# OctoFedS: 엣지에서 객체 검출을 위한 스플릿 컴퓨팅 연합 학습 시스템

**Bich-Ngoc Doan, Thanh-Tung Nguyen, 이동만**
**한국과학기술원 전산학부**
**{ngocdb160900, tungnt, dlee}@kaist.ac.kr**

# OctoFedS: A Federated Split Learning System for Object Detection at the Edge

**Bich-Ngoc Doan, Thanh-Tung Nguyen, Dongman Lee**
**School of Computing, KAIST**

**Abstract**

The large amount of data generated continuously by user devices at the Edge of the network can be leveraged to further improve state-of-the-art deep learning models. However, this practice presents new challenges in terms of data privacy. In this paper, we design OctoFedS, a federated split learning system that adopted Federated Learning and Split Computing to train object detection models without exposing data to privacy threats. Our experiments with model YOLOv1 using the PascalVOC dataset in distributed settings proved the validity of our approach while still achieving mean Average Precision of 0.66 (mAP).

## 1. Introduction and Related Work

In recent years, the amount of image and video data generated by users is astonishing. This data can be used for training Deep Learning (DL) models to serve a wide variety of tasks such as object detection. In traditional DL, data is collected and used to train models in centralized environments such as cloud data centers. However, this practice presents privacy risks especially for data containing private information (e.g., home surveillance footage). To tackle such a challenge, different learning methods such as Federated Learning (FL) have been proposed. In FL [1], Liu *et al.* proposed FedVision, a platform for training detection models. In FedVision, DL models are trained locally and then sent to a remote server, where they are aggregated to produce the *global* model. This allows privacy preservation, but it poses new challenges as training a full DL model is demanding for resource-limited user devices. Another method is Split Computing (SC), where a DL model is split into partial models, located at different devices. In [2], the authors leveraged SC to train DL models for medical usage without exposing sensitive medical records to privacy risks as data is accessible to only the partial model at the user's device. This method also allows computation-sharing among several devices. However, since the training data is generated by only one user, training performances are limited. In [3] and [4], the authors proposed different learning methods combining FL and SC. However, their applications were only limited to image classification with simple datasets such as MNIST and CIFAR10.

In this paper, we present OctoFedS, an Edge Computing system for training DL models for object detection, which is considered a more complex task compared to image classification. It uses SC to protect users' privacy and leverages Federated Aggregation to take advantage of a large distributed dataset. We used OctoFedS to train a YOLOv1 model distributedly and achieved nearly the mean Average Precision (mAP) of 0.66 compared to 0.715 of the traditional training method with a centralized dataset.

## 2. A Federated Split Learning System for Object Detection at the Edge

Our system adopts Split Computing [2] as the base framework. Under SC, a DL model is split into partial models and located at several devices, which communicate only the intermediate data output by the partial models. This approach provides two main benefits: secured privacy as data is locally preserved, and shared computational capability among multiple devices. In our system, the model is divided into two parts, *head* and *tail*, distributed to the client and the server respectively. This is illustrated in Figure 1(a).

### 2.1 Client

In OctoFedS, we consider each client a low-end edge device (e.g., Jetson Nano) or personal computer (e.g., smartphone), which has limited resources for training DL models, and thus, can benefit from sharing the computation load with another device. As shown in Figure 1(a), only the client has access to its data source, which allows privacy preservation. Moreover, each client holds a copy of the model's *head* that processes locally generated data and extracts intermediate features up to the split point. These intermediate features are also called *smashed data* as it has been processed by several convolutional layers and bears little resemblance to the original data. The intermediate features are then sent to the server, which will complete forward-propagation and sends back intermediate gradients so that the client can perform local weight update.

