DynSplit: A Dynamic Split Learning Scheme for 5G-Enpowered Metaverse

Yunmeng Shu, Pengwenlong Gu, Cédric Adjih, Chung Shue Chen, Ahmed Serhrouchni

DynSplit: A Dynamic Split Learning Scheme for 5G-Enpowered Metaverse

Yunmeng Shu*†, Pengwenlong Gu‡, Cédric Adjih‡, Chung Shue Chen§ and Ahmed Serhrouchni¶

*Shanghai Jiao Tong University, Shanghai, China
†MINES Paris, PSL University, France
‡Inria, Saclay Center, 91120 Palaiseau, France
§Nokia Bell Labs, Paris-Saclay, 91300 Massy, France
¶LTCI, Telecom Paris, ENST, Institut Polytechnique de Paris, France

***Abstract*—The Metaverse is a virtual world based on numerous technologies, which enables users to interact socially in a persistent online 3-D virtual environment. To generate high-level imaginary environments, extremely low latency data transmission and learning-based sensor data analysis are required. With the development of 5G techniques, processing and learning methods, both the transmission delay and high-quality scene generation have been significantly improved in meta-applications. However, many Metaverse devices are battery-powered, and local processes and learning are still too costly. To address this issue, in this paper, by taking full architectural advantage of 5G networks, we propose a novel dynamic split learning scheme for enabled Metaverse systems. In our proposed scheme, each neural network is split into two segments, and the upper segment is stored at the base station (BS) side. Thus, between two segments, multiple pathways are featured, each with distinct compression ratios, accompanied by a gating mechanism that intelligently guides the selection of paths for each input data. This design excels in adapting to diverse Metaverse applications and network conditions, enhancing both the learning and computing phases of split models. Simulation results underscore the efficacy of our proposed scheme, revealing that it does not impede the convergence of split learning models. Furthermore, the scheme demonstrates notable performance gains in terms of communication overhead, prediction accuracy, and adaptability to resource constraints.**

***Index Terms*—5G, split learning, dynamic network**

I. Introduction

The Metaverse is a virtual world based on numerous technologies, which enables users to interact in a persistent online 3-D virtual environment. As discussed in [1], sensing is the foundation of Metaverse. The meta equipment needs to intelligently sense and update users' gestures, surrounding environments, and objects that users interact with. To generate a high-level imaginary environment, it requires learning methods to help analyze sensor data and enhance the augmented reality (AR) scenes [2], and the data exchange among users should be with extremely low latency. In the past few years, with the help of developed 5G techniques and learning methods, both the transmission delay and the scene generation have been well improved in meta-applications.

This work was supported by the 5G-mMTC project which was funded by the French government as part of the "Plan de Relance et du Programme d'investissements d'avenir".

Nevertheless, the constraints posed by computation capacity, energy level, and storage, particularly for battery-powered Metaverse devices, continue to present an important challenge when it comes to execute complex model training or inferencing [3]. In the present landscape, numerous strategies have arisen to alleviate the substantial computational burdens. These methods can be broadly categorized into two groups: the development of lightweight learning models tailored to match the resource constraints of devices, and the offloading of certain segments of the learning model from devices, redirecting them to cloud-based or edge servers. Among these various approaches, split learning stands out as a noteworthy solution [4] [5]. It effectively tackles the computational limitations by decoupling a learning model into two segments and offloading one of them to a robust remote server. Simultaneously, split learning can preserve the privacy of user data since only smashed data is exchanged between segments, making it a promising solution in this context. Especially for Metaverse applications in the context of 5G Ultra-Reliable Low Latency Communications (URLLC) [6], 5G base stations or edge servers near the 5G core can act as reliable remote servers and can provide information to help meta users sense better the surrounding environment. However, within the realm of split learning, the significant communication overhead between the split segments remains a major challenge for the wireless network's capabilities.

To tackle the challenge of communication overhead, many research efforts have been made in the past few years. One straightforward solution is to reduce the total bits users transfer in each round or the total number of rounds [7] [8], which is used in both federated learning and split learning schemes. One major drawback of this solution is that it does not guarantee the convergence of model training [9]. Thus, many efforts like local-loss-based training [10], gradient or activation compression [11] and neural network structural optimization [12] have been made to improve the training convergence and latency in split or FedSplit learning methods. However, to adapt to rich meta-applications and various network states, case-by-case model training and optimization are normally necessary for the techniques mentioned above. It encourages us to develop a widely suitable streamlined architectural framework that can adapt to a wide range of

meta-applications under different network conditions.

In this paper, we introduce a novel dynamic split learning scheme *DynSplit* for 5G-enabled Metaverse systems, which can dynamically adapt to diverse network conditions, enhancing both the learning and computing phases of split models. Specifically, for one meta-application that requires user-side learning, we split the neural network into two segments and upload the upper segment to the BS. Then, multiple pairs of autoencoders with different compression ratios are implemented at the split layer, and a gating mechanism can intelligently decide the compression ratio of each activation and activate the corresponding autoencoder. The gating mechanism and autoencoders are trained in a unified training process to establish an efficient dynamic compression mechanism. Simulation results demonstrate that our proposed scheme maintains the original split learning model's convergence rate. Moreover, by dynamically compressing meshed data between meta devices and the BS at varying rates, our proposed scheme can substantially diminish communication overhead, effectively optimizing the interaction between devices and the BS and maintaining the quality of training and computation.

The rest of this paper is organized as follows. Some related work is introduced and analyzed in Section II. We introduce the system model and our dynamic split learning scheme in Sections III and IV respectively. The performance of the proposed schemes is evaluated via simulations in Section V. Finally, conclusions are drawn for this paper in Section VI.

