Traffic-Efficient Split Computing Mechanism for Internet of Things

Hojin Yeom, Haneul Ko
Department of Electronics and Information Convergence Engineering
Kyung Hee University
Yongin-si, Gyeonggi-do, Korea
{hjy5323,heko}@khu.ac.kr

Sangheon Pack
School of Electrical Engineering
Korea University
Seoul, Korea
shpack@korea.ac.kr

***Abstract*—In the split computing approach, channel coding can be exploited to transmit the intermediate data, balancing the tradeoff between accuracy and traffic overhead. In this paper, we propose a traffic-efficient split computing mechanism (TESC) in which the IoT device decides whether to exploit channel coding and its error correction capability according to the channel condition to transmit the intermediate data to the edge cloud. Evaluation results show that TESC can reduce the traffic overhead by over 80% while maintaining the accuracy of the model at a high level compared to the retransmission-based scheme.**

***Index Terms*—Split computing, Split ML, distributed computing, inference, Internet of Things**

I. Introduction

Deep neural networks (DNNs) have emerged as a dominant machine learning technique for intelligent IoT applications due to their reliable and precise inference capabilities [1]. However, despite the increased computational capability of the latest IoT devices, they still face difficulty in achieving the desired inference latency when using complex DNN models [2]. As one of the promising solutions to reduce the inference latency, there is increasing interest in split computing [3]. For the split computing, the DNN model is split into head and tail models, and these models are allocated to IoT device and the computing node (e.g., edge cloud) at the network side, respectively. After that, IoT device can conduct the inference of the head model and transmit the output of the head model, called intermediate data, to the computing node. Subsequently, the computing node can use the intermediate data as an input of the tail model to derive the final result of the DNN model. After obtaining the result, the computing node returns it to IoT device.

Meanwhile, the intermediate data should be transmitted through a wireless link having relatively high error rate in IoT networks. Errors in the intermediate data can degrade the accuracy of the DNN model. To avoid the accuracy degradation, when detecting a packet error, the computing node can request retransmission of the intermediate data to IoT device for the error correction. However, if IoT device has poor channel conditions (leading to numerous and frequent errors), there is a significant increase of the traffic overhead.

Instead of the retransmission, to correct the errors with small traffic overhead, the channel coding can be utilized [4]. The channel coding is a digital communication technique that the sender adds a few bits of redundant data to the original data for the bit error correction at the receiver side. Unlike the retransmission method, which repeatedly sends the entire packet until it succeeds, when using the channel coding, IoT device transmits the channel coded packet (i.e., original data plus redundant data) once. Thus, when using the channel coding, the traffic overhead can be reduced compared to when using the retransmission.

Meanwhile, the ability how many bit errors can be corrected at the receiver side (called the error correction capability) depends on the number of redundant bits. Intuitively, to reduce the traffic overhead while correcting most bit errors, the number of redundant bits (i.e., error correction capability) should be decided by considering how many bit errors are expected to occur. For example, if the small number of bit errors is expected, even though only few redundant bits are added to the original data, most of errors can be corrected. On the other hand, under poor channel conditions, a higher error correction capability can be selected (i.e., more bits can be added to the original data) to correct more bit errors.

In this paper, we propose a traffic-efficient split computing mechanism (TESC). In TESC, to minimize the traffic overhead while achieving high accuracy of the model, IoT device decides whether to exploit the channel coding (i.e., Reed-Solomon (RS) code) and the error correction capability of the channel coding according to the channel condition to transmit the intermediate data to the edge cloud. Specifically, if the channel condition is better than a specific channel state threshold, IoT device does not use the channel coding. Otherwise, IoT device exploits the channel coding with the appropriate error correction capability for the channel condition. Evaluation results show that TESC can reduce the traffic overhead by over 80% while maintaining the accuracy of the model at a high level compared to the retransmission-based scheme.

The remainder of this paper is structured as follows: Section II introduces TESC; Section III presents and discusses the evaluation results; and Section IV presents the final conclusions.

II. Traffic-Efficient Split Computing Mechanism

Basically, we consider the system model where IoT device carries out the inference of the DNN model by using the split computing approach with the edge cloud. Specifically, to reduce the traffic overhead while maintaining high accuracy of the model, IoT device introduces TESC. The detailed procedure of TESC (i.e., flow chart) is shown in Figure 1.

