Performance Analysis of Distributed Learning in Edge Computing on Handwritten Digits Dataset

Tinh Phuc Vo[1], Viet Anh Nguyen[2], Xuyen Bao Le Nguyen[2], Duc Ngoc Minh Dang[2](✉), and Anh Khoa Tran[1]

[1] Modeling Evolutionary Algorithms Simulation and Artificial Intelligence, Faculty of Electrical and Electronics Engineering, Ton Duc Thang University, Ho Chi Minh City, Vietnam
41702149@student.tdtu.edu.vn, trananhkhoa@tdtu.edu.vn

[2] Computing Fundamental Department, FPT University, Ho Chi Minh City, Vietnam
{anhnvse170371,xuyennlbse170455}@fpt.edu.vn, ducdnm2@fe.edu.vn

Abstract. Deep learning models often consist of millions or even billions of parameters, making it challenging to deploy them on devices with limited resources. Therefore, this study presents scenarios to assess the computational capability of edge devices to provide an evaluation of the learning performance of distributed learning methods. It focuses on using Deep Neural Network and the handwritten digit dataset (MNIST) in edge computing to evaluate the performance of distributed learning methods (no-offloading, full-offloading, split computing, and federated computing) in both ideal and realistic conditions. The performance evaluations are based on Precision, Recall, Accuracy, F1-score, and Estimated time complexity. The findings indicate that the full-offloading method achieved the highest performance in ideal conditions. However, in realistic situations, the split computing and federated computing methods performed better than the others.

Keywords: Edge Computing · Split Computing · Deep Neural Networks · computation offloading

1 Introduction

Deep Neural Networks (DNNs) have become increasingly popular in recent years due to their ability to learn and represent complex features in data. It consists of multiple layers of interconnected nodes used to process and transform input data and generate output predictions. By utilizing DNNs in mobile devices, we can develop predictive models that analyze user behavior and learning patterns, then generate personalized recommendations for individual users. However, the processing power and memory requirements of DNNs are significant, which is a challenge for resource-constrained mobile devices. Mobile Edge Computing

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2023
Published by Springer Nature Switzerland AG 2023. All Rights Reserved
N.-S. Vo and H.-A. Tran (Eds.): INISCOM 2023, LNICST 531, pp. 159–169, 2023.
https://doi.org/10.1007/978-3-031-47359-3_12

(MEC) was first proposed in 2014 to reduce latency and improve the performance of mobile applications by processing data closer to the end user rather than in a centralized data center. MEC can help to offload some of the processing requirements of DNNs from mobile devices to the edge cloud that allows faster and more efficient processing of user data and enables new and innovative applications for self-studying systems in mobile devices. The current approach would focus on the "Machine learning" branch. MEC is predicted to promote self-learning approaches in the "Distributed computing methodologies" branch with less human intervention in processing input data. This research represents the performance analysis of 4 methods.

The rest of the paper is organized as follows. Section 2 provides an overview of related works. The main model used in this research is outlined in Sect. 3, and Sect. 4 presents evaluations of different approaches based on the selected standards derived from the experiments. Finally, Sect. 5 offers a conclusion.

2 Related Works

There are continuous new technologies and a list of updated research studies to optimize the user's experiment with higher speed and lower latency, increased capabilities and coverage, enhanced network reliability, efficient spectrum utilization, etc. This section focuses on recent research studies to overview the ways human beings reduce the computational complexity of DNNs. Some lightweight models such as MobileNets [1–3] are specially designed with very small, low latency models to easily match with resource-constrained devices. A different approach to building a small DNNs model is compressing a large model [4] has been proposed in the literature. The compression model changes the initial structure of DNNs by trying to remove parameters that are not crucial to model performance. Another approach, called Early Existing (EE) [5,6], adds early exit points after hidden layers of DNNs to give the chance for inputs to be classified early before reaching the final model's exit point. EE provides a "sub-branch" into the DNNs models so that full computation of the model can be halted, and the prediction result can be returned earlier than traditional ones - if the back is not necessary and it is highly confident about the prediction. Besides, there are numerous papers on distributed learning methods [6–10].

