Towards Effective and Efficient 3D Visual Perception

by

Ziyang Hong

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfilment of the Requirements for
the Degree of Doctor of Philosophy
in the Department of Electronic and Computer Engineering

March 2024, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature redacted

Ziyang Hong

27 March 2024

Towards Effective and Efficient 3D Visual Perception

by

Ziyang Hong

This is to certify that I have examined the above PhD thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.

Signature redacted

Prof. Chik Patrick YUE, Thesis Supervisor

Signature redacted

Prof. Andrew Wing On POON, Head of Department

Thesis Examination Committee

1. Prof. Chik Patrick YUE (Supervisor)	Department of Electronic and Computer Engineering
2. Prof. Qifeng CHEN	Department of Electronic and Computer Engineering
3. Prof. Xiaomeng LI	Department of Electronic and Computer Engineering
4. Prof. Albert Chi Shing CHUNG	Department of Computer Science and Engineering
5. Prof. Hao YU (External Examiner)	School of Microelectronics Southern University of Science and Technology

Department of Electronic and Computer Engineering

27 March 2024

For my parents

who have been giving me unconditional love

Table of Contents

List of Figures

List of Tables

Towards Effective and Efficient 3D Visual Perception

by

Ziyang Hong

Department of Electronic and Computer Engineering

The Hong Kong University of Science and Technology

Abstract

We are witnessing developments in artificial intelligence (AI), where the most relevant parts to us are embodied AI systems with robotic agents interacting with human users in real-world environments. To serve in the embodied AI system, the agent robots need visual intelligence, namely the three-dimensional (3D) perception of the surrounding environment. This thesis is dedicated to enabling real-time low-cost edge deployment of visual perception which is advocated by us as the future solution for embodied AI applications.

First, an end-to-end deep neural network pipeline for machinery visual perception, CDRNet, is proposed. It jointly perceives a 3D scene's geometry structure and semantic labels. While conventional volumetric approaches for 3D perception tend to focus on the global coherence of their reconstructions, which leads to a lack of local geometric detail, CDRNet leverages the latent geometric prior knowledge in 2D image features by explicit depth prediction and anchored feature generation, to refine the occupancy learning in TSDF volume.

Besides, we find that this cross-dimensional feature refinement methodology can also be adopted for the semantic segmentation task by utilizing semantic priors, to extract both 3D mesh and 3D semantic labeling in real-time. Beyond public datasets for testing, we further implement a real-time messaging system to support these aforementioned perception tasks in real-life scenarios.

Finally, a software-hardware co-optimization system, Efficient-Grad is proposed to enable the online AI model fine-tuning. It improves both throughput and energy saving with negligible accuracy degradation during model training for deep convolutional neural networks, by utilizing sparsity and asymmetry residing in the gradients for conventional back propagation. Furthermore, the dedicated hardware architecture for sparsity utilization and efficient data movement is optimized to support the Efficient-Grad algorithm in a scalable manner, which leads to its superiority in terms of energy efficiency.

Acknowledgements

This wonderful doctoral pursuing journey is coming to an end. There are many people I would like to thank from the bottom of my heart.

I would like to express my utmost gratitude to my supervisor, Prof. Chik Patrick Yue for his encouragement, patience, valuable support, and guidance throughout my research. Still recall since we very first met on the basketball court here on the HKUST campus, I have been impressed by his infinite enthusiasm, curiosity, and broad knowledge across multiple disciplines. That day, we turned out not to play hoops at all but to discuss research while standing on the court. His constant encouragement and leadership along the way enlightened me to conduct all the research here at HKUST.

I would like to thank Prof. Qifeng Chen, Prof. Xiaomeng Li, Prof. Albert Chi Shing Chung, and Prof. Hao Yu for serving as my thesis examination committee members and providing precious comments and suggestions to my research. I took their postgraduate courses before and learned valuable knowledge about computer vision and digital circuit design.

I would like to thank Prof. Wenxiao Fang at SYSU and Dr. Yunfei En at CEPREI Lab. They led me on the trip to do electronic and computer engineering research when I just hit the road. I would never set foot in this field without their help.

I would like to thank the ICDC fellow alumni, particularly Dr. Bo Zhang, Dr. Vera Xizi Chen, Dr. Qian Yu, Dr. Jingyang Zhu, Dr. Summer Zechun Liu, Mr. Charles Jingbo Jiang, Mr. Dennis Jingyun Liu, and Dr. Dong Zhang for their kind guidance and discussion in my research.

I would like to thank my colleagues in ICDC and the OWL lab, including Prof. Xianbo Li, Dr. Li Wang, Dr. Babar Hussain, Dr. Milad Kalantari, Mr. Sam Can Wang, Mr.

Jian Kang, Dr. Yiru Wang, Dr. Bo Xu, Dr. Rehan Azmat, Mr. Chongyun Zhang, Mr. Funzhan Chen, Ms. Elsie Zilu Liu, Ms. Xinyi Liu, Mr. Hamed Fallah, Mr. Johar Abdekhoda, Mr. Johnny Hoi Chuen Cheng, Mr. Shaokang Zhao, Mr. Tony Zhengdong Li, Mr. Matthew Ruitao Ma, Ms. Sarah Shuo Feng, and Mr. Jeffry Wicaksana for their numerous discussions and joyful time together.

Finally, I would like to show love to my childhood pals, my girlfriend, Zichen, and my family, especially my parents: my Dad, Jianping, and my Mom, Ruixiang. Again, this dissertation is dedicated to my parents, who have been giving me unconditional love. It is they who inspire me to be who I am today.

1 Introduction

"Great research is often motivated by great needs."

— Charles R. Qi, *Deep Learning on Point Clouds for 3D Scene Understanding*

1.1 Background and Motivation

Artificial intelligence (AI) will soon come true thanks to the drastic evolvement of the underlying algorithms and supporting computing hardwares at an unprecedented pace throughout the last two decades. The advent of AI will have a profound impact on mankind amid its surging demand. Based on the market size, the major sub-fields of AI can be roughly categorized as natural language processing, computer vision, and data science. Among which, the top two fields of AI are just like the sensations of human beings, which are the key intelligence that is essential for the subsequent complex intelligent tasks. Enabling machine to mimic the perceptual and cognitive functions of human mind is intriging but still far from being realized.

Computer vision is regarded as the most significant research problem for the AI community. The recognition may partially coming from the fact that vision plays the most significant role in the history of evolution. There are archaeology and zoology researches proving that, the Cambrian explosion is triggered by sudden evolution of the vision sense, which means that vision dominates the evolution of animals [1]. Furthermore, going back to human, approximately 50% of human's neocortex is involved in visual processing [2].

As human beings live in the three-dimensional (3D) world, 3D computer vision sheds light on most practical and daily-life problems. Currently, the demand for 3D vision has become the largest among all AI tasks, the global market reached USD 22.28 billion

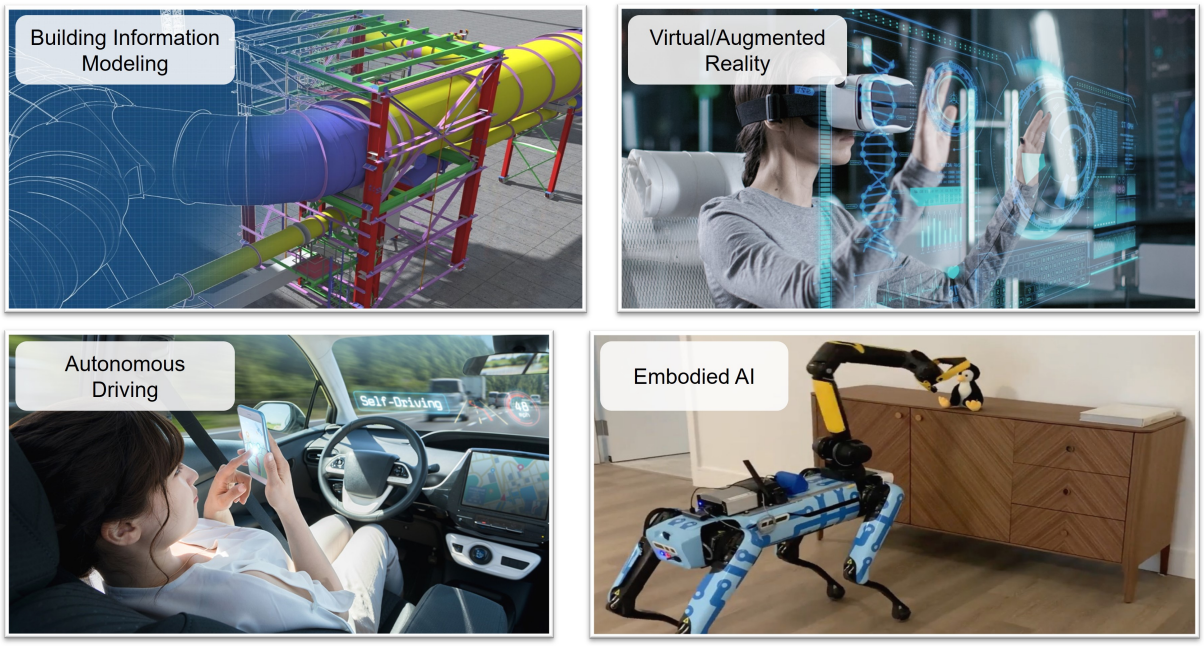


Figure 1.1: **Overview of the 3D perception applications.**

in 2023 and will reach USD 50.98 billion in 2030 with a high compound annual growth rate as 12.6% [3]. The roaring market growth is mainly due to its various applications in many mainstay industries, such as building information modeling, virtual/augmented reality, autonomous driving, and embodied AI robotics, as shown in Fig. 1.1. Obviously, in addition to the great needs in terms of quantity, these applications reflect the impact of 3D vision is rather huge, or "great" — by following the pun intended in the epigraph of this chapter. This is true especially for the embodied AI robotics because they not only increase the productivity for industries, also substitute the labor work just like humanoids. Therefore, embodied AI robotics is of great importance and has appealed significant research attention nowadays.

1.2 Prior Arts on 3D Visual Perception System

A typical embodied AI system is enabled by two major techniques, navigation for the agent and the manipulation for the operator. It is noteworthy that both techniques require the **3D perception** and physical control to ambient environment.

Fig. 1.2 illustrates a representative embodied AI platform whose verbal instructions are shown at the bottom of each row. Each instruction can be dissected into four steps which are shown in each column. In each row (namely for each task), the agent search for an unseen object at an assigned location and move it to the target receptacle. It

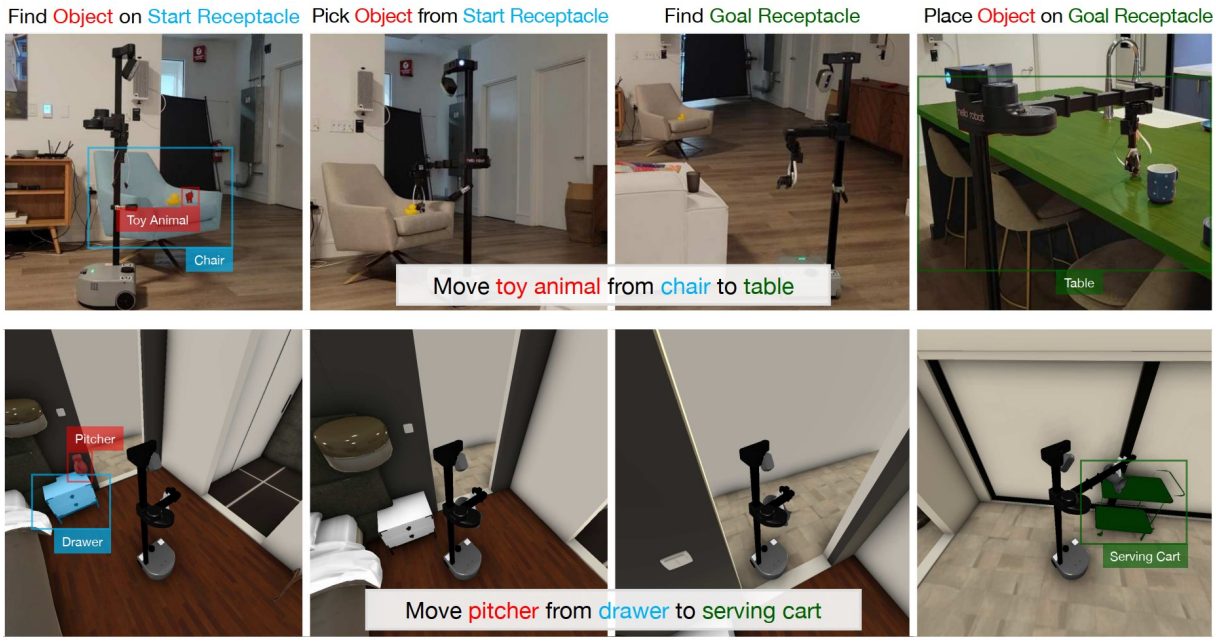


Figure 1.2: **A typical embodied AI robotic system [4].** Cases of the robotic agents doing chores according to the verbal instructions are included in both real-life environment (in the top row) and a synthetic scene (in the bottome row).

is noteworthy that the agent needs to have 3D perception to the enviroment so as to navigate itself, observe then differentiate different objects, and conduct operations. The 3D perception of agents is visualized in segmentation masking in colors and categories bounding boxes. For instance, in the scene of the first row, first column, the agent searches through according to the verbal instruction *"Move toy animal from chair to table"* and then in 3D space it identifies the target object as "toy animal", and the target start receptacle as "chair", as shown in the bounding boxes. Segementation in colors indicates different roles: Red indicates the target object, blue for the start receptacle, and green for the goal receptacle. Thus as per the agent's perception, the chair is shaded as blue and the toy animal is shaded as red. With all these inferences, the agent will subsequently conduct the operation in this step once the perception shown in the scene is established.

In this particular example, 3D perception is achieved by constructing a semantic voxel map which is built from 2D first-person semantic segementation. The 2D semantic segmentation is inferred by a deep convolutional neural network (DCNN) and then further back-project into the 3D counterpart. Researches about using DCNN and other NN variants are thriving nowadays, enabling many opportunites of better performance. They create metric-semantic reconstruction that can be used for the robotic perception more effectively.

Meanwhile, the robotic agent in Fig. 1.2 uses a RGB-D camera (Intel RealSense

Table 1.1: Computing Device Hierarchy with Normalized AlexNet Training Runtime

Hierarchy	Edge Devices	System on Chip	Personal Computer	Data Center	Supercomputer
Sample Device					
Typical Application					
HW. Cost (in USD)	<100	600	2k	176k (1 yr)	1B
Power (in Watt)	0.5	5	350	51k	60M
Normalized Training Runtime	10.4h (INT)	3.3h	5.6min	2.4s	<1s

D435i) with no mobile GPU on board. All the intensive model computations must be sent back to the GPU powered workstation for processing. Because the controller on board cannot afford the extensive computations in DNN models, which are normally handled by personal workstations or beyond in terms of compute capability as shown in the current computing platform hierarchy in Table 1.1. However, centralizing data from agents and computing on the workstation impose great burden on both the communication bandwidth between the agents and the workstation, and latency and privacy requirements on each agent.

Table 1.1 compares the mainstream compute devices and their typical applicaitons across the board. Controllers and processors on the robotic agent belong to the System-on-Chip category. Although the hardware beyond personal computers nowadays can handle the embodied AI tasks pretty well from the throughput/latency perspectives, they are totally not affordable in terms of robotic agents. Besides, the consumed power on the fly will also be a huge issue for a single robot. This dilemma raises a second research topic about distributing the compute efforts into local edge devices for both model training and inference.

Fig. 1.3 illustrates the necessity of on-device training. Data can be categorized as either sensitive or non-sensitive from the users' perspective. Chances are as the end user, we do not wish to expose our sensitive data to the giant corporates for whatever reasons. Furthermore, so far there is still transmission latency and bandwidth constraint for wireless channels from time to time. A good paradigm without data uploading is to pretrain the network on cloud, and then adopt it on the edge devices. The on-device training module in the center plays a critical role becuase it is the most challenging and computationally expenssive part in this paradigm.

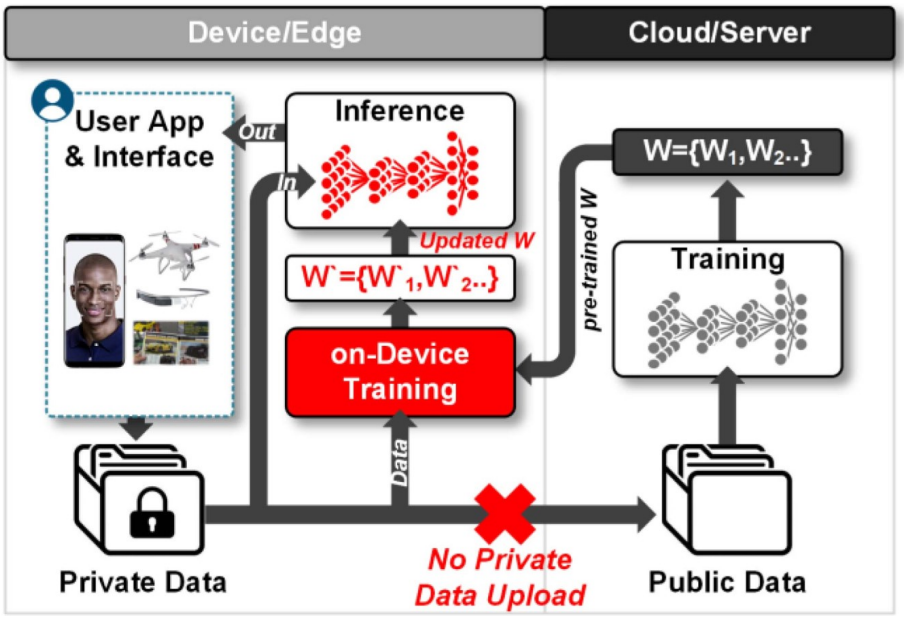


Figure 1.3: **Overview of the on-device learning** [5]**.** The pretrained models on cloud are deployed to the edge devices and updated by the on-device training using the local private data. As models are updated locally, inference can be conducted with the in-situ refreshed model for many intelligent applications, such as facial recognition, drone navigation, and object recognition.

1.3 Challenges Faced by the Perception System

Conventionally, 3D reconstruction is done by the point-clouds fusion which relies on range sensors. M. Levoy et al. [6, 7] first proposed the fusion technique using the back-projected 3D points of each depth map pixel to construct a 3D mesh in 1996. As the arising and democratization of the commodity computing device, this depth map fusion technique was able to come out of the laboratory and to be deployed for consumers' usage. Kinect-Fusion [8] is the first work that utilized the depth map fusion on the consumer level electronics to conduct real-time 3D reconstruction.

However, the measurement from range sensors is often noisy and suffering from low albedo issue. It is a natural attempt to explore solely visual perception so as to get rid of the dependence on range sensors. The major challenge for those visual-input methods with differential geometry is two-fold: Incompleteness given the intensive computations. It is reasonable because these multiview stereo algorithms are lifting the complexity of the task from 2D images to 3D volume, which are too complex and cumbersome to solve by hand-crafted designing rules and optimization methods. Therefore, learning-based methods, such as neural networks are widely explored aiming to fit a high-dimensional vector as weights to replace the nearly impossibly analytical solution of above.

On the other hand, from the hardware perspective of a humanoid, the power consumption that it takes to handle 3D perception models is mostly unaffordable for the processors on board, unlike the workstation GPUs. What's worse, the technology gains are diminishing with the increasingly apparent quantum tunneling effects as the size of transistors approaches the physical limit, meaning that we can not simply rely on the momentum of technology gains to nurture another miracle for the embodied AI just like the internet and mobile devices for the past three decades. Or, just gambling on quantum computing and hoping that one day it can be reliable. It has to be hardware efficiency improvement from the ground up of the computing architecture design. This will help us successfully achieve 3D perception for robotics and therewith enable ubiquitous intelligence.

There are research efforts dedicated to design the high-efficiency chips that accelerate 3D perception [9, 10]. Needless to say, the 3D perception in Fig. 1.2 is critical to embodied AI. Enhancing the energy efficiency of both software and hardware shown in Fig. 1.3, intrigues practitioners who are building real embodied AI agents despite the great challenges ahead.

1.4 Dissertation Organization and Contribution

This dissertation focuses on the algorihtm and hardware co-design methodology of the 3D visual perception system, for embodied AI applications. The previous contents so far have covered the "Why?" and "What?" regarding 3D perception in a high-level comprehensive survey manner. The following contents in the dissertation will answer the "How?" problem dedicated to the pursuit of effective and efficient 3D visual perception. An illustration for the roadmap is shown in Fig. 1.4. The dissertation covers the essentials for building an embodied AI agent's perception system. From the perspective of this humanoid, the visual data will first come in through the peripheral sensors (which are just like eyes and cerebella) and its frontend and then be processed by the onboard backend. The frontend shaded in green which consists of data capturing and perception algorithm, is covered in Chapter 2 and Chapter 3. It's like the control center of the humanoid's brain. The backend of the brain shaded in blue, where the actual learning and processing

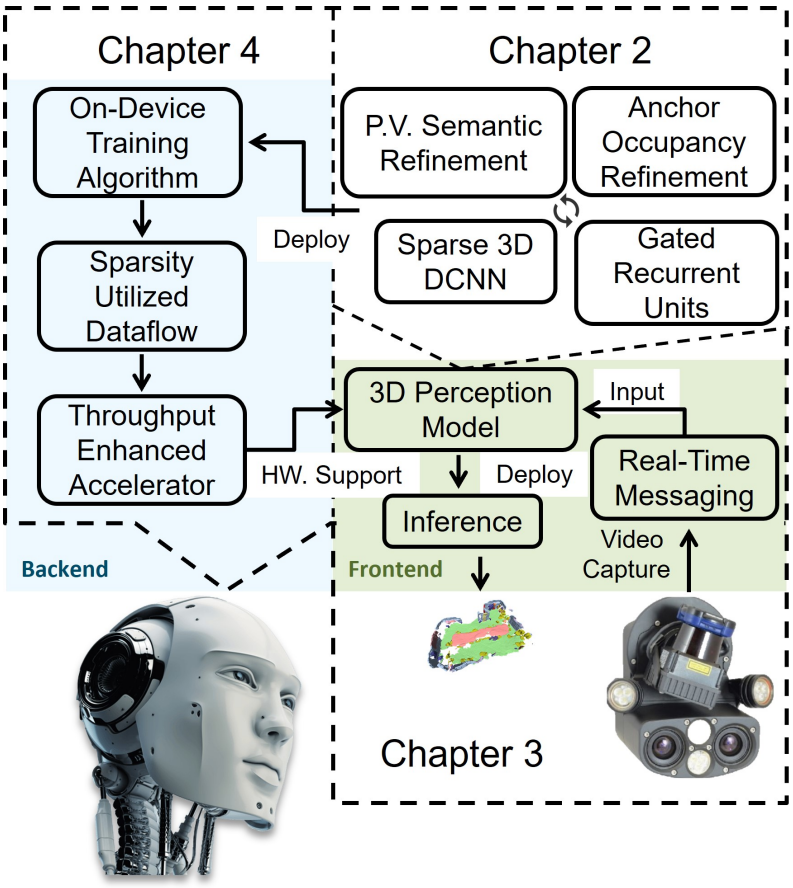


Figure 1.4: **Dissertation organization and contribution.** Dash lines encircle each of the following three chapters. Together they construct the 3D visual perception system of the embodied AI agent.

are conducted, is covered in Chapter 4.

Specially, we introduce the cross-dimensional refinement learning for 3D visual perception using video input in Chapter 2. We propose the neural network and its maximum-a-posteriori optimization paradigm that is capable of to have effective metric-semantic 3D mapping. In Chapter 3, we propose the real-time data streaming system for the 3D perception neural network and build up the prototype on mobile phones for demonstration on the edge side. In Chapter 4, we investigate the throughput-power budgets for visual perception especially on edge devices, and propose the gradient-pruned sign-symmetric feedback alignments for updating CNN at the edge with high energy efficiency. Chapter 5 concludes the dissertation.

2 3D Visual Perception from Monocular Video

"In order to investigate a subfield of a science, one bases it on the smallest possible number of principles, which are to be as simple, intuitive, and comprehensible as possible, and which one collects together and sets up as axioms."
— David Hilbert, *The New Grounding of Mathematics: First Report*

This chapter discusses a LiDAR-free 3D perception algorithm, which is sparse, 2D-prior-aided, and temporally coherent.

2.1 Introduction

Recovering 3D geometry and semantics of objects or environment scenes prevails these days with the advent of ubiquitous digitization. Not only can the digitization of the world where people live help them better understand their ambient environments, but also enables robots to comprehend what they need to know about the world and then conduct assigned tasks. Generally, with the ambient environment measurements as input, 3D reconstruction and 3D semantic segmentation are two key 3D perception techniques [11–13] in the computer vision society, which enable a wide range of applications, including digital twins [14, 15], virtual/augmented reality (VR/AR) [8, 16], building information modeling [17, 18], and autonomous driving [19, 20].

Tremendous research efforts have been made for 3D perception techniques. Conventionally, research on 3D perception utilizes active range sensors to capture surface geometry information. Originated from KinectFusion [8], the commodity RGB-D range sensor is

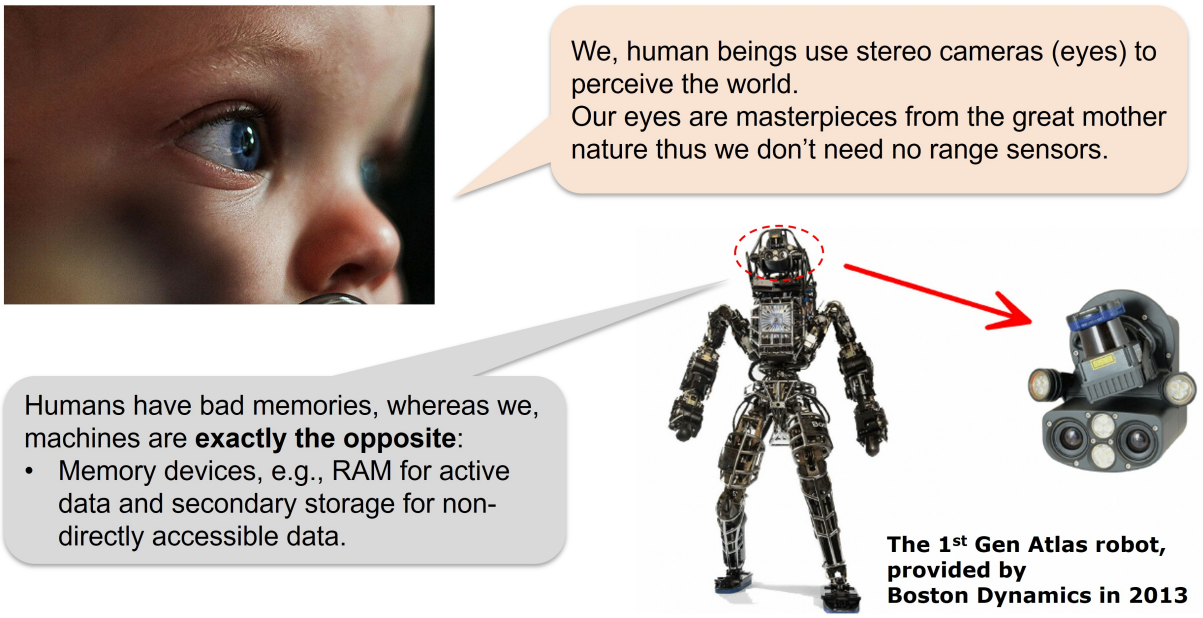


Figure 2.1: **Debate on 3D perception: Human beings vs. Machines, who's better?** Human and machines have their own advantages in different data modalities. The example of machines here, Atlas, which was released in 2013, is equipped with LiDAR and stereo cameras for 3D perception.

used to measure depth data first and then fuse it into Truncated Signed Distance Function (TSDF) volume for 3D reconstruction. Although the follow-up depth-based TSDF fusion methods [21–25] achieve detailed dense reconstruction results, they still suffer from global incoherence due to the lack of sequential correlation, the tendency of noise disturbance due to redundant overlapped calculations, and the incapability of semantic deduction due to the lack of texture features.