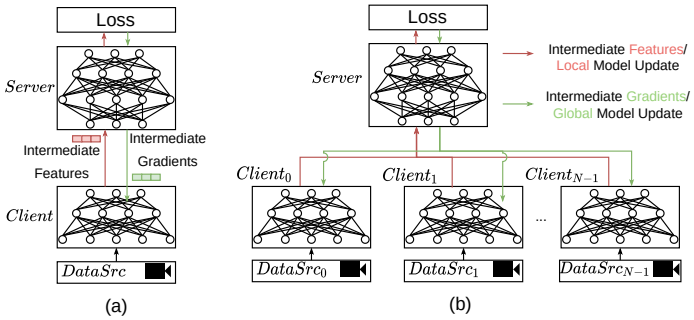


Figure 1: DL model trained with (a) Split Computing and (b) OctoFedS to take advantage of a larger dataset without unnecessary privacy risks

Furthermore, to overcome the obstacle of data distributedness and reap the full benefits of clients' data, we apply Federated Learning (FL) as shown in Figure 1(b). Now, instead of training each client separately, a cluster of clients are trained together. As each *head* model extracts a unique set of features, our system effectively expands the feature space to yield better training outcomes. This will be discussed in detail in the next sections.

### 2.2 Server

As shown in Figure 1(a), the server holds the *tail* model. Upon the reception of intermediate features from the clients, the *tail* model uses these features as input to complete forward propagation, extracting the most discriminative features. After working out the loss, it performs backward propagation and sends back the *intermediate gradients* to the client, which will update its model using the gradients. Furthermore, there is one server for each cluster of clients. This design is reasonable since that while the server may be computationally rich compared to clients, running several *tail* models is a non-trivial task that requires a large amount of GPU memory. Holding only one copy of the *tail* allows the server to serve a large number of clients.

### 2.3 Federated Learning

The training process of OctoFedS has two main phases in which a cluster of clients and the server will participate in sequence. It consists of global epochs, each of which has $N$ local epochs, where $N$ is the number of participating clients.

1. **Pre-training**: During this phase, each client acquire a sufficient amount of training data with labels. We assume the data is labeled either manually or using a *teacher* model [5]. We believe that this is a problem orthogonal to our focus and thus does not affect the study validity. Once inputs from a determined number of clients are ready, these clients will form a cluster and get prepared for training. The server will send fundamental training configurations including number of global epochs and learning rate, to each of the clients.

2. **Federated Training**: During this phase, each client inputs their data into the *head* model, which will accomplish the work up to the split point. Intermediate data will then be sent to the server. After receiving smashed data from the client, the server will finish the remaining forward propagation, calculating function loss then back-propagate until the split point. As the server's weights are updated, these updates will be sent back to the client, where backpropagation is continued and finalized. This completes a *local* epoch. After all the clients have finished their local epochs, they send their updated weights to the server, which will aggregate these weights using Federated Averaging as follow:

$$W_{global} = \frac{1}{N}\sum_{i=1}^{N} W_{local_i} \quad (1)$$

The *global* model will then be distributed to all the clients, which completes a *global* epoch. This process is repeated for $N_e$ times, where $N_e$ is the number of *global epochs*, specified in the **Pre-training phase**.

## 3. Experiments and Result Analysis

### 3.1 Experimental Settings

For object detection, we choose YOLOv1 with a pre-trained ResNet-18 as the backbone. The split point is positioned after first pool layer of the ResNet backbone. Both the server and the clients are on a single machine with a RTX3090 GPU, running Ubuntu 20.04, CUDA 11.4, Python 3.8.10, and PyTorch 1.10.0.

The models are trained and validated with PascalVOC 2007 and 2012 datasets. To emulate users' data, we divide the dataset randomly into $N$ equal portions, where $N$ is the number of clients, varied from 3 to 7. Each client $i$ is assigned with data portion $D_i$. We conduct two experiments:

1. Experiment 1: The model is trained under two scenarios: (1) each client is trained separately and (2) all clients are trained using OctoFedS; and compare the mean Average Precision (mAP) results.

2. Experiment 2: We divided the dataset for 6 clients but varying the number of clients that do not participate in training (inactive).

