Slimmer: Accelerating 3D Semantic Segmentation for Mobile Augmented Reality

Huanle Zhang
University of California, Davis
Davis, California, USA
dtczhang@ucdavis.edu

Bo Han*
George Mason University
Fairfax, Virginia, USA
bohan@gmu.edu

Cheuk Yiu Ip
AT&T Labs Research
Bedminster, New Jersey, USA
ipcy@research.att.com

Prasant Mohapatra
University of California, Davis
Davis, California, USA
pmohapatra@ucdavis.edu

***Abstract*—Three-Dimensional (3D) semantic segmentation is an essential building block for interactive Augmented Reality (AR). However, existing Deep Neural Network (DNN) models for segmenting 3D objects are not only computation-intensive but also memory heavy, hindering their deployment on resource-constrained mobile devices. We present the design, implementation and evaluation of Slimmer, a generic and model-independent framework for accelerating 3D semantic segmentation and facilitating its real-time applications on mobile devices. In contrast to the current practice that directly feeds a point cloud to DNN models, Slimmer is motivated by our observation that these models remain high accuracy even if we remove a fraction of points from the input, which can significantly reduce the inference time and memory usage of these models. Our design of Slimmer faces two key challenges. First, the simplification method of point clouds should be lightweight. Otherwise, the reduced inference time may be canceled out by the incurred overhead of input-data simplification. Second, Slimmer still needs to accurately segment the removed points from the input to create a complete segmentation of the original input, again, using a lightweight method. Our extensive performance evaluation demonstrates that, by addressing these two challenges, Slimmer can dramatically reduce the resource utilization of a representative DNN model for 3D semantic segmentation. For example, if we can tolerate 1% accuracy loss, the reduction could be $\sim$20% for inference time and $\sim$9% for memory usage. The reduction increases to around $\sim$27% for inference time and $\sim$15% for memory usage when we can tolerate 2% accuracy loss.**

I. Introduction

3D segmentation is a process where a given 3D input (*e.g.,* a point cloud) is divided into partitions that share the same local properties [1]. It enables numerous novel applications. Autonomous driving can leverage 3D semantic segmentation for separating vehicles from pedestrians [2]. By providing the crucial functionality of understanding the surrounding 3D environment, semantic segmentation is becoming an essential building block of AR [3]. For instance, a user can build a visual-based system to (1) "moves" objects by changing the location of a table in a room without actually moving it and visualize how the scene looks like without that table [4]; (2) "plays" with objects by shooting a virtual ball into the scene and watching how it bounces off different surfaces [5]; (3) "controls" objects in the camera view by turning off a lamp via making a gesture [6]; and (4) "merge" objects into the Virtual Reality (VR) so that a user can both view the virtual world and interact with physical objects (*i.e.,* augmented virtual reality) [7]. With the advance of machine learning, especially deep learning, DNN models achieve the best accuracy for 3D semantic segmentation, for example, in the ScanNet indoor segmentation benchmark [8] and the Semantic3D outdoor segmentation benchmark [9].

One issue of these DNN models for 3D semantic segmentation is that they are not only computation expensive but also memory intensive. The state-of-the-art SparseConvNet [10] takes on average 4.2 seconds and 2.8 GB memory for segmenting a point cloud of an indoor scene on a powerful laptop. The overwhelming inference time and memory usage of these DNN models hinder their deployment on devices with limited resources and for applications with strict latency requirements. In general, it is challenging to optimize a DNN model's inference time and memory usage while not degrading its accuracy [11]. Existing acceleration techniques such as parameter quantization [12] and network pruning [13] either require time-consuming retraining of models and/or result in a low accuracy. Hence, it is highly desirable that a pre-trained DNN model is *untouched* during the optimization, which does not necessitate laboring engineering of repeatedly tuning the same model.

Besides the above generic methods, the mobile computing community has proposed many specific technologies to accelerate DNN models for processing 2D images, such as offloading to the cloud [14], model selection [15], and hardware parallelism [16]. However, it is non-trivial to extend these methods for 3D objects, which brings several unique challenges. The input 3D object of semantic segmentation could be either a point cloud or a 3D mesh (more details in Section II). We focus on point cloud in this paper, mainly because of its simplicity and flexibility compared to 3D mesh.

*Work was done at AT&T Labs Research.

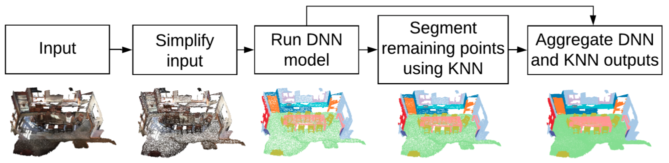

Input
Simplify input
Run DNN model
Segment remaining points using KNN
Aggregate DNN and KNN outputs

Fig. 1: System architecture of Slimmer: It first sparsifies the input and executes the pre-trained DNN model on this simplified input. Then, Slimmer segments the removed points using the KNN technique. Finally, Slimmer aggregates the DNN output (for the simplified input) and KNN output (for the removed points), and creates a complete segmentation for the original full-size input.

Differently from 2D images that have a regular shape (*i.e.,* an image has a fixed number of rows and columns for its pixels), point clouds usually have an irregular shape and a varied number of points in them, due to the non-uniform distribution of sampled points in a 3D space. The non-uniform point density of point clouds makes the estimation of inference time for these DNN models difficult, which is essential for model selection and hardware parallelism.

We propose Slimmer, a generic and model-independent framework, for accelerating 3D semantic segmentation. The design of Slimmer is motivated by our observation that the DNN models for 3D semantic segmentation remain high accuracy even if we remove a fraction of points from the input. However, this input simplification may significantly reduce inference time and memory usage, compared to the current practice that directly feeds a point cloud to the DNN models. Although this idea sounds straightforward, there are two key challenges for making Slimmer practical. First, we need to determine a lightweight simplification method to sparsify the point clouds. Otherwise, the extra computation time for input-data simplification may be higher than the reduced inference time, which cancels out the benefit of Slimmer. Second, we still need to accurately segment the removed points from the original full-size input to create a complete segmentation. The method for segmenting removed points should also be lightweight.

We illustrate the system architecture of Slimmer in Figure 1. Instead of directly feeding the input to the pre-trained DNN model, Slimmer sparsifies the point cloud and executes the DNN model on the simplified input, for reducing the inference time and memory usage. We identify several simple but effective methods for sparsifying point clouds such as the random simplification, the grid simplification [17], and the hierarchy simplification [18]. Afterward, Slimmer segments the removed points using a lightweight K-Nearest-Neighbors (KNN) based technique, which assigns segmentation labels to these points by considering the segmentation results of the DNN model for their neighboring points in the simplified point cloud. By aggregating the DNN output (for the simplified input) and the KNN output (for the removed points), Slimmer generates a complete segmentation for the original full-size point cloud.

