Efficient Deep Learning with Sparsity: Algorithms, Systems, and Applications

by

Zhijian Liu

B.Eng., Shanghai Jiao Tong University (2018)
S.M., Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by: Zhijian Liu
Department of Electrical Engineering and Computer Science
May 17, 2024

Certified by: Song Han
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Efficient Deep Learning with Sparsity: Algorithms, Systems, and Applications

by

Zhijian Liu

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

Deep learning has been used across a broad spectrum of applications, including computer vision, natural language processing, and scientific discovery. However, behind its remarkable performance lies an increasing gap between the demand for and supply of computation. On the demand side, the computational costs of deep neural networks have surged dramatically, driven by ever-larger input and model sizes. On the supply side, as Moore's Law slows down, hardware no longer delivers increasing performance within the same power budget.

In this dissertation, we present our solutions across the algorithm, system, and application stacks to address the demand-supply gap through the lens of sparsity. In Part I, we first develop algorithms, SparseViT and SparseRefine, which identify sparsity within dense input data. We then introduce new sparse primitives, PVCNN and FlatFormer, to efficiently process inputs with sparsity. In Part II, we introduce system libraries, TorchSparse, to optimize existing sparse primitives and effectively translate theoretical savings from sparsity into practical speedups on hardware. In Part III, we apply sparsity to accelerate a wide range of computation-intensive AI applications, such as autonomous driving and language modeling. We conclude this dissertation with a vision towards building more efficient and accessible AI.

Thesis supervisor: Song Han
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

My deepest gratitude goes to my advisor, Song Han. I have been incredibly fortunate to have Song as my PhD advisor for the past six years. A good advisor is full of interesting research ideas, knowledgeable, encouraging, insightful, and supportive. Song embodies all these qualities to the fullest extent. I still remember him telling me that pursuing a PhD is about the experience. With him, I have had the privilege of working on impactful research, getting a taste of the startup world, and so much more. Song is the best role model for me in both research and life, and I know I still have much to learn from him.

I want to extend my sincere appreciation to all my thesis committee members, Kurt Keutzer, Marco Pavone, and Justin Solomon. They have gone above and beyond their roles as committee members. Kurt and Marco have shared invaluable career advice and have been incredibly supportive of my job search. Justin's early work on DGCNN sparked my interest in the field of 3D deep learning and later sparsity.

I also deeply appreciate my undergraduate advisors, Cewu Lu, for introducing me to the world of AI, and Yong Yu, for advising me during my undergraduate study. I would like to thank Jiajun Wu, Jun-Yan Zhu, Chen Sun, Kevin P. Murphy, Antonio Torralba, William T. Freeman, and Joshua B. Tenenbaum for mentoring me during my research internship at MIT.

Before I started my PhD, I heard from many people that it would be a lonely journey, but luckily, my experience has proven otherwise. I would like to express my appreciation for the inspiration and support from my labmates: Han Cai, Wei-Ming Chen, Muyang Li, Ji Lin, Yujun Lin, Haotian Tang, Jiaming Tang, Hanrui Wang, Wei-Chen Wang, Shang Yang, Guangxuan Xiao, Zhekai Zhang, and Ligeng Zhu. Their insightful discussions and encouragement have been invaluable throughout this journey.

During my job search, I was extremely fortunate to receive support from many people. I especially want to thank Saman Amarasinghe, Remzi Arpaci-Dusseau, Deming Chen, Tianlong Chen, Tianqi Chen, Roger Chen, Chuang Gan, Philip Harris, Mark Horowitz, Qixing Huang, Shih-Chieh Hsu, Richard Lethin, Pan Li, Yunzhu Li, Wei-Chiu Ma, Lianhui Qin, Karu Sankaralingam, Cong Wang, Xuhai Xu, Yuzhe Yang, Xiangyao Yu, Shenlong

Wang, Yue Wang, Zhaoran Wang, and Yue Zhao for their encouragement, which helped me through this stressful process.

Part of this research is funded by National Science Foundation (NSF), NVIDIA, Intel, Qualcomm, Arm, Xilinx (now part of AMD), Semiconductor Research Corporation (SRC), Apple, Google, Amazon, Meta, IBM, Adobe, Samsung, Sony, Ford, and Hyundai. I thank the generous support from all the funding agencies that made this dissertation possible.

I would like to thank my furry friends: a lovely dog, Sudo, and two cute chinchillas, Mochi and Mocha. They have brought so much joy into my life. Sudo is adorable but often distracts me to play with her; without her, I might have finished this dissertation sooner.

I would also like to thank my mother, Lili Yang, for her unconditional love and support. She has always had strong confidence in me and stood by me through all the ups and downs. Without her, I would not have come this far.

The greatest treasure during my PhD has been getting to know my partner, Keting Yang. She is always energetic and brings so much love and joy into my life. Thank you for supporting and completing me, and I look forward to many many more together!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past decade or so, deep learning has revolutionized various domains, such as computer vision [1, 2], natural language processing [3, 4], speech recognition [5, 6], and scientific discovery [7, 8]. It has also been the driving force behind recent AI applications such as large language models like ChatGPT [9] and Gemini [10], as well as generative AI models like DALL·E [11] and Sora [12]. However, behind such impressive performance lies an ever-increasing gap between the **demand for** and **supply of** computation.

On the demand side, the computational costs of deep neural networks have surged dramatically. LeNet [13], one of the first "deep" neural networks published in 1989, had exactly *1 thousand* parameters and processed images with *256* pixels (16×16). In contrast, the largest vision transformer model, ViT-22B [14], now contains over *20 billion* parameters, and we can easily capture images with *200 million* pixels using our mobile phones (*e.g.*, Samsung Galaxy S24 Ultra). This represents a **7 orders of magnitude** increase in model size and a **6 orders of magnitude** increase in input size over the past 35 years. A similar, if not faster, trend is observed in the language domain. Model sizes have grown from around *100 million* parameters in GPT-1 [15] to more than *100 billion* parameters in GPT-3 [16], while context lengths have increased from *512* tokens in GPT-1 [15] to *1 million* tokens in Gemini 1.5 [10] over the past five years. This increased computational complexity drives the improved performance of these models, a phenomenon known as the "scaling law".

On the supply side, the slowdown of Moore's Law has resulted in hardware no longer delivering increased performance within the same power budget. When deploying deep neural networks to edge devices, such as mobile phones, this challenge is exacerbated by constraints such as form factors, battery life, and heat dissipation. For instance, users of mixed-reality headsets should not be expected to carry a backpack full of computers, and self-driving vehicles cannot accommodate a trunk filled with workstations. These

constraints impose significant limitations on the hardware that can be integrated into these devices for their specific applications, thereby further restricting their computational capabilities.

In summary, the **exponential** growth in demand coupled with the **constrained** growth in supply creates a significant and also widening gap for deep learning computing. This expanding disparity poses a substantial risk to the advancement and broad adoption of these resource-intensive yet powerful deep learning models. Therefore, this dissertation aims to investigate and propose solutions to bridge this critical demand-supply gap.

Sparsity

This dissertation will explore my research efforts to bridge the demand-supply gap through the lens of sparsity. To set the foundation, we will address three questions: "**What** is sparsity", "**Where** can we find sparsity?", and "**Why** is sparsity the right approach?".

What? According to the definition by the Merriam-Webster dictionary, "sparsity" refers to the presence of *few* and *scattered* elements. When applied to a matrix, sparsity indicates that the matrix contains many zero elements, with the non-zero elements dispersed sporadically. In the remainder of this dissertation, we will adhere to this specific definition of "sparsity". However, many techniques and approaches discussed herein are also applicable to broader interpretations of "sparsity", such as lower precision or rank.

Where? Any deep learning workload consists of a model and the input data it processes. Model weights and input activations are essentially matrices with many values. Despite their large size, these matrices contain significant redundancy, as not every parameter in the model or bit of information in the input is necessary. By zeroing out these redundant parameters (in the model/weights) or bits (in the input/activations), we can introduce computation workloads with substantial sparsity. During the early stages of my PhD, I explored model/weight sparsity with a focus on automating model compression [17–20]. This dissertation, however, will focus on my research efforts in input/activation sparsity.

Why? Efficiency can be considered the art of strategic "laziness" — for example, by rejecting, reusing, and postponing computation. Sparsity exemplifies this principle by enabling the elimination of redundant computations. Given that deep neural networks heavily rely on multiplication, and any number multiplied by zero remains zero, these

zero-valued elements do not impact the final output. Ideally, bypassing all computations involving these zero elements can result in significant computational savings and substantial acceleration potential.

Approach

To fully unlock the efficiency potential of deep learning, it is essential to work across the entire stack, from applications to algorithms and systems, down to the hardware. This dissertation will focus on the first three components: developing efficient algorithms, providing appropriate system support, and deeply integrating them with applications. This *cross-stack co-design* approach will offer a holistic perspective on the problem, enabling us to make compromises at the best stack, as some challenges can be more effectively resolved at the algorithm level, while others are better addressed at the system level.

Algorithm. Unlike conventional computational workloads, deep neural networks are surprisingly resilient to *relaxations* and *approximations*. One example is sparsity, as discussed in the previous section, which skips computations that do not contribute significantly to the outcome. The goal at the algorithm level is to identify such relaxations and approximations that can reduce computational complexity while maintaining the same overall performance. For sparsity, we need to find redundancies either within the dense neural network or the dense input data. As mentioned earlier, this dissertation will focus on the latter.

System. Theoretical savings from algorithmic relaxations do not always translate into actual speed gains on hardware. For instance, modern parallel hardware prefers *regular* and *balanced* workloads, while sparsity often creates *unbalanced* and *irregular* workloads. Effective system support acts as a bridge between the algorithm and the hardware. The goal at the system level is to implement the same workload in a way that is more compatible with the hardware. Alternatively, the workload itself can also be redefined to be more hardware-friendly to begin with. This dissertation will explore both approaches.

Application. Generic algorithms and systems are powerful because they can be applied to a wide range of applications out of the box. However, their benefits will eventually saturate and diminish. Fortunately, ample opportunities exist in the *application-specific* domain, where significant acceleration can potentially be achieved by closely examining the specific model in use, the data being processed, and the manner in which these models

are used. The goal at the application level is to integrate both generic and domain-specific solutions. We will present several examples of such integration in this dissertation.

Dissertation Structure

This dissertation is structured into three parts, all centered on the theme of accelerating deep learning through sparsity. Each part addresses a different layer of the stack: algorithms, systems, and applications.

- **Part I is about algorithms**.

  In Chapter 2, we discuss algorithms for identifying sparsity within dense data. We explore two approaches: SparseViT, which progressively prunes the least important bits from the data, and SparseRefine, which initially performs fast inference with downscaled data and subsequently refines the least certain bits sparsely. SparseViT was previously published as Chen *et al*. [21], while SparseRefine is under review at the time of writing this dissertation.

  In Chapter 3, we introduce new primitives designed to efficiently process inputs with sparsity. We explore two approaches: PVCNN, which proposes a hybrid sparse-dense primitive combining the memory efficiency of sparse representations with the computational regularity of dense convolutions, and FlatFormer, which introduces flattened sparse window attention, partitioning the sparse input into groups of equal sizes rather than windows of equal shapes, thereby avoiding the uneven workload problem. PVCNN was previously published as Liu *et al*. [22, 23], and FlatFormer was previously published as Liu *et al*. [24].

- **Part II is about systems**.

  In Chapter 4, we optimize the system implementation of existing sparse primitives to improve their hardware efficiency. We primarily discuss TorchSparse, which groups workloads of similar sizes from different weights to improve computational regularity while controlling padding overhead. TorchSparse was previously published as Tang *et al*. [25].

- **Part III is about applications**.

  In Chapter 5, we apply sparsity to accelerate 3D perception for autonomous driving. We focus on BEVFusion, which unifies sparse LiDAR data and dense camera data

in a shared bird's-eye view space to efficiently support diverse perception tasks. BEVFusion was previously published as Liu *et al*. [26].

In Chapter 6, we apply sparsity to accelerate language modeling for natural language processing. We focus on Lite Transformer, which proposes a hybrid convolution-attention primitive that combines convolution's local context modeling with attention's long-distance relationship modeling capabilities. Lite Transformer was previously published as Wu *et al*. [27].

We conclude and discuss future directions in Chapter 7.

Part I

Algorithms

Chapter 2

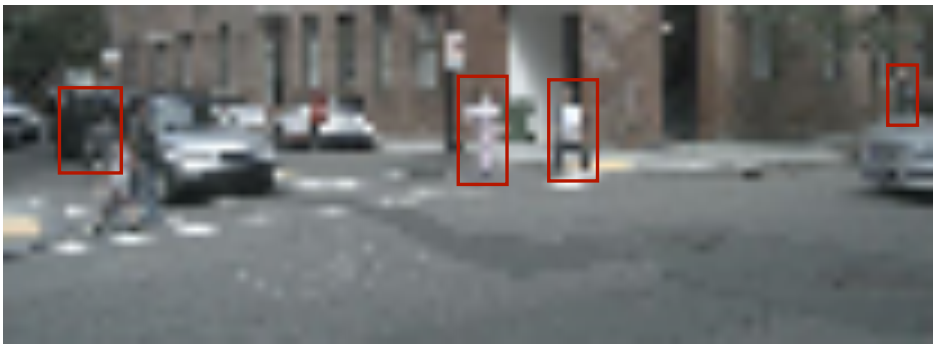
Finding Sparsity from Dense Data

2.1 SparseViT

High-resolution images enable neural networks to learn richer visual representations. However, this improved performance comes at the cost of growing computational complexity, hindering their usage in latency-sensitive applications. As not all pixels are equal, skipping computations for less-important regions offers a simple and effective measure to reduce the computation. This, however, is hard to be translated into actual speedup for CNNs since it breaks the regularity of the dense convolution workload. We introduce SparseViT that revisits activation sparsity for recent window-based vision transformers (ViTs). As window attentions are naturally batched over blocks, actual speedup with window activation pruning becomes possible: *i.e.*, $\sim$50% latency reduction with 60% sparsity. Different layers should be assigned with different pruning ratios due to their diverse sensitivities and computational costs. We introduce sparsity-aware adaptation and apply the evolutionary search to efficiently find the optimal layerwise sparsity configuration within the vast search space. SparseViT achieves speedups of **1.5**$\times$, **1.4**$\times$, and **1.3**$\times$ compared to its dense counterpart in monocular 3D object detection, 2D instance segmentation, and 2D semantic segmentation, respectively, with negligible to no loss of accuracy.

2.1.1 Introduction

With the advancement of image sensors, high-resolution images become more and more accessible: *e.g.*, recent mobile phones are able to capture 100-megapixel photos. The increased image resolution offers great details and enables neural network models to learn richer visual representations and achieve better recognition quality. This, however, comes at the cost of linearly-growing computational complexity, making them less deployable for



(a) **Direct Downsample**: Lower Resolution (**0.5x**), Dense (**100%**)

(b) **Window Activation Pruning**: Higher Resolution (**1.0x**), Sparse (**25%**)

Figure 2.1: *Sparse, high-resolution* features are far more informative than *dense, low-resolution* ones. Compared with direct downsampling, activation pruning can retain important details at a higher resolution, which is essential for most image recognition tasks.

resource-constrained applications (*e.g.*, mobile vision, autonomous driving).

The simplest solution to address this challenge is to downsample the image to a lower resolution. However, this will drop the fine details captured from the high-resolution sensor. What a waste! The missing information will bottleneck the model's performance upper bound, especially for small object detection and dense prediction tasks. For instance, the detection accuracy of a monocular 3D object detector will degrade by more than 5% in mAP by reducing the height and width by 1.6×*. Such a large gap cannot be easily recovered by scaling the model capacity up.

Dropping details uniformly at all positions is clearly sub-optimal as not all pixels are equally informative (Figure 2.1a). Within an image, the pixels that contain detailed object features are more important than the background pixels. Motivated by this, a very natural idea is to skip computations for less-important regions (*i.e.*, activation pruning). However, activation sparsity cannot be easily translated into the actual speedup on general-purpose

*BEVDet [28] (with Swin Transformer [29] as backbone) achieves 31.2 mAP with 256×704 resolution and 25.90 mAP with 160×440 resolution.

hardware (*e.g.*, GPU) for CNNs. This is because sparse activation will introduce randomly distributed and unbalanced zeros during computing and cause computing unit under-utilization [30]. Even with dedicated system support [31], a high sparsity is typically required to realize speedup, which will hurt the model's accuracy.

Recently, 2D vision transformers (ViTs) have achieved tremendous progress. Among them, Swin Transformer [29] is a representative work that generalizes well across different visual perception tasks (such as image classification, object detection, and semantic segmentation). We revisit the activation sparsity in the context of window-based ViTs. Different from convolutions, window attentions are naturally batched over windows, making real speedup possible with window-level activation pruning. We re-implement the other layers in the model (*i.e.*, `FFN`s and `LN`s) to also execute at the window level. As a result, we are able to achieve around 50% latency reduction with 60% window activation sparsity.

Within a neural network, different layers have different impacts on efficiency and accuracy, which advocates for a non-uniform layerwise sparsity configuration: *e.g.*, we may prune layers with larger computation and lower sensitivity more, while pruning layers with smaller computation and higher sensitivity less. To this end, we make use of the evolutionary search to explore the best per-layer pruning ratios under a resource constraint. We also propose *sparsity-aware adaptation* by randomly pruning a different subset of the activations at each iteration. This effectively adapts the model to activation sparsity and avoids the expensive re-training of every candidate within the large search space. Our SparseViT achieves speedups of **1.5**×, **1.4**×, and **1.3**× compared to its dense counterpart in monocular 3D object detection, 2D instance segmentation, and 2D semantic segmentation, respectively, with negligible to no loss of accuracy.

2.1.2 Related Work

Vision Transformers. Transformers [3] have revolutionized natural language processing (NLP) and are now the backbone of many large language models (LLMs) [4]. Inspired by their success, researchers have explored the use of transformers in a range of visual recognition tasks [32]. ViT [33] was the first work in this direction, demonstrating that an image can be divided into 16×16 words and processed using multi-head self-attention. DeiT [34] improves on ViT's data efficiency. T2T-ViT [35], Pyramid ViT [36, 37], and CrossFormer [38] introduce hierarchical modeling capability to ViTs. Later, Swin Transformer [29, 39] applies self-attention to non-overlapping windows and proposes window shifting to enable cross-window feature communication. There have also been extensive

studies on task-specific ViTs, such as ViTDet [40] for object detection, and SETR [41] and SegFormer [42] for semantic segmentation.

Model Compression. As the computational cost of neural networks continues to escalate, researchers are actively investigating techniques for model compression and acceleration [17, 43]. One approach is to design more efficient neural network architectures, either manually [44–48] or using automated search [23, 49–53]. These methods are able to achieve comparable performance to ResNet [2] with lower computational cost and latency. Another active direction is neural network pruning, which involves removing redundant weights at different levels of granularity, ranging from unstructured [43, 54] to structured [55, 56]. Although unstructured pruning can achieve higher compression ratios, the lower computational cost may not easily translate into measured speedup on general-purpose hardware and requires specialized hardware support. Low-bit weight and activation quantization is another approach that has been explored to reduce redundancy and speed up inference [2, 18, 19, 57].

Activation Pruning. Activation pruning differs from static weight pruning as it is dynamic and input-dependent. While existing activation pruning methods typically focus on reducing memory cost during training [58–60], few of them aim to improve inference latency as activation sparsity does not always lead to speedup on hardware. To overcome this, researchers have explored adding system support for activation sparsity [25, 31, 61]. However, these libraries often require extensive engineering efforts and high sparsity rates to achieve measurable speedup over dense convolutions.

Efficient ViTs. Several recent works have explored different approaches to improve the efficiency of ViTs. For instance, MobileViT [62] combines CNN and ViT by replacing local processing in convolutions with global processing using transformers. MobileFormer [63] parallelizes MobileNet and Transformer with a two-way bridge for feature fusing, while NASViT [64] leverages neural architecture search to find efficient ViT architectures. Other works have focused on token pruning for ViTs [65–72]. However, these approaches mainly focus on token-level pruning, which is finer-grained than window pruning.

2.1.3 Method

In this section, we first briefly revisit Swin Transformer and modify its implementation so that all layers are applied to windows. We then introduce how to incorporate the

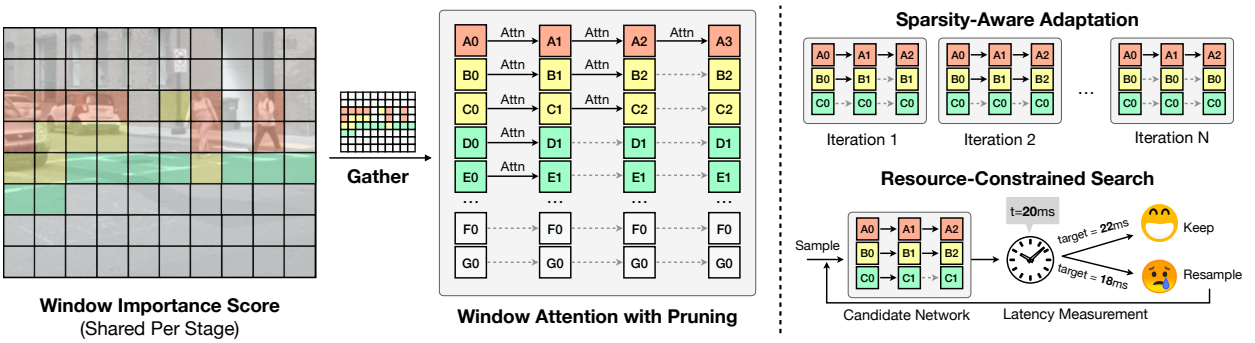


Figure 2.2: Overview of SparseViT. **(Left)** SparseViT first computes the L2 norm of each window activation as its importance score. After that, it first gathers the features from the windows with the highest importance scores, then runs self-attention on selected windows, and finally scatter the results back. **(Right upper)** SparseViT leverages sparsity-aware adaptation that samples a different layerwise activation sparsity at each training iteration to accommodate the activation sparsity. **(Right lower)** SparseViT makes use of evoluationary search to explore the best layerwise sparsity configuration given a latency constraint.

window activation sparsity into the model. Finally, we describe an efficient algorithm (based on sparsity-aware adaptation and evolutionary search) to find the layerwise sparsity ratio.

Swin Transformer Revisited

Swin Transformer [29] applies multi-head self-attention (MHSA) to extract local features within non-overlapping image windows (*e.g.*, 7×7). The transformer design follows a standard approach, involving layer normalization (LN), MHSA, and a feed-forward layer (FFN) applied to each window. Swin Transformer uses a shifted window approach that alternates between two different window partitioning configurations to introduce cross-window connections efficiently.

Window-Wise Execution. The original implementation of Swin Transformer applies MHSA at the window level, while FFN and LN are applied to the entire feature map. This mismatch between the two operations requires additional structuring before and after each MHSA, making window pruning more complicated as the sparsity mask must be mapped from the window level to the feature map. To simplify this process, we modify the execution of FFN and LN to be window-wise as well. This means that all operations will be applied at the window level, making the mapping of the sparsity mask very easy. In practice, this modification incurs a minimal accuracy drop of less than 0.1% due to padding, even

Backbone	Resolution	Width	#MACs (G)	Latency (ms)	mAP (↑)	ATE (↓)	ASE (↓)	AOE (↓)	AVE (↓)	AAE (↓)	NDS (↑)
Swin-T	256×704	1×	140.8	36.4	31.2	69.1	27.2	52.3	90.9	24.7	39.2
SparseViT	288×792	1×	113.9	34.5	**32.0**	72.8	27.2	53.8	79.4	25.7	**40.1**
Swin-T	224×616	1×	78.5	23.0	29.9	71.8	27.4	60.9	79.0	26.0	38.4
Swin-T	256×704	0.6×	56.0	22.6	29.9	69.9	27.5	59.9	81.4	25.8	38.5
SparseViT	256×704	1×	78.4	23.8	**31.2**	70.9	27.5	58.7	83.1	27.2	**38.9**
Swin-T	192×528	1×	67.1	18.7	28.7	74.3	27.9	59.5	76.7	27.8	37.7
Swin-T	256×704	0.4×	20.4	17.6	27.6	74.2	27.9	63.4	91.0	26.2	35.5
SparseViT	256×704	1×	58.6	18.7	**30.0**	72.0	27.5	59.7	81.7	26.6	**38.3**

Table 2.1: Results of monocular 3D object detection on nuScenes.

without re-training. By making the execution of all operations window-wise, our method simplifies the pruning process.

Window Activation Pruning

Not all windows are equally important. We define the importance of each window as its L2 activation magnitude. This is much simpler than other learning-based measures since it introduces smaller computational overhead and is fairly effective in practice.

Given an activation sparsity ratio (which will be detailed in the next section), we first gather those windows with the highest importance scores, then apply `MHSA`, `FFN` and `LN` only on these selected windows, and finally scatter outputs back. Figure 2.2 shows the workflow of window activation pruning. To mitigate information loss due to coarse-grained window pruning, we simply duplicate the features of the unselected windows. This approach incurs no additional computation, yet proves highly effective in preserving information, which is critical for dense prediction tasks such as object detection and semantic segmentation.

Shared Scoring. Unlike conventional weight pruning, importance scores are input-dependent and need to be computed during inference, which can introduce significant overhead to the latency. To mitigate this, we compute the window importance score only once per stage and reuse it across all the blocks within the stage, amortizing the overhead. This also ensures that the window ordering remains consistent within a stage. We simplify the gathering operation using slicing, which does not require any feature copying.

Mixed-Sparsity Configuration Search

Using a uniform sparsity level throughout a model may not be the best strategy because different layers have varying impacts on both accuracy and efficiency. For example, early layers typically require more computation due to their larger feature map sizes, while later layers are more amenable to pruning as they are closer to the output. Thus, it is more beneficial to apply more pruning to layers with lower sensitivity and higher costs. However, manually exploring layerwise sparsity can be a time-consuming and error-prone task. To overcome this challenge, we propose a workflow that efficiently searches for the optimal mixed-sparsity pruning.

Search Space. We first design the search space for mixed-sparsity activation pruning. For each Swin block, we allow the sparsity ratio to be chosen from $\{0\%, 10\%, \ldots, 80\%\}$. Note that each Swin block contains two `MHSAs`, one with shifted window and one without. We will assign them with the same sparsity ratio. Also, we enforce the sparsity ratio to be non-descending within each stage. This ensures that a pruned window will not engage in the computation again.

Sparsity-Aware Adaptation. To identify the best mixed-sparsity configuration for a model, it is crucial to evaluate its accuracy under different sparsity settings. However, directly assessing the original model's accuracy with sparsity might produce unreliable results (see Section 2.1.5). On the other hand, retraining the model with all possible sparsity configurations before evaluating its accuracy is impractical due to the significant time and computational costs involved. We therefore propose *sparsity-aware adaptation* as a more practical solution to address this challenge. Our approach involves adapting the original model, which was trained with only dense activations, by randomly sampling layerwise activation sparsity and updating the model accordingly at each iteration. After adaptation, we can obtain a more accurate estimate of the performance of different sparsity configurations without the need for full retraining. This enables us to efficiently and effectively evaluate different mixed-sparsity configurations and identify the optimal one for the model. Notably, our approach differs from super network training (used in NAS) as we only randomly sample activations, without changing the number of parameters.

Resource-Constrained Search. With an accurate estimate of the model's performance through sparsity-aware adaptation, we can proceed to search for the best sparsity configurations within specified resource constraints. We consider two types of resource constraints: hardware-agnostic theoretical computational cost, represented by the number

Backbone	Resolution	Width	#MACs (G)	Latency (ms)	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{msk}	AP^{msk}_{50}	AP^{msk}_{75}
Swin-T	640×640	1×	161.8	46.6	42.0	63.3	45.7	38.3	60.3	40.9
Swin-T	576×576	1×	149.5	41.3	41.0	62.1	44.9	37.2	59.0	39.6
Swin-T	640×640	0.9×	122.3	41.8	40.4	61.9	43.8	37.1	58.9	39.8
SparseViT	672×672	1×	139.5	41.3	**42.4**	63.3	46.4	**38.5**	60.3	41.3
Swin-T	544×544	1×	119.8	34.8	40.5	61.2	43.8	36.8	58.2	39.1
Swin-T	640×640	0.8×	90.5	35.9	39.4	60.7	42.8	36.4	57.9	38.8
SparseViT	672×672	1×	116.5	34.1	**41.6**	62.5	45.5	**37.7**	59.4	40.2
Swin-T	512×512	1×	117.5	32.9	39.6	60.1	43.4	36.0	57.0	38.2
Swin-T	640×640	0.6×	63.4	31.7	38.7	60.2	41.6	35.7	57.0	38.0
SparseViT	672×672	1×	105.9	32.9	**41.3**	62.2	44.9	**37.4**	59.1	39.7

Table 2.2: Results of 2D instance segmentation on COCO.

of multiply-accumulate operations (#MACs), and hardware-dependent measured latency. To perform the sparsity search, we adopt the evolutionary algorithm [51]. We first initialize the population with n randomly sampled networks within the search space and using rejection sampling (*i.e.*, repeated resampling until satisfaction) to ensure every candidate meets the specified resource constraint. In each generation, we evaluate all individuals in the population and select the top k candidates with the highest accuracy. We then generate the population for the next generation through $(n/2)$ mutations and $(n/2)$ crossovers using rejection sampling to satisfy the hard resource constraints. We repeat this process to obtain the best configuration from the population in the last generation.

Finetuning with Optimal Sparsity. The resulting model from our resource-constrained search has been trained under a variety of sparsity configurations during the adaptation stage. To further optimize its performance, we finetune the model with the fixed sparsity configurations identified in the search process until convergence.

2.1.4 Experiments

In this section, we evaluate our method on three diverse tasks, including monocular 3D object detection, 2D instance segmentation, and 2D semantic segmentation.

Latency Measurement. We report the latency of the backbone in all our results as our method is only applied to the backbone. The latency measurements are obtained using a single NVIDIA RTX A6000 GPU. To ensure accurate measurements, we perform 500

inference steps as a warm-up and subsequently measure the latency for another 500 steps. To minimize the variance, we report the average latency of the middle 250 measurements out of the 500.

Monocular 3D Object Detection

Dataset and Metrics. We use nuScenes [73] as the benchmark dataset for monocular 3D object detection, which includes 1k scenes with multi-modal inputs from six surrounding cameras, one LiDAR, and five radars. We only employ camera inputs in our experiments. We report official metrics, including mean average precision (mAP), average translation error (ATE), average scale error (ASE), average orientation error (AOE), average velocity error (AVE), and average attribute error (AAE). We also report the nuScenes detection score (NDS), which is a weighted average of the six metrics.

Model and Baselines. We use BEVDet [28] as the base model for monocular 3D object detection. It adopts Swin-T [29] as the baseline and employs FPN [74] to fuse information from multi-scale features. Following BEVDet [28], we resize the input images to 256×704 and train the model for 20 epochs. We compare our SparseViT against two common model compression strategies: reducing resolution and width. For reduced resolution, we re-train the model with different resolutions. For reduced width, we uniformly scale down the model to 0.4× and 0.6×, then pre-train it on ImageNet [75] and finally finetune it on nuScenes [73].

Compared with Reduced Resolution. The accuracy of monocular 3D object detection is highly influenced by resolution scaling. With fine-grained features in higher-resolution images, our SparseViT outperforms the baseline with smaller resolutions, with comparable or faster latency. The results in Table 2.1 show that SparseViT achieves the same accuracy as Swin-T with **1.8**× lower #MACs and **1.5**× faster inference latency. Furthermore, when compared to the baseline with 192×528 resolution, SparseViT achieves 30.0 mAP and 38.3 NDS at 50% latency budget of the full Swin-T backbone, which is **1.3** mAP and 0.6 NDS better, respectively.

Comparison with Reduced Width. Reducing the model's width considerably lowers #MACs. However, this decrease in computation cost might not necessarily translate into a measured speedup due to low device utilization. SparseViT outperforms the baseline with 0.6× width by 1.3 mAP and the one with 0.4× width by 2.4 mAP at similar latency. This

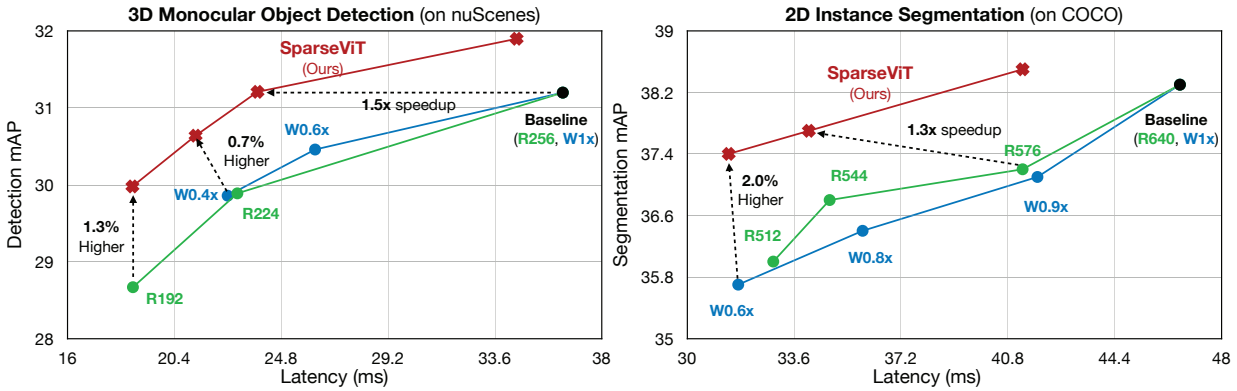


Figure 2.3: SparseViT delivers a significantly better accuracy-efficiency trade-off than the baselines with reduced resolutions and widths on monocular 3D object detection (**left**) and 2D instance segmentation (**right**).

indicates that activation pruning is more effective than model pruning in latency-oriented compression.

2D Instance Segmentation

Dataset and Metrics. We use COCO [76] as our benchmark dataset for 2D instance segmentation, which contains 118k/5k training/validation images. We report the box/mask average precision (AP) over 50% to 95% IoU thresholds.

Model and Baselines. We use Mask R-CNN [77] as the base model. The model uses Swin-T [29] as its backbone. We adopt 640×640 as the default input resolution and train the model for 36 epochs. We compare our SparseViT against baselines with reduced resolutions and widths. For reduced resolution, we train the model using random scaling augmentation [29, 77] and evaluate the model under different resolutions. For reduced width, we uniformly scale down the model to 0.6×, 0.8× and 0.9×, pre-train it on ImageNet [75] and finetune it on COCO [76].

Comparison with Reduced Resolution. As in Table 2.2, SparseViT consistently outperforms the baseline with less computation across various input resolutions from 512×512 to 640×640. Our key insight is that starting with a high resolution of 672×672 and aggressively pruning the activation is more efficient than directly scaling down the input resolution. This observation aligns with the visualizations in Figure 2.1, where fine-grained details become indistinguishable under low resolution. Despite using a higher resolution, SparseViT achieves 1.2× smaller #MACs than the baseline while delivering 0.4% higher

Backbone	**Resolution**	**Latency** (ms)	**Mean IoU**
Swin-L	1024×2048	329.5	83.3
Swin-L (R896)	896×1792	256.5	82.8
SparseViT	1024×2048	250.6	**83.2**

Table 2.3: Results of 2D semantic segmentation on Cityscapes.

AP^{box} and 0.2% higher AP^{msk}. With similar accuracy, SparseViT has **1.4**× lower #MACs, resulting in a perfect **1.4**× speedup. This is because our SparseViT performs *window-level* activation pruning, which is equivalent to reducing the batch size in `MHSA` computation and is easy to accelerate on hardware. Similarly, to match the accuracy of the baseline with 90% resolution, SparseViT is **1.3**× faster and consumes **1.4**× less computation. Remarkably, despite using 30% larger resolution (*i.e.*, 1.7× larger #MACs to begin with!), SparseViT is more efficient than the baseline at 512×512 resolution, while providing significantly better accuracy (**+1.7** AP^{box} and **+1.4** AP^{msk}).

Comparison with Reduced Width. In Table 2.2, we also compare SparseViT with the baseline with reduced channel width. Although reducing channel width leads to a significant reduction in #MACs, we do not observe a proportional increase in speed. For example, the baseline with 0.6× channel width on 640×640 inputs consumes only 63.4G MACs, yet it runs slightly slower than SparseViT on 672×672 inputs with 105.9G MACs (which is actually 1.7× heavier!). GPUs prefer wide and shallow models to fully utilize computation resources. Pruning channels will decrease device utilization and is not as effective as reducing the number of windows (which is equivalent to directly reducing the batch size in `MHSA` computation) for model acceleration.

2D Semantic Segmentation

Dataset and Metrics. Our benchmark dataset for 2D semantic segmentation is Cityscapes [78], which consists of over 5k high-quality, fully annotated images with pixel-level semantic labels for 30 object classes, including road, building, car, and pedestrian. We report mean intersection-over-union (mIoU) as the primary evaluation metric on this dataset.

Model and Baselines. We use Mask2Former [79] as our base model, which uses Swin-L [29] as its backbone. We train the model for 90k iterations with an input resolution of 1024×2048. Here, we only compare to the baseline with reduced resolution.

	#MACs (G)	Latency (ms)	mAP Drop
DynamicViT	98.6	33.3	-0.1
DynamicViT	91.7	32.6	-0.4
SparseViT	**78.4**	**23.8**	**0.0**

Table 2.4: Window pruning (SparseViT) is more efficient and effective than learnable token pruning (DynamicViT).

Results. Based on Table 2.3, SparseViT model attains comparable segmentation accuracy to the baseline while delivering a speedup of 1.3×. In contrast, reducing the resolution results in a more substantial decrease in accuracy. By utilizing spatial redundancy, SparseViT delivers competitive results while being more efficient than direct downsampling.

2.1.5 Analysis

In this section, we present analyses to validate the effectiveness of our design choices. All studies are conducted on monocular 3D object detection (on nuScenes), except the evolutionary search one in Figure 2.4(c), which is conducted on 2D instance segmentation (on COCO).

Window pruning is more effective than token pruning. Table 2.4 demonstrates that SparseViT achieves a low computational cost and latency without any loss of accuracy, whereas DynamicViT [68], a learnable token pruning method, experiences a substantial decrease in accuracy of 0.4 mAP with only a minor reduction in computational cost. These findings offer valuable insights into the comparative performance of these pruning methods. Furthermore, it is worth noting that token pruning requires more fine-grained *token-level* gathering, which has inferior memory access locality and tends to be slower on GPUs, unlike window pruning in our SparseViT that only necessitates *window-level* gathering.

Pruning from higher resolution matters. A key design insight in SparseViT is that it is more advantageous to start with a *high-resolution* input and *prune more*, rather than beginning with a *low-resolution* input and *pruning less*. While counterintuitive, starting with a high-resolution input allows us to retain fine-grained information in the image. The abundance of uninformative background windows provides us with ample room for activation pruning. Quantitatively, as in Table 2.5, starting from the highest resolution (*i.e.*, 256×704) produces the best accuracy under the same latency constraint.

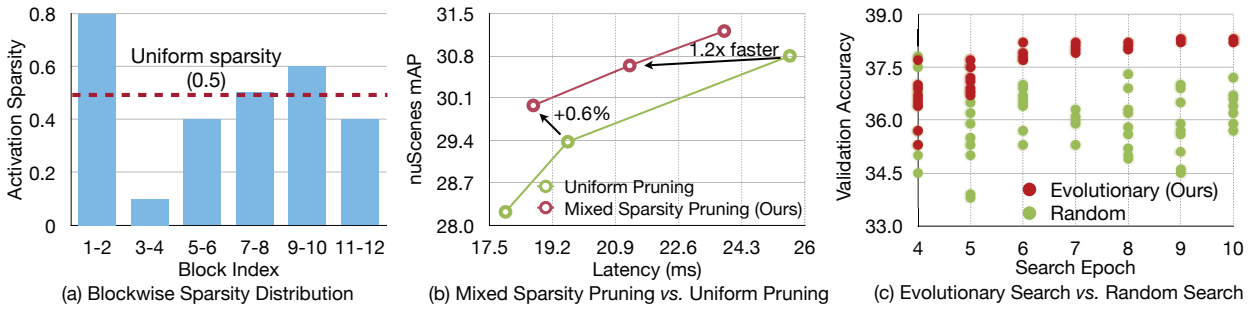


Figure 2.4: Non-uniform sparsity is better than uniform sparsity (a, b). Evolutionary search is more sample-efficient than random search (c).

Input Resolution	#MACs (G)	Latency (ms)	mAP
192×528	67.1	**18.7**	28.7
224×616	64.9	19.1	29.7
256×704	**58.6**	**18.7**	**30.0**

Table 2.5: Starting from a high-resolution input and pruning more is better than starting from a low-resolution input and pruning less.

Mixed-sparsity pruning is better than uniform pruning. In Figure 2.4(a), we show the pruning strategy used by SparseViT to achieve 50% overall sparsity. Unlike uniform sparsity ratios applied to all layers, SparseViT favors non-uniform sparsity ratios for different layers based on their proximity to the input. Specifically, the smaller window sizes in the first and second blocks allow for more aggressive pruning, while larger window sizes in later layers result in less aggressive pruning. This non-uniform sparsity selection leads to better accuracy, as in Figure 2.4(b). Compared to uniform pruning, SparseViT achieves similar accuracy but is up to 1.2× faster. Alternatively, when compared at similar speeds, SparseViT achieves 0.6% higher accuracy than uniform pruning.

Evolutionary search is better than random search. We demonstrate the efficacy of evolutionary search in selecting the sparsity ratios. Figure 2.4(c) compares the results obtained by evolutionary search and random search, by visualizing the validation mAP of the best-performing models found in the last seven epochs. The accuracy of the models discovered by evolutionary search converges to a high value of 37.5 after the sixth epoch, whereas the models found by random search still exhibit high variance until the final epoch.

Sparsity-aware adaptation offers a better accuracy proxy. Conventional pruning methods [43, 54] typically scan different sparsity ratios at each layer, zero out corresponding

Figure 2.5: SparseViT effectively prunes irrelevant background windows while retaining informative foreground windows. Each window's color corresponds to the number of layers it is executed. Brighter colors indicate that the model has executed the window in more layers.

Sparsity-Aware Adaptation	Sparsity Ratios	mAP w/o FT	mAP w/ FT
	(0.5, 0, 0, 0)	28.31	31.40
	(0, 0.5, 0, 0)	30.93	31.37
✓	(0.5, 0, 0, 0)	30.83	31.45
	(0, 0.5, 0, 0)	30.63	31.44

Table 2.6: Sparsity-aware adaptation (SAA) improves the correlation between the accuracy before and after finetuning (FT).

weights, and evaluate accuracy directly on a holdout validation set. While this approach works well for weight pruning in CNNs, it falls short when generalizing to activation pruning in ViTs. For example, as in Table 2.6, the naive accuracy sensitivity analysis approach asserts that pruning away 50% of the windows in stage 2 is much better than doing so in stage 1 (*i.e.*, 30.93 mAP *vs*. 28.31 mAP). However, if both models are finetuned, the accuracy difference is almost zero. In contrast, sparsity-aware adaptation provides a much better accuracy proxy (*i.e.*, 30.83 mAP *vs*. 30.63 mAP), which can serve as a better feedback signal for evolutionary search.

Sparsity-aware adaptation improves the accuracy. In Table 2.7, we explore the impact of sparsity-aware adaptation on model convergence during the finetuning stage. Using the same window sparsity ratios, we compare three approaches: **(a)** training from scratch, **(b)** finetuning from a pre-trained Swin-T, and **(c)** finetuning from a pre-trained Swin-T

		mAP
(a)	Training from scratch	29.67
(b)	Finetuning from a pre-trained Swin-T	30.83
(c)	Finetuning from a pre-trained Swin-T with sparsity-aware adaptation	**31.21**

Table 2.7: Sparsity-aware adaptation improves the convergence.

with sparsity-aware adaptation (SAA). Our results indicate that approach **(c)** achieves the best performance. We speculate that sparsity-aware adaptation could serve as a form of implicit distillation during training. Models with higher window sparsity can benefit from being co-trained with models with lower window sparsity (and therefore higher #MACs and higher capacity), resulting in improved overall performance.

SparseViT learns to keep important foreground windows. In Figure 2.5, we visualize the window pruning strategy discovered by SparseViT, where the color represents the number of layers each window is executed. Notably, on the first row, SparseViT automatically learns the contour of the objects, as demonstrated in the third and fourth figures, where the computer and sportsman are respectively outlined. Furthermore, on the second row, the windows corresponding to foreground objects are not pruned away. Despite being a small object, the pedestrian in the last figure is retained throughout the entire execution, illustrating the effectiveness of SparseViT.

L2 magnitude-based scoring is simple and effective. Table 2.8 demonstrates that the L2 magnitude-based scoring is simple and effective, outperforming the learnable window scoring that utilizes MLP and Gumbel-Softmax for predicting window scores. We also include the regularization loss on pruning ratios, following Rao *et al*. [68], to restrict the proportion of preserved windows to a predetermined value in the learnable window scoring baseline. However, the added complexity of the learnable scoring results in higher latency and #MACs compared to the L2 magnitude-based scoring. Achieving an optimal balance between pruning regularization loss and detection loss is not an easy task, as evidenced by a 0.4 mAP drop observed in the learnable scoring method.

Scoring	#MACs (G)	Latency (ms)	mAP Drop
Learnable	89.6	33.1	-0.6
L2 Magnitude	**78.4**	**23.8**	**0.0**

Table 2.8: L2 magnitude-based scoring is simple and effective, achieving a better accuracy-efficiency trade-off than learnable scoring.

	Sparsity	#MACs (G)	Latency (ms)	mAP
Independent scoring	50%	70.68	23.3	30.08
Shared scoring	50%	70.68	**23.0**	**30.17**

Table 2.9: Shared scoring per stage reduces the cost of score calculation, leaving room for effective computation and offering a better accuracy-latency trade-off than independent scoring per block.

Shared scoring is better than independent scoring. We compare our proposed shared scoring strategy with the independent scoring. Despite being an approximation, sharing window scores per stage, as in Table 2.9, does not negatively impact the performance. This strategy amortizes the cost of score calculation, which allows for more effective computation and results in a better accuracy-latency trade-off.

2.1.6 Discussion

Although activation pruning is a very powerful technique for preserving high-resolution information, it does not offer actual speedup for CNNs. We revisit activation sparsity for recent window-based ViTs and propose a novel approach to leverage it. We introduce sparsity-aware adaptation and employ evolutionary search to efficiently find the optimal layerwise sparsity configuration. As a result, SparseViT achieves 1.5×, 1.4×, and 1.3× measured speedups in monocular 3D object detection, 2D instance segmentation, and 2D semantic segmentation, respectively, with minimal to no loss in accuracy. We hope that our work inspires future research to explore the use of activation pruning for achieving better efficiency while retaining high-resolution information.

2.2 SparseRefine

Semantic segmentation empowers numerous real-world applications, such as autonomous driving and augmented/mixed reality. These applications often operate on high-resolution images (*e.g.*, 8 megapixels) to capture the fine details. However, this comes at the cost of considerable computational complexity, hindering the deployment in latency-sensitive scenarios. We introduce SparseRefine, a novel approach that enhances *dense low-resolution* predictions with *sparse high-resolution* refinements. Based on coarse low-resolution outputs, SparseRefine first uses an entropy selector to identify a sparse set of pixels with high entropy. It then employs a *sparse* feature extractor to efficiently generate the refinements for those pixels of interest. Finally, it leverages a gated ensembler

to apply these sparse refinements to the initial coarse predictions. SparseRefine can be seamlessly integrated into any existing semantic segmentation model, regardless of CNN- or ViT-based. SparseRefine achieves significant speedup: **1.5 to 3.9 times** when applied to HRNet-W48, SegFormer-B5, Mask2Former-T/L and SegNeXt-L on Cityscapes, with negligible to no loss of accuracy.

2.2.1 Introduction

Semantic segmentation is a fundamental computer vision task with critical applications in autonomous driving, augmented reality, and mixed reality. Deep neural networks have significantly boosted semantic segmentation performance in recent years [42, 80–84]. Yet, deploying these computationally intensive models on resource-constrained edge devices remains a challenge.

Significant efforts have been dedicated to designing compact neural networks with reduced computational complexity [44–48]. However, in dense-prediction tasks like semantic segmentation, the image resolution makes greater impact to model's inference latency than the model size. This is because real-world segmentation applications often involve megapixel high-resolution images, which surpass the typical image classification workload by 1-2 orders of magnitude.

Reducing the image resolution through downsampling can result in a noticeable increase in speed. But, this comes at the cost of accuracy degradation. Segmentation models are generally more adversely affected by reduced resolution compared to classification models as low-resolution images result in the loss of fine details, including small or distant objects. The missing information can be safety-critical (*e.g.*, for autonomous driving).