Inspired by the recent advancements of both 2D computer vision [26] and supporting hardware [27, 28], we believe that the ultimate modality for 3D perception should be visual input, and therefore we aim to develop a 3D perception system that requires visual input only. Although conventionally, machines rely on range sensors for 3D perception which seems disadvantageous from the perspective of building a humanoid, they still have their own advantage by having their nearly perfect working memory. This leads to an interesting debate between themselves and humans regarding visual perception in Fig. 2.1, wherein the accurate memory that machines possess enables their potential for 3D visual perception with just monocular and sequential input images, and no need for the binocular stereo as human eyes. Such a debate raises two interesting research questions. 1) **Mimicking**: *Can the machine achieve 3D perception with 2D RGB input only effectively?* 2) **Surpassing**: *Can the visual 3D perception mentioned be achieved with a monocular camera only?*

Before we delve into the discussion of the visual-based solutions for 3D perception, it is noteworthy that there is a line of research called Structure from Motion [29] that relies on RGB images to restore camera poses and ambient 3D structures. However, they are commonly designed for remote sensing and not able to meet the real-time requirements of robotic applications.

Other than that, some explorations on 3D perception with RGB cameras on mobile devices emerged given the ready availability of camera modules along with inertial measurement units. The problem of reconstructing 3D geometry with posed RGB images input only is referred to as multi-view stereo (MVS). Existing methods for MVS that are based on deep learning, tend to adopt a volumetric scheme by directly regressing the TSDF volume [16, 30–32] either as a whole or in fragments. However, these volumetric learning methods extract 3D geometric feature representation simply from the back projection of 2D image features, resulting in a mismatch to the 2D information priors for the predicted 3D reconstruction. Moreover, the intrinsic end-to-end learning manner and the lack of local details on the reconstructed mesh of these volumetric schemes result in an inferior semantic deduction based on its 3D reconstruction prediction.

What's worse, these learning-based methods tend to store their entire computational graphs in memory for aggregation and require prohibitive 3D convolution operations [30, 31, 33], which keeps them from being deployed on robots due to the real-time and low-latency requirements in SLAM. These limitations motivate our key idea to utilize 2D explicit predictions to further impose a light-weight feature refinement on the 3D features input in a sparse manner, while keeping the global coherence within the fragments. Unlike these preceding learning-based volumetric works, we conjecture that the utilization of 2D prior knowledge coming out of explicit predictions as a latent feature refinement plays a significant role in learning the feature representation in 3D perception. In addition, the feature refinement brought by 2D explicit prediction can be operated within the fragment input for keeping the computation redundancy and thus overhead low, while having the global coherence by correlating different fragments to extract the target 3D semantic mesh.

In this section, we propose a novel framework, *CDRNet*, to accomplish both 3D meshing and 3D semantic labeling tasks in real-time. This section is an extension of its conference version [34] with more analysis and discussions on the construction of fragment bounding volume, the derivation of the feature refinement under the maximum a posteriori optimization, and the implementation of the real-time 3D perception system.

Our key contributions are as follows.

1. We propose a novel, end-to-end trainable network architecture, which refines the 3D features cross-dimensionallywith the prior knowledge extracted from the explicit estimations of depths and 2D semantics.
2. The proposed cross-dimensional refinements yield more accurate and robust 3D reconstruction and semantic segmentation results. We highlight that the explicit estimations of both depths and 2D semantics serve as efficient yet effective prior knowledge for 3D perception learning.
3. To achieve real-time 3D perception capability, our approach performs both geometric and semantic localized updates to the global map. We present a progressive 3D perception system that is capable of real-time interaction with input data streaming from cellphones with a monocular camera.

We organize the remainder as below. A brief review of related works is presented in Sec. 2.2. In Sec. 2.3.1, we introduce the joint fragment learning on depth, 2D semantic category, intermediate TSDF, and occupancy using key frames input, for the following cross-dimensional refinements of TSDF and 3D semantics. For each fragment, the geometric features are progressively extracted in a coarse-to-fine hierarchy using binomial inputs GRU to build the learned representations of 3D. Sec. 2.3.2 describes the cross-dimensional refinements for 3D features that refines 3D features with anchored features and semantic pixel-to-vertex correspondences enabled by the depth and 2D semantic predictions, which helps the learning of not only the TSDF value but also the 3D semantic labeling in a sparsified manner. We also present the implementation details including loss design in Sec. 2.3.3.

2.2 Related Work

2.2.1 Voxelized 3D Semantic Segmentation

The learning of semantic segmentation on the voxelized map started from [35], which extends TSDF fusion pipeline [8] with per-pixel labels. 3DMV [36] and MVPNet [37] further combined both depth and RGB modalities to train an end-to-end network with 3D semantics for voxels and point clouds, respectively. PanopticFusion [38] performed map regularization based on adopting a CRF on the predicted panoptic labels. Kimera [39]

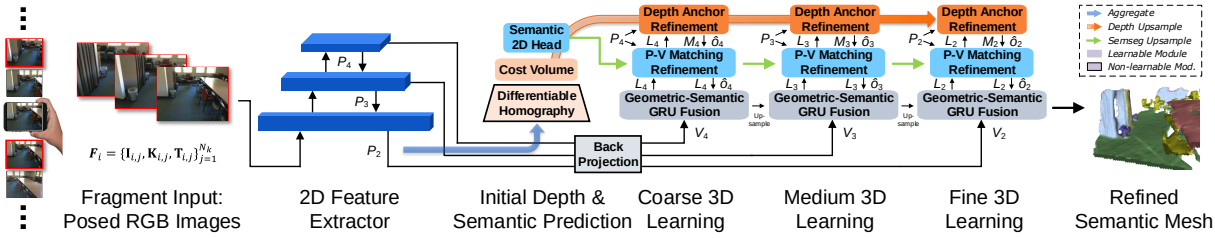


Figure 2.2: **Overview of CDRNet.** Posed RGB images from monocular videos are wrapped as fragment input for 2D feature extraction, which is used for both depth and 2D semantic predictions for cross-dimensional refinement purposes. To learn the foundational 3D geometry before conducting refinements, the extracted 2D features are back-projected into raw 3D features $\mathcal{V}_s$ in different resolutions without any 2D priors involved. At each resolution, after being processed by the GRU, the output feature $\boldsymbol{L}_s$ in the local volume is further fed into depth and semantics refinement modules sequentially to have a 2D-prior-refined feature with better representations.

incorporated visual odometry, multi-frame meshing, and 2D semantic annotation techniques together in a modular way, where an off-the-shelf 2D network was used to generate the 2D semantics and then directly annotate to each 3D point during the bundled ray-casting. Atlas [30] utilized its extracted 3D features and passed them to a set of semantic heads for voxel labeling, the pyramid features are proven to have strong semantics at all scales than the gradient pyramid in nature, as proven in [26]. BPNet [40] proposed to have a joint-2D-3D reasoning in an end-to-end learning manner. Two derivative works [41, 42] of RoutedFusion incorporated semantic priors into their depth fusion scheme and removed their routing module for less overhead. However, none of these works utilize the prior knowledge within the estimated 2D semantics as a 3D feature refinement.

2.2.2 Volumetric 3D Surface Reconstruction

Volumetric TSDF fusion became prevalent for 3D surface reconstruction starting from the seminal work KinectFusion [8] due to its high accuracy and low latency. A follow-up work, PSDF-Fusion [43] augmented TSDF with a random variable to improve its robustness to sensor noise. Starting from DeepSDF [43], the learned representations of TSDF using depth input dominates the current fad. These learning-based substitutes [15, 21–25, 44] to TSDF fusion achieve impressive 3D reconstruction quality compared to the baseline method with the availability of RGB-D range sensors.

Given the fact that range sensors have relatively higher cost and energy consumption than RGB cameras, MonoFusion [45] is one of the first works to learn TSDF volume from RGB images by fusing the estimated depth into an implicit model. Atlas [30] started the

trend of learning-based methods by a direct regression on TSDF volume. NeuralRecon [16] achieved a real-time 3D reconstruction learning capability by utilizing sparse 3D convolutions and recurrent networks with key frames as input. TransformerFusion [46] and VoRTX [31] introduced transformers [47] to improve the performance by more relevant inter-frame correlation. These learning-based methods prevail thanks to the availability of these general 2D feature extractors, such as FPN [26] and U-Net [48]. 2D information in RGB images can be effectively extracted and further utilized for constructing their 3D perception counterparts.

However, the learning of the explicit representations of 2D latent geometric features, such as depths and semantics, is **typically ignored** by all the prior arts. They only treat the 2D feature as an intermediate in the network and then conduct ray back-projection upon it, without considering the explicit representations for their 3D embodiment, which we found are significant prior knowledge for 3D perception. To extract depth as the explicit 2D representation, VolumeFusion [32] and SimpleRecon [49] performed local MVS and further fused it into TSDF volume with its customized network, while 3DVNet [33] performed sparse 3D convolutions on the feature-back-projected point cloud. Different from above, our method extracts the 2D representations from light-weight network modules, including a portion of MVSNet [50] for depth and a simple 2D MLP head for 2D semantics, to conduct the 3D feature refinements. The refinement incorporates the geometric and semantic prior information to improve the generalizability of our network by correlating the 2D representations in their 3D counterparts.

To the best of our knowledge, we present the very first learning-based method which uses posed RGB images input only to conduct 3D perception tasks in real time, including 3D meshing and semantic labeling.

2.3 Methods

Given a posed image sequence $\mathbf{I}$, our goal is to extract a 3D mesh model that can represent both **3D geometry** and **3D semantic labeling**, i.e., 3D meshing with vertices $\mathcal{K} \in \mathbb{R}^3$, surfaces $\mathcal{G} \in \mathbb{N}^3$, and its corresponding 3D semantic labeling $\mathcal{S} \in \mathbb{N}$. We achieve this goal by jointly predicting TSDF value $\boldsymbol{T} \in [-1, 1]$ and semantic label $\boldsymbol{S} \in \mathbb{N}$ for each voxel, and then extracting the mesh with the marching cubes [7]. Moreover, our proposed method aims to establish a real-time capable perception system for the two tasks. To quantitatively evaluate the efficiency of conducting these tasks simultaneously, we define

a 3D perception efficiency metric η_{3D} in Sec. 2.4.1 by involving frames per second (FPS) in runtime. The proposed network architecture is illustrated in Fig. 2.2.

2.3.1 Sparse Joint-Fragment Learning in a Coarse-to-Fine Manner

With the inherent nature of great sparsity in the ordinary real-world 3D scene, we efficiently extract the 3D feature from each input scene by sparse 3D convolutions to without redundant computations. Even though, the memory overhead of processing a 3D scene is still prohibitive, thus we fragment the whole 3D scene to release the memory burden of holding up the huge 3D volume data and progressively handle each of them. Inspired by [16, 30, 31, 33, 51], we adopt a resolution-varying coarse-to-fine learning paradigm for the sparse 3D convolutions to effectively exploit the representation of 3D features in multiple scales. At each resolution, the raw features before refinements in a fragment bounding volume (FBV) is extracted from a GRU by correlating local features and global feature volume, as described respectively in the subsections below.

FBV Construction by Image Features

Following [16, 52], we select a set of key frames as the input sequences out of a monocular RGB video by querying on each frame's pose, namely the relative translation and optical center rotation with empirical thresholds, θ_{key} and t_{key}. Key frames $\mathbf{I}$, and their camera intrinsics $\mathbf{K}$, and transform matrices $\mathbf{T} \in SE(3)$ which is an inversion of the camera pose, are all wrapped into a fragment $\mathbf{F}_i = \{\mathbf{I}_{i,j}, \mathbf{K}_{i,j}, \mathbf{T}_{i,j} \mid j = 1 \dots N_k\}$ as the input to the network, where i, j, and N_k denote the fragment index, the key frame index, and the number of key frames in each fragment, respectively. View frustums of each key frame's in the fragment combine into an FBV that is valid only at runtime. Fig. 2.3 illustrates the construction process of the corresponding FBV of a fragment. The blue-shaded planes represent the bottom of the view frustums whose minimum and maximum coordinates will be accumulated to form the boundary of FBV.

Once the fragment $\mathbf{F}_i$ is constructed, it is processed by a 2D feature extractor pyramid to extract image features. In the decoder part of the extractor pyramid, three different resolutions of feature maps are extracted sequentially as $\mathcal{P}_s \in \{\boldsymbol{P}_2, \boldsymbol{P}_3, \boldsymbol{P}_4\}$, where the suffix notation of $\boldsymbol{P}$ denotes the scaling ratio level in $\log_2$ similar to [26]. The extracted feature $\mathcal{P}_s$ in the Y-up camera coordinate is then back-projected into a local raw 3D features

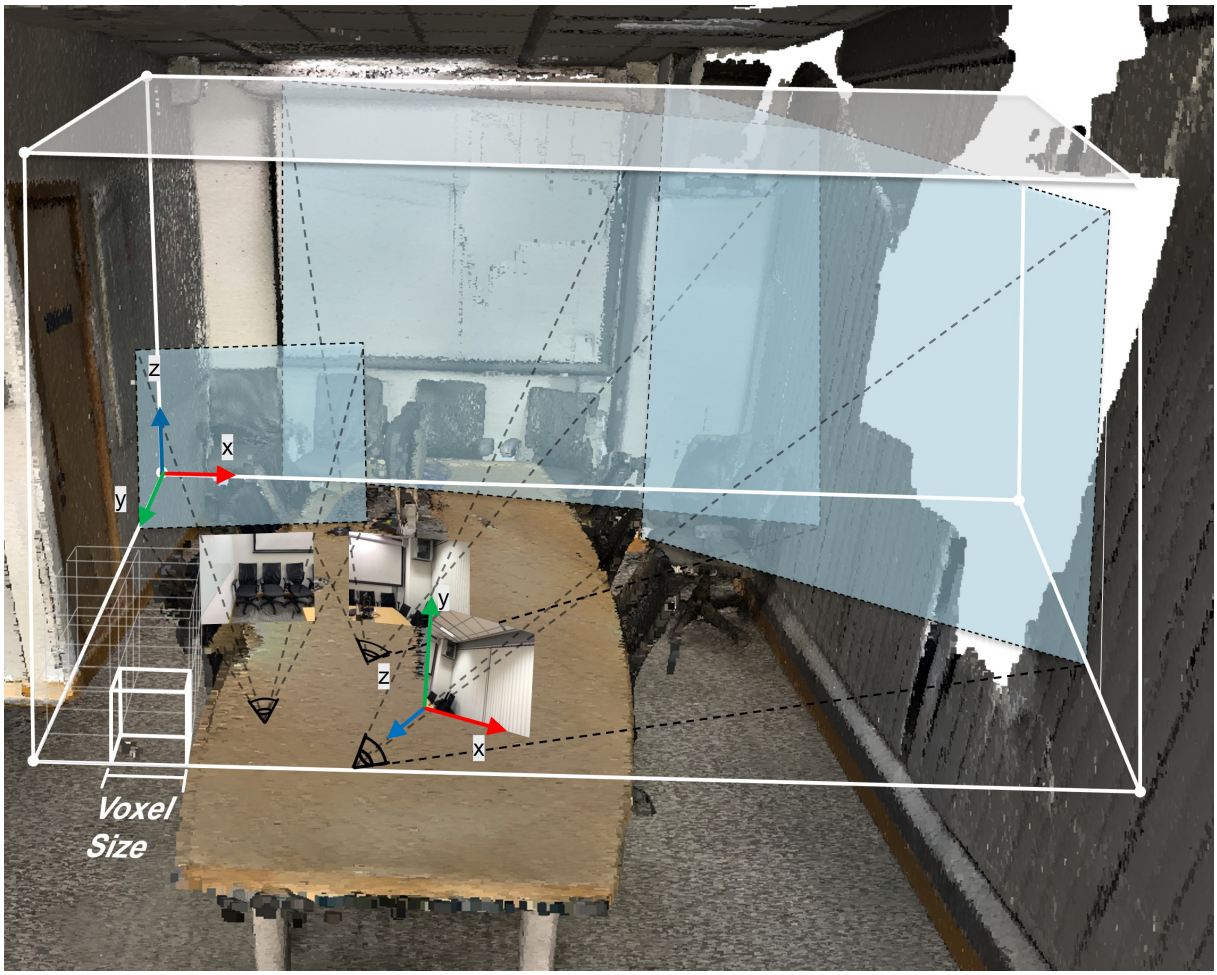


Figure 2.3: **Illustration of the FBV construction process.** The white box constructed with view frustums is defined as FBV. For simplicity, this toy example is constructed with $N_k = 3$ and voxelized with a given voxel size.

$\mathcal{V}_s$ in the Z-up world coordinate, according to the projection matrix of each frame in $\mathbf{F}_i$. Raw 3D features in the FBV are first voxelized and sparsified starting from the coarse stage, and then used to index and produce features of the next stage. We hereby define the FBV of the current resolution as $\mathcal{F}_{s,i} = \{\boldsymbol{T}_{s,i}^{x\times y\times z}, \boldsymbol{S}_{s,i}^{x\times y\times z}\}$ that is conditioned on the pyramid layers $\mathcal{P}_s$, where all the 3D voxels that are casted in the view frustums of current $\mathbf{F}_i$ are included.

Initial Depth and 2D Semantics Learning

With the fine feature $\boldsymbol{P}_2$ as input, we build up differentiable homography fronto-parallel planes for the coarse-level depth prediction $\hat{\boldsymbol{D}}_4$. Likewise, 2D semantics prediction $\hat{\boldsymbol{S}}_4^{2D}$ is extracted with a pointwise convolutional decoder as the 2D semantic head using $\boldsymbol{P}_2$. The resolution gap between the input and output feature map provides generalizability. The initial depth estimation is retrieved from the features using a light-weight multi-view

stereo network via plane sweep [50]. For each source feature map x in P_2, we conduct the planar transformation $\mathbf{x}_j \sim \mathbf{H}_j(d) \cdot x$, where "$\sim$" denotes the projective equality and $\mathbf{H}_j(d)$ is the homography of the j^{th} key frame at depth d. The j^{th} homography[1] in a given $\mathbf{F}_i$ is defined as:

$$\mathbf{H}_j(d) = d \cdot \mathbf{K}_j \cdot (\mathbf{T}_j \cdot \mathbf{T}_1^{-1}) \cdot \mathbf{K}_1^T \ . \tag{2.1}$$

To measure the similarity after conducting homography warping, we calculate the variance cost of $\mathbf{x}_j$ and further process it with an encoder-decoder-based cost regularization network. The output logit from the regularization network is treated as the depth probability on each plane and the *soft argmin* [50] is conducted to have initial depth predictions.

Geometric and Semantic GRU Fusion

As mentioned in Sec. 2.3.1, as the 2D features are extracted in different resolutions, they are back-projected from each of the pyramid level in $\mathcal{P}_s$ into raw geometric 3D features $\mathcal{V}_s \in \{V_2, V_3, V_4\}$, which are further sparsified by sparse 3D convolutions. To improve the global coherence and temporal consistency of the reconstructed 3D mesh, following [16], we first correlate the sparse geometric feature $\mathcal{V}_s$ in the current $\mathcal{F}_{s,i}$ using GRU, with the local FBV hidden states $\boldsymbol{H}_{s,i-1}$ whose information coming from all of the previous fragments $\mathcal{F}_{s,i'}, i' < i$ and coordinates are masked to be the same as $\mathcal{V}_s$. Such correlation outputs a temporal-coherent local feature $\boldsymbol{L}_{s,i}$ for each stage s, which is used to generate dense occupancy intermediate $o_{s,i}$, and passed to the 2D-to-3D cross-dimensional refinements. To fuse the global feature volume for the entire scene $\boldsymbol{G}_{s,i}$, we first densify $\boldsymbol{G}_{s,i-1}$ and $\boldsymbol{L}_{s,i}$ into real world point coordinates and further sparsify them with the point coordinates for upcoming sparse convolutions. In the GRU, $\boldsymbol{G}_{s,i-1}$ and $\boldsymbol{L}_{s,i}$ are first processed by sparse convolutions and then serving as the hidden state and candidate input, respectively. $\boldsymbol{H}_{s,i}$ is accordingly updated with the coordinates of $\mathcal{V}_s$ as masks.

Unlike [16], we reuse the same parameters in GRU to process the back-projected and upsampled 3D semantic features to generalize better for the semantic prediction $\hat{\boldsymbol{S}}$ in the current FBV. This is because inputting TSDF and semantic features sequentially into GRU enables its selective fusion across modalities, thus the feature extracted from the hidden state incorporates more semantic information, as pointed out in [53]. For the sake of learning 3D features consistently between scales, we update $\mathcal{V}_s$ at each stage by fusing with the upsampled $\boldsymbol{L}_{s+1,i}$. Inspired by the *meta data* mechanism proposed in [49], we

[1]For brevity's sake, the transformation from homogeneous coordinates to Euclidean coordinates in the camera projection is omitted here.

further concatenate sparse features, with sparse TSDF, occupancy and semantics after masking with $o_{s,i}$, as the meta feature $\boldsymbol{L}_{s+1,i}$ to be upsampled. We found the inclusion of semantic information in the hidden state of GRU helps build up a good starting point for the upcoming feature refinements, which is verified in the ablation.

2.3.2 2D-to-3D Cross-Dimensional Refinements

The raw coherent features from GRUs lack detailed geometric descriptions, leading to unsatisfactory meshing and semantic labeling results. To overcome these issues, we propose to leverage the 2D feature that is latent after incorporating the learning of depth and semantic frame for refinement purposes. We notice that with the learning of depth and 2D semantics, the 2D features now reside in the latent space which can generalize to more accurate 3D geometry and semantics via cross-dimensional refinements.

2D-to-3D Prior Knowledge

Consider a probabilistic prior in the latent space of the output coherent feature coming from GRU, which accounts for the prior knowledge that the pixel information in both depth predictions and 2D semantic predictions should produce high confidence matching with regard to their own 3D representations. The prior conditioned 3D feature for both perception tasks is defined as:

$$X_{prior} = f(\boldsymbol{L}_{s,i}) = f\left(\boldsymbol{H}_{s,i}(\mathcal{V}_s, \boldsymbol{H}_{s,i-1} \mid \mathcal{F}_{s,i})\right) \ , \tag{2.2}$$

where $f(\cdot)$ is the 2D-to-3D feature refinement process for either 3D meshing or 3D semantic labeling, whose input is $\boldsymbol{L}_{s,i}$ extracted from $\mathcal{V}_s$ and $\boldsymbol{H}_{s,i-1}$ given $\mathcal{F}_{s,i}$. We borrow the notation of $\boldsymbol{H}_{s,i}$ to be a constructor function $\boldsymbol{H}_{s,i}(\cdot)$ indicating GRU. For each voxel in $\mathcal{F}_{s,i}$, both TSDF and semantic labeling predictions can be formulated as:

$$\hat{\boldsymbol{I}}_{s,i} = \epsilon h\left(\boldsymbol{H}_{s,i}(\mathcal{V}_s, \boldsymbol{H}_{s,i-1} \mid \mathcal{F}_{s,i})\right) + (1-\epsilon)X_{prior} \ , \tag{2.3}$$

where $\hat{\boldsymbol{I}}_{s,i} \in \mathcal{F}_{s,i}$ is the refined prediction; ϵ is a random variable for the respective prior, which is jointly learned by the feature refinement modules representing the 2D-to-3D priors and the GRU network trained with maximum likelihood estimation losses; $h(\cdot)$ is the prediction head. The proof of Eq. (2.3) can be found in Sec. 2.3.2.

The key insight is that the voxels back-projected from either depth prediction or

semantic label prediction of the input images has strong evidence on its 3D counterparts. We hereby define anchored voxels $\boldsymbol{\alpha}_i$, as those voxels in $\mathcal{F}_{s,i}$ that are incorporating all the back-projected depth points, given the fact that the 3D reconstruction task is essentially an inverse problem. We propose two progressive feature refinement modules as follows to learn the high confidence of the refined features in latent space such that a more accurate $\hat{\boldsymbol{I}}_{s,i}$ can be extracted with the help of 2D-to-3D prior knowledge.

MAP Optimization

The derivation for Eq. (2.3) is obtained here.

Proof. Considering the 3D feature extraction network described in Sec. 3.1 of the main body **without** CDR priors, the temporal-coherent local feature $\boldsymbol{L}_{s,i}$ at stage s can be inferred by a parametric GRU fusion with the input of a concatenation between the raw geometric feature $\mathcal{V}_s$ and the upsampled 3D feature from previous stage $\boldsymbol{L}_{s+1,i}$ as,

$$\boldsymbol{L}_{s,i} = \boldsymbol{H}_{s,i}\big(\text{Concat}(\mathcal{V}_s, \text{Up}(\boldsymbol{L}_{s+1,i})), \boldsymbol{H}_{s,i-1}\big) \ , \tag{2.4}$$

$$\boldsymbol{H}_{s,i} = \boldsymbol{G}_{s,i}[\sum_{i'=1}^{i} \mathcal{F}_{s,i'}.\text{coords}()] \ , \tag{2.5}$$

where the hidden state of $\boldsymbol{H}_{s,i}$ under current fragment $\mathbf{F}_i$ is extracted from the global feature volume $\boldsymbol{G}_{s,i}$ with the masking coming from the coordinates of $\mathcal{V}_s$ as valid. The target quantity prediction is estimated from the previous hidden state $\boldsymbol{H}_{s,i-1}$ as below,

$$\hat{\boldsymbol{I}}_{s,i} = h(\boldsymbol{L}_{s,i}) = h\big(\boldsymbol{H}_{s,i}(\mathcal{V}_s, \boldsymbol{H}_{s,i-1} \mid \mathcal{F}_i)\big) = g_{\hat{\theta}}(\mathbf{F}_i) \ . \tag{2.6}$$

where the hidden layer $h(\cdot)$ is constructed with a single layer perceptron for the target quantity prediction head, θ denotes the overall trainable parameters in the network, and naturally $g_{\hat{\theta}}(\cdot)$ denotes the entire neural network mapping from the posed image fragment to the target quantity prediction under the trained $\hat{\theta}$.