In general, there is a tradeoff between resource utilization and model accuracy. The more points we remove from the input point cloud, the higher reduction we can achieve for inference time and memory usage, but at the same time the higher the accuracy loss of the 3D semantic segmentation models could be. Hence, to quantify the improvement of Quality of Experience (QoE), we propose a metric that explores the design space by jointly considering the segmentation accuracy, the inference time, and the memory usage. Using this metric, we extensively evaluate the performance of Slimmer using the SparseConvNet [10] model, and investigate the impact of various design factors, including the amount of remaining points after simplification, the simplification method, and the configuration of KNN.

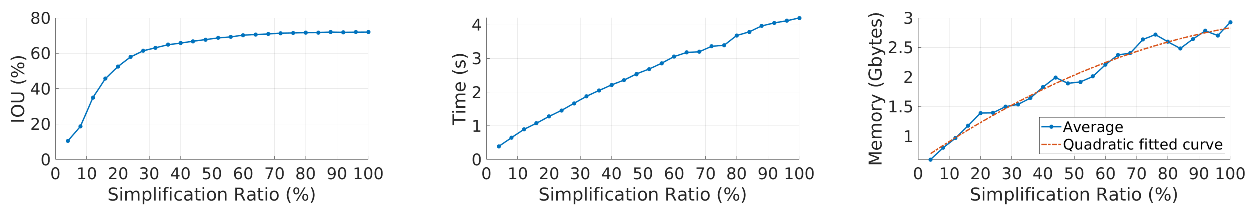
To the best of our knowledge, Slimmer is the first model-independent framework for accelerating 3D semantic segmentation based on data simplification. Compared with existing solutions, the key advantage of Slimmer is that it does not require any modifications to the pre-trained DNN model, and thus is generic and widely applicable. In summary, we make the following contributions.

1) We observe that input data simplification can reduce the inference time and memory usage of 3D semantic segmentation models, but with limited impact on segmentation accuracy.
2) Motivated by the above observation, we propose Slimmer that executes the DNN model on a simplified point cloud, and leverages a lightweight KNN technique to segment the removed points.
3) We implement a prototype of Slimmer and extensively evaluate its performance using a state-of-the-art DNN model for 3D semantic segmentation and an open dataset.

II. Related Work

In this section, we present related work on deep neural networks for 3D semantic segmentation, mobile augmented reality, and methods of accelerating DNN models for mobile systems.

Deep Neural Networks for 3D Semantic Segmentation. As a pioneer work of 3D classification and segmentation, PointNet [19] directly consumes points in a point cloud and then applies max pooling to reserve the permutation invariance of points. Recently, sparse convolutional layers [20], [21] are designed specifically for sparse data such as point clouds, which supplement conventional dense convolutional layers. To combat the active-site dilation problem in sparse convolutional layers, submanifold sparse deep neural networks are proposed [10]. In submanifold sparse neural networks, the active sites remain the same across the whole network, which in turn have the same number as the input layer (*i.e.,* the number of points). The most accurate DNN models for the ScanNet indoor segmentation are SparseConvNet [10] and Minkowski [22], both are based on submanifold sparse neural



(a) Accuracy (b) Inference Time (c) Memory Usage

Fig. 2: The performance of the pre-trained model over the sparsified point clouds with different simplification ratios (using random simplification). The performance at 100% simplification ratio represents the model performance without data simplification.

networks. In this paper, we focus on submanifold sparse neural networks due to their high efficiency and accuracy.

Mobile Augmented Reality. GLEAM [23] builds a photo-realistic AR system by estimating the illumination of the virtual objects so that the virtual objects can vividly blend into the physical world. On the contrary, Tian *et al.* [7] build a augmented virtual reality system in which the physical objects are merged into the virtual world. Schutt *et al.* [6] apply a 2D semantic segmentation DNN model to consecutive RGB images of surrounding environment, and then build a 3D scene for it. There are emerging works on leveraging 3D semantic segmentation for AR in the past few years, which offers deeper understanding of surrounding environment compared to solutions that use 2D images as input. For example, Ishikawa *et al.* [4] apply a 3D semantic segmentation DNN model to indoor point clouds for users to move objects in the scene. Immersive gaming is another application of mobile AR, in which a user can play games in the surrounding 3D environment [5]. In this paper, we aim to accelerate 3D semantic segmentation for mobile augmented reality.

Accelerating DNN Models for Mobile Systems. There are generic techniques to accelerate DNN models, such as parameter quantization [12] and network pruning [13]. Although they can greatly reduce the inference time and memory usage and thus are widely adopted by the industry, they often lead to much lower accuracy than the original model. Another popular technique is offloading computation-intensive tasks to a cloud server, in light of that a cloud server is usually equipped with much more powerful hardware than mobile devices. However, offloading is not always practical because of the large size of data transmission and privacy concerns. Smartphones are equipped with many computation resources such as CPU, GPU, and DSP. It can speed up model execution if these resources could be used in parallel. MobiSR [16] partitions an image into small patches, and runs these patches on CPU, GPU, and DSP in parallel. Differently from the above approaches, Slimmer is a model-independent solution to accelerate DNN models for 3D semantic segmentation, without requiring any modifications to the pre-trained models.

III. Motivation

In this section, we measure the performance of a representative DNN model for 3D semantic segmentation to motivate the design of Slimmer. More specifically, we present our observation that the model remains high accuracy even if we remove a fraction of points from the input, which can significantly reduce its inference time and memory usage.

We use the ScanNet dataset [8] that includes 1513 indoor point clouds. They capture various indoor scenes including living rooms, classrooms, and offices (more details in Section VI). Each point of a ScanNet point cloud is classified into 20 classes such as wall, floor, and desk. We employ the widely used metric Intersection Over Union (IOU) to quantify the segmentation accuracy. This metric is an average of each class's Critical Success Index (CSI), *i.e.,*

$$\text{IOU} = \frac{1}{N} \times \sum_{i=0}^{N-1} CSI(i) \tag{1}$$

where N is the number of classes. $CSI(i)$ is the critical success index of class i, which is defined as:

$$CSI(i) = \frac{TP(i)}{TP(i) + FP(i) + FN(i)} \tag{2}$$

where $TP(i)$, $FP(i)$, and $FN(i)$ stand for true positive, false positive, and false negative for class i, respectively. IOU has merits over point-wise accuracy because different classes have highly biased number of points. For example, a model that generalizes poorly can still achieve high point-wise accuracy by predicting all points belongs to floor for a point cloud in which most points represent the floor. By comparison, IOU equally treats each class by averaging their CSIs.