We introduce SparseRefine as a novel and complementary approach to address this problem. We find that the differences between the dense low-resolution predictions and the dense high-resolution predictions primarily emerge in a sparse set of pixels. As shown in Figure 2.6, our idea is to enhance *dense low-resolution* predictions (based on downsampled inputs) with *sparse high-resolution* refinements. SparseRefine only refines a sparse set of carefully-selected pixels, enabling it to avoid unnecessary high-resolution computations at the fine-grained pixel level. Besides, SparseRefine is compatible with both CNN- and ViT-based semantic segmentation models.

SparseRefine achieves remarkable and consistent speedup: **1.5 to 3.9 times** when applied to HRNet-W48, SegFormer-B5, Mask2Former-T/L and SegNeXt-L on Cityscapes, while maintaining accuracy. We also validate the general effectiveness of SparseRefine on many other datasets including PASCAL VOC [85], BDD100K [86], DeepGlobe [87], and

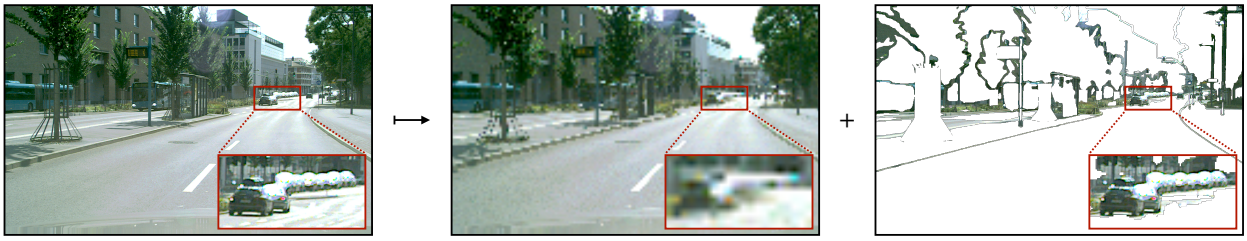


Figure 2.6: Processing *dense high-resolution* inputs is computationally expensive. We propose an alternative approach by integrating *dense low-resolution* and *sparse high-resolution* inputs, which provide complementary information about the overall scene layout and intricate object details. Leveraging the lower resolution and sparsity of these inputs allows for more efficient processing.

ISIC [88]. SparseRefine also exhibits superior performance compared with related methods including Token Pruning [21], Mask Refinement [89], and Patch Refinement [90–92].

2.2.2 Related Work

Semantic Segmentation. Semantic segmentation is a fundamental task in computer vision which assigns a class label to each pixel in an image. Following FCN [80], early deep learning models [93, 94] for semantic segmentation relied on CNN-based architectures. DeepLab and PSPNet [95] improved FCN by introducing atrous convolution [96], spatial pyramid pooling [95, 97, 98], encoder-decoder mechanism [99], depthwise convolution [100] and neural architecture search [101]. Follow-up research proposed attention mechanism [102] and object context modeling [103]. Recently, researchers also studied efficient segmentation architectures [84, 104–112]. Recent advances in vision transformers [29, 33, 35, 36, 39, 113] also inspired the design of attention-based semantic segmentation models. SegFormer [42], SETR [41], Segmenter [114], HRFormer [115], SwinUNet [116] and EfficientViT [117] designed transformer-based backbones for segmentation, while MaskFormer [79] and Mask2Former [83] modeled semantic segmentation as mask classification.

Activation Sparsity. Activation sparsity naturally exists in videos [118, 119], point clouds [60, 120] and masked images in self-supervised visual pre-training [121–124]. It can also be introduced through activation pruning [65–68, 125, 126], token merging [127, 128] or clustering [129]. These methods are specifically designed for classification or detection tasks, where there is no need to preserve information from all pixels. However, they are not suitable for semantic segmentation, which requires per-pixel predictions. An exception is SparseViT [21], which skips computation on pruned windows while retaining

their features. As such, SparseViT also works for semantic segmentation tasks. We will demonstrate that SparseRefine achieves superior efficiency compared with SparseViT in Section 2.2.4. Recently, system and architecture researchers also created high-performance GPU libraries [25, 31, 61, 130–132] and specialized hardware [69, 133–136] to exploit activation sparsity.

Mask Refinement. Mask refinement for segmentation has been studied even before the prevalence of deep learning. Traditional methods [137–140] formulated the task of semantic segmentation as graph cuts. The mask outputs were then post-processed using a conditional random field (CRF) [141–143], which aimed to minimize energy and capture local consistency in predicted labels. While CRF continues to impact the field in the deep learning era [98, 130, 144], its inefficiency eventually led to the development of PointRend [89] and RefineMask [145]. Inspired by graphics rendering, PointRend [89] first identifies uncertain pixels from deeper and lower resolution feature maps. These pixels are then refined using a PointNet [146], leveraging interpolated shallower and higher resolution features. RefineMask [145] gradually upsamples the predictions and incorporates the fine-grained features to alleviate the loss of details for high-quality instance mask prediction. Both PointRend and RefineMask upscale the *output resolution* with the help of high-resolution *features*, while SparseRefine is focused on reducing the *input resolution* and retains fine-grained details from full-scale *raw RGB pixels*. While PointRend and RefineMask prioritize improving *accuracy*, SparseRefine aims to minimize *latency*. Therefore, our method is fundamentally orthogonal to existing mask refinement strategies.

Multi-Scale Models. Multi-scale models have garnered popularity in high-resolution visual recognition tasks due to the diverse range of object sizes within an image. In early segmentation approaches, multi-resolution features were fused either using an FPN [108, 109, 147, 148] or right before the prediction head [97–99]. Subsequently, new primitives such as OctaveConv [149], HRNet [82], and DDRNet [136] were designed to more effectively leverage multi-scale features within the backbone. There have also been explorations on refining the predictions in a patch-wise manner [90–92]. Unlike SparseRefine, which enhances dense low-resolution predictions with *sparse* high-resolution details, existing methods focus on performing *dense* refinements. Also, while existing multi-scale models employ a *parallel* design for their low-resolution and high-resolution modules, SparseRefine adopts a *sequential* counterpart. This makes our method orthogonal to these designs. We will show in Section 2.2.4 that SparseRefine could bring further improvements to multi-scale models (*e.g.*, HRNet).

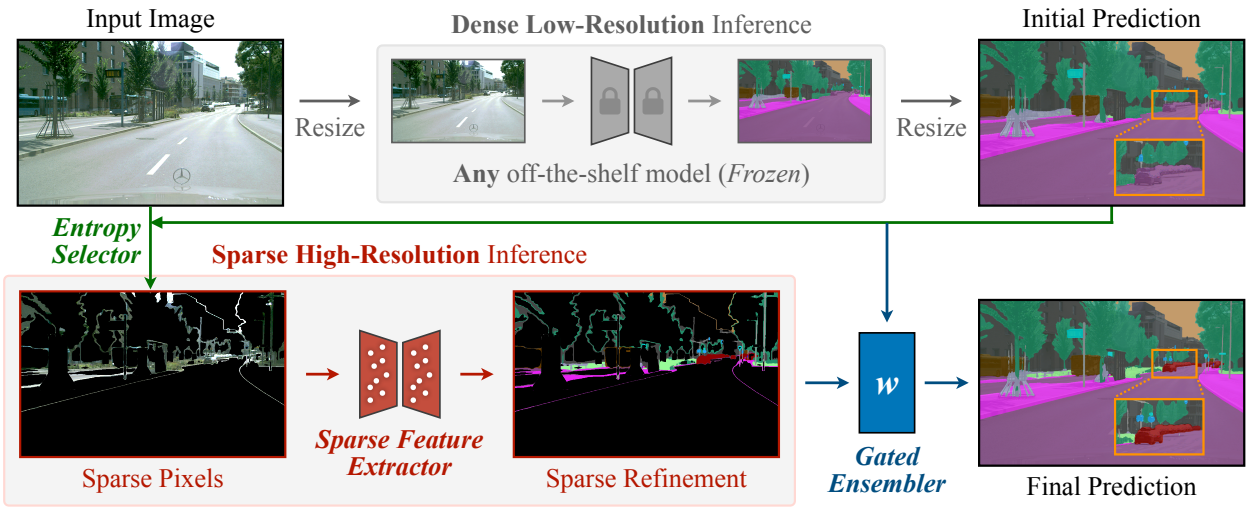


Figure 2.7: SparseRefine improves initial *dense low-resolution* predictions with *sparse high-resolution* refinements. It first performs the dense low-resolution inference on the downsampled image to obtain the initial prediction. Subsequently, it uses an entropy selector to identify a sparse set of pixels with high entropy, and then employs a *sparse* feature extractor to efficiently generate refinements for those selected pixels. Afterwards, it applies these sparse refinements to the initial predictions with a gated ensembler.

2.2.3 Method

SparseRefine improves dense low-resolution predictions with sparse high-resolution refinements. Figure 2.7 provides an overview of our pipeline. It first downsamples the input image and performs the dense low-resolution inference to obtain the initial prediction. Then, it uses an entropy selector to identify a sparse set of pixels from the input image that exhibit high entropy in the initial prediction. High-entropy pixels demonstrate high prediction uncertainty and are likely misclassified ones. Subsequently, a sparse feature extractor is employed on the selected pixels to efficiently generate refinements. The sparse feature extractor operates on high-resolution pixels, allowing it to capture fine-grained details that may be overlooked in the dense low-resolution inference. Finally, these sparse refinements are applied to the initial predictions using a gated ensembler to obtain the final prediction. Training and inference share the same pipeline, where the model used in the dense low-resolution inference is trained in advance and keeps frozen during the SparseRefine training process.

Dense Low-Resolution Prediction

One of the most straightforward ways for accelerating inference is downsampling the input image. For instance, halving the resolution of HRNet-W48 [82] will lead to a 3.7×

speedup, close to the theoretical computation reduction of $4\times$. To enable inference on downsampled images, we employ an off-the-shelf segmentation model and train it on downsampled images. Since semantic segmentation models inherently support varying image resolutions, no modifications to the model architecture are necessary. For clarity, we will refer to the model trained here as the dense baseline model.

We first obtain coarse predictions from the downsampled images. Our refinement process then proceeds independently of the original dense segmentation model. This makes our refinement module an *add-on* that can seamlessly enhance any off-the-shelf model. Next, we upsample the coarse predictions (using nearest neighbor interpolation) to match the original input resolution. All subsequent refinements will be built upon this.

The downsampling process inevitably leads to information loss, causing a decline in accuracy. In the following section, we will demonstrate how our method effectively addresses this accuracy gap through efficient sparse refinement on selected high-resolution pixels.

Sparse High-Resolution Refinement

Low-resolution predictions are fast but not as accurate as high-resolution predictions. Fortunately, *the differences in their predictions primarily emerge in a sparse set of pixels*, often associated with small or distant objects and object boundaries. Building upon this observation, our objective is to *sparsely* refine the less accurate predictions so that we could bridge the accuracy gap efficiently.

Entropy Selector. The selection of sparse pixels plays a critical role in our entire pipeline as it directly determines the number and specific pixels on which we apply the refinement process. Ideally, we would want to choose those pixels that have been misclassified in the initial dense low-resolution prediction, but this is not feasible in practice. Inspired by recent works [90, 150] that utilize entropy maps to identify uncertain pixels, we adopt a similar thought and employ entropy as the criterion for selecting the pixels we need. Our intuition is that "*less confident predictions are more likely to be wrong*".

The entropy selector uses the model's logits as input. Logits are the outputs of the segmentation model's final layer, produced just before applying the softmax operation. The size of the logits is $H \times W \times C$ where C is the number of classes. The selector calculates the entropy of each pixel using $e = -\sum_c p_c \log p_c$ (where $p \in R^C$). Pixels with high entropy (*i.e.* exceeding a threshold α) are selected. These selected pixels can then be extracted from the input image for further refinement.

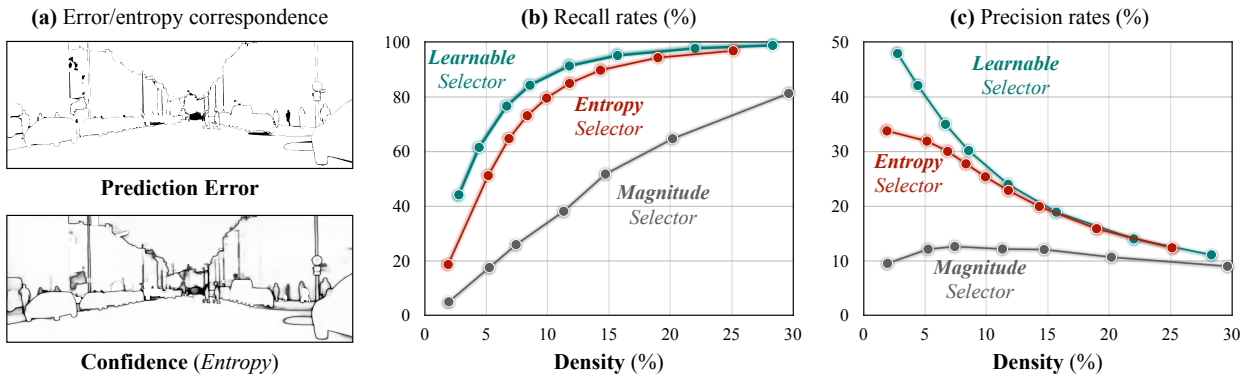


Figure 2.8: The entropy map exhibits a strong correlation with the error map **(a)**. Recall rates **(b)** and precision rates **(c)** for the entropy selector, magnitude selector, and learnable selector.

Visual verification from Figure 2.8a confirms a strong correspondence between the entropy map and the error map. Quantitatively, our entropy selector is able to identify around 80% of the misclassified pixels while selecting only 10% of the total pixels (Figure 2.8b). In contrast, the magnitude selector can only recover 40% of them with a similar density. Although the learnable selector can achieve slightly higher results than the entropy selector, it introduces significantly higher latency, requiring 17.2ms on an NVIDIA RTX 3090 GPU. In contrast, our entropy selector is remarkably efficient, requiring only 2ms. Detailed comparison among the entropy selector, magnitude selector and the learnable selector is in Section 2.2.5. The precision (Figure 2.8c) is less relevant in our case as the recall sets the accuracy upper bound for our refinement process.

Sparse Feature Extractor. Having obtained a set of sparse pixels from the entropy selector, we then generate refinements for each. Processing sparse pixels presents unique challenges due to their irregular patterns compared to dense images. Interestingly, they share similarities with 3D point clouds, where occlusion is common and only geometric outlines are evident. Although sparse, the pixels retain a well-defined shape. The successful exploration of point cloud segmentation in previous works [53, 130, 144] provides valuable insight indicating that sparse pixels should also contain contextual information that can support our sparse refinement approach.

We utilize a modified version of MinkowskiUNet [130] as our *sparse feature extractor*. It follows the standard ResNet [2] basic block design with sparse convolutions and deconvolutions. Sparse convolution [120] is the sparse equivalent of conventional dense convolution with two main distinctions: firstly, sparse convolution avoids unnecessary computations for zero activations, and secondly, it preserves the same activation sparsity

pattern throughout the model. These two properties make it much more efficient in processing our sparse pixel set. Furthermore, recent advances in system support for sparse convolution [25, 61, 130–132] enable us to translate the theoretical computational reduction, resulting from sparsity, into actual measured speedup. Please note that though we have chosen sparse convolution, alternative designs are also feasible, such as point-based convolutions [151–153] and more recent point cloud transformers [24, 154–156].

The input to our sparse feature extractor is simply the raw RGB values of the selected pixels. We have explored adding more information from the low-resolution inference as input, such as final prediction or intermediate feature. While these additional features do contribute to faster convergence, they do not yield any improved performance. The output of our sparse feature extractor comprises multi-channel features for all selected pixels. We attach a simple linear classification head to generate the refinements for these pixels of interest.

Gated Ensembler. After obtaining the refinement predictions, the straightforward approach is to directly substitute the initial predictions at the corresponding pixels. However, this approach is not always optimal. This is because, compared to dense pixels, the context information available for sparse pixels in high-resolution is relatively limited. Incorporating initial predictions from dense low-resolution images, which provide more comprehensive context information, can be beneficial.

We introduce the *gated ensembler* to intelligently combine the initial predictions (y_1) and the refined predictions (y_2). The key idea is to generate a weighting factor $w \in [0, 1]$ for each pixel of interest and utilize it to fuse the two predictions. Concretely, the final predictions are generated by

$$y = f(w \cdot y_1 + (1 - w) \cdot y_2), \quad \text{where } w = \text{sigmoid}(g([y_1; y_2; e_1; e_2])). \tag{2.1}$$

Here, $f(\cdot)$ and $g(\cdot)$ are two-layer multi-layer perceptrons (MLPs). To generate the weighting factor, we provide both the raw predictions ($y_{1,2}$) and their corresponding entropies ($e_{1,2}$) as inputs to g.

2.2.4 Experiments

Setup

Dataset. We evaluate SparseRefine primarily on Cityscapes [78], a dataset of 5,000 high-resolution (1024×2048) urban scene images with pixel-level annotations for 19 semantic

	Resolution	#Params (M)	#MACs (T)	Latency (ms)	Mean IoU
HRNet-W48	1024×2048 (D)	65.9	0.75	53.4	80.7
HRNet-W48	512×1024 (D)	65.9	0.19 (2.3×)	14.5 (1.8×)	79.2 (+0.2)
+ **SparseRefine**	1024×2048 (S)	85.7	0.32	30.3	80.9
SegFormer-B5	1024×2048 (D)	82.0	1.16	140.6	81.1
SegFormer-B5	512×1024 (D)	82.0	0.17 (3.6×)	18.5 (3.9×)	78.7 (+0.1)
+ **SparseRefine**	1024×2048 (S)	101.8	0.32	36.2	81.2
Mask2Former-T	1024×2048 (D)	36.7	0.62	66.8	81.1
Mask2Former-T	512×1024 (D)	36.7	0.16 (1.6×)	19.1 (1.5×)	78.6 (+0.2)
+ **SparseRefine**	1024×2048 (S)	56.5	0.39	44.8	81.3
Mask2Former-L	1024×2048 (D)	207.0	1.99	150.8	83.0
Mask2Former-L	512×1024 (D)	207.0	0.51 (2.4×)	45.4 (1.8×)	80.9 (+0.0)
+ **SparseRefine**	1024×2048 (S)	226.8	0.84	84.4	83.0
SegNeXt-L	1024×2048 (D)	48.8	0.53	86.3	83.0
SegNeXt-L	640×1280 (D)	48.8	0.21 (1.7×)	33.6 (1.8×)	80.8 (−0.2)
+ **SparseRefine**	1024×2048 (S)	68.6	0.32	49.1	82.8

Table 2.10: SparseRefine effectively closes the accuracy gap between low-resolution and high-resolution predictions, achieving a remarkable reduction in computational cost by **1.6 to 3.6 times** and inference latency by **1.5 to 3.9 times**. In this table, (D) and (S) denote dense and sparse inputs, respectively.

categories. To demonstrate generalizability, we validate our approach on four additional datasets: PASCAL VOC [85] for common objects, BDD100K [86] for autonomous driving, DeepGlobe [87] for aerial images, and ISIC [88] for medical images. We use mean Intersection-over-Union (mIoU) as the primary metric.

Baselines. To showcase the generalizability of our method across diverse architectures, we employ five models spanning both convolutional and transformer-based approaches. We choose HRNet-W48 [82] as the convolution-based baseline, and SegFormer-B5 [42], Mask2Former-T [83], Mask2Former-L [83], and SegNeXt-L [84] as our transformer-based baselines. We reproduce the results of all high-resolution and low-resolution baselines using MMSegmentation v1.0.0 [157]. We adhere to the default training settings, only making minimal adjustments to data augmentation parameters for lower resolutions. Due to observed instability in Mask2Former's results, we report the mean of three runs for a more robust evaluation.

		PASCAL VOC		BDD100K		DeepGlobe		ISIC	
	Resolution	Lat.	mIoU	Lat.	mIoU	Lat.	mIoU	Lat.	mIoU
HRNet-W48	Full (D)	14.7	77.8	23.5	63.6	146.4	73.4	157.7	82.3
HRNet-W48	Half (D)	5.0	77.2	6.1	60.7	38.7	72.9	40.6	80.8
+ **SparseRefine**	Full (S)	8.1	78.2	15.6	63.5	92.9	73.4	79.4	82.5

Table 2.11: SparseRefine generalizes across common object, autonomous driving, aerial and medical datasets, achieving a **1.5-2.0**× measured speedup with no loss of accuracy. The unit of latency is milliseconds.

Model Details. As the prediction logits vary across different baseline model architectures, we trained separate sparse refinement module based on their respective low-resolution output logits to achieve the optimal results. We set a different entropy threshold for each baseline model in our entropy selector. Our sparse feature extractor is a modified MinkowskiUNet that has five stages with channel dimensions of 32, 64, 128, 256, and 512 for each stage. At each stage, there are two ResNet basic blocks before downsampling and another two after upsampling. Our gated ensembler employs two linear layers to produce the weighting factor and an additional two layers to combine the predictions, both with a hidden dimension of 64.

Training Details. SparseRefine is trained independently from the dense baselines. We use the same data augmentation and training strategy employed by the dense baseline to ensure that performance improvements stem solely from our method. For data augmentation, we apply standard techniques such as random scaling (between 0.5 and 2.0), horizontal flipping, cropping (with a size of 512×1024), and photometric distortion. We apply the standard cross entropy loss to supervise the model. We adopt AdamW [158] as our optimizer, with an initial learning rate of 0.0003 and a weight decay of 0.05. We gradually decay the learning rate following the cosine-annealing schedule [159]. We train the model for 500 epochs with a batch size of 32. The training takes around 12 hours on 8 NVIDIA RTX A6000 GPUs.

Latency Details. We use cuBLAS [160] for all dense operations and utilize TorchSparse++ [25, 131] for all sparse operations. We measure the inference latency of all methods using a single NVIDIA RTX 3090 GPU with FP16 precision and a batch size of 4. We omit batch normalization layers in latency measurement as they can be folded into preceding convolution layers. We report the average latency over 500 inference steps, with a 100-step warm-up period.

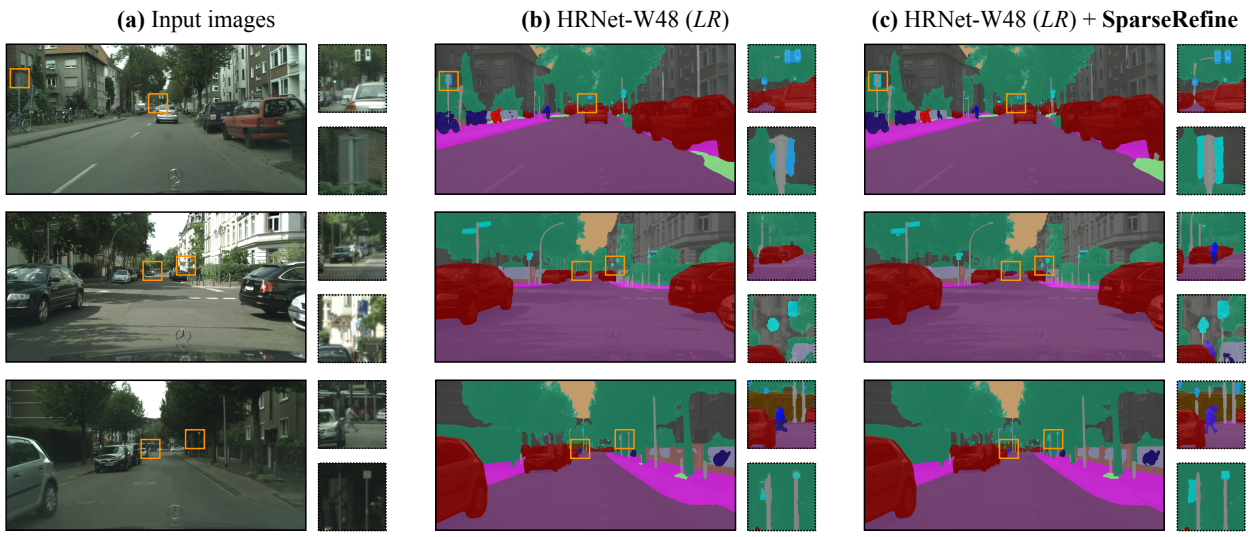


Figure 2.9: SparseRefine improves the low-resolution (*LR*) baseline with substantially better recognition of small, distant objects and finer detail around object boundaries.

	Latency (ms)	mIoU
Mask2Former-L	150.8	83.0
+ SparseViT	132.3	83.2
+ **SparseRefine**	84.4 (1.6×)	83.0

(a) Comparison to token pruning.

	Latency (ms)	mIoU
HRNet-W48	53.4	80.7
+ PointRend	30.5	79.8
+ **SparseRefine**	30.3	80.9 (+1.1)

(b) Comparison to mask refinement.

Table 2.12: SparseRefine is more efficient/effective than token pruning and mask refinement approaches.

Results

We present our key experimental results in Table 2.10. SparseRefine achieves substantial improvements in #MACs and latency, while maintaining competitive or even better accuracy compared to the baselines. Specifically, SparseRefine accelerates the baselines by at least **1.5 times** and reduces the MACs by at least **1.6 times**. Notably, SparseRefine achieves a significant speedup of **3.9 times** for SegFormer-B5. This huge improvement can be attributed to the fact that SegFormer incorporates a vanilla self-attention module with high computational complexity ($O(H^2W^2)$), and our "*downsample then sparsely refine*" strategy in SparseRefine can drastically reduce the computational cost via downsampling while recovering the accuracy through refining. From Table 2.11, SparseRefine demonstrates effective generalization across common object (PASCAL VOC), driving (BDD100K), aerial (DeepGlobe), and medical (ISIC) datasets. It delivers a consistent speedup from **1.5** to **2.0** times without compromising accuracy.

	Latency (ms)	Mean IoU
Baseline	53.4	80.7
PatchRefine (512×512)	44.1	80.8
PatchRefine (256×256)	55.4	80.8
SparseRefine	**30.3**	**80.9**

Table 2.13: Sparse refinement is faster and more accurate than patch refinement.

In addition to the quantitative results, we also present qualitative results in Figure 2.9. In the middle column, it can be observed that the low-resolution baseline struggles to accurately classify pixels in distant areas and often misclassifies details near the edges. Our SparseRefine significantly improves the ambiguous predictions, as shown in the third column. The second row is a notable example, where the segmentation on low-resolution images fails to detect a person in the far distance, while SparseRefine accurately predicts their presence. Furthermore, SparseRefine even achieves accurate predictions in challenging cases, such as the thin rod of traffic lights in the first row. These results further demonstrate the effectiveness of our method.

Comparison to Token Pruning. SparseViT [21] is a pioneering work that demonstrates the viability of token pruning for dense prediction tasks like semantic segmentation. As in Table 2.12a, SparseRefine exhibits a notable advantage over SparseViT in terms of latency reduction, achieving a 1.6× speedup while maintaining comparable accuracy. SparseRefine operates independently from token pruning methods, making it potentially compatible for use alongside them.

Comparison to Mask Refinement. We compare SparseRefine with PointRend [89], a mask refinement method. To ensure fair comparisons, we adjust the images to a resolution of 672×1344 for PointRend, aligning its latency with our SparseRefine. As in Table 2.12b, PointRend suffers a performance decline, while our method demonstrates an improvement over the baseline. PointRend relies on an MLP-based mask refinement approach using hidden features. However, when low-resolution images are used as input, the refinement process struggles to effectively compensate for information loss caused by downsampling. SparseRefine, by working directly on the high-resolution image, minimizes information loss and consequently boosts performance.

Criteria	Density	Recall	Latency	mIoU
Random	10.0%	10.0%	-	79.2
Magnitude	11.3%	38.1%	0.5ms	80.2
Learnable	11.8%	91.2%	17.2ms	81.1
Entropy	11.8%	84.9%	2.0ms	80.9
Oracle	3.3%	100%	-	92.8

(a) **Pixel Selector.** Entropy is an effective and efficient indicator for identifying misclassified pixels.

α	Density	Recall	Latency	mIoU
0.8	3.4%	32.2%	23.1 ms	79.9
0.6	6.9%	64.6%	27.0 ms	80.4
0.3	11.8%	84.9%	30.3 ms	80.9
0.1	19.0%	94.3%	37.3 ms	81.1
–	100.0%	100.0%	–	80.5

(b) **Entropy Threshold.** Performance improves with more pixels kept, but latency also increases.

RGB	Logits	Features	mIoU
✓	✗	✗	80.9
✓	✓	✗	80.2
✓	✗	✓	80.4
✓	✓	✓	80.4

(c) **Input of sparse feature extractor**. Raw RGB provides enough information.

	# of Channels (32×)	mIoU
MinkUNet	{1,2,4,8}	80.5
MinkUNet	{1,2,4,8,16}	80.9
MinkUNet	{1,2,4,8,12,16,24,32}	80.9
PointNet	–	79.3

(d) **Architecture of sparse feature extractor**. MinkUNet is much better than PointNet.

Strategy	mIoU
Direct	77.7
Entropy	80.3
Gated	80.9
Oracle	85.3

(e) **Ensembler.** Gated ensembler is the best.

Table 2.14: Ablation experiments to validate our design choices. Default settings are marked in blue.

Comparison to Patch Refinement. Some existing methods [90–92] refine the prediction in a *coarse-grained patch* level, while SparseRefine refines the prediction in a *fine-grained pixel* level. As depicted in Figure 2.8a, errors tend to be *scattered sparsely* across the entire image, making fine-grained sparsity a more suitable solution. Patch-based refinement can often lead to substantial redundant computation, as not every pixel within a patch may need refinement. This inefficiency renders the patch-based methods less effective. From Table 2.13, SparseRefine outperforms patch refinement baselines (with a patch size of 256 or 512), achieving a speedup of **1.5** to **1.8** times while also delivering higher accuracy.

2.2.5 Analysis

In this section, we analyze various alternative designs for the components of our method. Additionally, we provide detailed breakdowns of the improvements in both accuracy and efficiency. We use HRNet-W48 as the baseline model for all analyses in this section.

Pixel Selector. We compare our proposed entropy-based pixel selector to other alternatives, including random selector, magnitude selector, and learnable selector as shown in

	#MACs	Latency
Entropy Selector	0	2.3 ms
Sparse Feature Extractor	0.129T	10.9 ms
Gated Ensembler	0.001T	2.2 ms

Table 2.15: **Breakdown of #MACs and latency**. Entropy selector and gated ensembler are lightweight.

Backend	Activation	Latency
cuBLAS	Dense	52.0 ms
SpConv v2.3.5	Sparse	14.2 ms
TorchSparse v2.1.0	Sparse	**10.9 ms**

Table 2.16: **Sparse inference backend**. Sparse inference is more efficient than dense inference.

Table 2.14a. The random selector randomly selects pixels based on a density hyperparameter that we set. Compared to the entropy-based selector, the random selector shows a substantial performance drop of 1.7 mIoU, failing to achieve any improvement over the low-resolution baseline. This decline can be attributed to the notably low recall rate (10%) of the random selection approach and its lack of principled criteria to ensure the selection of misclassified pixels.

Another alternative is the magnitude-based selector. It calculates the L2 magnitude on the output of the last layer, just before the segmentation head, in order to obtain an importance score for each pixel. This approach is commonly employed in token pruning works to identify and remove unimportant areas. As depicted in Table 2.14a, it is evident that the magnitude-based selector still exhibits significantly lower recall and precision compared to the entropy-based selector. Consequently, the magnitude-based selector performs worse than our entropy selector by 0.7 mIoU.

The other alternative is the learnable selector. We first apply two dense convolutions on the RGB image to obtain a feature representation. Next, we combine the features with the dense baseline logits and entropy, sending the concatenated input through a single hidden layer perceptron. We supervise the training of this selector independent of the SparseRefine pipeline. As shown in Table 2.14a, the learnable selector achieves a better recall when compared to our entropy selector, however the incurred latency cost is extremely high. Compared with the learnable selector, the entropy selector is much more efficient.

Furthermore, we also showcase the performance of the oracle setting, wherein we select incorrect predictions solely based on the ground truth. This highlights the considerable room for improvement and underscores the immense potential of our proposed paradigm.

Entropy Threshold. We present an analysis of the impact of different entropy thresholds on latency and accuracy, as outlined in Table 2.14b. In essence, the entropy threshold involves a trade-off between latency and accuracy: a lower entropy threshold leads to the

selection and refinement of more pixels, resulting in improved performance but increased latency. We select a moderate setting with $\alpha = 0.3$ for HRNet-W48, which matches the accuracy of the high-resolution baseline with the largest speedup. The optimal α for different models could be different. We also investigate incorporating all the pixels to conduct "dense" refinement and obtain a result of 80.5, lower than 80.9 we got with 11.8% density. This is reasonable because our goal is to refine the pixels that are "difficult to learn in the low-resolutio" ones. However, incorporating all the pixels would also involve including many pixels that are easy to learn. This can serve as a shortcut for the model to easily achieve low loss with weaker capability.

Input of Sparse Feature Extractor. We examine the use of various inputs for our sparse feature extractor, as illustrated in Table 2.14c. Our default setting is to directly feed RGB pixels into the sparse feature extractor. In comparison to other settings that introduce additional logits and features, using purely RGB as input offers greater flexibility and underscores the advantage of SparseRefine as a plug-and-play module. Furthermore, we observe that incorporating more features in the inputs does not enhance performance. This suggests that the presence of ambiguous or even erroneous logits and features corresponding to misclassified pixels may mislead the sparse feature extractor, ultimately hindering performance improvement.

Architecture of Sparse Feature Extractor. We analyze how different model capacities impact performance, as demonstrated in 2.14d. Specifically, we incrementally increase the capacity of MinkUNet by expanding channels and adding more stages. The results indicate that performance improves as the model becomes larger, but eventually reaches saturation at 80.9 mIoU. We hypothesize that this occurs because we do not select all low-confidence pixels, which hinders further improvement in larger models. Additionally, we explore using PointNet as the sparse feature extractor. Similar to PointRend, its performance is also limited. The subpar performance may be attributed to the challenges associated with handling per-point RGB values without considering the contextual information.

Ensembler. We investigate different ensemble strategies in Table 2.14e. The simplest approach is to directly replace the initial predictions with the refined predictions. However, this is suboptimal due to SparseRefine's limited context (discussed in Section 2.2.3). Another alternative is the entropy-based ensembler that compares the entropy before and after refinement to determine which predictions to choose. In comparison, our gated ensembler offers a softer and more compact way to incorporate refinement into the predic-

	Road	Sidewalk	Building	Wall	Fence	Pole	Traffic Light	Traffic Sign	Vegetation	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorcycle	Bicycle	mIoU
HRNet-W48 (512×1024)	98.3	86.1	92.8	**57.8**	**66.8**	65.7	70.0	78.9	92.4	**64.7**	94.8	81.1	62.3	95.0	84.3	88.9	82.6	64.8	77.4	79.2
+ **SparseRefine**	**98.4**	**86.7**	**93.4**	57.7	66.7	**70.5**	**75.0**	**82.0**	**93.0**	64.4	**95.4**	**84.3**	**67.4**	**95.8**	**85.6**	**89.6**	**83.3**	**67.9**	**80.1**	**80.9**
HRNet-W48 (1024×2048)	98.4	86.6	93.2	55.7	64.9	71.5	75.8	82.9	92.8	65.4	95.4	84.6	65.8	95.7	80.4	91.5	83.2	70.1	80.1	80.7

Table 2.17: SparseRefine consistently improves the performance of the low-resolution baseline across different categories, particularly for small objects.

tion. It is noteworthy that the gated ensembler outperforms the entropy-based ensembler by 0.6 mIoU. Additionally, we analyze the performance of the oracle setting, where we choose the better of the predictions before and after refinement. This analysis reveals a substantial room for improvement of 4.4 mIoU, further emphasizing the potential for future enhancements.

Breakdowns. Our per-class performance in Table 2.17 reveals that SparseRefine consistently improves the performance of the low-resolution baseline in almost every category, particularly for small instances such as person, rider, pole. These categories also exhibit the most significant degradation in the low-resolution baseline when compared to the high-resolution baseline. This observation highlights the effectiveness of SparseRefine in capturing fine-grained details, thanks to its ability to utilize sparse high-resolution information.

#MACs and latency breakdown for each component is presented in Table 2.15. The entropy selector and gated ensembler introduce minimal computational overhead, with the feature extractor remaining the primary computational component. We have implemented the sparse feature extractor using different inference backends. As shown in Table 2.16, our input activation has high (approximately 90%) sparsity. Therefore, sparse inference backends like SpConv and TorchSparse are more suitable than dense inference backends such as cuBLAS.

2.2.6 Discussion

We present SparseRefine that enhances low-resolution dense predictions with high-resolution sparse refinements. It first incorporates an entropy selector to identify a sparse set of pixels with the lowest confidence, followed by a sparse feature extractor that efficiently generates refinements for those selected pixels. Finally, a gated ensembler is utilized to integrate these sparse refinements with the initial coarse predictions. Notably,

SparseRefine can be seamlessly integrated into various existing semantic segmentation models, irrespective of their model architectures. Empirical evaluation on the five dataset demonstrated remarkable speed improvements, with negligible to no loss of accuracy. We believe that the speedups that accrue from our approach of combining *low-resolution* prediction followed by *sparse high-resolution* refinement, will further enable the deployment of high-resolution semantic segmentation in latency-sensitive applications.

Chapter 3

Designing Efficient Sparse Primitives

3.1 (Sparse) PVCNN

3D neural networks are widely used in real-world applications (*e.g.*, AR/VR headsets, self-driving cars). They are required to be fast and accurate; however, limited hardware resources on edge devices make these requirements rather challenging. Previous work processes 3D data using either voxel-based or point-based neural networks, but both types of 3D models are not hardware-efficient due to the large memory footprint and random memory access. We study 3D deep learning from the efficiency perspective. We first systematically analyze the bottlenecks of previous 3D methods. We then combine the best from point-based and voxel-based models together and propose a novel hardware-efficient 3D primitive, *Point-Voxel Convolution (PVConv)*. We further enhance this primitive with the sparse convolution to make it more effective in processing large (outdoor) scenes. Based on our designed 3D primitive, we introduce *3D Neural Architecture Search (3D-NAS)* to explore the best 3D network architecture given a resource constraint. We evaluate our proposed method on six representative benchmark datasets, achieving state-of-the-art performance with **1.8-23.7**$\times$ measured speedup.

3.1.1 Introduction

Deep learning has received increased attention thanks to its wide applications. It has been applied in AR/VR headsets to understand the layout of indoor scenes; it has also been used in the LiDAR perception that serves as the eyes of autonomous driving systems to understand the semantics of outdoor scenes to parse the drivable area (*e.g.*, roads, parking areas). These real-world applications require high accuracy and low latency at the same time: *i.e.*, AR/VR headsets aim to offer an instant and accurate response for better user

experience, and self-driving cars are expected to drive safely even at a relatively high speed. However, the computational resources on these devices are tightly constrained by the form factor (since we do not want a full backpack of hardware or a whole trunk of workstations) and heat dissipation. Thus, it is crucial to design efficient and effective 3D neural network models with limited hardware resources.

Collected by LiDAR sensors, 3D data usually comes in the format of point clouds. Conventionally, researchers rasterize the point cloud into voxel grids and process them using 3D volumetric convolutions [161]. With low resolutions, there will be information loss during voxelization: multiple points will be merged together if they lie in the same grid. Therefore, a high-resolution representation is needed in order to preserve the fine details in the input data. However, the computational cost and memory requirement both increase *cubically* with voxel resolution. Thus, it is infeasible to train a voxel-based model with high-resolution inputs: *e.g.*, 3D-UNet [162] requires more than 10 GB of GPU memory on $64 \times 64 \times 64$ inputs with batch size of 16, and the large memory footprint makes it rather difficult to scale beyond this resolution.

Recently, another stream of models attempt to directly process the 3D point clouds [146, 151, 152, 163]. These point-based models require much lower GPU memory than voxel-based models thanks to the sparse data representation. However, they neglect the fact that the *random memory access* is also very inefficient. As the input points are scattered over the entire 3D space in a very irregular manner, processing them introduces many random memory accesses. Most point-based models [152] mimic the 3D volumetric convolution: *i.e.*, they compute the feature of each point by aggregating its neighboring features. However, neighbors are not stored contiguously in the point representation; therefore, indexing them requires the costly nearest neighbor search. To trade space for time, previous methods replicate the entire point cloud for each center point in the nearest neighbor search, and the memory cost will be $\mathcal{O}(n^2)$, where n is the number of input points. Another very large overhead is introduced by the dynamic kernel computation. Since the relative positions of neighbors are not fixed, these point-based models have to generate the convolution kernels dynamically based on different offsets.

Designing efficient 3D neural networks needs to take the hardware into consideration. Compared with arithmetic operations, memory operations are much more expensive: *i.e.*, they consume two orders of magnitude *higher* energy and have two orders of magnitude *lower* bandwidth (Figure 6.1a). Another very important aspect is the memory access pattern: *i.e.*, the random access will introduce memory bank conflicts and decrease the throughput (Figure 6.1b). From the hardware perspective, conventional 3D models are inefficient due to large memory footprint and random memory access.

(a) Voxel-Based Models: Memory Grows Cubically

(b) Point-Based Models: Large Memory Overheads

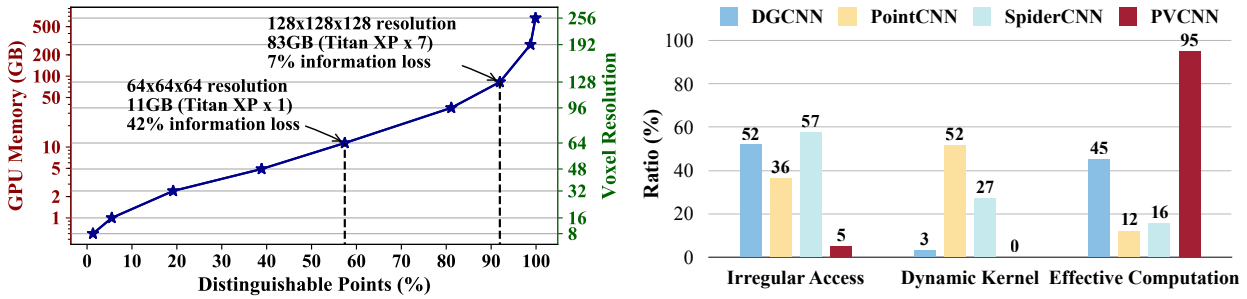


Figure 3.1: Both conventional voxel-based and point-based models are inefficient. **(a)** Voxel-based models suffer from the large information loss at acceptable GPU memory consumption. **(b)** Point-based model suffer from large irregular memory access and dynamic kernel computation overheads.

This section studies 3D deep learning from the perspective of hardware efficiency. After analyzing the bottlenecks of previous methods, we introduce a novel hardware-efficient 3D primitive, *Point-Voxel Convolution (PVConv)*, that brings the best from previous point-based and voxel-based models. It is composed of a fine-grained point-based branch that keeps the 3D data in high resolution without large memory footprint, and a coarse-grained voxel-based branch which aggregates the neighboring features without random memory accesses. However, as the resolution of its voxel-based branch is still constrained by the memory, this primitive is not effective in processing large scenes. To this end, we propose *Sparse Point-Voxel Convolution (SPVConv)* that enhances our PVConv with the sparse convolution to enable higher resolutions in the voxel-based branch. Based on these efficient 3D primitives, we then propose *3D Neural Architecture Search (3D-NAS)* to automatically explore the optimal 3D network architecture given a resource constraint.

As 3D deep learning has been used in various real-world scenarios (*e.g.*, indoor scenes for AR/VR, and outdoor scenes for autonomous driving), we demonstrate the effectiveness of our proposed method on extensive benchmarks including 3D part segmentation (for objects), 3D semantic segmentation (for indoor and outdoor scenes) as well as 3D object detection (for outdoor scenes). Across all these datasets, our method consistently achieves the state-of-the-art performance with **1.8-23.7**$\times$ measured speedup.

3.1.2 Related Work

3D Neural Networks

Conventionally, researchers relied on the volumetric representation and applied the convolution to process 3D data [164–169]. To name a few, Maturana *et al*. [166] proposed the vanilla volumetric CNN; Qi *et al*. [167] extended 2D CNNs to 3D and systematically analyzed the relationship between 3D CNNs and multi-view CNNs; Wang *et al*. [170] incorporated the octree into the volumetric CNN to reduce the memory consumption. Recent studies suggest that the volumetric representation can be used in 3D shape segmentation [168, 171, 172] and 3D object detection [169] as well. Due to the sparse nature of 3D data, the dense volumetric representation is inherently inefficient and also inevitably introduces information loss.

PointNet [146] takes advantage of the symmetric function to directly process the unordered point sets in 3D. Later research [151, 153, 163] proposed to stack PointNets hierarchically to model neighborhood information and increase model capacity. Instead of stacking PointNets as basic blocks, another type of methods [152, 173, 174] abstract away the symmetric function using dynamically generated convolution kernels or learned neighborhood permutation function. Some other research, such as SPLATNet [175] that naturally extends the 2D image SPLAT to 3D, and SONet [176] which uses the self-organization mechanism with the theoretical guarantee of invariance to point order, also show great potential in general-purpose 3D modeling with point clouds as input. Apart from these general-purpose models, there are also some attempts for specific 3D tasks. SegCloud [177], SGPN [178], SPGraph [179], ParamConv [180], SSCN [120] and RSNet [181] are specialized for 3D semantic and instance segmentation. As for 3D object detection, F-PointNet [182] is based on the RGB detector and point-based regional proposal networks; PointRCNN [183] follows the similar idea while removing the RGB detector.

Recently, researchers started to pay attention to the efficiency of 3D models. Hu *et al*. [184] proposed to aggressively downsample the point cloud to reduce the computation cost. Riegler *et al*. [161], Wang *et al*. [170, 185] and later Lei *et al*. [186] proposed to reduce the memory of volumetric representation using octrees where areas with lower density occupy fewer voxel grids. Graham *et al*. [120] and Choy *et al*. [130] proposed the sparse convolution to accelerate the vanilla volumetric convolution by keeping the activation sparse and skipping the computations in the inactive regions. However, all these methods still require considerable random memory accesses, which are very inefficient.

Hardware-Efficient Deep Learning

Extensive attention has been paid to hardware-efficient deep learning for real-world applications. For instance, researchers have proposed to reduce the memory access cost by pruning and quantizing the models [17, 18, 43, 54, 187] or directly designing the compact models [44–48, 188]. These approaches are general-purpose and suitable for any neural networks. We instead accelerate 3D neural networks based on some domain-specific properties: *e.g.*, 3D point clouds are highly sparse and spatially structured.

Neural Architecture Search

To alleviate the burden of manually designing neural networks [44–48], researchers have introduced neural architecture search (NAS) to automatically architect the neural network with high accuracy using reinforcement learning [52, 189] and evolutionary search [190]. A new wave of research started to design efficient models with neural architecture search [191–194] for edge deployment. However, conventional frameworks require high computation cost and considerable carbon footprint [195]. In order to tackle these, researchers have introduced different techniques to reduce the search cost, including differentiable architecture search [196], path-level binarization [49], single-path one-shot sampling [50, 51, 197], and weight sharing [50, 194, 198]. Furthermore, neural architecture search has also been used in compressing and accelerating neural networks, including pruning [17, 199–202] and quantization [18–20, 51]. Most of these methods are tailored for 2D visual recognition, which has many well-defined search spaces [203]. Lately, researchers have applied neural architecture search to 3D medical image segmentation [204–209] as well as 3D shape classification [210, 211]. However, they are not directly applicable to 3D scene understanding since 3D medical data are still in the similar format as 2D images (which are entirely different from 3D scenes), and 3D objects are of much smaller scales than 3D scenes (which makes them less sensitive to the resolution).

3.1.3 Analysis of Efficiency Bottlenecks

3D data usually comes in the format of point clouds:

$$\boldsymbol{x} = \{\boldsymbol{x}_k\} = \{(\boldsymbol{p}_k, \boldsymbol{f}_k)\}, \tag{3.1}$$

where $\boldsymbol{p}_k$ is the coordinate of the k^{th} point, and $\boldsymbol{f}_k$ is the feature corresponding to $\boldsymbol{p}_k$. The voxelized representation can also be unified into this formulation, where $\boldsymbol{p}_k$ stands for the coordinate of the k^{th} voxel grid. Based on this, voxel-based and point-based convolution

can be formulated as

$$\boldsymbol{y}_k = \sum_{\boldsymbol{x}_i \in \mathcal{N}(\boldsymbol{x}_k)} \mathcal{K}(\boldsymbol{x}_k, \boldsymbol{x}_i) \times \mathcal{F}(\boldsymbol{x}_i). \tag{3.2}$$

During the convolution, we iterate $\boldsymbol{x}_k$ over the entire input. For each center $\boldsymbol{x}_k$, we first index its neighbors $\boldsymbol{x}_i$ in $\mathcal{N}(\boldsymbol{x}_k)$, then convolve the neighboring features $\mathcal{F}(\boldsymbol{x}_i)$ with the kernel $\mathcal{K}(\boldsymbol{x}_k, \boldsymbol{x}_i)$, and produces the corresponding output $\boldsymbol{y}_k$.