3 System Model

Figure 1 presents 4 approaches that this paper focuses on. With no-offloading (Fig. 1a), the entire DNN runs on the mobile device itself, using its local resources. This can be beneficial in situations where there is limited connectivity or where the privacy of the data is a concern. However, running the DNNs locally can be slow and resource-intensive, leading to increased power consumption and reduced battery life. Full-offloading (Fig. 1b) refers to transferring some of the computational tasks from the mobile device to a more powerful resource, such as a cloud or a remote server. This approach can be used to speed up the

computation of DNNs and reduce the energy consumption of mobile devices. There are also hybrid offloading approaches, where some parts of the DNN are run on the mobile device, and others are offloaded to the MEC server: Split Computing (Fig. 1c) and Federated Computing (Fig. 1d). This can balance the benefits and drawbacks of both no-offloading and full-offloading, resulting in improved performance and reduced power consumption.

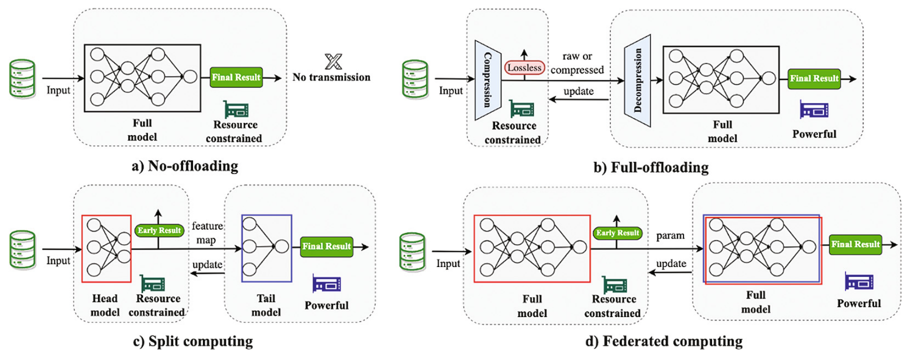

Fig. 1. Diagram of different learning approaches.

Full-Offloading. The entire DNN is offloaded to the MEC server, which runs the DNNs on its resources. This can significantly speed up the DNN, as the MEC server typically has more computing power and memory than the mobile device. However, full offloading requires a stable and reliable network connection, which may not always be available.

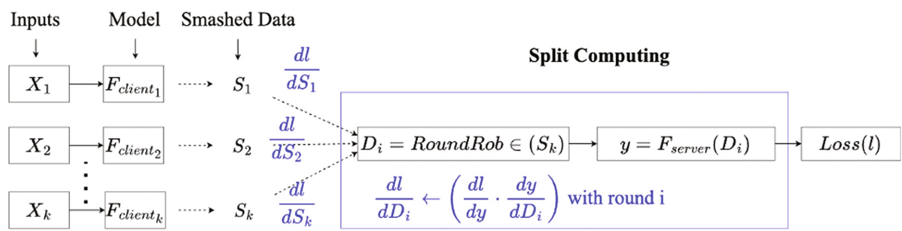

Fig. 2. Diagram of Split Computing.

Split Computing. Each head model network is a model that is cut in half as $F = F_{client} + F_{server}$ with the same output and derivative $\frac{\mathrm{d}l}{\mathrm{d}S}$ to make the data broken and then passed to train a deep server network. A DNN can be

defined as a function F (Fig. 2), where the device is the head model and the edge server is the tail model, which can be described as a sequence of D_i at each time point i is a S_k (partial aggregation method) of the values after the head model's calculation and reduction of the necessary parameters and smaller than the input X_k. Then, calculate the loss $Loss(l)$) at the tail model (F_{server}). Split learning has the potential to reduce the computational and energy requirements of the local applications as they only need to perform a partial computation.

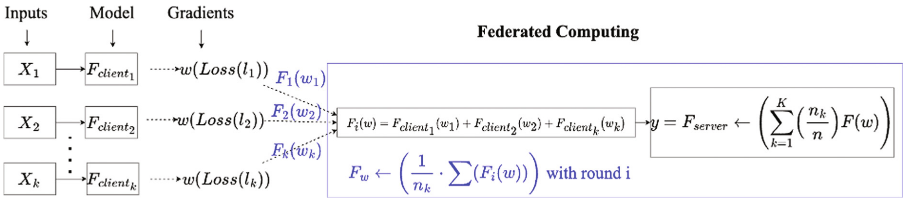

Fig. 3. Diagram of Federated Computing.

