS.

A. *Experimental Testbed*

We set up an end-to-end MEC testbed leveraging two Ettus USRP B210, operating at 2.4 GHz with 20 MHz of bandwidth, for the transmitter (Tx) and receiver (Rx) radios, as depicted at the top of Fig. 4. We used a Jetson Orin Nano – powered by a 1024-core NVIDIA Ampere GPU and a 6-core Arm Cortex-A78AE CPU – as the mobile device. The edge server was implemented by a Linux-based machine, equipped with an Intel(R) Xeon(R) Gold 6246R CPU, and a Quadro RTX 8000 GPU. We conducted the experiments in an office environment and a conference room in both LoS and NLoS, as illustrated in Fig. 4. In the office, the Tx-Rx distance was 3.6 m for LoS and 7.8 m for NLoS. In the conference room, the distance was fixed to 7 m for both LoS and NLoS.

B. *Performance Benchmarks*

The selected full-stack and physical-layer MEC approaches used as the baseline for comparison are presented below, together with the transmission process adopted for these two classes.

Full-Stack MEC: The considered baselines in this class leverage full-stack IEEE 802.11a Wi-Fi for offloading the representations. The latent features generated by the head model for each batch of images are segmented based on the maximum transmission unit (MTU) size suitable for TCP transmissions and encapsulated all the way to the physical layer before transmission. We used TCP as the transmission protocol to be in line with the existing distributed inference implementations.

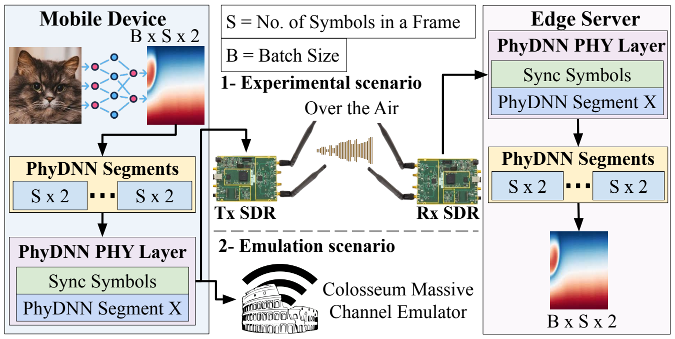


Fig. 5: Transmission procedure of PhyDNNs waveforms.

Using UDP makes the task execution unreliable as the edge server drops the corrupted representations, not providing any results. We consider the following approaches:

• **Split Computing (SC):** The head DNN consists of the first layer of backbone ResNet only (as in the seminal work [12]).

• **BottleFit (BF):** To reduce the amount of transferred data, a compressed intermediate representation is learned. According to the two-stage algorithm in [22], we reduce the width of the last convolutional layer in the first ResNet layer and utilize KD to compensate for the accuracy drop. We use two versions of BF having a reduced width of 1 and 2 kernels.

Physical-Layer MEC: The head DNN directly outputs channel-encoded symbols for each batch of images. As shown in Fig. 5, the encoded symbols are divided into segments, where each segment is then encapsulated with a PHY header for packet detection and synchronization, and transmitted to the edge server. In addition to PhyDNNs, we consider DT-JSCC for comparison as detailed below. Note that we do not consider retransmissions of corrupted latent features as both approaches are designed to compensate for channel impairments and the edge server is always able to return a task result given a latent signal.

• **DT-JSCC:** This method has been proposed in [10] and represents the most recent task-oriented JSCC work. As discussed in Section II, the authors introduce custom architectures for two different datasets (MNIST and CIFAR-10). Hence, we selected the DT-JSCC architecture designed for CIFAR-10 for our comparison. The encoder consists of 4 convolutional layers and 2 residual blocks, which outputs 512 feature maps in the latent space. A quantizer is then used to discretize the latent for transmission. As DT-JSCC has not been implemented on real hardware in [10], we used the same PHY transmission procedure designed for PhyDNNs for a fair comparison.

• **PhyDNNs (our):** We set the length of each embedding to match the number of feature maps at the output of the head DNN. The learning rate of the embedding layer parameters δ is 10^{-3}, while it is set to 10^{-4} for $\{\phi, \theta\}$ parameters. We used 12 dB for the channel SNR when training PhyDNNs. We start by setting the trade-off parameter $\beta = 10^{-4}$ in Eq. (7) and increase it linearly during the fine-tuning process with an overall increase of $10\times$ over all the epochs.