Then at the time of training, given N_b batches of $\mathbf{F}_i$ as the input and the voxel data pair values in the sparsified FBV, $\boldsymbol{I}_i \in \mathcal{F}_i = \{\boldsymbol{T}_i^{N_o}, \boldsymbol{S}_i^{N_o}\}$ as the groundtruth of the target quantity estimation for each stage of the coarse-to-fine hierarchy while the number of occupied voxels is defined as,

$$\boldsymbol{N}_o = \text{len}\big(\mathcal{V}_s^{x\times y\times z}[\hat{o}_i].\text{flatten}()\big) \ , \tag{2.7}$$

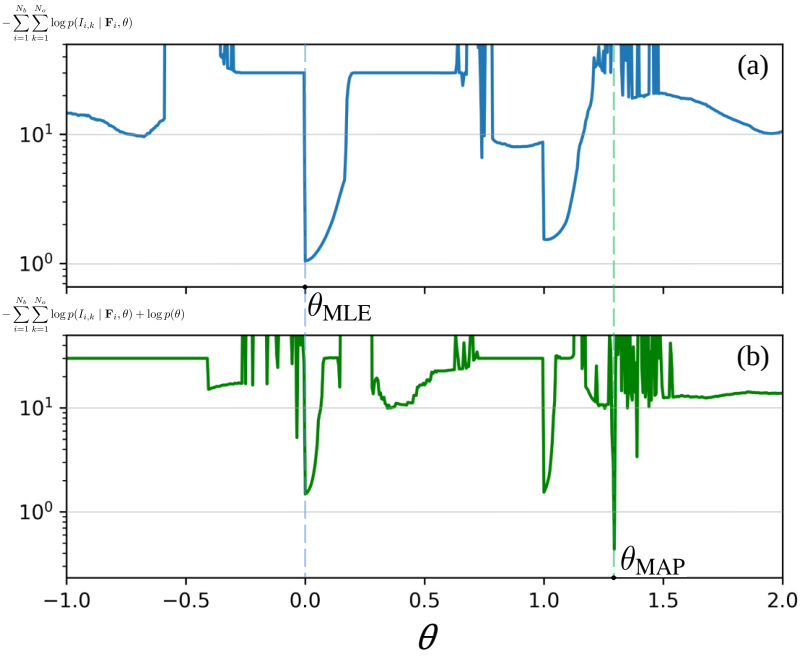


Figure 2.4: **Loss function against trainable θ on the ScanNet validation dataset.** (a) Negative log-likelihood against θ, namely the target function in Eq. (2.8); (b) Negative log-product of posterior and evidence against θ, namely the target function in Eq. (2.10). θ_{suffix} denotes the optimal θ in the "suffix" optimization situation. Optimizing with MAP in (b) can generalize more expressive θ as justified in the experiments.

θ can be estimated following *maximum likelihood estimation* (MLE) as below:

$$\begin{aligned}
\hat{\theta}_{\text{MLE}} &= \arg\max_{\theta} \prod_{i=1}^{N_b} \prod_{k=1}^{N_o} p(\boldsymbol{I}_{i,k} \mid \mathbf{F}_i, \theta) \\
&= \arg\max_{\theta} \sum_{i=1}^{N_b} \sum_{k=1}^{N_o} \log p(\boldsymbol{I}_{i,k} \mid \mathbf{F}_i, \theta) \\
&= \arg\min_{\theta} \sum_{l \in \{T,S\}} \sum_{i=1}^{N_b} \sum_{k=1}^{N_o} \mathcal{L}_l(\hat{\boldsymbol{I}}_{i,k}, \boldsymbol{I}_{i,k}) \ ,
\end{aligned} \tag{2.8}$$

where $\mathcal{L}_l$ is the respective loss that is responsible for each type of target quantity. It can be categorized as either a regression or a classification problem and hence derived from the likelihood $p(\boldsymbol{I}_{i,k} \mid \mathbf{F}_i, \theta)$. Readers are suggested to refer to Sec. 5.5 in [54] for further details about the loss. Without loss of generality, we plotted the $\mathcal{L}_l(\theta)$-θ characteristic in Fig. 2.4 by adopting the same visualization methods described in [55, 56]. Fig. 2.4a shows the mean absolute error loss for Truncated Signed Distance Function (TSDF) prediction, where θ is optimized with MLE and finalized as $\hat{\theta}_{\text{MLE}} = \theta_{\text{MLE}}$. Thus, at inference with

the trained parameters substituted in Eq. (2.6),

$$\hat{I}_i = g_{\hat{\theta}_{\mathrm{MLE}}}(\mathbf{F}_i) = g_{\theta_{\mathrm{MLE}}}(\mathbf{F}_i) \ . \tag{2.9}$$

Via Bayes' rule, we hope to optimize the training process in a *maximum a posteriori* (MAP) perspective with the CDR priors, namely,

$$\begin{aligned}
\hat{\theta}_{\mathrm{MAP}} &= \arg\max_{\theta} \prod_{i=1}^{N_b} \prod_{k=1}^{N_o} p(I_{i,k} \mid \mathbf{F}_i, \theta) \times p(\theta) \\
&= \arg\max_{\theta} \sum_{i=1}^{N_b} \sum_{k=1}^{N_o} \big(\log p(I_{i,k} \mid \mathbf{F}_i, \theta) + \log p(\theta)\big) \\
&= \arg\min_{\theta} - \sum_{i=1}^{N_b} \sum_{k=1}^{N_o} \big(\log p(I_{i,k} \mid \mathbf{F}_i, \theta) + \log p(\theta)\big) \ ,
\end{aligned} \tag{2.10}$$

where $p(\theta)$ denotes the CDR prior probability on network parameters which incorporates 2D information to construct the 3D counterpart. The intuitive mechanism is explained in Sec. III-B1. To generate the CDR priors, we abstract the CDR modules as a dedicated hidden-layers network $h_{prior}(\cdot)$ whose 2D prior distribution $p(\theta)$ is included. The target quantity estimation with CDR priors involved can be retrieved as,

$$\begin{aligned}
\hat{I}_{i_{\mathrm{MAP}}} &= g_{\theta_{\mathrm{MAP}}}(\mathbf{F}_i) \\
&= h\big(h_{prior}(H_{s,i})\big) \\
&= \epsilon g_{\theta_{\mathrm{MLE}}}(\mathbf{F}_i) + (1-\epsilon) g_{\theta_{prior}}(\mathbf{F}_i) \ ,
\end{aligned} \tag{2.11}$$

where $g_{\theta_{prior}}(\cdot)$ is the functional mapping from $\mathbf{F}_i$ to $h_{prior}(H_{s,i})$ similar to Eq. (2.6); ϵ denotes the weighting factor that is learnt by the network to generalize the offset relationship between θ_{prior} and θ_{MLE}. Merging the distributions of θ_{MLE} and θ_{prior}, we have the posterior corresponding to $g_{\theta_{\mathrm{MAP}}}(\mathbf{F}_i)$ and the optimization $\hat{\theta}_{\mathrm{MAP}} = \theta_{\mathrm{MAP}}$, which are shown in Fig. 2.4b.

At the time of inference when training is done, with the estimated parameter $\hat{\theta}$ in the trained network, the hidden state output from GRU fusion is equivalent to the MLE optimized parameter's result as in Eq. (2.6). While the prior-conditioned 3D feature is

equivalent to prior-optimized result, namely,

$$g_{\theta_{\text{MLE}}}(\mathbf{F}_i) = h\big(H_{s,i}(\mathcal{V}_s, H_{s,i-1} \mid \mathcal{F}_i)\big) \ , \tag{2.12}$$

$$g_{\theta_{prior}}(\mathbf{F}_i) = X_{prior} \ . \tag{2.13}$$

Thus, by substituting Eq. (2.12) and Eq. (2.13) into Eq. (2.11), Eq. (3) holds. Similarly, $\hat{\theta}$ that is responsible for semantics predictions can be retrieved by enforcing $\mathcal{L}_l = \mathcal{L}_{\text{CE}}$. The proof is complete. □

It is noteworthy that, to visualize the loss landscape for such a high-dimensional deep neural network (beyond 12 million parameters in CDRNet), it is impossible to compare the loss sweeping through all variable θ entries even after some dimension decomposition techniques such as PCA. Therefore, to investigate the loss landscape of the neural network, one needs to find a workaround to bypass the θ sweeping. To this end, we adopt the same loss visualization methods as in [55, 56] by computing:

$$\begin{aligned} \mathcal{L}_l(g_\theta(\mathbf{F}_i), \boldsymbol{I}_i) &= \mathcal{L}_l(\theta) \\ &= \mathcal{L}_l((1-\alpha)\cdot\theta_{\text{MLE}} + \alpha\cdot\theta_{prior}) \ , \end{aligned} \tag{2.14}$$

where $\mathcal{L}_l(\theta)$ is achieved by substituting Eq. (2.6) into Eq. (2.8) for varying α who serves as a linear interpolation between two different parameter vectors in parameter space. Such visualization enables one to comprehend how is CDR priors benefit to the learning of the backbone network by optimizing into θ_{MAP} with a lower testing loss in parameter space. Fig. 2.4 shows the $\mathcal{L}_T(\theta)$-θ characteristics under MLE and MAP optimizations, respectively. In Fig. 2.4a, we remove feature refinement modules in CDRNet to retain the network in an MLE optimization fashion, the network parameters are learned as $\hat{\theta} = \theta_{\text{MLE}}$; whereas in Fig. 2.4b, the parameters in CDRNet are learned as $\hat{\theta} = \theta_{\text{MAP}}$ which is lower than θ_{MLE} with the help of the 2D-to-3D prior knowledge $p(\theta)$ during optimization, proving the efficacy in generalizability of the proposed 2D-to-3D refinement method.

Depth-Anchored Occupancy Refinement

Unlike the volumetric methods [16, 30] that directly regress on the TSDF volume, we propose a novel module in each stage s that can explicitly refine the initial depth, predict depths in resolutions, and further create the 3D anchored features with the depth predic-

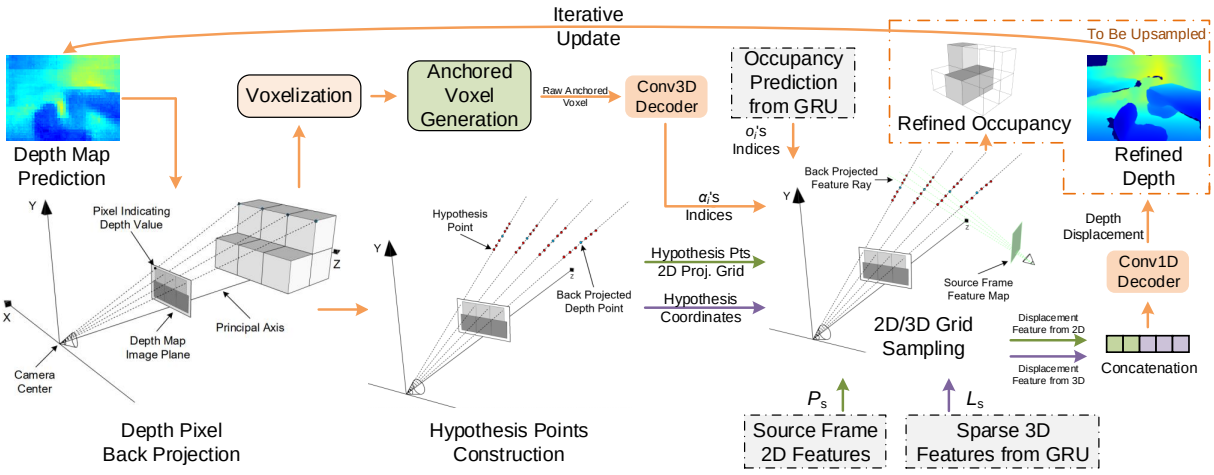


Figure 2.5: **Workflow of the depth anchor refinement.** Anchored voxels are extracted from depth points and further serve as a geometric prior for the occupancy refinement.

tion, as shown in Fig. 2.5. The anchored feature is generated by 3D sparse convolutions with an anchored voxel on o_i [2]. Intuitively, the anchored voxel has higher confidence of achieving a valid o_i and $T_{s,i}$ close to zero. We imposed the anchored feature on the occupancy feature to reinforce the occupancy information brought by the depth prior.

Inspired by [33, 57], we conduct PointFlow algorithm for each stage in the coarse-to-fine structure $\mathcal{V}_s$ to determine the depth displacement on the initial depth prediction such that finer depth prediction can be achieved. Different from the PointFlow algorithm used in [33], we utilize the back-projected depth points from all N_k views in the fragment to query an anchored voxel, which can be further aggregated with o_i. Fig. 2.6 illustrates how these hypothesis points are selected and turned into depth displacement prediction, such that the anchored voxel can be generated.

The anchored voxel index in the 3D volume is sparsified as a mask to update the occupancy prediction as $\hat{o}_i$ in the following:

$$\hat{o}_i = o_i \cap \alpha_i \quad . \tag{2.15}$$

The enhanced occupancy prediction $\hat{o}_i$ is used to condition the TSDF volume at the current stage to generate the refined $\hat{T}_i$, which is further sparsified with a light-weight pointwise convolution and upsampled to concatenate with $L_{s,i}$.

[2]The universal stage suffix s is hereinafter omitted for brevity.

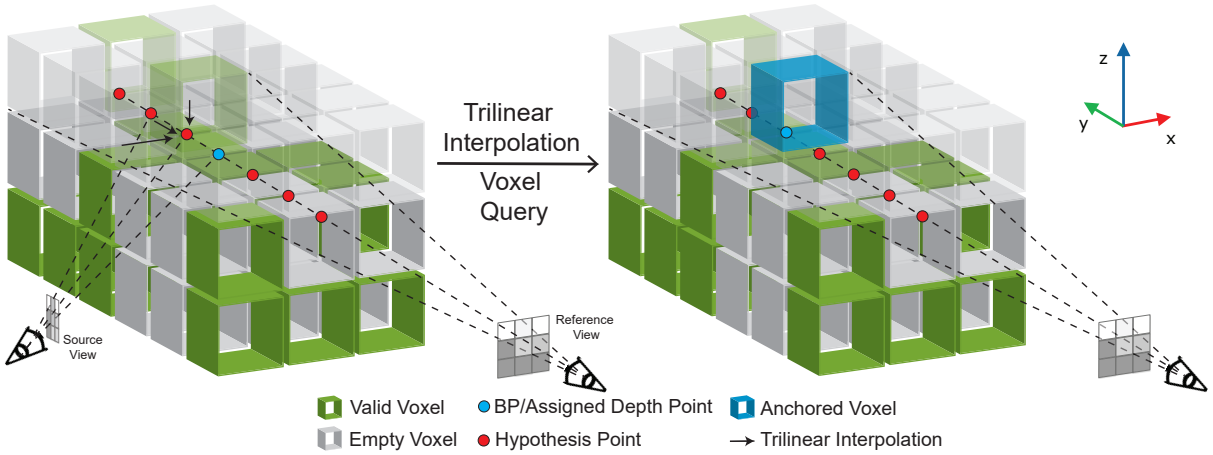


Figure 2.6: **Anchored voxel generation for occupancy refinement.** An example of occupancy refinement happening on the middle row of a $3 \times 6 \times 3$ FBV is shown with geometrically valid voxel highlighted in green. The initial depth prediction is back-projected into FBV and displaced by trilinear interpolation on all depth points, in the range of 6 additional hypothesis points for each depth point. The voxels on the top are set as half transparent for clarity.

Pixel-to-Vertex Matching Semantic Refinement

In addition to the depth anchor refinement, we propose a semantic cross-dimensional refinement which utilizes the semantic prior that lies in the 2D semantic prediction to have a refined 3D voxel semantic prediction. First, we impose a 2D semantic head on the 2D feature backbone for learning the 2D semantic prior information that is useful for 3D voxel semantic labeling. Second, the sparse 3D feature $L_{s,i}$ is passed to pointwise 3D convolution layers and comes up with the initial 3D voxel semantic labeling predictions in respective scales. Third, to conduct the semantic feature refinement, we observed that there is a sole 3D voxel counterpart in $\mathcal{F}_{s,i}$ for each pixel on a 2D semantic prediction of $\mathbf{I}_{i,j}$, since the surface edges are encoded as vertices. We define these vertices as the one-on-one matching correspondences to their camera-projected pixels, which are recorded in a matching matrix for masking the 2D features $\mathcal{P}_s$.

The upper part of Fig. 2.7 illustrates the design of such matching matrix so as to correlate the pixel-vertex pairs for each frame $\mathbf{I}_{i,j}$ across all vertices in $\mathcal{F}_{s,i}$. We construct the matching matrix $\mathbf{M} = \{\vec{m}_{idx}\}_{idx=1}^{N}$ for each semantic labeling frame, where N is the number of the vertices in the volume $\mathcal{F}_{s,i}$. Each column of the matching matrix $\mathbf{M}$ is defined as:

$$\mathbf{M}(idx) = \vec{m}_{idx} = \begin{bmatrix} u_{idx} \\ v_{idx} \\ \text{mask} \end{bmatrix}. \tag{2.16}$$

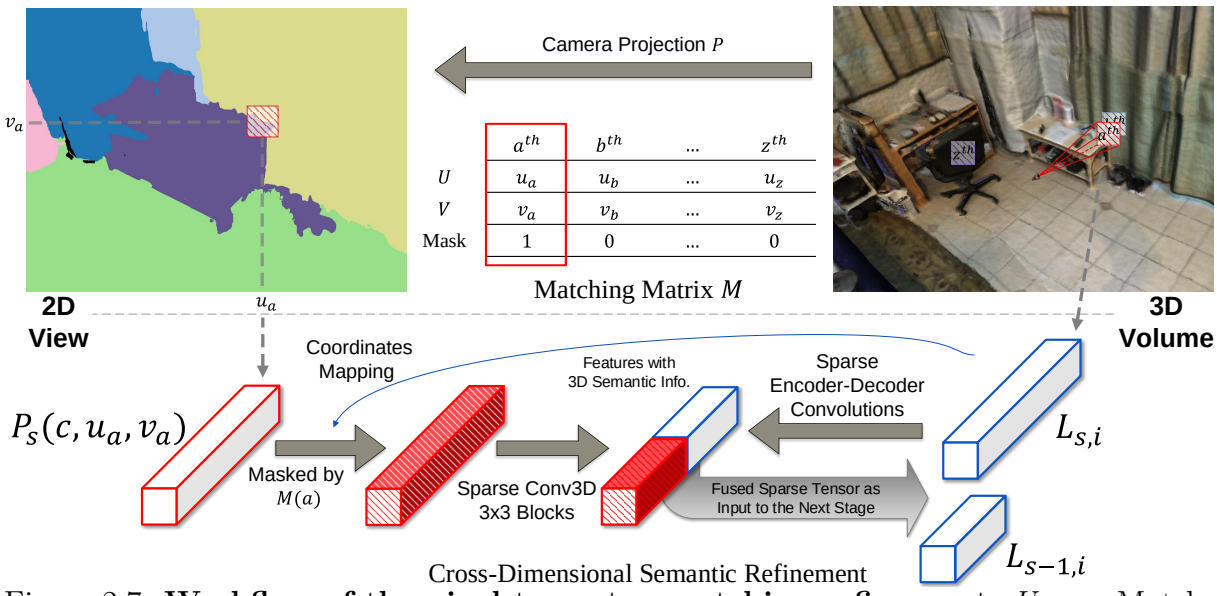


Figure 2.7: **Workflow of the pixel-to-vertex matching refinement.** *Upper:* Matching matrix $\mathbf{M}$ for pixel-to-vertex correspondence is constructed with camera projection. The red-shaded boxes in the 3D volume denote an example of valid correspondence pairs of the 2D semantic prediction $\vec{m}_a$ and its surrounding 3D scene. The green and purple boxes in the 3D volume view denote the occluded vertex and out-of-view vertex that is not imaged in the 2D semantic prediction, which correspond to $\vec{m}_b$ and $\vec{m}_z$, respectively; *Lower:* The 2D features are further masked by $\mathbf{M}(a)$ with the mapped coordinates from the sparse 3D features of the scene that are valid for the current view.

For each column, each pixel-vertex pair recorded in the matching matrix, i.e., the idx^{th} vertex in the 3D volume on the right-hand side of the upper part and its correspondence pixel on the left-hand side is recorded. The last entry of the pixel-vertex pair represents a mask which is recorded as valid when the 2D correspondence for $\mathbf{M}$ is in the current view frustum of the frame.

In the lower part of Fig. 2.7, the constructed matching matrix $\mathbf{M}$ will be used for masking each of the feature maps $\mathcal{P}_s$ with the $\log_2$ scale of s to create a refined feature, whose voxel number is the same as the number of sparse 3D features. Meanwhile, the coordinates of the sparse 3D features $\boldsymbol{L}_{s,i}$ are mapped as the coordinate of the refined feature. By doing so, the underlying semantic information from the $\mathcal{P}_s$ can be incorporated by $\boldsymbol{L}_{s,i}$, such that better 3D semantic prediction can be achieved. Then we use the sparse pointwise convolution to extract its underlined feature from 2D semantics and concatenate it with $\boldsymbol{L}_{s,i}$ to create $\boldsymbol{L}_{s-1,i}$ with semantic information for the refinement in the next finer stage, so as to ensure the 2D semantic priors to have reliable refinement on the sparse coherent features. Significant model operations in the refinement modules are shown in Fig. 2.8.

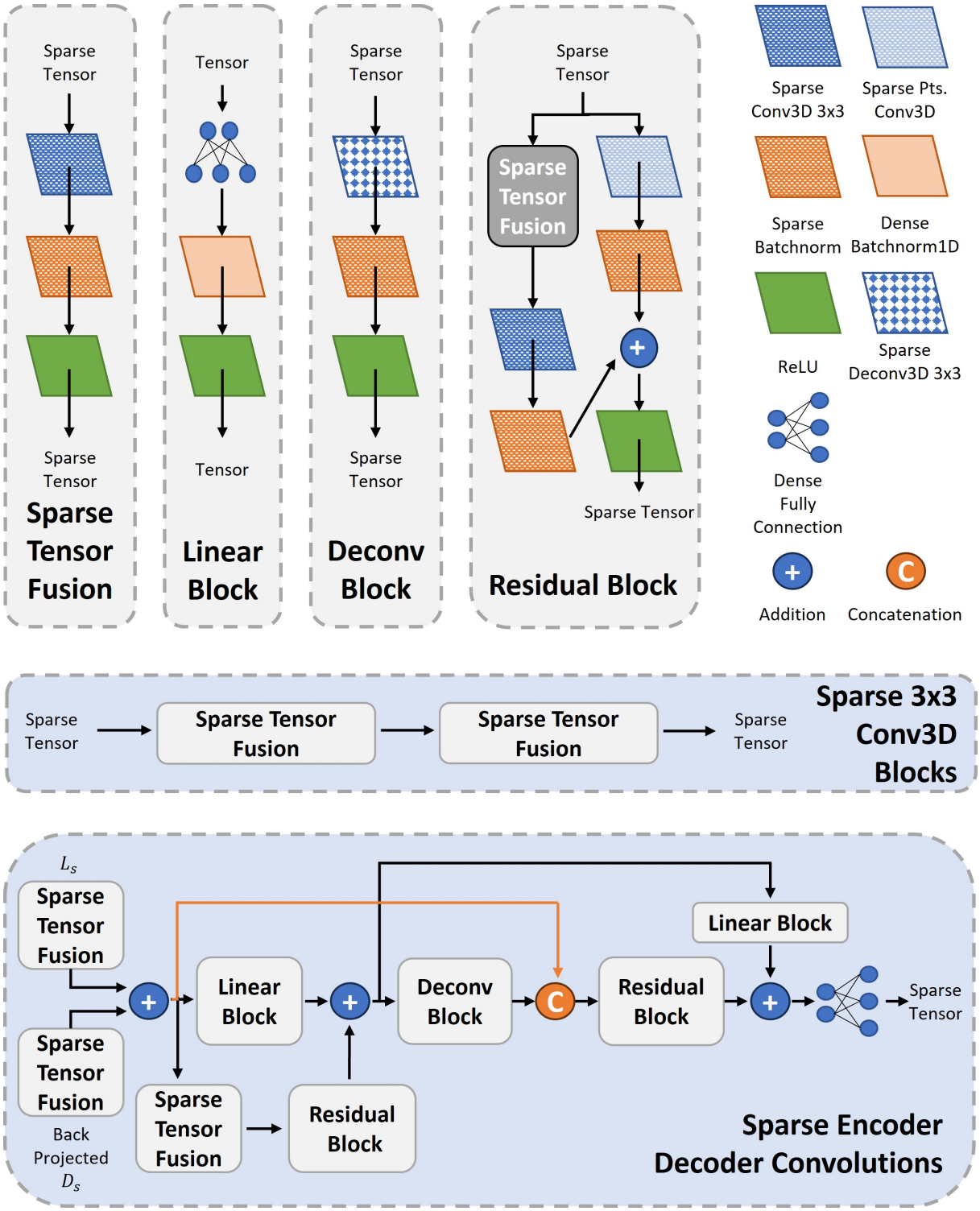


Figure 2.8: **Network module operations.** The sparse 3x3 3D and encoder-decoder convolutions in the pixel-to-vertex matching refinement are illustrated.

2.3.3 Implementation Details

Our model is implemented in PyTorch, trained and tested on an NVIDIA RTX3090 graphics card.

Network Related Choices

Following the existing sophisticated network layer block definitions in [26, 58], we first define a layer block as the combination of either a parameterized operation (e.g., a convolutional layer or a fully-connected layer) with a batch normalization layer, or a parameterized operation with a batch normalization layer plus an activation, such as ReLU. We train our network in an end-to-end manner with randomly initialized weights except that we load the 2D feature extractor, MNasNet from the ImageNet-pretrained checkpoints [59]. We adopt trilinear interpolation for refining both displaced depth predictions and anchored features, and nearest-neighbor interpolation for upsampling features to the next stage in the coarse-to-fine hierarchy. At each stage, we use Sigmoid activation to fuse the GRU hidden state and current state to create output coherent feature from GRU.

