Towards Edge-assisted Real-time 3D Segmentation of Large Scale LIDAR Point Clouds

Towards Edge-assisted Real-time 3D Segmentation of Large Scale LIDAR Point Clouds

***Abstract*—Light detection and ranging (LIDAR) has become a cost-effective and accessible sensor for a broad range of embedded devices including mobile phones and drones. Vision applications of these embedded devices require fast and accurate inferences to drive them, while at the same time power consumption should be kept low. Achieving both these requirements is hard due to the size of high quality LIDAR point cloud data streams – significantly larger than vision inputs such as images. The complexity of point cloud segmentation adds further difficulty for achieving high quality, realtime LIDAR data driven vision applications on battery powered embedded devices. We consider edge offloading as a potential approach to reconcile these conflicting requirements. Specifically, we present an experimental characterization study exploring the benefit of edge-assisted LIDAR point cloud segmentation, considering diverse set of embedded devices and state-of-the-art semantic segmentation models. Our results indicate that edge offloading is always beneficial from a device energy efficiency perspective and can also significantly reduce inference latency, especially with compressive edge offloading. These latency improvements however fall short of meeting real-time requirements. We outline a number of potential follow-on research directions to enable edge assisted accurate and real-time LIDAR point cloud segmentation.**

I. Introduction

Light detection and ranging (LIDAR) due to its speed of data generation and the accuracy has a significant advantage over photogrammetry based approaches for computer vision applications (e.g., semantic and instance segmentation for real-time autonomous navigation, interactive augmented reality). With advances in solid-state technology, LIDAR is being increasingly integrated into resource constrained, and embedded devices. For example, Apple's recent iPad Pro and iPhone 12 Pro use their on-board LIDAR as the default method for image reconstruction. Lightweight drones are another example that largely rely on LIDAR for land surveying.

Although LIDAR has become a cost-effective and accessible sensor, the heavy computational demands associated with processing high quality LIDAR point cloud data is a hindrance to real-time sophisticated perception tasks such as 3D Segmentation on resource constrained and battery operated devices. Autonomous drones, for example, may require real-time decisions for navigation [1]. This hindrance is becoming increasingly prominent with the emergence and adoption of high-resolution LIDARs. The cutting-edge LIDAR (e.g., RoboSense[1]) is able to generate 200,000 points per frame with the point number expected to keep growing. State-of-the-art (SOTA) deep learning based 3D-Segmentation methods (e.g., [2]) can provide high segmentation accuracy, but are known to be computationally expensive and power hungry, rendering them ill-suited for processing large scale point clouds in real-time on resource-constrained platforms. Reducing computational complexity through simplification of raw point cloud data comes at the expense of accuracy. The key unresolved challenge in this context is balancing between high quality, real-time LIDAR data processing and low power consumption on resource constrained devices.

In this paper, for the first time, we examine the potential of edge computing to help address the aforementioned challenge. We explore the benefit of offloading computationally intensive inference operations on LIDAR point cloud data to a nearby edge compute infrastructure, which promises to preserve inference accuracy while relieving the device from the inference related computation and power consumption burden. To this end, we present an experimental characterization study considering semantic segmentation as a representative application of LIDAR point cloud data. Specifically, we experiment with several SOTA deep learning based models and using the widely used LIDAR point cloud Semantic-KITTI [3] dataset. We consider a wide spectrum of embedded devices and an emulated edge infrastructure to study the effect of different inference strategies – on-device processing, edge offloading with and without compression – on inference latency and energy per inference under diverse network conditions.

Our results show that both forms of edge offloading yield significant reductions in energy per inference performance. Edge offloading also provides gains in inference latency relative to on-device processing for severely resource constrained devices whereas improvements are more consistent with compressive edge offloading. While we consider only the common Octree point cloud data compression, these results highlight the importance of point cloud data communication cost to inference latency. However, ensuring real-time inference is still a challenge which remains to be addressed. Real-time can be considered to be anything over 30Hz [4], providing a challenging inference latency target of at most 33ms. To that end, we present several directions for building on the work presented in this paper towards edge assisted real-time LIDAR point cloud segmentation.

