PointSplit: Towards On-device 3D Object Detection with Heterogeneous Low-power Accelerators

Keondo Park
Graduate School of Data Science, Seoul National University
Seoul, Republic of Korea
gundo0102@snu.ac.kr

You Rim Choi
Graduate School of Data Science, Seoul National University
Seoul, Republic of Korea
yrchoi@snu.ac.kr

Inhoe Lee
Graduate School of Data Science, Seoul National University
Seoul, Republic of Korea
inhoelee@snu.ac.kr

Hyung-Sin Kim
Graduate School of Data Science, Seoul National University
Seoul, Republic of Korea
hyungkim@snu.ac.kr

ABSTRACT

Running deep learning models on resource-constrained edge devices has drawn significant attention due to its fast response, privacy preservation, and robust operation regardless of Internet connectivity. While these devices already cope with various intelligent tasks, the latest edge devices that are equipped with multiple types of low-power accelerators (i.e., both mobile GPU and NPU) can bring another opportunity; a task that used to be too heavy for an edge device in the single-accelerator world might become viable in the upcoming heterogeneous-accelerator world. To realize the potential in the context of 3D object detection, we identify several technical challenges and propose *PointSplit*, a novel 3D object detection framework for multi-accelerator edge devices that addresses the problems. Specifically, our *PointSplit* design includes (1) 2D semantics-aware biased point sampling, (2) parallelized 3D feature extraction, and (3) role-based group-wise quantization. We implement *PointSplit* on TensorFlow Lite and evaluate it on a customized hardware platform comprising both mobile GPU and EdgeTPU. Experimental results on representative RGB-D datasets, SUN RGB-D and Scannet V2, demonstrate that *PointSplit* on a multi-accelerator device is 24.7× faster with similar accuracy compared to the full-precision, 2D-3D fusion-based 3D detector on a GPU-only device.

CCS CONCEPTS

• **Computing methodologies → Object detection**; • **Computer systems organization → Embedded software**.

KEYWORDS

3D object detection, On-device machine learning, Edge computing, Quantization

ACM Reference Format:
Keondo Park, You Rim Choi, Inhoe Lee, and Hyung-Sin Kim. 2023. PointSplit: Towards On-device 3D Object Detection with Heterogeneous Low-power Accelerators. In *The 22nd International Conference on Information Processing in Sensor Networks (IPSN '23), May 09–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3583120.3587045

1 INTRODUCTION

On-device machine learning (ML), which runs deep neural networks (DNNs) directly on an edge device (e.g., mobile phone), has drawn increased attention due to its potential to enable real-time and private ML applications. Development of low-power AI accelerators (e.g., mobile GPU and NPU), model compression schemes (e.g., quantization, pruning, and knowledge distillation), and system execution techqnies has enabled to run various intelligent tasks on a device, such as 2D object detection and language processing models [1, 2, 6, 14, 25, 57, 71].

Furthermore, although an edge device used to have a single type of AI processor, the recent emergence of heterogeneous processor System-on-Chips (SoCs) [66] has made the state-of-the-art mobile devices equipped with both high-end mobile GPU and NPU. The new class of edge devices with *multi-type accelerators* present an opportunity to investigate interesting issues in the regime of on-device ML, such as intra-device parallelism and algorithm-system co-optimization by understanding different characteristics of the accelerators. With such evolution of low-power hardware, systems, and deep learning models together, more complex tasks that used to be far from resource-constrained devices, such as 3D object detection, might be able to run directly on device in real-time. Specifically, running 3D object detection directly on resource-constrained devices, instead of powerful remote servers, has the potential to significantly expand the scope of AI applications. For example, as shown in Figure 1, a fast understanding of 3D indoor scenes directly on an edge device can be an important building block of the upcoming mixed reality [20]. This work aims to investigate this new opportunity: *on-device 3D object detection using both GPU and NPU.*

Challenges. However, we identify that even with the latest edge devices containing both GPU and NPU, enabling on-device 3D object detection without sacrificing accuracy is challenging in many

IPSN '23, May 09–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0118-4/23/05…$15.00
https://doi.org/10.1145/3583120.3587045

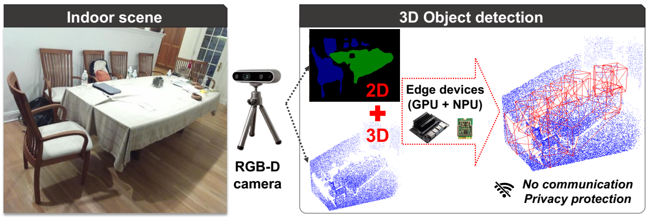


Figure 1: Target scenario: On-device 3D indoor scene understanding via RGB-D camera. On-device detection provides advantages on privacy, latency, and communication burden. 2D-3D fusion can improve detection accuracy while utilizing both GPU and NPU can accelerate on-device inference speed.

ways: (1) 3D object detection is typically designed as a sequential process, making it hard to utilize GPU and NPU in parallel. (2) Since GPU and NPU have different strengths, a 3D object detection model should be analyzed thoroughly to distribute its computation to the two processors synergistically. (3) Fusing 2D vision information with a 3D point cloud (e.g., using an RGB-D camera) can improve detection performance [4, 43, 47] but makes the computational burden even heavier on the edge devices. (4) Quantization is necessary to reduce computation as well as to utilize NPU but given that 3D object detection is a sophisticated task, a naïve approach would significantly degrade the accuracy.

Approach. To tackle the challenges, we propose *PointSplit*, a novel framework that provides system-driven model structure optimization for on-device 3D object detection. For the baseline deep neural network (DNN), we exploit VoteNet [46], a popular 3D object detection network based on the PointNet++ backbone [44] for indoor 3D scene understanding, and borrow the idea of PointPainting [61] to augment features in a 3D point cloud (only geometric features) with 2D image semantics. Building on the baseline, we devise three components for *PointSplit* as below:

- **2D semantics-aware biased 3D point sampling** aims to perform point sampling, a necessary process for processing a point cloud, more efficiently considering multi-type accelerator environments. To this end, we paint each 3D point using 2D image semantics and utilize the information to sample two complementary point sets, one from all points and the other more focused on the painted points (i.e., object-related points). We perform set abstraction (SA) process for the two point sets separately, called SA-normal and SA-bias, respectively. In this way, we generate two different views and perform two individual SA pipelines from a single 3D point cloud scene, which cooperate with each other to improve accuracy.
- **Parallelized 3D feature extraction** comes from the idea that widely used 3D point set abstraction methods [44] comprise two operations, (1) point sampling and ball query that can be run only at GPU and (2) a neural net called PointNet [3] to process the sampled points that can be run at NPU. The two AI processors execute the two SA pipelines (SA-normal and SA-bias) interchangeably: GPU processes sampling and ball query for SA-normal while NPU processes PointNet for SA-bias, and vice versa.
- **Role-based group-wise quantization** is to compress neural networks without sacrificing accuracy and is motivated by the fact that layer-wise quantization significantly degrades accuracy while channel-wise quantization requires many quantization parameters. To find the sweet spot, we investigate each channel's weight and activation distribution and find out that the distribution heavily depends on the *channel's role*. Based on the observation, we group channels according to their role and perform group-wise quantization, which preserves accuracy with only a few quantization parameters.

Contributions. Our contributions can be summarized as follows:

- This work is the first to investigate on-device 3D object detection with heterogeneous low-power AI processors. Specifically, we propose *PointSplit*, a novel framework that jointly designs system and algorithm to effectively reduce inference latency on resource-constrained devices.
- We deeply analyze the characteristics of a 2D-3D fusion-based 3D object detection model and design three unique components to reduce and parallelize computation without sacrificing accuracy: 2D semantics-aware point sampling, parallelized 3D feature extraction, and role-based group-wise quantization.
- We implement VoteNet, a popular 3D object detection model, on TensorFlow from scratch[1] and our *PointSplit* on TensorFlow Lite.[2] Furthermore, we build a test resource-constrained platform by combining NVIDIA Jetson Nano (including mobile GPU) and Google EdgeTPU (an NPU type).[3]
- Experiments show that on two representative datasets for indoor 3D object detection, SUN RGB-D [56] and Scannet V2 [15], *PointSplit* is up to 24.7 times faster than the full-precision, GPU-only baseline while providing similar accuracy.

2 RELATED WORK

Given that this work is related to various fields, this section clarifies what techniques we leverage or are inspired by and what aspects our *PointSplit* newly explores.

2.1 On-device Machine Learning

On-device ML refers to running deep neural network (DNN) inference locally without sending user data to the cloud. There has been a growing interest in on-device ML due to its advantages in latency and privacy. However, it is challenging to run DNN directly on edge devices because their memory, computational resource, and power consumption are strictly constrained. To address this problem, a number of lightweight DNN architectures [51, 58–60] and model compression techniques [19, 21–23, 28] have been proposed. In addition, the development of low-power AI accelerators (e.g., mobile GPU and NPU) has enabled various DNN-based ML applications to be run on devices and showed notable results for some tasks [2, 25, 57]. Furthermore, with the emergence of heterogeneous processor System-on-Chips (SoCs), scheduling or pipelining

[1] VoteNet and other state-of-the-art 3D object detection models are implemented on Pytorch (edge-unfriendly platform so far) but not on TensorFlow, which is a non-trivial entry barrier to research on-device 3D object detection. To the best of our knowledge, this work provides the first open implementation of VoteNet on TensorFlow.

[2] Code is available at https://github.com/KeondoPark/votenet_tf

[3] This platform is a single device but not a system-on-chip (SoC). We expect performance improvement of *PointSplit* when using a SoC including multi-type accelerators.

techniques have been developed to efficiently utilize multiple processors [30, 52, 68].

However, to our knowledge, there has not been any successful attempt for on-device 3D object detection even though heterogeneous low-power AI processors are given. As a step forward, this work presents a *system-algorithm joint design* of 3D object detection to effectively reduce inference latency by fully leveraging the capacity of NPU and GPU on an edge device.