Hyperparameter Design

At the fine stage, the output truncation distance of our predicted TSDF value is set as 12 centimeters. We empirically set the optimizer as Adam without weight decay [60], with an initial learning rate of 0.001, which goes through 3 halves throughout the training. The first momentum and second momentum are set to 0.9 and 0.999, respectively. For key frame selection, following [16, 52], we set thresholds θ_{key}, t_{key} and fragment input number N_k as 15 degrees, 0.1 meters, and 9, respectively. A fraction of FPN [26] is adopted as the 2D backbone with its classifier as MNasNet [61]. MinkowskiEngine [62] is utilized as the sparse 3D tensor library.

Loss Design

Our model is trained in an end-to-end fashion except for the pre-trained 2D backbone. Since our target is to learn the 3D geometry and semantic segmentation of the surrounding scene given the posed images input, we regress the TSDF value with the mean absolute error (MAE) loss, classify the occupancy value with the binary cross-entropy (BCE) loss and the semantic labeling with cross-entropy (CE) loss as:

$$\begin{aligned}\mathcal{L}_{3D} = \sum_{s=2}^{4} \alpha_s \mathcal{L}_{\text{MAE}}(T_s, \hat{T}_s) + \lambda \alpha_s \mathcal{L}_{\text{BCE}}(O_s, \hat{O}_s) \\ + \beta_s \mathcal{L}_{\text{CE}}(S_s, \hat{S}_s) \ , \end{aligned} \tag{2.17}$$

where $\boldsymbol{T}$, $\boldsymbol{S}$, and $\boldsymbol{O}$ denote TSDF value, semantic labeling, and occupancy predictions. α_s, β_s, and λ are the weighting coefficients in different stages for TSDF volume, semantic volume, and positive weight for BCE loss, respectively. We set $\alpha = \{1, 0.8, 0.64\}$ while scaling down $\beta = \{0.1, 0.08, 0.064\}$ to balance the semantic volume learning with TSDF volume learning. γ and μ is set as 0.5 and 0.1 for each s for the same reason. By doing so, the learning process stays most sensitive and relevant to the supervisory signals in the coarse stage which has the greatest receptive fields, and less fluctuating as the 3D features become finer with the upsampling, after log-transforming the predicted and ground-truth TSDF value following [30].

To conduct cross-dimensional refinements, we regress the depth estimation with MAE loss and classify the 2D semantic segmentation with CE loss:

$$\begin{aligned}\mathcal{L}_{2D} =& \mathcal{L}_{\text{MAE}}(d_{init}, \hat{\boldsymbol{D}}_{init}) + \mathcal{L}_{\text{CE}}(\boldsymbol{S}_2^{2D}, \hat{\boldsymbol{S}}_2^{2D}) \\ &+ \sum_{s=2}^{4} \gamma_s \mathcal{L}_{\text{MAE}}(\boldsymbol{D}_s, \hat{\boldsymbol{D}}_s) \ , \end{aligned} \tag{2.18}$$

where $\boldsymbol{D}$ and γ_s denote depth and the weighting coefficient for depth estimation in different stages. We further wrap the losses into an overall loss $\mathcal{L} = \mathcal{L}_{3D} + \mu \mathcal{L}_{2D}$, where μ is the coefficient to balance the joint learning of 2D and 3D.

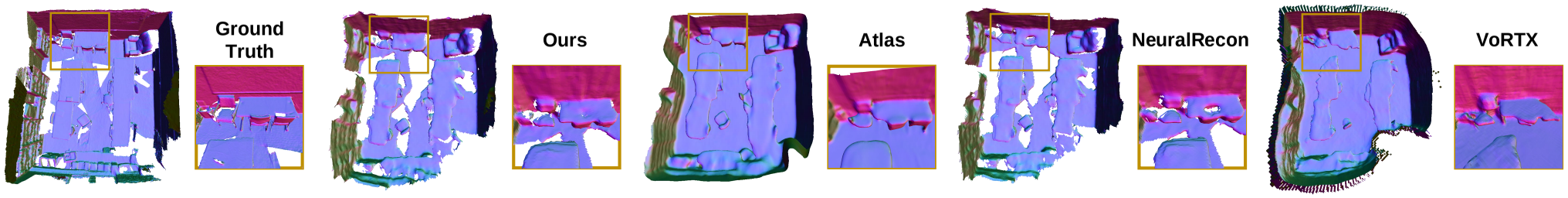


Figure 2.9: **Qualitative 3D reconstruction results on ScanNet.** Our method is capable of reconstructing consistent and detailed geometry which is neither overly smooth as the one from Atlas [30] nor eroded with holes as from NeuralRecon [16].

Table 2.1: Quantitative 3D reconstruction results on ScanNet

Method	Acc. ↓	Comp. ↓	Prec. ↑	Recall ↑	F-Score ↑
Atlas [30]	0.124	0.074	0.382	**0.711**	0.499
NeuralRecon [16]	0.073	0.106	0.450	0.609	0.516
3DVNet [33]	**0.051**	0.075	**0.715**	0.625	0.665
SimpleRecon [49]	0.061	**0.055**	0.686	0.658	**0.671**
VoRTX [31]	0.089	0.092	0.618	0.589	0.603
Ours	0.068	0.062	0.609	0.616	0.612

Table 2.2: Quantitative 3D voxel semantic segmentation and overall 3D perception results on ScanNet

Method	FPS ↑	KFPS ↑	FLOPF ↓	mIoU ↑	η_{3D} ↑
3DMV [36]	7.04	N/A	65.06G	44.2	N/A
BPNet [40]	4.46	N/A	141.06G	74.9	N/A
Atlas [30]	66.3	N/A	267.04G	34.0	11.25
NeuralRecon [16] + Semantics-Heads	**228**	**30.9**	**42.38G**	27.9	32.82
VoRTX [31] + Semantic-Heads	119	13.5	150.23G	13.2	9.47
Ours	158	21.4	90.62G	**39.1**	**37.81**

2.4 Experiments

2.4.1 Datasets and Metrics

We conduct the experiments on two indoor scene datasets, ScanNet (v2) [63] and SceneNN [64]. The model is trained on the ScanNet train set, tested and reported on the ScanNet test set and further verified on SceneNN data set. To quantify the 3D reconstruction and 3D semantic segmentation capability of our method, we use the standard metrics defined in Appendix 2.5, following [16, 30]. Completeness Distance (Comp.), Accuracy Distance (Acc.), Precision, Recall, and F-score, are used for 3D reconstruction, while mean Intersection over Union (mIoU) is used for 3D semantic segmentation.

To evaluate how much robustness a model can achieve while targeting 3D perception tasks in real time, we define the 3D perception efficiency metric $\eta_{3D} = \text{FPS} \times \text{mIoU} \times \text{F-score}$, since F-score is regarded as the most suitable 3D metric for evaluating 3D reconstruction quality by considering Precision and Recall at the same time [16, 30, 49]. It is noteworthy that for fairness across methods, FPS for processing speed is measured in the inference across all captured frames in a given video sequence rather than key frames only, since the input is the same for different methods regardless of their key frame selection scheme.

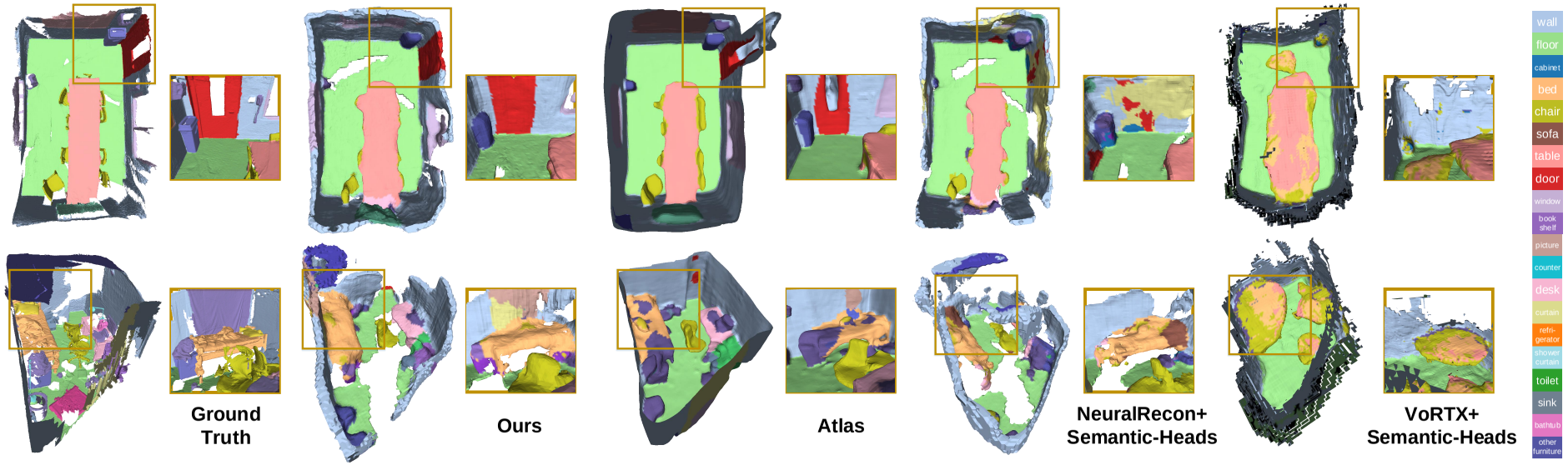


Figure 2.10: **Qualitative 3D semantic segmentation results on ScanNet.** Our method consistently outperforms baseline models and sometimes even surpasses the ground-truth labeling, e.g., in the bottom row, the photo-printed curtain above the bed is correctly recognized as "curtain" and "picture", whereas the ground truth mistakes it as "other furniture".

2.4.2 Evaluation Results and Discussion

3D Perception

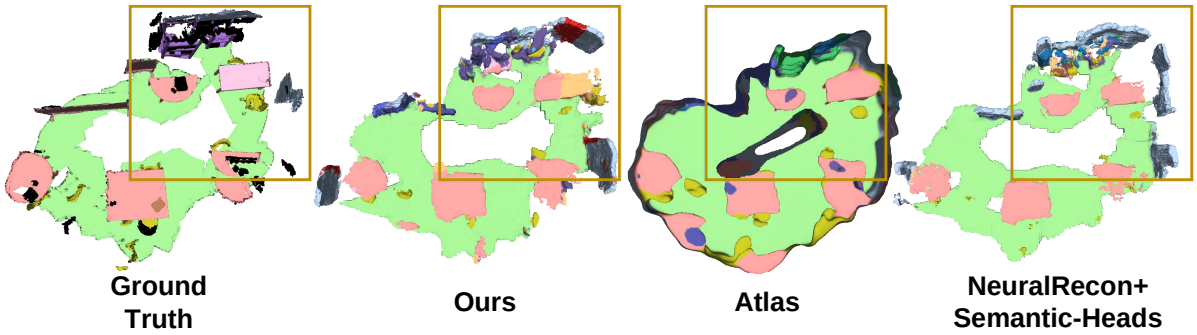
To evaluate the 3D perception capability, we mainly compare our methods against state-of-the-art works in two categories: volumetric 3D reconstruction and voxelized 3D semantic segmentation methods.

For 3D reconstruction capability, we compare our proposed method with the canonical volumetric methods [16, 30] and several state-of-the-art 3D reconstruction methods with posed images input [33, 49]. Fig. 2.9 demonstrates the superiority of our method in terms of 3D reconstruction by showing the 3D meshing results in normal mapping. Table 2.1 shows that our method outperforms two main baseline methods in terms of 3D meshing accuracy. Our method is superior to two main baselines, Atlas and NeuralRecon, and as competitive as other prior arts on 3D reconstruction. We further compare both state-of-the-art depth estimation methods and volumetric methods in depth metrics in the supplement to justify from the depth extraction perspective.

It is noteworthy that Atlas [30] compared their post-processed ground truth using TSDF fusion to create a new mesh from ground-truth depths, sometimes leading to implausible performance, e.g., on Recall. We avoid this issue by staying with the vanilla ground-truth meshes themselves for evaluation.

For 3D semantic segmentation quality, we compare Atlas, NeuralRecon with semantic heads, and VoRTX with semantic heads with our methods in Table 2.2. In the upper part, two representative state-of-the-art methods for semantic segmentation whose input requires either depth or 3D mesh, respectively. No key-frame selection and F-score are involved due to their input modality. In the lower part, RGB-input-only volumetric methods. Key-frame FPS (KFPS) is measured with the same selection scheme across all methods. FLOPF is measured with PyTorch operation counter across operations of neural network's learnable modules.

We augment three stages of MLP heads on top of the flattened 3D features to predict the semantic segmentation for both baselines. Due to its lack of 3D feature extraction, SimpleRecon, as one of the state-of-the-art baselines, is intrinsically incapable of following this modification for semantics as well as being combined with our proposed cross-dimensional refinement techniques. We evaluate mIoU by creating the confusion matrix as suggested in the official ScanNet [63] evaluation script. To achieve a better protocol for comparison, the ground-truth mesh needed to be directly used to generate point clouds



Method	Acc. ↓	Comp. ↓	FPS ↑	F-Score ↑	mIoU ↑	η_{3D} ↑
Atlas [30]	0.074	0.164	54.7	0.499	31.4	8.57
NeuralRecon [16] + Semantic-Heads	0.138	0.216	**178**	0.510	15.9	14.43
Ours	**0.068**	**0.143**	121	**0.611**	**36.7**	**27.09**

Figure 2.11: **Qualitative and quantitative 3D pereception results on SceneNN dataset.** Our method is proven to be generalized to SceneNN without pre-training on the SceneNN train set.

and conduct a point-wise comparison, rather than unnecessary post-processing. Table 2.2 shows that our method outperforms these two baselines.

Besides mIoU for semantic segmentation, we include FPS and η_{3D} for 3D perception efficiency in the comparison. We also include two state-of-the-art 3D semantic segmentation methods, 3DMV [36] and BP-Net [40]. It shows that our method can achieve mIoU results nearly comparable to 3DMV but with only RGB images as input. Overall, our method achieves the best 3D semantic segmentation performance and the highest 3D perception efficiency among all the volumetric methods. Fig. 2.10 and Fig. 2.11 illustrate the 3D semantic labeling results. We found that the semantic information generation on VoRTX is unsatisfying, mostly caused by its bias on geometric features brought by the projective occupancy mentioned in [31].

Efficiency

Since our main goal is to achieve real-time processing performance while solving 3D perception tasks, we compare the computational efficiency of our model against other RGB-input-only volumetric methods in Table 2.2. The 3D perception efficiency metric η_{3D} for several 3D semantic segmentation works are shown there. We employ FPS, which is commonly used to measure efficiency for 2D-input 3D perception methods [16, 30, 31], as a metric to bring out and emphasize the nature of real-time systems. We also include the floating-point operations per frame (FLOPF) to compare the learnable parameters' operations across different methods. The superiority in η_{3D} of our method manifests that it has better deployment potential for real-life 3D perception applications. From the human

Table 2.3: Ablation study

	GRU Input	Depth		Semantics		F-Score↑	mIoU↑	FPS ↑	η_{3D} ↑
		DE	AR	SE	PVR				
Neucon + Sem.	Geo.					0.516	27.9	228	32.82
(a)	Geo.	✓	✓	✓	✓	0.477	31.7	190	28.73
(b)	Geo.+ Sem.	✓		✓		0.479	27.1	**232**	30.12
(c)	Geo.+ Sem.	✓		✓	✓	0.482	34.5	169	28.10
(d)	Geo.+ Sem.	✓	✓	✓		0.556	26.8	226	33.68
(e)	Geo.+ Sem.	✓	✓	✓	✓	**0.612**	**39.1**	158	**37.81**

user's and robotic SLAM's points of view, our method greatly surpasses the threshold of being real-time, 90.17 FPS, as elaborated in Appendix 3.2. It shows that our method is more suitable for real-time industrial scenarios with input data from low-cost portable devices compared to baseline methods.

2.4.3 Ablation Study

To analyze the effectiveness of cross-dimensional refinement, we present 3D perception efficiency η_{3D} and its components of with different modifications in Table 2.3, where DE, AR, SE, and PVR denote depth estimation, anchored refinement, 2D semantics estimation, and point-to-vertex refinement, respectively. We assess our method by removing each of the proposed feature fusion techniques on ScanNet. In other experiments above, we adopt (e) as our method.

Binomial GRU Fusion

In (a), we remove the back-projected semantics input to GRU in the pipeline. Compared with (e), both F-score and mIoU of the removal degrade since no hidden semantic information from last FBV is fused with GRU anymore. Although FPS increases due to fewer computations, the efficiency η_{3D} is worse. Compared with *NeuralRecon + Semantic Heads*, the semantic loss in (a) is more converged by higher mIoU, but it cannot get the anchor occupancy learned well with a high F-score mostly because of the increasing recall but too low precision (high recall is better for having high mIoU). Therefore, we are motivated to push both 2D RGB and semantic features to GRU such that the GRU weights are more generalized.

Depth Refinement

In (c), we remove the depth anchor refinement in the pipeline. The loss in F-score and mIoU manifests that the geometric feature without depth anchor refinement becomes inferior, which means depth anchor refinement can improve 3D reconstruction performance.

Semantic Refinement

We validate the semantic refinement in the pipeline by removing this module and, as shown in (d). The mIoU drops due to the insufficient learning information from semantic heads only. This result demonstrates the effectiveness of our semantic refinement scheme based on pixel-to-vertex matching for improving 3D semantic segmentation performance. We also experiment with no refinements but depth and 2D semantics learning setup in (b), which gives the highest FPS but not satisfying 3D perception performance.

2.5 3D Perception Evaluation Metrics

Table 2.4: Definitions of metrics

Metrics	
L1	$\text{mean}_{t^*<1}\lvert t - t^*\rvert$
Acc.	$\text{mean}_{p\in P}(\min_{p^*\in P^*} \lvert\lvert p - p^*\rvert\rvert)$
Comp.	$\text{mean}_{p^*\in P^*}(\min_{p\in P} \lvert\lvert p - p^*\rvert\rvert)$
Prec.	$\text{mean}_{p\in P}(\min_{p^*\in P^*} \lvert\lvert p - p^*\rvert\rvert < .05)$
Recall	$\text{mean}_{p^*\in P^*}(\min_{p\in P} \lvert\lvert p - p^*\rvert\rvert < .05)$
F-Score	$\frac{2\times\text{Prec.}\times\text{Recall}}{\text{Prec.}+\text{Recall}}$
mIoU	$\frac{1}{C_{sem}} \sum_{i=0}^{C_{sem}} \frac{v_{ii}}{\sum_{j=0}^{C_{sem}} v_{ij} + \sum_{j=0}^{C_{sem}} v_{ji} - v_{ii}}$

Annotation: n is the number of pixels with both valid ground truth and predictions. d and d^* are the predicted and ground truth depths. t and t^* are the predicted and ground truth TSDFs while p and p^* are the predicted and ground truth point clouds. v_{ij} is the number of vertices of category i out of C_{sem} categories that are predicted to be of category j.

2.6 Conclusion

In this section, we proposed a lightweight volumetric method, *CDRNet*, that leverages the 2D latent information about depths and semantics as the feature refinement to handle

3D reconstruction and semantic segmentation tasks effectively. To answer the research questions about the **mimicking** and **surpassing** of human visual perception raised at the very beginning of this section, we illustrated that our proposed method for 3D visual perception could achieve good performance and relatively high efficiency, which indicates that it is feasible for machines to not only mimic humans' 3D perception system with solely visual input, but also surpass it with solely monocular input thanks to our recurrent memory modules.

We further demonstrated that our method has real-time 3D perception capabilities, and justified the significance of utilizing 2D prior knowledge when solving 3D perception tasks. Extensive experiments on various datasets justify the 3D perception performance improvement of our method compared to prior arts. From the application point of view, the scalability of *CDRNet* supports the notion that 2D priors should not be disregarded in 3D perception tasks and opens up new avenues for achieving real-time 3D perception using input data from readily accessible portable devices such as smartphones and tablets. It also paves the way for developing solutions that are capable of prevailing over human visual perception in the future.

3 Real-Time 3D Perception with Data Streaming

"It would be very hard to picture any next generation intelligent robots without any on-board visual sensors. The level of intelligence of the future robots will be very much determined by how well the on-board computer processes information collected from the visual sensors."
— Yi Ma, *A Differential Geometric Approach to Computer Vision*

In this chapter, we develop a real-time 3D perception system based on the CDRNet algorithm proposed in 2.3. Thanks to MAP optimization, we can achieve optimial metric-semantic reconstruction results. However, it is noteworthy that our method is different with bayesian inference in many aspects which happens to be highly relying on the prior knowledge as well.

3.1 Related Work

The prosperity of deep learning hardwares enables both inference and training at the edge [27, 28], thus it consolidates the foundation to deploy more and more learning-based 3D perception techniques in real time. KinectFusion [8] first brought in the concept of handling 3D reconstruction tasks in real time with commodity RGB-D sensors. Han et al. [13] presented a real-time 3D meshing and semantic labeling system similar to our work, however, depth measurements from RGB-D sensors are required as input in their work. Pham et al. [65] built up 3D meshes with voxel hashing, and then fuse the initial semantic labeling with super-voxel clustering and a high-order conditional random field (CRF) to improve labeling coherence. Menini et al. [41] extended RoutedFusion [21]

by merging semantic estimation in its TSDF extraction scheme for each incoming depth-semantics pair. NeuralRecon [16] adopted sparse 3D convolutions and the gated recurrent unit (GRU) to achieve a real-time 3D reconstruction on cellphones, without the capability of semantic deduction. For depth estimation and semantic segmentation, there are also works achieving real-time processing capability [38, 39, 52, 65].

In addition to the SDF based and voxel hashing methods that mentioned above, NeRF [66] is one of most popular paradigms nowadays that takes position and polar rotation as a 5D input to construct an MLP for 3D reconstruction. The output of NeRF is a 4D vector that represents the three channel color in RGB and one channel transparency for the input pose. With such a structure, NeRF is originally designed for novel view synthesis but can also be adpated to 3D reconstruction and perception by feeding the 3D space's grid input and some adaptation, respectively. NeRF achieves impressive fidelity after a lot of iterations. However, due to the lack of representation capability, the 3D mesh/point clouds generated from NeRF tends to have foggy artifacts given a relatively large number of computations. To this end, there are research works such as NeuS [67] that explored to combine the advantages of both by using the same 5D input to infer SDF in an end-to-end manner. Predictably, the even larger MLP network in Neus is even slower and cannot be acceptable for the real-time embodied AI application.

3.2 Requirements of Being Real-Time

With a handheld monocular-camera cellphone as the input source, we assume the average human pace as 1.4 m/s. Given the length of each fragment is 96 voxels in CDRNet and voxel size is 4 cm, we can calculate the maximum update time for CDRNet one fragment is $\boldsymbol{T}_{\text{update}} = \frac{96\times0.04\,\text{m}}{1.4\,\text{m/s}} = 2.743\,\text{s}$. As tested out in multiple trials of our experiments, the total latency under Wi-Fi data transmission and RTMP streaming on the server is around 2 s. Thus, the processed time that can be accepted for one fragment computing of CDRNet is $\boldsymbol{T}_{\text{proc}} = \boldsymbol{T}_{\text{update}} - 2\,\text{s} = 0.743\,\text{s}$. On average, each fragment entertains 67 frames (as $\boldsymbol{N}_k = 9$ with key-frame selection), which makes the FPS of the according $\boldsymbol{T}_{\text{proc}}$ as $\text{FPS}_{\text{real-time}} = \frac{67\text{Frames}}{0.743\,\text{s}} = 90.17$.

In summary, the 3D perception system must meet the following specifications to provide real-time 3D metric-semantic reconstruction mapping service to the robotic agent.

1. Runtime speed over 90.17 FPS, which directly affects the latency to be less than 0.743 second.

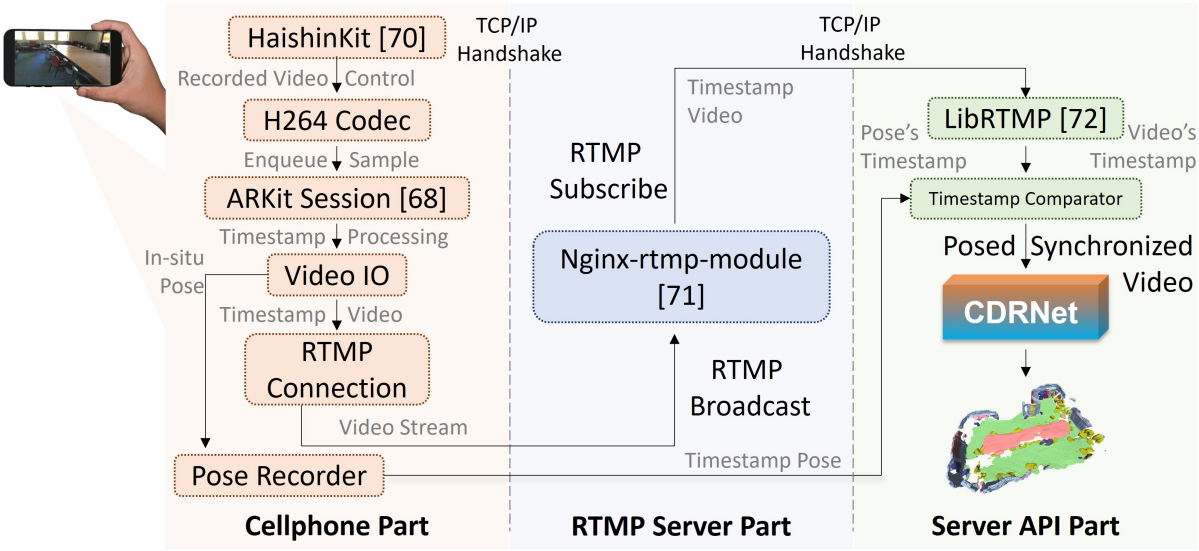


Figure 3.1: **The pipeline of real-time 3D perception with CDRNet.** RGB frames and camera transform matrices are extracted utilizing the camera and IMU on the cellphone via the ARKit interfaces and RTMP server, respectively.

2. GRAM size needs to be reasonably small, perferably lower than 1GB for a typical 200 square feet conference room.

3.3 Real-Time 3D Perception with Data Streaming

In Fig. 3.1, we present the progressive 3D perception system that is capable of real-time interaction with a monocular camera on the cellphone. The sender-to-receiver implementation flow is listed out modularly. We chose ARKit [68] on the iOS platform to synchronize the RGB, camera intrinsics, and camera pose recordings, meanwhile, there is also a counterpart in the Android platform named ARCore [69]. After the input fragment is recorded on the cellphone, each fragment input is passed to CDRNet thanks to the RTMP server, and the semantic mesh inference within the current FBV can be achieved readily. They are passed to the CDRNet model for real-time 3D perception.

3.3.1 Video and Pose Recording on the Cellphone

On the cellphone side, we are using an iPhone 11 with an RGB camera only for data capturing. HaishinKit for iOS [70] is used for recorded video control on ARKit [68] session to record the repective camera pose and intrinsics for frames.