The focus on 3D point cloud data sets our work apart from prior research on edge offloading (e.g., [5]–[7]) that mainly dealt with image data. In our case, not only are inference tasks computationally intensive but the associated point cloud data is significantly larger than image data, the latter making the communication cost a bigger concern for edge offloading.

II. Background

A. 3D Segmentation

Segmentation is a process in which each point/pixel in a point cloud/image is assigned to an object category to enable

[1]https://www.robosense.ai/en

high-level understanding of a particular scene, and is key to many LIDAR based vision tasks. Segmentation tasks can be divided into three major categories: Semantic, Instance and Part (or Hierarchical) Segmentation. Modern approaches to segmentation of 3D point clouds fall into two main categories: 1) Projection based methods (e.g., [8]); and 2) Point based methods (e.g., [2]). Projection methods give a point cloud more structure by projecting them into several 2D images. Each of these images are analyzed separately, and a final semantic interpretation is determined by fusing the images back together [9]. This may result in information loss as some of the geometric structure of the point clouds may not be retained. Point based methods, on the other hand, use point clouds as direct inputs to their permutation invariant models. This allows all geometric information to be retained and reduces the amount of pre-processing required on point clouds. However, as the data is unstructured, finding neighboring points (e.g., by k-nearest neighbors (KNN)) is very expensive.

B. Edge Computing

Edge-assisted DNN. Edge computing aims to take the processing burden off the device, while keeping the overall latency to a minimum by leveraging nearby edge infrastructure. A typical approach for deep neural network (DNN) related edge offloading involves partitioning the DNN model so that computationally expensive layers can be executed at the edge [5], [6]. Additionally, intermediate results of DNNs can be compressed for offloading to reduce communication costs [7]. Prior works on DNN related edge offloading have focused on the mature image and video processing field, and not on the larger data sized, and rapidly evolving, 3D point clouds field as in our case. Moreover, the unstructured nature of point cloud files prevents them from being directly used within the standard convolutional neural networks (CNNs), which are not input permutation invariant.

Edge-assisted Point Cloud Processing. DNN related offloading often requires retraining and modification of neural network architectures for early exit. Conversely, efficient processing of point cloud data can be achieved by model partitioning and compressive sensor data offloading which is adaptive to various workloads and network conditions. Perception latency and coverage can then be improved by offloading LIDAR data to edge servers for collaborative inference [10]. Although computational graph based model partitioning, together with dynamic scheduling, is general enough to be applied on any type of neural network, CNN workloads are often the intended use case [11], [12].

C. Efficient Segmentation Models

Tiny DNN models can help to lower model runtime memory requirements such that they are well suited for edge devices and embedded GPUs. Whilst realtime service level objectives are more likely attainable, this comes with the inevitable cost of compromising accuracy in the region of 5% up to even 20% in segmentation tasks [13]. Tiny models can be both efficient and accurate in single object detection tasks, however, they often fail to generalize to semantic segmentation [14]. Several attempts have been made to improve inference efficiency without degrading accuracy, however, the realtime inference is far from being satisfied [15]. The challenge of efficient on-device segmentation not only depends on DNN architecture, but also input data size [16], and when it comes to urban scale segmentation, the frame size becomes a major bottleneck.

	NVIDIA		
Device	Jetson Nano	Jetson TX2	AGX Xavier
CPU Capability	4× 1.48GHz	2× 2.26GHz 4× 1.48GHz	8× 2.26GHz
GPU (TFLOPs)	0.472	1.33	5.5
Memory	4GB	8GB	16GB
Power	1.8W	2.2W	5.3W

TABLE I
HARDWARE SPECIFICATIONS OF REPRESENTATIVE EMBEDDED DEVICES. MEMORY IS SHARED BETWEEN CPU AND GPU.

III. METHODOLOGY

Here we describe our experimental characterization methodology, covering the point cloud segmentation models considered, dataset used, inference strategies under study, evaluation metrics and the experimental testbed setup.