2.2 3D Object Detection

3D object detection is an essential component in robotics, AR/VR and autonomous driving, which require accurate 3D localization of objects. Here 3D localization includes measuring the distance between a user (or robot/vehicle) and an object and the size of the object (i.e., bounding box). For example, in AR/VR applications, inaccurate 3D localization cloud lead to unrealistic display of scenes or user dissatisfaction.

Various methods have been proposed to estimate 3D bounding boxes of objects from point clouds. Many studies rely on voxel-based approaches to process 3D data, such as 3D CNN [41, 65] and Voxel transformer [40]. To reduce the quantization error as well as large memory and computation cost inherent in voxel-based approaches, voxel feature encoding [73], hybrid voxel network [70], or point-voxel fusion methods have been proposed [37, 54, 55]. Another group of methods process point clouds directly for 3D scene understanding. PointNet [3] and PointNet++ [44] use symmetric functions to extract features from irregularly distributed points. VoteNet [46] exploits voting information from the features extracted from points by PointNet++. More recent work uses graph convolution to improve the feature extraction process [5] or an enhanced voting scheme to improve detection accuracy [72].

RGB information can be supplemented to understand 3D scenes. MV3D [12] generates 3D object proposals from a bird's-eye view and uses deep fusion to combine 3D and 2D information. Frustum-PointNet [45] utilizes 2D object detection results to guide 3D object detection. 3D-SIS [24] projects extracted features from 2D convolutions back to a 3D voxel grid to detect objects in a 3D scene. Given that these fusion techniques do not achieve expected performance improvement over 3D-only approaches, PointPainting [61] proposes a sequential fusion as an alternative. It obtains 2D semantic segmentation scores and appends the information to each projected point in 3D space. Despite its advantage on accuracy, the sequential fusion significantly degrades latency.

In terms of 3D object detection model architecture, this work takes a point-based, 2D-3D fusion approach, inspired by VoteNet and PointPainting. With our design choices tailored for a multi-type accelerator environment, *PointSplit* takes advantage of 2D-3D fusion to improve accuracy without sacrificing latency.

2.3 2D Semantic Segmentation

We utilize 2D semantic segmentation to fuse 2D image semantics with 3D point cloud to improve detection accuracy. In this regime, early work first suggested that convolutional neural network provides significant performance improvement over methods relying on hand-crafted features [18, 39]. U-net [49] proposed a U-shaped architecture to improve the capacity of the decoder by connecting expanding paths to contracting paths. Deeplab [7–10] further improved segmentation accuracy by using atrous convolution and a more advanced encoder-decoder structure. We use Deeplabv3+ [8] as our semantic segmentation network.

2.4 Deep Neural Network Quantization

Quantization is an active research area with the rising popularity of edge devices. It aims to carry out the inference with low-bit operations for the efficient use of resources while preserving accuracy. Ternary weight networks [33] or Binary Neural Networks [27] binarize weights and activations of neural networks. Jacob et al. [28] proposed an integer arithmetic only quantization scheme, which significantly accelerates inference and can run on accelerators that support only integer operations, such as EdgeTPU. In this work, we take the full quantization approach to run the model on EdgeTPU.

While most work on quantization targets image classification tasks, a few recent studies [11, 17, 34] suggest quantization techniques optimized for 2D object detection. To our knowledge, however, there has been no work that specifically targets the quantization of a *3D object detector*. In doing so, we focus on quantization granularity, one of the key considerations in quantization. Layer-wise quantization [32] determines the clipping range of the quantization from the statistics of the entire layer. On the other hand, statistics from each channel are used to calculate the clipping range in channel-wise quantization [26, 28]. Q-BERT [53] groups multiple channels to decide the clipping range for quantizing the transformer network. Although our approach also groups multiple channels, we find out that doing it in a different manner is more effective for 3D object detection: taking model semantics into account, rather than grouping evenly.

3 BASELINE AND MOTIVATION

This section presents the baseline network for 2D-3D fusion-based 3D object detection that our *PointSplit* builds upon, and analyzes the problems when naïvely applying the baseline for a multi-type accelerator environment, which motivates *PointSplit*.

3.1 The Baseline: PointNet++ and PointPainting

Our baseline is a 3D object detection model that fuses a 2D image and a 3D point cloud from an RGB-D scene. We choose VoteNet [46] as the baseline 3D object detector, which is widely-used for indoor scene understanding. VoteNet utilizes PointNet++ [44] as the backbone to extract features from a 3D point cloud. For 2D-3D fusion, we take the approach in PointPainting [61], performing 2D semantic segmentation first and utilizing the semantic information for more accurate 3D object detection. We use Deeplabv3+ [8] as the 2D semantic segmentation model and MobileNetV2 [51] as its lightweight feature extractor.

While the baseline sequentially runs Deeplabv3+ and VoteNet, its essence, highly related to our *PointSplit* design, is in the PointNet++ backbone and the fusion method in PointPainting, which are described below.

PointNet++ for 3D Point Set Abstraction. Extracting meaningful features from a set of 3D points is important to detect objects from a 3D scene. While 2D image features can be extracted purely with a neural net due to the dense nature of the RGB image, due to the

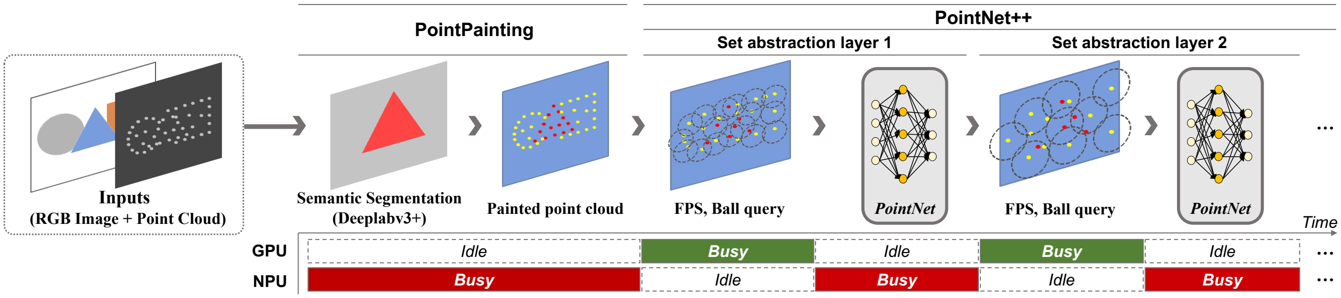


Figure 2: Illustration of naïve workload distribution to run the sequential pipeline of PointPainting on a GPU-NPU combined environment. Among the three figures in the input scene, only the triangle shape is assumed to be a valid object (foreground points). Either of the processors is always idle, waiting for the other to finish its job.

sparse nature of 3D point clouds, it is essential for a 3D point set abstraction method to intermingle point manipulation with neural nets. To this end, PointNet++ [44] has *set abstraction (SA) layer* that includes both point manipulation and neural net.

Specifically, given a point cloud, an SA layer first constructs multiple groups of neighboring points by performing point sampling and ball query sequentially. To sample center point for each group, PointNet++ utilizes the farthest point sampling (FPS) method, which samples a new point that is most distant from the already sampled points. Ball query draws a ball around each center point and groups neighboring points in each ball. After the point manipulation, a local feature vector is extracted for each ball by processing a neural net called PointNet [3]. Since each ball is represented as its center point, the SA layer can be performed again based on the set of center points as a new point cloud input to extract higher-level features. PointNet++ repeats the SA layer four times to extract high-level features hierarchically from a raw-level point cloud.

PointPainting for 2D-3D Fusion. Before processing a point cloud, PointPainting first performs semantic segmentation on a 2D image of the same scene, which divides the image pixels into two groups: foreground (object-related) and background groups. The semantic information is given to each 3D point as an additional feature. Then 3D object detection is performed based on the semantic-aware 3D point cloud, which improves accuracy.

3.2 Motivation: A Naïve Application of the Baseline on Multi-type Accelerators

Running the fusion-based sequential 3D object detection pipeline on a GPU-only environment suffers from long latency, which is also recognized in [61]. To mitigate the problem, the authors proposed a consecutive matching method, which reuses 2D segmentation results of a previous scene for detecting objects on the current scene. However, this approach is vulnerable to the difference between the current and previous scenes and cannot be applied to single-shot detection scenarios. By using GPU and NPU together, we aim to provide *concurrent matching* that performs both 2D semantic segmentation and 3D object detection on the current scene.

When running the baseline on a multi-type accelerator environment (GPU and NPU), it is important to consider what operations can be executed on NPU since it is faster than GPU but supports limited operations. As a neural network accelerator, NPU can process only Deeplabv3+ and PointNet, neither point sampling nor ball query. Therefore, to utilize both NPU and GPU, it is natural to perform point sampling and ball query on GPU, and PointNet and Deeplabv3+ on NPU. Figure 2 depicts this naïve approach.

As shown in Figure 2, however, without changing the baseline's sequential process, the naïve workload distribution inevitably causes idle time on both processors. When processing PointNet++, NPU has to wait while GPU performs point sampling and ball query, and GPU also needs to wait while NPU processes PointNet. The same issue arises when fusing 2D and 3D information; while NPU performs 2D semantic segmentation via Deeplabv3+, GPU waits for the semantic segmentation results in the idle state. Although running these neural nets on NPU instead of GPU has its own speed gain, we aim to step further by reducing the idle time.

4 POINTSPLIT

This section presents our *PointSplit* design, which aims to answer the following questions: (1) Can we create two parallel SA pipelines to utilize both GPU and NPU simultaneously without sacrificing accuracy? (2) Can GPU do something meaningful using the point cloud while NPU processes 2D semantic segmentation? (3) How can we minimize accuracy drop when fully quantizing the baseline 3D object detector?

Figure 3 illustrates our parallel processing of the baseline network. To divide the SA process in PointNet++ into two *lightweight parallel pipelines*, we design point sampling and ball query for each SA pipeline to generate *only half the number of balls* (i.e., the number of center points) while being processed on GPU. While NPU processes PointNet with the reduced number of balls for an SA pipeline (called SA-1), GPU performs point sampling and ball query again to generate the other half of balls for the other SA pipeline (called SA-2) in parallel. This method reduces computation for each SA pipeline and parallelizes point manipulation and neural net operations, which reduces each processor's idle time.