Federated Computing. Federated Computing (FC) enables multiple devices to collaborate on a shared computation task without requiring them to share raw data. FC is based on the "FedAvg" federated averaging method (Fig. 3). Let $n = 1, 2, 3, ..K$ denote end devices while F(w) shows the loss in device k. The goal of each round of $FedAvg$ is to reduce the global model's objective w, which is just the total of the weighted average of the local device loss. A random device is chosen. Each device receives the model and executes the SGD on its loss function. After that, it transmits the learned model to the F_{server} for model aggregation. The server then uses the average of these local models to update its global model. The above process of local training-global aggregation is repeated for multiple rounds until achieving a certain level of accuracy. Each device performs local computation on its data and shares only the parameter with a central server.

4 Performance Evaluation

We have established an edge server environment consisting of a central server and ten end devices. The server is configured for training using the Linux operating system, Python 3.7, and TensorFlow 2.10 with the CuDNN library for GPU utilization during training. Full channel conditions are assumed, and available bandwidth between the devices and the server is ignored during simulation. The experiments are conducted in a loop environment similar to that of real testing devices. This project is building a 6-layer MLP model for the classification task at hand. The input data shape and the number of layers are passed as arguments to the nodes. The loss function is *sparse_categorical_crossentropy*. The

Multilayer Perceptron (MLP) model with 60,970 computational parameters is used to train the MNIST dataset, which includes 50,000 training samples, 10,000 validation samples, and 10,000 test samples. The training samples are uniformly divided among the ten devices without overlapping. The entire test dataset is available on the server. The SGD function is used as the optimization tool with an initial learning rate of 0.01, which gradually decreases over the course of the task's iterations. For faster experimentation and development, this project uses a High-Performance Computing (HPC) platform built on the Google Cloud Workspace platform, using the Colab Notebook to compile with a partner GPU for computation. This research runs both device and server on the same HPC platform provided. Devices and server scenarios change from no-offloading, full-offloading, and split computing, with data transfer being reduced in the number of times compared to full transfer. Finally, the federated learning principle is that only the MLP model parameters are shared.

Our research is evaluated based on the correlation between model predictions and actual results. The overall result of the system is evaluated using the following metrics: Precision, Recall, Accuracy, F1-score, and Estimated Training Time. The research initially evaluated the use of scenario-based and random models for devices using the Round Robin Scheduling Algorithm to ensure equal participation of all 10 members at each parameter model time point. The choice of using the MNIST dataset for training is popular and well-suited for DNN models, therefore, the training process consisted of approximately 100 iterations. In addition, our research performed training on the main purpose MNIST dataset and consulted the Fashion MNIST dataset, which is a dataset of 60,000 28×28 grayscale images belonging to 10 fashion and clothing categories, for example. such as shoes, t-shirts, and skirts, ... Mapped data of all integers 0–9 corresponding to MNIST-like class labels (top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, boot) were used up in all 10 classes for the task of training in Figs. 4, 5, 6, and 7. The experiment consists of 10,000 images equally divided into 1,000 images for each class to compare the final prediction performance at the server (Figs. 8 and 9). This dataset can be used as an optional alternative to MNIST for evaluating machine learning algorithms, as it has the same 28×28 image size, 2D data format training, validation, and testing split.

Considering the results of Fig. 4 and with positive criteria, device performance differs significantly between scenarios and there is no overfitting. In the theoretical condition (no noise), no-offloading only gives 92% validation data compared to the case of full-offloading, split computing, and federated computing achieved 100%, 98%, and 98%, respectively. Specifically, no-offloading with no edge server involvement for more learning (edge device data is independent) makes the performance curve grow slower by only 90% at the 10th round of communication. For Fashion data MNIST (Fig. 5) without input noise, within 100 communication rounds the training to find the best accuracy takes longer than with the MNIST data, and there is a clear difference between the scenarios. In general, the lowest no-offloading and full-offloading work best when a complete model is learned with all data free of noise.