First, IoT device carries out the DNN inference for the head model. This head inference produces intermediate data. After that, the channel condition C (i.e., signal-to-noise ratio (SNR)) is checked before transmitting the intermediate data to the edge cloud. The channel condition C is then compared with several pre-defined channel condition thresholds $\Delta = [\delta_0, \delta_1, \delta_2, \delta_4, \delta_8, \delta_{16}]$, where δ_t indicates the channel condition threshold with the error correction capability t. Through comparing C to Δ, TESC selects the appropriate error correction capability. For example, if $\delta_2 > C \geq \delta_4$, the error correction capability t is set to 4. Note that, when $C \geq \delta_0$, which means that the channel condition is good enough, the channel coding is not needed. Additionally, if t becomes 32, which means that there is no more δ_t in Δ to iterate, TESC moves on to the next step. Note that, since higher error correction capability indicates that more additional bits for the channel coding should be added to the original data, appropriate channel correction capability should be selected to correct the errors with small traffic overhead. After selecting the error correction capability t, TESC applies the channel coding to the intermediate data. Specifically, TESC exploits the RS code [5] with the error correction capability t, represented by RS(t).[1] After applying RS(t), IoT device transmits the channel coded intermediate data to the edge cloud. After receiving the intermediate data, the edge cloud corrects the errors in the intermediate data (if t is 0, since no channel coding is applied, skip correcting error). Then, by using the intermediate data as an input of the tail model, the edge cloud conducts the inference of the tail model. Then, the edge cloud transmits the inference result to IoT device.

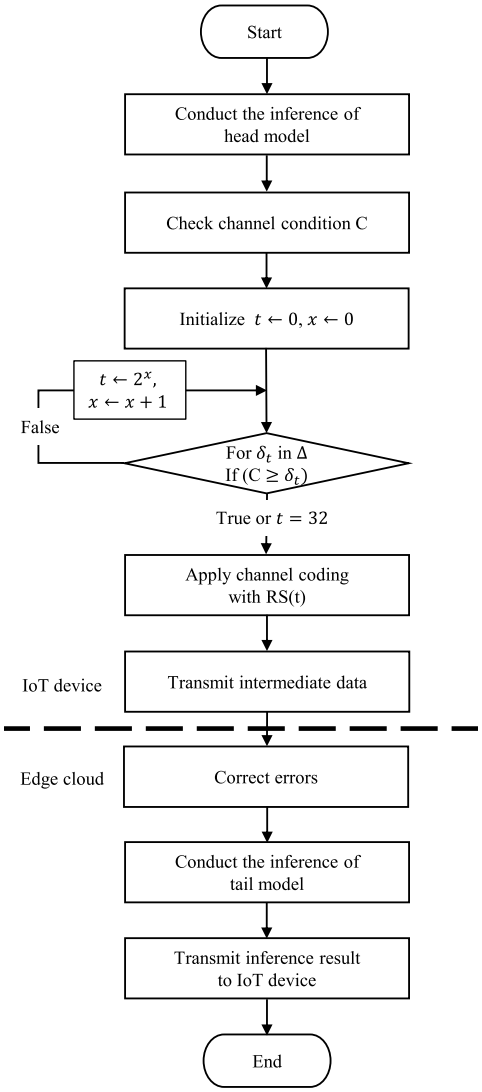


Fig. 1. Flow chart of TESC.

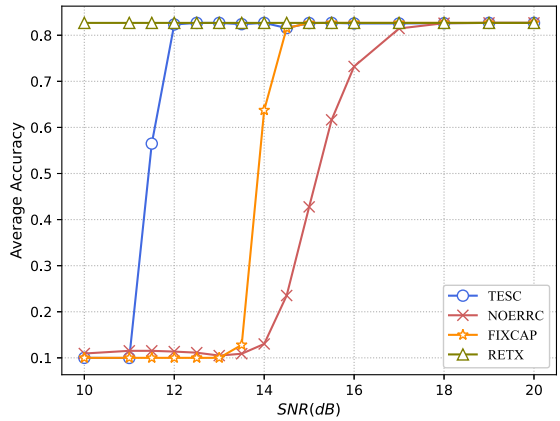
III. Evaluation results

To show the effectiveness of TESC on the inference accuracy and traffic overhead, we compare it with the following three schemes: 1) NOERRC where the intermediate data is transmitted without any error correction method; 2) FIXCAP where the intermediate data is transmitted using the RS code with the fixed error correction capability regardless of channel condition; 3) RETX where the intermediate data is retransmitted until the data arrives successfully at the edge cloud.