Voxel-Based Models: Large Memory Footprint

Conventionally, researchers rasterize the point cloud into voxel grids and process them with 3D volumetric convolutions [161]. Voxel-based representation is regular and has good memory locality. However, it requires very high resolution in order not to lose much information. When the resolution is low, multiple points are bucketed into the same voxel grid, and these points will no longer be *distinguishable*. A point is kept only when it exclusively occupies one voxel grid. In Figure 3.1a, we investigate the number of distinguishable points and the memory consumption (during training with batch size of 16) with different resolutions. On a single GPU (with 12 GB of memory), the largest affordable resolution is 64, which will lead to **42%** of information loss. To keep more than 90% of the information, we then need to double the resolution to 128, consuming 7.2× GPU memory (**82.6 GB**), which is prohibitive in constrained scenarios. Although the GPU memory increases cubically with the resolution, the number of distinguishable points has a diminishing return. Therefore, the voxel-based solution is not scalable.

Point-Based Models: Sparse Data Organization

Recently, another stream of models process the point cloud directly [146, 151–153, 174]. These point-based models require much lower GPU memory than voxel-based models thanks to the sparse representation. Among them, PointNet [146] is also computation efficient, but it lacks the local context modeling capability. All the other models [151–153, 174] improve the expressiveness of PointNet by aggregating the neighborhood information in the point-based domain. However, this will lead to the irregular memory access pattern and introduce the dynamic kernel computation overhead, which becomes the efficiency bottleneck.

Irregular Memory Access. Different from the voxel-based representation, neighboring points $\boldsymbol{x}_i \in \mathcal{N}(\boldsymbol{x}_k)$ in the point-based representation will not be laid out contiguously in the memory. Besides, 3D points are scattered in $\mathbb{R}^3$; therefore, we need to explicitly

identify who are in the neighboring set $\mathcal{N}(\boldsymbol{x}_k)$, rather than by direct indexing. Point-based methods often define $\mathcal{N}(\boldsymbol{x}_k)$ as nearest neighbors in the coordinate space [152, 174] or the feature space [153]. Either requires explicit and expensive KNN computation. After KNN, gathering all neighbors $\boldsymbol{x}_i$ in $\mathcal{N}(\boldsymbol{x}_k)$ will require large amount of random memory accesses, which is not cache friendly. Combining the cost of neighbor indexing and data movement, the state-of-the-art point-based models spend **36**% [152], **52**% [153] and **57**% [174] of the total runtime on structuring the irregular data and accessing the memory randomly (Figure 3.1b).

Dynamic Kernel Computation. For 3D volumetric convolutions, the kernel value $\mathcal{K}(\boldsymbol{x}_k, \boldsymbol{x}_i)$ can be directly indexed because the relative positions of the neighbor $\boldsymbol{x}_i$ are fixed for different center $\boldsymbol{x}_k$: *e.g.*, each axis of the offset $\boldsymbol{p}_i - \boldsymbol{p}_k$ can only be $0, \pm 1$ for the convolution with size of 3. For the point-based convolution, the points are scattered over the entire 3D space irregularly; therefore, the relative positions of neighbors become unpredictable. In this case, we will have to calculate the kernel $\mathcal{K}(\boldsymbol{x}_k, \boldsymbol{x}_i)$ for each neighbor $\boldsymbol{x}_i$ *on the fly*. For instance, SpiderCNN [174] leverages the third-order Taylor expansion as a continuous approximation of the kernel $\mathcal{K}(\boldsymbol{x}_k, \boldsymbol{x}_i)$; PointCNN [152] permutes the neighboring points into a canonical order with the feature transformer $\mathcal{F}(\boldsymbol{x}_i)$. Both will introduce additional matrix multiplications. Empirically, the overhead of dynamic kernel computation for PointCNN can be more than **50**% (Figure 3.1b).

The combined overhead of irregular memory access and dynamic kernel computation ranges from **55**% (DGCNN) to **88**% (PointCNN). Thus, most computations are wasted on dealing with the irregularity of point-based representation.

3.1.4 Designing Efficient 3D Primitives

Based on our analysis of efficiency bottlenecks, we introduce a novel hardware-efficient 3D primitive for small 3D objects as well as indoor scenes: *Point-Voxel Convolution (PVConv)*. It combines the advantages of point-based methods (small memory footprint) and voxel-based methods (good data locality and regularity). For large outdoor scenes, we further propose *Sparse Point-Voxel Convolution (SPVConv)* that enhances PVConv with the sparse convolution to enable higher resolutions in the voxel-based branch.

Point-Voxel Convolution (PVConv)

Our PVConv disentangles the *fine-grained* feature transformation and the *coarse-grained* neighbor aggregation so that each branch can be implemented very efficiently and ef-

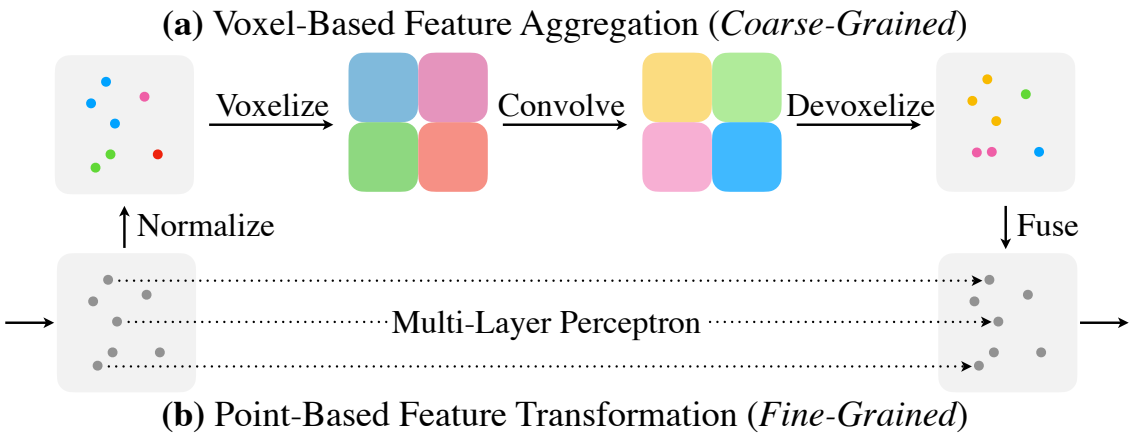


Figure 3.2: Point-Voxel Convolution (PVConv) is composed of a *low-resolution* voxel-based branch and a *high-resolution* point-based branch. The voxel-based branch extracts *coarse-grained* neighborhood information, which is supplemented by *fine-grained* individual point features extracted from the point-based branch.

fectively. As in Figure 3.2, the upper voxel-based branch first transforms the points into *low-resolution* voxel grids, then it aggregates the neighboring points with voxel-based convolutions, followed by the devoxelization to convert them back to points. Either voxelization or devoxelization requires a single scan over all points, making the memory cost low. The lower point-based branch extracts the features for each individual point. As it does not aggregate the neighbor's information, it is able to afford a very *high resolution*.

Voxel-Based Feature Aggregation

A key component of convolution is to aggregate the neighboring information in order to extract the local features. We choose to perform this feature aggregation in the volumetric domain due to its regularity.

Normalization. The scale of different point clouds might be different. Thus, we normalize the coordinates $\{\boldsymbol{p}_k\}$ before converting the point cloud into the volumetric domain. First, we translate all points into the local coordinate system with the gravity center as the origin. After that, we normalize the points into the unit sphere by dividing all coordinates by $\max\|\boldsymbol{p}_k\|_2$, and then scale and translate the points to $[0, 1]$. We denote the normalized coordinates as $\{\hat{\boldsymbol{p}}_k\}$.

Voxelization. We transform the normalized point cloud $\{(\hat{\boldsymbol{p}}_k, \boldsymbol{f}_k)\}$ into the dense voxelized representation $\{\boldsymbol{V}_{u,v,w}\}$ by averaging all of the features $\boldsymbol{f}_k$ whose coordinate $\hat{\boldsymbol{p}}_k =$

$(\hat{\boldsymbol{x}}_k, \hat{\boldsymbol{y}}_k, \hat{\boldsymbol{z}}_k)$ falls into the voxel grid (u, v, w):

$$\boldsymbol{V}_{u,v,w,c} = \sum_{k=1}^{n} \mathbb{I}[\lfloor \hat{\boldsymbol{x}}_k r \rfloor = u, \lfloor \hat{\boldsymbol{y}}_k r \rfloor = v, \lfloor \hat{\boldsymbol{z}}_k r \rfloor = w] \times \boldsymbol{f}_{k,c} / N_{u,v,w}, \tag{3.3}$$

where r denotes the voxel resolution, $\mathbb{I}[\cdot]$ is the binary indicator of whether the coordinate $\hat{\boldsymbol{p}}_k$ belongs to the voxel grid (u, v, w), $\boldsymbol{f}_{k,c}$ denotes the c^{th} channel feature corresponding to $\hat{\boldsymbol{p}}_k$, and $N_{u,v,w}$ is the normalization factor (*i.e.*, the number of points that fall in that voxel grid).

Feature Aggregation. After converting the points into the voxel grids, we apply a stack of 3D volumetric convolutions to aggregate the features. Similar to conventional 3D models, we apply batch normalization [212] and nonlinear activation function [213] after each 3D volumetric convolution.

Devoxelization. As we need to fuse the information with the point-based feature transformation branch, we transform the voxel-based features back to the domain of point cloud. A simple implementation of the voxel-to-point mapping is the nearest-neighbor interpolation (*i.e.*, assign the feature of a grid to all points in it). However, this implementation will make the points in the same voxel grid always share the same feature values. Therefore, we instead leverage the trilinear interpolation to transform the voxel grids to points to make sure that the features mapped to each point are distinct.

Point-Based Feature Transformation

The voxel-based feature aggregation branch fuses the neighborhood information in a coarse granularity. However, in order to model finer-grained individual point features, low-resolution voxel-based methods alone might not be sufficient. Hence, we directly operate on each point to extract individual point features using an MLP. Though simple, the MLP outputs distinct and discriminative features for each point. Such high-resolution individual point information is very critical to supplement the coarse-grained voxel-based information. With individual point features and aggregated neighborhood information, we can fuse two branches efficiently with an addition as they are providing complementary information.

Analysis

PVConv is much better than previous voxel-based and point-based models in terms of both efficiency and effectiveness.

Better Locality and Regularity. Our PVConv is more efficient than conventional point-based convolutions due to its better data locality and regularity. Our voxelization and devoxelization both require only $\mathcal{O}(n)$ random memory accesses, where n is the number of points, since we only need to iterate over all points once to scatter them to their corresponding grids. However, for conventional point-based methods, gathering neighbors for all points will require at least $\mathcal{O}(kn)$ random memory accesses, where k is the number of neighbors. Thus, our PVConv is $k\times$ more efficient from this viewpoint. As the typical value for k is 32/64 in PointNet++ [151] and 16 in PointCNN [152], PVConv empirically reduces the number of incontiguous memory accesses by 16-64$\times$, achieving better data locality. Besides, as our convolutions are done in the voxel domain, which is regular, our PVConv does not require KNN computation and dynamic kernel computation, which are usually quite expensive.

Higher Resolution. As our point-based feature extraction branch is implemented as MLP, a natural advantage is that we are able to maintain the same number of points throughout the whole network while still having the capability to model neighborhood information. Let us make a comparison between our PVConv and the set abstraction (SA) module in PointNet++ [151]. Suppose we have a batch of 2048 points with 64-channel features (with batch size of 16), and we then aggregate information from 125 neighbors of each point and transform the aggregated feature to output the features with the same size. In this case, the SA module requires 75.2 ms of latency and 3.6 GB of memory, while our PVConv only requires 25.7 ms of measured latency and 1.0 GB of memory. The SA module will have to downsample to 685 points (*i.e.*, around 3$\times$ downsampling) to match up with the latency of our PVConv, while the memory consumption will still be 1.5$\times$ higher. Therefore, with the same latency, our PVConv is capable of modeling the full point cloud, while the SA module has to downsample the input aggressively, which will inevitably induce information loss.

Sparse Point-Voxel Convolution (SPVConv)

PVConv is very efficient and effective especially for small 3D objects and indoor scenes as their scales are fairly small; however, it is less suitable for large outdoor scenes due to

	Input	Voxel Size (m)	Latency (ms)	Mean IoU
PVConv	Sliding Window	0.05	35640	–
	Entire Scene	0.80	146	39.0
SPVConv	Entire Scene	0.05	**85**	**58.8**

Table 3.1: Comparison between PVConv and SPVConv in large outdoor scenes. PVConv is not suitable for large scenes. If processing with sliding windows, its latency is not affordable for deployment. If taking the whole scene, its resolution is too coarse to capture useful information.

(a) Stage 1: Super Network Training

(b) Stage 2: Evolutionary Architecture Search

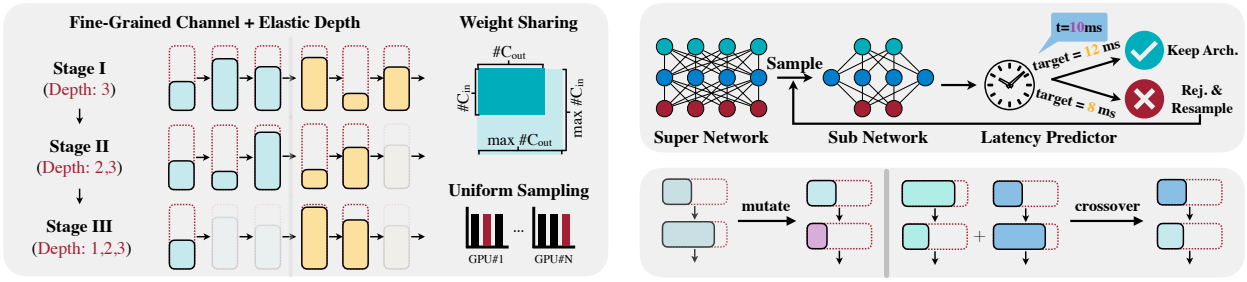


Figure 3.3: We propose a two-stage 3D Neural Architecture Search (3D-NAS) framework to automatically design efficient 3D deep learning architectures. **(a)** In the first stage, we train a *super network* that supports all candidate networks within the design space. **(b)** In the second stage, we perform *evolutionary architecture search* to find the best candidate network given a specific resource constraint.

the coarse voxelization. PVConv-based model can afford the resolution of at most 128 in its voxel-based branch on a single GPU (with 12 GB of memory). For a large outdoor scene (with size of 100m$\times$100m$\times$10m), each voxel grid will correspond to a large area (with size of 0.8m$\times$0.8m$\times$0.1m). In this case, small instances (*e.g.*, pedestrians) will only occupy very few voxel grids. From such few points, PVConv can hardly learn any useful information from the voxel-based branch, leading to a relatively low performance (see Table 3.1). Alternatively, we can process the large 3D scenes piece by piece so that each sliding window is of smaller scale. In order to preserve the fine-grained information, we found empirically that the voxel size needs to be lower than 0.05m. In this case, we have to run the model once for each of the 244 sliding windows, which will take 35 seconds to process a single scene. Such a large latency is not affordable for most real-time applications (*e.g.*, autonomous driving).

To this end, we introduce *Sparse Point-Voxel Convolution (SPVConv)* to effectively and efficiently process the large 3D scene. It follows a similar two-branch architectural design as PVConv except that the volumetric convolution in the voxel-based branch is replaced with

the sparse convolution [130]. As point clouds are intrinsically very sparse, this modification can significantly reduce the memory consumption if we use a higher resolution in the voxel-based branch. Furthermore, SPVConv can also be considered as adding a high-resolution point-based branch to the vanilla sparse convolution so that the fine details (*i.e.*, small instances) can be preserved.

Most of the operations in PVConv can be directly adapted to the sparse voxelized representation. However, the voxelization and devoxelization is not as trivial since we cannot easily index a given 3D coordinate in the sparse voxelized representation. A straightforward implementation based on the iterative coordinate comparison will require $\mathcal{O}(mn)$ of time, where m is the number of points in the point cloud representation, and n is the number of activated points in the sparse voxelized representation. As m and n are typically at the order of 10^5, this naive implementation is not practical for real-time applications. Instead, we propose to use the parallel hash tables on GPUs to accelerate the sparse voxelization and devoxelization. Note that existing implementations [120, 130] use CPU-based hash tables for kernel map construction in the sparse convolution, which is much slower. Concretely, we first construct a hash table for all activated points in the sparse voxelized representation, which can be finished in $\mathcal{O}(n)$ of time. After that, we iterate over all points, and for each point, we then query its coordinate in the hash table to obtain its corresponding index in the sparse voxelized representation. As the lookup over the hash table requires $\mathcal{O}(1)$ time in the worst case, this query step will in total take $\mathcal{O}(m)$ time. Therefore, the total time of coordinate indexing will be reduced from $\mathcal{O}(mn)$ to $\mathcal{O}(m+n)$.

3.1.5 Searching Efficient 3D Architectures

Even with the efficient 3D primitive, designing an efficient neural network model is still challenging. We need to carefully adjust the network architecture (*e.g.*, channel numbers and kernel sizes of all layers) to meet the requirements for real-world applications (*e.g.*, latency, energy, and accuracy). In this section, we propose *3D Neural Architecture Search (3D-NAS)* to automatically design efficient 3D models (Figure 3.3). We first carefully design a search space tailored for 3D and then introduce our training paradigm that supports a large number of neural networks within a single super network. Finally, we use the evolutionary search to explore the best candidate in the design space given a resource constraint.

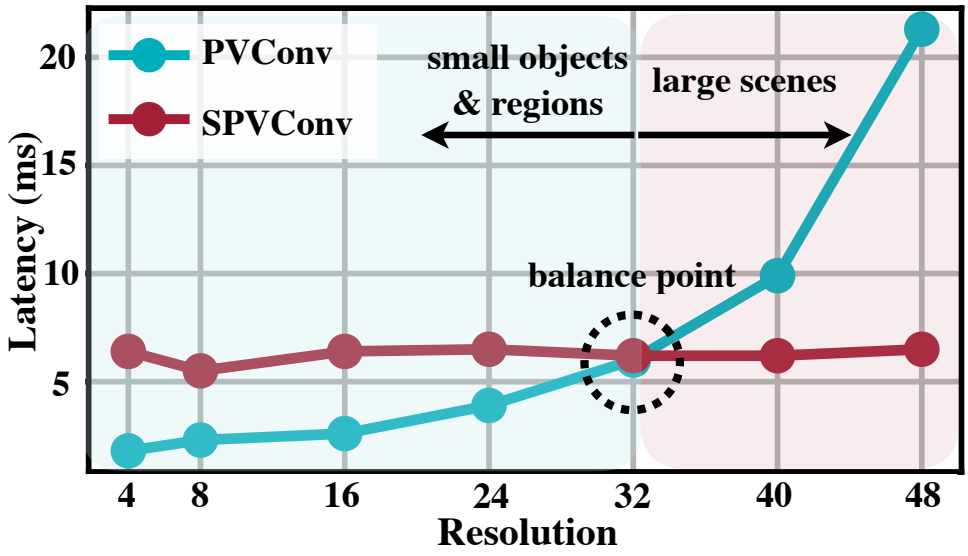


Figure 3.4: PVConv is more efficient and effective at smaller resolutions while SPVConv is more efficient at larger resolutions. Here, the GPU latency is measured on NVIDIA GTX 1080 Ti.

Design Space

The performance of neural architecture search is impacted by the design space quality. In our search space, we incorporate fine-grained channel numbers and elastic network depths; however, we do not support different kernel sizes.

Primitive Selection

PVConv accesses the memory contiguously and has fairly small memory footprint for small objects and indoor scans. However, it does not scale up very well to large-scale outdoor scenes (Table 3.1) as the resolution of its voxel-based branch is still constrained by the memory. SPVConv, on the other hand, can efficiently scale up to large scans but falls short in modeling small objects and regions due to the irregular overhead introduced by sparse operations. As in Figure 3.4, the latency of SPVConv almost stays constant when the voxel resolution increases from 4 to 48, but the latency of PVConv quickly grows when the resolution scales up. As a result, at smaller resolutions, PVConv can be up to 3.6× faster than SPVConv, while SPVConv becomes 3.3× faster than PVConv once the resolution is larger than 48. As the spatial range of single objects/indoor scans is smaller than 1.5m×1.5m×1.5m, it is sufficient to use voxel resolution smaller than 32, and therefore, PVConv is favored over SPVConv. On the other hand, SPVConv is favored when the spatial range scales far beyond 1.5m×1.5m×1.5m. We will validate this in Section 3.1.6.

Fine-Grained Channel Numbers

The computation cost increases quadratically with the number of channels; therefore, the channel number selection has a large influence on the network efficiency. Most existing neural architecture frameworks [49] only support the coarse-grained channel number selection: *e.g.*, searching the expansion ratio of the ResNet/MobileNet blocks over a few (2-3) choices. In this case, only intermediate channel numbers of the blocks can be changed; while the input and output channel numbers will still remain the same. Empirically, we observe that this limits the variety of the search space. To this end, we enlarge the search space by allowing all channel numbers to be selected from a large collection of choices (with size of $O(n)$). This fine-grained channel number selection largely increases the number of candidates for each block: *e.g.*, from 2-3 to $\mathcal{O}(n^2)$ for a block with two convolutions.

Elastic Network Depth

For 3D CNNs, reducing the channel numbers alone cannot achieve significant measured speedup, which is different from normal 2D CNNs. For example, by shrinking all channel numbers in MinkowskiNet by $4\times$ and $8\times$, the number of MACs is reduced to 7.5 G and 1.9 G, respectively. However, although the number of MACs is drastically reduced, their measured latency on the GPU is very similar: 105 ms and 96 ms (on NVIDIA GTX 1080 Ti GPU). This suggests that scaling down the number of channels alone is not able to offer us with very efficient models, even though the number of MACs is very small. This might be because 3D modules are usually more memory-bounded than 2D modules; the number of MACs decreases quadratically with channel number, while memory decreases linearly. Thus, we incorporate the elastic network depth into our design space so that layers with very small computation (and large memory cost) can be removed and merged into their neighboring layers.

Small Kernel Matters

Kernel sizes are usually included into the search space of 2D CNNs. This is because a single convolution with larger kernel size can be more efficient than multiple convolutions with smaller kernel sizes on GPUs. However, it is not the case for 3D CNNs. From the perspective of computation cost, a single 2D convolution with kernel size of 5 requires only $1.4\times$ more MACs than two 2D convolutions with kernel sizes of 3; while a single 3D convolution with kernel size of 5 requires $2.3\times$ more MACs than two 3D convolutions with kernel sizes of 3 (if applied to dense voxel grids). This larger computation cost makes it less

suitable to use large kernel sizes in 3D CNNs. Furthermore, the computation overhead of 3D modules is also related to the kernel sizes. For example, the sparse convolution requires $\mathcal{O}(k^3n)$ time to build the kernel map, where k is the kernel size and n is the number of points, which indicates that its cost grows cubically with respect to the kernel size. Based on these reasons, we decide to keep the kernel size of all convolutions to be 3 and do not allow the kernel size to change in our search space. Even with the small kernel size, we can still achieve a large receptive field by changing the network depth, which can achieve the same effect as changing the kernel size.

Training Paradigm

Searching over a fine-grained design space is very challenging since it is impossible to train every sampled candidate network randomly from scratch [191]. Motivated by Guo *et al*. [51], we incorporate all candidate networks into a single super network, and after training this super network once, each candidate network can then be extracted with inherited weights. The total training cost during the neural architecture search can then be reduced from $\mathcal{O}(n)$ to $\mathcal{O}(1)$, where n is the number of candidate networks.

Uniform Sampling

At each training iteration, we randomly sample a candidate network from the super network: *i.e*., randomly select the channel number for each layer, and then randomly select the network depth (*i.e*. the number of blocks to be used) for each stage. The total number of candidate networks to be sampled during training is very limited; thus, we choose to sample different candidate networks on different GPUs and average their gradients at each step so that more candidate networks can be sampled. For 3D, this is more critical because the 3D datasets usually contain fewer samples than the 2D datasets: *e.g*., 20K on SemanticKITTI [214] *vs*. 1M on ImageNet [75].

Weight Sharing

As the number of candidate networks is enormous, every candidate network will only be optimized for a small fraction of the total schedule. Therefore, uniform sampling alone is not enough to train all candidate networks sufficiently (*i.e*., achieving the same level of performance as being trained from scratch). To tackle this, we adopt the weight sharing technique so that every candidate network can be optimized at each iteration even if it is not sampled. Specifically, given the input channel number C_{in} and output channel number C_{out} of each convolution layer, we simply index the first C_{in} and C_{out} channels from the

weight tensor accordingly to perform the convolution [51]. For each batch normalization layer, we similarly crop the first c channels from the weight tensor based on the sampled channel number c. Finally, with the sampled depth d for each stage, we choose to keep the first d layers, instead of randomly sampling d of them. This ensures that each layer will always correspond to the same depth index within the stage.

Progressive Depth Shrinking

Suppose we have n stages, each of which has m different depth choices ranging from 1 to m. If we sample the depth d_k for each stage k randomly, the expected total depth of the network will be $\mathbb{E}[d] = \sum_{k=1}^{n} \mathbb{E}[d_k] = n(m+1)/2$, which is much smaller than the maximum depth nm. Furthermore, the probability of the largest candidate network (with the maximum depth) being sampled is extremely small: m^{-n}. Therefore, the largest candidate networks are poorly trained due to the small possibility of being sampled. To this end, we introduce progressively depth shrinking to alleviate this issue. We divide the training epochs into m segments for m different depth choices. During the k^{th} training segment, we only allow the depth of each stage to be selected from $m-k+1$ to m. This is essentially designed to enlarge the search space gradually so that these large candidate networks can be sampled more frequently.

Search Algorithm

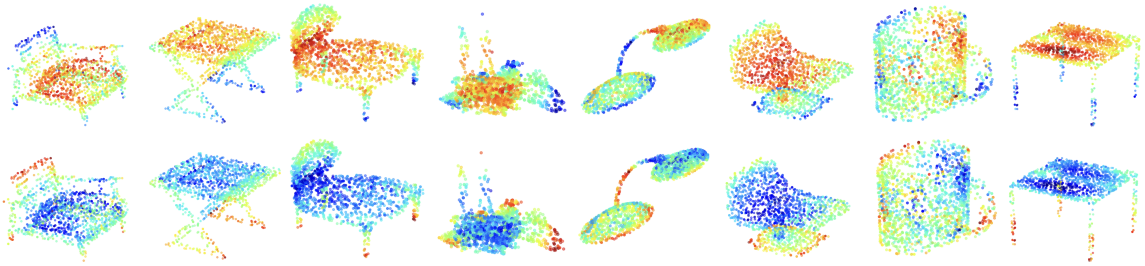
After the super network is fully trained, we use the evolutionary architecture search to find the best network architecture under a certain resource constraint.

Resource Constraint

We support #MACs and measured latency as our resource constraint for candidate networks.

#MACs Constraint. #MACs is an implementation- and hardware-agnostic efficiency metric. Unlike dense 2D CNNs, #MACs of sparse 3D CNNs cannot be simply determined by the input size and network architecture. This is because the sparse convolution only performs the computation over the active synapses; thus, its computation cost is also related to the size of kernel map, which is determined by the input sparsity pattern. To address this, we first estimate the average kernel map size over the entire dataset for each convolution layer and then sum them up to compute the average #MACs.

(a) Top Row: Features Extracted from *Coarse-Grained* Voxel-Based Branch (Large, Continuous)



(b) Bottom Row: Features Extracted from *Fine-Grained* Point-Based Branch (Isolated, Discontinuous)

Figure 3.5: Two branches are providing complementary information: the voxel-based branch focuses on the large, continuous parts, while the point-based focuses on the isolated, discontinuous parts.

Latency Constraint. We support to directly use the measured latency on the target hardware as our resource constraint as well. We first encode each candidate network into a 1D architecture vector. We then randomly sample 50,000 candidate networks from the design space and measure their latency on the target hardware. With the collected pairs of architecture vector and measured latency, we finally train an MLP regressor to estimate the latency based on the network architecture. The resulting predictor can accurately predict the latency of different candidate networks with a relative error of less than 2% on both edge and cloud GPUs.

Evolutionary Search

We automate the architecture search with the evolutionary algorithm [51]. We initialize the starting population with n randomly sampled candidate networks. At every iteration, we evaluate all candidate networks in the population and select the k models with the highest accuracy (*i.e.*, the fittest individuals). The population for the next iteration is then generated with $(n/2)$ mutations and $(n/2)$ crossovers. For each mutation, we randomly select one among the top-k candidates and modify each of its architectural parameters (*e.g.*, channel numbers, network depths) with a pre-defined probability; for each crossover, we select two from the top-k candidates and produce a new network by fusing them together randomly. Finally, the best network architecture is obtained from the population of the last iteration. During the evolutionary search, we ensure that all candidate networks in the population always meet the given resource constraint (we will resample another candidate network until the resource constraint is satisfied).

	#Params (M)	#MACs (G)	Memory (G)	Latency (ms)	mIoU
PointNet [146]	2.5	5.3	1.5	15.1	83.7
3D-UNet [162]	8.1	2996.9	8.8	682.1	84.6
RSNet [181]	6.9	1.4	0.8	74.6	84.9
PointNet++ [151]	1.8	4.9	2.0	77.9	85.1
DGCNN [153]	1.5	18.5	2.4	87.8	85.1
SPVCNN (0.25×C)	0.3	0.3	1.1	20.2	84.4
PVCNN (0.25×C)	**0.3**	1.0	**0.8**	8.3	**85.2**
SPVNAS	1.7	1.0	1.2	18.4	85.2
PVNAS-A	0.4	**0.9**	0.9	**7.0**	**85.2**
SpiderCNN [174]	2.6	10.6	6.5	170.7	85.3
SPVCNN (0.5×C)	1.1	1.3	1.4	24.7	85.1
PVCNN (0.5×C)	1.1	3.9	**1.0**	16.0	85.5
PVNAS-B	**0.6**	**2.2**	**1.0**	**11.6**	85.5
PVNAS-C	0.7	2.9	**1.0**	14.0	**85.6**
PointConv [215]	21.6	**11.6**	6.5	163.7	85.7
PointCNN [152]	8.3	26.9	2.5	135.8	86.1
SPVCNN (1×C)	4.2	5.3	2.0	38.0	85.6
PVCNN (1×C)	**4.2**	15.3	**1.6**	**41.4**	**86.2**

Table 3.2: Results of object part segmentation on ShapeNet Part. On average, PVCNN outperforms point-based models with **5.5**× speedup and **3**× memory reduction, and outperforms the voxel-based baseline with **59**× measured speedup and **11**× memory reduction.

3.1.6 Experiments

In this section, we evaluate our models on six representative 3D benchmark datasets:

- ShapeNet [165] (coarse-grained object part segmentation),
- PartNet [216] (fine-grained object part segmentation),
- S3DIS [217, 218] (indoor scene segmentation),
- SemanticKITTI [214] (outdoor scene segmentation),
- nuScenes [73] (outdoor scene segmentation),
- KITTI [219] (outdoor object detection).

On these datasets, we evaluate four variants of our method:

- **PVCNN** (manually-designed model with PVConv),

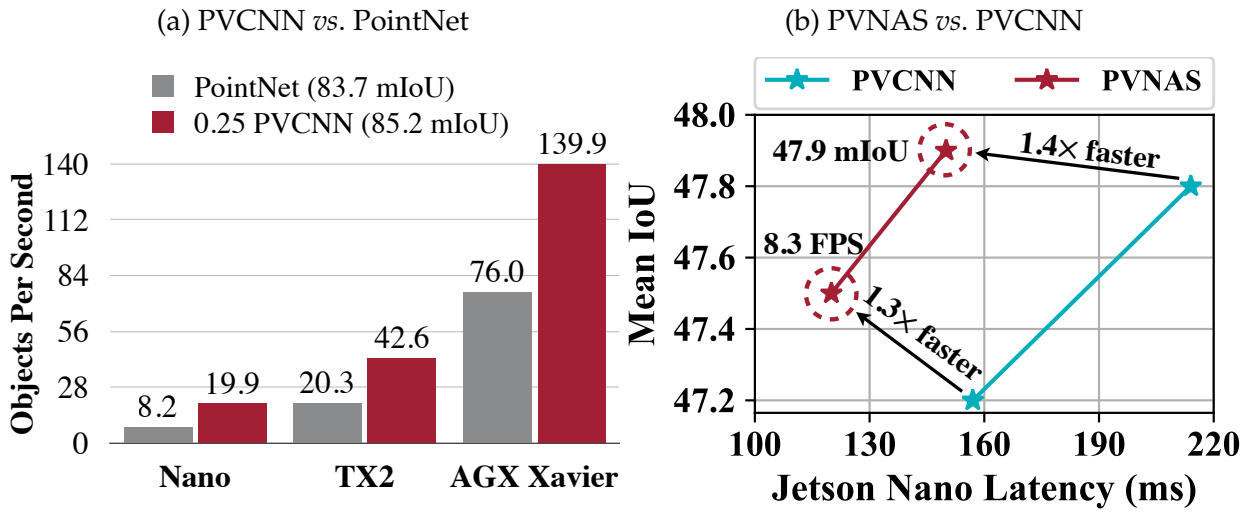


Figure 3.6: PVCNN achieves **real-time** 3D object segmentation with 2,048 input points on edge devices. With PVNAS, we further boost the efficiency on NVIDIA Jetson Nano, achieving **8.3 FPS** with **10,000** input points.

- **PVNAS** (3D-NAS applied to PVCNN),
- **SPVCNN** (manually-designed model with SPVConv),
- **SPVNAS** (3D-NAS applied to SPVCNN).

3D Object Part Segmentation

ShapeNet

We first evaluate our method on 3D part segmentation and conduct experiments on the large-scale 3D object dataset, ShapeNet [165]. As 3D objects are very small, it is suitable to process them using PVConv. Therefore, we build our PVCNN by replacing the MLP layers in PointNet [146] with PVConv's.

Baselines. We use PointNet [146], RSNet [181], PointNet++ [151] (with multi-scale grouping), DGCNN [153], SpiderCNN [174] and PointCNN [152] as point-based baselines. We reimplement 3D-UNet [162] as voxel-based baseline. For a fair comparison, we follow the same evaluation protocol as in Li *et al*. [152] and Graham *et al*. [120]. The evaluation metric is mean intersection-over-union (mIoU): we first calculate the part-averaged IoU for each of the 2,874 test models and average the values as the final metrics. Besides, we report the measured GPU latency and GPU memory consumption on a single NVIDIA GTX 1080 Ti GPU. We ensure the input data to have the same size with 2048 points and batch size of 8.

	#Params (M)	#MACs (G)	Latency (ms)	Mean IoU
PointNet [146]	2.5	25.1	9.4	35.6
PointNet++ [151]	1.8	5.4	26.0	42.5
Deep LPN [220]	2.7	3.0	206.8	38.6
SpiderCNN [174]	2.6	52.2	2170.8	37.0
PointCNN [152]	8.3	71.9	106.6	46.4
ResGCN [221]	3.8	55.9	771.3	45.1
SPVCNN (0.5×C)	0.3	1.6	13.5	43.6
PVCNN (0.5×C)	0.3	2.3	4.4	47.2
SPVCNN (0.72×C)	0.6	3.3	15.4	44.3
PVCNN (0.72×C)	0.6	4.7	5.5	47.3
SPVCNN (1×C)	1.1	6.4	16.9	45.0
PVCNN (1×C)	1.1	9.1	6.9	47.8
SGAS [211]	–	–	143*	48.3
LC-NAS-14 [222]	–	–	152*	48.6
LC-NAS-18 [222]	–	–	185*	46.6
SPVNAS	0.4	1.2	13.2	46.8
PVNAS-A	0.3	2.7	4.2	47.5
PVNAS-B	0.4	3.9	4.9	48.0

Table 3.3: Results of fine-grained object part segmentation on PartNet (*: numbers are from Li *et al*. [222], which are measured on a single NVIDIA RTX 2080 GPU).

Results. As in Table 3.2, PVCNN outperforms all previous models. It directly improves the accuracy of its backbone (PointNet) by 2.5% with even smaller overhead compared with PointNet++. Besides, we also design narrower versions of our PVCNN by reducing the number of channels to 25% (0.25×C) and 50% (0.5×C). The resulting model requires only 46.4% latency of PointNet, and it still outperforms several point-based methods with sophisticated neighborhood aggregation including RSNet, PointNet++ and DGCNN, which are almost an order of magnitude slower. PVCNN achieves a much better accuracy *vs*. latency trade-off compared with all point-based methods. With similar accuracy, PVCNN is **24**× faster than SpiderCNN and **3.3**× faster than PointCNN. Our PVCNN also achieves a significantly better accuracy *vs*. memory trade-off compared with the voxel-based baseline. With better accuracy, PVCNN can save the GPU memory consumption by **10**× compared with 3D-UNet. We further apply 3D-NAS to PVCNN under different latency constraints to obtain a family of PVNAS models. With the same accuracy as PVCNN (0.25×C), PVNAS-A achieves 1.2× faster inference speed. It also outperforms PointNet by 1.5 mIoU with **3**× measured speedup. In comparison with PVCNN (0.5×C), PVNAS-B achieves 1.4× speedup with no loss of accuracy, and PVNAS-C achieves 1.1× speedup with better mIoU.

PV/SPVConv. On ShapeNet, PVCNN/PVNAS achieves far better efficiency-accuracy trade-offs compared with SPVCNN/SPVNAS. Specifically, PVCNN achieves similar or better mIoU than SPVCNN with **2.4**× measured speedup. Similarly, PVNAS-A achieves the same accuracy as SPVNAS with **2.6**× lower latency. These results validate our claim in Section 3.1.5 that PVConv is more favorable for modeling small 3D objects.

Visualization. We visualize the voxel and point features from the final PVConv, where warmer color indicates larger magnitude. As in Figure 3.5, the voxel branch tends to capture large, continuous parts (*e.g.*, table top, lamp head) while the point branch captures isolated, discontinuous details (*e.g.*, table legs, lamp neck). Two branches indeed provide complementary information. This is aligned with our design.

PartNet

We also conduct experiments on the more challenging fine-grained 3D object part segmentation benchmark, PartNet [216]. Different from ShapeNet, where each object is annotated with 2 to 6 parts, objects in PartNet can have as many as 50 parts. To perform fine-grained segmentation, the models usually take in a batch of 10,000 points instead of 2,048 points.

Baselines. We compare PVCNN with state-of-the-art point-based methods including PointNet [146], PointNet++ [151], Deep LPN [220], SpiderCNN [174], PointCNN [152] and ResGCN [221]. We also compare PVNAS with automatically-designed point cloud segmentation networks including SGAS [211] and LC-NAS [222]. To ensure fair comparisons, we follow the original experiment setting in Mo *et al*. [216] to train separate models for different object classes. The results of our method are averaged over at least three runs. For the other methods, we report the accuracy from their papers and measure #Params, #MACs and latency with their open-source implementation.

Results. As in Table 3.3, PVCNN achieves superior results compared with all manually-designed point-based methods. Specifically, PVCNN (0.5×C) outperforms PointCNN with **27.7**× model size reduction, **31.3**× computation reduction, and **23.7**× measured speedup. This indicates that PVConv scales much better (w.r.t. the number of points) than other point-based primitives. Then, we apply 3D-NAS to PVCNN under 4.2/4.9 ms latency constraints on NVIDIA GTX 1080 Ti GPU. The resulting PVNAS outperforms PVCNN (0.72×C) and PVCNN (1×C) with 1.3× and 1.4× speedup, respectively. It also compares

	#Params (M)	#MACs (G)	Memory (GB)	Latency (ms)	mIoU
PointNet [146]	1.2	3.6	1.0	12.1	43.0
PVCNN (0.125×C)	0.04	0.3	**0.6**	**6.2**	**46.9**
DGCNN [153]	1.0	36.9	2.4	168.1	48.0
RSNet [181]	6.9	2.2	1.1	111.5	52.0
SPVCNN (0.25×C)	0.2	0.4	1.1	21.1	50.4
PVCNN (0.25×C)	0.2	0.9	**0.7**	**9.0**	**52.3**
3D-UNet [162]	14.0	349.6	6.8	574.7	55.0
SPVCNN (1×C)	2.7	6.6	1.8	41.0	54.9
PVCNN (1×C)	2.6	13.0	1.3	**39.1**	56.1
PVCNN++ (0.5×C)	3.4	6.6	**0.7**	40.9	**57.6**
PointCNN [152]	11.5	17.5	4.6	282.3	57.3
PVCNN++ (1×C)	13.7	26.2	**0.8**	**67.2**	**59.0**

Table 3.4: Results of indoor scene segmentation on S3DIS. On average, PVCNN and PVCNN++ outperform point-based models with **8**× speedup and **3**× memory reduction. They outperform the voxel-based baseline with **14**× speedup and **10**× memory reduction.

favorably with AutoML-based approaches. With more than **29**× speedup, PVNAS achieves similar IoU compared with SGAS and LC-NAS.

PV/SPVConv. On PartNet, PVCNN (0.5×C) is **3.8**× faster than SPVCNN (1×C), while achieving **2.2%** higher accuracy. 3D-NAS improves the performance of SPVCNN significantly; however, SPVNAS is still **3.1**× slower than PVNAS-A. This again emphasizes the importance of primitive selection.

Deployment. We deploy our PVCNN and PVNAS on edge devices. As in Figure 3.6a, PVCNN runs at real time (20 FPS) on NVIDIA Jetson Nano, which only consumes the power of a light bulb (5 W). Even when the number of input points increases by 5× on PartNet, PVNAS can still run at 8.3 FPS on NVIDIA Jetson Nano (Figure 3.6b). Thus, PVNAS can empower efficient 3D vision on low-power devices, which becomes increasingly important with the rise of AR/VR applications.

3D Indoor Scene Segmentation

We then evaluate our method on 3D semantic segmentation and conduct experiments on the large-scale indoor scene dataset, S3DIS [217, 218]. Though indoor scenes are usually much larger (*e.g.*, 6m×6m×3m) than single 3D objects, it is still affordable to process them using sliding windows (*e.g.*, 1.5m×1.5m×3m). Note that the evaluation protocol

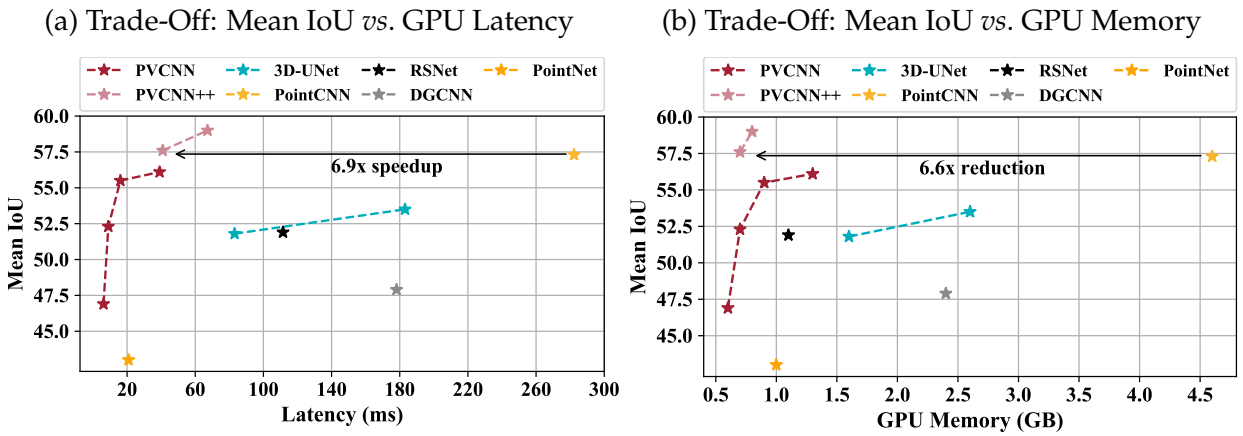


Figure 3.7: PVCNN achieves a much better trade-off between accuracy and efficiency than the point-based and voxel-based baselines on S3DIS.

is different from recent methods [130] that take in the entire scene. Our setting is closer to the real-world scenario since depth cameras only capture a small region in a single pass. Similar to object part segmentation, we build PVCNN based on PointNet [146] and PVCNN++ based on PointNet++ [151].

Baselines. We compare our models with the state-of-the-art point-based models [146, 152, 153, 181] and the voxel-based baseline [162]. Following Tchapmi *et al*. [177] and Li *et al*. [152], we train the models on area 1,2,3,4,6 and test them on area 5 because it is the only one that does not overlap with any other area. Both data processing and evaluation protocol are the same as PointCNN [152] for fair comparison. We measure the latency and memory consumption with 32768 points per batch on a single NVIDIA GTX 1080 Ti GPU.

Results. As in Table 3.4, PVCNN improves its backbone (*i.e*., PointNet) by more than **13%** in mIoU and outperforms DGCNN (which involves sophisticated graph convolutions) by a large margin in both accuracy and latency. Remarkably, our PVCNN++ outperforms the state-of-the-art point-based model (PointCNN) by 1.7% in mIoU with **4**× lower latency, and the voxel-based baseline (3D-UNet) by 4% in mIoU with more than **8**× lower latency and GPU memory consumption. Similar to object part segmentation, we also design compact models by reducing the number of channels in our PVCNN to 12.5%, 25% and 50% and our PVCNN++ to 50%. Remarkably, the narrower version of our PVCNN outperforms DGCNN with **15**× measured speedup, and RSNet with **9**× measured speedup. Furthermore, it achieves 4% improvement in mIoU over PointNet while still being **2.5**× faster than this extremely efficient 3D model (which does not have any neighborhood aggregation). We refer the readers to Figure 3.7 for accuracy *vs*. latency and accuracy *vs*.

	#Params (M)	#MACs (G)	Latency (ms)	mIoU
PointNet [146]	3.0*	–	500*	14.6
SPGraph [179]	0.3*	–	5200*	17.4
PointNet++ [151]	6.0*	–	5900*	20.1
TangentConv [171]	0.4*	–	3000*	40.9
RandLA-Net [184]	1.2	66.5	880 (256+624)†	53.9
KPConv [223]	18.3	207.3	–	58.8
MinkowskiNet [130]	21.7	114.0	294	63.1
PVCNN	2.5	42.4	146	39.0
SPVCNN	21.8	118.6	317.1	65.3
SPVNAS-A	2.6	15.0	110	63.7
SPVNAS-B	12.5	73.8	259	**66.4**

Table 3.5: Results of outdoor scene segmentation on SemanticKITTI (3D). SPVNAS outperforms MinkowskiNet with **2.7**× speedup. †: computation time + post-processing time. *: results from Behley *et al*. [214].

memory trade-offs.

PV/SPVConv. With the same network structure, PVCNN is more effective than SPVCNN, achieving 1.2-2.1% higher mIoU (Table 3.4). Similar to our results on object segmentation, SPVCNN fails to achieve large speedups with small channel numbers: with 4× channel reduction, SPVCNN achieves only 1.9× speedup while PVCNN achieves 4.3× speedup. Thus, PVCNN is a superior choice for indoor scene segmentation.

3D Outdoor Scene Segmentation

SemanticKITTI

We evaluate our method on 3D semantic segmentation and conduct experiments on the large-scale outdoor scene dataset, SemanticKITTI [214]. This dataset contains 23,201 LiDAR point clouds for training and 20,351 for testing, and it is annotated from all 22 sequences in the KITTI [228] Odometry benchmark. We train all models on the entire training set and report the mean intersection-over-union (mIoU) on the official test set under the single scan setting. We measure the latency on a single NVIDIA GTX 1080 Ti GPU with batch size of 1. We use `TorchSparse v1.0.0`* as our inference backend.