II. Related Work

Some edge-cutting learning efficiency-improving techniques for split learning are briefly discussed in the previous section. Many great research efforts especially in recent years fall into the following two categories: (i) gradient or activation compression and (ii) dynamic neural network with early exits, since they have the potential to adapt to heterogeneous highly dynamic network environments.

Gradient compression techniques encompass four major themes [13]: sparsification, quantization, low-rank approximation and gradient similarity. Sparsification aims to sparsify the gradient tensor by setting values below a certain threshold to zero. In this way, only the "top-k" [11] or a small portion of randomly selected gradients [14] need to be transmitted to the server side in each training round. Test results show that even with a 99% sparsification ratio, a 99.42% learning accuracy can be achieved [15]. With quantization techniques like Monte Carlo quantization [16], nested dithered quantization [17] or stochastic quantization [18], gradients can be compressed to a fixed length, which can significantly reduce communication bits and maintain a high learning accuracy. Low-rank approximation methods do the compression by factorizing the gradient matrix into two low-rank matrices, typically smaller than the original matrix. One representative method is PowerSGD [19], with which the approximation is computationally light-weight and can avoid prohibitively expensive Singular Value Decomposition. Another method ATOMO [20] aims to minimize the variance of quantized stochastic gradients to achieve a low-rank approximation of the gradient. Gradient similarity methods are designed for multiple-user scenarios, which focus on correlations among gradients from different nodes. For example, ScaleCom [21] proposed by Chen *et al.* leverages similarity in the gradient distribution amongst learners to provide significantly improved scalability. Activation compression can also significantly reduce the communication overhead in split learning, typically achieved through the application of autoencoders [22].

Besides the gradient and activation compression methods, the dynamic neural network is another key technique that can adapt the neural network's structure to a particular task during inference, ensuring high efficiency, adaptiveness, and generality. Dynamics in neural networks include dynamic architecture and dynamic parameters training. In terms of dynamic architecture, layer skipping [23], multi-task learning [24], channel skipping [25] and dynamic routing [26] provide dynamic and help reduce computation cost in learning procedures. Recently, dynamic parameters training has been proposed to address the geometric variation issue in object recognition. For example, in [27], the convolutional weights are adjusted according to the attention mask generated from the input feature at each layer. And in [28] the authors instantiated one possible solution as Deformable Kernels (DKs), which can sample weights within the kernel space to modify the effective reception field (ERF) while keeping the reception field unchanged.

In summary, we discussed two main solutions for enhancing split learning efficiency: compressing intermediate data and increasing neural network training dynamics. In the following sections, we will explore combining these techniques to propose a dynamic compression scheme for smashed data, aiming to improve model accuracy and reduce training complexity.

III. Split Learning in 5G Metaverse

In this paper, we consider a simplified split learning scheme for a 5G Metaverse system as given in Fig. 1, in which a branch of Metaverse devices connects to a 5G BS via wireless links. In real systems, the AR/VR headset, AR glasses, or other types of battery-powered equipment like watches or sensors are the most widely discussed Metaverse devices. Thus, although with a split learning model, one major concern is whether the battery life of the device is enough to support long-term 5G data transmission.

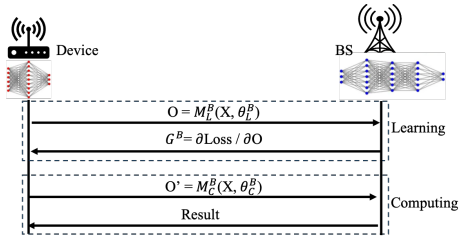


Fig. 1: Simplified 5G Metaverse communication model and an overview of split learning and computing.

The published 3GPP Release 17 offers a solution, that scales down the wideband 5G new radio (NR) design, reduces the number of receive antennas at the device and limits

the duplexing mode and the modulation order during the communication procedures. These reductions are expected to achieve a 70% energy saving on the devices [29] for 5G transmissions. Following the standard TR 38.875 [29], we assume that each device is equipped with a single antenna and the duplexing model between devices and the base station is half-duplex FDD, which means that each device can either transmit or receive in the one-time slot.

To implement split learning and computing within this network, we employ a neural network architecture between each device and the BS. In order to preserve data privacy, the neural networks are trained locally. In this approach, each neural network is divided into two distinct segments: the lower segment is stored on the Metaverse device, and the upper segment is situated at the BS. We assume that M_L^B is the device-side model and the vector θ_L^B represents its weight. Consequently, every training iteration encompasses a device-to-BS forward pass for transmitting the smashed data, followed by a BS-to-device backward pass, which serves the purpose of updating the gradients of the lower segment. This process can be outlined as follows:

- Forward pass (inference and forward propagation): the device first feeds the sample features X to its local model M_L^B and gets the intermediate output $O = M_L^B(X, \theta_L^B)$, then transmits it to the remote model at the BS. Using the intermediate output O, the remote model continues and gets the final output $\hat{Y}$.
- Backward pass (backpropagation): The BS calculates the loss function $Loss(\hat{Y}, Y)$ and the gradients for both the remote model G^T and the local model G^B. At last, G^B will be transmitted back to the local model for update.

After the model training is completed, the split computing is performed in the same way as the forward pass. Only this time, the trained model M_L^B is used to compute the intermediate output O' at the device side and then transmit it to the BS. At last, the remote model at the BS continues to calculate the result R and transmit it back to the device.