A. Testbed Setup

We choose a wide spectrum of embedded GPU equipped devices from NVIDIA (Table I) to emulate different types of devices (a drone can be equipped with, for example), where the computational power is limited and power consumption has to be low to prolong battery life. The devices of choice allow for power efficient processing in a small form-factor without massively sacrificing performance. They also range from low-end to high-end equipment. Baseline power readings were determined based on each of the boards respective internal power monitors, thirty minutes after startup, with no processing load.

Our testbed (Figure 1) emulates end-to-end communication between embedded devices and the edge. Each of the embedded devices were connected to a laptop (which acts as the edge) running Ubuntu 18.04 (Intel i7-7500U CPU @ 2.70GHz, 8GB Samsung 2133MHz RAM) via a direct wired Gigabit Ethernet connection. We use Linux Traffic Control (TC) with NetEm to emulate different network conditions. With this setup, both effective bandwidth and delay could be configured to desired values for both upstream and downstream traffic. As the edge in this setup is only meant to emulate an edge cloud infrastructure, we do not actually run the model inference on it, but mimic it with a predetermined delay. This delay is the same for all models (30ms), and is based on running PolarNet [17] on a Google Cloud Compute Server with a NVIDIA T4 GPU.

Depending on the inference and edge offloading strategy considered within each experiment, a point cloud data compression module (using Octree [18]), can be switched on or off. Monitors are placed on embedded devices to measure its power consumption and model inference latency.

B. SOTA Semantic Segmentation Models

We consider a representative set of SOTA point cloud segmentation models (Table II) for exploring the benefit of

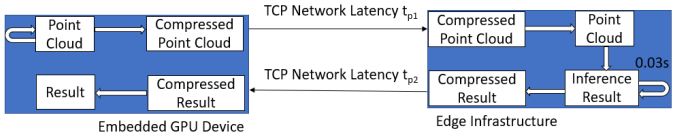

Fig. 1. Schematic of our testbed experimental setup.

Model	Lib	Approach	MAC	#Param	Acc
RandLA-Net [2]	PT	Point	362B	1.24M	52.6%
RandLA-Net [2]	TF	Point	66.5B	1.24M	53.9%
3D-MiniNet [8]	PT	Projection	-	4M	55.8%
3D-MiniNet [8]	TF	Projection	-	4M	55.8%
PolarNet [17]	PT	Projection	136B	14M	54.3%
RangeNet++ [19]	PT	Projection	378B	50M	52.2%
RangeNet++ [19]	TRT	Projection	-	50M	51.9%

TABLE II
MULTIPLY AND ACCUMULATE OPERATIONS (MAC), NUMBER OF PARAMETERS (#PARAM) AND ACCURACY (ACC) [20] OF EACH MODEL. '-' INDICATES THAT THE MAC WAS NOT REPORTED BY THE AUTHORS AND COULD NOT BE DETERMINED.

edge assisted processing of LIDAR point cloud data. These models use different semantic segmentation techniques (point and projection based approaches) and are implemented using different DNN frameworks/libraries – TensorFlow (TF), PyTorch (PT), TensorRT (TRT). Where possible, the same model is evaluated using different frameworks with the intention of making our conclusions as framework independent as possible.

It is however important to note that due to the differences in how models are realized in TF (particularly TF1) and PT, direct comparisons should not be made between the properties of these models. Additionally, multiply and accumulate (MAC) operations represent the number of multiply/accumulate operations within a model, a common task within DNNs which generally employ matrix vector maths.

C. Dataset

We use Semantic-KITTI [3], a widely used 3D segmentation benchmarking dataset, for evaluation and summarize it in Table III. This dataset is aimed at vision training of autonomous vehicles, providing an understanding of objects and surfaces surrounding a moving vehicle, and contains sequences of real-world LIDAR generated point clouds. The realism, data size and well-established nature of Semantic-KITTI makes 3D-Segmentation and energy efficient model inference a challenging task. All tests were run on the official evaluation sequence: *Sequence 8*.