In addition, to utilize 2D semantic information for both lightweight SA pipelines without significant delay, one SA process (SA-1) *jump-starts* on GPU without waiting for the segmentation results from NPU since the segmentation results are needed for PointNet, not point manipulation. After GPU and NPU finish the point manipulation (for SA-1) and 2D segmentation tasks, respectively, NPU

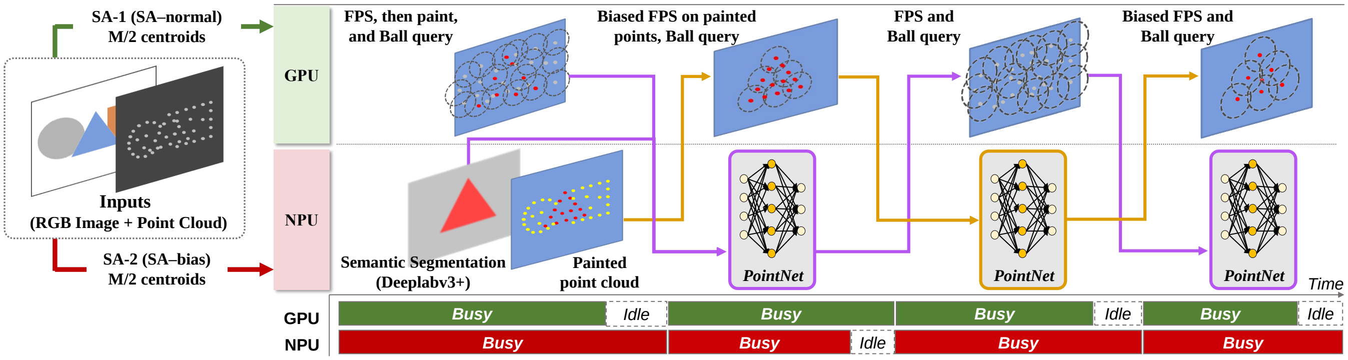


Figure 3: Illustration of *PointSplit*'s parallelized set abstraction (SA) pipeline. Each lightweight SA process in *PointSplit* generates only half the number of balls compared to the conventional SA layers in PointNet++. When GPU processes point manipulation for an SA pipeline, NPU processes PointNet for the other SA pipeline in parallel, which reduces idle time on each processor.

computes PointNet for SA-1 by using the semantic information and GPU performs point manipulation for the other SA pipeline SA-2.

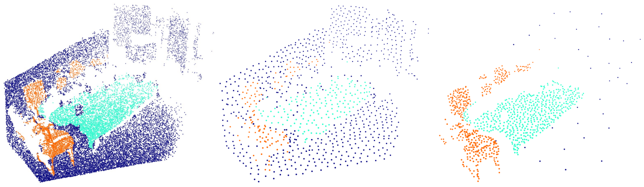
While the fundamental pipelining structure is effective in terms of latency, we aim to go further by applying different point sampling strategies for SA-1 and SA-2 to create synergy between the two for accuracy improvement. In addition, since the NPU-based acceleration is meaningful only when the object detection model can maintain accuracy after fully quantized, we develop a new quantization scheme for 3D object detectors.

4.1 2D Semantics-aware Biased Point Sampling

To create synergy between the two lightweight SA processes (SA-1 and SA-2), we focus on the fact that SA-1 starts before the 2D segmentation task is finished but SA-2 starts after the 2D segmentation. This means that while both SA-1 and SA-2 utilize the semantic information when processing PointNet for feature augmentation, SA-2 can *utilize the 2D semantic information also for its point manipulation*, if it is useful. Given that PointPainting utilizes the semantic information only for neural net operations, the idea of 2D semantics-aware point manipulation is new.

Specifically, since 2D semantic information distinguishes foreground points (those on valid 3D objects) from background points, we propose *semantics-aware biased point sampling* by giving different priorities for foreground and background points when performing point sampling (FPS in the case of PointNet++). The intuition is that point sampling with a biased distribution can generate an augmented view for PointNet from the same 3D scene, which improves the model's detection performance. Since a 3D input scene for PointNet consists of *sampled points* instead of the whole point cloud, multiple different (augmented) inputs can be generated from an original point cloud scene depending on how the input points are sampled.

To apply the semantics-aware biased sampling strategy for the FPS method, we manipulate the distance between two 3D points p_1 and p_2, denoted as $d(p_1, p_2)$, according to the type of the two points (foreground or background). Considering a point set $\mathcal{S}$ and its subset $\mathcal{A}$ ($\subset \mathcal{S}$) comprising the foreground points in $\mathcal{S}$, we re-define the distance metric $d(p_1, p_2)$ as follows:



(a) Point cloud input, painted with semantics (b) Result of normal FPS ($w_0 = 1$) (c) Result of biased FPS ($w_0 = 10$)

Figure 4: Illustration of *PointSplit*'s semantics-aware biased point sampling. Using the same point cloud scene, our biased sampling can create significantly different multiple views by controlling the weight coefficient w_0.

$$\begin{aligned} d(p_1, p_2) &= w * \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}, \\ \text{where } w &= \begin{cases} w_0 & \text{if } p_1 \in \mathcal{A} \text{ or } p_2 \in \mathcal{A} \\ 1 & \text{otherwise} \end{cases} \end{aligned} \tag{1}$$

Here (x_1, y_1, z_1) and (x_2, y_2, z_2) are the 3D coordinates of p_1 and p_2, respectively. In addition, w_0 is a weight coefficient that can prioritize (when $w_0 > 1$) or de-prioritize (when $w_0 < 1$) foreground points in the FPS process. For example when w_0 is larger than 1, the distance metric intentionally increases distance between p_1 and p_2 if at least one of them is included in $\mathcal{A}$. If both points are in $\mathcal{A}^c$, their distance is calculated normally. Thus, points in $\mathcal{A}$ are more likely to be selected as the farthest point in each iteration of FPS.

Figure 4 illustrates the impact of different w_0 values on the result of FPS. When $w_0 = 1$, points are sampled equally from both the foreground and background areas as the regular FPS does (Figure 4(b)). When a large weight is given to the painted (foreground) area ($w_0 = 10$), most points are sampled from the painted area (Figure 4(c)). The impact of w_0 value on the performance will be evaluated in Section 6.2. Overall, for a single point cloud input (Figure 4(a)), our biased sampling strategy can produce different multiple views.

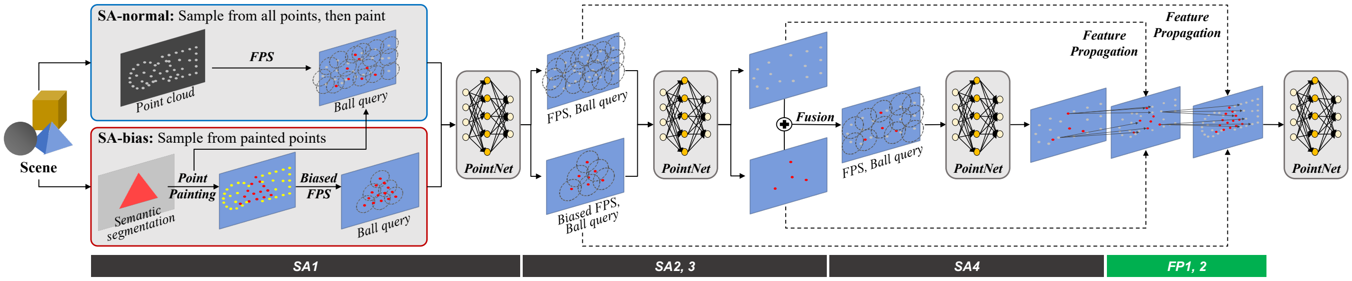


Figure 5: Illustration of PointNet++ structure optimized for *PointSplit*. (1) An input point cloud is divided into two heterogeneous SA pipelines, one with regular FPS and the other with biased FPS. (2) The two SA pipelines share a single PointNet for data augmentation effect. (3) The two SA pipelines are merged before the fourth SA layer. (4) After SA layers, two FP layers are processed back to back and the last single PointNet produces the final output.

Table 1: Comparison of the amount of computation and model size between the feature propagation (FP) layers in PointNet++ and *PointSplit*

Components	**PointNet++** Two PointNets	***PointSplit*** One modified PointNet
# of Parameters	398,336	197,888
MAdd	304 M	202 M

4.2 PointNet++ Optimization for Parallism

With the two separate lightweight SA pipelines, SA-1 with regular sampling and SA-2 with biased sampling, we optimize the PointNet++ architecture to perform the two SA pipelines simultaneously on GPU and NPU, as illustrated in Figure 5. From now, we call SA-1 SA-normal and SA-2 SA-bias. Assume that an input point cloud for a regular SA layer has N points and $M(< N)$ centroids are sampled in the point manipulation stage. For SA-normal in *PointSplit*, $M/2$ centroids (*half compared to the regular SA*) are sampled under regular FPS without using 2D semantic information. These centroids help the network to capture the overall context of the 3D scene. For SA-bias, another set of $M/2$ centroids is sampled under biased FPS with more weight given to the foreground points. The biased point set contains more information for objects. We use $w_0 = 2$ for biased FPS on foreground points, which will be discussed in Section 6.2.

We fine-tune the PointNet++ architecture to improve accuracy. Importantly, among the four SA layers[4] in PointNet++, the SA-bias pipeline uses biased FPS only for its first two SA layers; normal FPS is applied for the subsequent SA layers to capture the overall context at the end. In addition, the two sets of centroids from SA-normal and SA-bias are fused before the last (fourth) SA layer. As for the neural network part (i.e., PointNet), we do not separately train two versions of PointNet for the two lightweight SA pipelines but train a single PointNet for both SA-normal and SA-bias. By doing so, we not only keep the network size from increasing but also expose the network to more diverse inputs with different characteristics, enabling robust detection (i.e., data augmentation effect).