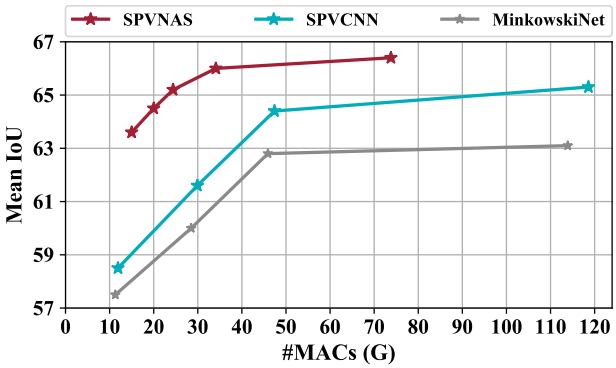
Results. As in Table 3.5, SPVNAS outperforms the previous state-of-the-art MinkowskiNet [130] by **3.3%** in mIoU with 1.7× model size reduction, 1.5× computation reduction

*https://github.com/mit-han-lab/torchsparse

	#Params (M)	#MACs (G)	Latency (ms)	mIoU
DarkNet21Seg [214]	24.7	212.6	73 (49+24)†	47.4
DarkNet53Seg [214]	50.4	376.3	102 (78+24)†	49.9
SqueezeSegV3-21 [224]	9.4	187.5	97 (73+24)†	51.6
SqueezeSegV3-53 [224]	26.2	515.2	238 (214+24)†	55.9
3D-MiniNet [225]	4.0	–	–	55.8
PolarNet [226]	13.6	135.0	62	57.2
SalsaNext [227]	6.7	62.8	71 (47+24)†	59.5
SPVNAS	**1.1**	**8.9**	89	**60.3**

Table 3.6: Results of outdoor scene segmentation on SemanticKITTI (2D). SPVNAS outperforms the 2D projection-based methods with over **7.1**× computation reduction. (†: computation time + projection time)

(a) Trade-Off: Mean IoU *vs*. #MACs



(b) Trade-Off: Mean IoU *vs*. GPU Latency

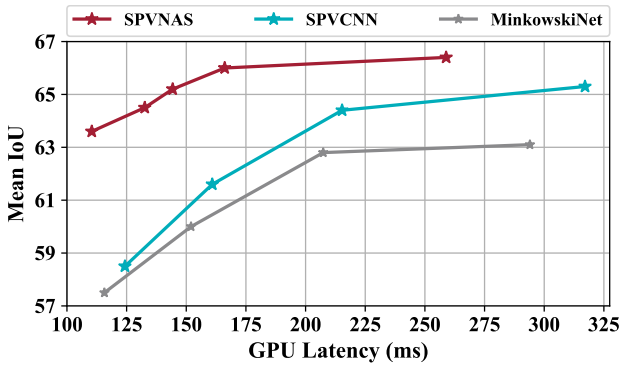


Figure 3.8: An efficient 3D primitive (SPVConv) and a well-designed network architecture (3D-NAS) are both important to the performance of SPVNAS.

and 1.1× measured speedup. Further, we downscale our SPVNAS by setting the resource constraint to 15G MACs. This offers us with a much smaller model that outperforms MinkowskiNet with **8.3**× model size reduction, **7.6**× computation reduction, and **2.7**× measured speedup.

In Table 3.6, we compare SPVNAS with 2D projection-based models. With a smaller backbone (by removing the decoder layers), SPVNAS outperforms DarkNets [214] by more than **10%** in mIoU with 1.2× speedup even though 2D CNNs are much better optimized by modern deep learning libraries. Compared with other 2D methods, our SPVNAS achieves at least **8.5**× model size reduction and **15.2**× computation reduction while being much more accurate. Furthermore, SPVNAS achieves higher mIoU than KPConv [223], which is the previous state-of-the-art point-based model, with **17**× model size reduction and **23**× computation reduction.

	#Params (M)	#MACs (G)	Latency (ms)	Mean IoU
PolarSeg [226]	13.6	45.6	**42.8**	47.8
MinkowskiNet [130]	21.7	22.2	81.0	53.7
SPVCNN	21.8	23.1	89.9	54.7
SPVNAS	**9.6**	**10.6**	59.1	**54.7**

Table 3.7: Results of outdoor scene segmentation on nuScenes. SPVNAS outperforms MinkowskiNet with **2.1**× computation reduction and **1.4**× measured speedup.

Analysis. In Figure 3.8, we present both mIoU *vs*. #MACs and mIoU *vs*. latency trade-offs, where we uniformly scale the channel numbers in MinkowskiNet and SPVCNN down as our baselines. It can be observed that a better 3D module (SPVNAS) and a well-designed 3D network architecture (3D-NAS) are equally important to the final performance boost. Remarkably, SPVNAS outperforms MinkowskiNet by more than 6% in mIoU at 110 ms latency. Such a large improvement comes from non-uniform channel scaling and elastic network depth. In these manually-designed models (MinkowskiNet and SPVCNN), 77% of the total computation is distributed to the upsampling stage. With 3D-NAS, this ratio is reduced to 47-63%, making the computation more balanced and the downsampling stage (feature extraction) more emphasized.

nuScenes

The distribution of point clouds varies significantly across different sensors. To verify the generalizability of our SPVCNN and SPVNAS, we also conduct experiments on a recent 3D segmentation benchmark, nuScenes [73]. The dataset contains 34,149 LiDAR point clouds for training and 6,008 for testing, and provides point-level semantic annotations for each point cloud. We follow the official instruction to split the training set into `train` split (consisting of 28,130 LiDAR point clouds) and `val` split (consisting of 6,019 LiDAR point clouds). We train all manually-designed models on the `train` split and evaluate them on the `val` split. For our SPVNAS, we search the best model on a 20% holdout `minival` split from the `train` split and perform evaluation on the `val` split. We report the official 31-class mIoU as the evaluation metric. Similar to SemanticKITTI, we measure the latency on an NVIDIA GTX 1080 Ti GPU with batch size of 1.

Results. As in Table 3.7, SPVNAS achieves a better accuracy *vs*. efficiency trade-off than both PolarSeg [226] and MinkowskiNet [130]. It outperforms MinkowskiNet by **1.0%** in mIoU with **2.3**× model size reduction, **2.1**× computation reduction, and **1.4**× speedup.

	Memory (G)	Latency (ms)	Car			Cyclist			Pedestrian		
			Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
F-PointNet	1.3	29.1	85.2	71.6	63.8	77.1	56.5	52.8	66.4	56.9	50.4
F-PointNet++	2.0	105.2	84.7	72.0	64.2	75.6	56.7	53.3	68.4	60.0	52.6
PVCNN	1.4	58.9	**85.3**	**72.1**	**64.2**	**78.1**	**57.5**	**53.7**	**70.6**	**61.2**	**56.3**

Table 3.8: Results of Outdoor Object Detection on KITTI (Two-Stage). PVCNN outperforms F-PointNet++ in all categories significantly with **1.8**× measured speedup and **1.4**× memory reduction.

	Car			Cyclist			Pedestrian		
	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
SECOND [61]	89.8	80.9	78.4	82.5	62.8	58.9	**68.3**	60.8	55.3
SPVCNN	**90.9**	**81.8**	**79.2**	**85.1**	**63.8**	**60.1**	68.2	**61.6**	**55.9**

Table 3.9: Results of one-stage outdoor object detection on KITTI. SPVCNN outperforms SECOND in most categories especially in cyclists.

It achieves **6.9%** mIoU improvement over the projection-based PolarSeg. Similar to our observations on SemanticKITTI, both efficient 3D primitive design (SPVConv, **+1.0** mIoU) and network architecture search (3D-NAS, **2.2**× computation reduction, **1.5**× measured speedup) contribute to the accuracy and efficiency boost of SPVNAS.

3D Outdoor Object Detection

We evaluate our method on 3D object detection and conduct experiments on the large-scale outdoor scene dataset, KITTI [228]. We follow the conventional training-validation split, where 3,712 samples are used for training and 3,769 samples are left for validation. We report the 3D average precision (with 40 recall positions) on the validation set under IoU thresholds of 0.7 for car, 0.5 for cyclist and pedestrian.

Results. We first apply our method to F-PointNet [182], which is a two-stage 3D object detection framework. It first leverages the off-the-shelf 2D object detector to produce frustrum proposals. For each frustrum, it then applies PointNets to classify the content. As the frustrums are relatively small, it is suitable to process them using PVConv. Thus, we build our PVCNN based on F-PointNet by replacing the MLP layers within its instance segmentation network with PVConv and keep its box proposal and refinement networks unchanged. We compare our PVCNN with F-PointNet (whose backbone is PointNet) and

F-PointNet++ (whose backbone is PointNet++). In Table 3.8, even if our PVCNN does not aggregate neighboring features in the box estimation network while F-PointNet++ does, PVCNN still outperforms it in all classes with **1.8**× lower latency. Remarkably, our model achieves **2.4%** average mAP improvement in the most challenging pedestrian class. Compared with F-PointNet, our PVCNN obtains up to **4-5%** mAP improvement in pedestrians, which indicates that our model is both efficient and expressive.

Apart from the frustrum-based framework, we also apply our method to SECOND [61], which is a single-stage 3D object detection framework. It consists of a sparse encoder using 3D sparse convolutions and a region proposal network that performs 2D convolutions after projecting the encoded features into the bird's-eye view (BEV). Unlike F-PointNet, SECOND is applied to the entire outdoor scenes, which are of large scale. Therefore, we choose to replace the sparse encoder with SPVCNN. As a fair comparison, we reimplement and retrain SECOND, which already outperforms the results claimed in the original paper [61]. As summarized in Table 3.9, our SPVCNN achieves consistent improvements in all categories, especially in cyclist and pedestrian. This is because the high-resolution point-based branch carries more information for small objects.

3.1.7 Discussion

We studied 3D deep learning from the hardware efficiency perspective. We systematically analyzed the bottlenecks of previous point-based and voxel-based models and proposed a novel hardware-efficient 3D primitive to combine the best from both. We further enhanced this primitive with the sparse convolution to make it more effective for large (outdoor) scenes. Based on our designed 3D primitive, we then introduced the 3D neural architecture search framework to automatically explore the best network architecture that satisfies a given resource constraint. Extensive experiments on multiple 3D benchmark datasets validate the efficiency and effectiveness of our proposed method.

3.2 FlatFormer

Transformer, as an alternative to CNN, has been proven effective in many modalities (*e.g.*, texts and images). For 3D point cloud transformers, existing efforts focus primarily on pushing their accuracy to the state-of-the-art level. However, their latency lags behind sparse convolution-based models (**3**× **slower**), hindering their usage in resource-constrained, latency-sensitive applications (such as autonomous driving). This inefficiency comes from point clouds' sparse and irregular nature, whereas transformers are designed

for dense, regular workloads. We present **FlatFormer** to close this latency gap by trading spatial proximity for better computational regularity. We first flatten the point cloud with window-based sorting and partition points into **groups of equal sizes** rather than **windows of equal shapes**. This effectively avoids expensive structuring and padding overheads. We then apply self-attention within groups to extract local features, alternate sorting axis to gather features from different directions, and shift windows to exchange features across groups. FlatFormer delivers state-of-the-art accuracy on Waymo Open Dataset with **4.6**× speedup over (transformer-based) SST and **1.4**× speedup over (sparse convolutional) CenterPoint. This is the first point cloud transformer that achieves real-time performance on edge GPUs and is faster than sparse convolutional methods while achieving on-par or even superior accuracy on large-scale benchmarks.

3.2.1 Introduction

Transformer [3] has become the model of choice in natural language processing (NLP), serving as the backbone of many successful large language models (LLMs) [4, 16]. Recently, its impact has further been expanded to the vision community, where vision transformers (ViTs) [29, 33, 34] have demonstrated on-par or even superior performance compared with CNNs in many visual modalities (*e.g.*, image and video). 3D point cloud, however, is not yet one of them.

Different from images and videos, 3D point clouds are intrinsically sparse and irregular. Most existing point cloud models [229] are based on 3D sparse convolution [120], which computes convolution only on non-zero features. They require dedicated system support [25, 61, 130] to realize high utilization on parallel hardware (*e.g.*, GPUs).

Many efforts have been made toward point cloud transformers (PCTs) to explore their potential as an alternative to sparse convolution. Global PCTs [230] benefit from the regular computation pattern of self-attention but suffer greatly from the quadratic computational cost (w.r.t. the number of points). Local PCTs [231, 232] apply self-attention to a local neighborhood defined in a similar way to point-based models [151] and are thus bottlenecked by the expensive neighbor gathering [22]. These methods are only applicable to single objects or partial indoor scans (with $<$4k points) and cannot be efficiently scaled to outdoor scenes (with $>$30k points).

Inspired by Swin Transformer [29], window PCTs [154, 155] compute self-attention at the window level, achieving impressive accuracy on large-scale 3D detection benchmarks. Despite being spatially regular, these windows could have drastically different numbers of points (which differ by more than **80**×) due to the sparsity. This severe imbalance results in

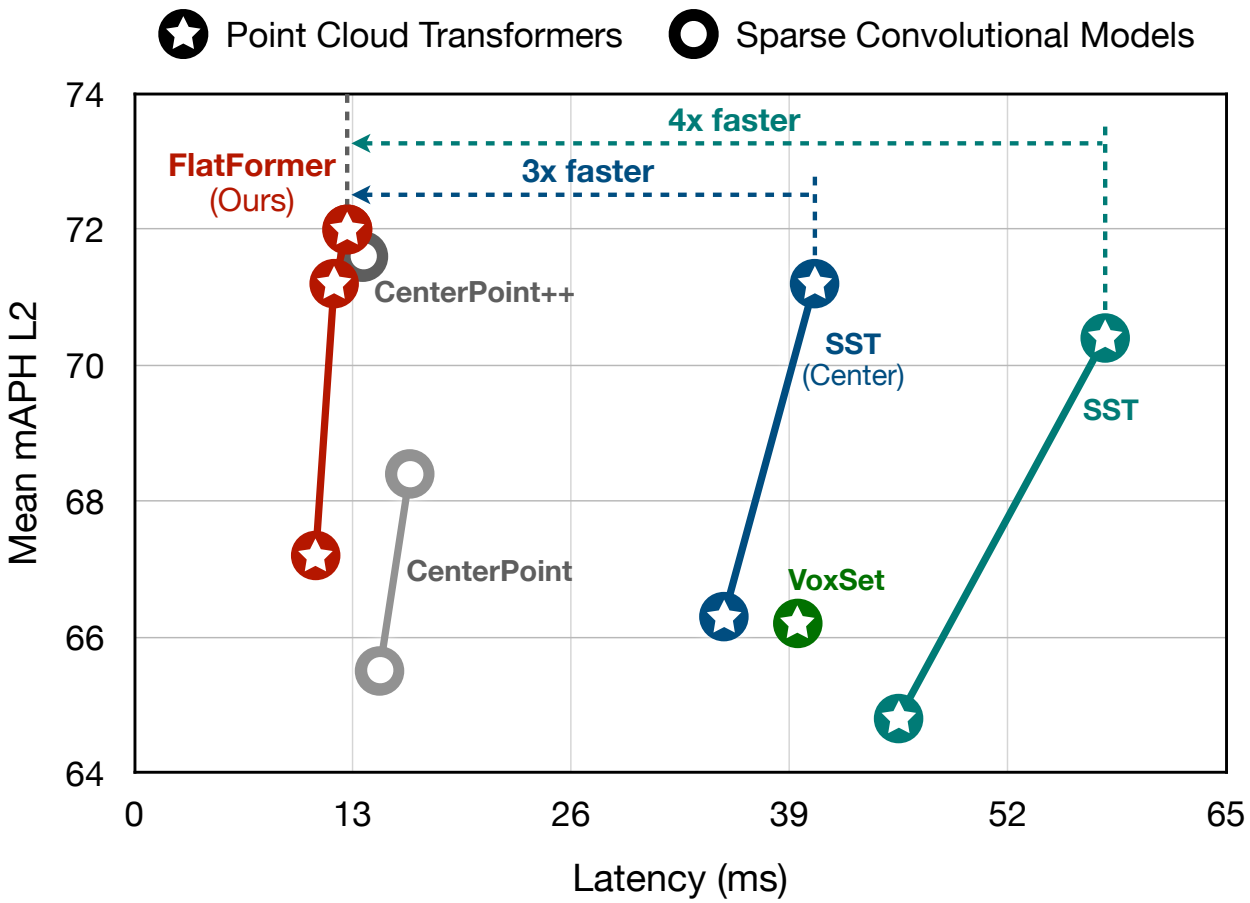


Figure 3.9: Previous point cloud transformers (⋆) are **3-4**× slower than sparse convolution-based models (●) despite achieving similar detection accuracy. FlatFormer is the first point cloud transformer that is faster than sparse convolutional methods with on-par accuracy. Latency is measured on an NVIDIA Quadro RTX A6000.

redundant computation with inefficient padding and partitioning overheads. As a result, window PCTs can be **3**× **slower** than sparse convolutional models (Figure 3.9), limiting their applications in resource-constrained, latency-sensitive scenarios (*e.g.*, autonomous driving, augmented reality).

We introduce **FlatFormer** to close this huge latency gap. Building upon window PCTs, FlatFormer trades spatial proximity for better computational regularity by partitioning 3D point cloud into *groups of equal sizes* instead of *windows of equal shapes*. It applies self-attention within groups to extract local features, alternates the sorting axis to aggregate features from different orientations, and shifts windows to exchange features across groups. Benefit from the regular computation pattern, FlatFormer achieves **4.6**× speedup over (transformer-based) SST and **1.4**× speedup over (sparse convolutional) CenterPoint while delivering the state-of-the-art accuracy on Waymo Open Dataset.

To the best of our knowledge, FlatFormer is the first point cloud transformer that achieves on-par or superior accuracy than sparse convolutional methods with lower latency. It is also the first to achieve real-time performance on edge GPUs. With better hardware support for transformers (*e.g.*, NVIDIA Hopper), point cloud transformers will have a huge potential to be the model of choice in 3D deep learning. We believe our work will inspire future research in this direction.

3.2.2 Related Work

Deep Learning on Point Clouds

Early research converts point clouds from 3D sensors to dense voxel grids and applies 3D CNNs [162, 164, 166, 167] on the volumetric inputs. However, the compute and memory consumption of volumetric CNNs grows cubically w.r.t. the input resolution, limiting the scalability of these methods. To overcome this bottleneck, later research [146, 151–153, 184, 215, 223, 233] directly performs feature extraction on point sets, while [161, 170, 185] convert point clouds to octrees and [61, 120, 130, 234, 235] perform sparse convolution on sparse voxels. Recently, researchers also explore point+voxel [22, 23, 236, 237] or point+sparse voxel [53, 236, 238, 239] hybrid representations for efficient 3D deep learning.

3D Object Detection

Extensive attention has been paid to 3D object detection [73, 228, 240] for autonomous vehicles. Early research [182, 241] generates object proposals on 2D images and refines the predictions in the lifted 3D frustums. VoxelNet [169] leads another line of research that directly detects 3D objects without 2D proposals. Following VoxelNet, PointPillars [242], SECOND [61, 243], 3DSSD [244] and MVF [245] are all single-stage anchor-based 3D detectors, while CenterPoint [229, 246], AFDet [247, 248], HotspotNet [249], MVF++ [250], RangeDet [251], PolarStream [252], ObjectDGCNN [253], M3DETR [254], PillarNet [255], LidarMultiNet [256] are single-stage anchor-free 3D detectors. PointRCNN [183], Fast Point R-CNN [257], Part-A^2Net [258], PV-RCNN [238, 239], LiDAR R-CNN [259], CenterFormer [260], FSD [261], MPPNet [262] add a second stage that refines the proposals from the region proposal network (RPN) in the 3D space. There are also recent explorations on multi-sensor 3D object detection [26, 263–269].

Vision Transformers

Motivated by the huge success of transformers [3, 4] in natural language processing (NLP), researchers have started to adapt transformers to various vision tasks [32]. The pioneering ViT [33] first demonstrates that an image can be viewed as 16×16 words and processed by multi-head attention layers. DeiT [34] further shows that ViTs can be trained in a data-efficient manner without pretraining on JFT [270]. T2T-ViT [35], Pyramid ViT [36, 37] and CrossFormer [38] introduce hierarchical modeling capability to ViTs. Swin Transformer [29, 39] limits self-attention computation to non-overlapping windows and enables cross-window information exchange via window shifting. There are also task-specific ViTs such as ViTDet [40] for object detection, SETR [41], and SegFormer [42] for semantic segmentation. Instead of adopting a fully-transformer backbone, another line of research, such as DETR [271], Deformable DETR [272], MaskFormer [79], PanopticSegFormer [273], DETR3D [274], BEVFormer [275], apply self-attention only to the task-specific heads and still uses CNNs for the backbone.

Point Cloud Transformers

Recently, fully-transformer architectures have begun to emerge in the point cloud domain. Similar to ViT, PCT [230] calculates self-attention globally on the entire point cloud, which falls short in scalability as its computation complexity scales quadratically as the number of points grows. PointASNL [276], PointTransformer [231, 277], Fast Point Transformer [278], PointFormer [279], VoTr [232], VoxSet [280] applies transformer-based architecture on the local neighborhood of each point. The efficiency of these local transformers is limited by neighborhood query and feature restructuring. Most related to our work are the window-based point cloud transformers, SST [154] and SWFormer [155]. Inspired by Swin Transformer, they project the point cloud into a bird's-eye view and divide the BEV space into non-overlapping windows with the same *spatial sizes* (but different *number of points*). Window shifting is used to communicate information across windows. SST suffers from large computation in window partition and padding overhead due to regional grouping, and achieves only **one-sixth** utilization compared with sparse convolutional models.

We only refer to those methods that adopt a transformer-based architecture in the *backbone* as point cloud transformers. As CenterFormer [260], FUTR3D [281] and UVTR [282] apply sparse convolutional backbones and only use the transformer in their detection heads, we still categorize them as sparse convolutional methods.

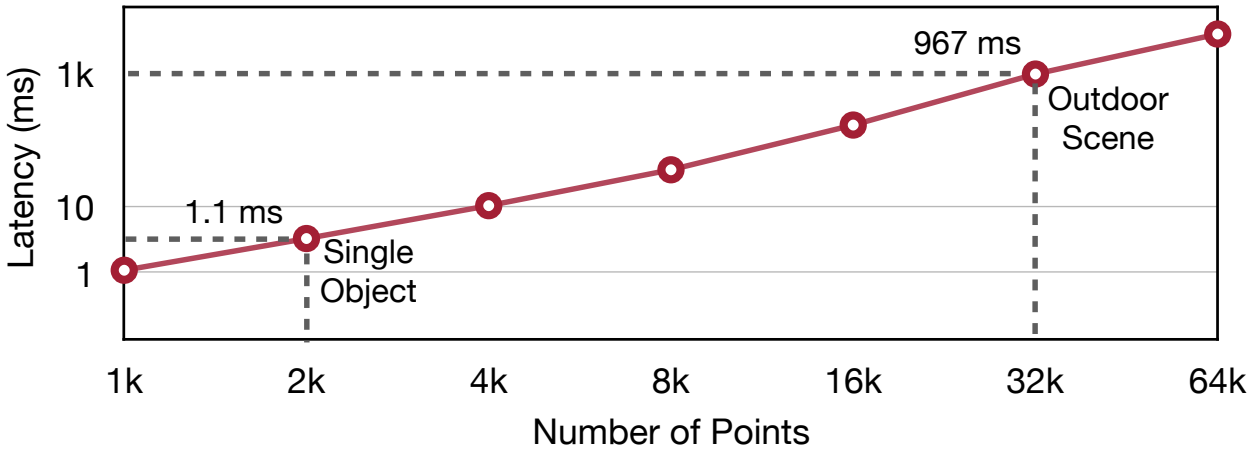


Figure 3.10: Latency of global PCTs scale quadratically with respect to the number of input points and cannot scale up to outdoor scenes.

3.2.3 Why are Sparse Point Cloud Transformers Slow?

Although sparse point cloud transformers (PCTs) start to catch up with the accuracy of sparse convolutional detectors, there is still a 3× latency gap between the fastest PCT (SST [154]) and sparse convolutional CenterPoint [229] (Figure 3.9). In this section, we dissect the efficiency bottleneck of PCTs, which lays a solid foundation for our FlatFormer design.

Global PCTs

Inspired by ViT [33], the most simple and straightforward design for transformers on point cloud is global PCTs [230], where each point is a token. They leverage multi-head self-attention (MHSA) [3] globally across the entire point cloud. While being effective on small-scale 3D objects, global PCTs fall short in scaling to large-scale scenes due to its $\mathcal{O}(N^2D)$ complexity, where N is the number of tokens and D is the number of channels. From Figure 3.10, the runtime of global PCTs [230] grows quadratically as the number of input points grows. For example, with 32k input points[†], the model takes **almost one second** to execute on an NVIDIA A6000 GPU, **66**× slower than CenterPoint [229].

Local PCTs

Researchers have proposed local PCTs [231, 232, 276–279] to solve the scalability issue of global PCTs. They apply MHSAs to the neighborhood of each point rather than the entire point cloud. Hence, their computational complexity is $\mathcal{O}(NK^2D)$, where N is the

[†]32k is the number of points left after 0.32m×0.32m BEV projection in a single-frame Waymo [240] scene.

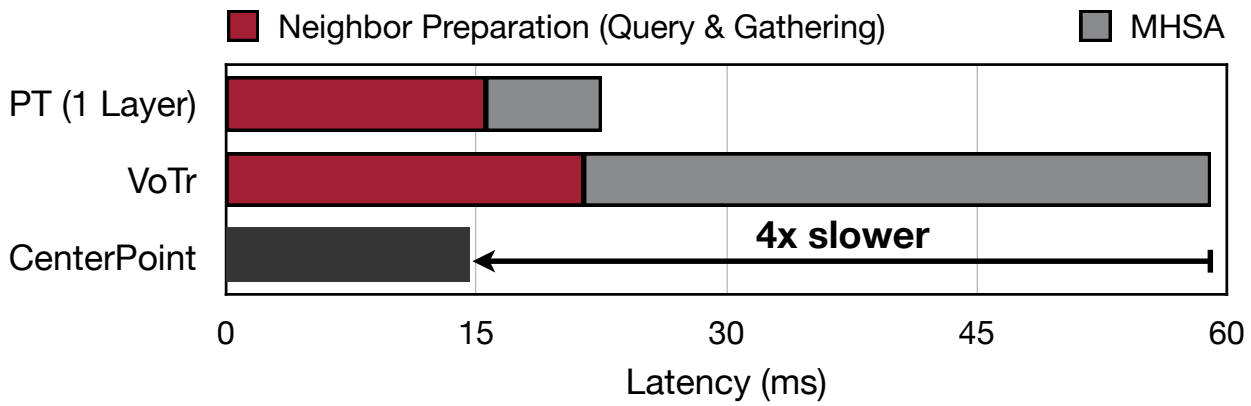


Figure 3.11: Local PCTs suffer from large neighborhood query and data restructuring overhead.

number of points, K is the number of neighbors for each point, and D is the number of channels. As N ranges from 20k to 100k for real workloads and K is less than 64 for local PCTs, their theoretical cost is much lower than global PCTs.

Local PCTs, however, suffer greatly from neighbor preparation overheads. As point cloud is sparse and irregular, we have to first *find the neighbors* of each point, and then re-*structure the data* from the $N \times D$ format to the $N \times K \times D$ format on which MHSAs can be applied. These two steps are slow, taking **22** ms (*i.e.*, **36%** of the total runtime) for VoTr [232] to execute for a single scene on Waymo, which is already slower than the entire CenterPoint model. For Point Transformer (PT) [231], the cost for preparing neighbors takes up to **70%** of the runtime. Such overhead in a *single* layer is already larger than the total runtime of CenterPoint!

Window PCTs

The great success of Swin Transformers [29, 39] in various visual recognition tasks motivates the design of window PCTs, among which, SST [154] is a representative work. It first projects the point cloud into the bird's-eye-view (BEV) space, then divides the BEV space into equally-shaped, non-overlapping windows, and applies MHSA within each window. Similar to Swin Transformer, SST uses window shifting to enable information exchange across windows.

Different from images, point clouds are sparse and non-uniformly distributed over the space. As a result, the number of point within each window is not the same and can differ by two orders of magnitude (Figure 3.12). As the vanilla MHSA kernel cannot efficiently support variable sequence lengths, SST [154] batches windows with similar sizes together and pad all windows in each batch to the largest group size within the batch (Figure 3.13l).

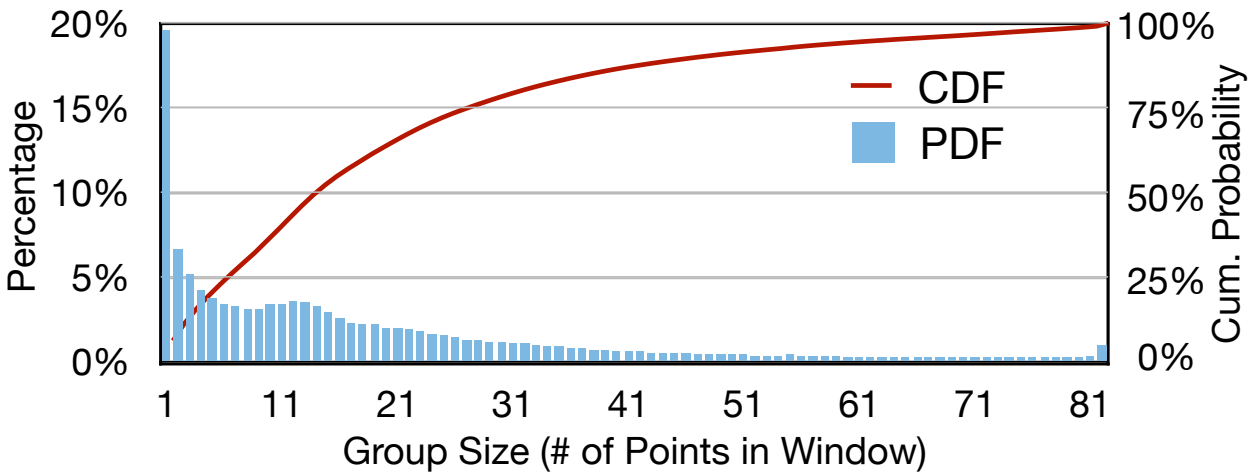


Figure 3.12: In SST [154], the number of points within each window has a large variance. Therefore, padding is necessary and leads to significant overhead for MHSA computation.

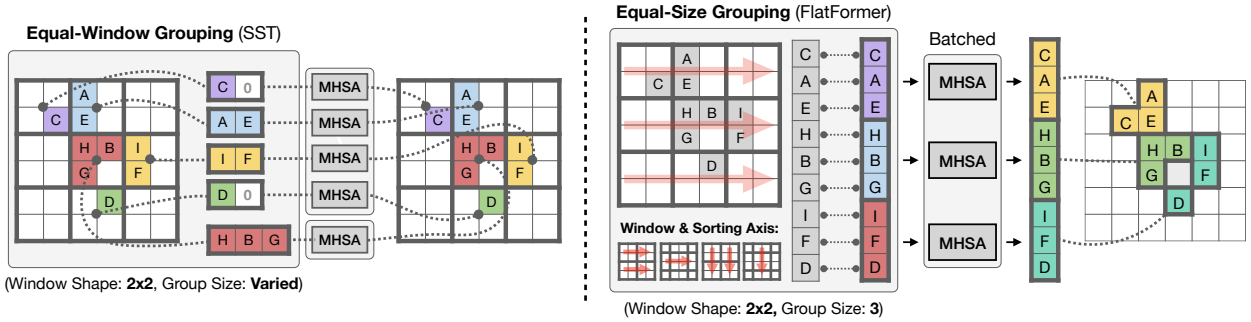


Figure 3.13: FlatFormer partitions the point cloud into groups of equal sizes (*right*), rather than windows of equal shapes (*left*). This effectively trades *spatial proximity* for better *computational regularity*. It then applies self-attention within each group to extract local features, alternates the sorting axis to aggregate features from different directions, and shifts windows to exchange features across groups.

It then applies MHSA within each batch separately. In practice, such padding introduces a **1.7**× computation overhead on Waymo. Worse still, partitioning points to equal windows also introduce significant latency overhead: it takes **18 ms** per scene on Waymo, even slower than the total runtime of CenterPoint. To sum up, the padding and partitioning overheads make SST less hardware-friendly compared with sparse convolutional methods.

3.2.4 Method

With all the lessons learned in Section 3.2.3, we will design our point cloud transformer to be scalable and efficient.

Overview

The basic building block of FlatFormer is Flattened Window Attention (FWA). As in Figure 3.13r, FWA adopts *window-based sorting* to flatten the point cloud and partitions it to *groups of equal sizes* rather than *windows of equal shapes*. This naturally resolves the group size imbalance problem and avoids the padding and partitioning overheads. FWA then applies self-attention within groups to extract local features, alternates sorting axis to aggregate features from different orientations, and shifts windows to exchange features across groups. Finally, we provide an implementation of FWA that further improves its efficiency and minimizes the overheads.

Sorting & Grouping

Window-Based Sorting. With a point cloud $\{(x, y)\}$[‡], we first quantize the coordinate of each point (x, y) to

$$\big(\underbrace{\lfloor x/w_x \rfloor, \lfloor y/w_y \rfloor}_{\text{window coordinates}}, \underbrace{x - \lfloor x/w_x \rfloor \cdot w_x, y - \lfloor y/w_y \rfloor \cdot w_y}_{\text{local coordinates within window}}\big), \tag{3.4}$$

where (w_x, w_y) is the window shape. Next, we sort all points first by *window coordinates* and then by *local coordinates* within the window. This step turns the unordered point cloud into an ordered one, where points within the same window will be next to each other.

Equal-Size Grouping. Conventional window PCTs [154] will then group the points within the same window together. However, as discussed in Section 3.2.3, each group can have drastically different numbers of points due to inherent sparsity. To overcome the padding overheads, we partition the point cloud into *groups of equal sizes* based on the sorted sequence. This step allows the subsequent group attention to enjoy a perfectly regular workload. From the implementation perspective, our grouping only involves a simple tensor reshaping (which is free since it does not change the layout) and is more efficient than window partitioning in SST [154].

Alternate Sorting Axis. Between the two axes, x has a higher priority in sorting. Thus, points with identical $\lfloor x/w_x \rfloor$ will be next to each other while points with the same $\lfloor y/w_y \rfloor$ can be very far away from each other in the sorted sequence, breaking the geometric locality. To solve this inequity, we alternate the sorting axis between x and y in different

[‡]We assume that the point cloud is in 2D for ease of notation, while our method applies to 3D or higher-dimension point clouds.

FWA blocks. This is very similar to spatially separable convolution that decomposes a 3×3 kernel into 3×1 and 1×3 kernels. Stacking FWA blocks with different sorting axes enables the model to aggregate local features from different directions.

Equal Size *vs.* Equal Window. The key design choice we made is to partition the point cloud into groups of equal sizes rather than windows of equal shapes. There is a trade-off: equal-window grouping maintains perfect *spatial proximity* (*i.e.*, each group has the same radius) but breaks the *computational regularity*, while equal-size grouping ensures balanced computation workload (*i.e.*, each group has the same number of points) but cannot guarantee the geometric locality. We show in Section 3.2.6 that computation regularity is more important since spatial irregularity can be partially addressed by our algorithm design: *i.e.*, window-based sorting offers a fairly good spatial ordering, and self-attention is robust to outliers (*i.e.*, distant point pairs).

Group Attention

With points partitioned, we then apply self-attention [3] within each group to extract local features. For each group of points with coordinates $\mathcal{C}$ and features $\mathcal{F}$, we follow the standard transformer block design:

$$\begin{aligned} \mathcal{F}' &= \mathcal{F} + \texttt{MHSA}(\texttt{LN}(\mathcal{F}), \texttt{PE}(\mathcal{C})), \\ \mathcal{F}'' &= \mathcal{F}' + \texttt{FFN}(\texttt{LN}(\mathcal{F}')), \end{aligned} \tag{3.5}$$

where $\texttt{MHSA}(\cdot)$, $\texttt{FFN}(\cdot)$ and $\texttt{LN}(\cdot)$ correspond to multi-head self-attention, feed-forward layer, and layer normalization, respectively. Different from SST [154], $\texttt{PE}(\cdot)$ gives global absolute positional embedding. Here, we use the most standard `softmax` attention formulation for $\texttt{MHSA}(\cdot)$. Our method will benefit from other more efficient attention variants, such as linear attention [283], which we leave for future work.

Window Shift. Benefit from the non-overlapping design, window-based attention typically has a larger receptive field than convolution (*e.g.*, 69 neighbors in our FWA *vs.* $\leq$27 neighbors in a sparse convolution of kernel size 3). However, its modeling power is limited as there is no feature exchange across groups. Similar to Swin Transformer [29, 39, 154], we adopt the shifted window approach that alternates the sorting configuration in consecutive FWA blocks. Specifically, we translate the coordinates of all points by $(w_x/2, w_y/2)$ for sorting in shifted FWA blocks. This mechanism introduces cross-group feature communication while effectively maintaining workload independence between groups. Note that

alternating sorting axis also enables feature exchange.

Efficient Implementation

Besides the algorithm design, we also provide an implementation that improves the efficiency of MHSA and FFN and minimizes the sorting and masking overheads. All these optimizations are specialized for our point cloud transformer design and are not applicable to sparse convolution models.

Efficient MHSA. Within MHSA, query $\mathcal{Q}$, key $\mathcal{K}$, and value $\mathcal{V}$ will first be transformed with separate linear layers. We pack these three linear projections into a batched matrix multiplication (since $\mathcal{Q}$, $\mathcal{K}$ and $\mathcal{V}$ have the same shape in our FWA) to improve the parallelism. In addition, standard attention implementations materialize $\mathcal{Q}\mathcal{K}^{\mathrm{T}}$ and $\texttt{softmax}(\mathcal{Q}\mathcal{K}^{\mathrm{T}})$. We leverage a recent efficient functional-preserving implementation (FlashAttention [284]) that uses tiling to reduce the number of memory reads/writes, achieving better efficiency.

Efficient FFN. FFN consists of two linear layers with a `GELU` activation in the middle. We implement a fused linear kernel (in `Triton`) that absorbs the activation into the layer before to avoid writing the intermediate results to DRAM. We also observe that our linear kernel (optimized by `Triton`) is even more efficient than `cuBLAS`, which is probably due to the unconventional tall-and-skinny workload.

Reuse Sorting. Sorting the coordinates of all points is a non-negligible overhead. As the coordinates remain identical (w/o downsampling), we reuse the sorting results (*i.e.*, ranks of each point) with the same axis and window. In practice, this reduces the sorting overhead in our model by 50%.

Drop Residual. The size of the input point cloud might not be divisible by the group size, generating a group with fewer points after partition. This minor irregularity will still result in some overheads in self-attention since we need to introduce masking to correctly zero them out. Instead, we directly drop the final non-full group. This only corresponds to less than 0.1% of all points, having a negligible impact on the model's performance ($<$0.1%).

	#Frames	#MACs (G)	Latency (ms)	Speedup (w.r.t. [229])	Mean L2 (mAP/APH)
SECOND [61][1]	1	–	–	–	61.0 / 57.2
PointPillars [242][1]	1	–	–	–	62.8 / 57.8
○ CenterPoint [229][2]	1	126.9	14.6	1.0×	67.9 / 65.5
• VoTr-SSD [232]	1	110.3	59.1†	0.2×	– / –
• SST [154][3]	1	204.9	45.5	0.3×	68.2 / 64.8
• SST-Center [154]	1	226.4	35.1	0.4×	69.3 / 66.3
• VoxSet [280]	1	189.4	39.5	0.4×	69.1 / 66.2
○ PillarNet [255]	1	138.3	11.1	1.3×	**69.9** / **67.2**
• **FlatFormer** (Ours)	1	177.2	**10.8**	**1.4**×	69.7 / **67.2**
○ CenterPoint [229][2]	2	137.6	16.4	1.0×	70.1 / 68.4
○ PillarNet [255]	2	148.8	**11.6**	**1.4**×	71.5 / 70.0
• **FlatFormer** (Ours)	2	186.6	11.9	**1.4**×	**72.7** / **71.2**
○ CenterPoint [229]	3	144.7	18.3	1.0×	– / –
○ CenterPoint++ [246][2]	3	113.0	13.6	1.3×	73.0 / 71.6
• SST [154][3]	3	250.0	57.8	0.3×	**73.6** / 70.4
• SST-Center [154][4]	3	243.1	40.5	0.5×	72.8 / 71.2
• SWFormer [155]	3	–	20.0‡	0.9×	– / –
• **FlatFormer** (Ours)	3	193.2	**12.7**	**1.4**×	73.5 / **72.0**

Table 3.10: Results of single-stage 3D detectors on Waymo Open Dataset (validation set). FlatFormer achieves 1.4× speedup over CenterPoint and 4.6× speedup over SST while being more accurate. Markers ○ and • refer to sparse convolutional models and point cloud transformers, respectively. Methods with <60 L2 mAPH are marked gray. ([1]: from FSD paper, [2]: from CenterPoint authors, [3]: from SST authors, [4]: reproduced by us, †: projected latency, ‡: latency on NVIDIA Tesla T4)

3.2.5 Experiments

Setup

Dataset. We carry out our experiments on the large-scale Waymo Open Dataset (WOD) [240] with 1150 LiDAR point cloud sequences. Each sequence has 200 frames, collected by a 360° FoV LiDAR sensor at 10 frames per second. There are four foreground classes, three of which (vehicles, pedestrians and cyclists) are used for detection metric evaluation.

Metrics. We follow the official metrics on Waymo to calculate the standard 3D mAP and heading-weighted 3D mAP (mAPH) of all methods. The matching IoU thresholds for vehicle, pedestrian and cyclist are set to default values (0.7, 0.5 and 0.5). Objects are divided into two difficulty levels, where objects with fewer than five laser points or annotated as

	#Frames	Latency (ms)	Mean L1 (mAP/APH)	Mean L2 (mAP/APH)
○ LiDAR R-CNN [259]†	1	–	71.9 / 67.0	65.8 / 61.3
○ PV-RCNN [238]†	1	–	73.4 / 69.6	66.8 / 63.3
○ Part-A^2 [258]†	1	–	73.6 / 70.3	66.9 / 63.8
○ PV-RCNN++ [239]†	1	–	74.8 / 71.0	68.4 / 64.9
○ CenterFormer [260]	1	33.8	75.4 / 73.0	71.2 / 69.0
○ FSD-SpConv [261]	1	47.8	**79.6** / **77.4**	**72.9** / **70.8**
• **FlatFormer**+FSD (Ours)	1	39.3	79.4 / 77.1	72.7 / 70.5
○ CenterFormer [260]	2	53.5	78.3 / 76.7	74.3 / 72.8
• **FlatFormer**+FSD (Ours)	2	51.8	**81.4** / **79.9**	**75.2** / **73.8**
○ CenterFormer [260]	4	85.8	78.5 / 77.0	74.7 / 73.2
○ MPPNet [262]	4	–	81.1 / 79.8	75.4 / 74.2
• **FlatFormer**+FSD (Ours)	3	60.6	**82.2** / **80.7**	**76.2** / **74.8**

Table 3.11: Results of two-stage 3D detectors on Waymo Open Dataset (validation set). FlatFormer achieves on-par or even higher accuracy compared with sparse convolutional two-stage detectors. Markers ○ and • refer to SpConv-based models and point cloud transformers, respectively. (†: from FSD paper)

hard are categorized into level 2 (L2) and other objects are defined as level 1 (L1).

Model. Based on FWA, we provide an instantiation of FlatFormer for 3D object detection. We follow the design of PointPillars [242] to first voxelize the point cloud into sparse BEV pillars (with MLPs) at a resolution of 0.32m×0.32m. We then apply eight consecutive FWA blocks with alternating sorting axes (*i.e.*, x or y) and window shifting configurations (*i.e.*, on or off). All FWA blocks have a window shape of 9×9 and a group size of 69. Following SST [154], we do not apply any spatial downsampling, which is beneficial for small objects. Finally, we apply regular BEV encoder and a center-based detection head following CenterPoint [229, 246].

Single-Stage Detectors

Baseline. We compare FlatFormer with state-of-the-art sparse convolutional [229, 246, 255] and transformer-based [154, 232, 280] single-stage 3D detectors. All models apply anchor- or center-based detection heads [61, 229, 246]. We compare models with different numbers of input frames separately.

Latency. We measure the latency on an NVIDIA Quadro RTX A6000 GPU using FP16 precision. We adopt SpConv v2.2.3 [61], the state-of-the-art 3D sparse convolution library,

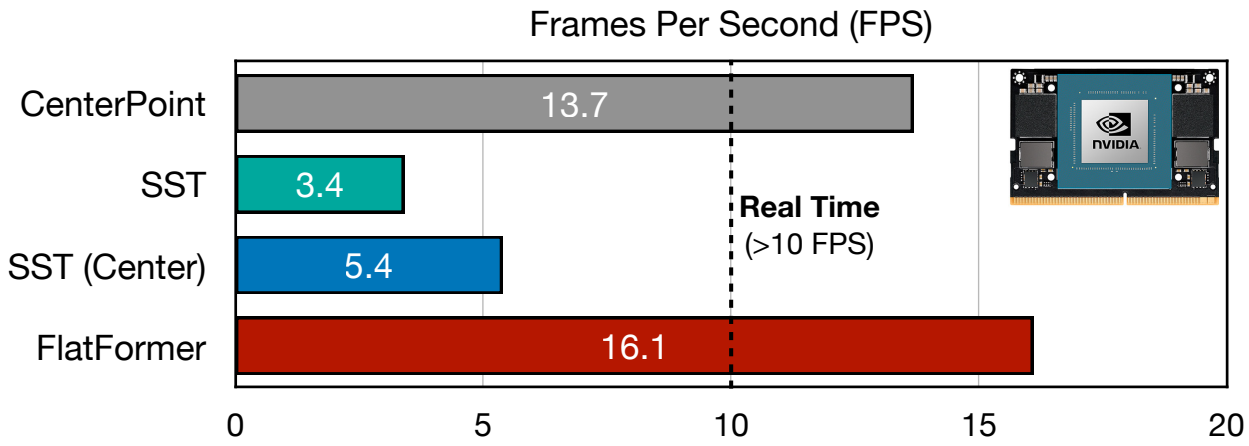


Figure 3.14: Measured latency on NVIDIA Jetson AGX Orin. FlatFormer is the first point cloud transformer that achieves real-time performance on edge GPUs.

to execute the 3D encoder of all sparse convolutional detectors. For transformer-based detectors, we use their official implementation to measure the runtime. All modules after the 3D encoder (*e.g.*, BEV encoder and detection head) are executed with TensorRT 8.4. We execute all the methods on the first 1,000 samples for 50 runs (with 10 warmup runs). We report the average latency (with outliers excluded). We do not include the data loading and post-processing time.

Results. As in Table 3.10, our FlatFormer achieves consistent performance improvements over both sparse convolutional and transformer-based detectors with much better efficiency. For one-frame models, FlatFormer is **4.2**×, **3.3**× and **3.7**× faster than SST, SST-Center and the recent VoxSet [280]. It also compares favorably with strong sparse convolutional baselines: **1.4**× faster than CenterPoint with 1.7 higher L2 mAPH and performs on par with PillarNet [255]. The accuracy advantage magnifies in the two-frame setting. Specifically, FlatFormer is 1.4× faster than CenterPoint with 2.8 L2 mAPH higher accuracy, and outperforms PillarNet by 1.3 L2 mAPH with a similar latency. With three input frames, FlatFormer is **4.6**× and **3.2**× faster than SST and SST-Center, respectively. It also achieves better latency-accuracy tradeoff (1.1× faster and 0.4% higher accuracy) compared with CenterPoint++ [246]. Remarkably, FlatFormer requires 1.7× more MACs than CenterPoint++ while it is still faster. This indicates that our design is more hardware-friendly than the sparse convolutional baselines.

Deployment. We deploy our FlatFormer on an NVIDIA Jetson AGX Orin. This is a resource-constrained edge GPU platform that is widely used in real-world self-driving cars. From Figure 3.14, FlatFormer runs at 16 FPS, which is **1.2**× faster than CenterPoint [229]

and **3**× faster than SST-Center. To the best of our knowledge, FlatFormer is the first point cloud transformer that achieves real-time inference (*i.e.*, >10 FPS, which is the LiDAR sensor frequency) on edge GPUs. We believe that it paves the way for efficient LiDAR-centric perception in real-world applications.

Two-Stage Detectors

Model. To verify the generalizability, we replace the 3D backbone in FSD [261], a state-of-the-art two-stage detector, and compare its results with previous two-stage models. We keep the same grid resolution, window shape and group size as in our single-stage experiments.

Baseline & Latency. We compare our model against state-of-the-art two-stage detectors in Table 3.11. We follow the same latency measurement protocol. For CenterFormer [260], we adapt the official implementation to support SpConv v2.2.3 backend in FP16 precision for a fair comparison.

Results. All existing high-performing two-stage detectors are sparse convolutional, while our FlatFormer is the only transformer-based method that achieves state-of-the-art level accuracy. It also shows better scalability with respect to the number of input frames compared with CenterFormer [260]. Note that we focus on optimizing the latency of *3D backbone*. However, two-stage detectors [260, 261] are usually bottlenecked by *the second stage* in runtime, which is out of our scope. We expect that the latency of FlatFormer could benefit from a more efficient second-stage design.

3.2.6 Analysis

We will present analyses to validate the effectiveness of our design choices. All experiments are based on our single-frame model trained with 20% data.