C. Experimental Results

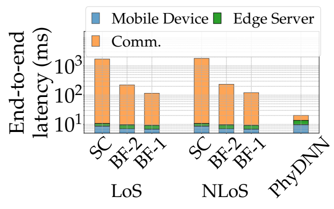
Our assessment focuses on the following performance metrics: (i) end-to-end latency, (ii) communication overhead, (iii) energy consumption, and (iv) performance of the inference task. In all of our experiments, we consistently used an input batch size of 64 to report the results. Note that, as PhyDNNs do not have retransmission mechanism, the results are obtained as the average of the LoS and NLoS scenarios.

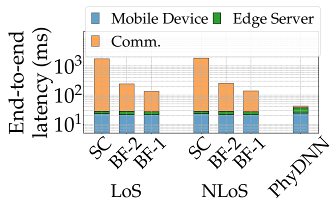
End-to-end Latency. We measure the total inference latency as the execution time of head and tail DNNs – at the mobile device and edge server respectively – plus the communication delay, where the latter mainly depends on the transmitted representation size and communication overhead. Fig. 6a, 6b, and 6c present the end-to-end latency of PhyDNNs compared to the reference full-stack approaches obtained through experimental evaluation in the office environment. The results for both LoS and NLoS show that SC has the highest latency, while PhyDNNs reach the lowest value for all three ResNet models. On top of that, PhyDNNs outperform BF-1 and BF-2 by 2.3 and 3.8 times respectively (averaged across the LoS and NLoS scenarios, and the three models). This is while the PhyDNNs head execution time on the mobile device takes only around 1 millisecond more than other distributed approaches.

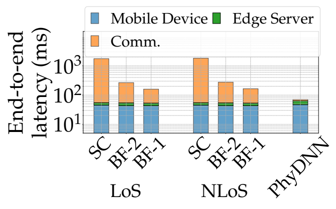
The end-to-end latency breakdown shows that the communication delay – being the largest contribution – of PhyDNNs is 16.75 and 17.56 times lower (on average across the models) than BF-1 for the LoS and NLoS scenarios, respectively. Indeed, PhyDNNs do not add overhead due to the protocol stack and directly feeds the channel-encoded waveforms to the PHY. The communication latency of baseline approaches increases by 4.8% on average in NLoS compared to LoS due to the packet loss and retransmission. Instead, PhyDNNs does not employ retransmission as it operates at the PHY and compensates for channel impairments.

The end-to-end latency of different approaches when evaluated in the conference room and in the Colosseum network emulator are depicted in Fig. 7a and 8a and are in line with the above findings, proving the robustness of PhyDNNs to different wireless environments.

Communication Overhead. Fig. 6d shows the amount of data transmitted (payload and protocol overhead) by various approaches under both LoS and NLoS conditions. The results indicate that for a batch size of 64, SC sends the highest amount of data, i.e., 4096 KB, whereas PhyDNNs require the least, i.e., 16 KB. Remarkably, the payload data transmitted by PhyDNNs is 16 and 32 times smaller than that of BF-1 and BF-2, respectively. This is because, while BF-1 and BF-2 transfer compressed representations, PhyDNNs directly transmits channel-encoded waveforms, which require a substantially lower amount of data. Fig. 6d shows that PhyDNNs advantage becomes even more in the NLoS scenario. PhyDNNs only add 256 bytes of control data, reducing the overall data overhead by $180\times$ and $363\times$ in the LoS scenario and by $237\times$ and $484\times$ in the NLoS scenario, compared to BF-1 and BF-2, respectively. The increase in the control data transmission due to TCP and MAC layer retransmissions is linked with the degradation of the channel quality.


(a) Latency – ResNet20


(b) Latency – ResNet56


(c) Latency – ResNet101

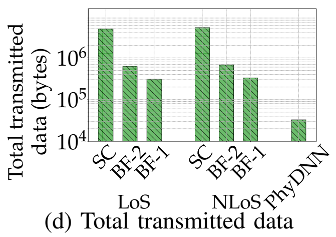
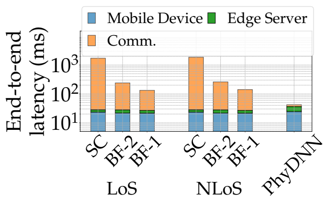

(d) Total transmitted data

Fig. 6: End-to-end latency and total transmitted data (payload and overhead combined) for both LoS and NLoS in the office environment. A batch size of 64 is considered. As PhyDNNs do not account for retransmissions, the results are measured as the average of the LoS and NLoS scenarios.