3.3.2 RTMP for Video Streaming

Real-Time Messaging Protocol (RTMP), originally developed by Adobe, is one of the live streaming protocols that provides TCP-based video streaming services between devices to ensure persistent connections and low-latency communication. We adopt the one that is provided with the front-end proxy and HTTP server, Nginx-RTMP [71], which is hosted as a docker image by a lightweight open-source web proxy server, Nginx.

3.3.3 Incremental Data Receiving and Synchronization

To incrementally takes data, we use Librtmp [72] on the server side to subscribe to the streamed video and direct subscription on a pose recorder while the synchronization is done through timestamp comparison. The output synchronized posed video will be used for the incremental 3D perception.

Therefore, CDRNet can be regarded as a real-time method by achieving 158 FPS as shown in Table 2.2, whereas the other baseline, Atlas, fails to achieve $\text{FPS}_{\text{real-time}}$ to be in real time.

3.4 Differentiation between MAP Optimization in our CDRNet and Gaussian Process

As mentioned in Sec. 2.3.2, we do MAP optimization during the training time in CDRNet. Therefore as shown in Fig. 2.4, we are able to achieve optimial learning results across all θ. Such a resulting model is used in the real-time 3D perception system for the best perceptual performance and efficiency.

However, in the Gaussian process (GP) regression, the bayesian inference needs to be done at the runtime, i.e. for each inference batch the prior will be calculated and adopted additionally. Together with MAP inference, these methods are typically used in robotic perception, and should be differentiated with our proposed MAP-optimization-based approach.

GP regression assumes that the estimated quantity is in a gaussian distribution with regard to the input variables, or the embeddings that extracted from the input by learnable networks. The intuition of GP in this case is to ultize the prior knowledge which is normally Gaussian during the estimation, but from a completely different perspective.

Normally, a physical similarity is involved. Here the covariance of the GP regression of z_j is set as a kernel function that measures the similarity between to input frame namely $\kappa(\boldsymbol{M}(t), \boldsymbol{M}(t'))$, as shown in Eq. 3.1. A typical GP regression with a encoder-decoder structure works as follows.

$$z_j(t) \sim \mathcal{GP}(0, \kappa(\boldsymbol{M}(t), \boldsymbol{M}(t'))) \ , \tag{3.1}$$

$$z_j(t_i) = y_{j,i} + \epsilon_{j,i} \ \ , \ \ \epsilon_{j,i} \sim N(0, \sigma^2) \ . \tag{3.2}$$

Consider posed depth maps where the depths are either measured from LiDAR sensor [43], or inferred from images multi-view stereo with a cost volume [73] as the input to the GP regression. At the time frame t, the encoder takes the depth $\boldsymbol{D}_t$ extract a learned intermediate representation y_t. The pose similarity between two frames is encoded in a kenerl function $\kappa(\boldsymbol{M}(t), \boldsymbol{M}(t'))$, which will be used in the computation of y_t to incorporate pose prior knowledge. Then, the GP priors respective embedding z_t is constrcuted with imposed Gaussian noise on y_t as in Eq. 3.2. Finally, the decoder takes z_t to output the estimated quantity, which is SDF in this case. Compared with Sec. 2.3.2, limitation of doing GP regression is two-fold. Firstly, it lifts a huge burden from both computation and memory perspectives. Secondly, the Gaussian prior is random during the inference, which degrades the interpretability and reproductivity of the estimation.

4 Edge Acceleration for Convolutional Neural Nets Training

"The end of Dennard scaling meant architects had to find more efficient ways to exploit parallelism."
— John L. Hennessy and David A. Patterson, *A New Golden Age for Computer Architecture*

This chapter covers AI hardware research dedicated to performance improvement despite the diminishing technology gains. A software-hardware co-optimization mindset is adopted for training neural networks at the edge.

4.1 Introduction

Latency. Privacy. Trust. These are the three primary concerns for designers of internet-of-things (IoT) edge devices. To achieve low-latency, privacy-ensured, and trustworthy IoT human-machine interaction, current edge devices need to be sufficiently intelligent in handling the ubiquitous machine learning and deep learning tasks, and must perform significant amounts of in-situ processing on the spot [74–78]. In deep learning, deep convolutional neural networks (DCNNs), as the representative models, require processing a huge amount of feature data for model training.

To deal with the enormous training data, the cloud server conventionally centralizes the data from edge devices for large-scale deep learning. However, there are limitations to centralizing the data for model training. Not all edge applications are Wi-Fi enabled or can rely on consistent communications or charging. Such communication bandwidth and energy constraints for ad-hoc neural network applications make it hard for edge devices to

upload data, especially during the network training. People are also becoming more and more conservative when it comes to sharing their private data. For instance, to address privacy concerns, facial recognition data cannot be uploaded to the cloud for training purposes. These issues have given rise to federated learning [78], which investigates collaborative model training and inference attacks and constructs robust DCNN models. In federated learning, rather than the local data, the client nodes, such as edge devices, always send the updated models or gradients to the central server, which requires them to support re-training. However, most federated learning studies [78–80] primarily work on the encryption and model sharing strategy. Hence, the lack of local model re-training capability for the underlying hardware remains an issue.

Unfortunately, solving the re-training issue is challenging. With current model training approaches, both the throughput and the power offered by edge devices are not sufficient for the federated learning scenario, due to its power constraints. Edge devices and systems-on-chips (SoCs) for IoT scenarios are normally portable, with power less than 10 W, as shown in Fig. 4.1.

What is worse, as the computational complexity of the myriad DCNNs is increasing, the demand for higher throughput is rising drastically. With the slow-down in Moore's Law, it has become harder to meet this requirement solely depending on technology scaling. As shown more concretely in Fig. 1, the computational capability (throughput) for edge computing can no longer benefit from technology scaling, especially with the 10-Watt constraint. Hence, the pressure on architects and circuit designers to improve energy efficiency with elaborate designs of specific accelerator architectures has increased. One of the significant factors limiting the energy efficiency of accelerators is the excessive external memory access. To include many epochs until convergence, the model training process of DCNNs consumes a great amount of energy by accessing off-chip DRAM for each training sample. M. Horowitz [81] shows that the energy consumption of basic arithmetic and memory operations in a processor of the 45nm CMOS process is dominated by the DRAM access, which is more than 200x larger than the average of other operations. In this case, the power consumption solely brought by the DRAM access is always beyond the power envelope of typical edge devices such as mobile phones, even in the inference process. This poses great challenges for the edge devices in training tasks.

With the considerations of reducing the DRAM energy and utilizing sparsity in mind, the aim of this work is to design an efficient yet effective DCNN training algorithm that leverages the redundancy of conventional model training such that the energy efficiency

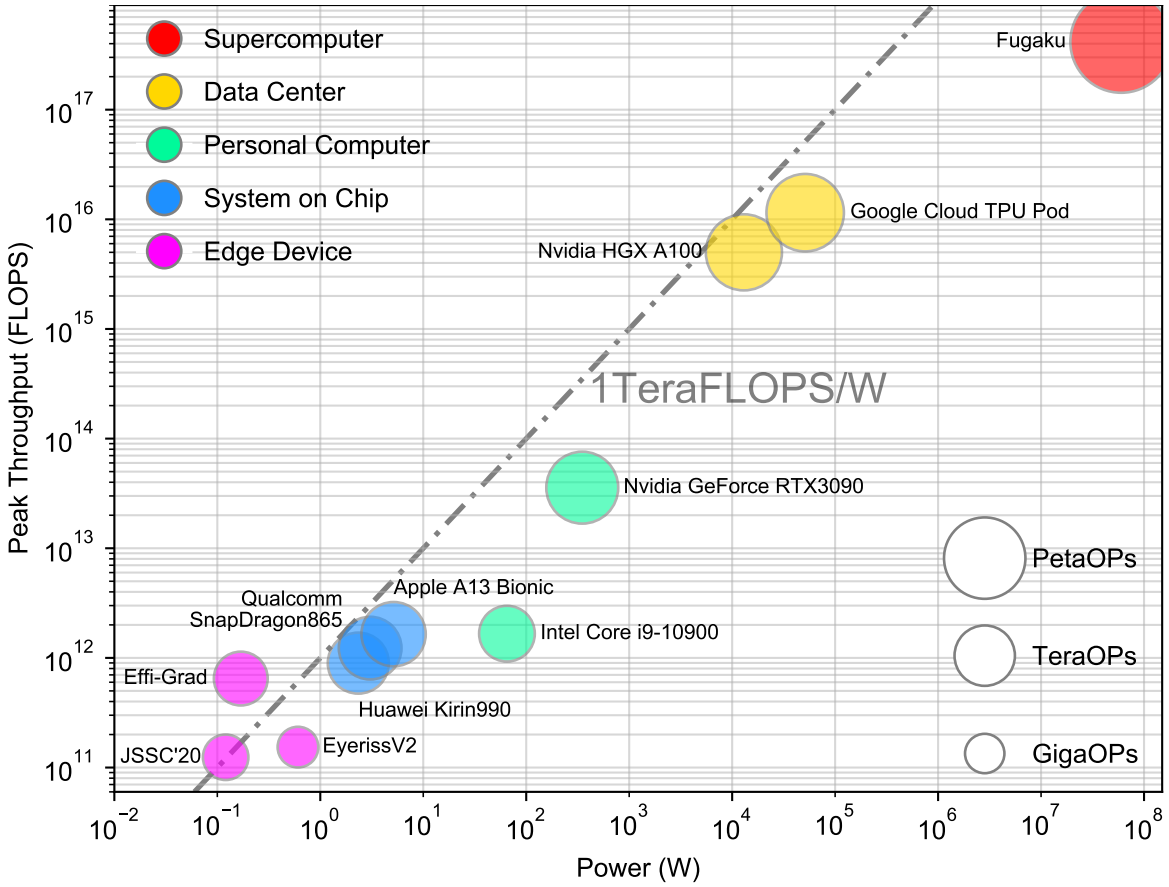


Figure 4.1: **Throughput vs. power for the modern computing landscapes.** The size of each representative device indicates the operation counts for their respective typical application in Table 1.1.

of the accelerators during training is improved. This paper proposes a novel algorithm-hardware co-design approach, *Efficient Training DCNNs with Gradient Optimizations*, dubbed Efficient-Grad. This approach can reduce the DRAM access while maintaining high throughput by utilizing sign-symmetric feedback. As we discuss in more detail in Sec.4.3, the symmetric modulator signal used in the conventional back-propagation-based training algorithm is replaced by sign-symmetric feedback, for both convolutional and fully-connected (FC) layers. To eliminate the overhead brought by minor gradients calculation in the backward phase while preserving the original validation accuracy, the error gradients generated by the sign-symmetric feedback are further pruned in a stochastic fashion. As an expansion of our prior work [82], this work has three main contributions, as follows:

- We propose an effective variant of back propagation (BP) for DCNN training, called Efficient-Grad. It imposes sign-symmetric fixed feedback as the modulator signals for error gradients and prunes the error gradients with a stochastic approach. The

learning capability is maintained by reaching low angles between the modulator signals prescribed by itself and BP.

- We design and implement a data reuse architecture in a hybrid-dataflow manner to fully utilize the algorithmic superiority of Efficient-Grad. This architecture exploits the gradient sparsity and memory access reduction brought by the algorithm, while maximizing the data reuse. By eliminating the transposed weight matrix fetching/storing and minor gradients being involved in the backward phase, the energy efficiency is dramatically increased.
- We design a cycle-accurate co-simulation platform based on two open-source tools for a fine-grained modeling of the overall energy cost, including external DRAM consumption.

The remainder of the article is organized as follows. Sec. 4.2 starts with the introduction of a vanilla BP algorithm without any optimization. Sec. 4.3 provides an overview of current efforts on both the algorithm and hardware sides to achieve edge training. Sec. 4.4 describes Efficient-Grad, the gradient-optimized training algorithm dedicated to edge devices. Sec. 4.5 introduces the supporting hardware accelerator designed specifically for Efficient-Grad. Sec. 4.6 shows the experimental setup and analysis, and Sec. 4.7 presents the evaluation results. At the end, Sec. 4.8 concludes the article.

4.2 Preliminaries

A typical training algorithm for DCNNs is shown as Algorithm 1 and illustrated in Fig. 4.2. It contains four phases: the **Forward Phase**, where the model is imposed on the input to compute activations and get inference results, the **Backward Phase**, where error gradients are computed using the next layer's information, the **Weight Gradient Phase**, where the weight gradients are updated relying on the error gradients, and the **Weight Update Phase**, where the weights in the model are updated by gradient-descent-based optimizers. In Fig. 4.2, an example of the computations of the first three phases (but not the weight update phase) is illustrated, as the first three phases dominate in terms of the amount of computation in the mini-batch stochastic gradient descent (SGD) setup. For the forward phase, an RGB image is used to convolve with four different convolutional filters, whose number of channels is three. Subsequently, each filter produces an output feature map, constructing a four-channel output feature map as a_{l+1}. For the backward phase, the error gradients in layer $l+1$ will be used to convolve with 180° rotated filters

and generate the error gradients of layer l whose channels are matched with the rotated filters individually. For the weight gradient phase, the weight gradient for each channel in each filter is simply extracted from the map-wise convolution between feature map a_l and next layer error gradient δ_{l+1}.

BP [83] and SGD [84] are the canonical algorithms used for DCNN training. They remain powerful and effective, and, are used in various current artificial intelligence (AI) systems. Generally, the BP algorithm is an efficient use of the chain rule for generating gradients, and the SGD algorithm takes the average of the gradients of a mini-batch input to update the weights. The conventional method of performing BP is illustrated in Fig. 4.2.

Algorithm 1: A Vanilla BP Convolutional Neural Network Training

Input: $[Img_1, Img_2, ..., Img_N]$: Input batch with the size of N images, $[W_1, W_2, ..., W_L]$: L layers of trainable weights
Output: Trained network for inference
/* Phase 1: Forward */
for $l \leftarrow 0$ **to** $L-1$ **do**
  $a_{l+1} = \sigma(W_{l+1} * a_l)$;
  if $l = L-1$ **then**
    $Loss = C(a_{l+1}, y)$
  end
end
/* Phase 2: Backward */
for $l \leftarrow L$ **to** 1 **do**
  if $l = L$ **then**
    $e = \frac{\partial Loss}{\partial a_l} = (a_l - y) \odot \sigma'(a_l)$
  else
    $\delta_l = W_{l+1}^T * \delta_{l+1} \odot \sigma'(a_l)$
  end
end
/* Phase 3: Weight Gradient and Update */
for $l \leftarrow L$ **to** 1 **do**
  $\Delta W_{l+1} = \frac{\partial Loss}{\partial W_{l+1}} = a_l * \delta_{l+1}$;
  $W_{l+1} = SGD(W_{l+1}, \Delta W_{l+1}, lr = \alpha, momentum = \mu)$
end

Algorithm 1 describes the procedure of a vanilla BP algorithm for DCNN training. A given learning rate α will be used as the coefficient of the weight gradients. Additionally, the previous updated value is also taken into account by a factor of the momentum μ to help accelerate SGD in the target direction and to mitigate oscillations. N denotes the batch size of each training iteration. Practically, we will neither take the whole training

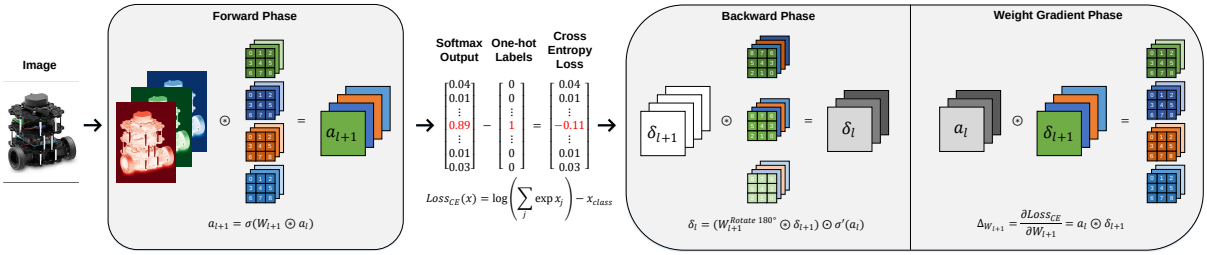


Figure 4.2: **Demonstration of the vanilla back propagation for DCNN training.**

data set as batch gradient descent (BGD) nor a single training sample for every weight update. Consequently, mini-batch SGD, which updates parameters by randomly selecting a mini-batch $\boldsymbol{N}$ of training samples from the training set, is utilized. It better helps the optimization to escape out of the local minimum compared to BGD.

4.3 Background and Motivation

Before we explain the details of Efficient-Grad, firstly, a comprehensive survey of recent efforts in network training paradigms will be given. These algorithms explore the possibility of realizing training at the edge. Next, we will analyze some state-of-the-art hardware accelerators for DCNN training with their pros and cons. Finally, we will introduce some common yet effective model compression methods for edge training.

4.3.1 Beyond BP: Modern NN Training

The biological implausibility of the conventional machine learning algorithm, namely, BP has always inspired researchers to explore the essence of machine learning. In 2016, Lillicrap et al. [85] at Google Brain proposed feedback alignment (FA), in which the modulator signal during the backward phase is replaced by a uniform random feedback matrix, rather than the transpose of the weight matrix as in most vanilla BP algorithms. A similar approach, called random backpropagation [86] also emerged. The term "alignment" has a two-fold meaning in this context. Firstly, it indicates the fact that the weights in the forward phase and backward phase maintain a soft alignment to obtain a learning capability, rather than a precise symmetric connectivity. Secondly, it means that the angle between the modulator signals from BP and this method drops below 90°, which reflects the similarity of the modulator signals and the learning capability to some extent.

A major drawback of FA that has stopped it from flourishing is its poor performance

on convolutional networks, which are still the cornerstone of most computer vision applications. In [85], the authors explain why FA fails on convolutional networks. The rectified linear unit (ReLU), which most DCNNs employ as the activation layer, tends to kill the negative side gradient when the asymmetric feedback weight is used for learning. This effect is especially aggravated by the powerful regularization property inherent in the convolutional layers.

Despite its limitations, many variants of FA have been proposed to explore better artificial neural network training. Direct feedback alignment (DFA) [87], which directly propagates the output error back to each layer, has proved to be as effective as BP for FC layers. Based on DFA, direct random target projection (DRTP) [88] further liberates the weight transport and update locking issues during BP by imposing direct target projection, which enables parallel updating of weights across layers. However, it only works well for spike-based neural network learning because of the limited learning capability. The trade-off between learning capability and energy efficiency in the DRTP's spiking neural network accelerator [89] makes it ill-suited even for perceptron-based neural networks, let alone the complex DCNN workloads that we are aiming at in this work.

DFA is also not suitable for DCNN training due to the lack of alignment [90]. Efforts have been made to address the update locking issue in BP, but they lack generality both on complex datasets or network types other than FC layers [91, 92], and they all sacrifice the learning capability to unlock the locking issue in the backward phase. In other words, these approaches are not suitable for DCNN training due to their defects in validation accuracy. Liberating such constraints on computation-intensive DCNN accelerators is impractical since the compute modules inside deal with data layer-wisely. [93] adopts a binary approach to FA (BFA), but it simply abandons FA learning for convolutional layers, only tuning on FC layers with FA for a predetermined object tracking purposes.

Research efforts towards efficient DCNN training, which strive to cut down computations and data movements in BP as much as possible, are now being made. E^2-Train [94] accomplishes data saving by stochastically dropping mini-batches of input data. To reduce redundant computations, it utilizes recurrent neural network gating cells for critical layers selection and an empirical gradient selector for gradient sign prediction. On top of E^2-Train, [95] proposes to drop trivial input data with a self-supervised importance metric, while some channels of error gradients are pruned with yet another importance score. Meanwhile, [75] further explores how BP-level optimal checkpointing improves the efficiency, thus enabling training at the edge. These works reveal the huge potential for

accelerating DCNN training through improving the BP algorithm and implementation efficiency.

4.3.2 DCNN Inference and Training Accelerators

To support the rapid growth in the use of DCNNs in deep learning, the demand for dedicated DCNN accelerators to improve throughput with limited power constraints has risen drastically. In particular, most edge devices need to conform to power constraints of less than 10 W, as shown in Fig. 4.1. Accordingly, to handle real visual applications using DCNNs, the energy efficiency (slope in Fig. 4.1) of the edge accelerators for the DCNNs needs to be sufficiently high. Eyeriss [77] and DianNao [96] are two early approaches aimed at lowering the required power footprint for processing DCNN workloads by a data reuse mechanism and parallel operations mapping. Subsequently, various hardware design efforts have also been presented, such as DaDianNao [97], with a multi-core system to hold up operands in eDRAM, and ShiDianNao [98], which moves the data sensor closer to the accelerator itself. Other works, such as [99], have achieved high energy efficiency for DCNN inference by adopting a mixed precision architecture.

Since DCNNs began to prevail in the era of AI, attempts to deploy DCNN training at the edge have never stopped. The limitation of the aforementioned works, however, is that none are well-designed for training. [100] is the groundbreaking work which firstly introduced DCNN training on an SoC, despite its inefficiency in terms of energy. Because the sparsity of gradients has made a more profound impact than the sparsity of inference operands, since the computations for one sample of DCNN training is over two times larger than the inference, works like [101] have further adapted DianNao to support DCNN training, while utilizing gradient sparsity by skipping or selectively loading gradients. Another work, [102], modifies the architecture of Eyeriss and utilizes the sparsity with a stochastic pruning of gradients to enhance the throughput. The training processor LNPU [103] employs a mixed-precision training scheme to highly improve the energy efficiency. However, it still suffers from external DRAM energy consumption by running the vanilla BP. To solve this issue, previous works [89, 93, 104] have adopted variants of FA and achieved higher efficiency. However, they focus only on FC layer training, not on convolutional layers.

4.3.3 Motivation

The training-specific hardware implementation for the vanilla BP cannot provide satisfactory energy efficiency due to the prohibitive cost of a large number of parameters and operands reading from the off-chip DRAM memory. From the BP algorithmic perspective, this lack of energy efficiency arises for two reasons.

Firstly, popular mobile DCNN models, such as ResNet [105] and EfficientNet [106], possess parameters larger than 10MB, whereas the on-chip SRAM in edge devices is normally limited to less than 1MB due to the cost and power concern of the SRAM technology. Hence, due to the limited capacity of the on-chip SRAM, parameters need to be frequently loaded in from the off-chip DRAM. In the backward phase, as shown in Fig. 4.2, the weight kernel matrix is stored in DRAM by a row-wise and channel-wise sequence. Such an arrangement is for the direct usage of data accessing for both the forward and weight gradient phases. However, during the backward phase, the weight kernel matrix needs to be read out in a transposed fashion, which leads to a long processing latency. Normally, for this end, the whole bank of the target weights in a particular rank will be read out, causing high DRAM access overhead. To deal with this issue, [107] utilizes a customized SRAM cell with an additional transistor for bitline and wordline connection, while [108] rearranges the storage scheme for SRAM. Both of these works reduce latency by increasing power and area consumption, but do not deal with the dominant part of the energy consumption, namely, the DRAM access overhead.

Secondly, the error gradient in the backward phase is also stored off-chip due to its size. Nevertheless, reading all the error gradients on-chip for computation is not necessary and causes insufficient throughput, along with some redundant data movement. In Sec. 4.4.1, we further explore how to successfully optimize the gradients to suppress this issue.

4.4 Algorithm

Given the prohibitive energy consumption of prior arts, we propose the Efficient-Grad algorithm with a supporting accelerator dedicated to energy-efficient yet effective DCNN training. This section first analyzes the FA algorithm specifically for DCNN training. Then, we introduce our proposed algorithm, Efficient-Grad. The architecture constraints of Efficient-Grad are described in detail as well.

4.4.1 Efficient-Grad: Efficient Training DCNNs with Gradient-Pruned Sign-Symmetric Feedback Alignment

Considering the original FA, the modulator signals in the backward phase of Algorithm 1 are a random feedback matrix $\boldsymbol{B}$, which eliminates the usage of a transposed or 180-degree-rotated weight matrix $\boldsymbol{W}^T$, as shown in Fig. 4.2. Consequently, the error gradient of the layer l, δ_l, is retrieved by the layer $l+1$ as

$$\delta_l = \boldsymbol{B}_{l+1} * \delta_{l+1} \odot \sigma'(a_l). \tag{4.1}$$

Nevertheless, as mentioned in [85], the limitation of FA is that the fixed feedback cannot be directly imposed on convolutional layers. This is because all the neurons within a convolutional layer share precisely the same receptive field, and such weight sharing aggravates the regularization effect of FA and leads to over-regularization. From our experiments, we observe that in the early training stages, the regularization effect of FA will often improperly impel the activation into the negative region, which will lead to dead neurons if a ReLU is applied, and these neurons will be irreversibly eliminated. We refer to this phenomenon as the "early-killed-neuron (EKN) effect" of applying a ReLU with FA. Beyond that, we make a compromise and replace the activation function, σ, as in [85], with the hyperbolic tangent (Tanh). Fig. 4.3(a) shows the performances of AlexNet under different training setups, where either FA or a hybrid method which has BP on convolutional layers and FA on FC layers is utilized. Both FA training and hybrid training with a ReLU (FA_ReLU and Hybrid_ReLU), suffer from the EKN effect brought by the ReLU on the FC layers as long as there is asymmetric feedback (modulator signal) under the FA training scheme. Even if the hybrid method has only FC layers adopting FA, the training diverges at the early stage. As shown in Fig. 4.3(b), the same training setups are also tested on MobileNetV2, except for the hybrid case of ReLU on convolutional layers and Tanh on FC layers due to the lack of FC activation in the network structure of MobileNetV2. The EKN effect is also observed for MobileNetV2, indicating that with the deeper and more sophisticated structures, networks whose convolutional layers are adopting FA suffer from a varying degree of the EKN effect.