For the semantic segmentation benchmark of ScanNet, there are only two DNN models that achieve higher than 70% IOU and have released model details, SparseConvNet [10] and Minkowski [22]. In this paper, we focus on SparseConvNet, since the design of Minkowski is based on SparseConvNet. We first train a SparseConvNet model with the configuration of 32 feature maps, residual-style and optional blocks. We then conduct experiments on a Dell Alienware laptop (6-core 2.90 GHz i9 CPUs, 16 GB RAM). The pre-trained model achieves IOU of 71.18%, and takes on average 4.21 seconds and 2.83 GB memory for processing a point cloud. It is clear that the state-of-art DNN model is too costly for mobile devices.

During the measurement study, we observe that the inference time and memory usage of a submanifold sparse

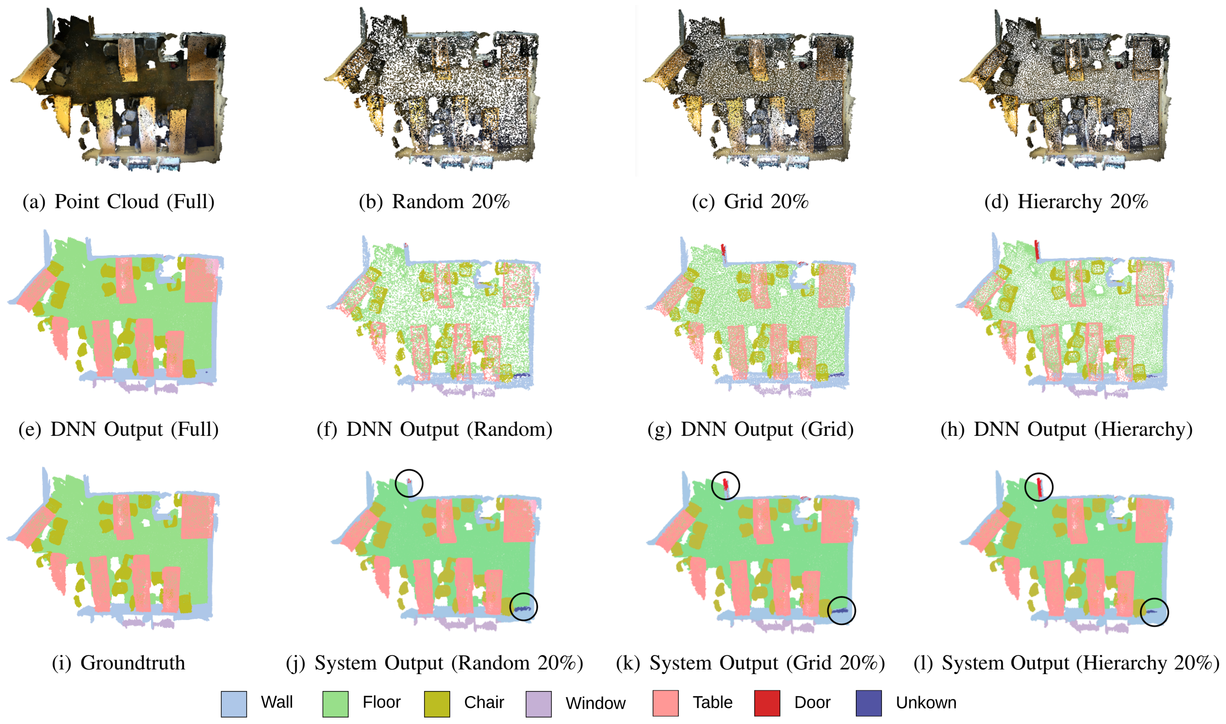


Fig. 3: Visualization of Slimmer output for a point cloud that is sparsified by different simplification techniques. (a) is a full-size point cloud, and (b) to (c) are a simplified point cloud that is 20% of its original size sparsified by the random, the grid, and the hierarchy simplifications. The second row shows the DNN output for the corresponding point clouds. (i) is the segmentation groundtruth, and (j) to (l) are the final output of Slimmer. We circle the regions that are falsely segmented in the outputs.

convolution based DNN model grow approximately linearly with the number of active sites of each layer, which in turn equals to the active sites in the input. Hence, if we simplify the input point cloud by reducing the number of points in it, we may decrease the inference time and memory usage of a pre-trained model when processing this sparsified point cloud. However, the input data simplification may affect the accuracy of semantic segmentation. We conduct the following experiments to understand this impact. We reduce the number of points in each point cloud by removing a fraction of randomly sampled points. We change the probability of keeping each point and thus generate simplified point clouds with different sizes. After that, we evaluate the performance of the pre-trained model using the simplified point clouds. In this paper, we define the simplification ratio as the ratio between the number of points in the resultant sparsified point cloud and the number of points in the original full-size point cloud.

Figure 2 shows the performance of this model with regards to the accuracy, inference time, and memory usage using randomly simplified point clouds with different sizes. We present the results averaged over five runs. The performance at 100% simplification ratio represents that of the model without data simplification (*i.e.,* the results of the pre-trained model using full-size point clouds). We have the following observations from the experimental results.

1) Model Accuracy. The IOU remains almost the same even when only circa 60% points are used. It slightly decreases to 60% IOU when point clouds are simplified to only 27% of the original size. The results indicate that real-world point clouds are highly redundant for the pre-trained 3D semantic segmentation model. The accuracy plummets when the size of a simplified point cloud is too small, (*e.g.,* smaller than 20% of the original size).
2) Inference Time. As we can see, the inference time is approximately linearly correlated with the simplification ratio. That is, the smaller the point cloud is, the shorter inference time the model has. A point cloud of 50% points takes about 60% processing time of the full-size point cloud. Therefore, it is preferred to sparsify point clouds as long as the accuracy loss is tolerable.
3) Memory Usage. The memory usage is also approximately linearly correlated with the simplification ratio. The smaller the point cloud is, the smaller the memory usage is. A point cloud of 50% points consumes about 70% memory of the full-size point cloud.

In summary, data simplification is effective for improving resource utilization. It can significantly reduce the inference time and memory usage and meanwhile maintain good accuracy. As long as the accuracy loss is tolerable, point clouds should be simplified as small as possible. However, since we simplify the input and thus the results do not necessarily represent the original requirement of segmenting a given full-size point cloud. Therefore, we also need to assign labels for the removed points in the original point cloud.