IV. Dynamic Multipath Split Scheme

As stated in the introduction, 5G networks stand apart from other use cases due to their distinctive focus on facilitating sophisticated applications and accommodating an extensive array of device types. Consequently, the adaptability of learning methods to diverse Metaverse devices and network environments, while ensuring consistently high accuracy, emerges as an enduring and paramount challenge.

In this section, we introduce a novel dynamic split learning approach tailored for Metaverse devices. Our discussion is structured to first delineate the DynSplit architecture, as referenced in IV-A, followed by an exposition of our lightweight auto-gating mechanism detailed in IV-B. Subsequently, the manuscript will elucidate our proposed algorithm and elaborate on the training methodologies in the ensuing sections.

A. Dynamic Smashed Data Compression

The idea of using a dynamic deep neural network (DNN) was first discussed for fast inferences by adding some side branches and allowing certain test samples to exit early [12]. Thus, more features like dynamic architecture and parameters have been discussed in the past few years [30]. However, within the 5G Metaverse landscape, the constrained computing capabilities of individual devices and the massive access of Metaverse devices pose a significant challenge. Customizing dynamic DNNs for each device independently would substantially escalate the intricacies of network configuration and management. Consequently, the key hurdle in designing dynamic DNN for the 5G Metaverse environment lies in establishing a versatile and streamlined architectural framework that can seamlessly adapt to diverse applications and network conditions.

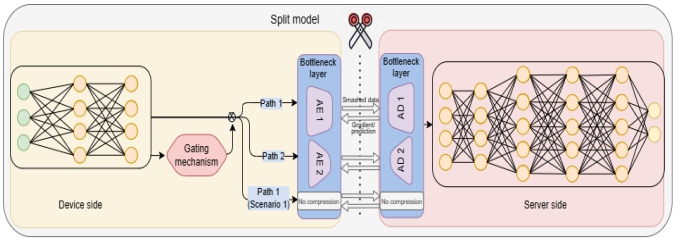


Fig. 2: The architecture of DynSplit. To adapt to diverse devices and network conditions, we introduce two distinct models, each offering varying options for training complexity: one with one compressless path and one compress path (Scenario 1), while the other one is with two compress paths with varying rates of compression (Scenario 2). For each compress path in both models, we implement a pair of pre-trained autoencoders, including encoders (AE) and decoders (AD), strategically positioned at both the device and the BS sides.

Autoencoder-injection is considered a promising technique to tackle this challenge since the bottleneck it introduces can significantly reduce the communication overheads and help to improve the performance of split models [31] [32] at the same time. Even traditional autoencoders with lightweight convolutional and transposed convolutional layers can contribute to good results. We follow the setting of autoencoders as [33]. Thus, in this paper, we propose a dynamic multipath compression split learning and computing scheme as illustrated in Fig. 2. In our scheme, a DNN model is split into two parts and stored on the device and the BS respectively. Between these two parts, we implement multiple pathways, denoted as $pathway_i$ for i-th paths, leveraging autoencoders with different intermediate layers to establish an efficient dynamic compression mechanism for both the learning and computing phases. A gating mechanism, a module that takes a_l, the activation of l-th cut layer as input, decides which path to take for a particular input data. Note that the path chosen by the activation will also be used for the backpropagation.

More details about our proposed scheme including the learning phase, the inference phase and the gating mechanism will be presented in the following sub-sections.

B. Gating Mechanism

The gating mechanism plays an important role in our dynamic multipath split learning scheme since the ratio distributed to different paths directly decides the communication overheads and the learning accuracy. We formulate the $ratio = \frac{|data_{Pathway1}|}{|data_{total}|}$, which indicates that the higher the ratio the lower the compression. Thus, inspired by [34] [35], we design a gating mechanism to decide which route to go through for a particular data, which can be defined as:

$$a_{l+1} = \sum_{i=1}^{n} E_i(a_l) \cdot g_i(a_l), \tag{1}$$

where $g_i(a_l) \in \{0, 1\}$ is the output of gate g_i for each path i using the activation at the l-layer as input and $\sum_{i=1}^{n} g_i(a_l) = 1$. E_i denotes the encoder for the i-th path. Thus, the gate module determines the path selection by assigning to the corresponding gate $g_i(a)$ a value 1, and to all other gates $g_j(a)$ (where $j \neq i$) a value 0. In this way, the two paths scenario can be considered as a hard attention mechanism.

This gating layer aims to make discrete decisions by considering the relevance scores, which will be extracted through several layers of neural networks. Rather than opting for a simplistic strategy of selecting the maximum relevance score to determine a particular route to go through, we consider the gate's uncertainty to avoid encountering rapid mode collapse, utilizing Gumbel distribution as its key property Gumbel-Max trick [36]. Thus, as suggested in [30], we opt to use a reparameterization method Gumbel SoftMax since it can convert hard decisions into a continuous probability distribution.

A random variable Gu follows a Gumbel distribution if it can be expressed as $Gu = \mu - \log(-\log(U))$, where μ is a real location parameter and $U \sim \mathcal{U}[0, 1]$. According to the GumbelMax method, if we sample K Gumbel distributions with location parameters $\{\mu_{k'}\}_{k'=1}^{K}$, the highest result among the K samples follows a softmax probability distribution based on its location parameter:

$$P(k^{th} \text{ is the largest} | \{\mu_{k'}\}_{k'=1}^{K}) = \frac{e^{\mu_k}}{\sum_{k'=1}^{K} e^{\mu_{k'}}}. \tag{2}$$

This allows us to parameterize discrete distributions using Gumbel random variables. More precisely, if there exists a discrete random variable G with probabilities $P(G = k) \propto \alpha_k$ and a sequence of independent and identically distributed (i.i.d.) Gumbel random variables $\{Gu_k\}_{k\in\{1,\dots,K\}}$ with a location of $\mu = 0$. We can sample G using Gumbel random variables as follows:

$$G = \arg \max_{k\in 1,\dots,K} (\log \alpha_k + Gu_k). \tag{3}$$

In our proposed method, the sampled pathway denoted as G is the selected route for the inference during the forward pass. Simultaneously, the corresponding path is assigned a value of 1, while the other paths are assigned a value of 0.