(a) Latency – ResNet56

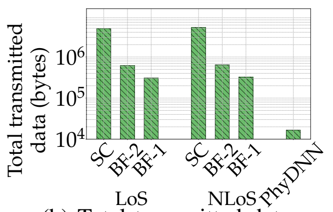
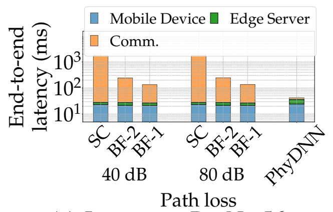

(b) Total transmitted data

Fig. 7: Evaluation in the conference room with 7 m distance between the transmitter and the receiver, in LoS and NLoS.


(a) Latency – ResNet56

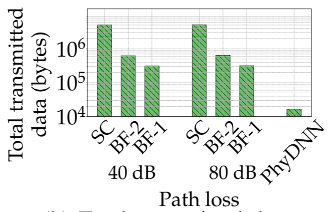

(b) Total transmitted data

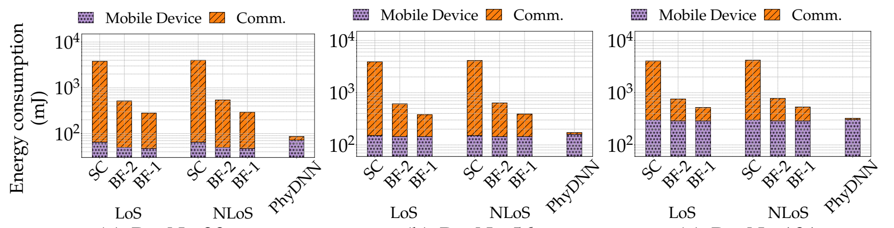
Fig. 8: Evaluation in Colosseum using the urban emulation scenario considering a path loss of 40 dB and 80 dB.

The same trend is visible in the results of the evaluation in the conference room and with the Colosseum emulator, reported in Fig. 7b and 8b. The only distinction is that the conference room experiences higher difference between LoS and NLoS results due to the ideal blockage of the direct path signal using the absorbing cones.

Energy Consumption. The primary factors contributing to this include head DNN execution at the mobile device and transmission of latent representations. We compare the energy consumption of different approaches in both LoS and NLoS conditions in Fig. 9a, 9b, and 9c. The results show that energy consumed for communication is the primary contributor to the total energy consumption of mobile device. However, it is more dominant with lightweight DNNs like ResNet20 in comparison to the more complex ones like ResNet110 due to their shorter execution time. PhyDNNs in comparison to BF-1 and BF-2, improve the overall energy consumption by 135.35% and 295.53% in LoS scenario and 142.89% and 312.41% in NLoS scenario on average across different DNNs. For different architectures, the head DNN execution energy of BF-1 and PhyDNNs varies only by 20.6 mJ which is only 5.16% of the total energy consumption of BF-1 on average. This proves the importance of transmission energy minimization to improve the overall energy consumption. Compared to BF-1, PhyDNNs reduce the energy consumption by 18.52 and 19.42 times in LoS and NLoS scenarios, respectively.

Inference Performance Robustness Against Changing Communication Environments. In this section, we evaluate the task inference accuracy (at the edge server) of PhyDNNs considering various propagation environments. We first carried out a systematic performance evaluation through simulations by changing the SNR in the range from 4 dB to 20 dB. Fig. 10 illustrates the classification error rate when using the trained PhyDNNs in different channel conditions for CIFAR-10 and CIFAR-100 datasets. While an AWGN channel model with a SNR of 12 dB is used for PhyDNNs training (see Section IV-B), we observe that PhyDNNs perform well also when tested in other channel conditions. For example, Fig. 10 shows that in the case of CIFAR-10 (left plot), the change in the task error rate is less than 3% and 5% for ResNet20 and ResNet110, respectively. This proves that PhyDNNs effectively extract robust features that guarantee consistent performance across varying channel conditions. Robustness to channel variations has also been evaluated experimentally in the office environment for the LoS and NLoS scenarios in Table I. The results show that the accuracy of ResNet56 and ResNet110 drops only by around 8% and 7%, respectively in the LoS scenario compared to the baseline approaches. We remind that, by relying on TCP, the full-stack approaches do not experience any accuracy drop when the DNN is executed in a distributed way. In the NLoS scenario, PhyDNNs face only about 4% more performance loss for both ResNet56 and ResNet110 than the LoS case. Moreover, the experimental accuracy is only around 5% less than the simulation-based results. Overall, this proves the PhyDNNs practicality in real-life environments.