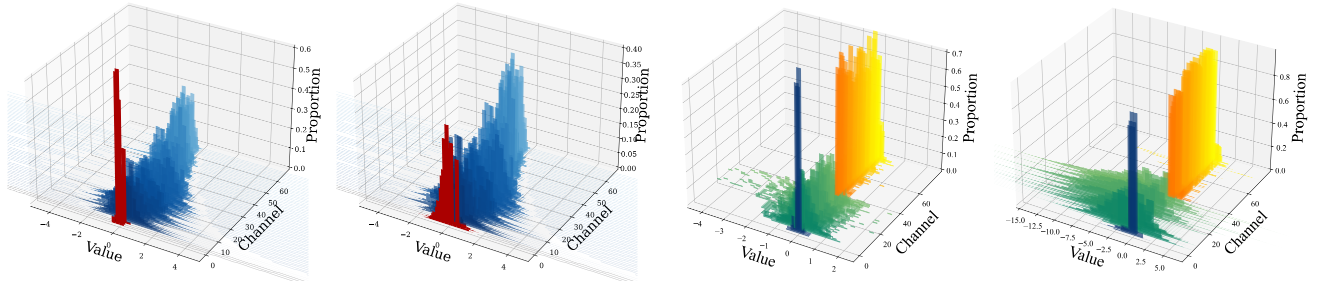
Lastly, PointNet++ has two feature propagation (FP) layers after the four SA layers, each of which includes point manipulation and PointNet similar to an SA layer. For the two FP layers, we do not maintain parallel processing because the two point sets are already fused before the fourth SA layer. In addition, we remove PointNet from each FP layer and attach a single shared fully-connected layer at the end of the second FP layer. As shown in Table 1, this simple modification achieves multiple advantages: reducing communication overhead between GPU and NPU, the number of model parameters 50.3%, and the computation overhead 33.6%. Despite the size and computation reduction from this change, we confirmed that it does not hurt the detection accuracy of the model.

4.3 Role-based Group-wise Quantization

Quantization is necessary to accelerate 3D object detection on an edge device but should be done carefully to not lose accuracy, given that 3D object detection is a complicated task. To this end, we carefully consider how to set quantization granularity. Various levels of granularity have been proposed, such as layer-, channel-, and group-wise quantization [26, 28, 32, 53]. As can be inferred from the names, these techniques determine the clipping range for weights or activations depending on their distributions in an entire layer, each channel, or a group of several channels, respectively. Channel-wise quantization is the most sophisticated method, providing the best accuracy but requiring the largest number of quantization parameters. On the other hand, layer-wise quantization requires relatively small overhead but results in significant accuracy loss. Group-wise quantization is halfway between channel- and layer-wise quantization in terms of both the overhead and the accuracy loss. However, simply selecting one of the existing options might end up with inefficient quantization since model characteristics are not considered.

For accurate quantization using a small number of parameters, we observe distributions of activations and weights in VoteNet, finding out that each channel's weight and activation distributions vary greatly in the last layer of voting and proposal modules. We analyze the model structure and reveal that different distributions between groups of channels at a single layer come from their *different roles*. Both the voting and proposal modules of VoteNet produce heterogeneous outputs that consist of xyz-coordinates, features of the resulting points, bounding box size, etc. For example, the proposal module consists of different channels in charge of center regression, heading bin regression and classification, size regression and classification, and object classification. We observe that the

[4] For the first three SA layers, the number of centroids for each normal SA and biased SA is 1024, 512, and 256, respectively. The radius for the ball query is 0.2, 0.4, 0.8, and 1.2 in each of the four SA layers, as in VoteNet.



(a) Voting weights (b) Voting activations (c) Proposal weights (d) Proposal activations

Figure 6: The distributions of weights and activations in voting/proposal module. Channels in different group are marked as different color in the figure. Since there are too many channels for visualization in the voting module, the values in 4 consecutive channels are grouped as a single distribution.

Table 2: Three groups divided according to the role of channels in the proposal module of VoteNet

Role-Group	Channels	# of channels
Group1	xyz-coordinates	3
Group2	Objectness score	2
	Heading bin classification	12
	Size classification	# of classes
	Objectness category classification	# of classes
Group3	Heading bin regression	12
	Size regression	# of classes × 3

distributions of weights and activations in channels appear similar according to their roles.

Importantly, we further discover that the distributions can be grouped according to whether the channel is responsible for classification or regression. To utilize this characteristic, we group channels in the layer according to each channel's role. In the voting module, channels are divided into two groups: the one in charge of predicting xyz-coordinates and the other for predicting the features. In the proposal module, channels are divided into three groups as shown in Table 2: the one in charge of predicting xyz-coordinates, another for heading bin, size cluster, and object classification, and the last group for regressing the size and orientation of the bounding box. We use post-training quantization [28] to fully quantize the weights and activations to 8-bit integer.

To clarify our role-based grouping, we rearrange the order of the channels in both the voting and proposal modules in the last layer of VoteNet according to their role-based groups and plot the distributions of their weights and activations in Figure 6. The figure confirms that each channel's weight and activation distributions vary greatly in the last layer of voting and proposal modules, depending on its role: which type of outputs to take charge of. For example, As shown in Figures 6(c) and 6(d), for the first three channels of the proposal module (i.e., blue bars, Group 1 in Table 2), weight and activation values are densely distributed around the mean value and min/max range is small. On the other hand, the next group of 24 channels (i.e., green bars, Group 2 in Table 2) has a more dispersed distribution of weights and activations. As another visualization, Figure 7 shows Kullback-Leibler (KL) divergence of activations in a proposal module of VoteNet. The figure confirms that distribution difference between activations from different channels is noticeable when channels are in different role-groups.

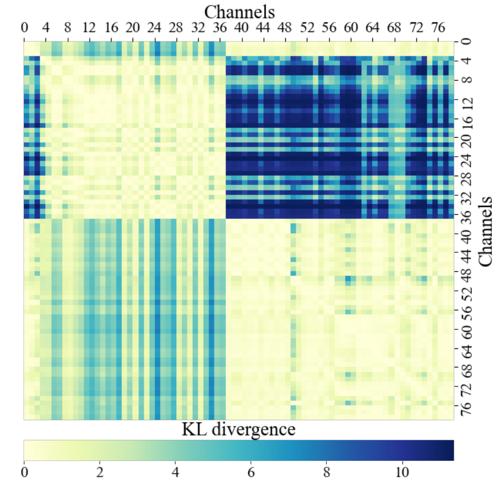


Figure 7: Kullback-Leibler (KL) divergence of activations in a proposal module of VoteNet. Dark blue implies larger KL divergence. KL divergence between different role-based channel groups has greater magnitude (e.g. group 3-27 vs 28-69).

5 IMPLEMENTATION

5.1 Datasets

We train/test *PointSplit* on two representative datasets for indoor 3D scene understanding: SUN RGB-D [56] and Scannet V2 [15].

SUN RGB-D (the primary dataset). Given that each SUN RGB-D image is a single RGB-D shot, SUN RGB-D is the primary dataset that fits our scenario in which an edge device performs inference on a single RGB-D shot. The SUN RGB-D dataset includes 10,335 RGB-D images taken indoors. 5,285 images are used for training and 5,050 images are used for validation. Segmentation annotations are provided for RGB images and 3D oriented bounding boxes of 37 categories are provided. We use the same data preparation step in VoteNet [46] including conversion of the depth images into point clouds and data augmentation.

Table 3: Per-class accuracy (mAP) at IoU threshold 0.25 of two VoteNet implementations on SUN RGB-D: (1) the original Pytorch implementation [46] and (2) our own TensorFlow implementation. Our implementation provides comparable performance to the original version, serving as an open implementation that can be converted into TensorFlow Lite for on-device inference.

Item	Bathtub	Bed	Bookshelf	Chair	Desk	Dresser	Nightstand	Sofa	Table	Toilet	Overall
VoteNet-Pytorch [46]	**74.4**	83.0	**28.8**	**75.3**	22.0	29.8	**62.2**	**64.0**	47.3	**90.1**	**57.7**
VoteNet-TensorFlow (ours)	72.4	**84.0**	25.3	74.1	**24.2**	**30.0**	61.4	61.6	**49.7**	86.8	56.9

Scannet V2 (the secondary dataset). In contrast to SUN RGB-D, a Scannet V2 scene is not constructed from a single RGB-D shot but ~1,500 shots with various different views that scan a ~20× wider area more completely, resulting in much less occlusion and richer annotations. However, due to the scanning process, it takes a long time to get a scene for inference, which is not proper for our scenario. Therefore, we utilize Scannet V2 as the secondary dataset.

Given that fusing 2D semantic information of all the 1,500 images is not practical, we randomly select *only three 2D images* to evaluate the impact of 2D-3D fusion. The unbalanced information between 2D and 3D (i.e., using only three out of 1,500 2D images but a 3D point cloud containing all the 1,500 shots) is unfavorable for *PointSplit*. The reason why we include Scannet V2 even though it is less practical and unfavorable for *PointSplit* is to show that *PointSplit* generally works well for multiple datasets.

Scannet V2 includes 1,513 scanned 3D indoor scenes and objects with 18 classes. 1,201 scans are used for training and 312 scans are used for validation. We also use RGB images and segmentation labels exported from the scanning stream for 2D-3D fusion.

5.2 Platform with Multi-type Accelerators

3D detector implementation on TensorFlow. Although there are various DNNs that run on powerful servers, implementing them to run on an edge device is a labor-intensive and time-consuming task. Specifically, given that 3D object detection has been unexplored in the on-device ML regime, VoteNet is implemented only on Pytorch that is not an edge-friendly platform yet. To overcome the hurdle, we implement and train VoteNet on Tensorflow *from scratch* and achieve comparable performance to the original Pytorch version [46], as shown in Table 3. Thus, this work serves as the *first open implementation* of VoteNet on TensorFlow, which can easily be converted into a TensorFlow Lite model to test on-device inference. Specifically, we use Adam optimizer with an initial learning rate of 0.001. The learning rate is decreased by 10 times after 80 and 120 epochs. We train the model for 180 epochs and it takes around 10 hours on one RTX 3090 with Intel Xeon® Silver 4216 CPU on SUN RGB-D dataset [56] and 5 hours on Scannet V2 dataset [15].