Subsequently, we introduce a continuous relaxation for (3) [37] [38] by replacing the Argmax function with a softmax function. In this way, a sample from the Gumbel-Softmax relaxation can be obtained as follows:

$$\tilde{G}_k = \text{softmax}\left(\frac{\log \alpha_k + Gu_k}{\tau}\right), \tag{4}$$

where $\tilde{G}_k$ is the k-th element in vector $\tilde{\mathbf{G}}$, and τ is the softmax temperature parameter which controls randomness. The vector obtained $\tilde{\mathbf{G}}$, is subsequently employed in the pathway selection process during backpropagation.

C. Split Learning and Computing algorithms

Detailed algorithms for both learning and computing phases are presented in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1: Dynamic split learning algorithm

Knowledge: Client c has the bottom model F_c, the encoder part of the bottleneck module $\mathcal{M}_E$, raw data (X, Y), gating mechanism $\mathcal{G}$. Server s has the top model F_s and decoder part of the bottleneck module $\mathcal{M}_D$.

Initialization: Model parameters of F_s, F_c, $\mathcal{M}_E$, $\mathcal{M}_D$, $\mathcal{G}$.

while $l > \epsilon_c$ **do**

Client Training Algorithm: *round* t

$\mathcal{A}^t \leftarrow F_c(X^t)$;
$g^t = \mathcal{G}(\mathcal{A}^t)$;
Smashed$^t \leftarrow \mathcal{M}_E(\mathcal{A}^t \cdot g^t)$;
Send g^t, *Smashed*^{t}, Y^t to s;
Receive ∇*Smashed*^{t} from s;
Compute gradient $\nabla_{\theta_c^t}(\nabla$*Smashed*$^t)$;
Update F_c, $\mathcal{M}_E$ and $\mathcal{G}$: $\theta_c^{t+1} = \theta_c^t - \eta\nabla_{\theta_c^t} l^t$

Server Training Algorithm: *round* t

Receive *Smashed*^{t} from c;
$Y_{pred}^t \leftarrow F_s \circ \mathcal{M}_D($*Smashed*$^t)$;
$l^t \leftarrow \mathcal{L}_{CE}(Y_{pred}^t, Y^t) + \lambda\mathcal{L}_{ratio}(g^t)$;
Compute gradient $\nabla_{\theta_s^t} l^t$;
Send ∇*Smashed*^{t} to c;
Update F_s and $\mathcal{M}_D$: $\theta_s^{t+1} = \theta_s^t - \eta\nabla_{\theta_s^t} l^t$

end

During the learning phase, at each round t, the client c selects a batch of raw data (X^t, Y^t) for activation learning with its local model F_c. The activation $\mathcal{A}^t$ is then passed to the gating $\mathcal{G}$ who determines the pathway to the server for this activation. Thus, using the corresponding encoder $\mathcal{M}_E$, the activation is compressed with a set ratio, and the smashed data $Smashed^t$ is transmitted to the server side.

After receiving the smashed data $Smashed^t$ and labels Y^t from the client c, the server s leverages $\mathcal{M}_D$ to decompress $Smashed^t$ and passes the result to the server-side model F_s to generate predicted labels Y_{pred}^t. This whole procedure at the server side is denoted as $F_s \circ \mathcal{M}_D$ in Algorithm 1. Subsequently, it computes the cross-entropy loss $\mathcal{L}_{CE}$. Thus, $\mathcal{L}_{CE}$ combined with the loss of ratio $\mathcal{L}_{ratio}$ through the coefficient λ, are used to calculate the gradient F_s and $\mathcal{M}_D$

at the server side. Note that $\mathcal{L}_{ratio}$ is designed to make the system's gate mechanism trained to control the ratio of data passing through each path.

Then the gradient of the smashed data $\nabla Smashed^t$ is sent back to the client c, and the client computes the gradient of the local model parameters and updates the models. The whole learning algorithm will stop when it reaches the convergence condition, which means that the difference in loss is smaller than a prefixed threshold ϵ_c.

Algorithm 2: Dynamic split computing algorithm

Input: X

For the client:
$\mathcal{A} \leftarrow F_c(X)$
$g = \mathcal{G}(\mathcal{A})$
Smashed $\leftarrow \mathcal{M}_E(\mathcal{A}) \cdot g$
Send (*Smashed*) to s

For the server:
Receive *Smashed* from c
$Y_{pred} \leftarrow F_s \circ \mathcal{M}_D(Smashed)$

Output: Y_{pred}

For the computing phase, the data X as the input first passes through the trained local model F_c. Subsequently, the gating module $\mathcal{G}$ determines which encoder of $\mathcal{M}_E$ each activation should go through and transmit the smashed data *Smashed* to the server. The server decodes *Smashed* with the decoder $\mathcal{M}_D$ and continues the inference with its model F_s to obtain the predicted label Y_{pred}.

The elaborated training process is listed below:

Step 1: We implement our whole system accordingly on both the Metaverse device and the cloud server. As in split learning scenarios, we choose to use the joint training method for 3 modules: split model, bottleneck layer and gating mechanism.