Semantic-KITTI point clouds contain both positional coordinate information and a float intensity value. As such, the point clouds do not perfectly suit the standard Octree compression algorithm which is designed for point clouds with either only positional information or positional information alongside an integer RGB color value. A workaround was devised such that the intensity value was encoded into the red component of the RGB variable of Octree compression. There is a slight loss of information due to float inaccuracies, as the float must be stored within an 8-bit integer.

Rate	#Classes	Labeling	#Points	#Scans	#Sequences
10 Hz	28	Pointwise	4.5B	43,552	21

TABLE III
SEMANTIC-KITTI DATASET SUMMARY.

D. Inference Strategies

We use on-device model inference as the baseline for assessing the benefits with edge offloading. In particular, we consider two edge offloading strategies: 1) offloading without sensor data compression; 2) offloading with sensor data compression. Comparison between these strategies can reveal the cost of point cloud data communication and the need for some form of data compression.

1) On-device Processing: The first strategy considers keeping inference of each of the chosen models fully on-device, allowing the latency and energy costs for each model to be determined. Care is taken to ensure that the GPU is fully utilized during the execution of each model for the best performance results. With this strategy, compression and the edge-offloading module (in Figure 1) are disabled.

2) Edge Offloading: The second strategy emulates a basic edge-assisted 3D semantic segmentation scenario. TC is used to emulate a range of network conditions between "ideal" and "poor" through control of bandwidth (between 25Mbps and 250Mbps) and latency (10ms or 30ms) as per [21]. This reflects the setting whereby inference takes place on the edge as opposed to the embedded device, with raw point clouds streamed to the edge for processing, and results returned.

3) Edge Offloading with Compression: This third strategy is a variant of the above edge-offloading strategy augmented with point cloud data compression. Specifically, we apply the Octree compression algorithm [22] prior to offloading, and a simple zip compression to the returning inference results (binary labels). Octree algorithm has a number of tunable parameters (such as Octree Resolution and Point Resolution) in order to provide flexibility in compressive ratio, compression speed, and ultimately accuracy. Due to space restrictions, only the best performing Octree configuration in terms of inference latency is shown.

E. Evaluation Metrics

We consider two key metrics to evaluate the benefit of edge offloading to point cloud segmentation: 1) *inference latency (IL) in seconds*; 2) *energy per inference (EI) in Joules*. These metrics shown in our results are an average of multiple experiments corresponding to each data point.

1) Inference Latency (IL): IL refers to the time to process a point cloud frame, averaged across all frames in the sequence. For on-device processing, Hadidi et al. [23] had a similar methodology. For the two edge offloading cases, network transmission and any pre/post processing also contributes to the total IL.

2) Energy per Inference (EI): For on-device processing, EI is defined as the inference latency multiplied by the average power consumption, as in [23]. For the edge offloading cases, EI is defined by (1) as the total power consumption across all inferences divided by frames per second (FPS). Here FPS denotes the average number of point cloud frames that can be processed every second. A differing calculation is required

as there can be multiple frames within the edge processing pipeline at any time.

$$\text{EI} = \frac{\#\text{inferences} \times \text{total power consumed}}{\text{FPS}} \quad (1)$$

IV. Results

A. On-device Processing

Through evaluation of each SOTA model on-device, it can be observed that inference performance is approximately proportional to MAC and device capability. Additionally, real-time processing cannot be achieved (§IV-A1), and greater computational capacity does not always translate to ultimate efficiency (§IV-A2).

1) Inference Latency (IL): Figure 2 details the IL against model for each of the devices. All models, regardless of device, have an IL too large to be considered real-time. This motivates an opportunity to offload the models in some way to allow the strict real-time requirements to be met on resource constrained devices. We have additionally evaluated a smaller version of the 3D-MiniNet model (3D-MiniNet-Tiny) on the TX2. This model has only 0.4M parameters, and achieves a non-SOTA accuracy of 49.0%. Even with this tiny model, the best achievable inference was 45ms – approximately 12ms over our real-time inference goal.

