Dynamic Split Computing for Efficient Deep Edge Intelligence

Arian Bakhtiarnia [1] **Nemanja Milošević** [2] **Qi Zhang** [1] **Dragana Bajović** [3] **Alexandros Iosifidis** [1]

Abstract

Deploying deep neural networks (DNNs) on IoT and mobile devices is a challenging task due to their limited computational resources. Thus, demanding tasks are often entirely offloaded to edge servers which can accelerate inference, however, it also causes communication cost and evokes privacy concerns. In addition, this approach leaves the computational capacity of end devices unused. Split computing is a paradigm where a DNN is split into two sections; the first section is executed on the end device, and the output is transmitted to the edge server where the final section is executed. Here, we introduce dynamic split computing, where the optimal split location is dynamically selected based on the state of the communication channel. By using natural bottlenecks that already exist in modern DNN architectures, dynamic split computing avoids retraining and hyperparameter optimization, and does not have any negative impact on the final accuracy of DNNs. Through extensive experiments, we show that dynamic split computing achieves faster inference in edge computing environments where the data rate and server load vary over time.

1. Introduction

The combination of deep learning and Internet of Things (IoT) has tremendous applications in fields such as healthcare, smart homes, transportation and industry (Ma et al., 2019). However, deep learning models typically contain millions or even billions of parameters, making it difficult to deploy these models on resource-constrained devices. One solution is to offload the computation to an edge or cloud server (Wang et al., 2020), as shown in Figure 1 (b). However, since the size of the inputs to deep learning models can be massive, particularly images and videos, this approach consumes a lot of bandwidth and energy, and leads to delays. Moreover, even though IoT devices are limited, they still possess computational capabilities that remain unused when the entire computation is offloaded, and utilizing these capabilities would reduce the load on the servers. In addition, in applications that process personal data such as health records, or in audio or visual streams with voice activity or human presence, privacy regulations such as European Union's GDPR (European Commission) or United States' HIPAA (Centers for Medicare & Medicaid Services, 1996) may apply. These regulations typically forbid direct access to non-anonymized data, leaving the options to either anonymize the data at the cost of additional computation and higher latency, or process the data at the source.

Split computing, depicted in Fig. 1 (c), alleviates these issues by splitting the deep model into a *head* section and a *tail* section (Matsubara et al., 2021). The head model is executed on the device, and its output (the intermediate representation at that particular layer of the deep network) is transmitted to the server, then processed by the tail model to obtain the final output. In a way, split computing is a *partial offloading* of the computation, as opposed to the *full-offloading* approach. Another benefit of split computing over full-offloading is that it can be used as a privacy preserving technique since intermediate representations are being transmitted instead of the actual inputs, and the original inputs cannot be easily reconstructed from the intermediate representations (Jeong et al., 2018). In addition, split computing can be combined with early exiting in order to obtain an early result on the device (Scardapane et al., 2020; Matsubara et al., 2021), as illustrated in Figure 1 (c), which is useful when transmission takes longer than expected.

Since split computing aims to decrease the communication cost, *natural bottlenecks*, which are the layers of the deep network where the size of the intermediate representation is smaller than the input size, can be used as splitting points for deep learning models. In this paper, we show that unlike older popular models, state of the art models such as EfficientNet (Tan & Le, 2019; 2021) possess many natural bottlenecks. Based on this fact, we propose a method called *dynamic split computing* where the best splitting point is automatically and dynamically determined based on input and channel conditions, as shown in Figure 1 (d). Since the underlying deep learning model is not modified, dynamic

[1]DIGIT, Department of Electrical and Computer Engineering, Aarhus University, Denmark [2]Faculty of Sciences, University of Novi Sad, Serbia [3]Faculty of Technical Sciences, University of Novi Sad, Serbia. Correspondence to: Arian Bakhtiarnia <arianbakh@ece.au.dk>.