IV. Overview of Slimmer

In light of the observation that a pre-trained model can segment a simplified point cloud with reduced inference time and memory usage, we propose a framework, called Slimmer, for accelerating 3D semantic segmentation. Figure 1 illustrates the system architecture of Slimmer. Instead of directly running the DNN model on the point cloud, Slimmer sparsifies the point cloud, and executes the DNN model on the simplified point cloud. After that, Slimmer segments the removed points using a KNN technique. Finally, Slimmer aggregates the DNN output (for the simplified point cloud) and the KNN output (for the removed points) to finish the segmentation.

Figure 3 visualizes the output of Slimmer using a point cloud as an example. Figure 3(a) is a full-size point cloud, and Figure 3(b) to Figure 3(d) show the simplified point cloud that is sparsified to 20% of its original size by the random, the grid [17], and the hierarchy simplification [18], respectively. The second row visualizes the output of the pre-trained DNN model for the corresponding point clouds in the first row of the figure. It clearly demonstrates that the pre-trained model can still accurately segment simplified point clouds. Figure 3(i) is the segmentation groudtruth for the point cloud, and Figure 3(j) to Figure 3(l) visualize the Slimmer output that aggregates the DNN output (the second row) and the KNN output (the number of neighbors in KNN is set to 1). We use black circles to annotate the areas that are segmented incorrectly. The system output falsely segments the wall in the top as the door, and couldn't segment the bottom-right area. The system output is less accurate than running the DNN model on the full-size point cloud (*i.e.,*, Figure 3(e)) because we excessively simplify the input to only 20% of its original size for the illustration purpose. From Figure 3 we can see that (1) Slimmer can segment the full-size point cloud based on the simplified input of the point cloud; (2) Simplification methods affect the segmentation accuracy as we can see that the areas of the falsely segmented regions are different.

Compared to the current practice of feeding the full-size point cloud to the DNN model, Slimmer reduces the inference time and memory usage of the DNN model (because of smaller point cloud), but introduces extra overheads from the data simplification and the KNN. As we will show in the evaluation, the overall inference time and memory usage of Slimmer is significantly smaller. In general, there is a tradeoff between resource utilization and model accuracy. The more points we remove from the input point cloud, the higher reduction we can achieve for inference time and memory usage, but at the same time the higher the accuracy loss of the 3D semantic segmentation models could be. Hence, to quantify the improvement of Quality of Experience (QoE), we propose a metric that explores the design space by jointly considering the segmentation accuracy, the inference time, and the memory usage. Using this metric, we can investigate the impact of various design factors, including the simplification ratio, the simplification method, and the configuration of KNN.

V. System Design of Slimmer

There are two essential components of Slimmer: the method of simplifying the input point cloud, and the method of segmenting the removed points from the input. To quantify the improvement for each combination of the simplification method, the simplification ratio, and the segmentation of removed points, we first propose our system QoS. Afterward, we present several simplification methods that might be effective and lightweight. Last, we explain our KNN-based method of segmenting the removed points, which together with the DNN output creates a complete segmentation for the original full-size input.

A. QoE Improvement based on Simplification Ratio

We propose a QoE that jointly considers the inference time improvement, the memory usage improvement, and the accuracy loss. Specifically, our QoE is based on the simplification ratio, and is defined as

$$QoE(r) = \alpha \cdot T(r) + \beta \cdot M(r) - I(r) \tag{3}$$

where r is the simplification ratio. $T(r)$ and $M(r)$ is the inference time improvement and the memory usage improvement, and $I(r)$ is the accuracy loss. α and β are the weights for the time and memory, which determines the preference of the user requirement. For example, a large α means that the system is more concerned with inference time, while accuracy loss is more tolerable. $T(r)$, $M(r)$, and $I(r)$ are defined as

$$T(r) = 1 - \frac{T_S(r) + T_D(r) + T_R(r)}{T_D(100))} \tag{4}$$

$$M(r) = 1 - \frac{\max(M_S(r), M_D(r), M_R(r))}{M_D(100)} \tag{5}$$

$$I(r) = 1 - \frac{I_O(r)}{I_D(100)} \tag{6}$$

where $T_D(100)$, $M_D(100)$, $I_D(100)$ is the inference time, the memory usage, and segmentation accuracy of the DNN model executing on the full-size point cloud, $T_S(r)$ and $M_S(r)$ is the processing delay and memory usage of the data simplification, $T_D(r)$ and $M_D(r)$ is the inference time and memory usage of the DNN model on the simplified point cloud, $T_R(r)$ and $M_R(r)$ is the processing delay and memory usage of the KNN, and $I_O(r)$ is the overall segmentation accuracy.

Since the inference time improvement and the memory usage improvement are highly correlated (both are approximately linear with the simplification ratio), we can simplify Equation (3) to

$$QoE(r) = \lambda \cdot T(r) - I(r) \tag{7}$$

where λ is the new weight, which determines the importance of the inference time improvement over the accuracy loss. For example, if λ equals 1, then the QoE equally treats the inference time improvement and the accuracy loss. In practice, it is usually preferred to not degrading the system accuracy too much, while reducing the inference time and memory usage as much as possible. Therefore, a reasonable range of λ is [0, 1.0].

Our system QoE (Equation (7)) is a convex function, when λ is in the reasonable range. The reasoning is as follows. When we decrease the simplification ratio r from 100, $T(r)$ increases while the accuracy $I(r)$ remains the same, and thus the QoE is increased. If we reduce the simplification ratio r too much, then the accuracy loss $I(r)$ would surpass the weighted inference time improvement $T(r)$, and thus QoE is reduced. Therefore, for each combination of the simplification method and the KNN configuration, there is an simplification ratio r that gives the highest QoE. Based on the QoE value of each combination of the simplification method, simplification ratio and the KNN configuration, we can identify the best combination that has the highest QoE than other combinations.

B. Simplification Methods

Different simplification methods generate sparsified point clouds with different characteristics. For example, the simplified point clouds from the hierarchy simplification look much sharper than those from the grid simplification (as illustrate in Figure 3). The data simplification method should be lightweight, otherwise it would cancel out the inference time reduction of executing the DNN model on the simplified input, which impairs the effectiveness of our data-simplification based acceleration. We identify the following three simple but effective methods for sparsifying point clouds.