Performance-Communication Trade-off. The trade-off between the communication efficiency and task performance is controlled by the parameter β in Eq. (7). Hence, β controls the number of transmitted kilobytes. To evaluate the impact of this parameter, Table II shows the final β value and its associated task accuracy achieved by PhyDNNs applied to different ResNet models using CIFAR-10 and CIFAR-100. The results are obtained through simulations using a SNR of 12 dB and show that when β is increased from 10^{-5} to 10^{-3}, more emphasis is put on the entropy term in Eq. (7) resulting in fewer symbols (shorter waveform) to be sent at the cost of reducing the inference accuracy. However, PhyDNNs only experience an accuracy loss of up to 3% when sending 16 KB of latent data instead of 32 KB. This can be attributed to the over-parameterization redundancy present in most state-



(a) ResNet20 (b) ResNet56 (c) ResNet101

Fig. 9: Energy consumption for LoS and NLoS scenarios with ResNet architectures.

TABLE I: Experimental task accuracy of PhyDNNs using CIFAR-10. The baseline refers to all the full-stack approaches.

	ResNet-20	ResNet-56	ResNet-110
LOS	78.23 %	82.34 %	84.85 %
NLOS	74.64 %	78.51 %	80.74 %
Baseline	87.52%	90.6%	92.26%

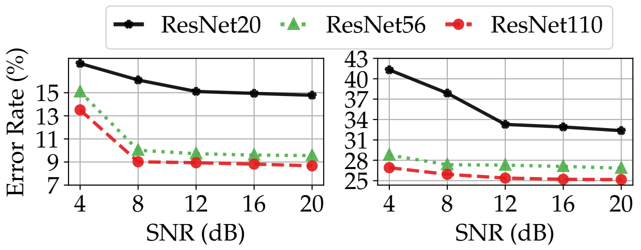


Fig. 10: Simulation-based task error rate when varying the SNR using (left) CIFAR-10 and (right) CIFAR-100 datasets.

TABLE II: Simulation-based trade-off between transmitted data and inference accuracy for CIFAR-10 and CIFAR-100.

CIFAR-10	32 KB $\beta = 10^{-5}$	24 KB $\beta = 10^{-4}$	16 KB $\beta = 10^{-3}$
ResNet-20	82.89 %	81.48 %	80.07 %
ResNet-56	88.59 %	87.29 %	85.9 %
ResNet-110	89.27 %	88.69 %	87.81 %

CIFAR-100	32 KB $\beta = 10^{-5}$	24 KB $\beta = 10^{-4}$	16 KB $\beta = 10^{-3}$
ResNet-20	67.72 %	66.6 %	65.75 %
ResNet-56	72.72 %	71.39 %	70.6 %
ResNet-110	74.71 %	74.04 %	73.88 %

of-the-art capacity-rich models. Such redundancy is effectively leveraged by PhyDNNs to compensate for channel losses even when sending 4 times less data to the edge server.

Comparison with DT-JSCC. In Table III, we showcased relevant metrics for comparison between PhyDNNs when using ResNet56 as the backbone and the PHY MEC baseline (DT-JSCC). As discussed above, the DT-JSCC encoder is more than 100 times larger than the PhyDNNs head. The whole ResNet20 has a total of 250 K parameters which is 17 times smaller than the DT-JSCC encoder alone. This is mainly because JSCC schemes are designed for data transmission rather than distributed inference, thus ignoring the resource constraints of the mobile device. As illustrated in Fig. 11, executing the DT-JSCC encoder on the mobile device takes around 7× more time and consumes about 8× more energy than the PhyDNNs head, making them impractical in time-sensitive applications. Also, on average, DT-JSCC transmits 2.5 times more data for each batch of images. More in detail, DT-JSCC sends 41.7 KB of data for a batch size of 64, while PhyDNNs experience only 1.79% performance drop by transmitting only 16 KB. Note that the reported accuracy in Table 11 and the results in Fig. 11 are the average over LoS and NLoS scenarios in the office environment.