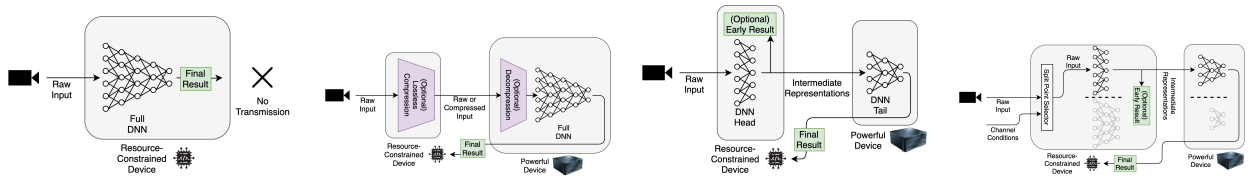

Figure 1. Overview of (a) no-offloading; (b) full-offloading; (c) split computing; and (d) dynamic split computing approaches.

split computing can be used as a plug-and-play method, meaning it can be employed without domain knowledge about the particular deep learning models that are being used. It is important to note that dynamic split computing is a complimentary efficient inference method that can be used in combination with other approaches, including model compression techniques such as pruning and quantization (Choudhary et al., 2020), as well as dynamic inference methods such as early exiting (Bakhtiarnia et al., 2021).[1]

2. Related Work

Several approaches for speeding up the inference of deep neural networks (DNNs) on resource-constrained devices exist in the literature. *Local computing* performs the entire computation on the device, yet modifies the architecture of the neural network in order to decrease the required computation, while causing a minimal negative impact on the accuracy. *Lightweight models* such as MobileNet (Howard et al., 2017; Sandler et al., 2018; Howard et al., 2019) are specifically designed to be deployed on such limited devices, whereas *model compression* techniques (Cheng et al., 2018) alter existing architectures in order to make them more lightweight, for instance, *pruning* removes the less impactful parameters (weights) of the neural network; *quantization* uses less bits to represent each parameter (Liang et al., 2021); and *knowledge distillation* aims to train a more compact model to reproduce outputs similar to a given larger neural network (Gou et al., 2021).

Dynamic inference methods (Han et al., 2021) can alter the architecture of existing neural networks to adapt their inference time at the cost of accuracy, meaning they will produce more accurate outputs the longer they are allowed to execute. Various approaches to dynamic inference exist, such as *early exiting* (Scardapane et al., 2020), where early exit branches are added after intermediate layers of a DNN that produce an output similar to the final output; *layer skipping* (Graves, 2016; Banino et al., 2021; Wang et al., 2018), where the execution of some of the DNN layers are skipped; and *channel skipping* (Gao et al., 2019), where less impactful channels of convolutional layers are ignored.

[1]Our code is available at `https://gitlab.au.dk/maleci/dynamicsplitcomputing`.

Even with local computing techniques, many high-performing DNNs exceed the computational capacity of devices, especially when the output is expected within a strict deadline. In such cases, the computation can be offloaded to external servers. When the computation of a DNN is offloaded, the inputs must be transmitted from a device to a server, yet this can introduce massive delays during data transmission, particularly when the input size is large, which may defeat the original purpose of speeding up the computation. This has led to a recent emerging paradigm called *edge computing* (Abbas et al., 2018) where the computation is offloaded to *edge servers* located much closer to end devices compared to cloud servers which are often located in data centers. Even though edge computing reduces the transmission delay, it still has some drawbacks. First, since the original inputs are being transmitted over a network, privacy issues arise. Furthermore, since typically multiple end devices are connected to the same edge server, if all of them offload their computation simultaneously, the edge server may experience a high load while the computational resources of each end device remain unused.

Split computing (Matsubara et al., 2021) (also known as *collaborative intelligence*) is an alternative approach that provides a balance between local computing and full-offloading, where some layers of the DNN are executed on the end device and the intermediate output is then sent over to the edge server where it is processed by the rest of the DNN layers. When the splitting point is chosen such that the size of the intermediate representation is lower than the input size, the transmission delay will consequently be lower than that of full-offloading.

However, not all deep learning models possess such natural bottlenecks, and even if they exist, they may be located in the final layers of the network where the bulk of the computation has already been carried out, and therefore it would not be sensible to offload the remaining computation. For instance, widely used models such as ResNet (He et al., 2016) and Inception (Szegedy et al., 2016) do not contain natural bottlenecks in their early layers (Matsubara et al., 2021). In such cases, *bottleneck injection* can be used, where the architecture of the network is modified to artificially insert a bottleneck (Matsubara et al., 2021). However, this approach requires time-consuming operations such as retraining the

model and optimizing hyperparameters such as the size of the inserted bottleneck. Furthermore, there is no guarantee that the new architecture can obtain an accuracy comparable to that of the original architecture, particularly when a limitation such as a small bottleneck is imposed. Therefore, bottleneck injection is far from ideal.