1) *Random Simplification*. Each point is independently kept with a given probability. Therefore, random simplification regards each point equally.
2) *Grid Simplification*. Each point cloud is partitioned into grid cells of a given size [17]. For each non-empty grid cell, a point is randomly selected among the points in that cell. Therefore, grid simplification favors sparse points than dense points. This is because only one point is kept no matter whether the grid cell is sparse (a few points) or dense (lots of points). We change the grid size to generate sparsified point clouds of different sizes.
3) *Hierarchy Simplification*. It provides an adaptive simplification of a point cloud through local clusters [18], which recursively splits the point set into smaller clusters until the clusters have less than a given size. Hierarchy simplification favors edge points than surface points and thus generates sharp and vivid point clouds. We change the target cluster size to generate sparsified point clouds of different sizes.

C. Segmenting Removed Points by KNN

The data simplification procedure splits points into two sets: points in the sparsified point cloud, and the removed points that are filtered out. Since we know the labels of the simplified point cloud from the DNN output, we only need to segment the removed points in order to generate a full-size segmentation. As with the data simplification method, the method of segmenting the removed points must be lightweight. Otherwise, it would cancel out the overhead reduction of running the DNN model on the simplified input. We propose to infer the label of a removed point by the majority label of its nearest neighbors that are in the simplified point cloud. For the sake of convenience, we denote the number of points of a point cloud by n_t, the number of points of the simplified point cloud n_s, and the number of removed points n_r. Obviously, $n_s + n_r = n_t$. We use small k for the degree of dimension in a k-d tree, and big K for the number of neighbors in KNN. Below are our algorithm of segmenting the removed points.

1) We construct a k-d tree (k = 3 as points are in 3D domain) for the simplified point cloud. Each point in the k-d tree has a segmentation label because the DNN model segments the simplified point cloud. The k-d tree construction is fast, with a worst-case complexity of $\mathcal{O}(n_s \cdot \log n_s)$.
2) For each removed point, we search its K nearest points in the k-d tree. The neighbor searching for each removed point is also fast, with complexity of $\mathcal{O}(\log n_s)$. Thus, the overall complexity of searching neighbors for all removed points is $\mathcal{O}(n_r \cdot \log n_s)$.
3) After we identify the K neighbors for each removed point, we assign the the majority label of its neighbors to the removed point. In the case that there is no majority label (*e.g.,* 2 neighbors are labeled as wall and 2 neighbors are labeled as floor when K equals 5), we randomly assign the removed point with one label of the largest number of neighbors. Each labeling for a removed point costs the complexity of $\mathcal{O}(1)$, and thus the overall labeling procedure has the complexity of $\mathcal{O}(n_r)$.

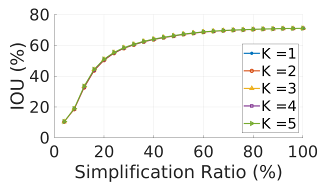
The overall complexity of segmenting the removed points by our KNN based method is $\mathcal{O}(n_s \cdot \log n_s) + \mathcal{O}(n_r \cdot \log n_s) + \mathcal{O}(n_r) \leq \mathcal{O}(n_t \cdot \log n_t)$. The complexity indicates that our KNN-based segmentation of the removed points is lightweight. It is robust against the simplification ratio (*i.e.,* n_s/n_t), and scales well with the number of points of the original full-size point cloud (*i.e.,* with regards to n_t).

There is one configuration in our KNN-based segmentation for removed points, that is, the number of neighbors K. It is straightforward that a larger K results in more processing delay. However, it is unclear how the accuracy changes versus the K.

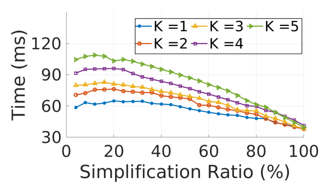
VI. Evaluation

In this section, we first introduce our experiment setup. After that, we evaluate the performance of the KNN. Then, we measure the performance of the simplification methods. Afterward, we apply the QoE metric to explore the design space. In the end, we present detailed overall system performance.

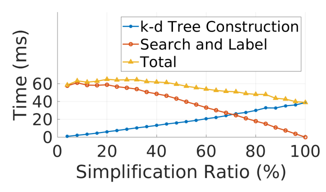
Setup: We conduct experiments on a Dell Alienware laptop that has 6-core 2.9 GHz i9 CPUs and 16 GB RAM. We use the ScanNet dataset to train a SparseConvNet DNN model, and use the validation dataset of it to evaluate our system.



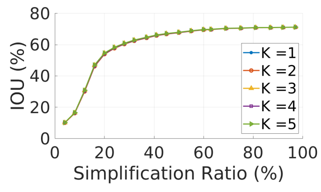
(a) Random IOU



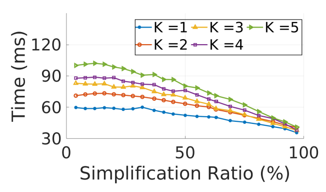
(b) Random Processing Time



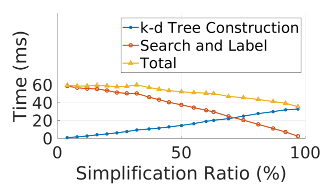
(c) Random Breakdown of K=1



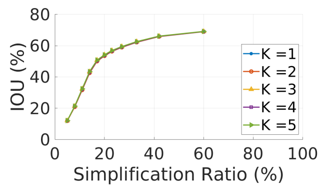
(d) Grid IOU



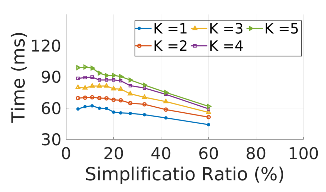
(e) Grid Processing Time



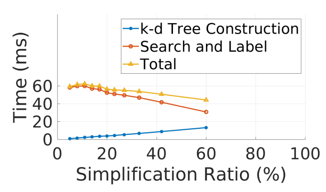
(f) Grid Breakdown of K=1



(g) Hierarchy IOU



(h) Hierarchy Processing Time



(i) Hierarchy Breakdown of K=1

Fig. 4: Study of different number K on performance of the random, the grid, and the hierarchy versus the simplification ratio. First column shows that different K do not affect the system accuracy. Second column shows that K = 1 has the least processing delay. Considering the accuracy and processing delay, we adopt K=1 in the rest of experiments. The breakdown of processing time of K=1 is plotted in the third column. The hierarchy simplification can only generate point clouds up to 60% of the original size.

As suggested by the ScanNet, we split the dataset into 1201 point clouds for training and 312 point clouds for validation. the point clouds of the ScanNet dataset are diverse: (a) The number of points in each point cloud varies from 32.8K to 438.6K, with ratio of 13 times; (b) he space sizes of point clouds vary from a small cubicle of 6.4 m^3 to a large office of 347.3 m^3, with ratio of 54 times; and (c) The point density of point clouds varies from 390.5 points/m^3 to 6851.6 points/m^3, with ratio of 17 time.