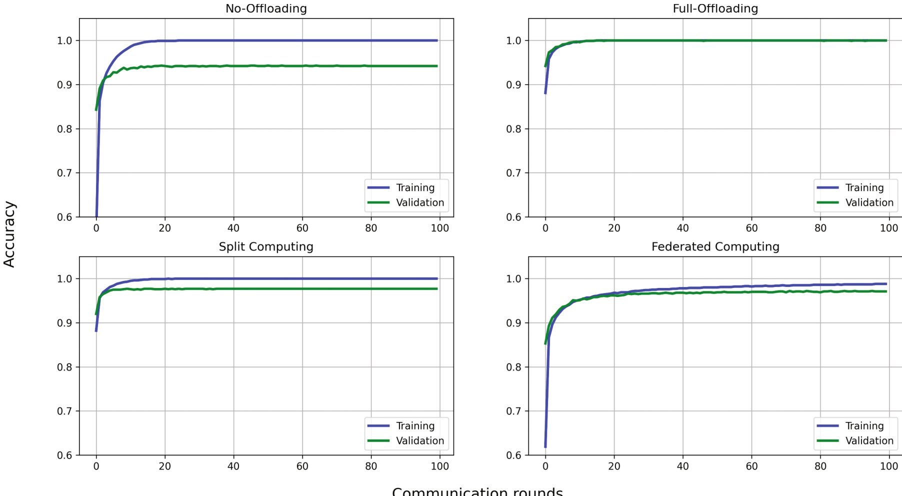

Fig. 4. The average accuracy with 10 devices during the MNIST data training process in a theoretical condition (without noise).

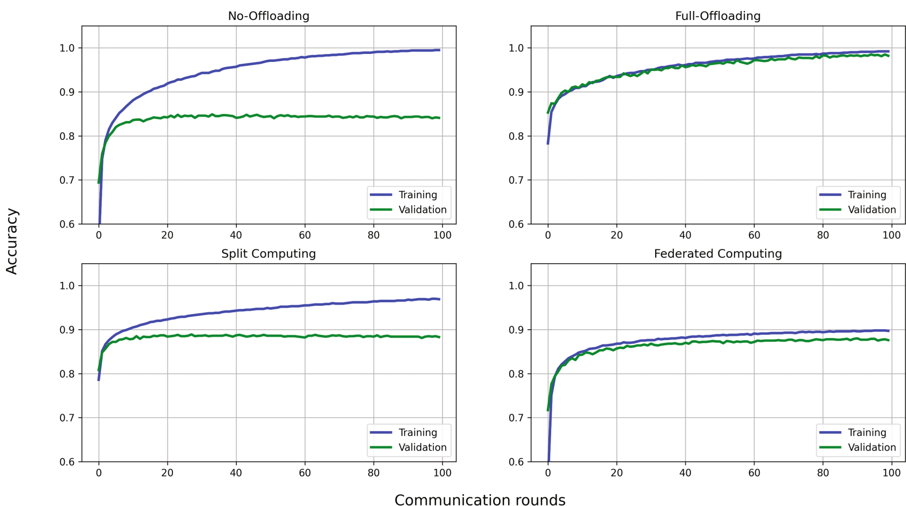

Fig. 5. The average accuracy with 10 devices during the Fashion MNIST data training process in a theoretical condition (without noise).