Step 2: Select the appropriate ratio according to the arithmetic memory of the displayed device and the communication conditions.

Step 3: We use the Cross-Entropy loss and the MSE loss for the target ratio. The equation is as follows:

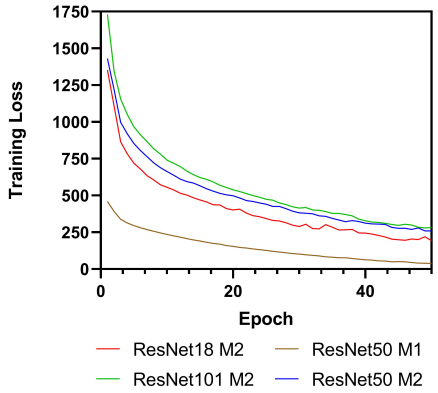
$$\mathcal{L} = -\sum_i y_i \log(\hat{y}_i) + (target_R - \mu_R)^2 \tag{5}$$

where y_i is the probability of class i in the true distribution, $\hat{y}_i$ is the probability of class i in the model's prediction, while $target_R$ and μ_R explain the set ratio and the batch-average ratio observed in real training.

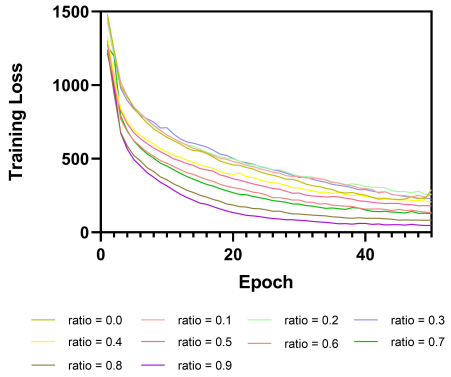
V. Experimental Results

A. Experimental Setup

We test our proposed multipath dynamic split learning scheme via simulations. In order to be as close to the IoT environment as possible, we choose the ResNet model [39] and use the Cifar-10 [40] data set for training and testing.



(a) Convergence for different models with $ratio = 0.2$.



(b) Convergence of model with different ratios with ResNet50.

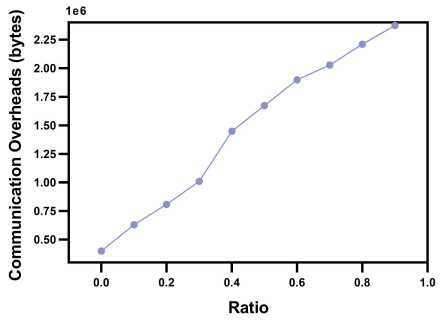
Fig. 3: Convergence analysis of different models in *Scenario 2*.

All training and testing are performed on an old server from the year 2018 using PyTorch. Specifically, both models are split at the second residual block following the guidelines provided in [33] and [41]. Additionally, each model features two pathways. In *Scenario 1*, Pathway 1 lacks an autoencoder, whereas Pathway 2 incorporates an autoencoder with a single intermediate channel. In contrast, *Scenario 2* equips both pathways with autoencoders: Pathway 1 is designed with twelve intermediate channels and Pathway 2 with two intermediate channels. Moreover, a quantization layer is appended subsequent to the encoders in both pathways.

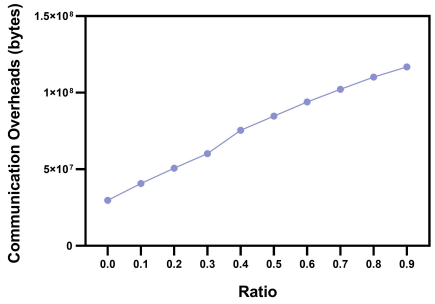
To ensure a rigorous comparison, we selected the original split learning model devoid of compression and a recently published split learning model equipped with a solitary autoencoder, as cited in [42], to serve as benchmark frameworks. The forthcoming sub-section will detail the simulation results. Note that non-pretrained ResNet models are consistently utilized, except in analysis where the application of pre-trained models is explicitly examined.

B. Performance Evaluation

Model convergence: First, we evaluate the learning convergence of our proposed scheme. We conduct tests with various models and ratios, training both models for 50 epochs. Results are illustrated in Fig. 3. Firstly, we tested both Scenarios with different ResNets with a fixed ratio $ratio = 0.2$, their convergences are given in Fig. 3a. Then we trained the model



(a) Communication cost for inference.



(b) Communication cost for training.

Fig. 4: Communication cost for inference and training.

in Scenario 2 with ResNet50 and different ratios, ranging from 0.0 to 0.9. Their convergences are illustrated in Fig. 3b. In Fig. 3b, the curve with $ratio = 0$ can be considered as a complete compressed scenario (2 intermediate channels), which confirms that our proposed dynamic multipath compression method does not delay the convergence of a split learning model.

Communication overheads: The communication overhead at the splitted layer remains unaffected by variations in model size, as it is solely associated with the number of intermediate channels in each autoencoder. Therefore we modified the intermediate channel setting in ResNet50, set Pathway 1 with 12 channels and Pathway 2 with 2 channels. Then we launched tests with different ratios, results are given in Fig. 4. Fig. 4a shows the communication cost for one inference and Fig. 4b shows the total communication overheads during the training procedure. We can see that communication overheads are almost linear, but the slope of the overheads is larger at ratios greater than 0.5 than at smaller ratios. This is due to the fact that the model tends to use a more accurate path when it is trained with a less stringent ratio setting.