Hardware platform. To measure inference speed, we build a low-power platform with multi-type accelerators by combining Google Coral M.2 accelerator with NVIDIA Jetson Nano, as shown in Figure 8. NVIDIA Jetson Nano includes a quad-core ARM A57 CPU, 128-core Maxwell GPU of 512 GFLOPS and 4 GB 64-bit LPDDR4 memory. Coral M.2 accelerator includes an EdgeTPU coprocessor - an ASIC chip built for neural network inference, which is capable of 4 trillion operations per second. Coral M.2 accelerator is connected to Jetson Nano via PCIe Gen2 x 1 and shares its main DRAM memory. Given that Coral EdgeTPU only supports integer operations, we quantize our model to INT8 and convert our model into TensorFlow Lite to compile it to be EdgeTPU-compatible. Note that although this platform is a single device including both GPU and EdgeTPU, it would be more ideal to utilize an integrated SoC as technology evolves. For example, although not available when we started this work, Apple's recent M1 architecture is designed for CPU, GPU, and NPU to share cache or memory.

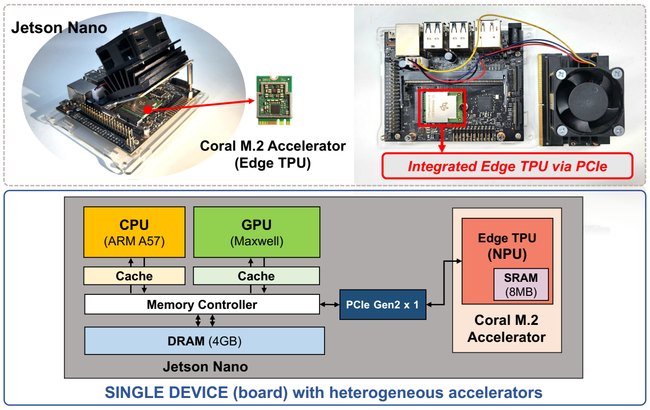


Figure 8: Test environment with heterogeneous processors. We use NVIDIA Jetson Nano (GPU) equipped with Google Coral M.2 accelerator (EdgeTPU).

5.3 Deeplabv3+ for 2D Semantic Segmentation

To implement PointPainting, we use Deeplabv3+ [8] with MobileNet V2 backbone [51] as a semantic segmentation network. We first pretrain Deeplabv3+ on COCO dataset [35] and fine-tune the weights on each target dataset, SUN RGB-D and Scannet V2. For fine-tuning for a target dataset, we use images and semantic segmentation labels in the target dataset along with COCO dataset. We oversample some under-represented classes 5 times for the model to better locate those classes, as proposed in [31]. The oversampled classes include desk, dresser, night stand, bookshelf, bathtub from SUN RGB-D and window, bookshelf, picture, counter, desk, curtain, shower curtain, garbage bin from Scannet V2. We use SGD optimizer with momentum 0.9 and initial learning rate 0.05, and decay learning rate 0.94 on every 2,000 training steps. The final mIoU on SUN RGB-D validation images is 40.7%, and the final mIoU on ScannetV2 validation images is 47.7%. Detailed per-class accuracy on both datasets are summarized in Tables 4 and 5, respectively.

6 EXPERIMENTS

6.1 Experimental setup

Following the recent practice in VoteNet, we use mean average precision (mAP) at 0.25 IoU threshold as our evaluation metric. The evaluation result is reported on the 10 most common categories for SUN RGB-D validation data, and 18 object categories for Scannet V2

Table 4: Semantic segmentation accuracy (mIoU) of Deeplabv3+ on 2D images in the SUN RGB-D validation dataset.

Item	Bathtub	Bed	Bookshelf	Chair	Desk	Dresser	Nightstand	Sofa	Table	Toilet	Overall
mIoU	34.4	50.4	17.9	55.9	16.9	25.8	22.2	41.4	40.1	54.9	40.7

Table 5: Semantic segmentation accuracy (mIoU) of Deeplabv3+ on 2D images in the Scannet V2 validation dataset.

Item	Cab	Bed	Chair	Sofa	Table	Door	Wind	Bkshf	Pic	Cntr	Desk	Curt	Fridg	Showr	Toil	Sink	Bath	Gbg	Overall
mIoU	45.8	53.2	50.8	55.4	60.6	40.7	25.5	20.7	28.5	28.1	39.5	43.2	54.3	45.4	79.3	54.4	66.5	35.1	47.8

Table 6: Per-class accuracy (mAP) at IoU threshold 0.25 of various 3D object detectors on SUN RGB-D (our primary dataset). *PointSplit* (FP32) provides the best accuracy for 4 out of 10 classes, resulting in the best overall mAP performance. After quantized, *PointSplit* still performs comparably to PointPainting and significantly outperforms VoteNet.

Item	Bathtub	Bed	Bookshelf	Chair	Desk	Dresser	Nightstand	Sofa	Table	Toilet	Overall
VoteNet (FP32)	**72.4**	84.0	25.3	74.1	24.2	30.0	61.4	61.6	49.7	86.8	56.9
PointPainting (FP32)	68.0	**86.5**	29.6	74.1	24.6	**39.9**	**61.8**	77.9	49.3	90.0	60.2
RandomSplit (FP32)	61.9	85.6	33.8	74.5	26.4	38.7	61.7	**79.7**	**52.8**	88.9	60.4
PointSplit (FP32)	69.0	86.0	**34.0**	**74.9**	**27.0**	39.7	60.1	78.5	51.8	**92.5**	**61.4**
PointSplit (INT8)	62.7	86.3	33.0	74.4	25.5	39.3	58.9	77.6	50.6	90.5	59.9

validation data. As in VoteNet, we do not consider the bounding box orientation for Scannet V2 evaluation. As mentioned in Section 5, to fuse 2D information with a 3D point cloud scene, we use a single RGB image for the SUN RGB-D dataset and three RGB images for the Scannet V2 dataset. In addition, according to the standard practice for each dataset, we randomly sample 20,000 points and 40,000 points from an original point cloud of SUN RGB-D and Scannet V2, respectively, to construct an input point cloud for the first SA layer of PointNet++. Given that a Scannet V2 scene covers nearly 20 times larger area than a SUN RGB-D scene, an input point cloud in Scannet V2 is sparser than that in SUN RGB-D. Note that the results in SUN RGB-D is more important since it fits our scenario, while those in Scannet V2 show *PointSplit*'s general applicability.

We measure the latency on Jetson Nano equipped with EdgeTPU. We do three warm-up runs and experiment 20 times, then report averaged latency. Per each experiment, the latency is measured to process four 3D scenes and averaged to report per-scene latency.

Table 7: mAP of various VoteNet-based 3D object detectors on SUN RGB-D and Scannet V2, measured at IoU thresholds of 0.25 and 0.5. *PointSplit* provides the best mAP in most cases.

Precision	Method	Dataset	
		SUN RGB-D @0.25 / @0.5	Scannet V2 @0.25 / @0.5
FP32	VoteNet	56.9 / 31.1	54.9 / 30.4
	PointPainting	60.2 / **32.8**	**56.4** / 31.7
	RandomSplit	60.4 / 32.0	55.2 / 31.2
	PointSplit	**61.4** / 32.7	56.1 / **32.4**
INT8	VoteNet	29.3 / 3.0	41.7 / 11.6
	PointPainting	32.3 / 3.2	48.8 / 18.2
	PointSplit	**59.9** / **32.5**	**55.7** / **30.3**

6.2 Detection Accuracy

Analysis on SUN RGB-D (primary dataset). Table 6 shows per-class detection accuracy (mAP) of various 3D object detectors on the SUN RGB-D dataset. VoteNet relies only on a point cloud without fusing 2D information, providing the lowest accuracy. PointPainting, the baseline network, combines Deeplabv3+ and VoteNet to fuse 2D semantic information for 3D object detection, which significantly improves accuracy over VoteNet (+3.3 mAP). This clearly shows the advantage of 2D-3D fusion for 3D object detection.

Interestingly, although our *PointSplit* (full precision) originally focuses on efficient pipelining for multi-type accelerator environments, it ends up with *even better accuracy* compared to PointPainting (+1.2 mAP). Specifically, out of 10 classes in SUN RGB-D, *PointSplit* (full precision) achieves the best accuracy for 4 classes and the second best accuracy for other 4 classes. This verifies the effectiveness of our semantics-aware biased point sampling. By building two separate lightweight SA pipelines that have different views of a single 3D scene, with regular FPS and biased FPS, respectively, and making both pipelines pass through the same PointNet, *PointSplit* trains PointNet more robustly with data augmentation.

For comparison, we also test an ablated version of *PointSplit*, called RandomSplit, which randomly divides an entire point set into two sets and applies regular FPS for both SA pipelines. Without biased sampling, RandomSplit does not provide an augmented view, resulting in similar accuracy to PointPainting. Lastly, when *PointSplit* is quantized with 8-bit precision, accuracy is dropped marginally due to our role-based group-wise quantization. As a result, *PointSplit*, even after quantized, performs comparably to the baseline PointPainting (full precision).

Analysis on multiple datasets. To show *PointSplit*'s accuracy gain more generally, Table 7 summarizes accuracy performance of various schemes on both SUN RGB-D and Scannet V2 datasets, before and after quantization. After quantized (layer-wise), both VoteNet and PointPainting experience remarkable performance degradation, which verifies our claim: activation and weight distributions in a single layer are too different to quantize using a single parameter set. In contrast, our *PointSplit* (INT8) improves performance with very large margins (up to +30.6 mAP@0.25) compared to both VoteNet and PointPainting in both datasets and performs

Table 8: mAP of *PointSplit* combined with GroupFree3D [38] and RepSurf [48] on SUN RGB-D and Scannet V2. Scannet V2 experiments use 5 2D images for 2D-3D fusion. (6,256) means that the GroupFree3D model has 6 decoder layers and uses 256 object candidates.