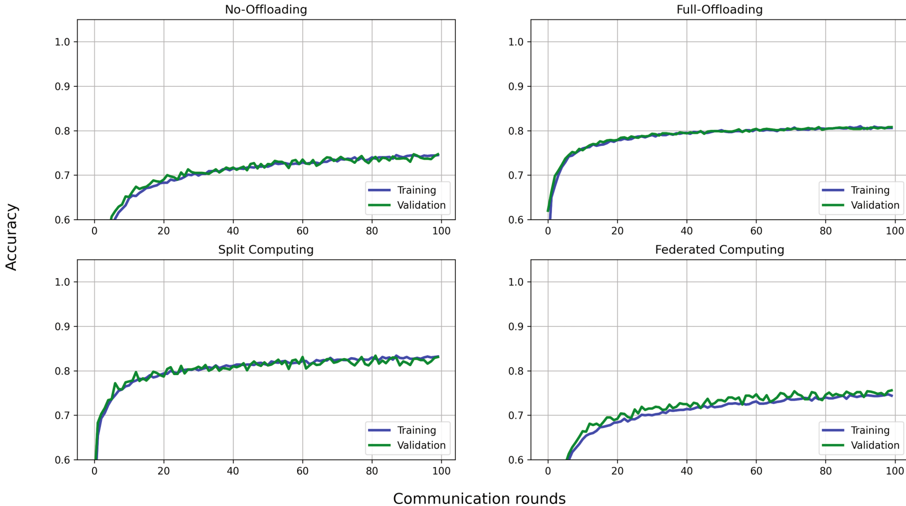

Fig. 6. The average accuracy with 10 devices during the MNIST data training process in realistic conditions (with Gaussian noise).

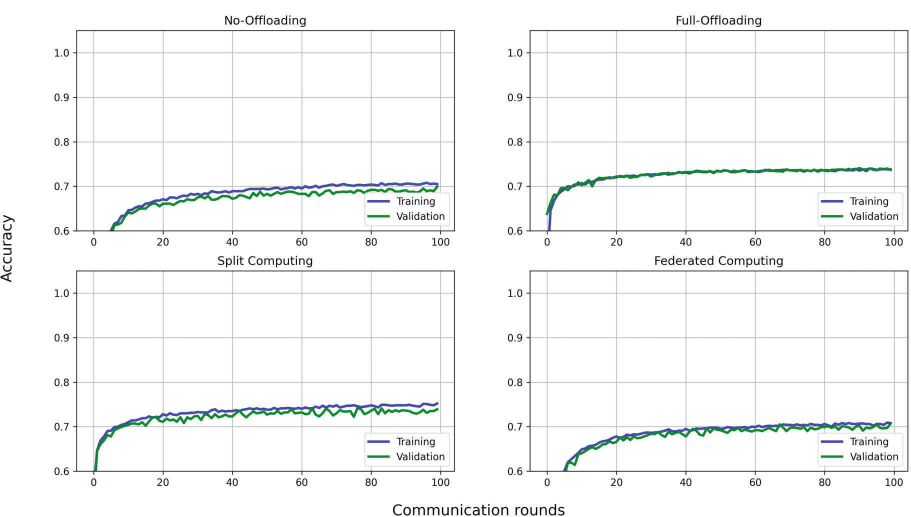

Fig. 7. The average accuracy with 10 devices during the Fashion MNIST data training process in realistic conditions (with Gaussian noise).

However, considering the actual conditions (Figs. 6 and 7), we assume that the input data has random Gaussian noise in the 2D image, just as in the real case the data will be lost through the transmission channel, the result will be lower than 15% on the same test conditions, within 100 rounds of communication. Split computing shows the greatest advantage when the validation data reaches 82% with MNIST data and Fashion MNIST data is about 75%. In the remaining cases of no-offloading, federated computing is not higher than 75% because there are many sub-model parameters on personal data, then full-offloading reaches 80% with MNIST data, and Fashion MNIST data is about 75 % because all data is learned on the same unified model structure.

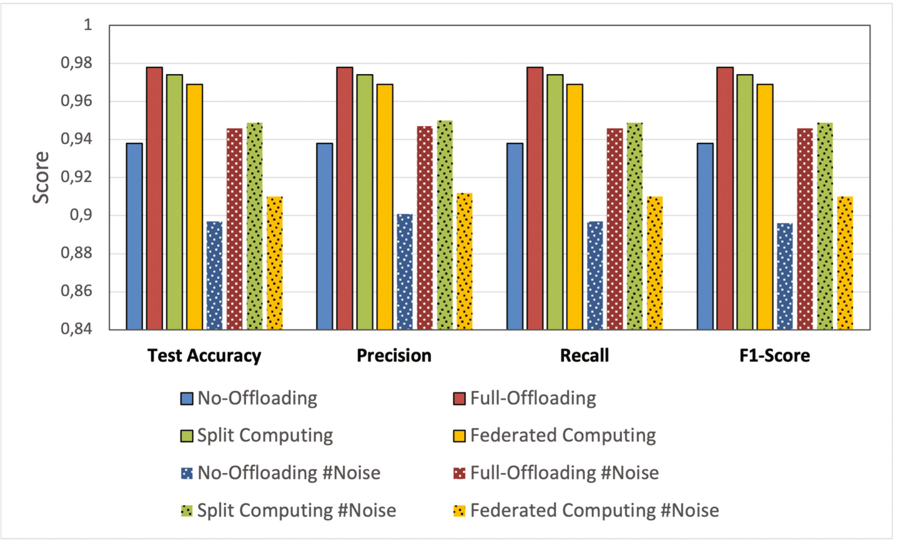

Fig. 8. The performance of the edge server during the MNIST data training process in ideal and realistic conditions.