3. Dynamic Split Computing

We assume a trained high-performing DNN is to be deployed on a device with access to a server, where the data rate of the communication channel and the number of inputs in the batch (batch size) may vary. The variations in the data rate may be due to fluctuations in wireless channel state or traffic congestion, and the variations in batch size may occur due to a different workload at different times. The goal of our method is to optimize the end-to-end inference time by dynamically detecting the best splitting point for a given DNN based on the communication channel state and batch size. Since we aim to design our method in a "plug-and-play" manner, such that it can be deployed in edge computing systems without creating new trade-offs involving the accuracy or the hassle of retraining, we avoid altering the underlying architecture or any lossy compression techniques that may affect the accuracy of the final result.

Formally, neural networks can be formulated as $f(x) = f_L(f_{L-1}(\dots f_1(x)))$ where x is the input, L is the total number of layers in the neural network and f_i is the operation performed at layer i. The intermediate representation at layer i, which is the output of the i-th layer is recursively formulated as $h_i = f_i(h_{i-1})$ where $h_0 = x$ is the input. Based on this notation, with split computing at layer j, the head and tail parts of the DNN are denoted by $f^h(x) = f_j(f_{j-1}(\dots f_1(x)))$ and $f^t(h_j) = f_L(f_{L-1}(\dots f_{j+1}(h_j)))$, respectively, and h_j is the intermediate representation that is transmitted.

The first step is to find the natural bottlenecks of the DNN by calculating the compression ratio $c_l = |h_l|/|x|$ for each layer l where $|h_l|$ and $|x|$ denote the size of intermediate representation at layer l and the input size, respectively. If $c_l < 1$, layer l is a natural bottleneck of the DNN. However, not all natural bottlenecks are useful in split computing. We define $T^h_{i,j}$ and $T^t_{i,j}$ as the inference time from layer i up to and including layer j ($i < j$) of the deep neural network measured on the device and the server, respectively. When layers m and n ($m < n$) have the same compression ratio, in other words when $c_m = c_n$, the total end-to-end inference time with split computing at layer m and layer n are

$$T_m = T^h_{1,m} + c_m T_{\text{FO}} + T^t_{m+1,n} + T^t_{n+1,L}, \quad (1)$$

$$T_n = T^h_{1,m} + T^h_{m+1,n} + c_n T_{\text{FO}} + T^t_{n+1,L}. \quad (2)$$

where T_{FO} is the transmission time of the entire input in full-offloading. Assuming the computational resources of the server are greater than that of the device, then $T^h_{m+1,n} > T^t_{m+1,n}$, thus it is favorable to choose the earlier layer as splitting point. Consequently, only natural bottlenecks with compression ratio lower than all previous natural bottlenecks are useful. We call such bottlenecks *compressive*. Compressive natural bottlenecks are defined by

$$C = \{j | c_j < 1, c_j < c_i \; \forall i < j\}. \quad (3)$$

The total end-to-end inference time for a given batch of inputs when the splitting point of the network is l is

$$T_l = T^h_{1,l} + \frac{D\, c_l}{r} + T^t_{l+1,L}, \quad (4)$$

where D is the data size of the original input, c_l is the compression ratio of the intermediate representation at layer l and r is the data rate of the communication channel. When inputs are images or video frames, the total load in bytes can be calculated as $D = BWHC$, where B is the batch size, W and H are the width and height of the images, and C is the number of channels in the images, for instance, $C = 3$ for color images and $C = 1$ for grayscale.

We define the end-to-end inference time in case of no-offloading as $T_L = T^h_{1,L}$ and in case of full-offloading as

$$T_0 = \frac{D}{r} + T^t_{1,L}. \quad (5)$$

Therefore, the optimal splitting point s_{opt} can be determined by optimizing for

$$s_{opt} = \underset{l \in \{0 \dots L\}}{\arg\min}(T_l). \quad (6)$$

Dynamic split computing finds the optimal split location for a given data rate and batch size based on Eq. (6). When full-offloading cannot be used, for instance, due to privacy requirements, the range is Eq. (6) is reduced to $\{0 \dots L-1\}$. Note that based on previous arguments, only compressive natural bottlenecks need to be investigated, therefore once all compressive natural bottlenecks are identified, we calculate the optimal splitting point for each batch size and data rate by measuring the inference time of head and tail models for each compressive bottleneck. It is important to note that the relationship between inference time of head or tail model and batch size is not strictly linear, therefore it needs to be measured for each batch size. Additionally, when the data rate is too low, it may not be sensible to use any form of offloading since it introduces too much delay. Therefore, dynamic split computing considers the no-offloading option alongside the optimal splitting point and switches between split computing and no-offloading when necessary.