Method	Dataset	
	SUN RGB-D	Scannet V2
	@0.25 / @0.5	@0.25 / @0.5
Baseline: GroupFree3D$^{(6,256)}$	58.0 / 38.3	63.7 / 38.8
Baseline + PointPainting	62.5 / **43.3**	66.7 / 41.2
Baseline + RandomSplit	61.9 / 40.4	66.6 / 33.7
Baseline + *PointSplit*	**62.6** / 42.5	**67.8** / **45.4**
Baseline: RepSurf-U + GroupFree3D$^{(6,256)}$	61.4 / 41.3	65.0 / 41.0
Baseline + PointPainting	63.1 / 41.8	67.4 / 42.7
Baseline + RandomSplit	62.5 / 40.8	67.0 / 43.3
Baseline + *PointSplit*	**63.5** / **42.1**	**68.5** / **46.7**

even comparably to *PointSplit* (FP32). This demonstrates the effectiveness of our role-based group-wise quantization scheme.

Although detailed trends are different due to different scene characteristics, the results in Scannet V2 also confirm that both semantics-aware biased point sampling and role-based group-wise quantization scheme significantly contribute to performance improvement. Specifically, we observe that RandomSplit degrades accuracy compared to PointPainting (-1.2 mAP@0.25) but *PointSplit* recovers performance successfully. Given that point representation in Scannet V2 is already sparse (much sparser than that in SUN RGB-D, as mentioned in Section 6.1), sampling only half the number of points for each SA pipeline should be done carefully to not lose accuracy. The performance gap between RandomSplit and *PointSplit* verifies the validity of our biased point sampling in this aspect.

Analysis on recent, heavy 3D object detectors. The latest state-of-the-art 3D object detectors on the SUN RGB-D and Scannet V2 leaderboards adopt heavy and edge-unfriendly transformer architectures [38, 48, 67, 69]. However, it is valuable to evaluate the effectiveness of our biased point sampling and parallel pipelining when applied to these models in terms of accuracy. To this end, we implement recent transformer-based GroupFree3D [38] and RepSurf [48] on TensorFlow and apply PointPainting, RandomSplit, and *PointSplit* to the two heavy baselines.[5] Given that this evaluation is not for efficiency, we do not apply quantization and utilize two PointNets at the FP layers again (i.e., excluding the optimization in Table 1) when implementing *PointSplit* to focus on better accuracy. The results in Table 8 demonstrate that *PointSplit* successfully improves mAP when combined with GroupFree3D and RepSurf on both SUN RGB-D and Scannet V2 using two parallel SA pipelines. This finding confirms that *PointSplit* is a viable technique for improving the accuracy of multiple 3D object detectors.

Deeper look into biased point sampling. We analyze the impact of detailed design choices for the semantics-aware biased point sampling on performance. To this end, Table 9 shows *PointSplit* performance on SUN RGB-D with varying w_0 value. As mentioned in Section 4.1, as w_0 increases, the biased point sampling mechanism selects more foreground points than background points. The results in Table 9 show that as w_0 increases, mAP performance first increases but decreases again. Specifically, the highest accuracy is achieved when $w_0 = 2$. The results show that proper balance between foreground and background points is important when constructing an augmented scene via the biased point sampling. Specifically, sampling more foreground points turns out to be beneficial but sampling too many foreground points is detrimental.

Table 9: Accuracy of *PointSplit* on SUN RGB-D with varying w_0 for the semantics-aware biased point sampling. The performance is maximized when point sampling is slightly biased toward foreground points.

Weight	0.5	1.0	1.5	2.0	2.5	3.5
mAP	60.3	60.4	61.3	**61.4**	59.6	59.4

Table 10: Accuracy of *PointSplit* on SUN RGB-D when the semantics-aware biased point sampling is applied to various SA layers.

SA layers with biased FPS	mAP
SA1 only	60.4
SA1 and SA2	**61.4**
SA1, SA2 and SA3	60.1
All SA layers	60.8

Next, we evaluate another design choice for the biased point sampling: which SA layers to apply the biased point sampling among the four SA layers in PointNet++. To this end, Table 10 shows *PointSplit* performance on SUN RGB-D when our biased sampling technique is applied to various SA layers, from the first (SA1) to the last (SA4). The results show that applying the biased point sampling to the first two layers provides the best performance but applying it to more SA layers causes performance degradation again. Given that applying the biased point sampling to multiple layers consecutively increases the bias level, the results verify again the need for balancing foreground and background points to maximize *PointSplit* performance.

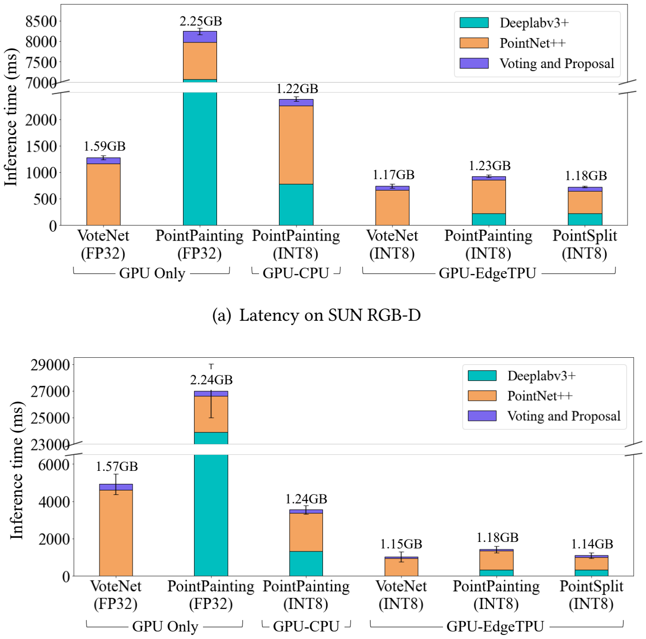
Impact of quantization methods. Table 11 evaluates *PointSplit* on the two datasets with varying quantization granularity: layer-wise, group-wise, channel-wise, and our role-based group-wise methods. For the group-wise method, we group the entire layer into 2 (for the voting module) or 3 (for the proposal module) groups of an equal number of channels without considering their roles.

The results show that both layer-wise and group-wise quantization methods suffer from significant quantization errors but the channel-wise method incurs only marginal errors. This verifies that channels in a single layer of a 3D object detector have very different weight and activation distributions, which requires fine-grained quantization. The channel-wise quantization, however, is inefficient since it requires more than 1,300 parameters to quantize a single layer. On the other hand, our role-based group-wise quantization achieves the sweet spot. It requires the same number of quantization parameters compared to the naïve group-wise quantization, 67× and 71× less than that of the channel-wise quantization on SUN RGB-D and Scannet V2, respectively. With such small number

[5] GroupFree3D employs a PointNet++ backbone and a transformer-based detection head [38] while RepSurf improves the input representation of GroupFree3D [48]. Both models have been re-implemented and trained on TensorFlow, leveraging the hyperparameters of their respective Pytorch implementations. As a result, the TensorFlow-based models achieved a lower mAP compared to their original counterparts. It is worth noting that identifying optimal hyperparameters for TensorFlow could potentially improve their performance, which is out of the scope of this paper.

Table 11: mAP at IoU threshold 0.25 of *PointSplit* on SUN RGB-D and Scannet V2, with various quantization methods. While performing similarly to the most fine-grained channel-wise method, our role-based group-wise quantization method remarkably reduces the number of quantization parameters, 67× and 71× less parameters on SUN RGB-D and Scannet V2, respectively.

Quant. method	Precision	SUN RGB-D			ScannetV2		
		mAP	Quant. error	# of quant. parameters	mAP	Quant. error	# of quant. parameters
No quant.	FP32	61.4	-	-	56.1	-	-
Layer-wise	INT8	24.2	37.2	8	51.9	4.2	8
Group-wise	INT8	26.3	35.1	20	52.3	3.8	20
Channel-wise	INT8	61.0	0.4	1352	55.5	0.6	1424
Role-based group-wise (ours)	INT8	59.9	1.5	20	55.4	0.7	20



(a) Latency on SUN RGB-D

(b) Latency on Scannet V2

Figure 9: Per-scene latency and peak memory of 3D object detectors. Latency on Scannet V2 is longer than that on SUN RGB-D due to more 3D points. Compared to running PointPainting (FP32) only on GPU, *PointSplit* (INT8) is faster 11.4 times on SUN RGB-D and 24.7 times on Scannet V2.

of parameters, our scheme dramatically improves accuracy over the group-wise quantization (+33.6 mAP on SUN RGB-D), and provides similar mAP compared to the heavy channel-wise quantization. This demonstrates the tight relationship between each channel's value distribution and its role in a 3D object detector.

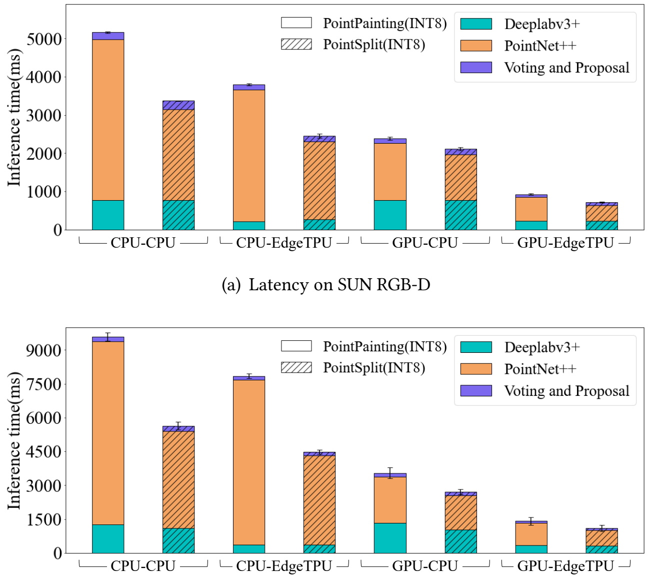
6.3 System Performance

Latency analysis. Figure 9 shows average latency for single-scene inference of various schemes, measured on our platform comprising Jetson Nano and Coral. Generally, latency on Scannet V2 is longer than that on SUN RGB-D. This is because an input point cloud is twice larger and Deeplabv3+ runs three times more for a Scannet V2 scene than a SUN RGB-D scene, as in Section 6.1. When using GPU only, PointPainting significantly increases latency compared to VoteNet, requiring more than 8 and 27 seconds in SUN RGB-D and Scannet V2, respectively; despite its accuracy gain, 2D-3D fusion is not a viable option on classic resource-constrained devices.