In comparing the performance of each scenario at each communication round and the final test with the test data set located at the edge server, we choose the frequency of the performance evaluation test to be 50 times out of a total of 100 rounds. Because of some resource optimization requirements, testing does not take place continuously. In Fig. 8, the full-offloading scenario gives the highest Fig. 97.8% and when there is noise, split computing gives the final result 95% higher than the remaining scenarios due to taking advantage of device performance and reducing resources transferred to the edge server. Second is full-offloading 94.6% but not feasible if privacy is required. Federated computing is the 3rd choice 91.3% in the condition that it makes sure the cloud device meets the recommended configuration. The performance tests are still stable above 90% and negligible compared to no-offloading 89.4% of the training plan training on the device and ignoring the edge server, the performance is not higher than the

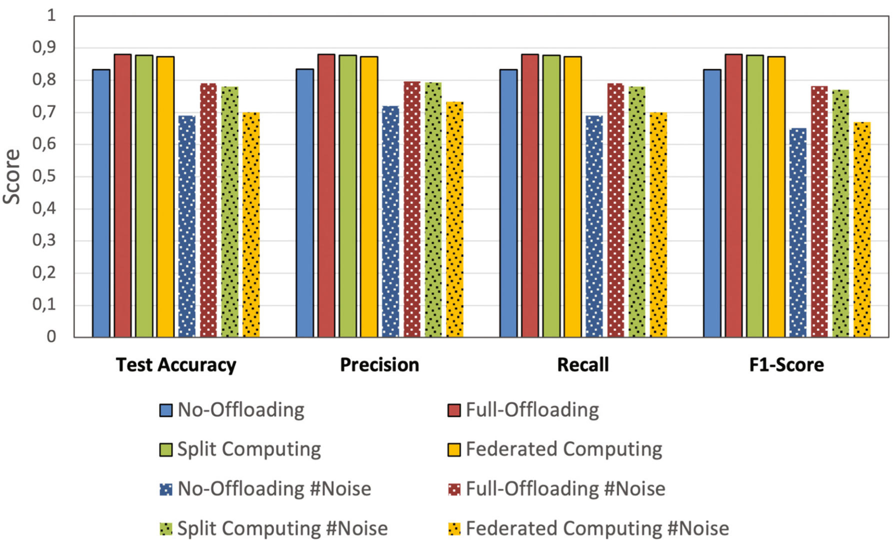

Fig. 9. The performance of the edge server during the Fashion MNIST data training process in ideal and realistic conditions.

link configuration, which separates the training process. It is possible to understand that partial loss recovery from interference per device by serial training at the server helps to maintain accuracy in split computing, regardless of edge devices, and interference from many devices. In Fig. 9, Fashion MNIST data also have no major difference in order and tasks compared to MNIST except the test results will be lower than MNIST due to slower convergence at the round 100th communication. The full-offloading, split computing and federated computing scenarios reached 89.8% in the condition without noise. When there was noise the full-offloading and split computing scenarios reach 79%, higher than the maintaining scenarios.

In Fig. 10, the average training time for the entire process on both the local device and the edge server is shown in detail. Split computing has the lowest total training time at 8.84 s in the Sequential simulation, and the on-device training time is lower than the local training cases by about 9.7 s for the setup. SC has the shortest one-round communication time in Sequential comparison with local cases for two main reasons. In the local comparison, split computing focuses only on providing smashed data without considering the calculation of metrics (accuracy, precision, recall, F1-score). In global comparison, split computing trains with a model that is half the size of 10 devices are sent to the server (full-offloading) in 6.49 s. Full-offloading takes about 14.01 s. In the case of parallel simulation, federated computing transmits the model parameters that have the shortest communication time of 1.2 s/round.