TABLE III: Comparison between DT-JSCC and PhyDNNs on CIFAR-10.

	DT-JSCC	PhyDNNs
#Params (Head)	4.36 M	42.5 K
#FLOPs (Head) [Mac]	618.4 M	291.65 M
Accuracy [%]	82.22	80.43

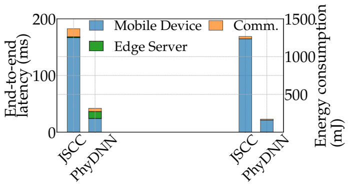


Fig. 11: Latency and energy consumption comparison between DT-JSCC and PhyDNNs.

V. Conclusions and Summary of Impact

In this work, we have proposed PhyDNNs, a new MEC approach where DNNs are modified to work directly at the physical layer. PhyDNNs avoid training the DNN from scratch and instead *modify* a pre-trained DNN to learn channel coding in addition to its inference task. Therefore, the extracted latent features can be transmitted at the waveform level significantly reducing both the communication delay and network load. We have developed a framework based on the IB theory to optimize the trade-off between the number of transmitted symbols and end task performance during channel coding. We have also prototyped PhyDNNs with an experimental testbed and shown that PhyDNNs decrease the inference latency, communication overhead, and energy consumption by up to 48×, 1385×, and 13× while keeping the accuracy within 7% with respect to the state-of-the-art approaches.

To the best of our knowledge, this paper is the first to experimentally demonstrate a task-oriented semantic communication system. As communication becomes the bottleneck in MEC applications, we believe this paper represents a step forward in addressing the current issue. As an interesting future research direction, we are planning to extend the proposed approach to the case where multiple mobile devices need to jointly minimize the amount of transmitted data necessary to perform a collaborative task. The authors have provided public access to their code and data at https://github.com/AbdiMohammad/PhyDistInf. **Overall, we hope this work will elicit further discussions and follow-up work in the semantic communications and mobile edge computing communities.**

Acknowledgments

This work is funded in part by the National Science Foundation (NSF) grant CNS-2134973, ECCS-2229472, and ECCS-2329013, by the Air Force Office of Scientific Research under contract number FA9550-23-1-0261, by the Office of Naval Research under award number N00014-23-1-2221.

References

[1] X. Wu, D. Sahoo, and S. C. Hoi, “Recent Advances in Deep Learning for Object Detection,” *Neurocomputing*, vol. 396, pp. 39–64, 2020.

[2] Y. Mo, Y. Wu, X. Yang, F. Liu, and Y. Liao, “Review the State-of-the-art Technologies of Semantic Segmentation Based on Deep Learning,” *Neurocomputing*, vol. 493, pp. 626–646, 2022.

[3] F. Tao and Q. Qi, “Make More Digital Twins,” *Nature*, vol. 573, no. 7775, pp. 490–491, 2019.

[4] M. Xu, W. C. Ng, W. Y. B. Lim, J. Kang, Z. Xiong, D. Niyato, Q. Yang, X. S. Shen, and C. Miao, “A Full Dive into Realizing the Edge-enabled Metaverse: Visions, Enabling Technologies, and Challenges,” *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2022.

[5] K. Raaen and I. Kjellmo, “Measuring Latency in Virtual Reality Systems,” in *International Conference on Entertainment Computing (ICEC)*, pp. 457–462, Springer, 2015.

[6] M.-T. Suer, P. Jose, and H. Tchouankem, “Experimental Evaluation of IEEE 802.11 ax - Low Latency and High Reliability with Wi-Fi 6?,” in *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, pp. 377–382, IEEE, 2022.

[7] M. Laha, D. Roy, S. Dutta, and G. Das, “AI-assisted Improved Service Provisioning for Low-latency XR over 5G NR,” *IEEE Networking Letters*, 2023.

[8] S. Iwasaki, X. Lei, K. Chida, Y. Sugito, K. Iguchi, K. Kanda, H. Miyoshi, and Y. Uehara, “The required video bitrate for 8k120-hz real-time temporal scalable coding,” in *2020 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–5, IEEE, 2020.

[9] R. Dilli, “Analysis of 5G Wireless Systems in FR1 and FR2 Frequency Bands,” in *International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pp. 767–772, IEEE, 2020.