In our platform including both GPU and EdgeTPU, however, the landscape can be shifted. First of all, running the point manipulation operation on GPU and the PointNet part on EdgeTPU significantly reduces latency, which shows the effectiveness of EdgeTPU that is optimized for neural net operations. In addition, although 2D-3D fusion using PointPainting increases latency on the multi-accelerator platform, the efficient pipelining scheme in *PointSplit* nullifies the slowdown in SUN RGB-D, resulting in inference speed comparable to VoteNet (INT8) with significantly better accuracy (+30.6 mAP@0.25, as in Table 7). Compared to running PointPainting (FP32) only on GPU, running *PointSplit* (INT8) on both GPU and EdgeTPU provides 11.4× and 24.7× faster inference in SUN RGB-D and Scannet V2, respectively. Overall, the results suggest that in the upcoming multi-type accelerator era, 2D-3D fusion-based 3D object detection, which used to be a complex task, can run on an edge device without notable latency degradation.

Peak memory analysis. Figure 9 also shows peak memory usage of each scheme. Note that while VoteNet (FP32) and PointPainting (FP32) run on TensorFlow, other four schemes run on TensorFlow Lite. Since TensorFlow Lite does not support CUDA, the GPU-only environment utilizes TensorFlow.

PointPainting (FP32) on TensorFlow consumes more than 2.2 GB memory, which is one of the reasons why its latency is significantly high. We evaluate another version of PointPainting (INT8) on TensorFlow Lite by running the point manipulation operation on GPU and the PointNet (INT8) and Deeplabv3+ (INT8) part on CPU (i.e., GPU-CPU combination). The results show the impact of using a lightweight software platform: running neural nets on CPU with TensorFlow Lite is much faster and requires much less memory than on GPU with TensorFlow. Lastly, VoteNet (INT8), PointPainting (INT8), and *PointSplit* (INT8) that run on the GPU-EdgeTPU environment and TensorFlow Lite consume similar peak memory. This verifies that compared to VoteNet and PointPainting, *PointSplit*'s parallel operation of GPU and EdgeTPU does not sacrifice memory for boosting inference speed.



(a) Latency on SUN RGB-D

(b) Latency on Scannet V2

Figure 10: Per-scene inference latency of VoteNet-based PointPainting (INT8) and *PointSplit* (INT8) on various combinations of multiple processors. *PointSplit* reduces latency regardless of hardware configurations.

More hardware configurations. Next, Figure 10 evaluates the impact of *PointSplit* on various processor combinations in our platform: (1) CPU-CPU, (2) CPU-EdgeTPU, (3) GPU-CPU, and (4) GPU-EdgeTPU (i.e., our platform). In each combination, the first processor executes point manipulation (in PointNet++) and the second processor executes neural nets, such as PointNet (in PointNet++), Deeplabv3+ and voting/proposal modules (in VoteNet). The results show that hardware configuration significantly impacts latency. Specifically, using GPU as the first processor, instead of CPU, accelerates point manipulation, which reduces latency for running PointNet++. Using EdgeTPU as the second processor improves latency of all the neural nets compared to using CPU. More importantly, the results verify that *PointSplit* reduces latency on every hardware configuration compared to PointPainting (INT8). Specifically, *PointSplit* improves latency performance most significantly in the CPU-CPU and CPU-EdgeTPU cases, 1.7× and 1.8×, respectively. Note that PointPainting (INT8) provides significantly lower mAP than *PointSplit* in Table 7.

Layer-wise analysis. To take a deeper look, Table 12 shows per-layer latency of PointPainting (INT8) and PointNet++ (INT8) when using GPU and EdgeTPU without parallelization. As the layer proceeds, computation on GPU (point manipulation) monotonically decreases due to the smaller number of sampled points while that on EdgeTPU (PointNet) first increases and decreases again due to the trade-off between the input size and the number of channels. The results verify that running point manipulation for SA-normal on GPU while fusing 2D-3D information on EdgeTPU for SA-bias significantly reduces latency. In addition, given that GPU needs much more time than EdgeTPU at SA1, adding more layers to PointNet in SA1 and process the layers on EdgeTPU using the idle time might improve accuracy without sacrificing latency.

Table 12: Per-layer latency of PointPainting (INT8) and PointNet++ (INT8) with sequential pipelining.

Layers	GPU	EdgeTPU
2D-3D fusion	-	222 ms
SA1	199 ms	47 ms
SA2	52 ms	71 ms
SA3	25 ms	84 ms
SA4	20 ms	21 ms

Table 13: Latency in communication and computation on GPU and EdgeTPU when *PointSplit* processes a single SUN RGB-D scene. For the ease of measurement, the latency to run DeeplabV3+ is not included and multithreading is not used (SA-normal and SA-bias are executed sequentially).

Processor	Latency (ms)		
	Communication	Computation	Total
GPU	80	248	328
EdgeTPU (estimates)	360	121	481

Inter-processor communication. Utilizing multiple accelerators requires data exchange between the accelerators, which brings a concern of inter-processor communication overhead. Table 13 quantifies the communication overhead by dividing *PointSplit*'s inference latency on a SUN RGB-D scene into communication and computation latency. To focus on PointNet++ operation, we exclude 2D-3D fusion. While latency on GPU is measured by NVIDIA profiler, without such a tool, that on EdgeTPU is estimated in the following way. We first measure the time required to execute each PointNet in EdgeTPU, denoted as $t_{p,total}$, which includes latency for both communication and computation: $t_{p,total} = t_{p,comp} + t_{p,comm}$. Then we build another PointNet that has the same size of input, output, and the number of parameters, but doubles the amount of computation. The time for executing the new PointNet model, $t_{p2,total}$, includes the same communication time but twice longer computation time: $t_{p2,total} = 2 \times t_{p,comp} + t_{p,comm}$. Therefore, we estimate computation latency on EdgeTPU as the difference between the two measurements: $t_{p,comp} = t_{p2,total} - t_{p,total}$. Then the remainder is regarded as communication latency: $t_{p,comm} = t_{p,total} - t_{p,comp}$.

The results verify that communication overhead on our platform is significant indeed, taking up 54.4% of the total latency. Specifically, communication time on EdgeTPU is 4.5× longer than that on GPU due to the use of a slower channel, PCIe Gen 2 x 1 (0.5 GB/s). With a short glimpse, the significant communication overhead seems to suggest that parallization among heterogeneous low-power processors might have limited gain. However, *the real implication is opposite*: once resource-constrained hardware evolves further and solves the communication problem, which is actually happening these days, *PointSplit*'s inference speed can be nearly doubled. With the latest off-the-shelf hardware equipped with multi-type accelerators, such as Apple's M1 architecture, we expect the field of on-device ML to evolve further with parallel processing.

7 DISCUSSION

In this section, we discuss applicability and limitations of *PointSplit*. In addition, we present practical challenges (i.e., entry barriers) for researching on-device 3D object detection, which we have experienced while developing *PointSplit*, the first framework for 100% on-device 3D object detection with heterogeneous accelerators.

7.1 Generalization and Limitations

2D semantics-aware biased point sampling (§4.1)). Although our biased point sampling method is implemented on PointNet++, the idea of biased point sampling is not specific to PointNet++. Our method can be directly applied to any DNN that utilizes farthest point sampling (FPS) and easily adapted for other point sampling techniques. A point sampling technique has its own metric (e.g., distance or density) and our technique is applied to the sampling method by slightly modifying the metric with point semantics. For example, in case of a density-based sampling technique [36], we can simply boost a point's density-based metric value if the point is in a specific group that needs to be sampled more intensely. On the other hand, there are 3D object detection networks that do not exploit point sampling (i.e., voxel-based 3D object detectors [16, 40]) where our point sampling technique is not applicable.

PointNet++ parallelization (§4.2). It is important that our parallelization technique is not specific to VoteNet nor GroupFree3D but their backbone PointNet++, which is one of the most widely used 3D backbone networks. Many recent state-of-the-art models for 3D object detection on SUN RGB-D (our primary dataset) and ScannetV2 (our secondary dataset) employ PointNet++ as their backbone [13, 38, 42, 48, 64, 67, 69, 72]. Specifically, out of top 10 ranked methods, 7 methods on SUN RGB-D and 6 methods on ScannetV2 use PointNet++, showing that PointNet++-based models are dominating currently.

Role-based group-wise quantization (§4.3). The role-based group-wise quantization is motivated by our observation that a layer's weight/activation distributions are impacted by what role each node has. Therefore, the role-based grouping scheme can be applied to any network layer that has multiple roles, not only for VoteNet. It would be an interesting future work to evaluate the impact of role-based grouping on other 3D object detectors. In addition, although we focus on quantization in this work, investigating other compression approaches, such as knowledge distillation and pruning, can be valuable future work.

7.2 Challenges in On-device 3D Object Detection

The field of 3D object detection has experienced significant growth in recent years within the deep learning community, with a range of datasets and model implementations now available. However, the deployment of state-of-the-art models on resource-constrained devices presents several nontrivial challenges.

Model size and complexity. Given that popular leaderboards on 3D object detection primarily evaluate accuracy, many of the top-ranked models rely on transformers [38, 48, 67, 69] or custom modules [5, 13, 50, 62–64, 69] that are too heavy to run on resource-constrained devices. For example, DeMF [69], which currently ranks first on the SUN RGB-D leaderboard, reaches a peak GPU memory of 2.5 GB and requires 173 GFLOPS for its 2D detector with deformable attention [74]. Effective model compression techniques must therefore be developed in order to enable the deployment of the latest models on edge devices.

Implementation burden. Although the latest 3D object detectors are implemented using Pytorch, which consumes significant resources, they are not currently implemented on edge-friendly platforms, such as TensorFlow Lite and MNN [29]. Furthermore, many state-of-the-art models rely on recent software packages, such as `mmdetection3d` and `Minkowski`, which are not currently supported by edge devices. As a result, significant time and effort is required to re-implement state-of-the-art models on lightweight platforms that perform comparably to their Pytorch versions. We believe that our open implementation of VoteNet on TensorFlow Lite can accelerate future research on on-device 3D object detection.