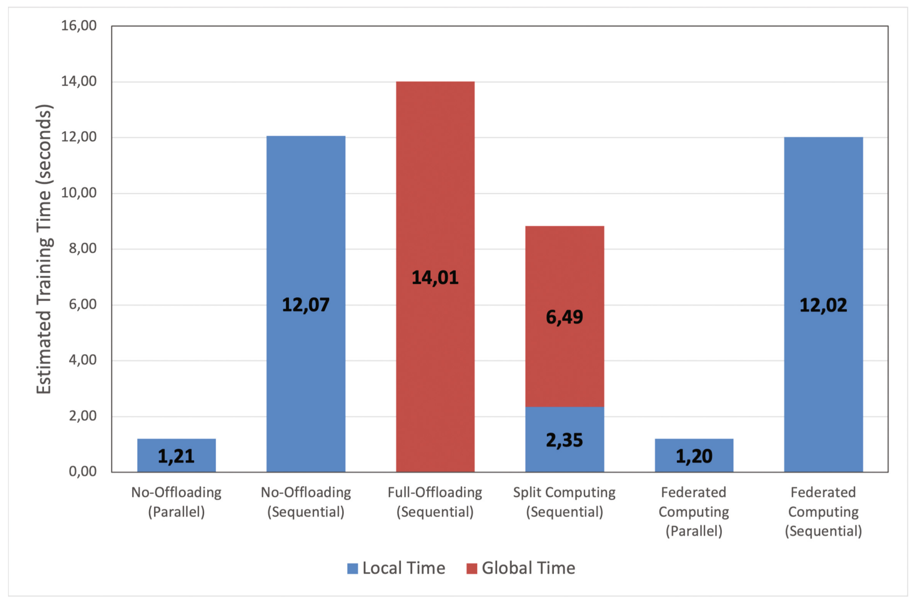

Fig. 10. Comparison chart of training time across learning scenarios.

5 Conclusion

This research paper investigated and analyzed the learning performance of distributed learning in the MEC environment. We have explored several methods such as split computing, federated computing, full-offloading, and no-offloading to improve the learning speed of DNNs in the mobile environment. These results can be applied to practical applications in the future, improving user experience and optimizing computing resources.

References

1. Howard, A.G., et al.: MobileNets: efficient convolutional neural networks for mobile vision applications (2017). https://arxiv.org/abs/1704.04861v1
2. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: MobileNetV2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 4510–4520 (2018). https://arxiv.org/abs/1801.04381v4
3. Howard, A., et al.: Searching for MobileNetV3. In: Proceedings of the IEEE International Conference on Computer Vision, vol. 2019-October, pp. 1314–1324 (2019)
4. Cheng, Y., Wang, D., Zhou, P., Zhang, T.: Model compression and acceleration for deep neural networks: the principles, progress, and challenges. IEEE Signal Process. Mag. **35**(1), 126–136 (2018)
5. Scardapane, S., Scarpiniti, M., Baccarelli, E., Uncini, A.: Why should we add early exits to neural networks? Cogn. Comput. **12**(5), 954–966 (2020)

6. Matsubara, Y., Levorato, M., Restuccia, F.: Split computing and early exiting for deep learning applications: survey and research challenges. ACM Comput. Surv. **55**, 1–30 (2022). https://doi.org/10.1145/3527155
7. Jeong, H.J., Jeong, I., Lee, H.J., Moon, S.M.: Computation offloading for machine learning web apps in the edge server environment. In: Proceedings - International Conference on Distributed Computing Systems, vol. 2018-July, pp. 1492–1499 (2018)
8. Wang, X., Han, Y., Leung, V.C., Niyato, D., Yan, X., Chen, X.: Convergence of edge computing and deep learning: a comprehensive survey. IEEE Commun. Surv. Tutor. **22**, 869–904 (2020)
9. Duan, Q., Hu, S., Deng, R., Lu, Z.: Combined federated and split learning in edge computing for ubiquitous intelligence in internet of things: state-of-the-art and future directions. Sensors **22**(16), 5983 (2022)
10. Ceballos, I., et al.: SplitNN-driven vertical partitioning (2020). https://arxiv.org/abs/2008.04137v1