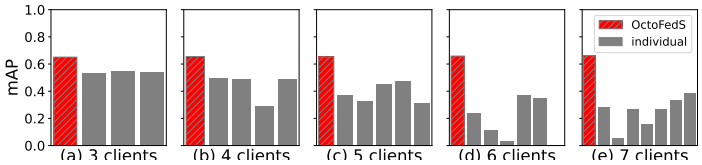


Figure 2: mAP results of model YOLOv1 trained with OctoFedS and the number of clients ranging from 3 to 7.

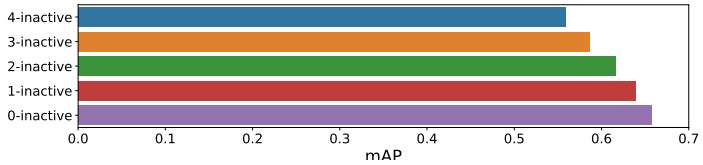


Figure 3: mAP results of OctoFedS when fixing data division and varying the number of inactive clients.

All training is done over 100 epochs using Stochastic Gradient Descent (SGD) with a learning rate of 0.001. For fair comparison, each training setting is repeated three times and the average results are reported.

### 3.2 Performance Analysis

**Experiement 1:** As shown in Figure 2, under the first scenario where clients are trained independently, mAPs are generally poor, reaching only a maximum of around 0.5. Also, the performances in this case decline when the number of clients increases. This stems from the fact that the dataset when divided into smaller fractions would yield imbalance distributions, in which classes might be unevenly available or missing in each of the clients. This indeed matches realistic conditions where data from different users are likely to be incomplete. The unavailability of certain classes in training data leads to failure to detect objects of those categories, which explains the low performances of individual clients. In contrast, implementing OctoFedS tackles the problem by applying FL to a distributed system of server and clients. mAPs achieved by OctoFedS are substantially higher, peaking at about 0.66, and consistent even under different client number settings. This proves the usefulness of FL that enables distributed learning, in which different participants can benefit from others' data without posing privacy risks. With this method, even though some classes might be absent in one client, the presence of them in other clients would help to re-balance this non-uniform behavior, eventually producing a more generalized global model for every participant.

**Experiment 2:** In the previous experiment, the results of OctoFedS remain consistent even though the number of clients increases. This is because even though the dataset of each client may shrink, the amount of data in the system remains the same. In this experiment, as the number of inactive clients decreases, the system has more and more data, which results in better training results as shown in Figure 3. However, it is worth noticing that the lowest mAP achieved by 4-inactive (approx. 0.5) is still better than those of individual clients trained in Experiment 1. Overall, it can be observed that performances were significantly improved with OctoFedS.

## 4. Conclusion

In this study, we develop OctoFedS, a system that applies FL to improve the training performance for object detection tasks. When trained independently, clients perform poorly due to the lack of data, however this issue can be resolved by jointly training with FL, applied in OctoFedS. Furthermore, the application of SC in the system can help with better data privacy and computational efficiency.

## Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-01126, Self-learning based Autonomic IoT Edge Computing).

## References

[1] Y. Liu, A. Huang, Y. Luo, H. Huang, Y. Liu, Y. Chen, L. Feng, T. Chen, H. Yu, and Q. Yang, "FedVision: An Online Visual Object Detection Platform Powered by Federated Learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 13172–13179, apr 2020.

[2] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv*, pp. 1–7, 2018.

[3] J. Jeon and J. Kim, "Privacy-Sensitive Parallel Split Learning," in *Proceedings of International Conference on Information Networking (ICOIN)*, pp. 7–9, IEEE, jan 2020.

[4] C. Thapa, M. A. P. Chamikara, S. Camtepe, and L. Sun, "SplitFed: When Federated Learning Meets Split Learning," *arXiv*, 2020.

[5] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya : Continuous Learning of Video Analytics Models on Edge Compute Servers," in *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*, pp. 1–18, 2022.