2) Energy per Inference (EI): Figure 3 shows the EI for each model (on each device), and its results provide similar motivation to offload, as well as providing a straw-man baseline when offloading. Interestingly, it can be observed that the EI of the PyTorch RangeNet++ is significantly higher than any other model. Firstly, compared to other models, its MAC value is very large (for a projection-based approach). This significantly impacts the number of operations which must be completed per inference. Secondly, the model's inference latency is high, while the GPU and CPU power usage is not (Figure 4); in combination this results in a high EI.

Furthermore, we found that the GPU and CPU usage of RandLA-Net is higher than any other model; this is framework independent but is not entirely unexpected. RandLA-Net is the only model following a point-based approach, which contains a very expensive KNN step to find point neighbors. This makes it surprising that RandLA-Net has inference speed comparable to projection-based models, especially when considering that the MAC is within the same order of magnitude.

The results of RangeNet++ are noteworthy. The expectation is that Tensor-RT will not only make the PyTorch version of a model more (energy) efficient, but it will also provide a speed boost (with a slight cost to accuracy). For the Range-Net++ model, this was not the case – only energy improvements were observed. However, without further experimentation on other models and model compression techniques, it is difficult to conclude whether this is a characteristic of the model itself or the application domain.

B. Edge Offloading

We next report the results of a basic form of edge offloading where raw point cloud data is streamed to the edge under varying network conditions.

BW (Mbps)	Network Latency (ms)	Xavier EI (J)	TX2 EI (J)	Nano EI (J)
250	10	0.27	0.15	0.14
100	10	0.69	0.30	0.27
50	10	2.03	0.91	0.76
25	10	4.29	1.67	1.41
250	30	0.31	0.17	0.16
100	30	1.02	0.43	0.24
50	30	2.73	1.33	0.99
25	30	7.33	3.38	2.51

TABLE IV
ENERGY PER INFERENCE WITH EDGE OFFLOADING FOR EACH DEVICE AND DIFFERENT NETWORK CONDITIONS.

1) Inference Latency (IL): We observe that even under perfect network conditions, this way of edge offloading is insufficient to achieve real-time processing. Specifically, the best achievable inference latency in this scenario is only 0.38s on the AGX Xavier, using the PolarNet model (Figure 5). Although worse than on-device processing, it should also be noted that in the case of Jetson TX2, similar latencies are observed, and for the Jetson Nano (the most resource constrained device) improvements can be clearly witnessed.

2) Energy per Inference (EI): Promisingly, we see significant improvements in EI across the board, with the best conditions on the AGX Xavier showing 91.5% improvements over the best on-device processing, and with the other devices achieving similar improvements of around 90% (Table IV). Unsurprisingly, we also observe that better network conditions usually yield a lower energy consumption as it reduces the time spent communicating the point cloud data to the edge.

In summary, when the basic form edge offloading is applied, although the energy per inference is greatly improved, model inference is still far from meeting the real-time requirements, even in ideal network conditions.

C. Edge Offloading with Compression

Finally we report the performance of edge offloading coupled with data compression. Depending on the compression parameters used, the size of the point cloud files could be compressed from around 2MB to between 305 and 585 KB, with the resultant loss and compression time also dependent on these parameters. Table V outlines the offloading results utilizing the best performing compression parameters (in terms of inference latency), while keeping at least 85% mean reconstruction (decompression) accuracy. There is no single optimal compression setting which suits all network conditions; rather it is dependent on compression time and resultant file size.

The main takeaways here are that significant improvements can be observed in terms of both IL and EI. However, even with idealistic conditions, this approach is unable to obtain close to real-time processing on any device or network conditions. As expected performance begins to degrade as network conditions do; however it does show strong promise as a technique which could contribute towards real-time point cloud processing (Table V).

The Jetson Nano is the biggest winner from compressed edge offloading. Even in the worst network conditions, it is

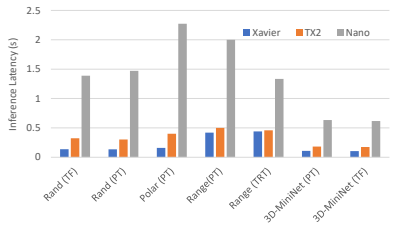

Fig. 2. Inference latency with on-device processing for each model.