Flattened Window Attention

In Figure 3.15, we visualize the learned attention weights in our FWA. The color represents the scale of attention weights, where warmer color means larger attention weights. Black points correspond to query points, and gray points are those with weights smaller than a threshold. For vehicles moving straight ahead, turning and parked, query points on the vehicle are always highly attended to nearby points on the same car, while faraway points have very small learned attention weights. Such an observation can

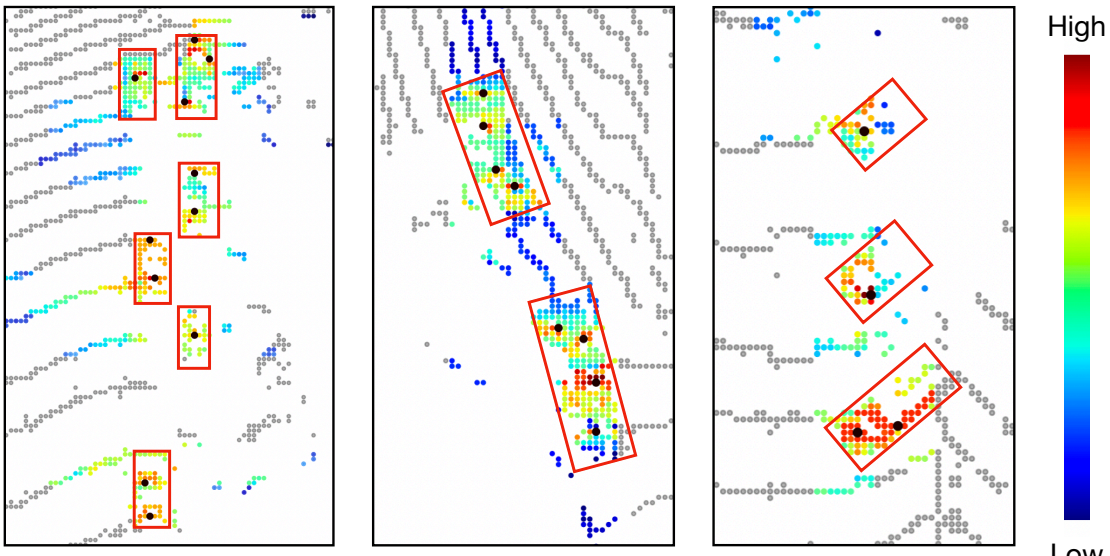


Figure 3.15: Visualization of attention weights in FlatFormer for vehicles that are moving straight ahead, turning and parking. High attention weights corresponds to the detected object.

Sorting Strategy	Mean L2 (mAPH)	Vehicle L2 (mAPH)	Pedestrian L2 (mAPH)	Cyclist L2 (mAPH)
Ours	**61.7**	**63.3**	**57.9**	**63.9**
w/o Quantization	60.4	61.9	57.6	61.7
w/o Axis Alternation	61.1	63.1	57.3	62.9
w/o Window Shift	61.2	62.9	57.8	62.8
Random Order	57.8	58.8	55.1	59.4
SST [154]	60.7	62.3	56.7	63.1

Table 3.12: Window-based sorting in FlatFormer provides even better performance than equally-shaped window partition in SST [154] and outperforms other sorting strategies.

partially explain the effectiveness of FWA: *i.e.*, even if equal-size grouping does not create spatially regular windows, the model can learn to suppress the importance of outlier points in the background and focus on important foreground points within each group.

Ablation Studies on Model Design

Sorting Strategy. We first analyze the effectiveness of our window-based, axis-alternating sorting strategy in Table 3.12. Randomly grouping points together without any spatial sorting will give $\sim$ 4% worse performance compared with FlatFormer. Furthermore, due to spatial discontinuities on the boundary regions, directly sorting the points by xyz

Window Shape	Mean L2 (mAPH)	Vehicle L2 (mAPH)	Pedestrian L2 (mAPH)	Cyclist L2 (mAPH)
6×6	61.1	62.9	57.0	63.4
9×9	**61.7**	**63.3**	**57.9**	**63.9**
13×13	61.3	**63.3**	**57.9**	62.9

Table 3.13: FlatFormer is not sensitive to the choice of window shapes.

Group Size	Mean L2 (mAPH)	Vehicle L2 (mAPH)	Pedestrian L2 (mAPH)	Cyclist L2 (mAPH)
81×50%	60.7	62.7	56.7	62.7
81×85%	**61.7**	**63.3**	**57.9**	**63.9**
81×125%	60.9	63.1	57.0	62.5

Table 3.14: Choosing a group size that is slightly smaller than the window shape (9×9) provides the best accuracy on Waymo.

coordinates or window sorting along a single axis both provide sub-optimal results. We also notice that window shifting brings about 0.5% improvement to the final performance. Interestingly, despite the fact that our sorting strategy does not guarantee the windows to be geometrically regular as in SST [154], FlatFormer still consistently outperforms SST in all three classes.

Window Shape. FlatFormer achieves robust performance under different window shapes. We choose the window shape of 9×9 (2.88m×2.88m, which is the size of a vehicle) in all experiments according to the results in Table 3.13, where we always fix the group size to be 85% of the window shape.

Group Size. We further study the choice of group sizes in Table 3.14. We fix the window shape to be 9×9 according to the results in Table 3.13 and sweep the group size in 50%, 85% and 125% of the window shape. The results show that setting group size to be 85% of the window shape gives the best performance. Intuitively, if the group size is too small, FlatFormer will not be able to have a large enough receptive field (*e.g.*, group size = 1, FWA will degenerate to MLP). When the group size is too large (say, group size = the entire point cloud), there will be a large number of outliers within each group, and FlatFormer will behave like a global PCT, which is not desired.

Input Resolution. From Table 3.15, 0.32m×0.32m input resolution is the sweet spot in the latency-accuracy tradeoff for FlatFormer while further increasing the input size will

Grid Resolution	Mean L2 (mAPH)	Vehicle L2 (mAPH)	Pedestrian L2 (mAPH)	Cyclist L2 (mAPH)
0.36m	60.7	63.0	56.7	62.4
0.32m	**61.7**	**63.3**	**57.9**	63.9
0.28m	**61.7**	63.1	57.8	**64.1**

Table 3.15: Ablation on input resolution in FlatFormer: 0.32m×0.32m is the best design choice that balances efficiency and accuracy.

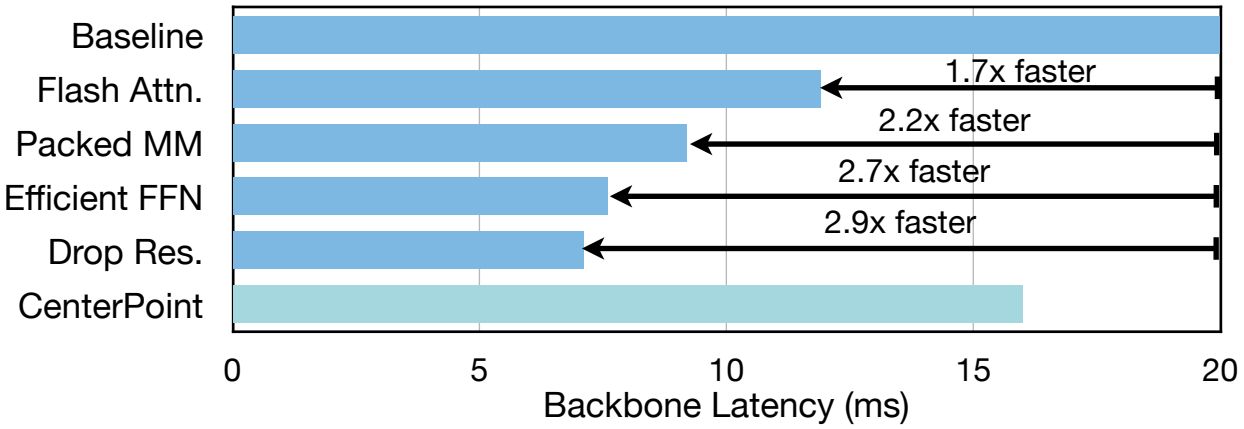


Figure 3.16: Improvement breakdown for system optimizations. We accelerate the backbone latency of FlatFormer by 2.9×, making it **2.3**× faster than CenterPoint.

only hurt the efficiency with no performance improvements.

Breakdown for System Optimizations

In Figure 3.16, we analyze the effectiveness of our system optimizations proposed in Section 3.2.4. An efficient MHSA implementation (FlashAttention) brings 1.7× improvement to our inference latency. Packing the computation for $\mathcal{Q}, \mathcal{K}, \mathcal{V}$ in a single linear kernel results in a 1.3× speedup. Fusing the linear and activation layers (in FFN) brings another 1.2× speedup. Finally, dropping the non-full window improves our inference latency by 1.1×. To sum up, our system optimizations improve the latency of our FlatFormer by **2.9**×, making its backbone **2.3**× **faster** than CenterPoint [229].

CenterPoint is backed by SpConv [61], which is a highly-optimized sparse convolution inference library built upon CUTLASS [285]. Nevertheless, FlatFormer still achieves the best efficiency on NVIDIA GPUs. We partially attribute our efficiency advantage to the equally-sized groups in FlatFormer which not only gives us the best computation regularity but also eliminates the computation overhead. SpConv, on the other hand, implements 3D sparse convolution with a masked implicit GEMM algorithm, which inevitably introduces

computation overhead when points within one thread block do not have exactly the same neighbor patterns. As such, FlatFormer can beat sparse convolutional models on GPUs despite their heavy system optimizations.

3.2.7 Discussion

We introduced FlatFormer to bridge the huge efficiency gap between sparse transformers and sparse convolutional models. It partitions the point cloud with equal-size grouping rather than equal-window grouping, trading spatial proximity for computational regularity. FlatFormer achieves state-of-the-art accuracy on Waymo Open Dataset with 4.6$\times$ speedup over previous sparse transformers. We hope that FlatFormer can inspire future research on designing efficient and accurate sparse transformers.

Part II

Systems

Chapter 4

Optimizing Sparse Primitives

4.1 TorchSparse

Deep learning on point clouds has received increased attention thanks to its wide applications in AR/VR and autonomous driving. These applications require low latency and high accuracy to provide real-time user experience and ensure user safety. Unlike conventional dense workloads, the sparse and irregular nature of point clouds poses severe challenges to running sparse CNNs efficiently on the general-purpose hardware. Furthermore, existing sparse acceleration techniques for 2D images do not translate to 3D point clouds. In this paper, we introduce TorchSparse, a high-performance point cloud inference engine that accelerates the sparse convolution computation on GPUs. TorchSparse directly optimizes the two bottlenecks of sparse convolution: **irregular computation** and **data movement**. It applies *adaptive matrix multiplication grouping* to trade computation for better regularity, achieving 1.4-1.5× speedup for matrix multiplication. It also optimizes the data movement by adopting *vectorized*, *quantized* and *fused locality-aware memory access*, reducing the memory movement cost by 2.7×. Evaluated on seven representative models across three benchmark datasets, TorchSparse achieves **1.6**× and **1.5**× measured end-to-end speedup over the state-of-the-art MinkowskiEngine and SpConv, respectively.

4.1.1 Introduction

3D point cloud becomes increasingly accessible over the past few years thanks to the widely available 3D sensors, such as LiDAR scanners (on the self-driving vehicles and, more recently, even the mobile phones) and depth cameras (on the AR/VR headsets). Compared with 2D RGB images, 3D point clouds provide much more accurate spatial/depth information and are usually more robust to different lighting conditions. Therefore, 3D

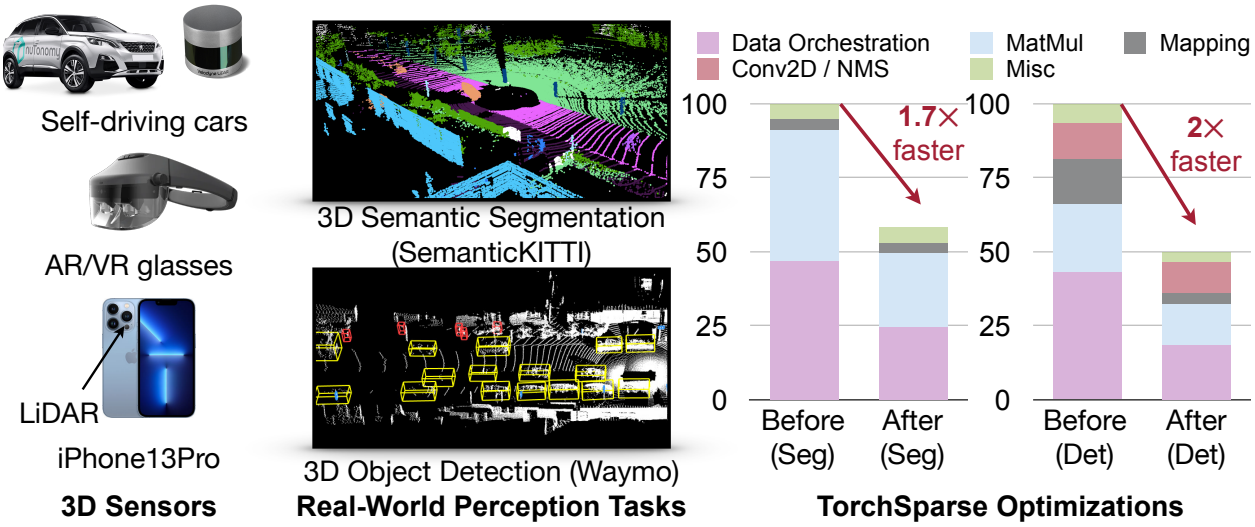


Figure 4.1: Widely available 3D sensors (left) have enabled more and more real-world AI applications to perceive the world using 3D point clouds (middle). However, processing 3D point cloud requires sparse computation that is not favored by general-purpose hardware. TorchSparse reduces the irregular computation and optimizes the data movement, achieving 1.7× to 2× measured speedup (right).

point cloud processing becomes the key component of many real-world AI applications: *e.g.*, to understand the indoor scene layout for AR/VR, and to parse the driveable regions for autonomous driving.

A 3D point cloud is an *unordered* set of 3D points. Unlike 2D image pixels, this data representation is highly sparse and irregular. Researchers have explored to rasterize the 3D data into dense volumetric representation [167] or directly process it in the point cloud representation [151, 152]. However, both of them are not scalable to large indoor/outdoor scenes [22]. Alternatively, researchers have also investigated to flatten 3D point clouds into dense 2D representations using spherical projection and bird's-eye view (BEV) projection. However, their accuracy is much lower due to the physical dimension distortion and height information loss.

Recently, state-of-the-art 3D point cloud neural networks tend to rely largely or fully on sparse convolutions [120], making it an important workload for machine learning system: all top 5 segmentation submissions on SemanticKITTI [214] are based on SparseConv, 9 of top 10 submissions on nuScenes [73] and top 2 winning solutions on Waymo [240] have exploited SparseConv-based detectors [229, 247]. Given the wide applicability and dominating performance of SparseConv-based point cloud neural networks, it is crucial to provide efficient system support for sparse convolution on the general-purpose hardware.

Unlike conventional dense computation, sparse convolution is not supported by ex-

isting inference libraries (such as TensorRT and TVM), which is why most industrial solutions still prefer 2D projection-based models despite their lower accuracy. It is urgent to better support the sparse workload, which was not favored by modern high-parallelism hardware. On the one hand, the sparse nature of point clouds leads to irregular computation workloads: *i.e.*, different kernel offsets might correspond to drastically different numbers of matched input/output pairs. Hence, existing sparse inference engines [61, 130] usually execute the matrix multiplication for each kernel offset separately, which cannot fully utilize the parallelism of modern GPUs. On the other hand, neighboring points do not lie contiguously in the sparse point cloud representation. Explicitly gathering input features and scattering output results can be very expensive, taking up to 50% of the total runtime. Due to the irregular computation workload and expensive data movement cost, SparseConv-based neural networks can hardly be run in real time: the latest sparse convolution library can only run MinkowskiNet at 8FPS on an NVIDIA GTX 1080Ti GPU, let alone other low-power edge devices.

In this paper, we introduce TorchSparse, a high-performance inference engine tailored for sparse point cloud computation. TorchSparse is optimized based upon two principles: (1) improving the computation regularity and (2) reducing the memory footprint. First, we propose the adaptive matrix multiplication grouping to batch the computation workloads from different kernel offsets together, trading #FLOPs for regularity. Then, we adopt quantization and vectorized memory transactions to reduce memory movement. Finally, we gather and scatter features in locality-aware memory access order to maximize the data reduce. Evaluated on seven models across three datasets, TorchSparse achieves **1.6**× and **1.5**× speedup over state-of-the-art MinkowskiEngine and SpConv, paving the way for deploying 3D point cloud neural networks in real-world applications.

4.1.2 Related Work

Deep Learning on Point Clouds. Early methods [162, 165, 167] first convert point clouds to the dense volumetric representation and apply dense CNNs to extract features. Another line of research [146, 151–153, 215, 223] directly performs convolution on the k-nearest neighbor or spherical nearest neighbor of each point. Both streams of methods struggle to scale up to large scenes due to large or irregular memory footprint [22, 23]. Recent state-of-the-art deep learning methods on point cloud segmentation / detection [53, 120, 130, 229, 238, 239] are usually based on sparse convolution, which is empirically proven to be able to scale up to large scenes and is the target for acceleration in this paper.

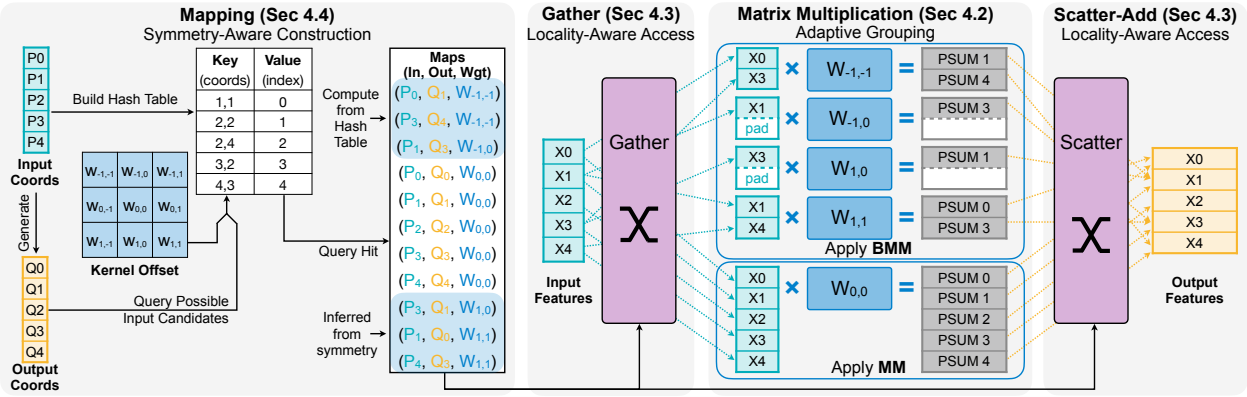


Figure 4.2: TorchSparse aims at accelerating *Sparse Convolution*, which consists of four stages: mapping, gather, matmul and scatter-accumulate. Our optimization follows two principles: ❶ improve the regularity of sparse workload ❷ reduce the memory footprint. To achieve that, TorchSparse exploits adaptively batched matmul (Principle ❶, Section 4.1.5); quantized, vectorized, locality-aware scatter/gather (Principle ❷, Section 4.1.5); and mapping kernel fusion (Principle ❷, Section 4.1.5).

Point Cloud Inference Engines. Researchers have extensively developed efficient inference engines for sparse convolution inference. SpConv [61] proposes grid-based map search and the gather-matmul-scatter dataflow. SparseConvNet [120] proposes hashmap-based map search and is later significantly improved (in latency) by MinkowskiEngine, which also introduces a new *fetch-on-demand* dataflow that excels at small workloads and allows generalized sparse convolution on >3D point clouds and on arbitrary coordinates.

4.1.3 Background

The point cloud can be formulated as an unordered set of points paired with features: $\{(\boldsymbol{p}_j, \boldsymbol{x}_j)\}$, where $\boldsymbol{x}_j \in \mathbb{R}^C$ is a C-dimensional feature vector for point $\boldsymbol{p}_j \in \mathbb{Z}^D$ in a D-dimensional space. For a convolution of kernel size K, let $\boldsymbol{W} \in \mathbb{R}^{K^D \times C^{\text{in}} \times C^{\text{out}}}$ be its weights and $\boldsymbol{\Delta}^D(K)$ be its kernel offsets (*e.g.*, $\boldsymbol{\Delta}^2(5) = \{-2,-1,0,1,2\}^2$ and $\boldsymbol{\Delta}^3(3) = \{-1,0,1\}^3$). The weights $\boldsymbol{W}$ can be broken down into K^D matrices of shape $C^{\text{in}} \times C^{\text{out}}$, denoted as $\boldsymbol{W}_{\boldsymbol{\delta}}$ for $\boldsymbol{\delta} \in \boldsymbol{\Delta}^D(K)$. With these notations, the convolution with stride s can be represented as

$$\boldsymbol{x}_k^{\text{out}} = \sum_{\boldsymbol{\delta} \in \boldsymbol{\Delta}^D(K)} \sum_j 1[\boldsymbol{p}_j = s\boldsymbol{q}_k + \boldsymbol{\delta}] \, (\boldsymbol{x}_j^{\text{in}} \cdot \boldsymbol{W}_{\boldsymbol{\delta}}), \tag{4.1}$$

where $\boldsymbol{p}_j \in \boldsymbol{P}^{\text{in}}$, $\boldsymbol{q}_k \in \boldsymbol{P}^{\text{out}}$, and $1[\cdot]$ is the binary indicator.

For dense convolution (Figure 4.3a), each nonzero input is multiplied with all nonzero weights, leading to rapidly growing nonzeros ($\boldsymbol{P}^{\text{in}} \subset \boldsymbol{P}^{\text{out}}$). On the other hand, the com-

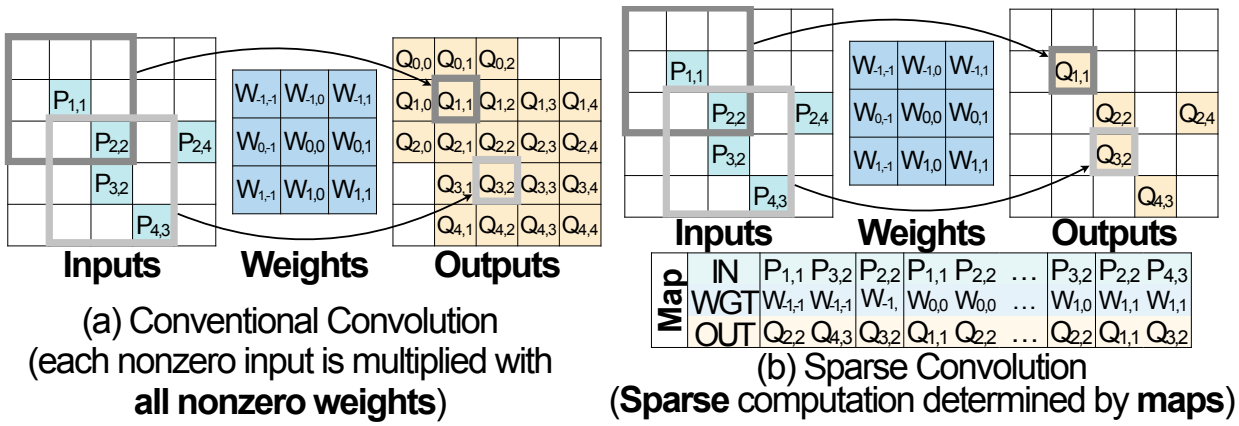


Figure 4.3: Sparse convolution (b) does *not* multiply each nonzero input with all nonzero weights as conventional convolution (a) does.

putation of sparse convolution (Figure 4.3b) is determined by ***maps*** $\mathcal{M} = \{(\boldsymbol{p}_j, \boldsymbol{q}_k, \boldsymbol{W}_\delta)\}$ in Equation 4.1 (also written as $\{(\boldsymbol{p}_j, \boldsymbol{q}_k, \boldsymbol{W}_n)\}$, where n is the weight index) and keeps the sparsity pattern unchanged ($\boldsymbol{P}^{\text{in}} = \boldsymbol{P}^{\text{out}}$). It iterates over all maps and performs $\boldsymbol{x}_k^{\text{out}} = \boldsymbol{x}_k^{\text{out}} + \boldsymbol{x}_j^{\text{in}} \cdot \boldsymbol{W}_\delta$.

Mapping Operations

Mapping is a step to construct the input-output *maps* $\mathcal{M} = \{(\boldsymbol{p}_j, \boldsymbol{q}_k, \boldsymbol{W}_\delta)\}$ for sparse convolution. Here, j is the index of input point $\boldsymbol{p}$ in $\boldsymbol{P}^{\text{in}}$, k is the index of output point $\boldsymbol{q}$ in $\boldsymbol{P}^{\text{out}}$, and $\boldsymbol{W}_\delta$ is the weight matrix for kernel offset $\boldsymbol{\delta}$. Generating maps typically requires two steps: calculating the output coordinates $\boldsymbol{P}^{\text{out}}$, and searching maps $\mathcal{M}$. These operations only take coordinates as input.

Output Coordinates Calculation. When the convolution stride is 1, the output coordinates are exactly the same as the input coordinates, *i.e.*, $\boldsymbol{P}^{\text{out}} = \boldsymbol{P}^{\text{in}}$. When the convolution stride is larger than 1, the nonzero input coordinates will first be dilated for each kernel offset (*i.e.*, $\boldsymbol{p} - \boldsymbol{\delta}$). After that, only these points on the strided grids within boundaries will become outputs $\boldsymbol{q}$, where $s \cdot \boldsymbol{q} = \boldsymbol{p} - \boldsymbol{\delta}$. Take the input coordinate $(3, 5)$ as an example (with stride of 2). For offset $\boldsymbol{\delta} = (1, 1)$, the output coordinate will be $((3, 5) - (1, 1))\ /2 = (1, 2)$, while for offset $\boldsymbol{\delta} = (0, 0)$, there is no valid output coordinate since $((3, 5) - (0, 0))$ is not a multiple of stride $s = 2$.

Map Search. As in Algorithm 1, map search requires iterating over all possible input coordinates for each output coordinate. A map is generated only when the input is nonzero.

Algorithm 1 Map Search

Input: input coordinates $\boldsymbol{P}^{\text{in}}$, output coordinates $\boldsymbol{P}^{\text{out}}$
kernel size K, stride s
Output: maps $\mathcal{M}$

```
N ← Δ^D(K).size
M ← {∅} × N
for k, q_k in enumerate(P^out) do
  # Traverse the neighbors of an output point.
  for n, δ in enumerate(Δ^D(K)) do
    # Calculate input coordinates.
    r ← s · q_k + δ
    # Add new map if input exists.
    if P^in.contain(r) then
      j ← P^in.getIndex(r)
      M[W_δ] ← M[W_δ] ∪ {(p_j, q_k, W_δ)}
    end if
  end for
end for
```

To efficiently examine whether the possible input $\boldsymbol{q}_j + \boldsymbol{\delta}$ is nonzero, a common implementation is to record the coordinates of nonzero inputs with a hash table. The key-value pairs are (key=input coordinates, value=input index), *i.e.*, $(\texttt{key} = \boldsymbol{p}_j, \texttt{value} = j)$. The hash function can simply be flattening the coordinate of each dimension into an integer.

Data Orchestration and Matrix Multiplication

After maps are generated, sparse convolution will multiply the input feature vector $\boldsymbol{x}_j^{\text{in}}$ with corresponding weight matrix $\boldsymbol{W}_{\boldsymbol{\delta}}$ and accumulate to the corresponding output feature vector $\boldsymbol{x}_k^{\text{out}}$, following the map $\{(\boldsymbol{p}_j, \boldsymbol{q}_k, \boldsymbol{W}_{\boldsymbol{\delta}})\}$.

The utilization of matrix-vector multiplication is rather low on GPU. Therefore, most existing implementations follow the gather-matmul-scatter computation flow in Algorithm 2. First, all input feature vectors associated with the same weight matrix are gathered and concatenated into a contiguous matrix. Then, matrix-matrix multiplication between feature matrix and weight matrix is conducted to obtain the partial sums. Finally, these partial sums are scattered and accumulated to the corresponding output feature vectors.

Difference from Other Tasks

***vs.* conventional convolution with sparsity.** The sparsity in conventional convolution comes from the ReLU activation function or weight pruning. Since there is no hard constraint on the output sparsity pattern, each nonzero input is multiplied with every

Algorithm 2 Gather-MatMul-Scatter

Input: input features $\boldsymbol{X}^{\text{in}}$, weights $\boldsymbol{W}$, maps $\mathcal{M}$
Output: output features $\boldsymbol{X}^{\text{out}}$

```
X^out ← 0
# Separately perform gather-matmul-scatter for each weight.
for δ in Δ^D(K) do
    F ← ∅
    # Gather features for w_n.
    for m, (p_j, q_k, W_δ) in enumerate(M[W_δ]) do
        F[m] ← X^in[j]
    end for
    # Matrix-matrix multiplication.
    F ← F · W_δ
    # Scatter partial sums to X^out.
    for m, (p_j, q_k, W_δ) in enumerate(M[W_δ]) do
        X^out[q_k] ← X^out[q_k] + F[m]
    end for
end for
```

nonzero weight, so the nonzeros will dilate during the inference, *i.e.*, $\boldsymbol{P}^{\text{in}} \subset \boldsymbol{P}^{\text{out}}$ (see Figure 4.3a). The existing sparse computation libraries leverage such computation pattern by travelling all nonzero inputs with all nonzero weights to accelerate the conventional convolution. On the contrary, sparse convolution requires $\boldsymbol{P}^{\text{in}} = \boldsymbol{P}^{\text{out}}$, and thus the relationship among inputs, weights and outputs requires to be explicitly searched with mapping operations, which makes it a hassle for previous sparse libraries.

***vs.* graph convolution.** In graph convolution, the relationship between inputs and outputs are provided in the adjacency matrix which stays constant across layers. Contrarily, sparse convolution has to search maps for **every** downsampling block. Furthermore, graph convolution shares the same weight matrix for different neighbors, *i.e.* all $\boldsymbol{W}_{\boldsymbol{\delta}}$ are the same. Hence, graph convolution only needs either one gather or one scatter of features: 1) first gather input features associated with the same output vertex, and then multiply them with shared weights and reduce to the output feature vector; or 2) first multiply all input features with shared weights, and then scatter-accumulate the partial sums to the corresponding output feature vector. However, sparse convolution uses different weight matrices for different kernel offset $\boldsymbol{\delta}$ and thus needs both gather and scatter during the computation. Consequently, existing SpMV/SpDMM systems for graph convolution acceleration [286, 287] are not applicable to sparse convolution.

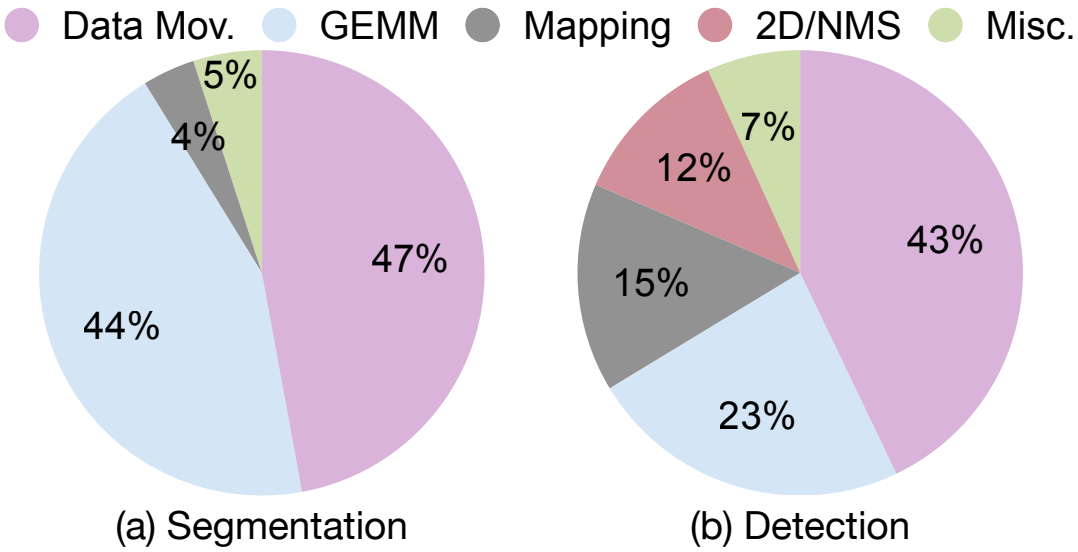


Figure 4.4: Data movement and GEMM constitute a significant proportion of the runtime of sparse CNNs.

4.1.4 Analysis

We systematically profile the runtime of different components in two representative sparse CNNs: one for segmentation (Figure 4.4a) and one for detection (Figure 4.4b). Based on observations in Figure 4.4, we summarize two principles for sparse convolution optimization which lays the foundation for our system design in Section 4.1.5.

Principle I. Improve Regularity in Computation Matrix multiplication is the core computation in sparse convolution and takes up a large proportion of total execution time (20%-50%). Algorithm 2 decouples the matrix multiplication computation from data movement so that we can use well-optimized libraries (such as cuDNN) to calculate $\boldsymbol{X}^{\text{out}} \leftarrow \boldsymbol{X}^{\text{in}} \cdot \boldsymbol{W}_{\delta}$. However, the computation workloads are very *non-uniform* due to the irregular nature of point clouds (detailed in Figure 4.12, where map sizes for different weights can differ by an order of magnitude, and most map sizes are *small*). As a result, the matrix multiplication in MinkUNet (0.5× width) runs at 8.1 TFLOP/s on RTX 2080 Ti with FP16 quantization, achieving only 30% device utilization. Therefore, *improving the regularity* of matrix multiplication will potentially be helpful: we boost the utilization to 44.2% after optimization (detailed in Table 4.2).

Principle II. Reduce Memory Footprint Data movement is the largest bottleneck in sparse CNNs, which takes up 40%-50% of total runtime on average. This is because

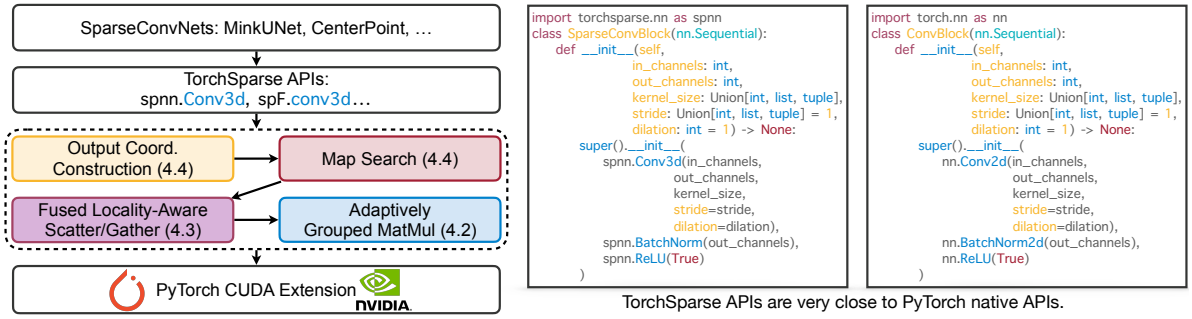


Figure 4.5: System overview for TorchSparse: our TorchSparse provides handy Python APIs similar to PyTorch and applies low-level optimizations to data movement, matrix multiplication and mapping operations in sparse convolution.

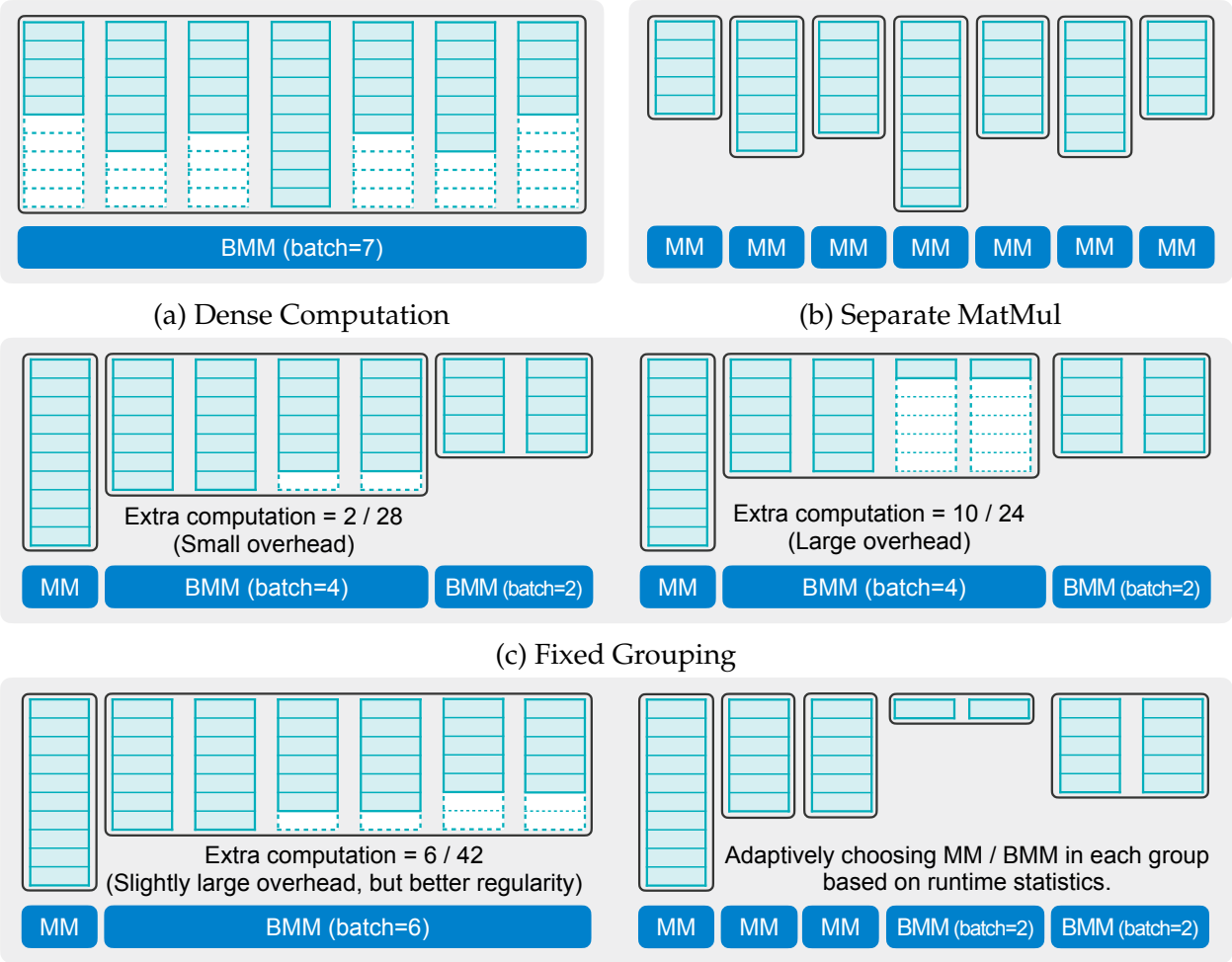
scatter-gather operations are bottlenecked by GPU memory bandwidth (*limited*) rather than computation resources (*abundant*). Worse still, the dataflow in Algorithm 2 completely separates scatter-gather operations for different kernel offsets. This further ruins the possibility of any reuse in the data movement, which will be detailed in Figure 4.9. It is also noteworthy that the large mapping latency in the CenterPoint detector (Figure 4.4b) also stems from memory overhead: hashmap construction and output coordinate calculation both require multiple DRAM accesses. Thus, *reducing memory footprint* is at the heart of data movement and mapping optimization.

4.1.5 System Design and Optimization

This subsection unfolds our TorchSparse as follows: Section 4.1.5 provides an overview and API design for TorchSparse, Section 4.1.5 introduces improvements on matrix multiplication operations, Section 4.1.5 elaborates the optimizations for data movement operations (scatter/gather), and Section 4.1.5 analyzes the opportunities to speed up mapping operations.

System Overview

Figure 4.5 provides an overview of our TorchSparse. At the top level, users define their sparse CNNs using TorchSparse APIs, which have minimal differences with native PyTorch APIs. Also, TorchSparse does not require users to add additional fields such as `indice_key` and `spatial_shape` in SpConv [61], and `coordinate_manager` in MinkowskiEngine [130] when defining modules and tensors. TorchSparse converts the high-level modules to primitive operations: *e.g.*, `Conv3d` is decomposed to output construction, mapping operations and gather-matmul-scatter. For each part, Python APIs interact with backend



(a) Dense Computation

(b) Separate MatMul

(c) Fixed Grouping

(d) Adaptive Grouping

Figure 4.6: Different matrix multiplication grouping strategies: (a) dense computation suffers from large FLOPs overhead; (b) separate matrix multiplication suffers from low device utilization and excessive kernel calls; (c) fixed grouping trades FLOPs for regularity; (d) adaptive grouping searches for the best balance point.

CUDA implementations via `pybind`. Note that TorchSparse also provides support for CPU inference and multi-GPU training, but this paper will focus on the GPU inference.

Matrix Multiplication Optimization

Matrix multiplication is the core computation in sparse convolution. Due to the irregularity of point clouds, existing implementations rely on cuDNN to perform many small matrix multiplications on different weights (Figure 4.6b), which usually do not saturate the utilization of GPUs. In order to increase the utilization, we propose to trade computation for regularity (Principle I) by grouping matrix multiplication for different weights

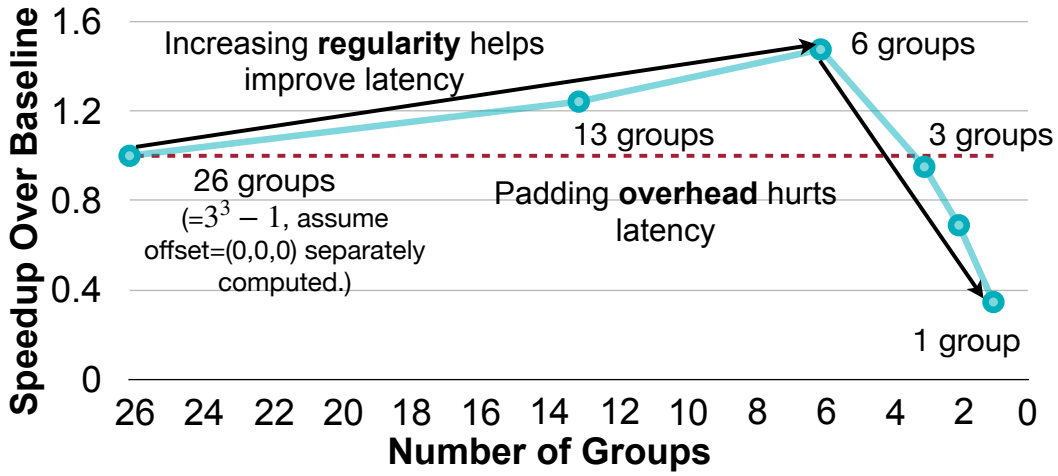


Figure 4.7: Trading FLOPs for computation regularity via batched matrix multiplication brings 1.5× speedup.

together. We find it helpful to introduce redundant computation but group more computation in a single kernel. In Figure 4.12, we collect the real workload for MinkUNet [130] on SemanticKITTI [214] and analyze the efficiency of matrix multiplication in the first sparse convolution layer with respect to the group size. It turns out that batched matrix multiplication can be significantly faster than sequentially performing the computation along the batch dimension, thanks to the better regularity. This motivates us to explore the opportunity of *grouping* in the matrix multiplication computation.

Symmetric Grouping. With sparse workloads, the map sizes for different weights within one sparse convolution layer are usually *different*. Fortunately, for sparse convolutions with odd kernel size and stride of 1, the maps corresponding to kernel offset (a, b, c) will always have the same size as the maps corresponding to the *symmetric* kernel offset $(-a, -b, -c)$. For a map entry $(\boldsymbol{p}_j, \boldsymbol{q}_k, \boldsymbol{W}_{a,b,c})$, we have $\boldsymbol{q}_k = \boldsymbol{p}_j + (a, b, c)$. Then, $\boldsymbol{p}_j = \boldsymbol{q}_k + (-a, -b, -c)$, which implies that $(\boldsymbol{q}_k, \boldsymbol{p}_j, \boldsymbol{W}_{-a,-b,-c})$ is also a valid map entry. As such, we can establish an one-to-one correspondence between maps for weights $\pm(a, b, c)$. Therefore, we are able to group the workload for symmetric kernel offsets together and naturally have a batch size of 2. Note that the workload corresponding to the kernel offset $(0, 0, 0)$ is processed separately since it does not require any explicit data movement. From Figure 4.7, the symmetric grouping (13 groups) can already be up to 1.2× faster than the separate matrix multiplication.

Fixed Grouping. Though symmetric grouping works well for sparse convolutions with the stride of 1, it falls short in generalizing to downsampling layers. Also, it cannot push

the batch size to > 2, which means that we still have a large gap towards the best GPU utilization in Figure 4.7. Nevertheless, we find that clear pattern exists in the map size statistics (Figure 4.12): for submanifold layers, the maps corresponding to $\boldsymbol{W}_0$ to $\boldsymbol{W}_3$ tend to have similar sizes and the rest of the weights other than the middle one have similar sizes; for downsampling layers, the maps for all offsets have similar sizes. Consequently, we can batch the computation into three groups accordingly. Within each group, we pad all features to the maximum size (Figure 4.6c). Fixed grouping generally works well when all features within the same group have similar sizes (Figure 4.6c left), and this usually happens in downsampling layers. For submanifold layers (Figure 4.6c right), the padding overhead can sometimes be large despite the better regularity, resulting in wasted computation.

Adaptive Grouping. The major drawback of fixed grouping is that it does *not* adapt to individual samples. This can be problematic since workload size distributions can vary greatly across different datasets (Figure 4.12). It is also very labor-intensive to design different grouping strategies for different *layers*, different *networks* on a diverse set of *datasets* and *hardware*. To this end, we design an adaptive grouping algorithm (Figure 4.6d) that *automatically* determines the *input-adaptive* grouping strategy for a given layer on arbitrary workload.

The adaptive grouping algorithm builds upon two auto-tuned parameters ϵ and S, where ϵ indicates our tolerance of redundant computation, and S is the workload threshold. Given ϵ, we scan over sizes of all maps in the current workload for $\boldsymbol{W}_0$ to $\boldsymbol{W}_{K-1}$ (where K is the kernel volume) and dynamically maintain two pointers indicating the start and end of the current group. We initiate a new group whenever the redundant computation ratio ($1 - \frac{\text{Theoretical FLOPs}}{\text{Actual FLOPs}}$) exceeds ϵ. Then, given S, we inspect the maximum workload size within each group. Each group performs `bmm` if the workload size is smaller than S and performs `mm` otherwise. This is because `bmm` can improve device utilization for small workloads but has little benefit for large workloads. We refer the readers to Appendix B for more details of this algorithm. Note that even if ϵ and S are fixed, the generated strategy itself is still *input-adaptive*. Since different input point clouds have different map sizes, even the same ϵ can potentially generate different group partition strategies for different samples. The (ϵ, S) parameter space is simple but diverse enough to cover dense computation ($\epsilon = 1$; $S = +\infty$), separate computation ($S = 0$) as well as symmetric grouping ($\epsilon = 0; S = +\infty$) as its special cases.

For a given sparse CNN, we determine (ϵ, S) for each layer on the target dataset and hardware platform via exhaustive grid search on a small subset (usually 100 samples) of

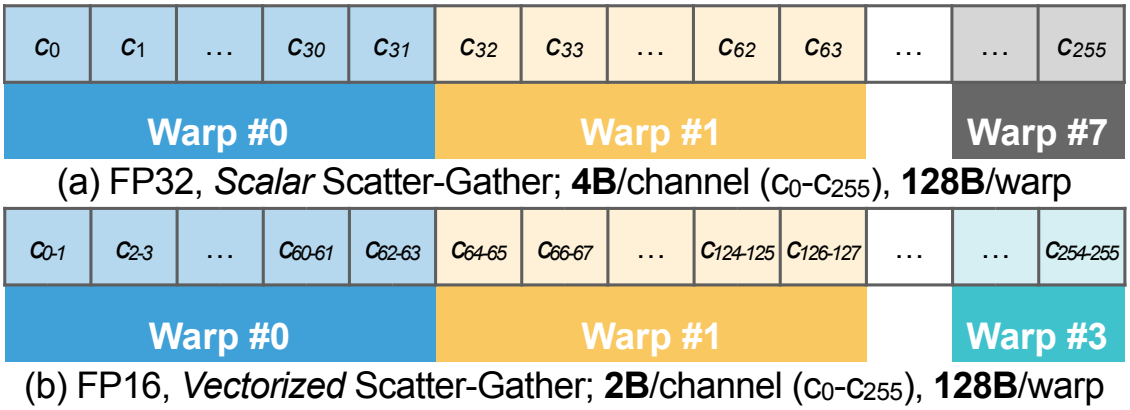


Figure 4.8: TorchSparse applies *vectorized* and *quantized* scatter-gather to greatly reduce the data movement latency.

the training set. We formalize this process in Appendix B. The search is inference-only. It explores a space of around 1,000 configurations and requires less than 10 minutes of search time on a desktop GPU. The strategy derived on the small subset can be directly applied and does not require any parameter optimization during the inference time.