[10] S. Xie, S. Ma, M. Ding, Y. Shi, M. Tang, and Y. Wu, “Robust information bottleneck for task-oriented communication with digital modulation,” *IEEE Journal on Selected Areas in Communications*, 2023.

[11] N. Tishby, F. C. Pereira, and W. Bialek, “The information bottleneck method,” *arXiv preprint physics/0004057*, 2000.

[12] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, “Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge,” *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[13] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt, “Sc2 benchmark: Supervised compression for split computing,” *Transactions on machine learning research*, 2023.

[14] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco, “Distributed inference acceleration with adaptive DNN partitioning and offloading,” in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 854–863, IEEE, 2020.

[15] C. Hu, W. Bao, D. Wang, and F. Liu, “Dynamic adaptive DNN surgery for inference acceleration on the edge,” in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1423–1431, IEEE, 2019.

[16] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, “JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services,” *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2019.

[17] A. E. Eshratifar, A. Esmaili, and M. Pedram, “BottleNet: A Deep Learning Architecture for Intelligent Mobile Cloud Computing Services,” in *Proceedings of IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, IEEE, 2019.

[18] J. Shao and J. Zhang, “BottleNet++: An End-to-end Approach for Feature Compression in Device-Edge Co-Inference Systems,” in *Proceedings of IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, IEEE, 2020.

[19] M. Jankowski, D. Gündüz, and K. Mikolajczyk, “Joint device-edge inference over wireless links with pruning,” in *2020 IEEE 21st international workshop on signal processing advances in wireless communications (SPAWC)*, pp. 1–5, IEEE, 2020.

[20] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, “Distilled Split Deep Neural Networks for Edge-Assisted Real-Time Systems,” in *Proceedings of the Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pp. 21–26, 2019.

[21] Y. Matsubara, D. Callegaro, S. Baidya, M. Levorato, and S. Singh, “Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems,” *IEEE Access*, vol. 8, pp. 212177–212193, 2020.

[22] Y. Matsubara, D. Callegaro, S. Singh, M. Levorato, and F. Restuccia, “BottleFit: Learning Compressed Representations in Deep Neural Networks for Effective and Efficient Split Computing,” in *Proceedings of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2022.

[23] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt, “Supervised Compression for Resource-Constrained Edge Computing Systems,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 2685–2695, 2022.

[24] E. Bourtsoulatze, D. B. Kurka, and D. Gündüz, “Deep joint source-channel coding for wireless image transmission,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 3, pp. 567–579, 2019.

[25] D. B. Kurka and D. Gündüz, “DeepJSCC-f: Deep joint source-channel coding of images with feedback,” *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 178–193, 2020.

[26] N. Farsad, M. Rao, and A. Goldsmith, “Deep learning for joint source-channel coding of text,” in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 2326–2330, IEEE, 2018.

[27] 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, 2020.

[28] Y. M. Saidutta, A. Abdi, and F. Fekri, “Joint source-channel coding over additive noise analog channels using mixture of variational autoencoders,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2000–2013, 2021.

[29] J. Dai, S. Wang, K. Tan, Z. Si, X. Qin, K. Niu, and P. Zhang, “Nonlinear transform source-channel coding for semantic communications,” *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 8, pp. 2300–2316, 2022.

[30] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, “Variational inference: A review for statisticians,” *Journal of the American statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[31] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy, “Deep variational information bottleneck,” *arXiv preprint arXiv:1612.00410*, 2016.

[32] J. Ballé, V. Laparra, and E. P. Simoncelli, “End-to-end optimized image compression,” *arXiv preprint arXiv:1611.01704*, 2016.

[33] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, “Variational image compression with a scale hyperprior,” *arXiv preprint arXiv:1802.01436*, 2018.

[34] D. Minnen, J. Ballé, and G. D. Toderici, “Joint autoregressive and hierarchical priors for learned image compression,” *Advances in neural information processing systems*, vol. 31, 2018.

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

[36] S. Singh, S. Abu-El-Haija, N. Johnston, J. Ballé, A. Shrivastava, and G. Toderici, “End-to-end learning of compressible features,” in *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 3349–3353, IEEE, 2020.

[37] T. Melodia, S. Basagni, K. R. Chowdhury, A. Gosain, M. Polese, P. Johari, and L. Bonati, “Colosseum, the world’s largest wireless network emulator,” in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pp. 860–861, 2021.

[38] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.