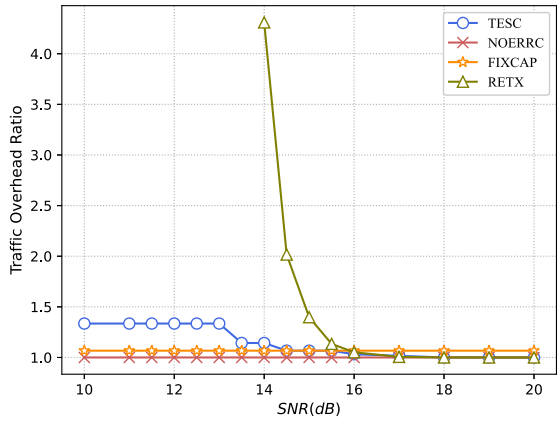
The default simulation settings are as follows. For the inference, we use VGG16 [6] with CIFAR10 dataset [7]. VGG16 model is divided in half and the first and second half of the models are used as the head and tail models, respectively. In addition, it is assumed that IoT device uses QPSK to transmit the intermediate data and bitwise errors in the intermediate data occur according to SNR.

Figures 2(a) and (b) show the effect of SNR on the average accuracy and the traffic overhead ratio, respectively. The traffic overhead ratio is normalized based on the traffic overhead of NOERRC. That is, the traffic overhead ratio represents the ratio between the traffic overhead of each scheme and that of NOERRC. From Figures 2(a) and (b), it can be observed that TESC can obtain high accuracy while maintaining the traffic overhead at low level. This is because TESC decides the error correction capability according to the current channel condition (i.e., SNR). For example, when the channel condition is good enough (i.e., SNR $\geq$ 17 dB) that the channel coding is not needed, TESC does not use the channel coding. On the other hand, when SNR is less than 17 dB, TESC selects

[1]The RS code is characterized by the total data length n, the original data length k, and the error correction capability t. The relationship between these parameters is given by $t = \frac{n-k}{2}$. Note that the RS code is able to correct up to t symbol errors.



(a) Average accuracy.



(b) Traffic overhead ratio.

Fig. 2. Effect of SNR.

appropriate channel correction capability to correct the error with small traffic overhead.

Meanwhile, from Figure 2(a), it can be shown that the average accuracy of RETX is always the highest regardless of SNR. However, this does not imply that RETX is better than TESC. Since RETX retransmits the intermediate data until it arrives at the edge cloud without any error, RETX experiences significant high traffic overhead when SNR is less than 15 dB (see Figure 2(b)).

IV. Conclusion

In this paper, we introduce a traffic-efficient split computing mechanism (TESC) for IoT applications. By adaptively selecting the error correction capability according to the channel condition, TESC optimizes the tradeoff between the inference accuracy and traffic overhead. Evaluation results demonstrated that TESC can achieve high accuracy while significantly reducing traffic overhead compared to conventional error correction techniques under various channel conditions. This implies that TESC is a promising approach for future IoT applications, especially in unstable and fluctuating channel condition scenarios. In our future work, we will devise a method to decide the channel condition threshold. In addition, we will extend the proposed mechanism to determine the optimal splitting point and error correction capability simultaneously.

Acknowledge

This research was supported by Institute for Information & communications Technology Planning & Evaluation (IITP) Grant (No. 2022-0-01015).

References

[1] K. Qu *et al.*, "Stochastic Cumulative DNN Inference with RL-Aided Adaptive IoT Device-Edge Collaboration," *IEEE IoT-J*, to appear.
[2] S. Kim and H. Ko, "Distributed Split Computing System in Cooperative Internet of Things (IoT)," *IEEE Access*, vol. 11, pp. 77669–77678, Jul. 2023.
[3] A. Ayad *et al.*, "Improving the Communication and Computation Efficiency of Split Learning for IoT Applications," in *IEEE GLOBECOM 2021*, Dec. 2021.
[4] T. Venugopal and S. Radhika, "A Survey on Channel Coding in Wireless Networks," in *Proc. ICCSP 2020*, Jul. 2020.
[5] S. Kumar and R. Gupta, "Bit error rate analysis of Reed-Solomon code for efficient communication system," *IJCA*, vol. 30, no. 12, pp. 11–15, Sep. 2011.
[6] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proc. ICLR 2015*, May 2015.
[7] A. Krizhevsky, Learning multiple layers of features from tiny images, 2009.