Besides, we can also confirm that the added computation or storage cost due to our added autoencoders and the gating mechanism is extremely low, which is lower than 0.2% (Note: $\frac{|AE|+|G|}{|M|} = 0.0016$, with $|AE|$, $|G|$, and $|M|$ denoting the sizes of the autoencoder, gating module, and whole model, respectively). It means our model can adapt well to battery-powered Metaverse devices.

Prediction accuracy: Table I illustrates the accuracy performance of different models with various model sizes. We tested

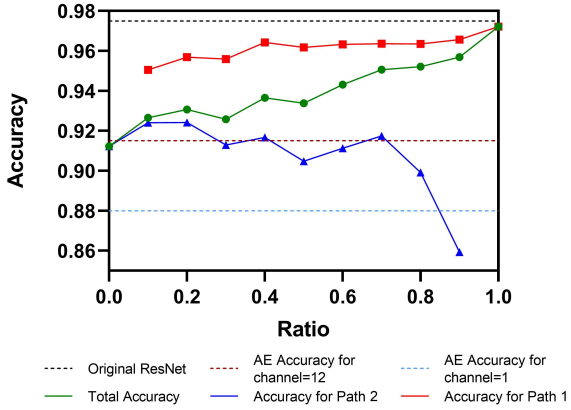


Fig. 5: Model performance for pre-trained ResNet in *Scenario 1*.

3 different-sized models: ResNet18, ResNet50 and ResNet101. It can be observed that for all models, a trade-off exists between the communication overhead and prediction accuracy. Hence, the allocation ratio among different paths remains subject to optimization.

Table III illustrates the accuracy performance among different algorithms. "Org" refers the original split learning scheme and "12-C" denotes the single autoencoder scheme with 12 intermediate channels. As expected, the prediction accuracy increases as the ratio increases. Next, we compare the prediction accuracy with the same bandwidth constraint. Our target bandwidth is consistent with the single autoencoder scheme with 4 intermediate channels. Thus, in our proposed *Scenario 2*, we set one path with 12 intermediate channels and the other path with 2 intermediate channels. The same bandwidth is

TABLE I: Model Accuracy for Different Non-pretrained Models of Scenario 2

Acc / Ratio	Model		
	ResNet18	ResNet50	ResNet101
0.0	0.727	0.735	0.738
0.2	0.743	0.767	0.755
0.5	0.784	0.769	0.777
0.7	0.800	0.787	0.782
0.9	0.811	0.804	0.797
1.0	0.828	0.818	0.800

TABLE II: Model Accuracy for Different Algorithms for Non-pretrained ResNet50

Ac / Ra	Algorithms					
	Org	Scenario 1	Scenario 2	12-C	4-C	2-C
0.0	0.828	0.735	0.735	0.793	**0.753**	0.715
0.2		0.767	**0.765**			
0.5		0.769	0.778			
0.7		0.787	0.789			
0.9		0.804	0.798			

TABLE III: Model Accuracy for Different Algorithms for Non-pretrained ResNet50

Ac / Ra	Algorithms				
	Org	Scenario 2	12-C	4-C	2-C
0.0	0.828	0.735	0.793	**0.753**	0.715
0.2		**0.765**			
0.5		0.778			
0.7		0.789			
0.9		0.798			

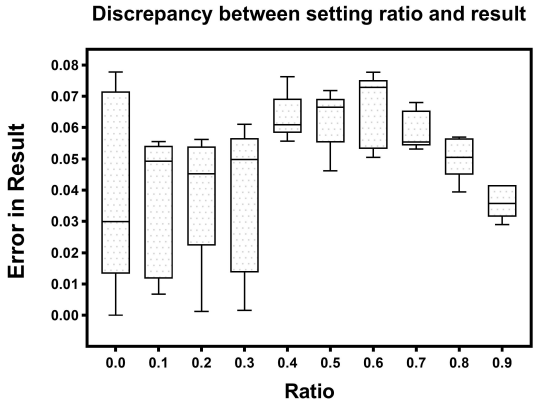


Fig. 6: Data distribution ratio error.

achieved at a ratio of 0.2 (note: $12 \times 0.2 + 2 \times 0.8 = 4$). Our proposed method surpasses the single autoencoder scheme by 0.012 in terms of prediction accuracy, from 0.753 to 0.765.

A notable enhancement in prediction accuracy is evident in Fig. 5, particularly where *Scenario 1*'s Pathway 2 incorporates a single intermediate channel. Remarkably, even with a ratio close to 0, our proposed scheme consistently outperforms the single autoencoder scheme (0.912 compared to 0.88). This implies that even a small portion of data passing through Pathway 1 without compression contributes to an improvement in the overall system's prediction accuracy. Moreover, when the ratio distributed to Pathway 1 increases, the total prediction accuracy experiences a proportional increase. This substantial improvement highlights the significant benefits stemming from the joint training of two paths in our model.

Gating mechanism efficiency:

In our proposed scheme, we set a gating mechanism to decide the path of each inference. Its efficiency decides directly if our scheme can dynamically adapt to the resource constraints in IoT scenarios. We test the efficiency of the gating mechanism by manually changing the ratio between the two paths, results are given in Fig. 6.

The analysis reveals that the trained model exhibits inference results wherein the distribution of data through each path closely mirrors the predefined proportions. The maximum observed error is approximately 5%.