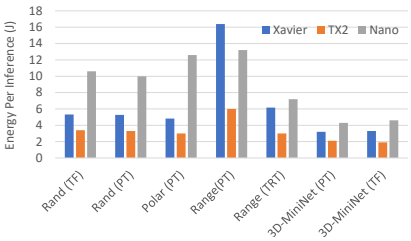

Fig. 3. Energy per inference with on-device processing for each model.

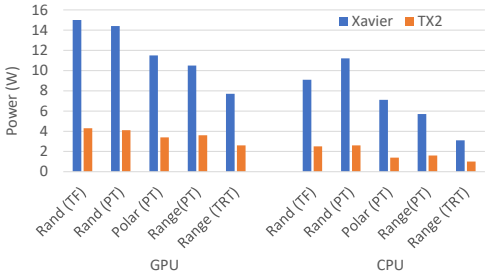

Fig. 4. GPU and CPU power consumption with on-device processing for each model; GPU vs CPU power split could not be measured on the Jetson Nano.

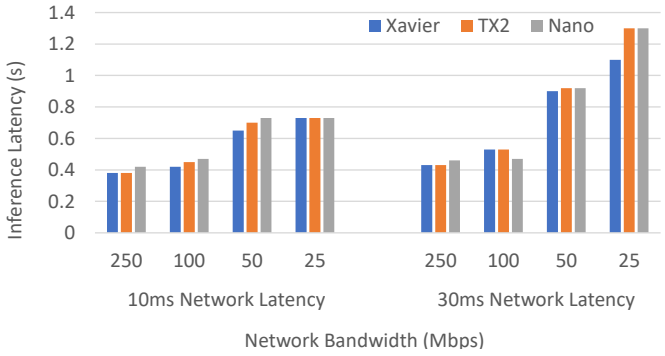

Fig. 5. Inference latency with edge offloading for PolarNet model and different devices.

still able to achieve an inference latency of 0.410s, a 32.8% and 81.9% improvement, respectively, over the best and worse inference latencies with on-device processing.

The EI is vastly improved compared to on-device inference. For the most efficient model, on-device can achieve 2.1J, 1.6J and 4J on AGX Xavier, Jetson TX2 and Jetson Nano, respectively. With compressive offloading, the worst achieved are 0.39J, 0.26J and 0.21J, respectively. EI, when compared to the uncompressed offloading, is similar and only marginal gains are made, implying that the addition of compression does not add too much overhead to device energy consumption.

Condition	Device	PR	OR	IL (s)	EI (J)
Ideal	Xavier	0.01	1	0.092	0.07
	TX2		1	0.104	0.05
	Nano		1	0.146	0.047
	Xavier	0.05	1	0.086	0.06
	TX2		1	0.097	0.043
	Nano		1	0.136	0.042
Poor	Xavier	0.01	0.5	0.345	0.39
	TX2		0.5	0.386	0.26
	Nano		0.5	0.41	0.21
	Xavier	0.05	0.25	0.307	0.34
	TX2		0.25	0.337	0.225
	Nano		0.5	0.385	0.155

TABLE V
COMPRESSIVE EDGE OFFLOADING RESULTS FROM IDEAL (250MBPS BANDWIDTH AND 10MS NETWORK LATENCY) AND POOR (25MBPS BANDWIDTH AND 30MS NETWORK LATENCY) NETWORK CONDITIONS. OR REPRESENTS OCTREE RESOLUTION. PR REPRESENTS POINT RESOLUTION.

V. DISCUSSION AND FUTURE WORK

Our results above have shown that even by offloading with compression and under idealistic conditions, the real-time inference latency goal cannot be met, although massive improvements in terms of energy efficiency are achieved with offloading regardless of condition or device. Future work in this field should *focus on improving the offloading methodology to make progress towards real-time processing, with a particular focus on suiting a wide range of devices and network conditions.* In foresight, there is also an urgent need to find solutions to cope with further expected increases in LIDAR density over the coming years. Cutting-edge LIDAR today is able to produce upto 5x more points than five years ago, reaching densities of 200,000 points per frame; this upward trend is likely to continue. This, in turn, will require larger DNNs to segment this data. The increasing popularity of LIDAR will also result in additional pressure to have mature techniques in place to effectively process LIDAR data in *real-time and at scale.* Here we chart some future directions and other related application domains of large-scale point cloud segmentation.