Data Movement Optimization

From Section 4.1.4, data movement usually takes up 40-50% of the total runtime. Thus, optimizing data movement will be of high priority as well (Principle II). Intuitively, it is most effective to reduce data movement cost by reducing the total amount of DRAM access and exploiting the data reuse.

Quantized and Vectorized Memory Access. FP16 quantization brings 2× theoretical DRAM access saving compared with the FP32 baseline. However, as in Figure 4.8, this reduction cannot be translated into real speedup without vectorized scatter/gather.

NVIDIA GPUs group memory access requests into *transactions*, whose largest size is 128 bytes. Considering the typical memory access pattern in scatter/gather, where a warp (32 threads) issues contiguous FP32 (4 bytes) memory access instructions simultaneously, the 128-byte transaction is fully utilized. However, when each thread in the warp issues an FP16 memory request, the memory transaction has only 64/128=50% utilization, and the total number of memory transactions are essentially unchanged. As a result, we observe far smaller speedup (1.3×) compared to the theoretical value (2×) on scatter/gather if *scalar* scatter/gather (Figure 4.8a) is performed.

Contrarily, *vectorized* scatter-gather Figure 4.8b doubles the workload of each thread, making the total work of each warp still 128 bytes, equivalent to a full FP32 memory

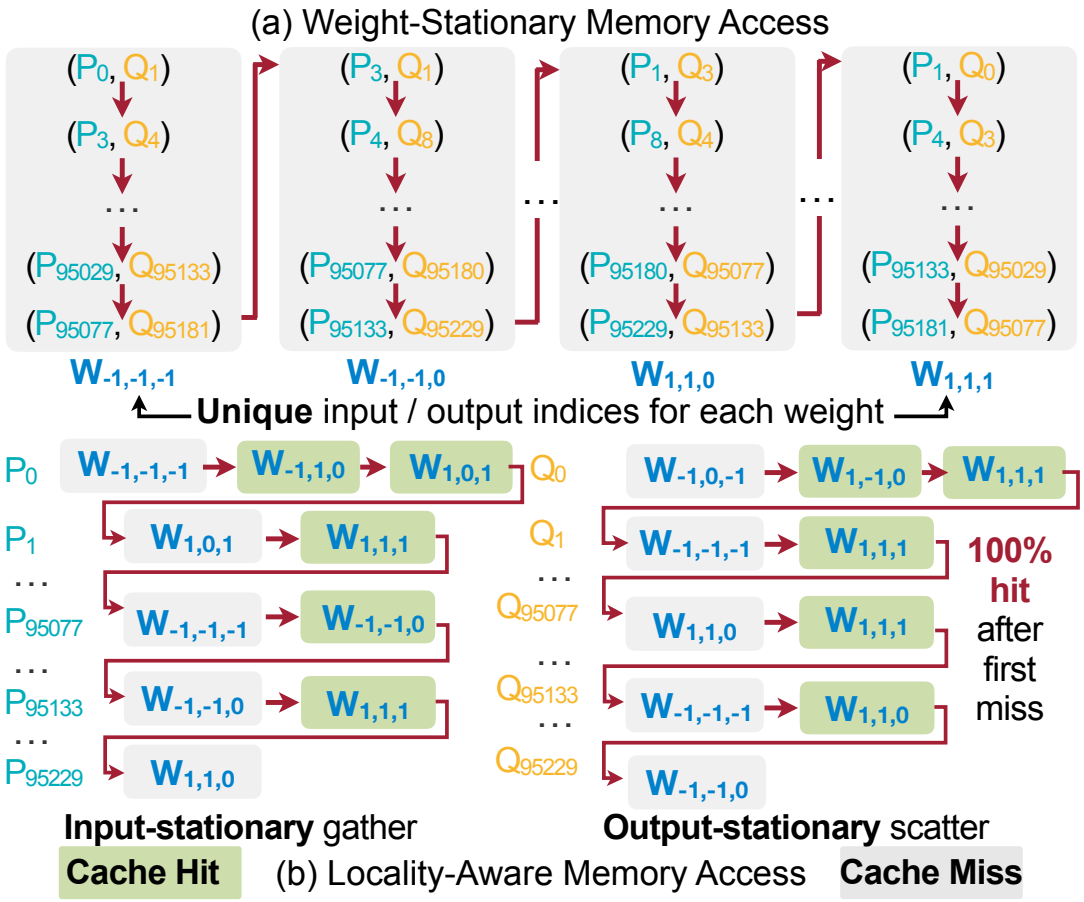


Figure 4.9: TorchSparse proposes cache-friendly *locality-aware* and memory access pattern. In contrary, baseline implementation (a) cannot exploit cache reuse due to uniqueness in input/output indices for each weight.

transaction. Meanwhile, the total number of memory transactions is *halved* while the work for each memory transaction is *unchanged*, and we observe **1.9**× speedup over FP32 data movement on various GPU platforms. This closely aligns with the theoretical reduction in DRAM access.

Further quantizing the features to INT8 offers diminishing return, as the multi-way reduction in the scatter operation requires more than 8-bit for the final result. In this case, all scatter operations are still in 16 bits since CUDA requires aligned memory access. Thus, scattering (which takes 60% of the data movement time) cannot not accelerated with the INT8 quantization, leading to limited overall speedup.

Fused and Locality-Aware Memory Access. Despite the limitation of aggressive feature quantization, it is still possible to achieve faster scatter/gather by exploiting locality. Intuitively, for a sparse convolution layer, the total amount of gather read and scatter write

is $N_1 = |\mathcal{M}|(C_{\text{in}} + C_{\text{out}})$, where $\mathcal{M}$ is the map for this layer (defined in Section 4.1.3), and C_{in} and C_{out} correspond to input and output channel numbers. However, the total feature size of this layer is $N_2 = N_{\text{in}}C_{\text{in}} + N_{\text{out}}C_{\text{out}}$. Empirically, the feature of each point is repetitively accessed for at least 4 times ($N_1 \geq 4N_2$). Based on this, we can ideally have **1.6**$\times$ more DRAM access saving for scatter/gather (the amount of gather write and scatter read is also N_1 and cannot be saved).

As shown in Algorithm 2 and Figure 4.9a, the current implementation completely separates gather/scatter for different weights. When we perform *gather* operation for $\boldsymbol{W}_{k+1}$, the GPU cache is filled with *scatter* buffer features for $\boldsymbol{W}_k$ as long as the GPU cache size is much smaller than N_1 (typically $>$ 40MB, much larger than the 5.5MB L2 cache of NVIDIA RTX 2080 Ti). Intuitively, for gather operation on $\boldsymbol{W}_{k+1}$, we hope that the cache is filled with gather buffer features from $\boldsymbol{W}_k$. This suggests us to first fuse *all* gather operations before matrix multiplication, and fuse *all* scatter operations afterwards. As such, the GPU cache will always hold data from the same type of buffer.

Moreover, the memory access order matters. In the weight-stationary order (Figure 4.9a), all map entries for weight $\boldsymbol{W}_k$ are *unique*, so there is *no* chance of feature reuse, and each gather/scatter leads to a cache miss. As in Figure 4.9b, we instead take a *locality-aware* memory access order. We gather the input features in the *input-stationary* order and scatter the partial sums in the *output-stationary* order.

Without loss of generality, we will focus on the implementation of input-stationary gather. We first maintain a neighbor set $\mathcal{N}_j$ for each input point $\boldsymbol{p}_j$: *i.e.*, for the i$^{\text{th}}$ map entry $(\boldsymbol{p}_j, \boldsymbol{q}_k, \boldsymbol{W}_n)$, we insert $(\boldsymbol{W}_n, i)$ into $\mathcal{N}_j$. Then, we iterate over every input point $\boldsymbol{p}_j$, fetch its feature vector $\boldsymbol{X}_j^{\text{in}}$ into the register, and write it to the corresponding DRAM location $\sum_{k=0}^{n-1} |\mathcal{M}[\boldsymbol{W}_k]| + i$ for each $(\boldsymbol{W}_n, i) \in \mathcal{N}_j$. Here, $\mathcal{M}[\boldsymbol{W}_k]$ is the map for weight $\boldsymbol{W}_k$. Note that each $\boldsymbol{X}_j^{\text{in}}$ is read from DRAM only once and held in the register. Hence, this algorithm achieves the optimal reuse for gather. Similar technique can be applied to scatter, where we read neighbors' partial sums for each output point from DRAM, perform reduction in the register, and write the result back only once. This optimization alone leads to **1.3-1.4**$\times$ speedup in data movement on real-world point cloud datasets.

Mapping Optimization

From Figure 4.4, mapping operations in our baseline implementation take up a significant amount of time (15%) in detectors on the Waymo [240] dataset. It is important to reduce the mapping overhead in sparse CNNs.

We first choose the map search strategy for each layer from [`grid`, `hashmap`] in a sim-

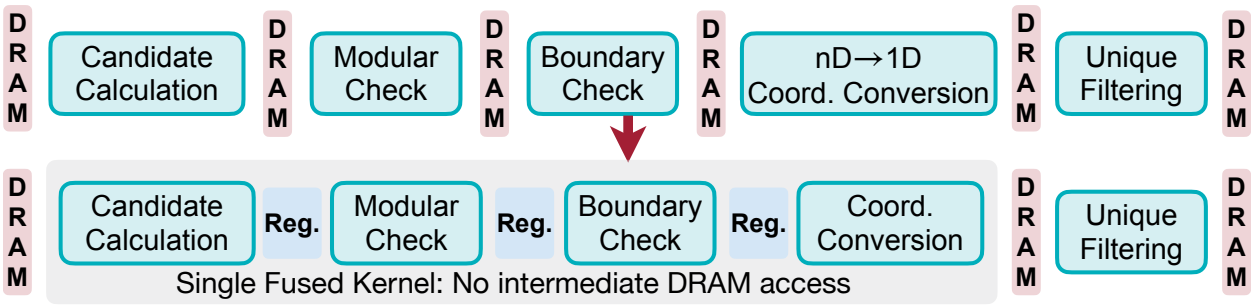


Figure 4.10: TorchSparse reduces mapping DRAM access and improves mapping latency via kernel fusion.

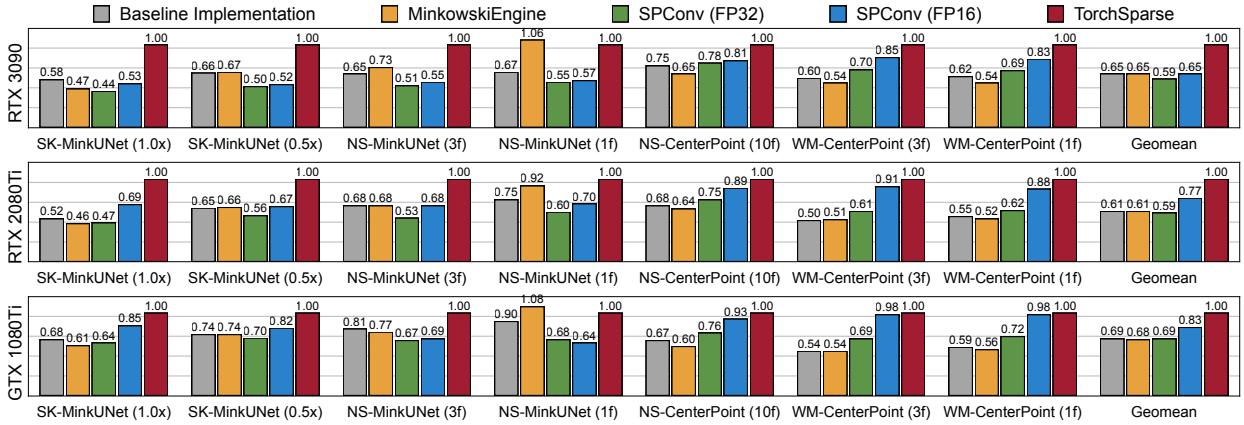
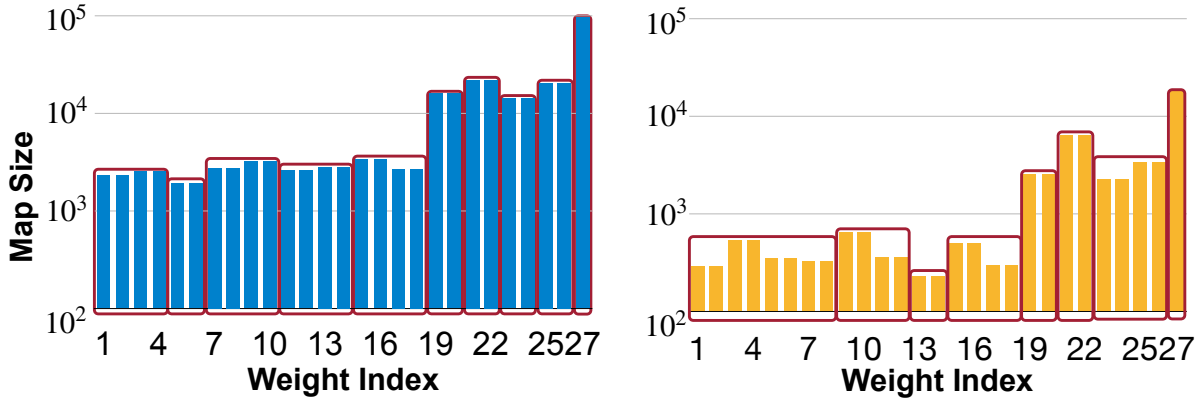


Figure 4.11: TorchSparse consistently outperforms state-of-the-art inference engines in both detection and segmentation benchmarks and achieves up to 1.5-1.6× geomean speedup, 2.3× single model speedup over MinkowskiEngine and SpConv.

ilar manner to the adaptive grouping. Here, `grid` corresponds to a naive collision-free grid-based hashmap: it takes larger memory space, but hashmap construction/query requires exactly one DRAM access per-entry, which is much smaller than conventional hashmaps. We then perform kernel fusion (Figure 4.10) on output coordinates computation for downsampling. The downsample operation applies a sliding window around each point. It ❶ calculates candidate activated points with `broadcast_add`, ❷ performs modular check, ❸ performs boundary check and generates a mask on whether each point is kept, ❹ converts the remaining candidate point coordinates to 1D values, and ❺ performs `unique` operation to keep final output coordinates (detailed in Appendix A). There are DRAM accesses between every two of the five stages, making downsampling kernels memory-bounded. We therefore fuse stages ❶ to ❹ into a single kernel and use registers to store intermediate results, which eliminates all intermediate DRAM write. For the fused kernel, we further perform control logic simplification, full loop unrolling and utilize the symmetry of submanifold maps. Overall, the mapping operations are accelerated by **4.6**×



(a) Map size on SemanticKITTI

(b) Map size on nuScenes

Figure 4.12: Grouping strategy on different datasets. Maps on nuScenes are much smaller than on SemanticKITTI for MinkUNet. Thus, to fully utilize GPU, the grouping strategy is more aggressive on nuScenes (8 groups *vs*. 10 groups).

on detection tasks with our optimizations.

4.1.6 Evaluation

Setup

TorchSparse is implemented in CUDA and provides easy-to-use PyTorch-like interfaces (described in Section 4.1.5). We build TorchSparse based on PyTorch 1.9.1 with CUDA 10.2/11.1 and cuDNN 7.6.5. Our system is evaluated against a baseline FP32 design without optimizations in Section 4.1.5 and the latest versions of two state-of-the-art sparse convolution libraries MinkowskiEngine v0.5.4 [130] and SpConv v1.2.1 [61] on three generations of NVIDIA GPUs: GTX 1080Ti, RTX 2080Ti and RTX 3090. Necessary changes are made to MinkowskiEngine to correctly support downsample operations in detectors and to SpConv to avoid OOM in large-scale scenes.

All systems are evaluated on seven top-performing sparse CNNs on large-scale datasets: MinkUNet [130] (0.5×/1× width) on SemanticKITTI [214], MinkUNet (1/3 frames) on nuScenes-LiDARSeg [73], CenterPoint [229] (10 frames) on nuScenes detection and CenterPoint (1/3 frames) on Waymo Open Dataset [240]. We report the normalized FPS for all systems (with TorchSparse to be 1).

Evaluation Results

Our TorchSparse achieves the best performance compared with the baseline design, MinkowskiEngine and SpConv.

From Figure 4.11, TorchSparse achieves up to 2.16× speedup on segmentation models and 1.6-2× speedup on detection models over MinkowskiEngine on RTX 3090. We achieve a smaller speedup for the 1-frame MinkUNet on nuScenes-LiDARSeg because MinkowskiEngine applies specialized optimizations to small models by using the *fetch-on-demand* dataflow [134] instead of the gather-matmul-scatter dataflow.

TorchSparse also demonstrates a 1.2× faster inference speed compared with the FP16 version of SpConv for detectors on RTX3090 thanks to our fused and locality-aware access pattern and almost perfect speedup from vectorized data movement. Note that we report *end-to-end* speedup in Figure 4.11. However, 10% of total total runtime in CenterPoint [229] is not related to point cloud computation (image convolution and non-maximum suppression, as in Figure 4.4). Therefore, our speedup ratio on sparse convolution is 10% more for CenterPoint. The performance gain over SpConv (FP16) is even larger on segmentation models on various hardware platforms thanks to the effectiveness of adaptively batched matrix multiplication, which will be discussed in Section 4.1.7. GPUs are usually more under-utilized for segmentation models as they usually have smaller workload compared with detectors, making it necessary to apply batching strategies to improve the device utilization.

TorchSparse achieves consistent speedup over other systems on GTX 1080Ti, which has *no* FP16 tensor cores. Compared with the baseline design, our TorchSparse still achieves a 1.5× speedup, only 11% less than the speedup we achieved on RTX 2080Ti with tensor cores. This validates that the native tensor-core speedup only constitutes a very minor proportion of our performance gain.

TorchSparse runs MinkUNet (1.0× on SemanticKITTI) at 36, 26 and 13 FPS on RTX 3090, RTX 2080Ti, GTX 1080Ti, respectively, all satisfying the real-time requirement ($\geq$ 10 FPS). For the 3-frame model on nuScenes-LiDARSeg, TorchSparse achieves 45, 40 and 25 FPS throughput on the three devices, at least 2× faster than the LiDAR frequency. Even for the heaviest 3-frame CenterPoint model on Waymo, TorchSparse is still able to achieve the real-time inference on GTX 1080Ti. As such, our system paves the way for real-time LiDAR perception on self-driving cars.

4.1.7 Ablation Study

Matrix Multiplication Optimizations

We first examine the performance of different grouping strategies on SemanticKITTI with MinkUNet (0.5×) and on nuScenes with MinkUNet (3 frames). From Figure 4.6, our adaptive grouping strategy outperforms all handcrafted, fixed strategy and achieves 1.4-

Specialization for Different Datasets		Optimized for	
		SemanticKITTI	nuScenes
Execute on	SemanticKITTI	**10.11**	10.87
	nuScenes	5.30	**4.67**

(a) Specialization for Datasets (MinkUNet, 2080Ti)

Specialization for Different Models		Optimized for	
		MinkUNet (1.0×)	MinkUNet (0.5×)
Execute on	MinkUNet (1.0×)	**10.11**	10.70
	MinkUNet (0.5×)	5.37	**4.72**

(b) Specialization for Model (SemanticKITTI, 2080Ti)

Specialization for Different Hardware		Optimized for	
		RTX2080Ti	GTX1080Ti
Execute on	RTX2080Ti	**4.67**	4.80
	GTX1080Ti	14.95	**14.01**

(c) Specialization for Hardware (nuScenes, MinkUNet)

Table 4.1: Specializing adaptive batching strategies for different datasets, models and hardware platforms helps improve efficiency (TFLOP/s) by up to 13.5%.

1.5× over no grouping baseline. Table 4.2 also suggests that manually-designed strategy cannot generalize to all datasets: fixed 3-batch grouping achieves large speedup (1.5×) on nuScenes, but is 13% slower than the separate computation baseline on SemanticKITTI. Note that although this strategy has the best device utilization (largest TFLOP/s) on nuScenes, it does not bring greater latency reduction than adaptive grouping due to much more extra computation, indicating the importance of ϵ in our adaptive grouping algorithm. We also show the effectiveness of grouping strategy specialization for different datasets, model and hardware in Table 4.1. In Table 4.1a, we found that the same model (1-frame MinkUNet) on the same hardware platform benefits more from the dataset-specialized strategy. This is because map size distributions (which decides the workload of matrix multiplication) significantly differ between SemanticKITTI and nuScenes, as shown in Figure 4.12. The maps on nuScenes are much smaller than those on SemanticKITTI. As

Grouping Method	MatMul speedup (SK)	MatMul Speedup (NS)
Separate	8.1 TFLOP/s (1.00×)	10.4 TFLOP/s (1.00×)
Symmetric	8.2 TFLOP/s (1.02×)	14.6 TFLOP/s (1.39×)
Fixed	8.7 TFLOP/s (0.87×)	**21.1 TFLOP/s** (1.50×)
Adaptive	**11.9 TFLOP/s (1.39×)**	16.9 TFLOP/s (**1.54**×)

Table 4.2: Ablation analysis on matrix multiplication: adaptive batching consistently outperforms all other strategies in latency and brings about 1.4×-1.5× speedup for matmul (SK=SemanticKITTI, NS=nuScenes). As we trade FLOPs for regularity, TFLOP/s and speedup are non-proportional.

FP16	Vec.	Fused	Loc.-Aware	Speedup (G)	Speedup (S)	Speedup (SG)
✗	✗	✗	✗	1.00	1.00	1.00
✓	✗	✗	✗	1.17	1.48	1.32
✓	✓	✗	✗	1.91	1.95	1.93
✓	✓	✓	✗	1.91	2.12	2.02
✓	✓	✓	✓	**2.86**	**2.61**	**2.72**

Table 4.3: Speedup breakdown of different optimizations to reduce data movement. Feature quantization, vectorized memory access, and fused and locality-aware access bring 1.3×, 1.5× and 1.4× speedup, respectively. Here, *G* and *S* denote gather and scatter.

a result, if we directly transfer SemanticKITTI strategy to nuScenes, the groups will not be large enough to fully utilize hardware resources. On the other hand, if the nuScenes strategy is transferred to SemanticKITTI, the efficiency will be bottlenecked by computation overhead. We notice similar effect for model and hardware specialization in Table 4.1b and Table 4.1c, where specialized strategies always outperform the transferred ones.

Data Movement Optimizations

We then perform ablation analysis on MinkUNet [130] (1.0×) on the SemanticKITTI dataset [214]. As in Table 4.3, naively quantizing features to 16-bit will not provide significant speedup for scatter/gather: especially for gather, the speedup ratio is only 1.17×, far less than the theoretical value (2×). Instead, quantized and *vectorized* scatter/gather improves the latency of scatter-gather by 1.93×, which closely matches the DRAM access reduction and verifies our analysis in Section 4.1.5 on memory transactions. We further observe that fusing gather/scatter itself will *not* provide substantial speedup, as the *weight-stationary* access pattern cannot provide good cache locality due to the uniqueness of maps for each weight. However, when combined with *locality-aware* access, we achieve **2.86**× speedup on gathering, **2.61**× speedup on scattering and **2.72**× overall speedup against

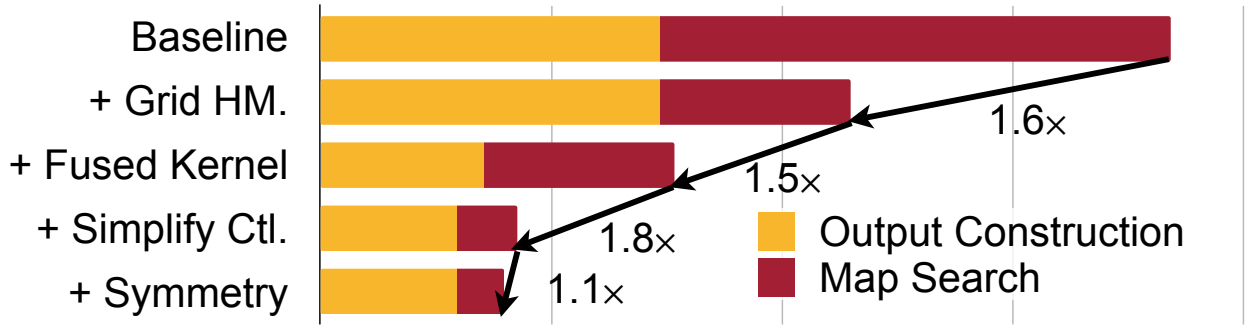


Figure 4.13: Speedup breakdown of mapping optimizations. Grid-based hashmap, fused kernel, simplified control logic and symmetry bring 1.6×, 1.5×, 1.8× and 1.1× measured speedup, respectively.

FP32. This demonstrates the fact that all techniques in Section 4.1.5 are crucial in improving the efficiency of data movement.

Mapping Optimizations

We finally present analysis on optimizing mapping operations in 3-frame CenterPoint [229] detector on Waymo [240]. Grid-based map search is 2.7× faster than a general hashmap-based solution thanks to its no-collision property, resulting in a 1.6× end-to-end speedup for mapping. Fusing four small kernels accelerates output construction by 2.1× and brings 1.5× further end-to-end mapping speedup. Finally, simplifying the control logic, loop unrolling and utilizing the symmetry of maps substantially accelerates map search by another 4× and pushes the final end-to-end mapping speedup to **4.6**×.

4.1.8 Discussion

We present TorchSparse, an open-source inference engine for efficient point cloud neural networks. Guided by two general principles: trade computation for regularity and reduce memory footprint, we optimize matrix multiplication, data movement and mapping operations in sparse convolutions, achieving up to 1.5×, 2.7× and 4.6× speedup on these three components, and up to 1.5-1.6× end-to-end speedup over previous state-of-the-art point cloud inference engines on both segmentation and detection tasks. We hope that our in-depth analysis on the efficiency bottlenecks and optimization recipes for sparse convolution can inspire future research on point cloud inference engine design.

Part III

Applications

Chapter 5

Accelerating 3D Perception for Autonomous Driving

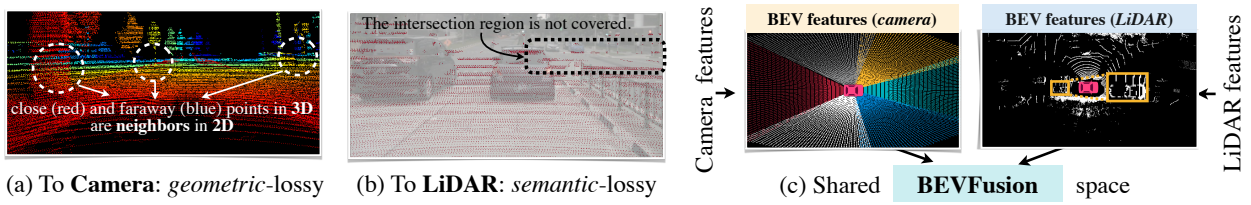
5.1 BEVFusion

Multi-sensor fusion is essential for an accurate and reliable autonomous driving system. Recent approaches are based on point-level fusion: augmenting the LiDAR point cloud with camera features. However, the camera-to-LiDAR projection throws away the semantic density of camera features, hindering the effectiveness of such methods, especially for semantic-oriented tasks (such as 3D scene segmentation). In this paper, we break this deeply-rooted convention with BEVFusion, an efficient and generic multi-task multi-sensor fusion framework. It unifies multi-modal features in the shared bird's-eye view (BEV) representation space, which nicely preserves both geometric and semantic information. To achieve this, we diagnose and lift key efficiency bottlenecks in the view transformation with optimized BEV pooling, reducing latency by more than **40**×. BEVFusion is fundamentally task-agnostic and seamlessly supports different 3D perception tasks with almost no architectural changes. It establishes the new state of the art on the nuScenes benchmark, achieving **1.3%** higher mAP and NDS on 3D object detection and **13.6%** higher mIoU on BEV map segmentation, with **1.9**× lower computation cost.

5.1.1 Introduction

Autonomous driving systems are equipped with diverse sensors. For instance, Waymo's self-driving vehicles have 29 cameras, 6 radars, and 5 LiDARs[*]. Different sensors provide complementary signals: *e.g.*, cameras capture rich semantic information, LiDARs provide

[*]https://www.wallpaper.com/transport/waymo-division-autonomous-car



(a) To **Camera**: *geometric*-lossy (b) To **LiDAR**: *semantic*-lossy (c) Shared **BEVFusion** space

Figure 5.1: BEVFusion unifies camera and LiDAR features in a *shared* BEV space instead of mapping one modality to the other. It preserves camera's *semantic density* and LiDAR's *geometric structure*.

accurate spatial information, while radars offer instant velocity estimation. Therefore, multi-sensor fusion is of great importance for accurate and reliable perception.

Data from different sensors are expressed in fundamentally different modalities: *e.g.*, cameras capture data in perspective view and LiDAR in 3D view. To resolve this view discrepancy, we have to find a *unified representation* that is suitable for multi-task multi-modal feature fusion. Due to the tremendous success in 2D perception, the natural idea is to project the LiDAR point cloud onto the camera and process the RGB-D data with 2D CNNs. However, this LiDAR-to-camera projection introduces severe geometric distortion (see Figure 5.1a), which makes it less effective for geometric-oriented tasks, such as 3D object recognition.

Recent sensor fusion methods follow the other direction. They augment the LiDAR point cloud with semantic labels [263], CNN features [265, 288] or virtual points from 2D images [264], and then apply an existing LiDAR-based detector to predict 3D bounding boxes. Although they have demonstrated remarkable performance on large-scale detection benchmarks, these point-level fusion methods barely work on semantic-oriented tasks, such as BEV map segmentation [289–292]. This is because the camera-to-LiDAR projection is semantically lossy (see Figure 5.1b): for a typical 32-beam LiDAR scanner, only 5% camera features will be matched to a LiDAR point while all others will be dropped. Such density differences will become even more drastic for sparser LiDARs (or imaging radars).

In this paper, we propose BEVFusion to unify multi-modal features in a shared bird's-eye view (BEV) representation space for task-agnostic learning. We maintain both geometric structure and semantic density (see Figure 5.1c) and naturally support most 3D perception tasks (since their output space can be naturally captured in BEV). While converting all features to BEV, we identify the major prohibitive efficiency bottleneck in the view transformation: *i.e.*, the BEV pooling operation alone takes more than 80% of the model's runtime. Then, we propose a specialized kernel with precomputation and interval reduction to eliminate this bottleneck, achieving more than **40**$\times$ speedup. Finally, we

apply the fully-convolutional BEV encoder to fuse the unified BEV features and append a few task-specific heads to support different target tasks.

BEVFusion sets the new state-of-the-art performance on the nuScenes benchmark. On 3D object detection, it ranks **1st** on the leaderboard among all solutions that do not use test-time augmentation and model ensemble. BEVFusion demonstrates even more significant improvements on BEV map segmentation. It achieves **6%** higher mIoU than camera-only models and **13.6%** higher mIoU than LiDAR-only models, while existing fusion methods hardly work. Moreover, BEVFusion is efficient, delivering all these results with **1.9**$\times$ lower computation cost.

BEVFusion breaks the long-standing belief that point-level fusion is the best solution to multi-sensor fusion. Simplicity is also its key strength. We hope this work will serve as a simple yet strong baseline for future sensor fusion research and inspire the researchers to rethink the design and paradigm for generic multi-task multi-sensor fusion.

5.1.2 Related Work

LiDAR-Based 3D Perception. Researchers have designed single-stage 3D object detectors [61, 169, 242–245] that extract flattened point cloud features using PointNets [151] or SparseConvNet [120] and perform detection in the BEV space. Later, Yin *et al*. [229] explores anchor-free 3D object detection, which is concurrent with [247, 249–253]. Another stream of research [183, 238, 239, 257–259] focus on two-stage object detector design, which adds an RCNN network to existing one-stage object detectors. There are also U-Net like models specialized for 3D semantic segmentation [23, 53, 120, 130, 293], an important task for offline HD map construction.

Camera-Based 3D Perception. Due to the high cost of LiDAR sensors, researchers spend significant efforts on camera-only 3D perception. FCOS3D [294] extends an image detector [295] with additional 3D regression branches, which is later improved by [296, 297] in depth modeling. Instead of performing object detection in the perspective view, DETR3D [274] and PETR [298] design a DETR [272, 299]-based detection head with learnable object queries in the 3D space. Inspired by the design of LiDAR-based detectors, another type of camera-only 3D perception models explicitly converts the camera features from perspective view to the bird's-eye view using a view transformer [289, 290, 300, 301]. BEVDet [28] and M^{2}BEV [302] extends LSS [290] and OFT [300] to 3D object detection and CaDDN [303] adds explicit depth estimation supervision to the view transformer. Recent research [275, 292] also studies using multi-head attention to perform the view

transformation.

Multi-Sensor Fusion. Recently, multi-sensor fusion arouses significant interest among the 3D detection community. Existing approaches can be classified into *proposal-level* and *point-level* fusion methods. Early approach MV3D [269] creates object proposals in 3D and projects the proposals to images to extract RoI features. F-PointNet [182], F-ConvNet [241] and CenterFusion [304] all lift image proposals into a 3D frustum. Recent work FUTR3D [281] and TransFusion [267] define object queries in the 3D space and fuses image features onto these proposals. All proposal-level fusion methods are *object-centric* and cannot trivially generalize to other tasks such as BEV map segmentation. Point-level fusion methods, on the other hand, usually paint image semantic features onto foreground LiDAR points and perform LiDAR-based detection on the decorated point cloud inputs. As such, they are both *object-centric* and *geometric-centric*. Among these methods, PointPainting [263], PointAugmenting [288], MVP [264], FusionPainting [305] and AutoAlign [306] are (LiDAR) input-level decoration, while Deep Continuous Fusion [266] and DeepFusion [265] are feature-level decoration.

Multi-Task Learning. Multi-task CNNs have also been well-studied in the 2D computer vision community. [307, 308] jointly perform object detection and instance segmentation, while [77, 82, 309, 310] are also applicable to pose estimation and human-object interaction. Recent concurrent research M^2BEV [302] and BEVFormer [275] jointly performs object detection and BEV segmentation in 3D. None of the above methods considers multi-sensor fusion. MMF [311] simultaneously works on depth completion and object detection with both camera and LiDAR inputs, but is still object-centric and not applicable to BEV map segmentation.

In contrast to all existing methods, BEVFusion performs sensor fusion in a shared BEV space and treats foreground and background, geometric and semantic information equally. It is a generic multi-task multi-sensor perception framework.

5.1.3 Method

BEVFusion focuses on *multi-sensor fusion* (*i.e.*, multi-view cameras and LiDAR) for *multi-task 3D perception* (*i.e.*, detection and segmentation). We provide an overview of our framework in Figure 5.2. Given different sensory inputs, we first apply modality-specific encoders to extract their features. We transform multi-modal features into a unified BEV representation that preserves both geometric and semantic information. We

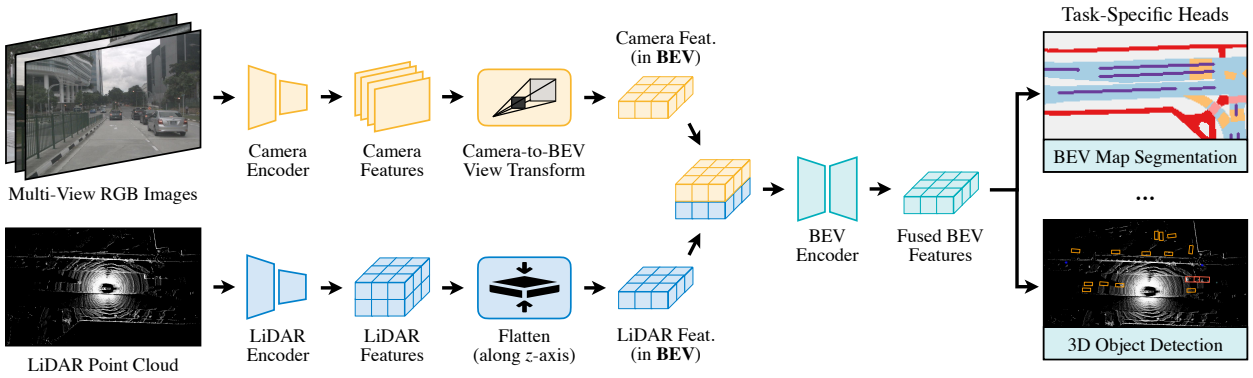


Figure 5.2: **BEVFusion** extracts features from multi-modal inputs and converts them into a shared bird's-eye view (BEV) space efficiently using view transformations. It fuses the unified BEV features with a fully-convolutional BEV encoder and supports different tasks with task-specific heads.

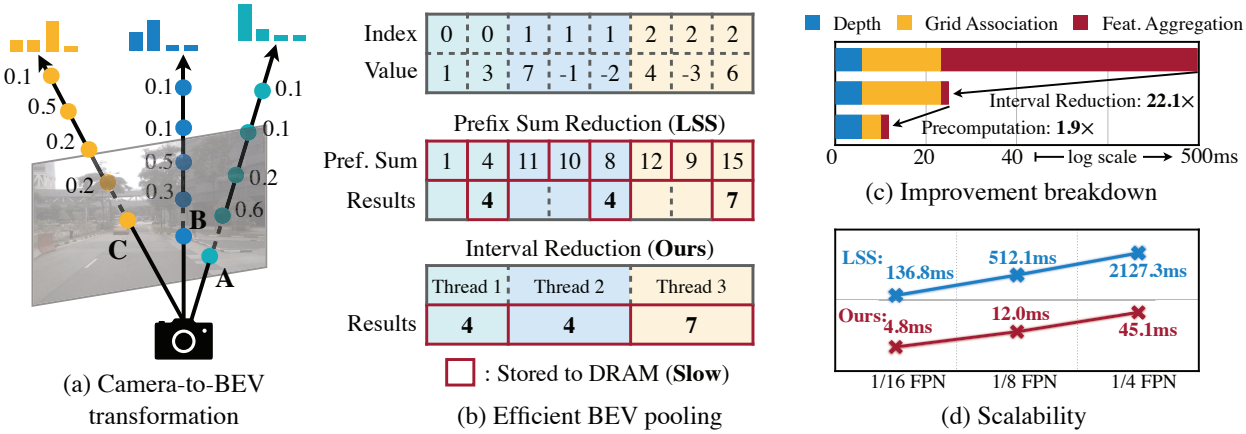
identify the efficiency bottleneck of the view transformation and accelerate BEV pooling with precomputation and interval reduction. We then apply the convolution-based BEV encoder to the unified BEV features to alleviate the local misalignment between different features. Finally, we append a few task-specific heads to support different 3D tasks.

Unified Representation

Different features can exist in different views. For instance, camera features are in the perspective view, while LiDAR/radar features are typically in the 3D/bird's-eye view. Even for camera features, each one of them has a distinct viewing angle (*i.e.*, front, back, left, right). This *view discrepancy* makes the feature fusion difficult since the same element in different feature tensors might correspond to completely different spatial locations (and the naïve elementwise feature fusion will not work in this case). Therefore, it is crucial to find a *shared* representation, such that (1) all sensor features can be easily converted to it without information loss, and (2) it is suitable for different types of tasks.

To Camera. Motivated by RGB-D data, one choice is to project the LiDAR point cloud to the camera plane and render the 2.5D sparse depth. However, this conversion is *geometrically lossy*. Two neighbors on the depth map can be far away from each other in the 3D space. This makes the camera view less effective for tasks that focus on the object/scene geometry, such as 3D object detection.

To LiDAR. Most state-of-the-art sensor fusion methods [263–265] decorate LiDAR points with their corresponding camera features (*e.g.*, semantic labels, CNN features or virtual



(a) Camera-to-BEV transformation

(b) Efficient BEV pooling

(c) Improvement breakdown

(d) Scalability

Figure 5.3: Camera-to-BEV transformation (a) is the key step to perform sensor fusion in the unified BEV space. However, existing implementation is extremely slow and can take up to 2s for a single scene. We propose efficient BEV pooling (b) using interval reduction and fast grid association with precomputation, bringing about **40**× speedup to the view transformation module (c, d).

points). However, this camera-to-LiDAR projection is *semantically lossy*. Camera and LiDAR features have drastically different densities, resulting in only less than 5% of camera features being matched to a LiDAR point (for a 32-channel LiDAR scanner). Giving up the semantic density of camera features severely hurts the model's performance on semantic-oriented tasks (such as BEV map segmentation). Similar drawbacks also apply to more recent fusion methods in the latent space (*e.g.*, object query) [267, 281].

To Bird's-Eye View. We *adopt the bird's-eye view (BEV) as the unified representation for fusion*. This view is friendly to almost all perception tasks since the output space is also in BEV. More importantly, the transformation to BEV keeps both geometric structure (from LiDAR features) and semantic density (from camera features). On the one hand, the LiDAR-to-BEV projection flattens the sparse LiDAR features along the height dimension, thus does not create geometric distortion in Figure 5.1a. On the other hand, camera-to-BEV projection casts each camera feature pixel back into a ray in the 3D space (detailed in the next section), which can result in a dense BEV feature map in Figure 5.1c that retains full semantic information from the cameras.

Efficient Camera-to-BEV Transformation

Camera-to-BEV transformation is non-trivial because the depth associated with each camera feature pixel is inherently ambiguous. Following LSS [290], we explicitly predict the

discrete depth distribution of each pixel. We then scatter each feature pixel into D discrete points along the camera ray and rescale the associated features by their corresponding depth probabilities (Figure 5.3a). This generates a camera feature point cloud of size $NHWD$, where N is the number of cameras and (H, W) is the camera feature map size. Such 3D feature point cloud is quantized along the x, y axes with a step size of r (*e.g.*, 0.4m). We use the *BEV pooling* operation to aggregate all features within each $r \times r$ BEV grid and flatten the features along the z-axis.

Though simple, BEV pooling is surprisingly inefficient and slow, taking more than 500ms on an RTX 3090 GPU (while the rest of our model only takes around 100ms). This is because the camera feature point cloud is very large: for a typical workload[†], there could be around 2 million points generated for each frame, two orders of magnitudes denser than a LiDAR feature point cloud. To lift this efficiency bottleneck, we propose to optimize the BEV pooling with precomputation and interval reduction.

Precomputation. The first step of BEV pooling is to *associate* each point in the camera feature point cloud with a BEV grid. Different from LiDAR point clouds, the coordinates of the camera feature point cloud are *fixed* (as long as the camera intrinsics and extrinsics stay the same, which is usually the case after proper calibration). Motivated by this, we precompute the 3D coordinate and the BEV grid index of each point. We also sort all points according to grid indices and record the rank of each point. During inference, we only need to reorder all feature points based on the precomputed ranks. This caching mechanism can reduce the latency of grid association from 17ms to 4ms.

Interval Reduction. After grid association, all points within the same BEV grid will be consecutive in the tensor representation. The next step of BEV pooling is to *aggregate* the features within each BEV grid by some symmetric function (*e.g.*, mean, max, and sum). As in Figure 5.3b, existing implementation [290] first computes the prefix sum over all points and then subtracts the values at the boundaries where indices change. However, the prefix sum operation requires tree reduction on the GPU and produces many unused partial sums (since we only need those values on the boundaries), both of which are inefficient. To accelerate feature aggregation, we implement a specialized GPU kernel that parallelizes directly over BEV grids: we assign a GPU thread to each grid that calculates its interval sum and writes the result back. This kernel removes the dependency between outputs

[†] $N = 6$, $(H, W) = (32, 88)$, and $D = (60 - 1)/0.5 = 118$. This corresponds to six multi-view cameras, each associated with a 32×88 camera feature map (which is downsampled from a 256×704 image by 8×). The depth is discretized into $[1, 60]$ meters with a step size of 0.5 meter.

(thus does not require multi-level tree reduction) and avoids writing the partial sums to the DRAM, reducing the latency of feature aggregation from 500ms to 2ms (Figure 5.3c).

Takeaways. The camera-to-BEV transformation is **40**× faster with our optimized BEV pooling: the latency is reduced from more than 500ms to 12ms (only 10% of our model's end-to-end runtime) and scales well across different feature resolutions (Figure 5.3d). This is a key enabler for unifying multi-modal sensory features in the shared BEV representation. Two concurrent works of ours also identify this efficiency bottleneck in the camera-only 3D detection. They approximate the view transformer by assuming uniform depth distribution [302] or truncating the points within each BEV grid [28]. In contrast, our techniques are *exact* without any approximation, while still being faster.

Fully-Convolutional Fusion

With all sensory features converted to the shared BEV representation, we can easily fuse them together with an elementwise operator (such as concatenation). Though in the same space, LiDAR BEV features and camera BEV features can still be spatially misaligned to some extent due to the inaccurate depth in the view transformer. To this end, we apply a convolution-based BEV encoder (with a few residual blocks) to compensate for such local misalignments. We also experiment with more advanced fusion mechanisms, such as gated [312], non-local [313] and deformable [314] fusion. However, we find that all these designs bring about at most 0.1% improvement for the final accuracy. We believe this is because the BEV encoder (with more than 10 layers) already has enough capacity. Our method could potentially benefit from more accurate depth estimation (*e.g.*, supervising the view transformer with ground-truth depth [303, 315]), which we leave for future work.

Multi-Task Heads

We apply multiple task-specific heads to the fused BEV feature map. Our method is applicable to most 3D perception tasks. We showcase two examples: 3D object detection and BEV map segmentation.

Detection. We use a class-specific center heatmap head to predict the center location of all objects and a few regression heads to estimate the object size, rotation, and velocity. We refer the readers to previous 3D detection papers [229, 267] for more details.

	Modality	mAP (*test*)	NDS (*test*)	mAP (*val*)	NDS (*val*)	MACs (G)	Latency (ms)
M^2BEV [302]	C	42.9	47.4	41.7	47.0	–	–
BEVFormer [275]	C	44.5	53.5	41.6	51.7	–	–
PointPillars [242]	L	–	–	52.3	61.3	65.5	34.4
SECOND [61]	L	52.8	63.3	52.6	63.0	85.0	69.8
CenterPoint [229]	L	60.3	67.3	59.6	66.8	153.5	80.7
PointPainting [263]	C+L	–	–	65.8*	69.6*	370.0	185.8
PointAugmenting [288]	C+L	66.8†	71.0†	–	–	408.5	234.4
MVP [264]	C+L	66.4	70.5	66.1*	70.0*	371.7	187.1
FusionPainting [305]	C+L	68.1	71.6	66.5	70.7	–	–
AutoAlign [306]	C+L	–	–	66.6	71.1	–	–
FUTR3D [281]	C+L	–	–	64.5	68.3	1069.0	321.4
TransFusion [267]	C+L	68.9	71.6	67.5	71.3	485.8	156.6
BEVFusion	C+L	**70.2**	**72.9**	**68.5**	**71.4**	**253.2**	**119.2**

Table 5.1: BEVFusion achieves state-of-the-art 3D object detection performance on nuScenes (`val` and `test`) without bells and whistles. It breaks the convention of decorating camera features onto the LiDAR point cloud and delivers at least 1.3% higher mAP and NDS with **1.5-2**$\times$ lower computation cost. (*: our re-implementation; †: with test-time augmentation)

Segmentation. Different map categories may overlap (*e.g.*, crosswalk is a subset of drivable space). Therefore, we formulate this problem as multiple binary semantic segmentation, one for each class. We follow CVT [292] to train the segmentation head with the standard focal loss [74].

5.1.4 Experiments

We evaluate BEVFusion for camera-LiDAR fusion on 3D object detection and BEV map segmentation, covering both geometric- and semantic-oriented tasks. Our framework can be easily extended to support other types of sensors (such as radars and event-based cameras) and other 3D perception tasks (such as 3D object tracking and motion forecasting).

Model. We use Swin-T [29] as our image backbone and VoxelNet [61] as our LiDAR backbone. We apply FPN [316] to fuse multi-scale camera features to produce a feature map of 1/8 input size. We downsample camera images to 256$\times$704 and voxelize the LiDAR point cloud with 0.075m (for detection) and 0.1m (for segmentation). As detection and segmentation tasks require BEV feature maps with different spatial ranges and sizes, we apply grid sampling with bilinear interpolation before each task-specific head to explicitly transform between different BEV feature maps.