We also study cases that penalize exceeding the set ratio. We considered a loss function with an added overflow value exceeding the set ratio, which can be represented as:

$$\mathcal{L} = CrossEntropyLoss + \max(\mu_R - target_R, 0)^2 \quad (6)$$

Fig. 7 shows the real ratio for every epoch. We can see that most of the final restricted ratios are close to the target ratio, with some small fluctuations possibly decreasing with the number of epochs. That is because the penalty term added in the loss function forces the real ratio to approach the set one. Fig. 8 shows the training accuracy with this new loss function. A larger fluctuation can be observed because we jointly considered the cross-entropy loss and the penalty to

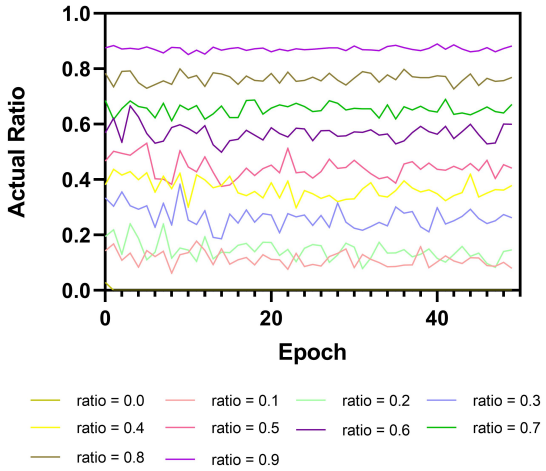


Fig. 7: Real ratio wave with penalty loss.

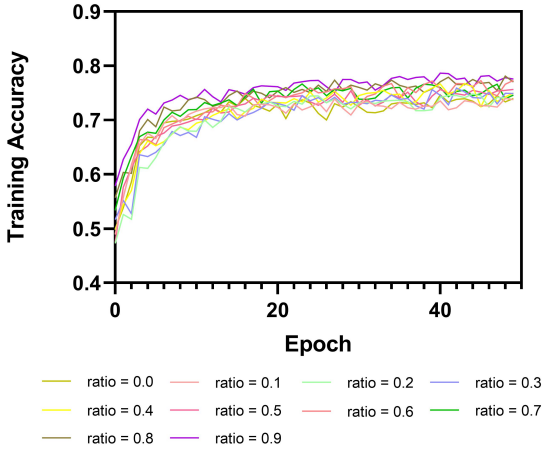


Fig. 8: Accuracy with loss penalty.

the exceeded ratio, which may extend the number of rounds required for convergence.

C. Rationality Analysis

The fundamental idea driving our motivation revolves around the concept of "simplicity" to predict with deep learning models within certain data. The rationale behind our choice of a gating mechanism is grounded in the principle that "simpler" data opt for traversing a more compressed route.

The entropy of the outcome is not solely dependent on data complexity, but also on the model's performance. To gauge this, we employ the average entropy computed for the original ResNet50 as the complexity indicator for each dataset. The average entropy values for both paths of *Scenario 1* are presented in Table IV. The complexity of the data through Pathway 2 is significantly lower than that of Pathway 1 (about half), indicating that simpler data has chosen a more compressed path, which validates the rationality of DynSplit.

VI. Conclusion and Future Work

Deploying deep neural networks in 5G Metaverse environments faces challenges due to computational capacity, energy

TABLE IV: Rationality entropy

	Entropy for path	Complexity for data
Data for Pathway 1	0.0021	0.0006
Data for Pathway 2	0.0019	0.0003

consumption, and storage limitations, especially for battery-operated devices. Existing methods like split learning and split computing only partially address these constraints. We propose a dynamic multipath split learning scheme for the 5G-enabled Metaverse, featuring multiple pathways with distinct compression ratios and a gating mechanism for intelligent activation routing. This approach leverages autoencoders for an efficient dynamic compression framework, adaptable to various meta-applications and network conditions. Simulations show that our scheme maintains split learning model convergence and significantly improves communication overhead, prediction accuracy, and resource adaptability in 5G Metaverse scenarios. Future research will focus on optimizing performance and communication, potentially by integrating multiple dynamic networks and split points to find optimal solutions within a broader optimization landscape.

ACKNOWLEDGEMENT

A part of the work has been carried out at the Laboratory for Information, Networking and Communication Sciences (www.lincs.fr) and INRIA-Nokia-Bell-Labs common lab. Part of this work has been conducted in the context of the BPI project 5G-mMTC; and also the Inria Challenge FedMalin.

REFERENCES

[1] F. Tang, X. Chen, M. Zhao, and N. Kato, "The Roadmap of Communication and Networking in 6G for the Metaverse," *IEEE Wireless Communications*, vol. 30, no. 4, pp. 72–81, 2023.

[2] Z. Zhang, F. Wen, Z. Sun, X. Guo, T. He, and C. Lee, "Artificial intelligence-enabled sensing technologies in the 5g/internet of things era: from virtual reality/augmented reality to the digital twin," *Advanced Intelligent Systems*, vol. 4, no. 7, p. 2100228, 2022.

[3] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An Efficient Training and Inference Engine for Intelligent Mobile Cloud Computing Services," Jan. 2018.

[4] C. Thapa, M. A. P. Chamikara, S. Camtepe, and L. Sun, "SplitFed: When Federated Learning Meets Split Learning," Feb. 2022.

[5] Y. Matsubara, M. Levorato, and F. Restuccia, "Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges," Mar. 2022.

[6] X. Lin, "An Overview of 5G Advanced Evolution in 3GPP Release 18," *IEEE Communications Standards Magazine*, vol. 6, no. 3, 2022.

[7] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated Learning: Strategies for Improving Communication Efficiency," 2017.