A. Dynamic Model Partitioning

A promising direction to explore is model partitioning, where the computation (layers) of a DNN are split across device and edge sequentially as a head and a tail. This takes edge offloading a step further, by leveraging the computational power of both the edge and the end device, by only offloading when deemed necessary. Neurosurgeon [24] dynamically determines a model partitioning point based on network conditions but only considers models in the computer vision (images), speech, and NLP domains. It would be interesting to explore how dynamic model partitioning techniques can be applied to point cloud segmentation tasks in which the DNNs evaluated may have vastly differing operators (e.g., PointConv [25]). The dynamic strategies will also need to take both device capability and model architectures into account – a more powerful device will be able to process more of a model than a less powerful one and as a result can reduce network latency. Additionally, certain models will be intrinsically suited to this approach if they contain bottleneck layers, where the model naturally narrows resulting in low data volume transmission at the partitioning point. To this end, layer-wise profiling will be needed to determine optimal model partitioning points. Moreover, tradeoffs between offloading sizes and end accuracy need to be take into account.

B. Bespoke DNN Architectures from Construction Phase

Effective model partition strategies seldom come alone but often require suitable DNN architectures in conjunction. An interesting direction is to identify which layers in DNNs can serve as an efficient partitioning point. **Bottleneck injection** could be adopted to reduce the intermediate result size in strategic place(s) [26]. This would allow for an optimal splitting point where offloading can occur that minimizes network latency and, if getting to this bottleneck is not computationally demanding, the overall latency may be reduced. However, the requirement of retraining the models once the bottleneck is added reduces the flexibility of such approach, requiring that the artificial bottleneck are determined a priori. Furthermore, model compression through bottleneck injection usually comes with accuracy loss. As far as can be seen, no point cloud based tasks have attempted this technique, and it may be beneficial to explore the benefits that this could provide. However, due to the retraining requirements, this technique could be impractical on large datasets. The ideas established may be useful in further exploring autoencoders as described below.

Given the size of LIDAR point cloud data, the computational demand for processing the first few layers in a 3D point cloud compatible DNN model may still be an issue for on-device processing. An **imbalanced autoencoder** is a more general framework which enables the encoding of offloaded data into small sizes, with limited computational impact on the end device [7]. The goal of the autoencoder is not to facilitate the complete reconstruction of the data, but to work alongside any existing models to achieve the best possible inference. No alterations need to be made to the original model making it easily appliable to a range of different models. It is worth establishing whether this approach to offloading is preferable to compressive offloading, and in what conditions. Additionally, it may be worth corroborating the claims made in [7] regarding the appropriate offloading point in splitting the processing across the end device and the edge, when considering LIDAR point cloud data. Furthermore, exploring whether autoencoders can be placed anywhere within a model may be worthwhile. This would mitigate the need to retrain the entire model for each partition point, as with artificial bottlenecks. If successful, this may have the ability to combine the flexibility of autoencoders/model partitoning with significant reduction in data sizes at partition points.

C. More Complicated Segmentation Tasks

In addition to semantic segmentation, **instance segmentation** and **panoptic segmentation** on point clouds are alternative applications worth investigating in future work. These tasks are known for their greater complexity over semantic segmentation and it would be interesting to determine if these point cloud data applications differ significantly in their latency and energy consumption characteristics and the gains various edge offloading techniques can bring.

REFERENCES

[1] V. Venugopal and S. Kannan, "Accelerating real-time LIDAR data processing using GPUs," in *MWSCAS*. IEEE, 2013, pp. 1168–1171.

[2] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds," in *IEEE/CVF CVPR*, 2020.

[3] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences," in *ICCV*, 2019, pp. 9296–9306.

[4] L. Liu, H. Li, and M. Gruteser, "Edge Assisted Real-time Object Detection for Mobile Augmented Reality," in *ACM MobiCom*, 2019.