	Modality	Drivable	Ped. Cross.	Walkway	Stop Line	Carpark	Divider	Mean
OFT [300]	C	74.0	35.3	45.9	27.5	35.9	33.9	42.1
LSS [290]	C	75.4	38.8	46.3	30.3	39.1	36.5	44.4
CVT [292]	C	74.3	36.8	39.9	25.8	35.0	29.4	40.2
M^2BEV [302]	C	77.2	–	–	–	–	40.5	–
BEVFusion	C	**81.7**	**54.8**	**58.4**	**47.4**	**50.7**	**46.4**	**56.6**
PointPillars [242]	L	72.0	43.1	53.1	29.7	27.7	37.5	43.8
CenterPoint [229]	L	75.6	48.4	57.5	36.5	31.7	41.9	48.6
PointPainting [263]	C+L	75.9	48.5	57.1	36.9	34.5	41.9	49.1
MVP [264]	C+L	76.1	48.7	57.0	36.9	33.0	42.2	49.0
BEVFusion	C+L	**85.5**	**60.5**	**67.6**	**52.0**	**57.0**	**53.7**	**62.7**

Table 5.2: BEVFusion outperforms the state-of-the-art multi-sensor fusion methods by **13.6%** on BEV map segmentation on nuScenes (`val`) with consistent improvements across different categories.

Training. We pre-train the image backbone on nuImage (for 2D object detection) following [263, 264, 267, 288]. Unlike existing approaches [263, 267, 288] that freeze the camera encoder, we train the entire model in an end-to-end manner. We apply both image and LiDAR data augmentations to prevent overfitting. Optimization is carried out using AdamW [158] with a weight decay of 10^{-2}.

Dataset. We evaluate our method on nuScenes [73] which is a large-scale outdoor dataset. It has diverse annotations to support all sorts of tasks (such as 3D object detection/tracking and BEV map segmentation). Each of the 40,157 annotated samples contains six monocular camera images with 360-degree FoV and a 32-beam LiDAR scan.

3D Object Detection

We first experiment on the geometric-centric 3D object detection benchmark, where BEVFusion achieves superior performance with lower computation cost and measured latency.

Setting. We use the mean average precision (mAP) across 10 foreground classes and the nuScenes detection score (NDS) as our detection metrics. We also measure the single-inference #MACs and latency on an RTX3090 GPU for all open-source methods. We use a single model without any test-time augmentation for both `val` and `test` results.

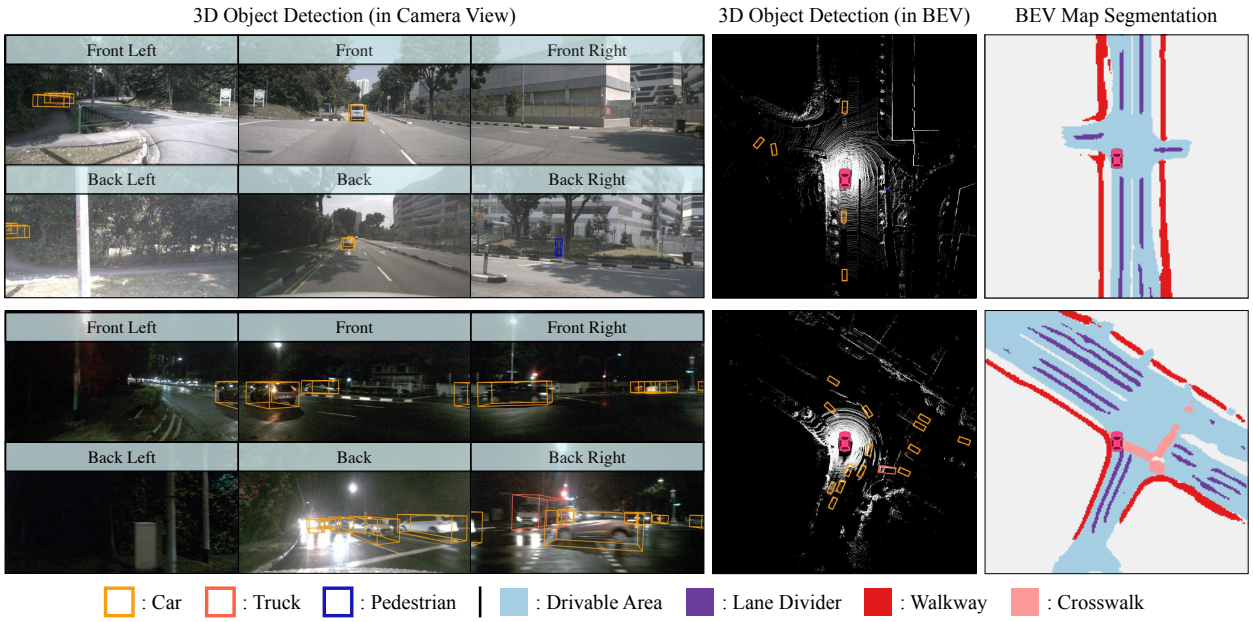


Figure 5.4: Qualitative results of BEVFusion on 3D object detection and BEV map segmentation. It accurately recognizes distant and small objects (top) and parses crowded nighttime scenes (bottom).

Results. As in Table 5.1, BEVFusion achieves state-of-the-art results on the nuScenes detection benchmark, with close-to-real-time (**8.4 FPS**) inference speed on a desktop GPU. Compared with TransFusion [267], BEVFusion achieve 1.3% improvement in `test` split mAP and NDS, while significantly reduces the MACs by **1.9**× and measured latency by **1.3**×. BEVFusion also compares favorably against representative point-level fusion methods PointPainting [263] and MVP [264] with **1.6**× speedup, **1.5**× MACs reduction and **3.8%** higher mAP on the `test` set. We argue that the efficiency gain of BEVFusion comes from the fact that we choose the BEV space as the share fusion space, which fully utilizes all camera features instead of just a 5% sparse set. As a result, BEVFusion can achieve the same performance with much smaller MACs. Combined with the efficient BEV pooling operator in Section 5.1.3, BEVFusion transfers MACs reduction into measured speedup.

BEV Map Segmentation

We further compare BEVFusion with state-of-the-art 3D perception models on the semantic-centric BEV map segmentation task, where BEVFusion achieves an even larger performance boost.

Setting. We report the Intersection-over-Union (IoU) on 6 background classes (drivable space, pedestrian crossing, walkway, stop line, car-parking area, and lane divider) and the class-averaged mean IoU as our evaluation metric. As different classes may have overlappings (*e.g.* car-parking area is also drivable), we evaluate the binary segmentation performance for each class separately and select the highest IoU across different thresholds [292]. For each frame, we only perform the evaluation in the [-50m, 50m]×[-50m, 50m] region around the ego car following [275, 290, 292, 302]. In BEVFusion, we use a single model that jointly performs binary segmentation for all classes instead of following the conventional approach to train a separate model for each class. This results in **6**× faster inference and training. We reproduced the results of all open-source competing methods.

Results. We report the BEV map segmentation results in Table 5.2. In contrast to 3D object detection which is a *geometric*-oriented task, map segmentation is *semantic*-oriented. As a result, our camera-only BEVFusion model outperforms LiDAR-only baselines by **8-13%**. This observation is the exact opposite of results in Table 5.1, where state-of-the-art camera-only 3D detectors got outperformed by LiDAR-only detectors by almost 20 mAP. Our camera-only model boosts the performance of existing monocular BEV map segmentation methods by at least **12%**. In the multi-modality setting, we further improve the performance of the monocular BEVFusion by **6** mIoU and achieved >**13%** improvement over state-of-the-art sensor fusion methods [263, 264]. This is because both baseline methods are *object*-centric and *geometric*-oriented. PointPainting [263] only decorates the *foreground* LiDAR points and MVP only densifies *foreground* 3D objects. Both approaches are not helpful for segmenting map components. Worse still, both methods assume that LiDAR should be the more effective modality in sensor fusion, which is not true according to our observations in Table 5.2.

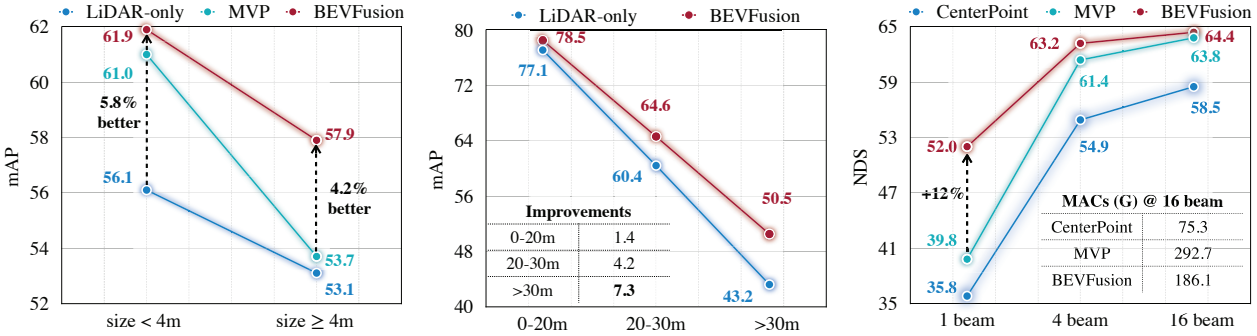
5.1.5 Analysis

We present in-depth analyses of BEVFusion over single-modality models and state-of-the-art multi-modality models under different circumstances.

Weather and Lighting. We systematically analyze the performance of BEVFusion under different weather and lighting conditions in Table 5.3. Detecting objects in rainy weather is challenging for LiDAR-only models due to significant sensor noises. Thanks to the robustness of camera sensors under different weathers, BEVFusion improves CenterPoint by **10.7** mAP, closing the performance gap between sunny and rainy scenarios. Poor

		Sunny		Rainy		Day		Night	
	Modality	mAP	mIoU	mAP	mIoU	mAP	mIoU	mAP	mIoU
CenterPoint [229]	L	62.9	50.7	59.2	42.3	62.8	48.9	35.4	37.0
TransFusion-L [267]	L	64.5	–	64.3	–	69.5	–	36.5	–
BEVFormer [275]	C	41.0	–	44.0	–	41.9	–	21.2	–
BEVFusion	C	–	59.0	–	50.5	–	57.4	–	30.8
MVP	C+L	65.9	51.0	66.3	42.9	66.3	49.2	38.4	37.5
BEVFusion	C+L	68.2	65.6	69.9	55.9	68.5	63.1	42.8	43.6

Table 5.3: BEVFusion is robust under different lighting and weather conditions, significantly boosting the performance single-modality models under challenging rainy and nighttime scenes.



(a) Different object sizes (b) Different object distances (c) Different LiDAR sparsity

Figure 5.5: BEVFusion consistently outperforms state-of-the-art single- and multi-modality detectors under different LiDAR sparsity, object sizes and object distances from the ego car, especially under the more challenging settings (*i.e.*, *sparser point clouds*, *small/distant objects*).

lighting conditions are challenging for both detection and segmentation models. For detection, MVP achieves a much smaller improvement compared to BEVFusion since it requires *accurate* 2D instance segmentations to generate virtual point generation. This can be very challenging in dark or overexposed scenes (*e.g.*, the second scene of Figure 5.4). For segmentation, even if the camera-only BEVFusion greatly outperforms CenterPoint on the entire dataset in Table 5.2, its performance is much worse at nighttime. Our BEVFusion significantly boosts its performance by **12.8** mIoU, which is even larger than the improvement in the daytime, demonstrating the significance of geometric clues when camera sensors fail.

Sizes and Distances. We also analyze the performance under different object sizes and distances. From Figure 5.5a, BEVFusion achieves consistent improvements over its

(a) Modality

	mAP	NDS	mIoU
L	57.6	64.9	48.6
C	33.3	40.2	56.6
L+C	**66.4**	**69.5**	**62.7**

(b) Data augmentation

	mAP	NDS	mIoU
Image	63.8	68.1	62.3
LiDAR	65.7	69.1	61.3
Both	**66.4**	**69.5**	**62.7**

(c) Image backbone

	mAP	NDS	mIoU
ResNet50	65.3	68.9	59.2
SwinT (freeze)	66.1	69.3	52.8
SwinT	**66.4**	**69.5**	**62.7**

(d) FPN size

	mAP	NDS	mIoU
1/16	66.2	69.4	**62.7**
1/8	**66.4**	**69.5**	**62.7**
1/4	66.0	69.2	58.8

(e) Voxel size

	mAP	NDS	mIoU
0.075	**67.1**	**70.2**	60.6
0.1	66.4	69.5	62.7
0.125	65.1	68.6	**63.7**

(f) Image size

	mAP	NDS	mIoU
128×352	64.0	68.2	60.5
256×704	66.4	69.5	**62.7**
384×1056	**67.2**	**70.0**	59.9

Table 5.4: Ablation experiments to validate our design choices. Default settings are marked in gray.

LiDAR-only counterpart for both small and large objects, while MVP has only negligible improvements for objects larger than 4m. This is because larger objects are typically much denser, benefiting less from those augmented multi-modal virtual points (MVPs). Besides, BEVFusion brings larger improvements to the LiDAR-only model for smaller objects (Figure 5.5a) and more distant objects (Figure 5.5b), both of which are poorly covered by LiDAR and can therefore benefit more from the dense camera information.

Sparser LiDARs. We demonstrate the performance of the LiDAR-only detector CenterPoint [229], multi-modality detector MVP [264] and our BEVFusion under different LiDAR sparsity in Figure 5.5c. BEVFusion consistently outperforms MVP under all sparsity levels with **1.6**× MACs reduction and achieves a **12%** improvement in the 1-beam LiDAR scenario. MVP decorates the input point cloud and directly applies CenterPoint on the painted and densified LiDAR input. Thus, it naturally requires the LiDAR-only CenterPoint detector to perform well, which is not valid under sparse LiDAR settings (35.8 NDS with 1-beam input in Figure 5.5c). BEVFusion, in contrast, fuses multi-sensory information in a shared BEV space, and thus does not assume a strong LiDAR-only detector.

Multi-Task Learning. This paper focuses on the setting where different tasks are trained separately. Here, we present a pilot study of joint 3D detection and segmentation training. We re-scale the loss for different tasks to the same magnitude and apply a separate BEV encoder for each task to provide the capability of learning more task-specific features. From Table 5.5, jointly training different tasks together has a negative impact on the performance of each individual task, which is widely known as "negative transfer". Separating BEV encoders partially alleviates this problem. A more sophisticated training scheme could

	3D Detection (NDS)	BEV Segmentation (mIoU)
Detection only	70.4	–
Segmentation only	–	58.5
Joint (*shared BEV encoders*)	69.7	54.0
Joint (*separate BEV encoders*)	**69.9**	**58.4**

Table 5.5: Joint detection and segmentation training (trained for 10 epochs).

further close this gap, which we leave for future work.

Ablation Studies. We present ablation studies in Table 5.4 to justify our design choices, where we use a shorter training schedule for detectors. Our experiments show that choosing BEV space as the unified representation for sensor fusion brings about the largest improvement for both detection and segmentation models.

For 3D object detection, we train BEVFusion with the same ResNet50 backbone, voxel size (0.075m) as TransFusion [267] and much smaller image and FPN sizes, achieving 68.1 mAP (+0.6 over TransFusion). TransFusion applies only LiDAR augmentations, and our accuracy improvement over TransFusion can largely be attributed to the additional image augmentations (show in Table 5.4b, +0.7 mAP). In addition, BEVFusion achieves 1.8× reduction in MACs and 1.5× reduction in latency, which we attribute to the choice of the fusion space (BEV *vs*. LiDAR). For BEV map segmentation, we break down the 13.6% improvement over PointPainting [263] in Table 5.2 into 1.4% in data augmentation (Table 5.4b), 3.5% in a better backbone (Table 5.4c) and **8.9%** on choosing BEV as the fusion space. This is because map segmentation is *semantic-oriented* and the BEVFusion is capable of fully preserve the semantic density of the input images, while point-level fusion methods [263, 264] only retain 5% camera features (Figure 5.1b). Besides the most prominent improvements from the choice of BEV space and data augmentation, we also observe that increasing the computation does not always translate to an improvement in accuracy. For example, in Table 5.4d, the largest FPN size (1/4) leads to the worst performance for both detectors and segmentation models; in Table 5.4e, the smallest voxel size (0.125m) results in the best segmentation performance. Finally, our results in Table 5.4f that we can further improve the performance of the detection variant of BEVFusion by scaling up the input resolution.

5.1.6 Discussion

We have proposed BEVFusion, an efficient and generic framework for multi-task multi-sensor 3D perception. BEVFusion unifies *dense* camera and *sparse* LiDAR features in a shared BEV space that fully preserves both geometric and semantic information. To achieve this, we accelerate the slow camera-to-BEV transformation by more than 40×. BEVFusion breaks the long-lasting common practice that point-level fusion is the golden choice for multi-sensor perception systems. BEVFusion achieves state-of-the-art performance on both 3D detection and BEV map segmentation tasks with 1.5-1.9× less computation and 1.3-1.6× measured speedup over existing solutions. We hope that BEVFusion can serve as a simple but powerful baseline to inspire future research on multi-task multi-sensor fusion.

Chapter 6

Accelerating Language Modeling for Natural Language Processing

6.1 Lite Transformer

Transformer has become ubiquitous in natural language processing (*e.g.*, machine translation, question answering); however, it requires enormous amount of computations to achieve high performance, which makes it not suitable for mobile applications that are tightly constrained by the hardware resources and battery. We present an efficient mobile NLP architecture, *Lite Transformer* to facilitate deploying mobile NLP applications on edge devices. The key primitive is the *Long-Short Range Attention* (LSRA), where one group of heads specializes in the **local** context modeling (by convolution) while another group specializes in the **long-distance** relationship modeling (by attention). Such specialization brings consistent improvement over the vanilla transformer on three well-established language tasks: machine translation, abstractive summarization, and language modeling. Under constrained resources (500M/100M MACs), Lite Transformer outperforms transformer on WMT'14 English-French by 1.2/1.7 BLEU, respectively. Lite Transformer reduces the computation of transformer base model by 2.5$\times$ with 0.3 BLEU score degradation. Combining with pruning and quantization, we further compressed the model size of Lite Transformer by **18.2**$\times$. For language modeling, Lite Transformer achieves 1.8 lower perplexity than the transformer at around 500M MACs. Notably, Lite Transformer outperforms the AutoML-based Evolved Transformer by 0.5 higher BLEU for the mobile NLP setting without the costly architecture search that requires more than 250 GPU **years**.

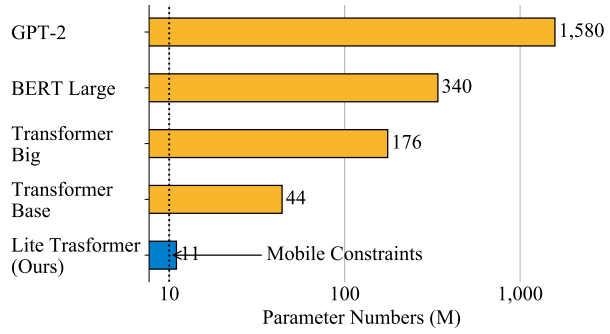
6.1.1 Introduction

Transformer [3] is widely used in natural language processing due to its high training efficiency and superior capability in capturing long-distance dependencies. Building on top of them, modern state-of-the-art models, such as BERT [4], are able to learn powerful language representations from unlabeled text and even surpass the human performance on the challenging question answering task.

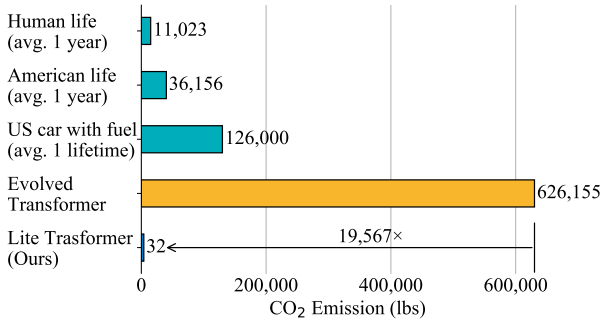
However, the good performance comes at a high computational cost. For example, a single transformer model requires more than 10G MACs in order to translate a sentence of only 30 words. Such extremely high computational resources requirement is beyond the capabilities of many edge devices such as smartphones and IoTs. Therefore, it is of great importance to design efficient and fast transformer architecture specialized for real-time NLP applications on the edge. Automatic neural architecture search [52, 317] is a choice for high accuracy model design, but the massive search cost (GPU hours and CO_2 emission) raises severe environmental concerns [195], shown in Figure 6.1b.

We focus on the efficient inference for mobile devices, where the total number of MACs is constrained below 500M. A straightforward way to reduce the computation of the transformer is to shrink the embedding size directly. Although it can effectively reduce both model size and computation, it also weakens the model capacity capturing the long and short distance relationship at the same time. To this end, we systematically studied the computation breakdown of the transformer and observed that the computation (MACs) is dominated by the feed-forward network (FFN). We discovered that the prevailing bottleneck-structured transformer block is not efficient. We then present a novel Long-Short Range Attention (LSRA) primitive. LSRA trades off the computation in FFN for wider attention layers. It stretches the bottleneck to introduce more dependency capturing capability for the attention layer, and then shrink the embedding size to reduce the total computation amount while maintaining the same performance. Instead of having one module for "general" information, LSRA dedicates *specialized* heads to model long and short distance contexts. Inspired by [318], LSRA introduces convolution in a parallel branch to capture **local** dependencies so that the attention branch can focus on **global** context capture. By stacking this primitive, we build Lite Transformer for mobile NLP applications.

Extensive experiments demonstrate that our Lite Transformer model offers significant improvements over the transformer on three language tasks: machine translation, abstractive summarization, and language modeling. For machine translation, on IWSLT 2014 German-English, it outperforms the transformer by 3.1 BLEU under 100M MACs; on WMT



(a) Parameter numbers of modern NLP models.



(b) The design cost measured in CO_2 emission (lbs).

Figure 6.1: Left: the size of recent NLP models grows rapidly and exceeds the mobile constraints to a large extent. Right: the search cost of AutoML-based NLP model is prohibitive, which emits carbon dioxide nearly 5× the average lifetime emissions of the car.

2014 English-German, it surpasses the transformer by 0.4 BLEU under 500M MACs and 1.2 BLEU under 100M MACs; on WMT 2014 English-French, it also achieves consistent improvements over the transformer: 1.2 BLEU under 500M MACs and 1.7 BLEU under 100M MACs. Further, combined with general model compression techniques (pruning and quantization), our Lite Transformer can achieve 18.2× model size compression. For the summarization task, on CNN-DailyMail, it reduces the computation of the transformer base model by 2.4×. For language modeling, it achieves 1.8 lower perplexity than the transformer around 500M MACs.

Guided by our design insights, our manually-designed Lite Transformer achieves 0.5 higher BLEU than the AutoML-based Evolved Transformer [317], which requires more than 250 GPU years to search, emitting as much carbon as five cars in their lifetimes (see Figure 6.1b). It indicates that AutoML is not a panacea: careful analysis and design insights (*i.e.*, removing the bottleneck, specialized heads) can effectively prune the search space and improve the sample efficiency.

6.1.2 Related Work

RNNs and CNNs. Recurrent neural networks (RNNs) have prevailed various sequence modeling tasks for a long time [319–322]. However, RNNs are not easy to parallelize across the sequence due to its temporal dependency. Recently, some work has demonstrated that RNN is not an essential component to achieve state-of-the-art performance. For instance, researchers have proposed highly-efficient convolution-based models [318, 323–325]. Convolution is an ideal primitive to model the local context information; however,

it lacks the ability to capture the long-distance relationship, which is critical in many sequence modeling tasks.

Transformers. As an alternative, attention is able to capture global-context information by pairwise correlation. Transformer [3] has demonstrated that it is possible to stack the self-attentions to achieve state-of-the-art performance. Recently, there have been a lot of variants to the transformer [326–333]. Among them, [327] proposed to scale up the batch size; [330] leverages the relative position representations; [326] introduces the weighted multi-head attention; [331] applies adaptive masks for long-range information on character-level language modeling with very long sequences. All these attempts are orthogonal to our work, as their methods can also be applied in our architecture.

Automated Model Design. Due to the vast architecture design space, automating the design with neural architecture search (NAS) becomes popular [52, 189, 201, 334]. To make the design efficient, integrating the hardware resource constraints into the optimization loop begins to emerge, such as MnasNet [191], ProxylessNAS [49] and FBNet [192]. In the NLP community, the evolved transformer [317] adopts the neural architecture search [52] to design basic blocks and finds a better #parameter-BLEU trade-off for the transformer. However, AutoML-based model designs require significant amount of GPU hours to find the 'best' model, which is not affordable for most researchers.

Model Acceleration. Apart from designing efficient models directly [22, 202], another approach to achieve efficient inference is to compress and accelerate the existing large models. For instance, some have proposed to prune the separate neurons [43, 54] or the entire channels [17, 55, 56]; others have proposed to quantize the network [18, 335–337] to accelerate the model inference. Recently, AutoML has also been used to automate the model compression and acceleration [17, 18, 199, 200]. All these techniques are compressing existing models and are therefore orthogonal to our approach. We aim to explore how to make use of the domain knowledge to design an efficient architecture from the beginning, rather than compressing an existing model.

6.1.3 Is Bottleneck Effective for 1-D Attention?

The attention mechanism has been widely used in various applications, including 1-D (language processing [3]), 2-D (image recognition), and 3-D (video recognition [338]). It computes pairwise dot-product between all the input elements to model both short-term and long-term relationships. Despite its effectiveness, the operation introduces

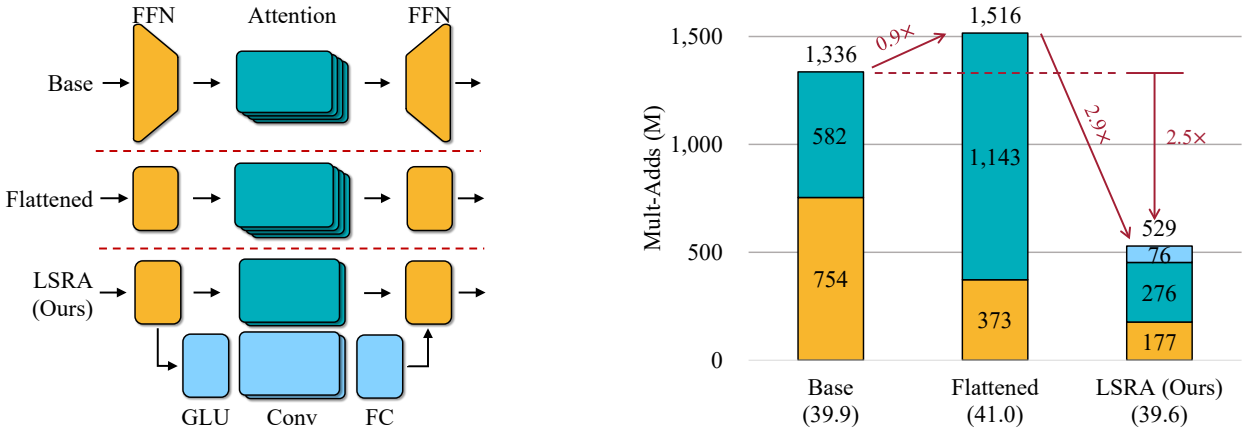
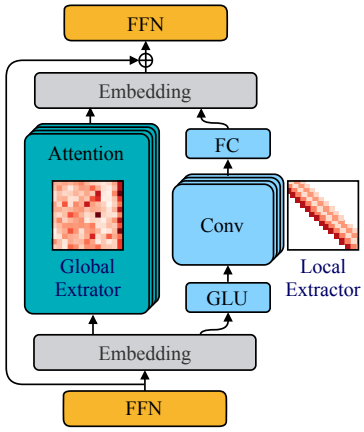


Figure 6.2: Flattening the bottleneck of transformer blocks increases the proportion of the attention versus the FFN, which is good for further optimization for attention in our LSRA.

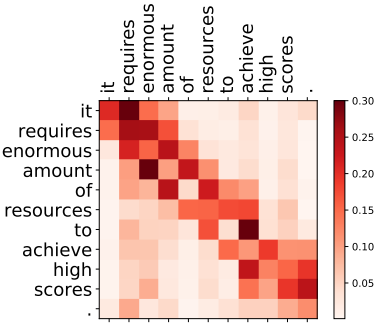
massive computation. Assume the number of elements (*e.g.*, length of tokens in language processing, number of pixels in image, *etc.*) fed to attention layer is N, and the dimension of features (*i.e.*, channels) is d, the computation needed for the dot-product is N^2d. For images and videos, N is usually very large. For example, the intermediate feature map in a video network [338] has 16 frames, each with 112×112 resolution, leading to $N = 2 \times 10^5$. The computation of convolution and fully-connected layers grows linearly w.r.t. N, while the computation of attention layers grows quadratically w.r.t. N. The computation of attention module will soon overwhelm with a large N.

To address the dilemma, a common practice is first to reduce the number of channels d using a linear projection layer before applying attention and increase the dimension afterwards (as shown in Figure 6.2). In the original design of transformer [3], the channel dimension in the attention module is 4× smaller than that in the FFN layer. Similarly, in the non-local video network [338], the channel number is first reduced by half before applying the non-local attention module. This practice saves the computation by 16× or 4×. Nevertheless, it also *decreases* the contexts capture ability of attention layers with a smaller feature dimension. The situation could be even worse for language processing, as attention is the major module for contexts capture (unlike images and videos where convolutions conduct the major information capture).

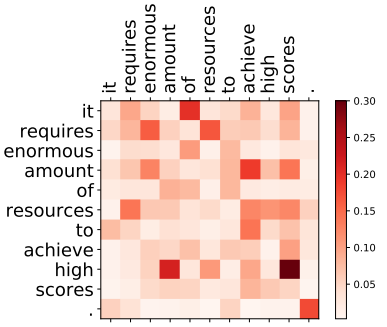
For tasks like translation, the length of the input sequence N tends to be small, which is around 20-30 in common cases. A transformer block consists of an attention (or two for decoder), followed by a feed-forward network (FFN). For the attention layer, the MACs would be $\mathcal{O}(4Nd^2 + N^2d)$; for FFN, the MACs is $\mathcal{O}(2 \times 4Nd^2)$. Given a small N, it is doubtful if the bottleneck design is a good trade-off between computation and accuracy on 1D attention. To verify the idea, we first profile the computation breakdown in the



(a) Lite Transformer block



(b) Conventional Attention. It captures local information on the diagonal and global context as sparse points. (Redundant)



(c) Attention in LSRA. It is specialized for long-term relationships, indicated as points away from the diagonal. (Efficient)

Figure 6.3: Lite Transformer architecture (a) and the visualization of attention weights. Conventional attention (b) puts too much emphasis on local relationship modeling (see the diagonal structure). We specialize the local feature extraction by a convolutional branch which efficiently models the locality so that the attention branch can specialize in global feature extraction (c).

transformer in Figure 6.2. Surprisingly, for the original transformer (denoted as 'Base' in the figure), the FFN layer actually consumes much of the computation. This is not desirable since FFN itself cannot perform any contexts captures. In conclusion, due to the small N, the bottleneck design cannot significantly reduce the computation in 1D attention, while the limited benefit for computation reduction is further compromised by the large FFN layer. It also harms the capacity of attention layer due to the smaller dimension, which is the major contexts capture unit in the transformer.

Therefore, we argue that the bottleneck design is not optimal for 1-D attention. We instead design a 'flattened' version of the transform block that does not reduce and increase the channel dimension. With the new design, the attention part now takes up the major computation in the flattened transformer model in Figure 6.2, leaving a larger space for further optimization. We also test the performance change of such modification on WMT'14 En-Fr dataset. We can achieve comparable performance at a slightly larger computation, which can be easily reduced with further optimization that is discussed in the next section.

6.1.4 Long-Short Range Attention (LSRA)

Researchers have tried to understand the contexts captured by attention. [339] and [340] visualized the attention weights from different layers in BERT. As shown in Figure 6.3b, the weights w illustrate the relationships between the words from the source sentence and

the target sentence (the same for self-attention). With a larger weight w_{ij} (darker color), the i-th word in the source sentence pays more attention to the j-th word in the target sentence. And the attention maps typically have strong patterns: sparse and diagonal. They represent the relationships between some particular words: the sparse for the long-term information, and the diagonal for the correlation in small neighborhoods. We denote the former as "global" relationships and the latter as "local".

For a translation task, the attention modules have to capture both global and local contexts, requiring a large capacity. That is not optimal compared with a specialized design. Taking the hardware design as an example, general-purpose hardware like CPUs is less efficient than specialized hardware like FPGAs. Here, we should specialize global and local contexts capture. When the model capacity is relatively large, the redundancy can be tolerated and may even provide better performance. However, when it comes to mobile applications, a model should be more efficient due to the computation and power constraints. Thus specialized contexts capture is more demanding. To tackle the problem, instead of having one module for "general" information, we propose a more specialized architecture, *Long-Short Range Attention (LSRA)*, that captures the global and local contexts separately.

As shown in Figure 6.3a, our LSRA module follows a two-branch design. The left branch captures global contexts, while the right branch models local contexts. Instead of feeding the whole input to both branches, we split it into two parts along the channel dimension, which will be mixed by the following FFN layer. Such practice reduces the overall computation by $2\times$. The left branch is a normal attention module as in [3], while the channel dimension is reduced by half. For the right branch of local relationships, one natural idea is to apply convolution over the sequence. With a sliding window, the diagonal groups can be easily covered by the module. To further reduce the computation, we replace the normal convolution with a lighter version [318] consisting of linear layers and depth-wise convolution. In this manner, we place the attention and the convolutional module side by side, encouraging them to have a different perspective of the sentence, globally and locally, so that the architecture can then benefit from the specialization and achieve better efficiency.

To have a better insight, we visualized the average attention weights of the same layer for a fully trained basic transformer and our Lite Transformer in Figure 6.3. It can be easily distinguished that instead of attempting to model both global and local contexts, the attention module in LSRA only focuses on the global contexts capture (no diagonal pattern), leaving the local contexts capture to the convolution branch.

6.1.5 Experimental Setup

Mobile Settings

Most of machine translation architectures benefit from the large model size and computational complexity. However, edge devices, such as mobile phones and IoTs, are highly computationally limited. Those massive architectures are no more suitable for real-world mobile applications. To formalize the problem, we define the mobile settings for NLP models in terms of the amount of computation and the parameter numbers:

- The floating-point performance of the ARM Cortex-A72 mobile CPU is about 48G FLOPS (4 cores @1.5GHz). To achieve the peak performance of 50 sentences per second, the model should be less than 960M FLOPs (480M MACs). That is a common constraint in the computer vision community. For example, Liu *et al*. [190] also uses 500M MACs as the constraint of its mobile setting. Therefore, we define the mobile settings for machine translation tasks: the computation constraint should be under **500M MACs** (or 1G FLOPs) with a sequence of 30 tokens (general length for machine translation).

- Additionally, we set a limitation for the parameters of the models. The constraint is based on the download and space limitation. Large mobile apps will take long time to be downloaded and even cost much money when using cellular networks. The run-time memory and disk size also constrain the parameter numbers. The parameters in MobileNet 7M parameters, we round it to the nearest magnitude, **10M parameters**, as our mobile constraint.

Datasets and Evaluation

Machine Translation. The results are based on three machine translation benchmarks: For IWSLT'14 German-English (De-En), we follow the settings in [341] with 160K training sentence pairs and 10K joint byte pair encoding (BPE) [342] vocabulary in lower case. For WMT English to German (En-De), we train the model on WMT'16 training data with 4.5M sentence pairs, validate on newstest2013, and test on newstest2014, the same as [318]. Moreover, the vocabulary used a 32K joint source and target BPE. For WMT English to Franch (En-Fr), we replicate the setup in [324] with 36M training sentence pairs from WMT'14, validate on newstest2012 and 2013, and test on newstest2014. Also, the 40K vocabulary is based on a joint source and target BPE factorization.

For evaluation, we use the same beam decoding configuration used by [3], where there is a beam size of 4 and a length penalty of 0.6. All BLEUs are calculated with case-sensitive

tokenization*, but for WMT En-De, we also use the compound splitting BLEU†, the same as [3]. When testing, we average the last 10 model checkpoints for IWSLT De-En and take the model with the lowest perplexity on the validation set for the WMT datasets. We omit the word embedding lookup table from the model parameters since the number of entries in the table would highly differ for various tasks using transformer. For the MACs, we calculate the total number of multiplication-addition pairs for a model translating a sequence with the length of 30 to a sequence with the same length, which is the average length for sentence-level machine translation tasks.

Abstractive Summarization. We also evaluate our Lite Transformer on CNN-DailyMail dataset [343] for abstractive summarization. The dataset contains 280K news articles paired with multi-sentence summaries. We truncate the articles to 1000 tokens and use a 30K BPE vocabulary. We use F1-Rouge as the metric, including Rouge-1 (R-1), Rouge-2 (R-2) and Rouge-L (R-L) [344]‡. We follow the generation settings in [345]. We omit the word embedding lookup table and softmax layer from both the model parameters and #MACs calculation. #MACs is calculated for the documents with the input length of 30, 100, and 1000 and the output length of 60 (the average tokens for the output of CNN-DailyMail dataset).

Language Modeling. We test our Lite Transformer for language modeling task on WIKITEXT-103, which comprises about 100M tokens and a 260K BPE vocabulary. We evaluate the perplexity on both the validation set and the training set. The model parameters and #MACs are also computed for the input with a length of 30, 100, and 1000.

Architecture

The model architecture is based on the sequence to sequence learning encoder-decoder [319]. For machine translation, our baseline model is based on the one proposed by [3] for WMT. For IWSLT, we follow the settings in [318]. We also adopt the same model as on WMT for summarization task. For language modeling, our model is in line with [346] but with smaller model dimension $d_{\text{model}} = 512$ and layer number $L = 12$ for the resource constraint. We use fairseq's reimplementation [347] of the transformer base model as the backbone.

*https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl
†https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/utils/get_ende_bleu.sh
‡https://github.com/pltrdy/files2rouge

	#Params	#MACs	BLEU	ΔBLEU
Transformer [3]	2.8M	63M	27.8	–
LightConv [318]	2.5M	52M	28.5	+0.7
Lite Transformer	2.8M	54M	**30.9**	**+3.1**
Transformer [3]	5.7M	139M	31.3	–
LightConv [318]	5.1M	115M	31.6	+0.3
Lite Transformer	5.4M	119M	**32.9**	**+1.6**
Transformer [3]	8.5M	215M	32.7	–
LightConv [318]	8.4M	204M	32.9	+0.2
Lite Transformer	8.9M	209M	**33.6**	**+0.9**

Table 6.1: Results on IWSLT'14 De-En. Lite Transformer outperforms the transformer [3] and the Lightweight convolution network [318] especially in mobile settings.

			WMT'14 En-De		WMT'14 En-Fr	
	#Params (M)	#MACs (M)	BLEU	ΔBLEU	BLEU	ΔBLEU
Transformer [3]	2.8	87	21.3	–	33.6	–
Lite Transformer	2.9	90	**22.5**	**+1.2**	**35.3**	**+1.7**
Transformer [3]	11.1	338	25.1	–	37.6	–
Lite Transformer	11.7	360	**25.6**	**+0.5**	**39.1**	**+1.5**
Transformer [3]	17.3	527	26.1	–	38.4	–
Lite Transformer	17.3	527	**26.5**	**+0.4**	**39.6**	**+1.2**

Table 6.2: Results on WMT'14 En-De and WMT'14 En-Fr. Lite Transformer improves the BLEU score over the transformer under similar MACs constraints.

In our architecture, we first flatten the bottleneck from the transformer base model and then replace the self-attention with the LSRA. More specifically, we use two specialized modules, an attention branch and a convolutional branch. Both the input and the output of the convolution are transformed by fully connected layers (GLU is applied for the input on WMT), and the kernel is dynamically calculated from the input using a fully connected layer in the WMT models. The kernel sizes are [3, 5, 7, 31$\times$3] for both the encoder and the decoder [318], and the number of heads for each module is 4 (half of the heads number in the transformer base model). The model for summarization is the same as the WMT model. For language modeling, the kernel sizes for the convolution branch are [15, 15, 31$\times$4, 63$\times$6].

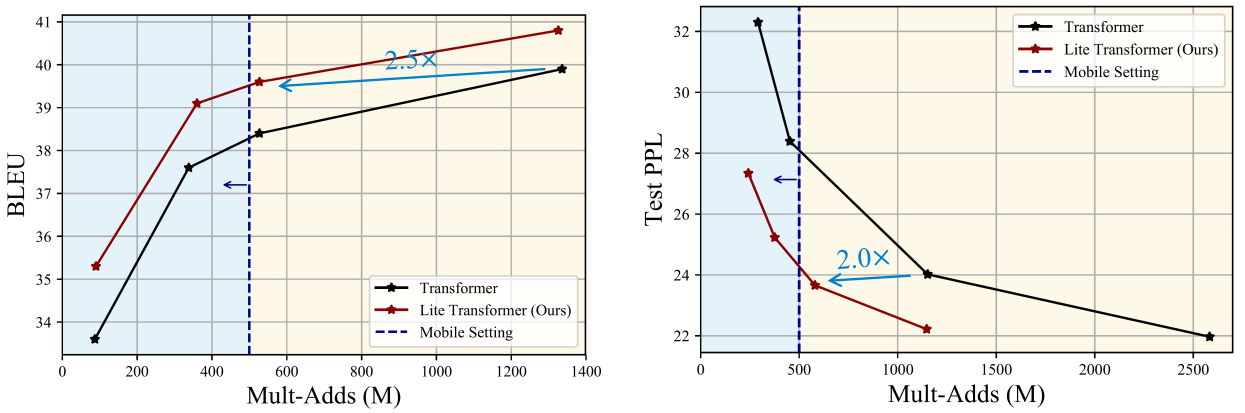
	#Params	#MACs	BLEU	GPU Hours	CO_2e (lbs)	Cloud Computation Cost
Transformer [3]	2.8M	87M	21.3	8×12	26	$68 - $227
Evolved Transformer [317]	3.0M	94M	22.0	8×274K	626K	$1.6M - $5.5M
Lite Transformer	2.9M	90M	**22.5**	8×14	32	$83 - $278
Transformer [3]	11.1M	338M	25.1	8×16	36	$93.9 - $315
Evolved Transformer [317]	11.8M	364M	25.4	8×274K	626K	$1.6M - $5.5M
Lite Transformer	11.7M	360M	**25.6**	8×19	43	$112 - $376

Table 6.3: Performance and training cost of an NMT model in terms of CO_2 emissions (lbs) and cloud compute cost (USD). The training cost estimation is adapted from [195]. The training time for transformer and our Lite Transformer is measured on NVIDIA V100 GPU. The cloud computing cost is priced by AWS (lower price: spot instance; higher price: on-demand instance).

Training Settings

All of our training settings for machine translation are in line with [318]. We use a dropout of 0.3 for both the WMT and IWSLT datasets and linearly scale down the dropout ratio when shrinking the dimension of the embeddings for the WMT datasets. Same as [318], we apply Adam optimizer and a cosine learning rate schedule [159, 348] for the WMT models, where the learning rate is first linearly warm up from 10^{-7} to 10^{-3} followed by a cosine annealing with a single cycle. For IWSLT De-En, we use inverse square root learning rate scheduling [3] with the linear warm-up. We use the same training settings for summarization. For the language modeling task, the training settings are in line with [346]. We decrease the dropout ratio for the FFN layer by half in our Lite Transformer due to the flattened layer.

We train WMT and summarization models on 16 NVIDIA RTX 2080Ti GPUs and IWSLT De-En on a single GPU for 50K steps. We also accumulate the gradients for 8 batches before each model update [327]. The gradients of IWSLT models are not accumulated. The maximum number of tokens in a batch is 4K for all the models. Label smooth of 0.1 is applied for the prior distribution over the vocabulary [349, 350]. For language modeling, we train the models on 24 GPUs for 286K steps, the same as the settings in [346].



(a) BLEU score *vs*. MACs (on WMT En-Fr)

(b) PPL *vs*. MACs (on WIKITEXT-103)

Figure 6.4: Trade-off curve for machine learning on WMT En-Fr and language modeling on WIKITEXT-103 dataset. Both curves illustrate that our Lite Transformer outperform the basic transformer under the mobile settings (blue region).

6.1.6 Results

Machine Translation

Results on IWSLT. We first report the results on IWSLT'14 De-En dataset. The baseline model is in line with [318], which provides the best results in the literature with 512 model dimension, 1024 FFN hidden dimension, and 4 heads for the attentions. Our Lite Transformer generally outperforms the transformer base under mobile constraints. With tighter computation limitations, our model achieves more significant improvement. That is because, when the dimension of the features decreases, it becomes much harder for the "general" attention to extract both the global and local features from the rather more compact information within the features. On the contrary, with the specialized LSRA, our model can capture the information from the features more efficiently.

In Table 6.1, we present the quantitative results of our Lite Transformer on IWSLT'14 De-En dataset, comparing to the transformer baseline as well as the LightConv [318]. Around 100M MACs, our model even achieves 1.6 BLEU score improvement than the transformer.

Results on WMT. We also show the result on the WMT'14 En-De and WMT'14 En-Fr dataset. Similar to the IWSLT, our Lite Transformer achieves a better trade-off with regard to transformer [3] against the total computation and the number of model parameters under mobile settings. The quantitative results in Table 6.2 indicates that our specialized Lite Transformer has 1.2 and 1.7 BLEU score improvement under 100M MACs and 0.5 and 1.5

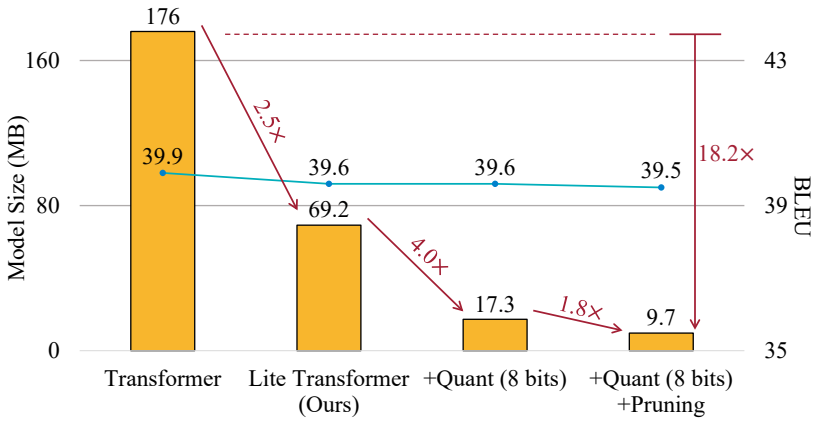


Figure 6.5: The model size and BLEU score on WMT En-Fr dataset with model compression. Our Lite Transformer can be combined with general compression techniques and achieves 18.2× model size compression. * 'Quant' indicates 'Quantization'.

around 300M MACs for WMT En-De dataset and WMT En-Fr dataset respectively. We also provide a tradeoff curve on WMT En-Fr in Figure 6.4a, where our Lite Transformer consistently outperforms the original transformer.

Amenable to Compression. As an efficient architecture, our Lite Transformer is orthogonal to general techniques for model compression (amenable to compression), *e.g.* pruning, and quantization. The results on WMT'14 En-Fr dataset with those techniques are shown in Figure 6.5. We quantize the model weight into 8 bits with K-means [43] and prune the model according to the sensitivity of each layer [54]. With the two model compression techniques, our method achieves 18.2× model size compression with negligible BLEU score degradation.

Comparison with Automated Design

Comparing with the AutoML-based Evolved Transformer (ET) [317], our Lite Transformer also shows a significant improvement in mobile settings. Moreover, within mobile settings, the Lite Transformer outperforms the ET by 0.5 and 0.2 BLEU scores under 100M and 300M MACs, respectively, as shown in Table 6.3. Our architecture design is different from ET's design: ET stacks attentions and convolutions sequentially, while our Lite Transformer puts them in parallel; also, ET does not flatten the FFN.

Though nowadays, neural architecture search has been proved to be very powerful for searching in a large design space, the huge cost, more than 626155 lbs CO_2 emissions and more than 250 GPU years, cannot be ignored. Instead, careful human design with intuitions for specific tasks can also be a great choice in practice to save a large number of

	#Params	#MACs (30)	#MACs (100)	#MACs (1000)	R-1	R-2	R-L
Transformer	44.1M	2.0G	3.6G	29.9G	41.4	18.9	38.3
Lite Transformer	**17.3M**	**0.8G**	**1.5G**	**12.5G**	41.3	18.8	38.3

Table 6.4: Results on CNN-DailyMail dataset for abstractive summarization. Lite Transformer achieves similar F1-Rouge (R-1, R-2 and R-L) to the transformer [3] with more than 2.4× less computation and 2.5× less model size. "#MACs (x)" indicates the #MACs required by the model with the input length of x.