[8] L. Li, D. Shi, R. Hou, H. Li, M. Pan, and Z. Han, "To Talk or to Work: Flexible Communication Compression for Energy Efficient Federated Learning over Heterogeneous Mobile Edge Devices," in *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1–10, 2021.

[9] L. Wang, W. Wang, and B. Li, "CMFL: Mitigating Communication Overhead for Federated Learning," in *IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 954–964, 2019.

[10] D.-J. Han, D.-Y. Kim, M. Choi, C. G. Brinton, and J. Moon, "SplitGP: Achieving Both Generalization and Personalization in Federated Learning," in *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1–10, 2023.

[11] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient Sparsification for Communication-Efficient Distributed Optimization," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[12] S. Teerapittayanon, B. McDanel, and H. Kung, "BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks," in *23rd International Conference on Pattern Recognition (ICPR)*, Dec. 2016.

[13] L. Abrahamyan, Y. Chen, G. Bekoulis, and N. Deligiannis, "Learned Gradient Compression for Distributed Deep Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 7330–7344, 2022.

[14] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Interspeech*, 2015.

[15] A. F. Aji and K. Heafield, "Sparse Communication for Distributed Gradient Descent," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 440–445, 2017.

[16] G. Mordido, M. Van Keirsbilck, and A. Keller, "Monte Carlo Gradient Quantization," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3087–3095, 2020.

[17] A. Abdi and F. Fekri, "Nested Dithered Quantization for Communication Reduction in Distributed Training," *CoRR*, vol. abs/1904.01197, 2019.

[18] D. Alistarh, D. Grubic, J. Z. Li, R. Tomioka, and M. Vojnovic, "QSGD: communication-efficient SGD via gradient quantization and encoding," in *Advances in Neural Information Processing Systems*, 2017.

[19] T. Vogels, S. P. Karimireddy, and M. Jaggi, *PowerSGD: Practical low-rank gradient compression for distributed optimization*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[20] H. Wang, S. Sievert, Z. Charles, S. Liu, S. Wright, and D. Papailiopoulos, "ATOMO: Communication-Efficient Learning via Atomic Sparsification," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 9872–9883, Dec. 2018.

[21] C.-Y. Chen, J. Ni, S. Lu, X. Cui, P.-Y. Chen, X. Sun, N. Wang, S. Venkataramani, V. Srinivasan, W. Zhang, and K. Gopalakrishnan, "ScaleCom: scalable sparsified gradient compression for communication-efficient distributed training," in *Proc. of the 34th International Conference on Neural Information Processing Systems*, 2020.

[22] I. Manakov, M. Rohm, and V. Tresp, "Walking the Tightrope: An Investigation of the Convolutional Autoencoder Bottleneck," 2020.

[23] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "SkipNet: Learning Dynamic Routing in Convolutional Networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[24] J. Ma, Z. Zhao, X. Yi, J. Chen, L. Hong, and E. H. Chi, "Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, p. 1930–1939, 2018.

[25] W. Hua, Y. Zhou, C. M. De Sa, Z. Zhang, and G. E. Suh, "Channel Gating Neural Networks," in *Advances in Neural Information Processing Systems*, 2019.

[26] Y. Li, L. Song, Y. Chen, Z. Li, X. Zhang, X. Wang, and J. Sun, "Learning Dynamic Routing for Semantic Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[27] H. Su, V. Jampani, D. Sun, O. Gallo, E. Learned-Miller, and J. Kautz, "Pixel-Adaptive Convolutional Neural Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[28] H. Gao, X. Zhu, S. Lin, and J. Dai, "Deformable Kernels: Adapting Effective Receptive Fields for Object Deformation," 2020.

[29] "Study on support of reduced capability NR devices," Technical Report (TR) 38.875, 3rd Generation Partnership Project (3GPP), Mar. 2021.

[30] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic Neural Networks: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 7436–7456, Nov. 2022.

[31] T. Guo, "Cloud-Based or On-Device: An Empirical Study of Mobile Deep Inference," in *IEEE International Conference on Cloud Engineering (IC2E)*, pp. 184–190, Apr. 2018.

[32] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt, "SC2 Benchmark: Supervised Compression for Split Computing," *Transactions on Machine Learning Research*, Mar. 2023.

[33] J. Shao and J. Zhang, "BottleNet++: An End-to-End Approach for Feature Compression in Device-Edge Co-Inference Systems," June 2020.

[34] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "SkipNet: Learning Dynamic Routing in Convolutional Networks," in *Proceedings of the European Conference on Computer Vision*, 2018.

[35] A. Veit and S. Belongie, "Convolutional Networks with Adaptive Inference Graphs," May 2020.

[36] E. J. Gumbel, "Statistical Theory of Extreme Values and Some Practical Applications," *Journal of The Royal Aeronautical Society*, vol. 58, no. 527, p. 792–793, 1954.
[37] E. Jang, S. Gu, and B. Poole, "Categorical Reparameterization with Gumbel-Softmax," *CoRR*, vol. abs/1611.01144, 2017.
[38] C. J. Maddison, A. Mnih, and Y. W. Teh, "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables," *CoRR*, vol. abs/1611.00712, 2016.
[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.
[40] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., University of Toronto, 2009.
[41] C.-Y. Hsieh, Y.-C. Chuang, An-Yeu, and Wu, "C3-SL: Circular Convolution-Based Batch-Wise Compression for Communication-Efficient Split Learning," July 2022.
[42] A. Ayad, M. Renner, and A. Schmeink, "Improving the Communication and Computation Efficiency of Split Learning for IoT Applications," in *IEEE Global Communications Conference*, 2021.