To address this limitation of the EKN effect of FA for DCNNs, we mitigate the over-regularization issue by assigning the fixed random feedback to the symmetric signs of its corresponding weights. Moreover, to restore the improperly killed neurons in the hidden

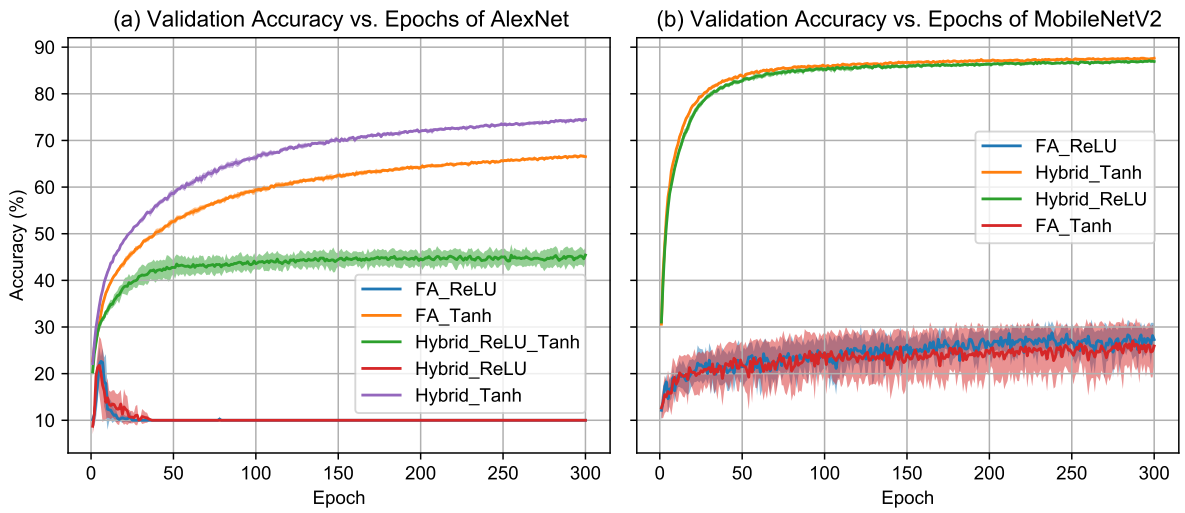


Figure 4.3: **The accuracy vs. epochs for varying FA-based training schemes.** In (a) AlexNet and (b) MobileNetV2. Note that for the cases where FA is imposed on all convolutional layers (i.e., FA and Hybrid in the legend) and ReLU together, the model loses its generality to the testing data, at the early stage of learning, around 30 epochs. The error bars are one standard deviation for 10 trials around the time-averaged mean. The strong regularization brought by the convolutional layers, ReLU and FA-based learning together tends to cancel out the neurons.

layers, we append batch normalization (BN) [109] layers in between, wherever the neurons tend to be killed. Consequently, the sign-symmetric feedback in the backward phase of Algorithm 1 can be obtained as:

$$\delta_l = sign(W_{l+1}) \odot |B_{l+1}| * \delta_{l+1} \odot \sigma'(a_l). \tag{4.2}$$

Additionally, the resulting error gradients in Eq. (4.2) turn out to be small in magnitude. We further observe that the error gradients of adopting sign-symmetric FA in the backward phase of Algorithm 1 is distributed in a long-tailed normal distribution. This means that the computation brought by Eq. (4.2) can be bypassed as long as its expectation remains unchanged afterwards. Inspired by [102], we propose a stochastic gradient pruning algorithm based on Eq. (4.2) to reduce these redundant gradient computations. The main idea is to prune the error gradients prescribed by the sign-symmetric feedback while maintaining their mathematical expectation.

To keep the expectation unchanged, rather than clamping all the pruned gradients into zero, it is natural to compensate the values of pruned gradients back to the pruned threshold. Consider the mathematical expectation of the gradient $E(\delta_l)$ and that of the

pruned gradient $E(\hat{\delta}_l)$:

$$E(\delta_l) = \delta_{l_K} + \delta_{l_J} = \frac{1}{n}\sum_{i=1}^{n}\delta_{l_i}, \tag{4.3}$$

$$\begin{aligned} E(\hat{\delta}_l) &= \frac{1}{n}(\sum_{k=1}^{K}\hat{\delta_{l_k}} + \sum_{j=1}^{J}\hat{\delta_{l_j}}) \\ &= \frac{1}{n}(\sum_{k=1}^{K}\delta_{l_k} + P_{ceil}\cdot\tau + (1-P_{ceil})\cdot 0) \\ &= \frac{1}{n}(\sum_{k=1}^{K}\delta_{l_k} + \frac{\delta_{l_J}}{\tau}\cdot\tau) = E(\delta_l), \end{aligned} \tag{4.4}$$

where δ_{l_K} is the subset of δ_l which satisfies $\delta_{l_K} > \tau$, and for δ_{l_J}, vice versa. Eq. (4.4) requires the rounding-up ratio P_{ceil} to be δ_{l_J}/τ to keep the expectation unchanged, where P_{ceil} is achieved by comparing δ_{l_i} with a uniformly distributed variable r as follows:

$$\hat{\delta_{l_i}} = \begin{cases} \delta_{l_i} & \text{if } |\delta_{l_i}| > \tau, \\ \tau \odot sign(\delta_{l_i}) & \text{if } \tau \geq |\delta_{l_i}| \geq r\tau \ , r \in [0,1], \\ 0 & \text{otherwise,} \end{cases} \tag{4.5}$$

where r is a uniform random number ranging from 0 to 1. Note that Eq. (4.5) is applied on top of Eq. (4.2) and we need to ensure that the angles of the error gradients with Efficient-Grad are well under 90°. Since the error gradients are pruned with the expectation that remains, the sign-symmetric feedback remains unchanged. Thus it is still causing the weight to be aligned with the random fixed feedback, as analyzed in [85]. As discussed in Sec. 4.4.2, the lower the angle between error gradients the better the learning capability. Compared to the original FA, the sign-symmetric FA with stochastic gradient pruning can even reach an angle under 45°. The angles of the 300 epochs' training on ResNet-18 [105] are shown in Fig. 4.4, and discussed in Sec. 4.4.2. The FC classifier layers stay aligned with the random feedback because the over-regularization is suppressed in FC layers, whereas in the convolutional layers, it drops rapidly but tends to be stable. This makes sense because the BN layer solves the neuron-turning-off problems mentioned above and restores the internal covariate shift layer-wisely.

One of the critical parts of the Efficient-Grad algorithm is to determine a dynamic pruning threshold τ that will preserve the original validation accuracy that a given DNN model can reach. Consider the cumulative density function (CDF) Φ of a given δ_l. If we use a pruning rate γ to control the gradient sparsity, i.e., make the expected pruned ratio

out of the whole error gradient distribution of every pass in the backward phase γ, then Eq. (4.6) holds:

$$\gamma = 1 - [1 - \Phi(\frac{\tau}{\sigma})] \times 2 = 2\Phi(\frac{\tau}{\sigma}) - 1, \tag{4.6}$$

$$\tau = \Phi^{-1}(\frac{1+\gamma}{2}) \cdot \sigma. \tag{4.7}$$

With Eq. (4.7), the ideal ratio of δ_l, which was stochastically pruned in Eq. (4.5), is set as γ by manipulating τ. The expectation of δ_l in Efficient-Grad is almost unchanged, leading to a negligible classification accuracy loss. Originally, in Algorithm 1, the backward phase requires the transposed weight matrix and the dense gradients matrix for computation, which causes an excessive energy cost and latency for edge devices. With Eq. (4.2) and Eq. (4.5) implemented on the supporting hardware accelerator, these two issues can be avoided by utilizing the low-cost feedback and the gradients sparsity to discard backward operations, for both the current layer and the upcoming layer. The details of the hardware design are discussed in Sec. 4.5.

4.4.2 Angle Analysis

To further avoid the EKN effect of FA on convolutional layers, we bring in a BN layer [109] to alleviate the regularization effect brought by the convolutional kernel, random feedback modulator signals, and ReLU activation function. As discussed in Sec. 4.3.1, the angle between modulator signals prescribed by FA and that prescribed by BP, indicates the learning capability of FA to some extent [85]. In FA, the training starts with the angle around 90°. This orthogonality embodies the irrelevance between the modulator signals from BP and those from FA. As learning goes through many epochs, the angle soon shrinks, and the learning capability of FA is observed to be of the same level as that of BP. As for Efficient-Grad, the sign-concordance is imposed on the feedback weights as shown in Eq. (4.2). We measure the angles, θ, of the modulator signals pair when training with ResNet-18 on CIFAR-10 in Efficient-Grad. These angles are shown in Fig. 4.4 as

$$\theta = \cos^{-1}(\frac{[sign(W_{l+1}) \odot |B_{l+1}| * \delta_{l+1}]^\top (W^\top * \delta_{l+1})}{\|sign(W_{l+1}) \odot |B_{l+1}| * \delta_{l+1}\| \cdot \|W^\top * \delta_{l+1}\|}), \tag{4.8}$$

where δ_{zero} denotes the ratio of zero gradients out of the whole error gradient distribution.

Initially, the angles of both convolutional layers and FC layers start at around 90° before learning. As the training proceeds, with increasing epochs, the angles of all layers shrink, especially the last two layers, FC6 and Conv5b. For γ over 90%, all the modulator

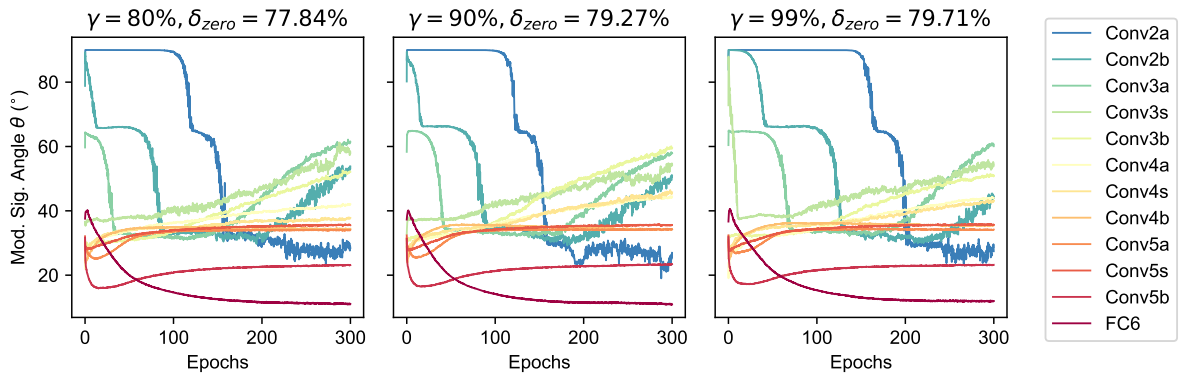


Figure 4.4: **The angle of each layer.** In ResNet-18 between the modulator signal prescribed by BP and by Efficient-Grad, respectively.

signals' angles, except for those of the last three layers, require longer epochs to decrease, and hence cause a low validation accuracy, as shown in Fig. 4.4. The shrinkage of angles with $\gamma \leq 90\%$ across all layers in ResNet-18 also supports the effectiveness of the Efficient-Grad algorithm for DCNN training. We adopt $\gamma = 80\%$ as the pruning rate setup for the optimal accuracy-sparsity trade-off.

4.5 Hardware Architecture

To enhance the energy efficiency of edge devices which are dedicated to the DCNN training task, we design an architecture that leverages the sparsity and memory access reduction brought by Efficient-Grad. This section describes and analyzes this supporting hardware architecture design. The baseline computing accelerator is inspired by Google's TPU [110] with a systolic array. Other components include dedicated compute components for non-convolution operations, such as activation and downsampling. As shown in Fig. 4.5, a unified on-chip SRAM buffer for the input feature maps, filter weights, and output feature maps for both the forward and backward phases are also included. A host processor for custom RISC-V instructions is utilized to control the accelerator tiling behaviors and external memory access. Finally, a directed memory access (DMA) module is also used to connect the on-chip buffers and the DRAM.

In this section, we describe the hardware components in detail. We introduce our proposed adaptive dataflow on the computing module for higher energy efficiency, as well.

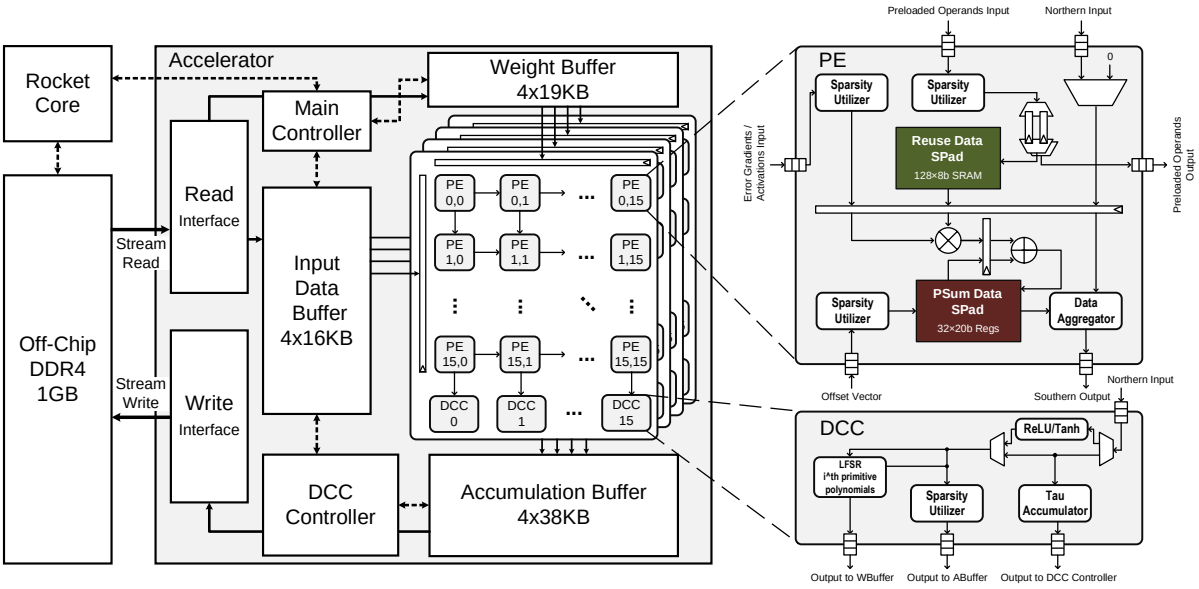


Figure 4.5: **The overall hardware architecture of Efficient-Grad.**

4.5.1 Hybrid-WS-OS Dataflow

The goal of the execution stages is to pipeline the operation of the processing element (PE) cluster for a higher operating frequency. We design the stages to be as fine-grained as possible, such that the stalls caused by the transition of the output feature map are minimized and the operand reuse can be maximized once it is read into the on-chip buffer. [77] proposed a well-recognized taxonomy for different types of dataflows, with their pros and cons. These dataflows, based on a systolic array, are superior to broadcasting operands with adder trees [93, 96] for convolutional and FC multiplication in terms of data reuse. Hence accelerators with these dataflows tend to have better energy efficiency. However, training different DCNNs normally requires different computation schemes, in which different loop tiling strategies are required, and none of the dataflows in the taxonomy can outperform the others in all DCNNs. It is, thus, hard to assign only one fixed dataflow to meet criteria such as maximizing input and weights reuse and minimizing output partial sum movement at the same time.

To deal with this issue, we design our dataflow in a hybrid manner, specifically, a combination of dataflows which are both weight stationary (WS) and output stationary (OS) with mode switching, depending on the target training workload model which it is handling. Even though, as in Gemmini [111], the synthesis results of a systolic array with a WS dataflow are slightly more energy and area efficient than those with an OS dataflow, we still adopt OS in our PE cluster for ResNet-18. With the depth-wise and

vanilla convolution making up a huge portion of ResNet-18, it dumps the partial sum more frequently than MobileNet. Thus, running an OS dataflow on ResNet-18 is more efficient. Compared with other dataflows, OS ones generally have the lowest DRAM access, which has a strong impact on the overall energy efficiency, as analyzed in [77]. If the workload DCNN model contains more 1x1 point-wise convolution than ResNet-18, such as MobileNet, we will use WS in the PE cluster due to its superiority for throughput when each element in the weight matrix is utilized at a maximum rate during the computation.

4.5.2 Processing Element

A PE is a combination of minimized control units, a multiply-accumulate (MAC) unit, and the register files for holding up input operands and partial sum aggregation. To minimize the energy cost brought by the internal data movement, we use the OS dataflow in the nomenclature of Eyeriss for either standard convolution or depth-wise convolution in DCNN training. Nevertheless, PE also supports WS for the reasons discussed in Sec. 4.5.1.

Architecture Design

To achieve better performance, power, and area (PPA) characteristics, the PE is designed in a homogeneous fashion; i.e., each PE is capable of handling all the general matrix multiplication (GEMM) of these four phases in training. For both the forward phase and backward phase in Algorithm 1, the convolutional operands read from on-chip buffers are pushed into the PE via the FIFO ports, while the offset vector is used by the sparsity utilizer to skip the zero gradient computation. The sparsity utilizer will firstly dissect the compressed sparse column (CSC) format of the operands in different phases to locate the desired computation index.

Next, to eliminate the stalls caused by switching tiles in a convolution, we use a set of ping-pong buffers in each PE to store the preloaded operands and perform MAC at the same time, as shown in Fig. 4.5. For WS mode, during the preloading stage, the ping buffer will accept the operands passed in from the preloaded operands input port, and it will also pass out the operands at the preloaded operands output port at the next cycle. Meanwhile, the pong buffer will be used for MAC at this stage, with the input from the northern input port. When it turns to the compute stage, the roles of the ping buffer and pong buffer will switch, such that the operands from the weight port can be

reused maximally. The ping-pong buffers will alternately take in a preloaded weight and reuse it for weight-computing-intensive convolutions, such as 1x1 point-wise convolution. The partial sum calculated in each cycle will be shipped out in the southern output port. Meanwhile, the switching mechanism still applies to OS mode, but the weight is taken in by the northern input port and dumped by the southern output port, while the preloaded operands input port preloads the partial sum. The ping buffer will accept the preloaded partial sum and thereafter be used for MAC until the accumulation for the output finishes, while the pong buffer keeps preloading the next targeted partial sum.

For the backward phase of Efficient-Grad, the PEs simply impose the sign bit from the global weight buffer on the random fixed magnitude preloaded in the reuse data scratchpad beforehand, to perform the convolution shown in Phase 2 of Algorithm 1. The hybrid-WS-OS dataflow still applies.

The on-chip reuse data scratchpad is used to hold the reuse data, such as input, weight, and random matrix. There is also another partial sum data scratchpad in each PE for direct output aggregation for all the input channels, which avoids unnecessary partial sum movement between the global buffer and PE.

Bit-width Setting

To achieve training quantization with decent performance, we follow the quantization scheme which is adopted by several prior arts [112, 113], by utilizing a dynamic fixed-point representation whose rounding process is conducted in a stochastic manner for the output value.

In the forward phase, the inputs a_l and W_{l+1} are in 8 bits, and the partial sum of a_{l+1} is in 19 bits for accumulation. The scaling factor, $S_{W_{l+1}}$, for W_{l+1} is in 8 bits and fixed once the convolutional result of a_{l+1} is finished. It is used to round a_{l+1} back to 8 bits together with a pseudo random number for stochastic rounding and a dynamic scaling factor, $S_{a_{l+1}}$, as in the dynamic fixed-point scheme.

For the backward phase, the input δ_{l+1} and the sign-symmetric feedback $\bar{B}$ are in 8 bits, while the partial sum of δ_l is in 19 bits for accumulation. Before being passed to layer $l-1$, the convolutional result of δ_l is rounded back into 8 bits similarly to the forward phase. For the weight gradient and update phase, since the gradients in Efficient-Grad are optimized for the sake of learning efficiency, compared to the vanilla BP, the input δ_l is read in 19 bits before the rounding process happens for δ_l.

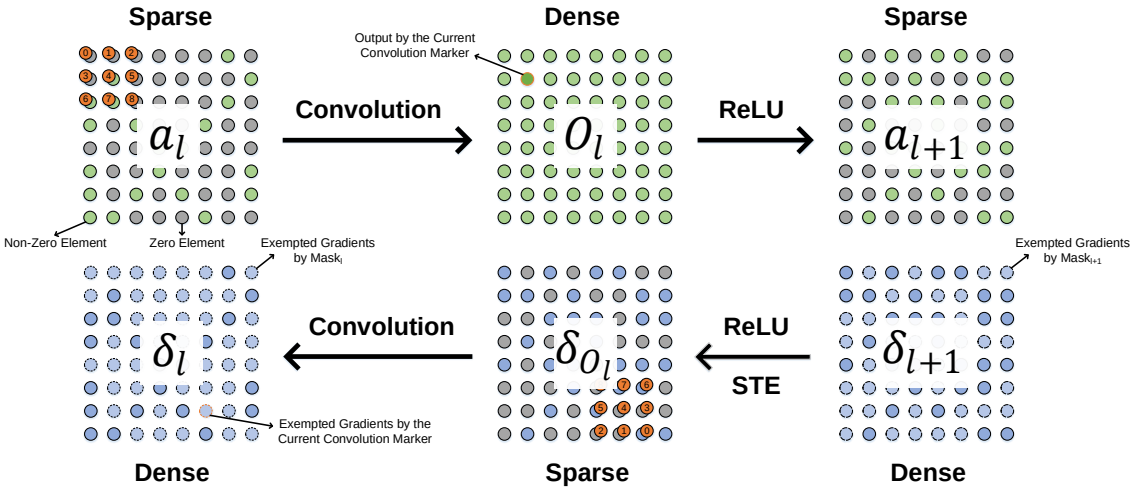


Figure 4.6: **Backward convolution exemption in Efficient-Grad.** The forward phase and backward phase for a particular layer l in ResNet-18. In this example, the feature map size and weight filter size are 8 and 3, respectively.

Backward Convolution Exemption with CSC Offset

During the forward phase, the offset vector in the CSC of each feature map is computed and preserved for the usage of Eq. (4.5) in the backward phase. During this phase, for a specific convolution between δ_{l+1} and W^T, the final sum will be aggregated at the southernmost PE in the cluster, whose output is sent to the sparsity utilizer residing in a dedicated compute component (DCC). Normally, the input feature map a_l is sparse due to the effect of ReLU activation ahead, and such sparsity can be utilized to discard redundant convolutions for $\delta_{O_l} = \delta_{l+1} \odot \sigma'(a_l)$ during the backward phase. Therefore, in this phase, we use the offset vector in the CSC which we encoded during the forward phase for the a_l to utilize this sparsity. Thus, the convolutions which generate the gradients that are about to be canceled by the ReLU are exempted by the zero-element mask of the offset. Subsequently, the offset vector of the returned δ_l will be independently encoded based on stochastic gradient pruning. As Eq. (4.5) finishes, the sparsity utilizer will directly encode the sparse $\hat{\delta_{l_i}}$ into the CSC format, namely, the offset vector and the data vector. Similar to [102], the offset vector indicates which operands need to be calculated. One of the reasons why we reuse the preloaded weight in the reuse data scratchpad, rather than feature map data, as shown in the block diagram in Fig. 4.5, is that both the activation in the forward phase and the output gradient in the backward phase in Algorithm 1, are sparser than the weights and thus can be processed by the sparsity utilizer whilst reusing the weights.

An example in Fig. 4.6 is given to illustrate the usage of the offset vector to eliminate the convolution in the backward phase. A particular layer, l, in ResNet-18 with an 8x8 feature map and 3x3 filter weights, conducts the forward phase in the top row and the backward phase in the bottom row. For the representation of the ReLU activation gradient, the straight-through estimator (STE) is used. The offset vectors recorded in a_l of the forward phase together constitute the $Mask_l$, which will be used to exempt the convolutions which generate δ_l in the backward phase. The backward convolution exemption applies for all the layers in the backward phase to greatly utilize the sparsity residing in the gradients, together with stochastic gradient pruning.

4.5.3 Systolic-Array-Based PE Cluster

Typically, all the PEs are uniform and fully pipelined. The PE cluster is a 2-D systolic array that handles both convolutional layers and FC layers. The Efficient-Grad accelerator contains four PE clusters, which comprise 1024 PEs in total, to provide the sufficient throughput for DCNN training. Unlike TPU [110], which is fixed in WS mode, ShiDianNao [98] and [114] in OS mode, and Eyeriss [77] in a tradeoff of both, named row-stationary (RS) mode, we adopt a hybrid-WS-OS dataflow, as discussed in Sec. 4.5.1, which is determined depending on the processed layer type on a layer by layer basis. Such flexibility in dataflows enables higher throughput and better data reuse with fewer stalls, compared to those approaches with fixed dataflows, as above. As mentioned in Sec. 4.5.2, we use an OS dataflow for better throughput and energy reduction in the ResNet-18 workload, while keeping the interconnect streamlined compared to the network-on-chip in Eyeriss.

Each PE cluster consists of 256 PEs in a 16x16 array to fully utilize the address vector. There are shift registers at the northern and western parts of the PE cluster to receive the input feature maps data and weights. These shift registers orchestrate the matching between sparse input operands from the western side of the PE cluster, and the weight from the northern side. When the control unit turns to OS mode, the southern PE receives the weight from the northern PE, while the eastern PE receives the input feature map from the western PE.

In the southernmost part of the PE cluster, each southernmost PE is connected to a DCC, which generates the uniform random numbers, performs activation functions, and accumulates τ for use in Eq. (4.5). Fig. 4.5 shows the architecture of the DCC. It contains

a pseudo-random number generator (PRNG) with a CSC encoder. To cater for the needs of the random numbers, namely, $\boldsymbol{B}$ in Eq. (4.2) and $\boldsymbol{r\tau}$ in Eq. (4.5), a series of seven linear feedback shift registers (LFSRs) are used as the PRNG. These LFSRs are connected on top of each column of the PE cluster. In the backward phase, the partial sum of δ_l is aggregated at the southernmost output of the PE cluster and processed by stochastic pruning with the generated random $\boldsymbol{r\tau}$. Meanwhile, the generated random $\boldsymbol{B}$ is passed towards the northern parts. Inspired by another work [112], which adopts stochastic data processing as well, we design a seven-LFSR group, each of which is a 19-bit Fibonacci LFSR whose bitlength is guaranteed to cover one of the weights. Meanwhile, randomness is guaranteed to be good enough. For each trial in training, we input different seeds from the seed table which is stored in the input data buffer, and the seeds remain the same for all iterations. In the seven-LFSR group, each LFSR is tapped by different primitive polynomial pairs, such that the generated randomness for each one is different from that of the others.

For each batch in the backward phase, τ will be pushed into a FIFO, dubbed the Tau accumulator, to accumulate different τ at each batch. Once the Tau accumulator is full, the gradients will be pruned by the average of τ residing in the Tau accumulator. Thereafter, the average of all τ stored in the Tau accumulator will be used in Eq. (4.2), to eliminate the repetitive threshold calculations on Eq. (4.7) for each batch.

4.5.4 On-Chip Global Buffer, Main Controller, and DCC Controller

For each PE cluster, the global buffer array is split into 19 KB for weights, 16 KB for the input feature map, and 38 KB for the output feature map. Setting the weight buffer to be larger is beneficial for more weight reuse during WS mode. This also helps in OS mode in holding the previous computed partial sum, which is in 19 bits. Each weight buffer is of 19-bit read/write port width, with 1024 words. Meanwhile, each input data buffer and accumulation buffer take an 8-bit and 19-bit read/write port width, with 2048 words, respectively. There are four sets of buffer arrays matching four PE clusters. Together they form a global buffer array with a size of 292 KB. The weight buffer stores an 8-bit weight and sign-symmetric feedback for the forward phase and backward phase. Notably, in the weight gradient and update phase, it stores a 19-bit error gradient. In the backward phase, the 8-bit random feedback and the corresponding sign will be loaded from the

DRAM, respectively. Since both the weights and feature maps are distributed around small positive and negative values, we adopt the sign-magnitude representation to avoid prohibitive switching activity for their value updates in training. In such a case, during the backward phase, one bit is valid as the sign bit for one feedback element in the 8-bit subword of the weight buffer.