A. Segmenting Removed Points using KNN

There is one parameter in KNN segmentation for removed points, that is, the number of neighbors K. We conduct experiments to determine the optimal K value that achieves high system accuracy and low KNN processing delay.

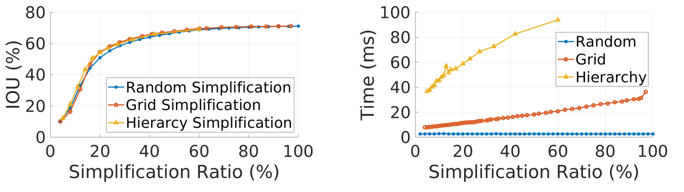
Figure 4 shows the KNN performance. The first column shows the system segmentation accuracy (*i.e.,* aggregated outputs from the DNN and the KNN) when the random, the grid, and the hierarchy simplification is applied. Unlike the random and the grid, the hierarchy simplification can only generate simplified point clouds of up to 60% of its original size. We can see that whatever simplification method is used, different K values result in almost same system segmentation accuracy. The second column shows the processing delay of the KNN. It clearly shows that the KNN with K=1 results in least processing delays, for the random, the grid, and the hierarchy simplification. Considering that higher K does not bring higher accuracy, but causes greater delay, we adopt K=1 in the rest of evaluation.

We dissect the processing delay of KNN with K=1 into two parts: the k-d tree construction time, and the searching and labeling time. We plot the results in the third column of Figure 4. We can see that whatever the simplification method is used, the k-d tree construction takes longer time for bigger simplified point cloud, and shorter time for the searching and labeling. Overall, it takes about 35 ms to 60 ms for the KNN to segment the removed points. Since that KNN operations consume less than 50 MB memory, and thus it does not cause extra memory usage (which is the maximum memory usage during the whole system execution).

B. Simplifying Point Clouds using the Random, the Grid, and the Hierarchy

Different data simplification method generates point clouds of various characteristics, and thus they affect the segmentation accuracy. In addition, each simplification method takes different amount of time to sparsify a point cloud.

Figure 5(a) shows the system segmentation accuracy of Slimmer when the point clouds are sparsified by the random, the grid, and the hierarchy. We have the following observations. (1) The accuracy loss dramatically increases when the point clouds are sparsified to be smaller than 20% of the full size. (2) The grid and the hierarchy have smaller accuracy loss than the random when the simplified point cloud are smaller



(a) System Segmentation Accuracy (b) Processing Delay

Fig. 5: Study of the random, the grid, and the hierarchy simplification versus the simplification ratio. (a) shows the system segmentation accuracy and (b) shows the processing delay, for each simplification method.

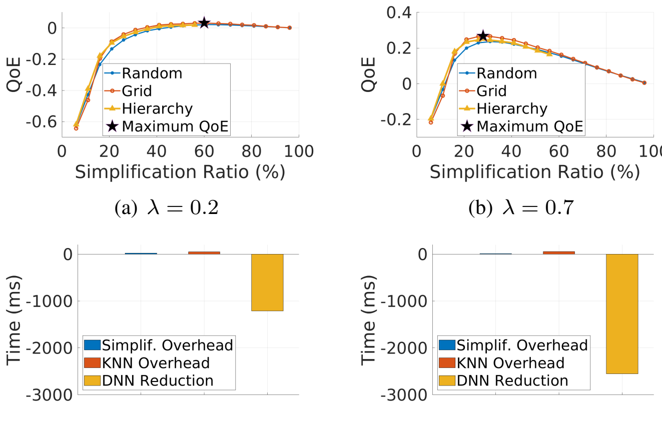
than 60% of its original size. For example, when the point cloud is simplified to 20% of its original size, the grid and the hierarchy is more than 3% IOU higher than the random. (3) When the size of the simplified point cloud is larger than 20% of the original one, the grid simplification is best. This is counter-intuitive since the simplified point cloud from the hierarchy is sharper than that from the grid. It indicates that a pre-trained DNN model "sees" differently from our human eyes. (4) When the size of the simplified point cloud is larger than 60%, the grid and the random have similar segmentation accuracy.

Figure 5(b) shows the data processing time for the random, the grid, and the hierarchy simplification. We have the following observations. (1) The random simplification takes the same time to generate a point cloud of different sizes. This is because that each point is being independently kept. The random simplification is fastest among all the simplification methods. It takes about 2.65 ms to generate a simplified point cloud. (2) The grid simplification takes a linearly-increased time to generate bigger point clouds. It takes 7.93 ms for 4% point clouds and increases to 31.40 ms for 95% point clouds. (3) The hierarchy simplification takes significantly longer time than the random and the grid simplification. It requires 36.75 ms for generating 5% point clouds and increases to 93.93 ms for 60% point clouds. Since that data simplifications consume much less memory than running the DNN model ($\sim$10 MB for the random, $\sim$30 MB for the grid, and $\sim$20 MB for the hierarchy), the data simplification does not cause extra memory usage.

C. Applying QoE to Compare Different Combinations of the Simplification Method and Ratio

We propose a system QoE (Equation (7)) to investigate the impact of the simplification method, the simplification ratio, and the KNN configuration. Since K = 1 is optimal for KNN, in this experiment, we explore the QoE for the simplification method and the simplification ratio.

The first row in Figure 6 shows the QoE curve of the random, the grid, and the hierarchy simplifications versus the simplification ratio when λ is 0.2 and 0.7, respectively. We have the following observations. (1) The QoE curves are convex, meaning that there is a simplification ratio that achieves



(a) $\lambda = 0.2$ (b) $\lambda = 0.7$

(c) $\lambda = 0.2$ (d) $\lambda = 0.7$

Fig. 6: Leveraging our QoE to investigate various design factors such as the simplification method and the simplification ratio. The first row shows the QoE curve for each simplification method versus the simplification ratio when the weight λ equals to 0.2 and 0.7 respectively. We use a star sign to indicate the maximum QoE. The second row shows the corresponding extra delays because of the data simplification and KNN, and the reduction of executing the DNN model on the simplified point cloud than on full-size point clouds, for the case of the maximum QoE.

the maximum QoE. (2) Different simplification methods have different QoE curves for the same λ. We can compare their performance based on their maximum QoE values. (3) We use a black star sign to annotate the point of the maximum QoE for all the simplification methods for each given λ. The optimal simplification ratio is smaller for larger weight λ. This is because large λ results in more aggressive scheme to sparsify the point clouds. The second row in Figure 6 plots the extra processing time for the data simplification and the KNN, and the reduction of DNN time, for the combination of the simplification method and ratio that achieves the highest QoE for each λ. We have the following observations. (1) The extra processing time of data simplification and KNN is significantly smaller than the reduction time of running the DNN model on simplified point cloud than on full-size point cloud, signifying the effectiveness of Slimmer in accelerating the 3D semantic segmentation. (2) As expected, larger λ reduces more DNN running time because of smaller simplification ratio that achieves the maximum QoE.