[5] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[6] L. Lin, X. Liao, H. Jin, and P. Li, "Computation Offloading Toward Edge Computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.

[7] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. F. Abdelzaher, "Deep Compressive Offloading: Speeding Up Neural Network Inference by Trading Edge Computation for Network Latency," in *SenSys*. ACM, 2020, pp. 476–488.

[8] I. Alonso, L. Riazuelo, L. Montesano, and A. C. Murillo, "3D-MiniNet: Learning a 2D Representation From Point Clouds for Fast and Efficient 3D LIDAR Semantic Segmentation," *IEEE Robotics Autom. Lett.*, vol. 5, no. 4, pp. 5432–5439, 2020.

[9] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep Learning for 3D Point Clouds: A Survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 12, pp. 4338–4364, 2021.

[10] X. Zhang, A. Zhang, J. Sun, X. Zhu, Y. E. Guo, F. Qian, and Z. M. Mao, "EMP: Edge-Assisted Multi-Vehicle Perception," in *ACM MobiCom*, 2021.

[11] L. Zhou, H. Wen, R. Teodorescu, and D. H. C. Du, "Distributing Deep Neural Networks with Containerized Partitions at the Edge," in *HotEdge*. USENIX Association, 2019.

[12] A. Banitalebi-Dehkordi, N. Vedula, J. Pei, F. Xia, L. Wang, and Y. Zhang, "Auto-Split: A General Framework of Collaborative Edge-Cloud AI," in *KDD*. ACM, 2021, pp. 2543–2553.

[13] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-Voxel CNN for Efficient 3D Deep Learning," in *NeurIPS*, 2019, pp. 963–973.

[14] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection," in *CVPR*, 2020.

[15] C. Xu, B. Wu, Z. Wang, W. Zhan, P. Vajda, K. Keutzer, and M. Tomizuka, "SqueezeSegV3: Spatially-Adaptive Convolution for Efficient Point-Cloud Segmentation," in *ECCV*, 2020.

[16] Q. Hu, B. Yang, S. Khalid, W. Xiao, N. Trigoni, and A. Markham, "Towards Semantic Segmentation of Urban-Scale 3D Point Clouds: A Dataset, Benchmarks and Challenges," in *IEEE CVPR*, 2021.

[17] Y. Zhang, Z. Zhou, P. David, X. Yue, Z. Xi, B. Gong, and H. Foroosh, "PolarNet: An Improved Grid Representation for Online LiDAR Point Clouds Semantic Segmentation," in *CVPR*, 2020.

[18] D. Meagher, "Geometric Modeling Using Octree Encoding," *Comput. Graph. Image Process.*, vol. 19, no. 1, p. 85, 1982.

[19] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, "RangeNet ++: Fast and Accurate LiDAR Semantic Segmentation," in *IROS*. IEEE, 2019, pp. 4213–4220.

[20] "Papers with code - semantickitti benchmark (3D semantic segmentation)." [Online]. Available: https://paperswithcode.com/sota/3d-semantic-segmentation-on-semantickitti

[21] A. A. Majeed, P. Kilpatrick, I. T. A. Spence, and B. Varghese, "Modelling Fog Offloading Performance," in *ICFEC*. IEEE, 2020, pp. 29–38.

[22] R. Schnabel and R. Klein, "Octree-based Point-Cloud Compression," in *PBG@SIGGRAPH*. Eurographics Association, 2006, pp. 111–120.

[23] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, "Characterizing the Deployment of Deep Neural Networks on Commercial Edge Devices," in *IISWC*, 2019.

[24] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. N. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," in *ASPLOS*. ACM, 2017, pp. 615–629.

[25] W. Wu, Z. Qi, and F. Li, "PointConv: Deep Convolutional Networks on 3D Point Clouds," in *CVPR*. Computer Vision Foundation / IEEE, 2019, pp. 9621–9630.

[26] Y. Matsubara, M. Levorato, and F. Restuccia, "Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges," *CoRR*, vol. abs/2103.04505, 2021.