During the backward phase, the weight buffer holds the magnitude of the fixed random feedback of each batch, with the instructions which come from the control unit. These magnitudes will be pushed into the PE cluster and stored in the reuse data scratchpad of each PE individually. Therefore, the external DRAM access is minimized by reading consecutive sign bits of the weight matrix using DMA ports.

The control unit in the accelerator tiles the input and weights of each layer batch-wisely for data reuse. The CSC format across all the on-chip global buffers helps densify all three phases in DCNN training. To provide fine-grained loop tiling control and handle efficient external DRAM data access, we adopt the Rocket core [115] as the host processor. The DMA module is based on the IceNet network interface controller module from Chipyard [116], which can be tightly adapted to the Rocket core. We can access the DRAM directly, bypassing the L2 cache of the Rocket core to read or write to the on-chip SRAM buffer in a bank size fashion. Both the read DMA interface and write DMA interface are specially adapted for the selective gradients, because the gradients are pruned and hence non-consecutive at the initial stage. After the gradients are encoded in CSC format, they turn consecutive and are thereafter stored in the accumulation buffer and further in DDR4 by the write DMA interface.

The DCC is used for non-linear activation functions, such as ReLU and Tanh. It also contains a scaling factor for layer-wise transformation. Sub-sampling such as MaxPooling and AvgPooling are also considered in this component. The DCC controller takes the DCC control signals passed from the main controller to configure all DCCs in the PE cluster. Moreover, it handles the BN layer in an integer fashion, as proposed in [117]. It realizes the BN function by calculating the mean and variance in the dynamic fixed-point format, as the output of convolutions is done. For the forward phase, once we finish the convolution and the output is rounding back to 8 bits, we use it to calculate the mean and variance on-chip. As for ResNet-18, the parameters amount of the largest BN layer is 1024. In such a case, the storage requirement of accommodating 64-batch-size BN layer parameters (the mean and variance) is 128KB. Therefore, the BN parameters can be fully stored on-chip in the accumulation buffer while doing the BN operations, to circumvent

dumping these BN parameters to off-chip DRAM with higher overhead.

Given the observation that in the transfer learning scenarios where most of the edge-training tasks are needed for model customization to new features, the moving mean and variance have converged in the pre-trained model [118], we fix the mean and variance and only update the γ and β in the backward phase. In such a case, the gradients with regard to the mean and variance are both zero, and the resulting input gradient of the BN layer turns to be linearly dependent on the output gradient with the coefficient of the value of variance. By fixing the mean and variance in the backward phase, we relieve the computational burden of the backward phase of BN and exclude the complex BN cache in this design, since the accumulation buffer is capable of buffering the data for BN.

4.6 Experiments

In this section, we introduce the experiment setup for both the algorithm and hardware of Efficient-Grad, as well as the test results on the CIFAR-10 [119] and ImageNet (ILSVRC2012) [120] datasets with popular benchmark DCNN models. First, synthesized results of Efficient-Grad are analyzed with algorithm-level testing verification. Second, to emulate the DRAM access, we adopt a cross-platform, co-simulation methodology, customizing two cycle-accurate simulators at different domains. We also prove the superiority of the Efficient-Grad design.

4.6.1 Algorithm Setup

Considering better convergence stability and on-chip implementation feasibility, an SGD optimizer is adopted for all the networks on both datasets. For CIFAR-10, the initial learning rate is set as 0.01, with a step decay of 0.1 over every 70 epochs. We use a weight decay of 5×10^{-4}, and a momentum of 0.9 for the optimizer. Meanwhile, for ImageNet, the initial learning rate is set as 0.1, with a cosine annealing scheduled decay. The weight decay is also employed for ImageNet, and set to 1×10^{-4} with a momentum of 0.9. The Efficient-Grad algorithm is tested on an NVIDIA GeForce RTX 3090 GPU card with PyTorch 1.8, which is a deep learning framework with good compatibility on edge devices, such as NVIDIA's mobile GPU, Jetson modules. The high potential of adopting Efficient-Grad on Jetson modules is discussed in more detail in Sec. 4.7.3.

4.6.2 RTL Implementation with Chisel Tester Environments

The pervasive object-oriented methodology in software design has started to improve the productivity of classical hardware descriptive language for digital hardware design. Among several emerging hardware generating languages [121–123], we use Chisel for the register transfer language (RTL) development on account of the advances brought by the scalability of modern software languages. The behavioral simulation of the Efficient-Grad RTL is done with Chisel Tester [124]. Based on the fork-join property of Chisel Tester, we are able to build a simulation-based timing model in Scala, to handle the intra-PE pipeline within the PE cluster. We synthesize the ASIC design using the Synopsys Design Compiler with a 14nm SMIC FinFET technology. For more accurate power and latency modeling, the SRAM/RegFile macros are generated by the Synopsys 14nm off-the-shelf memory compiler rather than a popular yet over-optimistic architecture-level open-source framework such as CACTI 7.0 [125].

4.6.3 Performance Co-Simulation Platform

To manifest the improvement of the Efficient-Grad algorithm and its hardware support in terms of energy-efficiency, we customize an open-source cycle-accurate DCNN inference simulator, SCALE-Sim [126], for sound modeling of the training workload. To adapt the training, as illustrated in Algorithm 1, both the topology and the data reuse computing logic of SCALE-Sim are redesigned, particularly for the backward phase. Subsequently, to better analyze the performance of the dumped traces, both SRAM and DRAM access traces are dissolved as one trace per line.

After obtaining the cycle-level memory traces of DCNN training, DRAMSim3 [127] is utilized for DRAM access modeling to estimate the access latency and energy consumption brought by DRAM traffic. DRAMSim3 is yet another cycle-accurate memory system simulator which supports trace-based simulations. Intrinsically, it offers the best and most accurate simulation performance among existing cycle-accurate DRAM simulators [125, 128, 129]. In the modeling of Efficient-Grad, we use a DDR4_8Gb_x8_3200 configuration with a 1GB size as a DRAM model in DRAMSim3, which provides 64 bytes ($8byte \times 8banks$) per access. In the memory system wrapper of DRAMSim3, the Efficient-Grad request is pushed into the transaction queue, where the traces containing commands (either read or write), address mapping, and clock cycle, are issued together into the ranks configured by the DRAMSim3 configuration.

Table 4.1: Evaluation Results for Efficient-Grad's Algorithm

Model	Dataset	Baseline (BP)		[93]		$\gamma = 90\%$		$\gamma = 80\%$	
		$acc(\%)$	$\delta_{zero}(\%)$	$acc(\%)$	$\delta_{zero}(\%)$	$acc(\%)$	$\delta_{zero}(\%)$	$acc(\%)$	$\delta_{zero}(\%)$
AlexNet	CIFAR-10	86.38	14.78	28.45	64.87	78.52	87.20	85.69	73.74
ResNet-18	CIFAR-10	95.56	0	53.66	0	90.05	81.67	93.31	77.84
AlexNet	ImageNet	56.38	9.45	7.39	51.34	50.99	88.10	56.26	69.13
ResNet-18	ImageNet	68.73	0	11.41	0	61.26	80.31	65.41	55.46

To estimate the total energy consumption in a training-sample-granularity, we adopt the equation combining the energy models in [101, 130] as follows:

$$Energy_{Total} = Power_{Accelerator} \times Cycles_{Total} + MA_{DRAM} \times E_{DRAM}, \tag{4.9}$$

where $Power_{Accelerator}$ is the total power made up of both the dynamic and leakage power, MA_{DRAM} is the total DRAM access of the accelerator, and E_{DRAM} is the energy consumption for each DRAM access, generated by DRAMSim3. The accuracy of the simulation results of DRAMSim3 is proven by its validation process, in which the timing is used against Verilog models.

4.7 Evaluation and Discussion

In this section, we present the overall performance of the Efficient-Grad algorithm and its supporting hardware accelerator. The performance evaluation with the energy consumption generated during DCNN training is extracted from the in-house cycle-accurate simulator we built.

4.7.1 Functionality Results

To examine the learning capability and sparsity utilization potential of Efficent-Grad, we present the average learning statistics for 10 trials in Table 4.1, where acc refers to validation accuracy of the testing dataset and δ_{zero} refers to the percentage of zero gradients during the model training. We trained several benchmark DCNN models, such as AlexNet, MobileNetV2 and ResNet-18, for 300 and 180 epochs on CIFAR-10 and ImageNet, respectively, to reach the model convergence.

As shown in Fig. 4.4, the learning capability of Efficient-Grad is guaranteed to be the same as that of BP, as indicated by the convergence of the modulator signals angles when $\gamma = 80\%$. Compared to AlexNet, ResNet-18 is equipped with residual blocks with BN

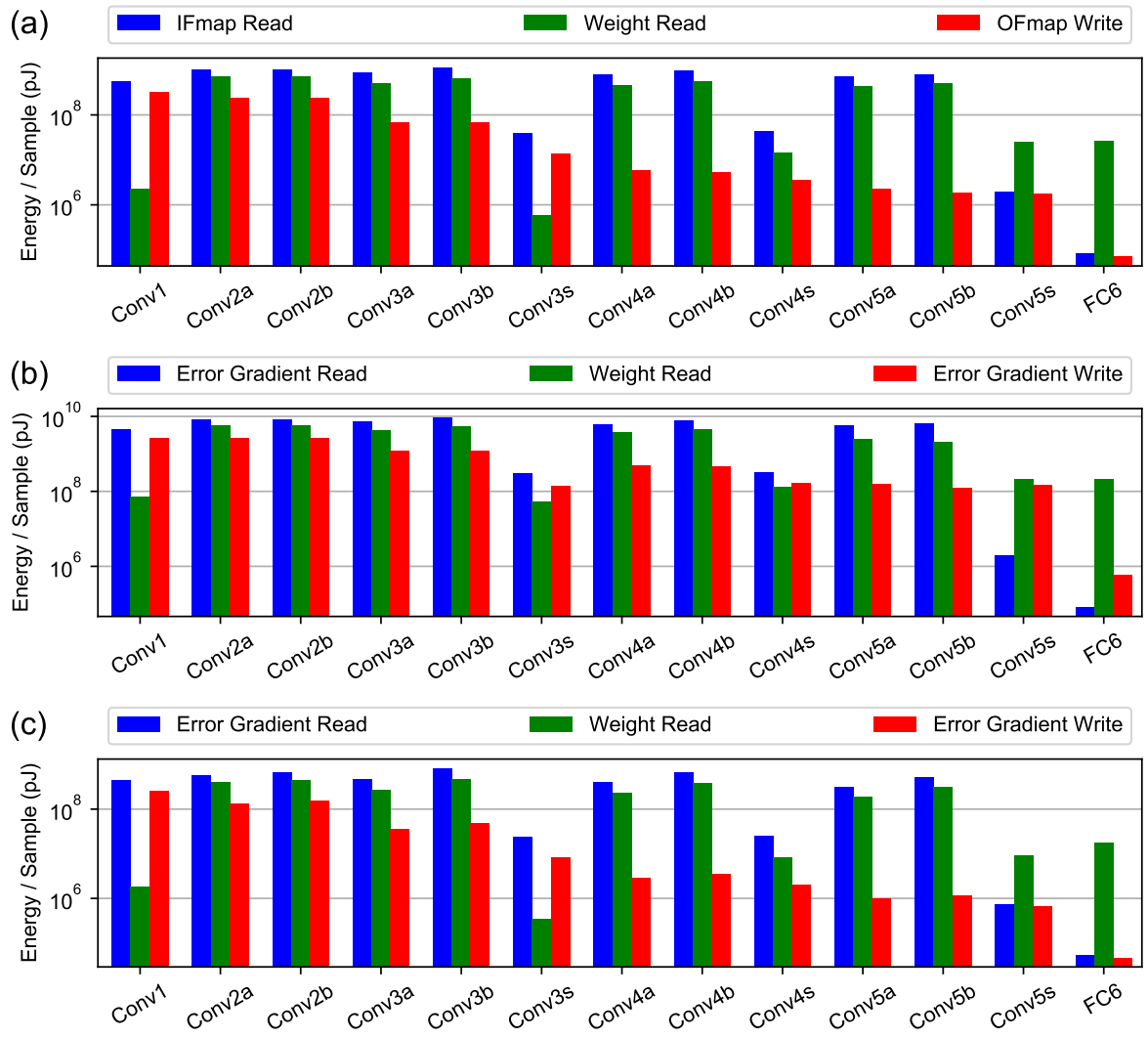


Figure 4.7: **DRAM energy dissection of the vanilla BP and Efficient-Grad.** On ResNet-18 under (a) the forward phase, (b) the vanilla backward phase, and (c) the Efficient-Grad backward phase.

layers which endow it a better learning capability, even with fewer parameters. However, due to the scaling effect in each BN layer, the amount of zero gradients for ResNet-18 is always zero, which leads to redundant computations with BP training, especially when the magnitudes of the gradients are tiny.

Thus, we perform a grid search on the γ of Efficient-Grad. Among the various γ, with $\gamma = 80\%$, the validation accuracy of Efficient-Grad is maximized for both AlexNet and ResNet-18 for two mainstream image classification datasets, namely, CIFAR-10 and ImageNet. The performance degradation of Efficient-Grad is below 1% and 3% for AlexNet and ResNet-18, respectively, on CIFAR-10. We also note that the BFA training adopted in [93] led to significant training performance loss due to its incompatibility with convo-

lutional layers, as we discussed in Sec. 4.3.1.

As on CIFAR-10, Efficient-Grad achieves great generality in model training for both AlexNet and ResNet-18 on ImageNet, with a validation accuracy of 56.26% and 65.41%, respectively. Among all benchmark networks, the maximum accuracy loss for ImageNet rises from 2.3% for CIFAR-10 to 3.3%. This increment of the accuracy loss is due to the increment of the data complexity of ImageNet. Meanwhile, BFA still suffers from performance degradation on ImageNet for the same reason as on CIFAR-10. Furthermore, Efficient-Grad attains a gradient sparsity increment of more than 50% for both AlexNet and ResNet-18 on ImageNet. The gradient sparsity utilization, together with the on-chip random feedback access in the backward phase, enables the dedicated hardware architecture to enhance the throughput while maintaining a low total energy cost.

4.7.2 Co-Simulation Results

Fig. 4.7 shows the DRAM energy consumption of both the forward and backward phase of training ResNet-18 with Efficient-Grad, where the layers with the suffix "s" hereinafter are the residual connection. In Fig. 4.7(a), the input feature map and weight are read from the DRAM into the PE cluster via the activation input port and preloaded operands input port, respectively, in the forward phase. According to the ResNet-18 topology, the Conv1 layer will have the highest output feature map write, thus maximizing the weight reuse, and the same applies for the point-wise convolution with the big input feature map of Conv3s. For the designed on-chip buffers, as shown in Fig. 4.5, the more the operands are reused, the less DRAM access is required. In ResNet-18, the subsequent layers are smaller in feature map size. Thus, for both the input and output feature map, the DRAM access energy declines as the network goes deeper. In Fig. 4.7(b), the error gradient and the raw weight will be read in, respectively. As discussed in Sec. 4.3.3, the transpose demand in the backward phase costs a large amount of energy for conventional BP. In Fig. 4.7(c), the error gradient and the optimized weight will be respectively read in. With the help of the Efficient-Grad algorithm, the DRAM access of the backward phase can be relieved to even lower than that of the forward phase. This is due to the avoidance of the transpose weight being read in and the effect of error gradient sparsity utilization.

Table 4.2 summarizes the read/write access bytes of the on-chip SRAM for a patch of the ResNet-18 workload under OS mode, according to the SRAM traces dumped in our co-simulation platform. In this platform, we retrieve the SRAM traces and further

Table 4.2: SRAM Read/Write Access Comparison between the Vanilla BP and Efficient-Grad

Training Phase	Total SRAM Access Bytes (Times)	
	Input Data Buffer / Weight Buffer Read	Accumulation Buffer Write
Forward Phase	95.8MB	2.2MB
Vanilla Backward Phase	766.2MB	18.0MB
Efficient-Grad Backward Phase	59.3MB	1.5MB

DRAM traces using SCALE-Sim. The trend of the SRAM access in the table is similar to that of the DRAM access shown in Fig. 4.7. It shows that Efficient-Grad is superior also in terms of the SRAM access, thanks to the utilization of the gradient sparsity with the CSC format and the asymmetric random feedback.

Fig. 4.8 shows the total energy consumption brought by DCNN training on a sample under BP and Efficient-Grad, respectively. The estimation comes from Eq. (4.9). Generally, Efficient-Grad increases the overall energy efficiency by more than five times compared to the vanilla BP. Readers may notice that the Conv5s layer has the highest efficiency improvement with an around 10-fold diminishment. The reasons are two-fold. Firstly, the point-wise convolution on a small feature map in Conv5s causes a higher weight transpose reading for the DRAM in the backward phase. Secondly, with the stochastic pruning of gradients in Efficient-Grad, the closest layer to the classifier, the FC layer, tends to have a higher sparsity to utilize due to the slighter regularization effect brought by sign-symmetric FA. The accuracy of the simulation results of DRAMSim3 is proven by its validation process, in which the timing is used against Verilog models.

4.7.3 Comparison with Prior Arts

We compare the specifications of the Efficient-Grad accelerator with the state-of-the-art DCNN mobile training devices. Table 4.3 presents the accuracy, throughput, and energy efficiency of various devices (works) for edge training, including, Efficient-Grad, DCNN training accelerators from academia, and the NVIDIA Jetson TX2, a commercial embedded GPU. A hyphen in Table 4.3 denotes nondisclosure in the context. Of the academic works, the processor in [93] is the only one that supports ImageNet training. However, due to its utilization of binary FA, it cannot support convolutional layer training. The shared exponents bias to successfully exploit the floating-point operations module in the edge accelerator meeting all the constraints is utilized in [131]. Nevertheless, neither of these works supports hybrid-WS-OS dataflows, while [114] is based on enhanced OS mode and [108] is based on multicasting without data reuse. Hence, Efficient-Grad reaches

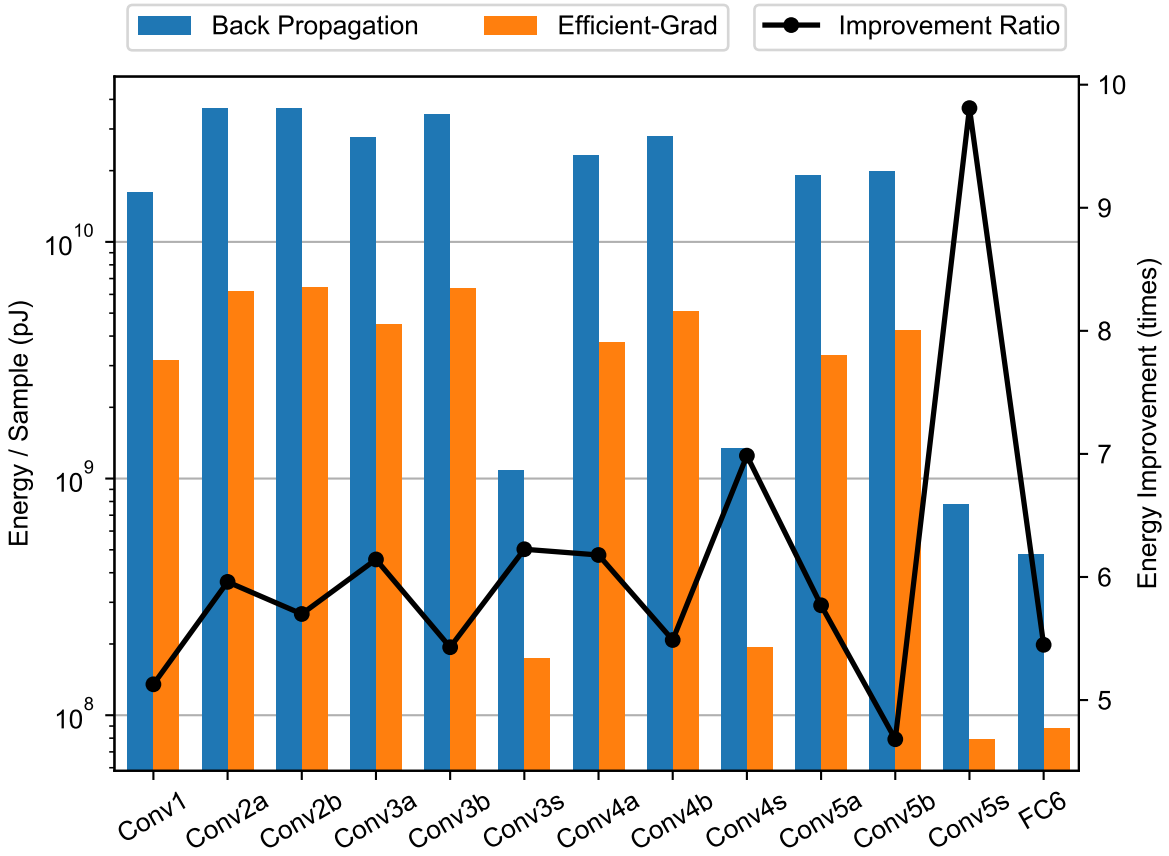


Figure 4.8: **Overall energy consumption on ResNet-18.** Efficient-Grad vs. BP. The improvement of Efficient-Grad on BP is shown by the black dots.

a higher clock rate and throughput. In addition, with the help of the optimization on gradients calculation, especially for the backward phase, the overall energy is greatly decreased. This leads to a satisfactory reduction of power consumption and improvement of energy efficiency for Efficient-Grad, which has about 3.7x the power efficiency of [114], and consumes only 56% of the energy consumption for one epoch of AlexNet training. This energy reduction is achieved by the gradient optimizations discussed in Sec. 4.4, whose instantiation on ResNet-18 is presented in Fig. 4.7. We also compare the total energy cost for DCNN training on a one-sample basis in the table, which shows that Efficient-Grad saves 44% of the training energy cost compared to [114].

The gradient optimization techniques in Efficient-Grad can also be adapted to some popular mobile GPU platforms such as the Jetson modules. Our ASIC-based prototype proves the effectiveness and efficiency of the Efficient-Grad. The PyTorch implementation of the Efficient-Grad algorithm can be adapted to the Jetson modules, ranging from the Jetson TX2, Jetson Xavier NX, to the Jetson Nano, after PyTorch is installed on-board. However, a degradation in performance is expected, and careful orchestration of

Table 4.3: Comparison to Popular DCNN Training Accelerators

	Efficient-Grad	ISSCC'21 [131]	JSSC'20 [114]	TCAS-I18 [93]	Jetson TX2
Technology	14nm	40nm	65nm	65nm	16nm
Die Area (mm^2)	8.90	6.25	10.24	3.52	43.6
Learning Support	CONV, FC	CONV, FC	CONV, FC	FC	CONV, FC
Sparsity Support	YES	YES	YES	YES	-
Supply Voltage (V)	0.81	0.75 ~ 1.1	0.63 ~ 1.0	1.2	5.5 ~ 19
Maximum Frequency (MHz)	633	180	160	200	1377
PE Bit-precision (Bit)	INT8,19,32	FP8, SEB	INT8	INT13,16	FP16
On-chip SRAM ($Byte$)	420K	293K	364K	119K	1.25M
Power Consumption (mW)	169	230	120.5	126	7500
Energy Efficiency ($TOPS/W$)	3.83	1.64	1.03	0.41	0.012
Training Energy Cost ($J/Sample$)	7.67m	-	13.7m	-	123m
Target DCNN Workload	AlexNet/ResNet-18	ResNet-18	VGG16/AlexNet	MDNet	-
Target Data Set	CIFAR-10/ImageNet	CIFAR-10	MNIST	ImageNet	-

dataflow and precision is required to circumvent the limited amount of available RAM. The application of Efficient-Grad on Jetson modules is thus left for future work.

4.8 Conclusion

In the current article, we present Efficient-Grad, an efficient yet effective substitute for the vanilla back-propagation-based DCNN training algorithm. It enables us to make full use of both the elasticity of the weight symmetry problem and the redundancy residing in the conventional back propagation algorithm. Hence, the sparsity of gradients can be utilized. Our algorithm is effective for DCNN training since it involves negligible validation accuracy loss. We also propose and implement a supporting hardware accelerator architecture for validation and evaluation. In addition to the high energy efficiency Efficient-Grad achieves, it greatly saves external DRAM access costs, and thus the energy unit cost of training a DCNN is greatly optimized. As demonstrated in the article, our proposed design increases the throughput by approximately 4.83x and reduces the energy unit cost by 44%. It consumes 169 mW at 633 MHz with an area cost of 8.90 mm^2, which leads to superior energy efficiency of up to 3.72x that of prior accelerators. Moreover, it consumes only 43 mJ and 7.67 mJ, respectively, on ResNet-18 and AlexNet for one sample training, achieving 3.83 TOPS/W energy efficiency.

We believe that Efficient-Grad will be highly beneficial to both DNN practitioners and computer architects, and it will push forward the realization of edge training.

5 Conclusions

"It is said that the darkest hour of the night comes just before the dawn."

— Thomas Fuller, *A Pisgah Sight of Palestine and the Confines Thereof*

In this dissertation, we aim to construct a platform that supports 3D visual perception from both perspective of algorithms and hardware. First, we proposed a sparse 3D metric-semantic perception algorithm that utilizes prior knowledge from both anchored occupancy constructed by depth hypothesis and pixel-to-vertex matching correlation to refine the regression of TSDF and the classification of semantics. Second, we proposed a real-time streaming system on mobile devices leveraging RTMP to accommodate the indoor scene 3D metric-semantic reconstruction on the fly. Finally, to further mitigate the power consumption and computation overhead brought by the 3D perception algorithms, especially in the training time, we proposed a software-hardware co-optimization scheme that accelerates by pruning the activation gradients and approximating the sign-symmetric feedback. Future work to this end is two-fold as follows.

3D Visual Perception Algorithm

In the perception pipeline design, there are more priors can be included, e.g., surface normal priors and Eikonal regularization priors, etc. The temporal correlation embedded in the metric-semantic GRU of CDRNet can be reformulated into the self-attention due to the latter's better generalizability. In addition to SDF that we worked on, there are also more data structure as 3D representations to explore, such as a voxelized grid, or point clouds, or radiance field as mentioned in Sec. 3.1. Each of them has their own pros and cons and to think about how to integrate them to have complementary synergy is interesting. Particularly, NeRF is too cumbersome and inefficient from the robotic agent

perspective, hence a faster NeRF with a hash encoding or replace RGBA radiance with efficient representation such as SDF within the original NeRF work will also be meaningful to explore.