D. Overall System Performance

We evaluate the overall performance of Slimmer versus weight λ for the random, the grid, and the hierarchy simplification. Different λ provide trade-offs between the inference time improvement, memory usage improvement, and the accuracy loss.

Table I tabulates the details of the overall performance of Slimmer. We show for each simplification method the QoE value, the system accuracy, the accuracy loss, the system

Weight λ	0.00	0.05	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00
Random QoE	0.000	0.000	0.002	0.022	0.051	0.087	0.133	0.183	0.236	0.294	0.353	0.414
IOU (%)	71.18	71.18	70.48	68.83	67.40	65.32	62.70	62.70	60.78	58.54	58.54	55.41
Accuracy Loss (%)	0.00	0.00	0.98	3.30	5.31	8.23	11.91	11.91	14.61	17.76	17.76	22.16
Time (s)	4.21	4.21	3.72	3.05	2.75	2.43	2.09	2.09	1.91	1.73	1.73	1.54
Time Improvement (%)	0.00	0.00	11.69	27.47	34.61	42.21	50.37	50.37	54.60	58.99	58.99	63.52
Memory (GB)	2.83	2.83	2.59	2.25	2.08	1.89	1.69	1.69	1.58	1.47	1.47	1.35
Memory Improvement (%)	0.00	0.00	8.48	20.73	26.69	33.26	40.42	40.42	44.23	48.20	48.20	52.31
Simplification Ratio (%)	100	100	80	60	52	44	36	36	32	28	28	24
Grid QoE	**0.001**	**0.002**	**0.009**	**0.032**	**0.065**	**0.109**	**0.156**	**0.210**	**0.267**	**0.326**	**0.389**	**0.453**
IOU (%)	71.22	70.93	70.42	69.62	67.03	66.13	62.90	62.90	60.85	60.85	58.24	58.24
Accuracy Loss (%)	-0.06	0.35	1.07	2.19	5.83	7.09	11.63	11.63	14.51	14.51	18.18	18.18
Time (s)	4.20	3.77	3.38	3.07	2.48	2.31	1.92	1.92	1.73	1.73	1.54	1.54
Time Improvement (%)	0.15	10.49	19.58	27.09	41.05	45.03	54.42	54.42	58.88	58.88	63.44	63.44
Memory (GB)	2.90	2.75	2.57	2.40	2.05	1.94	1.67	1.67	1.54	1.54	1.41	1.41
Memory Improvement (%)	-2.31	2.94	9.19	15.19	27.68	31.53	41.01	41.01	45.59	45.59	50.38	50.38
Simplification Ratio (%)	97	81	69	60	45	41	32	32	28	28	24	24
Hierarchy QoE	-0.030	-0.017	-0.004	0.022	0.056	0.098	0.141	0.192	0.246	0.307	0.370	0.434
IOU (%)	69.08	69.08	69.08	69.08	66.07	66.07	66.07	62.59	59.43	56.91	56.91	56.91
Accuracy Loss (%)	2.95	2.95	2.95	2.95	7.18	7.18	7.18	12.07	16.51	20.05	20.05	20.05
Time (s)	3.13	3.13	3.13	3.13	2.42	2.42	2.42	2.02	1.74	1.54	1.54	1.54
Time Improvement (%)	25.54	25.54	25.54	25.54	42.56	42.56	42.56	52.06	58.78	63.42	63.42	63.42
Memory (GB)	2.25	2.25	2.25	2.25	1.84	1.84	1.84	1.61	1.44	1.32	1.32	1.32
Memory Improvement (%)	20.73	20.73	20.73	20.73	34.99	34.99	34.99	43.27	49.21	53.36	53.36	53.36
Simplification Ratio (%)	60	60	60	60	42	42	42	33	27	23	23	23

TABLE I: Details of the system performance of the random, the grid, and the hierarchy simplification versus the weight λ. We highlight the maximum QoE among all the simplification methods for each λ.

inference time, the inference time improvement, the memory usage, the memory usage improvement, and the optimal simplification ratio. We change the weight λ from 0 to 1, where 0 means that we only care about the accuracy and 1 means that we equally treats the inference time improvement and the accuracy loss. From the table, we can have the following observations.(1) The simplification ratio decreases when the weight λ increases. For example, the simplification ratio is reduced to 24% for the random and the grid, when the weight λ equal to 1. With increased weight λ, time improvement and the memory improvement increases, while the accuracy loss is worsen; (2) We highlight the simplification method that achieves the best QoE than the others. The result shows that the grid simplification consistently has better QoE than the random and the hierarchy. This is because the grid simplification has high accuracy and the processing delay is not overwhelming compared to the random and the hierarchy; (3) Slimmer is an effective framework for accelerating 3D semantic segmentation. For example, Slimmer reduces 19.58% inference time, and 9.19 memory usage, with only 1.07% accuracy loss, using the grid simplification with simplification ratio of 69%, or reduces 29.09% execution time, and 15.19% memory usage overhead, with only 2.19% accuracy loss, using the grid simplification with simplification ratio of 60%. It is straightforward for Slimmer to provide different trade-offs between the inference time improvement, the memory usage improvement, and the accuracy loss, by adjusting the weight.

VII. Discussion and Future Work

In this section, we discuss some limitations of our work, and present future works.

We evaluate Slimmer using the state-of-art 3D semantic segmentation DNN model and a large-scale indoor point cloud dataset. Our experimental results show that Slimmer is an effective framework for accelerating 3D semantic segmentation. The ScanNet includes more than one thousand point clouds, and thus it is very diverse and representative. Nonetheless, we plan to evaluate our framework with different datasets, such as the Semantic3D outdoor dataset [9]. In addition, we plan to apply our framework for more models, such as the Minkowski [22], and explore more data simplification methods in addition to the random, the grid, and the hierarchy.

To evaluate the QoE of different combinations of the simplification method, the simplification ratio, and the KNN configuration, we need to pre-define the weight λ, and thus identifying the best combination is not fully automatic. However, we argue that different application scenarios require different trade-offs between the accuracy and running overheads, and thus there is no "best" combination once-and-for-all. Instead, Slimmer identifies the best combination for each given weight. In addition, it is straightforward to adjust the weight for system requirements: larger weight results in more inference time and memory usage reductions, while the accuracy tends to be worse.