8 CONCLUSION

This work began when we observed the emergence of multi-type low-power accelerators with different pros and cons. We envisioned that in the era of multi-type accelerators, a new class of intelligent tasks that used to be too heavy can be viable in the on-device ML regime when these accelerators are utilized synergistically. To investigate the potential, we have built a low-power hardware platform including both GPU and NPU, and studied on-device 3D object detection with 2D-3D information fusion.

Specifically, we propose *PointSplit*, a novel 3D object detection framework that provides system-algorithm joint optimization. First, *PointSplit* catches the difference between point manipulation and neural net operation in a representative 3D feature extractor (PoinNet++), executing the former on GPU and the latter on NPU. Second, *PointSplit* creates two separate but synergistic feature extraction pipelines by augmenting a point cloud scene with 2D semantic information (i.e., semantics-aware biased point sampling). The PointNet++ structure is further optimized to maximize accuracy and efficiency in the *PointSplit* framework. Third, *PointSplit* contains role-based group-wise quantization that quantizes a 3D object detector with a small number of parameters without sacrificing accuracy. Our experiments demonstrate the effectiveness of *PointSplit* in terms of both accuracy and latency. We believe that this work, by showing the potential of recently available edge devices equipped with heterogeneous low-power processors, and by providing open implementation, can inspire other researchers to run more various complex tasks on the new class of edge devices.

ACKNOWLEDGMENTS

This work was supported partly by Creative-Pioneering Researchers Program through Seoul National University and partly by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2023-00216821, Development of Beyond X-verse Core Technology for Hyper-realistic interactions by Synchronizing the Real World and Virtual Space). Hyung-Sin Kim is the corresponding author.

REFERENCES

[1] Kittipat Apicharttrisorn et al. 2019. Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In *Proceedings of the 17th ACM Conference on Embedded Networked Sensor Systems*. 96–109.

[2] Yuxuan Cai et al. 2021. YOLObile: Real-Time Object Detection on Mobile Devices via Compression-Compilation Co-Design. *Proceedings of the AAAI Conference on*

Artificial Intelligence 35, 2 (May 2021), 955–963.

[3] Qi. Charles et al. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 77–85.

[4] Jintai Chen et al. 2020. A Hierarchical Graph Network for 3D Object Detection on Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

[5] Jintai Chen et al. 2020. A hierarchical graph network for 3D object detection on point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 392–401.

[6] Kaifei Chen et al. 2018. Marvel: Enabling mobile augmented reality with low energy and low latency. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems.* 292–304.

[7] Liang-Chieh Chen et al. 2014. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062* (2014).

[8] Liang-Chieh Chen et al. 2017. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence* 40, 4 (2017), 834–848.

[9] Liang-Chieh Chen et al. 2017. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587* (2017).

[10] Liang-Chieh Chen et al. 2018. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV).* 801–818.

[11] Peng Chen et al. 2021. AQD: Towards Accurate Quantized Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 104–113.

[12] Xiaozhi Chen et al. 2017. Multi-View 3D Object Detection Network for Autonomous Driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[13] Bowen Cheng et al. 2021. Back-tracing representative points for voting-based 3d object detection in point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 8963–8972.

[14] Yousung Choi et al. 2022. ScriptPainter: Vision-based, On-device Test Script Generation for Mobile Systems. In *2022 21st ACM/IEEE IPSN.* IEEE, 477–490.

[15] Angela Dai et al. 2017. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 5828–5839.

[16] Jiajun Deng et al. 2021. Voxel r-cnn: Towards high performance voxel-based 3d object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 1201–1209.

[17] Caiwen Ding et al. 2019. REQ-YOLO: A resource-aware, efficient quantization framework for object detection on FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* 33–42.

[18] Michal Drozdzal et al. 2016. The importance of skip connections in biomedical image segmentation. In *Deep learning and data labeling for medical applications.* Springer, 179–187.

[19] Jonathan Frankle et al. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *7th International Conference on Learning Representations, (ICLR).*

[20] Yongjie Guan et al. 2022. DeepMix: mobility-aware, lightweight, and hybrid 3D object detection for headsets. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services.* 28–41.

[21] Song Han et al. 2015. Learning Both Weights and Connections for Efficient Neural Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1.* 1135–1143.

[22] Song Han et al. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations, (ICLR).*

[23] Geoffrey Hinton et al. 2015. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop.*

[24] Ji Hou et al. 2019. 3d-sis: 3d semantic instance segmentation of rgb-d scans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 4421–4430.

[25] Andrew Howard et al. 2019. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).*

[26] Qijing Huang et al. 2021. Codenet: Efficient deployment of input-adaptive object detection on embedded fpgas. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* 206–216.

[27] Itay Hubara et al. 2016. Binarized neural networks. *Advances in neural information processing systems* 29 (2016).

[28] Benoit Jacob et al. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

[29] Xiaotang Jiang et al. 2020. Mnn: A universal and efficient inference engine. *Proceedings of Machine Learning and Systems* 2 (2020), 1–13.

[30] Bogil Kim et al. 2020. Energy-Efficient Acceleration of Deep Neural Networks on Realtime-Constrained Embedded Edge Devices. *IEEE Access* 8 (2020), 216259–216270.

[31] Mate Kisantal et al. 2019. Augmentation for small object detection. In *9th International Conference on Advances in Computing and Information Technology (ACITY 2019).* Aircc Publishing Corporation.

[32] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* (2018).

[33] Fengfu Li et al. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711* (2016).

[34] Rundong Li et al. 2019. Fully Quantized Network for Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

[35] Tsung-Yi Lin et al. 2014. Microsoft coco: Common objects in context. In *Proceedings of the European conference on computer vision (ECCV).* Springer, 740–755.

[36] Minghua Liu et al. 2020. Morphing and sampling network for dense point cloud completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 11596–11603.

[37] Zhijian Liu et al. 2019. Point-voxel cnn for efficient 3d deep learning. *arXiv preprint arXiv:1907.03739* (2019).

[38] Ze Liu et al. 2021. Group-free 3d object detection via transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).* 2949–2958.

[39] Jonathan Long et al. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 3431–3440.

[40] Jiageng Mao et al. 2021. Voxel transformer for 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).* 3164–3173.

[41] Daniel Maturana et al. 2015. VoxNet: A 3D Convolutional Neural Network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* 922–928.

[42] Ishan Misra et al. 2021. An end-to-end transformer model for 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).* 2906–2917.

[43] Charles Qi et al. 2020. Imvotenet: Boosting 3d object detection in point clouds with image votes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

[44] Charles R Qi et al. 2017. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413* (2017).

[45] Charles R Qi et al. 2018. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 918–927.

[46] Charles R Qi et al. 2019. Deep hough voting for 3d object detection in point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).* 9277–9286.

[47] Xie Qian et al. 2020. MLCVNet: Multi-Level Context VoteNet for 3D Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

[48] Haoxi Ran et al. 2022. Surface representation for point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 18942–18952.

[49] Olaf Ronneberger et al. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention.* Springer, 234–241.

[50] Danila Rukhovich et al. 2022. FCAF3D: fully convolutional anchor-free 3D object detection. In *Proceedings of the European conference on computer vision (ECCV).* Springer, 477–493.

[51] Mark Sandler et al. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

[52] Wonik Seo et al. 2021. SLO-Aware Inference Scheduler for Heterogeneous Processors in Edge Platforms. *ACM Trans. Archit. Code Optim.* 18, 4, Article 43 (jul 2021), 26 pages.

[53] Sheng Shen et al. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 8815–8821.

[54] Shaoshuai Shi et al. 2020. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 10529–10538.

[55] Shaoshuai Shi et al. 2021. PV-RCNN++: Point-voxel feature set abstraction with local vector representation for 3D object detection. *arXiv preprint arXiv:2102.00463* (2021).

[56] Shuran Song et al. 2015. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 567–576.

[57] Zhiqing Sun et al. 2020. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.* 2158–2170.

[58] Mingxing Tan et al. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning.*

[59] Mingxing Tan et al. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[60] Mingxing Tan et al. 2020. EfficientDet: Scalable and Efficient Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[61] Sourabh Vora, Alex H Lang, Bassam Helou, and Oscar Beijbom. 2020. Pointpainting: Sequential fusion for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4604–4612.

[62] Thang Vu et al. 2022. Softgroup for 3d instance segmentation on point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2708–2717.

[63] Haiyang Wang et al. 2022. CAGroup3D: Class-Aware Grouping for 3D Object Detection on Point Clouds. *arXiv preprint arXiv:2210.04264* (2022).

[64] Haiyang Wang et al. 2022. Rbgnet: Ray-based grouping for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 1110–1119.

[65] Peng-Shuai Wang et al. 2017. O-CNN: Octree-Based Convolutional Neural Networks for 3D Shape Analysis. *ACM Trans. Graph.* 36, 4, Article 72 (July 2017), 11 pages.

[66] Siqi Wang et al. 2020. Neural Network Inference on Mobile SoCs. *IEEE Design Test* 37, 5 (2020), 50–57.

[67] Yikai Wang et al. 2022. Multimodal token fusion for vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 12186–12195.

[68] Zhiyuan Xu et al. 2021. A Co-Scheduling Framework for DNN Models on Mobile and Edge Devices with Heterogeneous Hardware. *IEEE Transactions on Mobile Computing* (2021), 1–1. https://doi.org/10.1109/TMC.2021.3107424

[69] Hao Yang et al. 2022. Boosting 3D Object Detection via Object-Focused Image Fusion. *arXiv preprint arXiv:2207.10589* (2022).

[70] Maosheng Ye et al. 2020. HVNet: Hybrid Voxel Network for LiDAR Based 3D Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[71] Juheon Yi, , et al. 2020. Heimdall: mobile GPU coordination platform for augmented reality applications. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.

[72] Zaiwei Zhang et al. 2020. H3dnet: 3d object detection using hybrid geometric primitives. In *Proceedings of the European conference on computer vision (ECCV)*. Springer, 311–329.

[73] Yin Zhou et al. 2018. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[74] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. 2020. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159* (2020).