Since different applications and environments may have different ranges for data rate and batch size and a unique pattern for their variations, we need a method to measure how beneficial dynamic split computing is in each

specific case. We define a scenario as a sequence of the state of the environment throughout time, i.e., $S = ((B_1, r_1), (B_2, r_2) \ldots, (B_T, r_N))$, where B_i and r_i are the batch size and data rate at time step i, respectively. The relative average gain of dynamic split computing in terms of end-to-end inference time over a specific method, for instance, static split computing at a specific location, can then be calculated by

$$G = \frac{1}{N} \sum_{1 \leq i \leq N} \frac{|T_{s_{opt}}(B_i, r_i) - T^{\text{SS}}(B_i, r_i)|}{T^{\text{SS}}(B_i, r_i)}, \quad (7)$$

where $T_{s_{opt}}(B_i, r_i)$ and $T^{\text{SS}}(B_i, r_i)$ are the end-to-end inference time using dynamic and static split computing, respectively, with batch size B_i and data rate r_i.

4. Results

We investigate 14 modern DNN architectures: seven variations of EfficientNetV2 (Tan & Le, 2021) and seven variations of EfficientNetV1 (Tan & Le, 2019). All these architectures were originally designed for image classification and have since been applied to various other problems such as speech recognition (Lu et al., 2020). The accuracy of these architectures on the ImageNet dataset (Deng et al., 2009) ranges from 77.1% to 85.7%.

First, we find the compressive natural bottlenecks for each architecture. The number of natural bottlenecks in these architectures ranges from 15 to 68, three to four of which are compressive. For comparison, VGG-16 (Simonyan & Zisserman, 2014), which is an older architecture, has only 5 natural bottlenecks. Subsequently, we find the optimal splitting point for each architecture in a wide range of states. We check data rates ranging from 1 MBps to 128 MBps and batch sizes of 1 to 64. Some larger models such as EfficientNetV2-L run into memory issues with large batch sizes, therefore, we reduce the maximum batch size to 32 or 24 in such cases. For the edge server, we use an Nvidia 2080 Ti GPU, and in order to simulate a resource-constrained device, we underclock the same type of GPU to 300 MHz (the normal GPU frequency is around 1800 MHz).

The results for the EfficientNetV1-B4 architecture are shown in Fig. 2, where for each state (data rate and batch size), the optimal split location derived based on Equation 6 is specified. It can be observed that each compressive bottleneck is an optimal split location in several states. Moreover, no-offloading is the optimal solution in some other states. Therefore, dynamically switching between split locations (as well as no-offloading) based on the state of the communication channel improves inference speed. This is also the case with the other 13 investigated architectures. The relative gain of dynamic split computing over split computing at a fixed location (block 10) for the EfficientNetV1-B4 architecture in terms of inference speed is shown in Fig. 3. This figure can be used to derive the gain of dynamic split computing compared to another method for a specific scenario based on on Equation 7. Notice that in states where split computing at block 10 is optimal, dynamic split computing swiches to this method and thus has no advantage over it, whereas dynamic split computing obtains some gain everywhere else by switching to a different method.

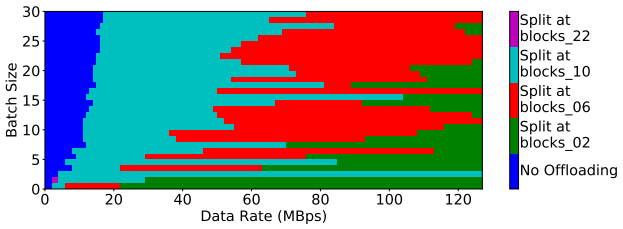

Figure 2. Optimal split location based on batch size and data rate for the EfficientNetV1-B4 architecture.

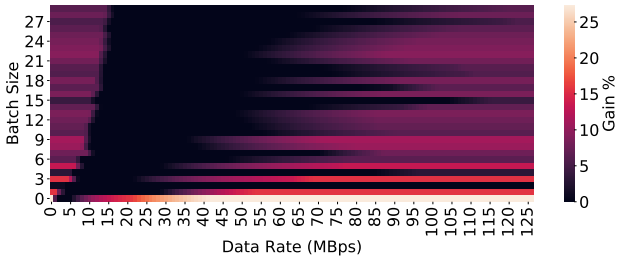

Figure 3. The relative gain of dynamic split computing in terms of end-to-end inference time over static split computing at block 10 in the EfficientNetV1-B4 architecture.