Last but not least, we plan to develop an AR application that is built on top of Slimmer as the acceleration engine. The AR application is preferred to (1) run in at least a few dozen frames per second for continuous vision [15], [24], [25], which is challenging as current 3D semantic segmentation takes more than one second per point cloud; and (2) support collaborative and persistent AR experience [26]–[28]. Collaborative and persistent AR is important because it is more likely to lead to both enjoyable user experience and more frequent usage [27]. Therefore, more research effort is required and more research opportunities exist for accelerating 3D semantic segmentation systems.

VIII. Conclusion

Slimmer is a generic and model-independent framework to accelerate 3D semantic segmentation for mobile augmented reality. By leveraging input data simplification, it can significantly reduce the inference time and memory usage, while remaining high accuracy for state-of-the-art DNN models of semantic segmentation. The key advantage of Slimmer over the existing solutions is that it does not require any modifications to pre-trained DNN models. In order to explore the design space, we propose a QoE metric to quantitatively compare different combinations of the simplification method, the simplification ratio, and the configuration of KNN for segmenting the removed points. By adjusting the weight λ, Slimmer provides various tradeoffs between the inference time improvement, the memory usage improvement, and the accuracy loss.

References

[1] L. E. Carvalho, A. C. Sobieranski, and A. von Wangenheim. 3D Segmentation Algorithms for Computerized Tomographic Imaging: a Systematic Literature Review. *Journal of Digital Imaging*, 31(6):799–850, 2018.

[2] Fawad Ahmad, Hang Qiu, Ray Eells, Fan Bai, and Ramesh Govindan. CarMap: Fast 3D Feature Map Updates for Automobiles. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[3] Mario Lorenz, Marc Busch, Loukas Rentzos, Manfred Tscheligi, Philipp Klimant, and Peter Frohlich. I'm There! The Influence of Virtual Reality and Mixed Reality Environments Combined with Two Different Navigation Methods on Presence. In *IEEE Virtual Reality (VR)*, 2015.

[4] Yuki Ishikawa, Ryo Hachiuma, Naoto Ienaga, Wakaba Kuno, Yuta Sugiura, and Hideo Saito. Semantic Segmentation of 3D Point Cloud to Virtually Manipulate Real Living Space. In *Asia Pacific Workshop on Mixed and Augmented Reality (APMAR)*, 2019.

[5] Lei Han, Tian Zheng, Yinheng Zhu, Lan Xu, and Lu Fang. Live Semantic 3D Perception for Immersive Augmented Reality. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 26(5):2011–2022, 2020.

[6] Peer Schutt, Max Schwarz, and Sven Behnke. Semantic Interaction in Augmented Reality Environments for Microsoft HoloLens. In *European Conference on Mobile Robots (ECMR)*, 2019.

[7] Yang Tian, Chi-Wing Fu, Shengdong Zhao Ruihui Li, Xiao Tang, Xiaowei Hu, and Pheng-Ann Heng. Enhancing augmented vr interaction via egocentric scene analysis. *Proceedings of the ACM on Interactive Mobile, Wearable and Ubiquitous Ubiquitous Technologies (IMWUT)*, 3(3):105:1–105:24, 2019.

[8] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Niebner. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[9] Timo Hackel, N. Savinov, L. Ladicky, Jan D. Wegner, K. Schindler, and M. Pollefeys. SEMANTIC3D.NET: A new large-scale point cloud classification benchmark. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2017.

[10] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[11] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A Survey of Model Compression and Acceleration for Deep Neural Netoworks. *arXiv e-prints*, page arXiv:1710.09282, 2019.

[12] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized Convolutional Neural Networks for Mobile Devices. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[13] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Prunning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR)*, 2016.

[14] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge Assisted Real-time Object Detection for Mobile Augmented Reality. In *Proceedings of ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2019.

[15] Biyi Fang, Xiao Zeng, and Mi Zhang. NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision. In *Proceedings of ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2018.

[16] Royson Lee, Stylianos I. Venieris, Lukasz Dudziak, Sourav Bhattacharya, and Nicholas D. Lane. MobiSR: Efficient On-Device Super-Resolution through Heterogeneous Mobile Processors. In *Proceddings of the ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2019.

[17] Jinhui Lan, Jian Li, Jiehui Li, Liujiang Zheng, Guangda Hu, and Guanglin He. Data Reduction based on Dynamic-threshold Uniform Grid-algorithm. *Optik*, 124(23):6461–6468, 2013.

[18] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient Simplification of Point-sampled Surfaces. In *IEEE Visualization (VIS)*, 2002.

[19] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[20] Ben Graham. Sparse 3D Convolutional Neural Networks. In *British Machine Vision Conference*, 2015.

[21] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[22] Christopher Choy, Jun Young Gwak, and Silvio Savarese. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[23] Siddhant Prakash, Alireza Bahremand, Linda D. Nguyen, and Robert LiKamWa. GLEAM: An Illumination Estimation Framework for Real-time Photorealistic Augmented Reality on Mobile Devices. In *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2019.

[24] Robert LiKamWa and Lin Zhong. Starfish: Efficient Concurrency Support for Computer Vision Applications. In *Proceedings of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2015.

[25] Robert LikamWa, Bodhi Priyantha, Matthai Philipose, Lin Zhong, and Paramvir Bahl. Energy Characterization and Optimization of Image Sensing Toward Continuous Mobile Vision. In *Proceedings of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2013.

[26] Wenxiao Zhang, Bo Han, Pan Hui, Vijay Gopalakrishnan, Eric Zavesky, and Feng Qian. CARS: Collaborative Augmented Reality for Socialization. In *Proceedings of ACM International Workshop on Mobile Computing Systems & Applications (HotMobile)*, 2018.

[27] Anhong Guo, Ilter Canberk, Hannan Murphy, Andres Monroy-Hernandez, and Rajan Vaish. Blocks: Collaborative and persistent augmented reality experiences. *Proceedings of the ACM on Interactive Mobile, Wearable and Ubiquitous Ubiquitous Technologies (IMWUT)*, 3(3):83:1–83:24, 2019.

[28] Ana Villanueva, Zhengzhe Zhu, Ziyi Liu, Kylie Peppler, and Thomas Redick Karthik Ramani. Meta-AR-App: An Authoring Platform for Collaborative Augmented Reality in STEM Classrooms. In *Proceedings of ACM International Conference on Human Factors in Computing Systems (CHI)*, 2020.