Although designing a cutting edge 3D perception algorithm is non-trivial, it will only counts only if the task is done on the robotic platform. Perception algorithms research that is closely related to robots will have its own niche. For instances, the collision cost during the agent's navigation can be penalized to the overall cost function; when the agent is conducting assigned tasks using semantic estimation, the estimation can be combined with both 2D prediction and the projection of 3D prediction through an extended kalman filter.

Approximate Computing at the Edge

There is no GPU on board in the HomeRobot example in Sec. 1.2 as the power consumption and the cost will be too high for an embodied AI agent with domestic purposes. It means if we want to offload the computations to the agent itself at the edge, the computing power of the controller hardware needs to be greatly enhanced. More customization to 3D processing can be done on-chip. For examples, the same occupancy voxel in the grid can be applied with a data reuse mechanism; the conventional back-projection on the pose can be substituted by the positional encoding such that a modulo-based sinusoidal function can be adopted. Meanwhile, another research direction is to customize the classical Von Neumann architecture for higher throughput and lower power consumption with near/in-memory computing, which overcomes the current I/O constraints as we are hitting the "memory wall" nowadays.

At the end of the day, a robotic agent's cost nowadays is still prohibitive, e.g., Boston Dynamics Spot robot costs $200,000 approximately up to now, which is totally unaffordable for both commodity and massive industrial usage. Nevertheless, the future of this burgeoning field still looks promising. Think about the evolutions of semiconductors since the 1950s at Bell Labs, and the internet commercialization since the 1990s. Akin to these, the availability of the embodied AI industry may take time to be fully accessible to the general public. It takes time, but as engineers, we believe that the advent day is coming soon.

References

[1] A. Parker, "In the blink of an eye: How vision sparked the big bang of evolution," 2003 (cit. on p. 1).

[2] J. Homman-Ludiye and J. A. Bourne, "Mapping arealisation of the visual cortex of non-primate species: Lessons for development and evolution," *Frontiers in Neural Circuits*, vol. 8, p. 79, 2014 (cit. on p. 1).

[3] Statista Inc., *Market insights - technology - artificial intelligence - computer vision*, https://www.statista.com/outlook/tmo/artificial-intelligence/computer-vision/worldwide (Dec. 2023) (cit. on p. 2).

[4] S. Yenamandra *et al.*, "Homerobot: Open-vocabulary mobile manipulation," *arXiv preprint arXiv:2306.11565*, 2023 (cit. on p. 3).

[5] J. Lee and H.-J. Yoo, "An overview of energy-efficient hardware accelerators for on-device deep-neural-network training," *IEEE Open Journal of the Solid-State Circuits Society*, vol. 1, pp. 115–128, 2021 (cit. on p. 5).

[6] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 303–312 (cit. on p. 5).

[7] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *SIGGRAPH*, vol. 21, no. 4, pp. 163–169, 1987 (cit. on pp. 5, 14).

[8] R. A. Newcombe *et al.*, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 2011, pp. 127–136 (cit. on pp. 5, 9, 12, 13, 37).

[9] Q. Cao and J. Gu, "A sparse convolution neural network accelerator for 3d/4d point-cloud image recognition on low power mobile device with hopping-index rule book for efficient coordinate management," in *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, IEEE, 2022, pp. 106–107 (cit. on p. 6).

[10] W. Sun *et al.*, "A 28nm 2d/3d unified sparse convolution accelerator with block-wise neighbor searcher for large-scaled voxel-based point cloud network," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, IEEE, 2023, pp. 328–330 (cit. on p. 6).

[11] M. Dahnert, J. Hou, M. Nießner, and A. Dai, "Panoptic 3d scene reconstruction from a single rgb image," in *NeurIPS*, vol. 34, 2021, pp. 8282–8293 (cit. on p. 9).

[12] P. Sun *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *CVPR*, 2020, pp. 2446–2454 (cit. on p. 9).

[13] L. Han, T. Zheng, Y. Zhu, L. Xu, and L. Fang, "Live semantic 3d perception for immersive augmented reality," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 5, pp. 2012–2022, 2020 (cit. on pp. 9, 37).

[14] Z. Jiang, C.-C. Hsu, and Y. Zhu, "Ditto: Building digital twins of articulated objects from interaction," in *CVPR*, 2022, pp. 5616–5626 (cit. on p. 9).

[15] A. Bozic, P. Palafox, M. Zollhofer, J. Thies, A. Dai, and M. Nießner, "Neural deformation graphs for globally-consistent non-rigid reconstruction," in *CVPR*, 2021, pp. 1450–1459 (cit. on pp. 9, 13).

[16] J. Sun, Y. Xie, L. Chen, X. Zhou, and H. Bao, "Neuralrecon: Real-time coherent 3d reconstruction from monocular video," in *CVPR*, 2021, pp. 15 598–15 607 (cit. on pp. 9, 11, 14, 15, 17, 22, 27, 29, 30, 32, 33, 38).

[17] J. Mahmud, T. Price, A. Bapat, and J.-M. Frahm, "Boundary-aware 3d building reconstruction from a single overhead image," in *CVPR*, 2020, pp. 441–451 (cit. on p. 9).

[18] C. A. Vanegas, D. G. Aliaga, and B. Benes, "Building reconstruction using manhattan-world grammars," in *CVPR*, 2010, pp. 358–365 (cit. on p. 9).

[19] A.-Q. Cao and R. de Charette, "Monoscene: Monocular 3d semantic scene completion," in *CVPR*, 2022, pp. 3991–4001 (cit. on p. 9).

[20] C. Li, J. Shi, Y. Wang, and G. Cheng, "Reconstruct from top view: A 3d lane detection approach based on geometry structure prior," in *CVPRW*, 2022, pp. 4370–4379 (cit. on p. 9).

[21] S. Weder, J. Schonberger, M. Pollefeys, and M. R. Oswald, "Routedfusion: Learning real-time depth map fusion," in *CVPR*, 2020, pp. 4887–4897 (cit. on pp. 10, 13, 37).

[22] S. Weder, J. L. Schonberger, M. Pollefeys, and M. R. Oswald, "Neuralfusion: Online depth fusion in latent space," in *CVPR*, 2021, pp. 3162–3172 (cit. on pp. 10, 13).

[23] D. Azinović, R. Martin-Brualla, D. B. Goldman, M. Nießner, and J. Thies, "Neural rgb-d surface reconstruction," in *CVPR*, 2022, pp. 6290–6301 (cit. on pp. 10, 13).

[24] Y. Xu, L. Nan, L. Zhou, J. Wang, and C. C. Wang, "Hrbf-fusion: Accurate 3d reconstruction from rgb-d data using on-the-fly implicits," in *ACM TOG*, vol. 41, ACM New York, NY, 2022, pp. 1–19 (cit. on pp. 10, 13).

[25] C. Sommer, L. Sang, D. Schubert, and D. Cremers, "Gradient-sdf: A semi-implicit surface representation for 3d reconstruction," in *CVPR*, 2022, pp. 6280–6289 (cit. on pp. 10, 13).

[26] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *CVPR*, 2017, pp. 2117–2125 (cit. on pp. 10, 13–15, 27).

[27] Y. LeCun, "1.1 deep learning hardware: Past, present, and future," in *2019 IEEE International Solid-State Circuits Conference (ISSCC)*, IEEE, 2019, pp. 12–19 (cit. on pp. 10, 37).

[28] Z. Hong and C. P. Yue, "Efficient-grad: Efficient training deep convolutional neural networks on edge devices with gradient optimizations," *ACM Transactions on Embedded Computing Systems*, vol. 21, pp. 1–24, 2022 (cit. on pp. 10, 37).

[29] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *CVPR*, 2016, pp. 4104–4113 (cit. on p. 11).

[30] Z. Murez, T. v. As, J. Bartolozzi, A. Sinha, V. Badrinarayanan, and A. Rabinovich, "Atlas: End-to-end 3d scene reconstruction from posed images," in *ECCV*, Springer, 2020, pp. 414–431 (cit. on pp. 11, 13, 15, 22, 28–30, 32, 33).

[31] N. Stier, A. Rich, P. Sen, and T. Höllerer, "Vortx: Volumetric 3d reconstruction with transformers for voxelwise view selection and fusion," in *2021 International Conference on 3D Vision (3DV)*, IEEE, 2021, pp. 320–330 (cit. on pp. 11, 14, 15, 30, 33).

[32] J. Choe, S. Im, F. Rameau, M. Kang, and I. S. Kweon, "Volumefusion: Deep depth fusion for 3d scene reconstruction," in *ICCV*, 2021, pp. 16 086–16 095 (cit. on pp. 11, 14).

[33] A. Rich, N. Stier, P. Sen, and T. Höllerer, "3dvnet: Multi-view depth prediction and volumetric refinement," in *2021 International Conference on 3D Vision (3DV)*, IEEE, 2021, pp. 700–709 (cit. on pp. 11, 14, 15, 23, 30, 32).

[34] Z. Hong and C. P. Yue, "Cross-dimensional refined learning for real-time 3d visual perception from monocular video," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, IEEE/CVF, Oct. 2023, pp. 2169–2178 (cit. on p. 11).

[35] T. Cavallari and L. D. Stefano, "Semanticfusion: Joint labeling, tracking and mapping," in *ECCV*, Springer, 2016, pp. 648–664 (cit. on p. 12).

[36] A. Dai and M. Nießner, "3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation," in *NeurIPS*, 2018, pp. 452–468 (cit. on pp. 12, 30, 33).

[37] M. Jaritz, J. Gu, and H. Su, "Multi-view pointnet for 3d scene understanding," in *ICCVW*, 2019, pp. 0–0 (cit. on p. 12).

[38] G. Narita, T. Seno, T. Ishikawa, and Y. Kaji, "Panopticfusion: Online volumetric semantic mapping at the level of stuff and things," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 4205–4212 (cit. on pp. 12, 38).

[39] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: An open-source library for real-time metric-semantic localization and mapping," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1689–1696, 2020 (cit. on pp. 12, 38).

[40] W. Hu, H. Zhao, L. Jiang, J. Jia, and T.-T. Wong, "Bidirectional projection network for cross dimension scene understanding," in *CVPR*, 2021, pp. 14 373–14 382 (cit. on pp. 13, 30, 33).

[41] D. Menini, S. Kumar, M. R. Oswald, E. Sandström, C. Sminchisescu, and L. Van Gool, "A real-time online learning framework for joint 3d reconstruction and semantic segmentation of indoor scenes," in *IEEE Robotics and Automation Letters*, vol. 7, IEEE, 2021, pp. 1332–1339 (cit. on pp. 13, 37).

[42] S.-S. Huang, H. Chen, J. Huang, H. Fu, and S.-M. Hu, "Real-time globally consistent 3d reconstruction with semantic priors," *IEEE Transactions on Visualization & Computer Graphics*, vol. 01, pp. 1–1, 2021 (cit. on p. 13).

[43] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *CVPR*, 2019, pp. 165–174 (cit. on pp. 13, 41).

[44] Z. Yu, S. Peng, M. Niemeyer, T. Sattler, and A. Geiger, "Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction," in *NeurIPS*, 2022 (cit. on p. 13).

[45] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche, "Monofusion: Real-time 3d reconstruction of small scenes with a single web camera," in *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 2013, pp. 83–88 (cit. on p. 13).

[46] A. Bozic, P. Palafox, J. Thies, A. Dai, and M. Nießner, "Transformerfusion: Monocular rgb scene reconstruction using transformers," in *CVPR*, vol. 34, 2021, pp. 1403–1414 (cit. on p. 14).

[47] A. Vaswani *et al.*, "Attention is all you need," in *NeurIPS*, vol. 30, 2017 (cit. on p. 14).

[48] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2015, pp. 234–241 (cit. on p. 14).

[49] M. Sayed, J. Gibson, J. Watson, V. Prisacariu, M. Firman, and C. Godard, "Simplerecon: 3d reconstruction without 3d convolutions," in *ECCV*, 2022 (cit. on pp. 14, 17, 30, 32).

[50] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan, "Mvsnet: Depth inference for unstructured multi-view stereo," in *ECCV*, 2018, pp. 767–783 (cit. on pp. 14, 17).

[51] X. Gu, Z. Fan, S. Zhu, Z. Dai, F. Tan, and P. Tan, "Cascade cost volume for high-resolution multi-view stereo and stereo matching," in *ICCV*, 2020, pp. 2495–2504 (cit. on p. 15).

[52] K. Wang and S. Shen, "Mvdepthnet: Real-time multiview depth estimation neural network," in *2018 International Conference on 3D Vision (3DV)*, IEEE, 2018, pp. 248–257 (cit. on pp. 15, 27, 38).

[53] L. Roldao, R. De Charette, and A. Verroust-Blondet, "3d semantic scene completion: A survey," *IJCV*, vol. 130, no. 8, pp. 1978–2005, 2022 (cit. on p. 17).

[54] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016 (cit. on p. 20).

[55] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," *NeurIPS*, vol. 31, 2018 (cit. on pp. 20, 22).

[56] I. J. Goodfellow, O. Vinyals, and A. M. Saxe, "Qualitatively characterizing neural network optimization problems," *ICLR*, 2015 (cit. on pp. 20, 22).

[57] R. Chen, S. Han, J. Xu, and H. Su, "Point-based multi-view stereo network," in *ICCV*, 2019, pp. 1538–1547 (cit. on p. 23).

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778 (cit. on p. 27).

[59] PyTorch, *Docs - models and pre-trained weights - mnasnet*, https://pytorch.org/vision/main/models/mnasnet.html (Dec. 2023) (cit. on p. 27).

[60] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR*, 2018 (cit. on p. 27).

[61] M. Tan *et al.*, "Mnasnet: Platform-aware neural architecture search for mobile," in *CVPR*, 2019, pp. 2820–2828 (cit. on p. 27).

[62] C. Choy, J. Gwak, and S. Savarese, "4d spatio-temporal convnets: Minkowski convolutional neural networks," in *CVPR*, 2019, pp. 3075–3084 (cit. on p. 27).

[63] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *CVPR*, 2017, pp. 5828–5839 (cit. on pp. 30, 32).

[64] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung, "Scenenn: A scene meshes dataset with annotations," in *2016 Fourth International Conference on 3D Vision (3DV)*, IEEE, 2016, pp. 92–101 (cit. on p. 30).

[65] Q.-H. Pham, B.-S. Hua, T. Nguyen, and S.-K. Yeung, "Real-time progressive 3d semantic segmentation for indoor scenes," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2019, pp. 1089–1098 (cit. on pp. 37, 38).

[66] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021 (cit. on p. 38).

[67] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction," *arXiv preprint arXiv:2106.10689*, 2021 (cit. on p. 38).

[68] Apple Inc., *Arkit developer document*, https://developer.apple.com/documentation/arkit (Dec. 2023) (cit. on p. 39).

[69] Alpha Inc., *Overview of arcore and dev*, https://developers.google.com/ar/develop (Dec. 2023) (cit. on p. 39).

[70] Shogo4405, *Haishinkit for ios, macos, tvos, visionos and android*, https://github.com/shogo4405/HaishinKit.swift (Dec. 2023) (cit. on p. 39).

[71] A. Gutierrez, *Docker-nginx-rtmp*, https://github.com/alfg/docker-nginx-rtmp (Dec. 2023) (cit. on p. 40).

[72] H. Chu and A. Stepanchuk, *Rtmp dump v2.4*, https://github.com/ossrs/librtmp (Dec. 2023) (cit. on p. 40).

[73] Y. Hou, J. Kannala, and A. Solin, "Multi-view stereo by temporal nonparametric fusion," in *ICCV*, 2019, pp. 2651–2660 (cit. on p. 41).

[74] Y. Cheng, G. Li, N. Wong, H.-B. Chen, and H. Yu, "Deepeye: A deeply tensor-compressed neural network for video comprehension on terminal devices," *ACM Transactions on Embedded Computing System*, vol. 19, no. 3, May 2020 (cit. on p. 42).

[75] N. Kukreja *et al.*, "Training on the edge: The why and the how," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019, pp. 899–903 (cit. on pp. 42, 48).

[76] Z. Hong *et al.*, "Electromagnetic pattern extraction and grouping for near-field scanning of integrated circuits by pca and k-means approaches," *IEEE Transactions on Electromagnetic Compatibility*, vol. 61, no. 6, pp. 1811–1822, 2019 (cit. on p. 42).

[77] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019 (cit. on pp. 42, 49, 56, 57, 60).

[78] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, Jan. 2019 (cit. on pp. 42, 43).

[79] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," in *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., vol. 1, 2019, pp. 374–388 (cit. on p. 43).

[80] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 54, PMLR, 2017, pp. 1273–1282 (cit. on p. 43).

[81] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14 (cit. on p. 43).

[82] Z. Hong and C. P. Yue, "Efficient training convolutional neural networks on edge devices with gradient-pruned sign-symmetric feedback alignment," in *IT Convergence and Security (ICITCS)*, H. Kim and K. J. Kim, Eds., Singapore: Springer Singapore, 2021, pp. 13–22 (cit. on p. 44).

[83] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986 (cit. on p. 46).

[84] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, Berlin, Heidelberg: Springer-Verlag, 1998, pp. 9–50 (cit. on p. 46).

[85] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random feedback weights support learning in deep neural networks," *Nature Communications*, 2016 (cit. on pp. 47, 48, 51, 53, 54).

[86] P. Baldi, P. Sadowski, and Z. Lu, "Learning in the machine: Random backpropagation and the deep learning channel," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2019, pp. 6348–6352 (cit. on p. 47).

[87] A. Nøkland, "Direct feedback alignment provides learning in deep neural networks," in *Advances in Neural Information Processing Systems 29*, Curran Associates, Inc., 2016, pp. 1037–1045 (cit. on p. 48).

[88] C. Frenkel, M. Lefebvre, and D. Bol, "Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks," *Frontiers in Neuroscience*, vol. 15, p. 20, 2021 (cit. on p. 48).

[89] C. Frenkel, J.-D. Legat, and D. Bol, "A 28-nm convolutional neuromorphic processor enabling online learning with spike-based retinas," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5 (cit. on pp. 48, 49).

[90] M. Refinetti, S. d'Ascoli, R. Ohana, and S. Goldt, *The dynamics of learning with feedback alignment*, 2020 (cit. on p. 48).

[91] Z. Huo, B. Gu, and H. Huang, "Training neural networks using features replay," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018 (cit. on p. 48).

[92] M. Jaderberg *et al.*, "Decoupled neural interfaces using synthetic gradients," in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, Jun. 2017, pp. 1627–1635 (cit. on p. 48).

[93] D. Han, J. Lee, J. Lee, and H.-J. Yoo, "A low-power deep neural network online learning processor for real-time object tracking application," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 5, pp. 1794–1804, 2019 (cit. on pp. 48, 49, 56, 65, 66, 68, 70).

[94] Y. Wang *et al.*, "E2-train: Training state-of-the-art cnns with over 80% energy savings," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019 (cit. on p. 48).

[95] Y. Wu, Z. Wang, Y. Shi, and J. Hu, "Enabling on-device cnn training by self-supervised instance filtering and error map pruning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3445–3457, 2020 (cit. on p. 48).

[96] T. Chen *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," ser. ASPLOS '14, Salt Lake City, Utah, USA: Association for Computing Machinery, 2014, pp. 269–284 (cit. on pp. 49, 56).

[97] Y. Chen *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47, Cambridge, United Kingdom: IEEE Computer Society, 2014, pp. 609–622 (cit. on p. 49).

[98] Z. Du *et al.*, "Shidiannao: Shifting vision processing closer to the sensor," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15, Portland, Oregon: Association for Computing Machinery, 2015, pp. 92–104 (cit. on pp. 49, 60).

[99] S. Yin *et al.*, "A high energy efficient reconfigurable hybrid neural network processor for deep learning applications," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 968–982, 2018 (cit. on p. 49).

[100] Z. Yuan *et al.*, "Sticker: A 0.41-62.1 tops/w 8bit neural network processor with multi-sparsity compatible convolution arrays and online tuning acceleration for fully connected layers," in *2018 IEEE Symposium on VLSI Circuits*, 2018, pp. 33–34. DOI: 10.1109/VLSIC.2018.8502404 (cit. on p. 49).

[101] G. Lee, H. Park, N. Kim, J. Yu, S. Jo, and K. Choi, "Acceleration of dnn backward propagation by selective computation of gradients," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6 (cit. on pp. 49, 65).

[102] P. Dai *et al.*, "Sparsetrain: Exploiting dataflow sparsity for efficient convolutional neural networks training," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6 (cit. on pp. 49, 52, 59).

[103] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "7.7 lnpu: A 25.3tflops/w sparse deep-neural-network learning processor with fine-grained mixed precision of fp8-fp16," in *2019 IEEE International Solid- State Circuits Conference (ISSCC)*, 2019, pp. 142–144 (cit. on p. 49).

[104] D. Han, J. Lee, and H.-J. Yoo, "Df-lnpu: A pipelined direct feedback alignment-based deep neural network learning processor for fast online learning," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 5, pp. 1630–1640, 2021 (cit. on p. 49).

[105] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778 (cit. on pp. 50, 53).

[106] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Sep. 2019, pp. 6105–6114 (cit. on p. 50).

[107] K. Bong, S. Choi, C. Kim, D. Han, and H.-J. Yoo, "A low-power convolutional neural network face recognition processor and a cis integrated with always-on face detector," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 1, pp. 115–123, 2018 (cit. on p. 50).

[108] C.-H. Lu, Y.-C. Wu, and C.-H. Yang, "A 2.25 tops/w fully-integrated deep cnn learning processor with on-chip training," in *2019 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2019, pp. 65–68 (cit. on pp. 50, 68).

[109] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 448–456 (cit. on pp. 52, 54).

[110] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12 (cit. on pp. 55, 60).

[111] H. Genc *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proceedings of the 58th Annual Design Automation Conference (DAC)*, 2021, pp. 1–6 (cit. on p. 56).

[112] C. Su, S. Zhou, L. Feng, and W. Zhang, "Towards high performance low bitwidth training for deep neural networks," *Journal of Semiconductors*, vol. 41, no. 2, p. 022 404, Feb. 2020 (cit. on pp. 58, 61).

[113] M. Wang, S. Rasoulinezhad, P. H. W. Leong, and H. K. H. So, *Niti: Training integer neural networks using integer-only arithmetic*, 2020 (cit. on p. 58).

[114] S. Choi, J. Sim, M. Kang, Y. Choi, H. Kim, and L.-S. Kim, "An energy-efficient deep convolutional neural network training accelerator for in situ personalization on smart devices," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 10, pp. 2691–2702, 2020 (cit. on pp. 60, 68–70).

[115] K. Asanović *et al.*, "The rocket chip generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr. 2016 (cit. on p. 62).

[116] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020 (cit. on p. 62).

[117] X. Chen, X. Hu, H. Zhou, and N. Xu, "Fxpnet: Training a deep convolutional neural network in fixed-point representation," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2494–2501 (cit. on p. 62).

[118] S. Choi, J. Shin, Y. Choi, and L.-S. Kim, "An optimized design technique of low-bit neural network training for personalization on iot devices," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6 (cit. on p. 63).

[119] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009 (cit. on p. 63).

[120] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255 (cit. on p. 63).

[121] J. Bachrach *et al.*, "Chisel: Constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12, San Francisco, California: Association for Computing Machinery, 2012, pp. 1216–1225 (cit. on p. 64).

[122] C. Papon. "Scala based hdl v1.4.0." (Mar. 2020), [Online]. Available: https://github.com/SpinalHDL/SpinalHDL (cit. on p. 64).

[123] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An introduction to high-level synthesis," *IEEE Design Test of Computers*, vol. 26, no. 4, pp. 8–17, 2009 (cit. on p. 64).

[124] UCB-BAR, *Chiseltest, a test harness for chisel-based rtl designs*, https://github.com/ucb-bar/chiseltest (Dec. 2023) (cit. on p. 64).

[125] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques," in *ICCAD: International Conference on Computer-Aided Design*, 2011, pp. 694–701 (cit. on p. 64).

[126] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 58–68 (cit. on p. 64).

[127] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020 (cit. on p. 64).

[128] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011 (cit. on p. 64).

[129] K. Chandrasekar, C. Weis, B. Akesson, N. Wehn, and K. Goossens, "Towards variation-aware system-level power estimation of drams: An empirical approach," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–8 (cit. on p. 64).

[130] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2220–2233, 2017 (cit. on p. 65).

[131] J. Park, S. Lee, and D. Jeon, "9.3 a 40nm 4.81tflops/w 8b floating-point training processor for non-sparse neural networks using shared exponent bias and 24-way fused multiply-add tree," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 1–3 (cit. on pp. 68, 70).

A List of Publications

Journal Publications

[1] Z. Hong and C. P. Yue, "Real-time 3d monocular perception by cross-dimensional refined learning for robotics," in *IEEE Transactions on Circuits and Systems for Video Technology*, Under Major Revision, 2023.

[2] Z. Hong and C. P. Yue, "Efficient-grad: Efficient training deep convolutional neural networks on edge devices with gradient optimizations," *ACM Transactions on Embedded Computing Systems*, vol. 21, pp. 1–24, 2022.

[3] Z. Hong *et al.*, "Electromagnetic pattern extraction and grouping for near-field scanning of integrated circuits by pca and k-means approaches," *IEEE Transactions on Electromagnetic Compatibility*, vol. 61, no. 6, pp. 1811–1822, 2019.

[4] C. Luo *et al.*, "Collocated and simultaneous measurements of rf current and voltage on a trace in a noncontact manner," *IEEE Transactions on Microwave Theory and Techniques*, vol. 67, no. 6, pp. 2406–2415, 2019.

Conference Publications

[1] Z. Hong and C. P. Yue, "Cross-dimensional refined learning for real-time 3d visual perception from monocular video," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, IEEE/CVF, Oct. 2023, pp. 2169–2178.

[2] Z. Hong and C. P. Yue, "Efficient training convolutional neural networks on edge devices with gradient-pruned sign-symmetric feedback alignment," in *IT Convergence and Security (ICITCS)*, H. Kim and K. J. Kim, Eds., Singapore: Springer Singapore, 2021, pp. 13–22.

[3] C. P. Yue, H. C. Cheng, and Z. Hong, "Enabling technologies for multi-robot human collaboration," in *Proceedings of the 2023 International Workshop on Signal Processing and Machine Learning (WSPML)*, Sep. 2023.

[4] H. C. Cheng, B. Hussain, Z. Hong, and C. P. Yue, "Leveraging 360° camera in 3d reconstruction: A vision-based approach," in *Proceedings of the 2nd International Conference on Video and Signal Processing (ICVSP)*, Oct. 2023.