5. Conclusion

In this paper, we showed that dynamic split computing offers improvements in terms of inference time over both no-offloading and split computing with a fixed split location. Moreover, as opposed to full-offloading, dynamic split computing can decrease the computation load on the server by performing parts of the computation on the device. Finally, by transmitting intermediate representations instead of inputs, dynamic split computing circumvents privacy issues that arise when using full-offloading.

Acknowledgements

The work received funding by the European Union's Horizon 2020 research and innovation programme under grant agreement No 957337, and by the Danish Council for Independent Research under Grant No. 9131-00119B.

References

Abbas, N., Zhang, Y., Taherkordi, A., and Skeie, T. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2018. doi: 10.1109/JIOT.2017.2750180.

Bakhtiarnia, A., Zhang, Q., and Iosifidis, A. Multi-exit vision transformer for dynamic inference. *The 32nd British Machine Vision Conference (BMVC 2021)*, 2021.

Banino, A., Balaguer, J., and Blundell, C. Pondernet: Learning to ponder. *arXiv:2107.05407*, 2021.

Centers for Medicare & Medicaid Services. The Health Insurance Portability and Accountability Act of 1996 (HIPAA). Online at http://www.cms.hhs.gov/hipaa/, 1996.

Cheng, Y., Wang, D., Zhou, P., and Zhang, T. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136, 2018. doi: 10.1109/MSP.2017.2765695.

Choudhary, T., Mishra, V., Goswami, A., and Sarangapani, J. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 53(7):5113–5155, Oct 2020. ISSN 1573-7462. doi: 10.1007/s10462-020-09816-7. URL https://doi.org/10.1007/s10462-020-09816-7.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

European Commission. 2018 reform of EU data protection rules. Online at https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu_en.

Gao, X., Zhao, Y., Dudziak, L., Mullins, R. D., and Xu, C. Dynamic channel pruning: Feature boosting and suppression. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=BJxh2j0qYm.

Gou, J., Yu, B., Maybank, S. J., and Tao, D. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, Jun 2021. ISSN 1573-1405. doi: 10.1007/s11263-021-01453-z. URL https://doi.org/10.1007/s11263-021-01453-z.

Graves, A. Adaptive computation time for recurrent neural networks. *arXiv:1603.08983*, 2016.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. Dynamic neural networks: A survey. *arXiv:2102.04906*, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

Jeong, H.-J., Jeong, I., Lee, H.-J., and Moon, S.-M. Computation offloading for machine learning web apps in the edge server environment. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1492–1499, 2018. doi: 10.1109/ICDCS.2018.00154.

Liang, T., Glossner, J., Wang, L., Shi, S., and Zhang, X. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2021.07.045. URL https://www.sciencedirect.com/science/article/pii/S0925231221010894.

Lu, Q., Li, Y., Qin, Z., Liu, X., and Xie, Y. Speech recognition using efficientnet. In *Proceedings of the 2020 5th International Conference on Multimedia Systems and Signal Processing*, ICMSSP 2020, pp. 64–68, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450377485. doi: 10.1145/3404716.3404717. URL https://doi.org/10.1145/3404716.3404717.

Ma, X., Yao, T., Hu, M., Dong, Y., Liu, W., Wang, F., and Liu, J. A survey on deep learning empowered iot applications. *IEEE Access*, 7:181721–181732, 2019. doi: 10.1109/ACCESS.2019.2958962.

Matsubara, Y., Levorato, M., and Restuccia, F. Split computing and early exiting for deep learning applications: Survey and research challenges. *arXiv:2103.04505*, 2021.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Scardapane, S., Scarpiniti, M., Baccarelli, E., and Uncini, A. Why should we add early exits to neural networks? *Cognitive Computation*, 12(5):954–966, Sep 2020. ISSN 1866-9964. doi: 10.1007/s12559-020-09734-4. URL https://doi.org/10.1007/s12559-020-09734-4.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Tan, M. and Le, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/tan19a.html.

Tan, M. and Le, Q. V. Efficientnetv2: Smaller models and faster training. *arXiv:2104.00298*, 2021.

Wang, X., Yu, F., Dou, Z.-Y., Darrell, T., and Gonzalez, J. E. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

Wang, X., Han, Y., Leung, V. C. M., Niyato, D., Yan, X., and Chen, X. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 22(2):869–904, 2020. doi: 10.1109/COMST.2020.2970550.