	#Params	#MACs (100)	#MACs (1000)	Speed (tokens/s)	Valid ppl.	Test ppl.
Adaptive Inputs	37.8M	3.9G	50.3G	7.6K	23.2	24.0
Lite Transformer	37.2M	3.9G	48.7G	10.2K	**21.4**	**22.2**

Table 6.5: Results on WIKITEXT-103 dataset for language modeling. We apply our Lite Transformer architecture on transformer base model with adaptive inputs [346] and achieve 1.8 lower test perplexity under similar resource constraint.

resources for the earth.

Abstractive Summarization and Language Modeling

We also test our Lite Transformer on longer input. In Table 6.4, we report results on CNN-DailyMail dataset for abstractive summarization. Our model achieves a similar F1-Rouge score as the transformer base model but requires 2.4× less computation and 2.5× storage resources. In Table 6.5, we provides the results of our Lite Transformer on WIKITTEXT-103 for language modeling task, compared with the adaptive inputs [346] baseline. Under similar resource constraints, our Lite Transformer can achieve 3.9 and 1.8 lower perplexity on valid and test set, respectively. In Figure 6.4b, we show the tradeoff curve for our model and the baseline transformer model on WIKITEXT-103 between the test perplexity and the #Multi-Adds for input sentence with 30 tokens. It indicates that our Lite Transformer achieves consistent improvement over the original transformer, especially under mobile settings. Despite the translation tasks, the specialization design of LSRA is effective for larger scale language tasks.

6.1.7 Discussion

We introduced *Long-Short Range Attention* (LSRA), where some heads specialize in the **local** context modeling while the others specialize in the **long-distance** relationship modeling. Based on this primitive, we design *Lite Transformer* that is specialized for the

mobile setting (under 500M MACs) to facilitate the deployment on the edge devices. Our Lite Transformer demonstrates consistent improvement over the transformer on multiple language applications. It also surpasses the Evolved Transformer that requires costly architecture search under mobile settings.

Chapter 7

Conclusion

In this dissertation, we have discussed our research efforts to bridge the demand-supply gap for deep learning computing through the lens of sparsity. We have covered solutions across algorithm, system, and application stacks. At the algorithm level, we discussed methods to identify sparsity within dense input data and introduced new sparse primitives to efficiently process inputs with sparsity. At the system level, we developed libraries to optimize existing sparse primitives, effectively translating theoretical savings from sparsity into practical speedups on hardware. At the application level, we applied sparsity to accelerate a wide range of computation-intensive AI applications, including autonomous driving and language modeling.

Research Impact

My past research has resulted in a series of impactful publications in top-tier venues in machine learning (*NeurIPS*, *ICLR*), systems (*MLSys*, *MICRO*), computer vision (*CVPR*, *ICCV*, *ECCV*), robotics (*ICRA*, *IROS*), and natural language processing (*ACL*). My work has received over 8,700 citations on Google Scholar and more than 20,000 stars on GitHub. My research has been supported by Qualcomm Innovation Fellowship and MIT Ho-Ching and Han-Ching Fund Award. I have been recognized as a Rising Star in ML and Systems by MLCommons and a Rising Star in Data Science by UChicago and UCSD.

My research in automated model compression [17–19] has landed in many industry products, such as Microsoft NNI and Intel OpenVino. It has also been commercialized by OmniML (acquired by NVIDIA) and has generated real-world revenue. My research in efficient 3D perception [26] has been adopted by many autonomous driving companies, and is selected by NVIDIA as a reference design to its customers in the automotive industry.

Future Directions

From efficient inference to training. This dissertation primarily focuses on improving the deployment efficiency of deep learning workloads. However, the techniques discussed, such as sparsity and compression, are also relevant to the training phase. For example, in LongLoRA [351], we implemented sparsified attention during training, resulting in an over 2× speedup for long-context large language model fine-tuning. There is significant potential in optimizing training efficiency, which is especially crucial given the prohibitive training costs of large foundation models for academic researchers. To address this barrier, we must develop efficient training algorithms and build efficient training systems.

Emerging computation-intensive applications. This dissertation explores two main applications: autonomous driving and language models. Many other emerging applications are also highly computation-intensive. One example is large foundation models, which present significant computational challenges for both cloud-based services (such as those provided by OpenAI) and edge deployments on devices like laptops and mobile phones. We should investigate how we can optimize these emerging workloads through efficient machine learning techniques such as sparsity and compression. Another example is AI for science. Significant progress has been made in solving scientific problems, such as protein structure prediction and drug discovery, through deep learning. However, the substantial computational costs involved severely hinder their widespread adoption. Given that molecules and proteins are typically sparsely connected, I believe that sparse algorithms and systems could be highly effective in addressing these challenges.

Full- and cross-stack optimizations. Deep learning optimization should span the entire stack, from algorithm and system to hardware. This dissertation mainly focuses on the algorithm and system layers. Specialized hardware design offers an additional opportunity for acceleration. For example, inspired by TorchSparse, a specialized accelerator design [134] offers another 2× speedup upon our system. Also, co-optimization across stacks is essential as it enables us to strategically make necessary compromises and maximize the efficiency of deep learning workloads. In NHAS [352], we co-optimized the model and hardware architecture, achieving a 1.3× speedup compared to separate optimizations. I believe there is a huge opportunity for full-stack and cross-stack efficient deep learning.

Retrospective and Outlook

As a concluding remark, I would like to reflect on the progress made in the last decade. Ten years ago, deep learning-based voice recognition systems like DeepSpeech [5] were considered large models that required desktop GPUs to run. Today, we can run these voice recognition systems on smartwatches locally, without internet access. This remarkable advancement is the result of tremendous efforts from researchers working on algorithms, systems, hardware, and applications.

We are currently in a unique and exciting era, witnessing the emergence of powerful AI models that enable new applications, such as large language models, generative AI, and scientific discoveries. However, these models are large and expensive. Looking ahead to the next decade, I believe these powerful models will become so efficient that they will be available on any device and accessible to everyone.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[5] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep Speech: Scaling Up End-to-end Speech Recognition," 2014. arXiv: 1412.5567.

[6] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep Speech 2: End-to-End Speech Recognition in English and Mandarin," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

[7] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Zidek, A. W. R. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Simonyan, S. Crossan,

P. Kohli, D. T. Jones, D. Silver, K. Kavukcuoglu, and D. Hassabis, "Improved Protein Structure Prediction using Potentials from Deep Learning," *Nature*, 2020.

[8] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Highly Accurate Protein Structure Prediction with AlphaFold," *Nature*, 2021.

[9] OpenAI, "GPT-4 Technical Report," 2023. arXiv: 2303.08774.

[10] Google, "Gemini 1.5: Unlocking Multimodal Understanding Across Millions of Tokens of Context," 2024. arXiv: 2403.05530.

[11] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical Text-Conditional Image Generation with CLIP Latents," 2022. arXiv: 2204.06125.

[12] T. Brooks, B. Peebles, C. Holmes, W. DePue, Y. Guo, L. Jing, D. Schnurr, J. Taylor, T. Luhman, E. Luhman, C. Ng, R. Wang, and A. Ramesh, "Video Generation Models as World Simulators," 2024. [Online]. Available: https://openai.com/research/video-generation-models-as-world-simulators.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, 1989.

[14] M. Dehghani, J. Djolonga, B. Mustafa, P. Padlewski, J. Heek, J. Gilmer, A. Steiner, M. Caron, R. Geirhos, I. Alabdulmohsin, R. Jenatton, L. Beyer, M. Tschannen, A. Arnab, X. Wang, C. Riquelme, M. Minderer, J. Puigcerver, U. Evci, M. Kumar, S. van Steenkiste, G. F. Elsayed, A. Mahendran, F. Yu, A. Oliver, F. Huot, J. Bastings, M. P. Collier, A. Gritsenko, V. Birodkar, C. Vasconcelos, Y. Tay, T. Mensink, A. Kolesnikov, F. Pavetić, D. Tran, T. Kipf, M. Lučić, X. Zhai, D. Keysers, J. Harmsen, and N. Houlsby, "Scaling Vision Transformers to 22 Billion Parameters," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.

[15] OpenAI, "Improving Language Understanding by Generative Pre-Training," 2018.

[16] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[17] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[18] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-Aware Automated Quantization with Mixed Precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[19] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Hardware-Centric AutoML for Mixed-Precision Quantization," *International Journal of Computer Vision (IJCV)*, 2020.

[20] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "APQ: Joint Search for Network Architecture, Pruning and Quantization Policy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[21] X. Chen, Z. Liu, H. Tang, L. Yi, H. Zhao, and S. Han, "SparseViT: Revisiting Activation Sparsity for Efficient High-Resolution Vision Transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[22] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-Voxel CNN for Efficient 3D Deep Learning," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[23] Z. Liu, H. Tang, S. Zhao, K. Shao, and S. Han, "PVNAS: 3D Neural Architecture Search with Point-Voxel Convolution," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021.

[24] Z. Liu, X. Yang, H. Tang, S. Yang, and S. Han, "FlatFormer: Flattened Window Attention for Efficient Point Cloud Transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[25] H. Tang, Z. Liu, X. Li, Y. Lin, and S. Han, "TorchSparse: Efficient Point Cloud Inference Engine," in *Proceedings of the Conference on Machine Learning and Systems (MLSys)*, 2022.

[26] Z. Liu, H. Tang, A. Amini, X. Yang, H. Mao, D. Rus, and S. Han, "BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[27] Z. Wu, Z. Liu, J. Lin, Y. Lin, and S. Han, "Lite Transformer with Long-Short Range Attention," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[28] J. Huang, G. Huang, Z. Zhu, and D. Du, "BEVDet: High-Performance Multi-Camera 3D Object Detection in Bird-Eye-View," 2021. arXiv: 2112.11790.

[29] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[30] Z. Xu, F. Yu, C. Liu, Z. Wu, H. Wang, and X. Chen, "FalCon: Fine-grained Feature Map Sparsity Computing with Decomposed Convolutions for Inference Optimization," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2022.

[31] M. Ren, A. Pokrovsky, and R. Urtasun, "SBNet: Sparse Blocks Network for Fast Inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[32] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in Vision: A Survey," *ACM Computing Surveys*, 2022.

[33] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[34] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training Data-Efficient Image Transformers & Distillation through Attention," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[35] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z.-H. Jiang, F. E. Tay, J. Feng, and S. Yan, "Tokens-to-Token ViT: Training Vision Transformers From Scratch on ImageNet," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[36] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[37] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "PVTv2: Improved Baselines with Pyramid Vision Transformer," *Computational Visual Media (CVM)*, 2022.

[38] W. Wang, L. Yao, L. Chen, B. Lin, D. Cai, X. He, and W. Liu, "CrossFormer: A Versatile Vision Transformer Hinging on Cross-scale Attention," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[39] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo, "Swin Transformer V2: Scaling Up Capacity and Resolution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[40] Y. Li, H. Mao, Girshick, and K. He, "Exploring Plain Vision Transformer Backbones for Object Detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[41] S. Zheng, J. Lu, H. Zhao, X. Zhu, Z. Luo, Y. Wang, Y. Fu, J. Feng, T. Xiang, P. H. Torr, and L. Zhang, "Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[42] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[43] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[44] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and $<$ 0.5MB Model Size," 2016. arXiv: 1602.07360.

[45] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017. arXiv: 1704.04861.

[46] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[47] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[48] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[49] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[50] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for All: Train One Network and Specialize it for Efficient Deployment," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[51] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single Path One-Shot Neural Architecture Search with Uniform Sampling," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[52] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[53] H. Tang, Z. Liu, S. Zhao, Y. Lin, J. Lin, H. Wang, and S. Han, "Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[54] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both Weights and Connections for Efficient Neural Networks," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[55] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning Efficient Convolutional Networks through Network Slimming," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017.

[56] Y. He, X. Zhang, and J. Sun, "Channel Pruning for Accelerating Very Deep Neural Networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017.

[57] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[58] M. A. Raihan and T. Aamodt, "Sparse Weight Activation Training," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[59] L. Liu, L. Deng, X. Hu, M. Zhu, G. Li, Y. Ding, and Y. Xie, "Dynamic Sparse Graph for Efficient Deep Learning," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[60] J. Liu, Y. Chen, X. Ye, Z. Tian, X. Tan, and X. Qi, "Spatial Pruned Sparse Convolution for Efficient 3D Object Detection," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[61] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely Embedded Convolutional Detection," *Sensors*, 2018.

[62] S. Mehta and M. Rastegari, "MobileViT: Light-Weight, General-Purpose, and Mobile-Friendly Vision Transformer," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[63] Y. Chen, X. Dai, D. Chen, M. Liu, X. Dong, L. Yuan, and Z. Liu, "Mobile-former: Bridging Mobilenet and Transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[64] C. Gong, D. Wang, M. Li, X. Chen, Z. Yan, Y. Tian, Q. Liu, and V. Chandra, "NASViT: Neural Architecture Search for Efficient Vision Transformers with Gradient Conflict aware Supernet Training," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[65] H. Yin, A. Vahdat, J. Alvarez, A. Mallya, J. Kautz, and P. Molchanov, "AdaViT: Adaptive Tokens for Efficient Vision Transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[66] Z. Kong, P. Dong, X. Ma, X. Meng, W. Niu, M. Sun, B. Ren, M. Qin, H. Tang, and Y. Wang, "SPViT: Enabling Faster Vision Transformers via Soft Token Pruning," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[67] B. Pan, R. Panda, Y. Jiang, Z. Wang, R. Feris, and A. Oliva, "IA-RED^2: Interpretability-Aware Redundancy Reduction for Vision Transformers," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[68] Y. Rao, W. Zhao, B. Liu, J. Lu, J. Zhou, and C.-J. Hsieh, "DynamicViT: Efficient Vision Transformers with Dynamic Token Sparsification," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[69] H. Wang, Z. Zhang, and S. Han, "SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[70] S. Kim, S. Shen, D. Thorsley, A. Gholami, W. Kwon, J. Hassoun, and K. Keutzer, "Learned Token Pruning for Transformers," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2022.

[71] Y. Tang, K. Han, Y. Wang, C. Xu, J. Guo, C. Xu, and D. Tao, "Patch Slimming for Efficient Vision Transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[72] T. Chen, Y. Cheng, Z. Gan, L. Yuan, L. Zhang, and Z. Wang, "Chasing Sparsity in Vision Transformers: An End-to-end Exploration," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[73] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuScenes: A Multimodal Dataset for Autonomous Driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[74] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017.

[75] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[76] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

[77] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017.

[78] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[79] B. Cheng, A. G. Schwing, and A. Kirillov, "Per-Pixel Classification is Not All You Need for Semantic Segmentation," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[80] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[81] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking Atrous Convolution for Semantic Image Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[82] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, W. Liu, and B. Xiao, "Deep High-Resolution Representation Learning for Visual Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021.

[83] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar, "Masked-attention Mask Transformer for Universal Image Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[84] M.-H. Guo, C.-Z. Lu, Q. Hou, Z. Liu, M.-M. Cheng, and S.-M. Hu, "SegNeXt: Rethinking Convolutional Attention Design for Semantic Segmentation," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[85] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision (IJCV)*, 2010.

[86] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[87] I. Demir, K. Koperski, D. Lindenbaum, G. Pang, J. Huang, S. Basu, F. Hughes, D. Tuia, and R. Raskar, "DeepGlobe 2018: A Challenge to Parse the Earth through Satellite Images," in *CVPR Workshop*, 2018.

[88] N. C. Codella, D. Gutman, M. E. Celebi, B. Helba, M. A. Marchetti, S. W. Dusza, A. Kalloo, K. Liopyris, N. Mishra, H. Kittler, *et al.*, "Skin Lesion Analysis toward Melanoma Detection: A Challenge at the International Symposium on Biomedical Imaging (ISBI) 2016, hosted by the International Skin Imaging Collaboration (ISIC)," in *ISBI*, 2018.

[89] A. Kirillov, Y. Wu, K. He, and R. Girshick, "PointRend: Image Segmentation as Rendering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[90] Y.-H. Huang, M. Proesmans, S. Georgoulis, and L. Van Gool, "Uncertainty Based Model Selection for Fast Semantic Segmentation," in *Proceedings of the International Conference on Machine Vision and Applications (MVA)*, 2019.

[91] T. Verelst and T. Tuytelaars, "SegBlocks: Block-Based Dynamic Resolution Networks for Real-Time Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2022.

[92] T. Wu, Z. Lei, B. Lin, C. Li, Y. Qu, and Y. Xie, "Patch Proposal Network for Fast Semantic Segmentation of High-Resolution Images," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[93] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation.," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017.

[94] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2015.

[95] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid Scene Parsing Network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[96] L.-C. Chen, G. Papandreou, K. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[97] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Neural Networks for Visual Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2015.

[98] L.-C. Chen, G. Papandreou, K. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2016.

[99] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Deeplabv3: Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[100] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[101] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei, "Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[102] J. Fu, J. Liu, H. Tian, Y. Li, Y. Bao, Z. Fang, and H. Lu, "Dual Attention Network for Scene Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[103] Y. Yuan, X. Chen, and J. Wang, "Object-Contextual Representations for Semantic Segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[104] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation," 2016. arXiv: 1606.02147.

[105] H. Wu, J. Zhang, K. Huang, K. Liang, and Y. Yizhou, "FastFCN: Rethinking Dilated Convolution in the Backbone for Semantic Segmentation," 2019. arXiv: 1903.11816.

[106] R. P. Poudel, S. Liwicki, and R. Cipolla, "Fast-SCNN: Fast Semantic Segmentation Network," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2019.

[107] S. Mehta, M. Rastegari, A. Caspi, L. Shapiro, and H. Hajishirzi, "ESPNet: Efficient Spatial Pyramid of Dilated Convolutions for Semantic Segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[108] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[109] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, "BiSeNet V2: Bilateral Network with Guided Aggregation for Real-time Semantic Segmentation," *International Journal of Computer Vision (IJCV)*, 2021.

[110] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "ICNet for Real-Time Semantic Segmentation on High-Resolution Images," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[111] W. Chen, X. Gong, X. Liu, Q. Zhang, Y. Li, and Z. Wang, "Fasterseg: Searching for Faster Real-time Semantic Segmentation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[112] Y. Hong, H. Pan, W. Sun, and Y. Jia, "Deep Dual-resolution Networks for Real-time and Accurate Semantic Segmentation of Road Scenes," *IEEE Transactions on Intelligent Transportation Systems (T-ITS)*, 2021.

[113] H. Fan, B. Xiong, K. Mangalam, Y. Li, Z. Yan, J. Malik, and C. Feichtenhofer, "Multiscale Vision Transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[114] R. Strudel, R. Garcia, I. Laptev, and C. Schmid, "Segmenter: Transformer for Semantic Segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[115] Y. Yuan, R. Fu, L. Huang, W. Lin, C. Zhang, X. Chen, and J. Wang, "HRFormer: High-Resolution Transformer for Dense Prediction," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[116] H. Cao, Y. Wang, J. Chen, D. Jiang, X. Zhang, Q. Tian, and M. Wang, "Swin-Unet: Unet-like Pure Transformer for Medical Image Segmentation," in *Proceedings of the ECCV Workshop on Medical Computer Vision (MCV)*, 2022.

[117] H. Cai, C. Gan, and S. Han, "EfficientViT: Lightweight Multi-Scale Attention for On-Device Semantic Segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[118] B. Pan, W. Lin, X. Fang, C. Huang, B. Zhou, and C. Lu, "Recurrent Residual Module for Fast Inference in Videos," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[119] B. Pan, R. Panda, C. Fosco, C.-C. Lin, A. Andonian, Y. Meng, K. Saenko, A. Oliva, and R. Feris, "VA-RED^2: Video Adaptive Redundancy Reduction," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[120] B. Graham, M. Engelcke, and L. van der Maaten, "3D Semantic Segmentation With Submanifold Sparse Convolutional Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[121] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked Autoencoders Are Scalable Vision Learners," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[122] P. Gao, T. Ma, H. Li, J. Dai, and Y. Qiao, "ConvMAE: Masked Convolution Meets Masked Autoencoders," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[123] K. Tian, Y. Jiang, Q. Diao, C. Lin, L. Wang, and Z. Yuan, "Designing BERT for Convolutional Networks: Sparse and Hierarchical Masked Modeling," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[124] L. Huang, S. You, M. Zheng, F. Wang, C. Qian, and T. Yamasaki, "Green Hierarchical Vision Transformer for Masked Image Modeling," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[125] Z. Song, Y. Xu, Z. He, L. Jiang, N. Jing, and X. Liang, "CP-ViT: Cascade Vision Transformer Pruning via Progressive Sparsity Prediction," 2022. arXiv: 2203.04570.

[126] Y. Liang, C. Ge, Z. Tong, Y. Song, J. Wang, and P. Xie, "Not All Patches Are What You Need: Expediting Vision Transformers via Token Reorganizations," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[127] D. Bolya, C.-Y. Fu, X. Dai, P. Zhang, C. Feichtenhofer, and J. Hoffman, "Token Merging: Your ViT But Faster," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[128] D. Bolya and J. Hoffman, "Token Merging for Fast Stable Diffusion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[129] X. Ma, Y. Zhou, H. Wang, C. Qin, B. Sun, C. Liu, and Y. Fu, "Image as Set of Points," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[130] C. Choy, J. Gwak, and S. Savarese, "4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[131] H. Tang, S. Yang, Z. Liu, K. Hong, Z. Yu, X. Li, G. Dai, Y. Wang, and S. Han, "TorchSparse++: Efficient Training and Inference Framework for Sparse Convolution on GPUs," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023.

[132] K. Hong, Z. Yu, G. Dai, X. Yang, Y. Lian, Z. Liu, N. Xu, and Y. Wang, "Exploiting Hardware Utilization and Adaptive Dataflow for Efficient Sparse Convolution in 3D Point Clouds," in *Proceedings of the Conference on Machine Learning and Systems (MLSys)*, 2023.

[133] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "SpArch: Efficient Architecture for Sparse Matrix Multiplication," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2020.

[134] Y. Lin, Z. Zhang, H. Tang, H. Wang, and S. Han, "PointAcc: Efficient Point Cloud Accelerator," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021.

[135] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[136] Y. Wang, C. Zhang, Z. Xie, C. Guo, Y. Liu, and J. Leng, "Dual-side Sparse Tensor Core," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2021.

[137] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2000.

[138] Y. Boykov, O. Veksler, and R. Zabih, "Fast Approximate Energy Minimization via Graph Cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2001.

[139] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient Graph-based Image Segmentation," *International Journal of Computer Vision (IJCV)*, 2004.

[140] C. Rother, V. Kolmogorov, and A. Blake, "GrabCut – Interactive Foreground Extraction Using Iterated Graph Cuts," *ACM Transactions on Graphics (TOG)*, 2004.

[141] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

[142] A. Blake, C. Rother, M. Brown, P. Perez, and P. Torr, "Interactive Image Segmentation Using an Adaptive GMMRF Model," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2004.

[143] P. Krähenbühl and V. Koltun, "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2011.

[144] B. Wu, A. Wan, X. Yue, and K. Keutzer, "SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[145] G. Zhang, X. Lu, J. Tan, J. Li, Z. Zhang, Q. Li, and X. Hu, "RefineMask: Towards High-Quality Instance Segmentation with Fine-Grained Features," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[146] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[147] G. Lin, A. Milan, C. Shen, and I. Reid, "Refinenet: Multi-path Refinement Networks for High-resolution Semantic Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[148] A. Kirillov, R. Girshick, K. He, and P. Dollár, "Panoptic Feature Pyramid Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[149] Y. Chen, H. Fan, B. Xu, Z. Yan, Y. Kalantidis, M. Rohrbach, S. Yan, and J. Feng, "Drop an Octave: Reducing Spatial Redundancy in Convolutional Neural Networks with Octave Convolution," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[150] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. R. Acharya, V. Makarenkov, and S. Nahavandi, "A Review of Uncertainty Quantification in Deep Learning: Techniques, Applications and Challenges," *Information Fusion*, 2021.

[151] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[152] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "PointCNN: Convolution on $\mathcal{X}$-Transformed Points," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[153] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic Graph CNN for Learning on Point Clouds," *ACM Transactions on Graphics (TOG)*, 2019.

[154] L. Fan, Z. Pang, T. Zhang, Y.-X. Wang, H. Zhao, F. Wang, N. Wang, and Z. Zhang, "Embracing Single Stride 3D Object Detector with Sparse Transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[155] P. Sun, M. Tan, W. Wang, C. Liu, F. Xia, Z. Leng, and D. Anguelov, "SWFormer: Sparse Window Transformer for 3D Object Detection in Point Clouds," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[156] H. Wang, C. Shi, S. Shi, M. Lei, S. Wang, D. He, B. Schiele, and L. Wang, "DSVT: Dynamic Sparse Voxel Transformer with Rotated Sets," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[157] MMSegmentation Contributors, *MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark*, https://github.com/open-mmlab/mmsegmentation, 2020.

[158] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[159] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[160] NVIDIA, *cuBLAS*. [Online]. Available: https://developer.nvidia.com/cublas.

[161] G. Riegler, A. O. Ulusoy, and A. Geiger, "OctNet: Learning Deep 3D Representations at High Resolutions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[162] O. Cicek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation," in *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2016.

[163] R. Klokov and V. S. Lempitsky, "Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017.

[164] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A Deep Representation for Volumetric Shapes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[165] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," 2015. arXiv: 1512.03012.

[166] D. Maturana and S. Scherer, "VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[167] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and Multi-View CNNs for Object Classification on 3D Data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[168] Z. Wang and F. Lu, "VoxSegNet: Volumetric CNNs for Semantic Part Segmentation of 3D Shapes," *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2019.

[169] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[170] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis," *ACM Transactions on Graphics (TOG)*, 2017.

[171] M. Tatarchenko, J. Park, V. Koltun, and Q.-Y. Zhou, "Tangent Convolutions for Dense Prediction in 3D," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[172] T. Le and Y. Duan, "PointGrid: A Deep Network for 3D Shape Understanding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[173] S. Lan, R. Yu, G. Yu, and L. S. Davis, "Modeling Local Geometric Structure of 3D Point Clouds using Geo-CNN," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[174] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, "SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[175] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz, "SPLATNet: Sparse Lattice Networks for Point Cloud Processing," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[176] J. Li, B. M. Chen, and G. H. Lee, "SO-Net: Self-Organizing Network for Point Cloud Analysis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[177] L. P. Tchapmi, C. B. Choy, I. Armeni, J. Gwak, and S. Savarese, "SEGCloud: Semantic Segmentation of 3D Point Clouds," in *Proceedings of the International Conference on 3D Vision (3DV)*, 2017.

[178] W. Wang, R. Yu, Q. Huang, and U. Neumann, "SGPN: Similarity Group Proposal Network for 3D Point Cloud Instance Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[179] L. Landrieu and M. Simonovsky, "Large-Scale Point Cloud Semantic Segmentation With Superpoint Graphs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[180] S. Wang, S. Suo, W.-C. Ma, A. Pokrovsky, and R. Urtasun, "Deep Parametric Continuous Convolutional Neural Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[181] Q. Huang, W. Wang, and U. Neumann, "Recurrent Slice Networks for 3D Segmentation on Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[182] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D Object Detection from RGB-D Data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[183] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[184] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[185] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "Adaptive O-CNN: A Patch-based Deep Representation of 3D Shapes," in *SIGGRAPH Asia*, 2018.

[186] H. Lei, N. Akhtar, and A. Mian, "Octree Guided CNN With Spherical Kernels for 3D Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[187] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[188] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[189] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[190] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive Neural Architecture Search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[191] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[192] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "FBNet: Hardware-aware Efficient Convnet Design via Differentiable Neural Architecture Search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[193] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[194] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "HAT: Hardware-Aware Transformers for Efficient Natural Language Processing," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[195] E. Strubell, A. Ganesh, and A. McCallum, "Energy and Policy Considerations for Deep Learning in NLP," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[196] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[197] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, and J. Sun, "DetNAS: Backbone Search for Object Detection," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[198] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-Path NAS: Designing Hardware-Efficient ConvNets in Less Than 4 Hours," 2019. arXiv: 1904.02877.

[199] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[200] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun, "MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[201] H. Cai, J. Lin, Y. Lin, Z. Liu, K. Wang, T. Wang, L. Zhu, and S. Han, "AutoML for Architecting Efficient and Specialized Neural Networks," *IEEE Micro*, 2019.

[202] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, "GAN Compression: Efficient Architectures for Interactive Conditional GANs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[203] I. Radosavovic, J. Johnson, S. Xie, W.-Y. Lo, and P. Dollar, "On Network Design Spaces for Visual Recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[204] Z. Zhu, C. Liu, D. Yang, A. Yuille, and D. Xu, "V-NAS: Neural Architecture Search for Volumetric Medical Image Segmentation," in *Proceedings of the International Conference on 3D Vision (3DV)*, 2019.

[205] S. Kim, I. Kim, S. Lim, W. Baek, C. Kim, H. Cho, B. Yoon, and T. Kim, "Scalable Neural Architecture Search for 3D Medical Image Segmentation," in *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2019.

[206] D. Yang, H. Roth, Z. Xu, F. Milletari, L. Zhang, and D. Xu, "Searching Learning Strategy with Reinforcement Learning for 3D Medical Image Segmentation," in *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2019.

[207] W. Bae, S. Lee, Y. Lee, B. Park, M. Chung, and K.-H. Jung, "Resource Optimized Neural Architecture Search for 3D Medical Image Segmentation," in *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2019.

[208] K. C. Wong and M. Moradi, "SegNAS3D: Network Architecture Search with Derivative-Free Global Optimization for 3D Image Segmentation," in *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2019.

[209] Q. Yu, D. Yang, H. Roth, Y. Bai, Y. Zhang, A. Yuille, and D. Xu, "C2FNAS: Coarse-to-Fine Neural Architecture Search for 3D Medical Image Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[210] Z. Ma, Z. Zhou, Y. Liu, Y. Lei, and H. Yan, "Auto-ORVNet: Orientation-Boosted Volumetric Neural Architecture Search for 3D Shape Classification," *IEEE Access*, 2020.

[211] G. Li, G. Qian, I. C. Delgadillo, M. Muller, A. Thabet, and B. Ghanem, "SGAS: Sequential Greedy Architecture Search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[212] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

[213] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2013.

[214] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2019.

[215] W. Wu, Z. Qi, and L. Fuxin, "PointConv: Deep Convolutional Networks on 3D Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[216] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su, "PartNet: A Large-Scale Benchmark for Fine-Grained and Hierarchical Part-Level 3D Object Understanding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[217] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "3D Semantic Parsing of Large-Scale Indoor Spaces," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[218] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese, "Joint 2D-3D-Semantic Data for Indoor Scene Understanding," 2017. arXiv: 1702.01105.

[219] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[220] E.-T. Le, I. Kokkinos, and N. J. Mitra, "Going Deeper with Lean Point Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[221] G. Li, M. Müller, G. Qian, I. C. Delgadillo, A. Abualshour, A. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs Go as Deep as CNNs?" In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[222] G. Li, M. Xu, S. Giancola, A. Thabet, and B. Ghanem, "LC-NAS: Latency Constrained Neural Architecture Search for Point Cloud Networks," 2020. arXiv: 2008.10309.

[223] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "KPConv: Flexible and Deformable Convolution for Point Clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[224] C. Xu, B. Wu, Z. Wang, W. Zhan, P. Vajda, K. Keutzer, and M. Tomizuka, "SqueezeSegV3: Spatially-Adaptive Convolution for Efficient Point-Cloud Segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[225] I. Alonso, L. Riazuelo, L. Montesano, and A. C. Murillo, "3D-MiniNet: Learning a 2D Representation from Point Clouds for Fast and Efficient 3D LIDAR Semantic Segmentation," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[226] Y. Zhang, Z. Zhou, P. David, X. Yue, Z. Xi, B. Gong, and H. Foroosh, "PolarNet: An Improved Grid Representation for Online LiDAR Point Clouds Semantic Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[227] T. Cortinhal, G. Tzelepis, and E. E. Aksoy, "SalsaNext: Fast, Uncertainty-aware Semantic Segmentation of LiDAR Point Clouds for Autonomous Driving," 2020. arXiv: 2003.03653.

[228] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[229] T. Yin, X. Zhou, and P. Krähenbühl, "Center-Based 3D Object Detection and Tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[230] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, "PCT: Point Cloud Transformer," *Computational Visual Media (CVM)*, 2021.

[231] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point Transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[232] J. Mao, Y. Xue, M. Niu, H. Bai, J. Feng, X. Liang, H. Xu, and C. Xu, "Voxel Transformer for 3D Object Detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[233] G. Qian, H. A. A. K. Hammoud, G. Li, A. Thabet, and B. Ghanem, "ASSANet: An Anisotropical Separable Set Abstraction for Efficient Point Cloud Representation Learning," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[234] Y. Chen, J. Liu, X. Qi, X. Zhang, J. Sun, and J. Jia, "LargeKernel3D: Scaling up Kernels in 3D Sparse CNNs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[235] Z. Liu, A. Amini, S. Zhu, S. Karaman, S. Han, and D. Rus, "Efficient and Robust LiDAR-Based End-to-End Navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[236] J. Mao, X. Wang, and H. Li, "Interpolated Convolutional Networks for 3D Point Cloud Understanding," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[237] Q. Xu, Y. Zhou, W. Wang, C. R. Qi, and D. Anguelov, "Grid-gcn for fast and scalable point cloud learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[238] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[239] S. Shi, L. Jiang, J. Deng, Z. Wang, C. Guo, J. Shi, X. Wang, and H. Li, "PV-RCNN++: Point-Voxel Feature Set Abstraction With Local Vector Representation for 3D Object Detection," *International Journal of Computer Vision (IJCV)*, 2021.

[240] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in Perception for Autonomous Driving: Waymo Open Dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[241] Z. Wang and K. Jia, "Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal 3D Object Detection," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[242] A. H. Lang, S. Vora, H. Caesar, L. Zhou, and J. Yang, "PointPillars: Fast Encoders for Object Detection from Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[243] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu, "Class-Balanced Grouping and Sampling for Point Cloud 3D Object Detection," 2019. arXiv: 1908.09492.

[244] Z. Yang, Y. Sun, S. Liu, and J. Jia, "3DSSD: Point-Based 3D Single Stage Object Detector," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[245] Y. Zhou, P. Sun, Y. Zhang, D. Anguelov, J. Gao, T. Ouyang, J. Guo, J. Ngiam, and V. Vasudevan, "End-to-End Multi-View Fusion for 3D Object Detection in LiDAR Point Clouds," *Proceedings of the Conference on Robot Learning (CoRL)*, 2019.

[246] T. Yin, X. Zhou, and P. Krähenbühl, "CenterPoint++ Submission to the Waymo Real-time 3D Detection Challenge," Tech. Rep., 2022.

[247] R. Ge, Z. Ding, Y. Hu, W. Shao, L. Huang, K. Li, and Q. Liu, "1st Place Solutions to the Real-time 3D Detection and the Most Efficient Model of the Waymo Open Dataset Challenge 2021," in *CVPRW*, 2021.

[248] Y. Hu, Z. Ding, R. Ge, W. Shao, L. Huang, K. Li, and Q. Liu, "AFDetV2: Rethinking the Necessity of the Second Stage for Object Detection from Point Clouds," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

[249] Q. Chen, L. Sun, Z. Wang, K. Jia, and A. Yuille, "Object as Hotspots: An Anchor-Free 3D Object Detection Approach via Firing of Hotspots," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[250] C. R. Qi, Y. Zhou, M. Najibi, P. Sun, K. Vo, B. Deng, and D. Anguelov, "Offboard 3D Object Detection from Point Cloud Sequences," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[251] L. Fan, X. Xiong, F. Wang, N. Wang, and Z. Zhang, "RangeDet: In Defense of Range View for LiDAR-Based 3D Object Detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[252] Q. Chen, S. Vora, and O. Beijbom, "PolarStream: Streaming Lidar Object Detection and Segmentation with Polar Pillars," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[253] Y. Wang and J. M. Solomon, "Object DGCNN: 3D Object Detection using Dynamic Graphs," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[254] T. Guan, J. Wang, S. Lan, R. Chandra, Z. Wu, L. Davis, and D. Manocha, "M3DETR: Multi-Representation, Multi-Scale, Mutual-Relation 3D Object Detection With Transformers," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2022.

[255] G. Shi, R. Li, and C. Ma, "PillarNet: Real-Time and High-Performance Pillar-based 3D Object Detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[256] D. Ye, W. Chen, Z. Zhou, Y. Xie, Y. Wang, P. Wang, and H. Foroosh, "LidarMultiNet: Unifying LiDAR Semantic Segmentation, 3D Object Detection, and Panoptic Segmentation in a Single Multi-task Network," in *Proceedings of the CVPR Workshop on "Autonomous Driving" (WAD)*, 2022.

[257] Y. Chen, S. Liu, X. Shen, and J. Jia, "Fast Point R-CNN," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[258] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li, "From Points to Parts: 3D Object Detection from Point Cloud with Part-aware and Part-aggregation Network," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[259] Z. Li, F. Wang, and N. Wang, "LiDAR R-CNN: An Efficient and Universal 3D Object Detector," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[260] Z. Zhou, X. Zhao, Y. Wang, P. Wang, and H. Foroosh, "CenterFormer: Center-based Transformer for 3D Object Detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[261] L. Fan, F. Wang, N. Wang, and Z. Zhang, "Fully Sparse 3D Object Detection," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[262] X. Chen, S. Shi, B. Zhu, K. C. Cheung, H. Xu, and H. Li, "MPPNet: Multi-Frame Feature Intertwining with Proxy Points for 3D Temporal Object Detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[263] S. Vora, A. H. Lang, B. Helou, and O. Beijbom, "PointPainting: Sequential Fusion for 3D Object Detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[264] T. Yin, X. Zhou, and P. Krähenbühl, "Multimodal Virtual Point 3D Detection," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[265] Y. Li, A. W. Yu, T. Meng, B. Caine, J. Ngiam, D. Peng, J. Shen, B. Wu, Y. Lu, D. Zhou, Q. V. Le, A. Yuille, and M. Tan, "DeepFusion: Lidar-Camera Deep Fusion for Multi-Modal 3D Object Detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[266] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep Continuous Fusion for Multi-Sensor 3D Object Detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[267] X. Bai, Z. Hu, X. Zhu, Q. Huang, Y. Chen, H. Fu, and C.-L. Tai, "TransFusion: Robust LiDAR-Camera Fusion for 3D Object Detection with Transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[268] Y. Chen, Y. Li, X. Zhang, J. Sun, and J. Jia, "Focal Sparse Convolutional Networks for 3D Object Detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[269] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-View 3D Object Detection Network for Autonomous Driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[270] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017.

[271] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-End Object Detection with Transformers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[272] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable Transformers for End-to-End Object Detection," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[273] Z. Li, W. Wang, E. Xie, Z. Yu, A. Anandkumar, J. M. Alvarez, T. Lu, and P. Luo, "Panoptic SegFormer: Delving Deeper into Panoptic Segmentation with Transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[274] Y. Wang, V. Guizilini, T. Zhang, Y. Wang, H. Zhao, and J. M. Solomon, "DETR3D: 3D Object Detection from Multi-view Images via 3D-to-2D Queries," in *Proceedings of the Conference on Robot Learning (CoRL)*, 2021.

[275] Z. Li, W. Wang, H. Li, E. Xie, C. Sima, T. Lu, Y. Qiao, and J. Dai, "BEVFormer: Learning Bird's-Eye-View Representation from Multi-Camera Images via Spatiotemporal Transformers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[276] X. Yan, C. Zheng, Z. Li, S. Wang, and S. Cui, "PointASNL: Robust Point Clouds Processing Using Nonlocal Neural Networks with Adaptive Sampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[277] X. Wu, Y. Lao, L. Jiang, X. Liu, and H. Zhao, "Point Transformer V2: Grouped Vector Attention and Partition-based Pooling," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[278] C. Park, Y. Jeong, M. Cho, and J. Park, "Fast Point Transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[279] X. Pan, Z. Xia, S. Song, L. E. Li, and G. Huang, "3D Object Detection with Pointformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[280] C. He, R. Li, S. Li, and L. Zhang, "Voxel Set Transformer: A Set-to-Set Approach to 3D Object Detection from Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[281] X. Chen, T. Zhang, Y. Wang, Y. Wang, and H. Zhao, "FUTR3D: A Unified Sensor Fusion Framework for 3D Detection," 2022.

[282] Y. Li, Y. Chen, X. Qi, Z. Li, J. Sun, and J. Jia, "Unifying Voxel-based Representation with Transformer for 3D Object Detection," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[283] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers Are RNNs: Fast Autoregressive Transformers with Linear Attention," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[284] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[285] N. Corporation, *CUTLASS: CUDA Templates for Linear Algebra Subroutines*, 2022. [Online]. Available: https://github.com/NVIDIA/CUTLASS.

[286] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Lin, and Z. Zhang, "Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks," 2019. arXiv: 1909.01315.

[287] Y. Hu, Z. Ye, M. Wang, J. Yu, D. Zheng, M. Li, Z. Zhang, Z. Zhang, and Y. Wang, "FeatGraph: A Flexible and Efficient Backend for Graph Neural Network Systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

[288] C. Wang, C. Ma, M. Zhu, and X. Yang, "PointAugmenting: Cross-Modal Augmentation for 3D Object Detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[289] B. Pan, J. Sun, H. Y. T. Leung, A. Andonian, and B. Zhou, "Cross-View Semantic Segmentation for Sensing Surroundings," *IEEE Robotics and Automation Letters (RA-L)*, 2020.

[290] J. Philion and S. Fidler, "Lift, Splat, Shoot: Encoding Images From Arbitrary Camera Rigs by Implicitly Unprojecting to 3D," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[291] Q. Li, Y. Wang, Y. Wang, and H. Zhao, "HDMapNet: An Online HD Map Construction and Evaluation Framework," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

[292] B. Zhou and P. Krähenbühl, "Cross-View Transformers for Real-Time Map-View Semantic Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[293] X. Zhu, H. Zhou, T. Wang, F. Hong, Y. Ma, W. Li, H. Li, and D. Lin, "Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[294] T. Wang, X. Zhu, J. Pang, and D. Lin, "FCOS3D: Fully Convolutional One-Stage Monocular 3D Object Detection," in *Proceedings of the ICCV Workshop on "3D Object Detection from Images" (3DODI)*, 2021.

[295] Z. Tian, C. Shen, H. Chen, and T. He, "FCOS: Fully Convolutional One-Stage Object Detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[296] T. Wang, X. Zhu, J. Pang, and D. Lin, "Probabilistic and Geometric Depth: Detecting Objects in Perspective," in *Proceedings of the Conference on Robot Learning (CoRL)*, 2021.

[297] H. Chen, P. Wang, F. Wang, W. Tian, L. Xiong, and H. Li, "EPro-PnP: Generalized End-to-End Probabilistic Perspective-n-Points for Monocular Object Pose Estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[298] Y. Liu, T. Wang, X. Zhang, and J. Sun, "PETR: Position Embedding Transformation for Multi-View 3D Object Detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[299] Y. Wang, X. Zhang, T. Yang, and J. Sun, "Anchor DETR: Query Design for Transformer-Based Detector," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

[300] T. Roddick, A. Kendall, and R. Cipolla, "Orthographic Feature Transform for Monocular 3D Object Detection," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2019.

[301] T. Roddick and R. Cipolla, "Predicting Semantic Map Representations from Images using Pyramid Occupancy Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[302] E. Xie, Z. Yu, D. Zhou, J. Philion, A. Anandkumar, S. Fidler, P. Luo, and J. M. Alvarez, "M^2BEV: Multi-Camera Joint 3D Detection and Segmentation with Unified Birds-Eye View Representation," 2022. arXiv: 2204.05088.

[303] C. Reading, A. Harakeh, J. Chae, and S. L. Waslander, "Categorical Depth DistributionNetwork for Monocular 3D Object Detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[304] R. Nabati and H. Qi, "CenterFusion: Center-Based Radar and Camera Fusion for 3D Object Detection," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2021.

[305] S. Xu, D. Zhou, J. Fang, J. Yin, B. Zhou, and L. Zhang, "FusionPainting: Multimodal Fusion with Adaptive Attention for 3D Object Detection," in *Proceedings of the IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2021.

[306] Z. Chen, Z. Li, S. Zhang, L. Fang, Q. Jiang, F. Zhao, B. Zhou, and H. Zhao, "AutoAlign: Pixel-Instance Feature Aggregation for Multi-Modal 3D Object Detection," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2022.

[307] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[308] Z. Cai and N. Vasconcelos, "Cascade R-CNN: Delving into High Quality Object Detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[309] K. Sun, B. Xiao, D. Liu, and J. Wang, "Deep High-Resolution Representation Learning for Human Pose Estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[310] G. Gkioxari, R. Girshick, P. Dollár, and K. He, "Detecting and Recognizing Human-Object Interactions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[311] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun, "Multi-Task Multi-Sensor Fusion for 3D Object Detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[312] J. Hu, L. Shen, and G. Sun, "Squeeze-and-Excitation Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[313] Y. Cao, J. Xu, S. Lin, F. Wei, and H. Hu, "Global Context Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[314] X. Zhu, H. Hu, S. Lin, and J. Dai, "Deformable ConvNets V2: More Deformable, Better Results," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[315] D. Park, R. Ambrus, V. Guizilini, J. Li, and A. Gaidon, "Is Pseudo-Lidar Needed for Monocular 3D Object Detection?" In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[316] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[317] D. R. So, C. Liang, and Q. V. Le, "The Evolved Transformer," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[318] F. Wu, A. Fan, A. Baevski, Y. Dauphin, and M. Auli, "Pay Less Attention with Lightweight and Dynamic Convolutions," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[319] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2014.

[320] M.-T. Luong, H. Pham, and C. Manning, "Effective Approaches to Attention-Based Neural Machine Translation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

[321] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[322] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," 2016. arXiv: 1609.08144.

[323] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. van den Oord, A. Graves, and K. Kavukcuoglu, "Neural Machine Translation in Linear Time," 2016. arXiv: 1610.10099.

[324] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. Dauphin, "Convolutional Sequence to Sequence Learning," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[325] L. Kaiser, A. N. Gomez, and F. Chollet, "Depthwise Separable Convolutions for Neural Machine Translation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[326] K. Ahmed, N. S. Keskar, and R. Socher, "Weighted Transformer Network for Machine Translation," 2017. arXiv: 1711.02132.

[327] M. Ott, S. Edunov, D. Grangier, and M. Auli, "Scaling Neural Machine Translation," in *Proceedings of the Conference on Machine Translation (WMT)*, 2018.

[328] M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, M. Schuster, N. Shazeer, N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, Z. Chen, Y. Wu, and M. Hughes, "The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

[329] R. Paulus, C. Xiong, and R. Socher, "A Deep Reinforced Model for Abstractive Summarization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[330] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-Attention with Relative Position Representations," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

[331] S. Sukhbaatar, E. Grave, P. Bojanowski, and A. Joulin, "Adaptive Attention Span in Transformers," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[332] S. Sukhbaatar, E. Grave, G. Lample, H. Jegou, and A. Joulin, "Augmenting Self-attention with Persistent Memory," 2019. arXiv: 1907.01470.

[333] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating Long Sequences with Sparse Transformers," 2019. arXiv: 1904.10509.

[334] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient Neural Architecture Search via Parameter Sharing," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[335] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," 2016. arXiv: 1602.02830.

[336] C. Zhu, S. Han, H. Mao, and W. Dally, "Trained Ternary Quantization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[337] R. Krishnamoorthi, "Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper," 2018. arXiv: 1806.08342.

[338] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-Local Neural Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[339] O. Kovaleva, A. Romanov, A. Rogers, and A. Rumshisky, "Revealing the Dark Secrets of BERT," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[340] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What Does BERT Look At? An Analysis of BERT's Attention," in *Proceedings of the ACL Workshop on "BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP"*, 2020.

[341] E. Grave, A. Joulin, M. Cissé, and H. Jégou, "Efficient Softmax Approximation for GPUs," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[342] R. Sennrich, B. Haddow, and A. Birch, "Neural Machine Translation of Rare Words with Subword Units," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.

[343] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom, "Teaching Machines to Read and Comprehend," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[344] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004.

[345] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[346] A. Baevski and M. Auli, "Adaptive Input Representations for Neural Language Modeling," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[347] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A Fast, Extensible Toolkit for Sequence Modeling," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[348] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[349] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[350] G. Pereyra, G. Tucker, J. Chorowski, L. Kaiser, and G. Hinton, "Regularizing Neural Networks by Penalizing Confident Output Distributions," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[351] Y. Chen, S. Qian, H. Tang, X. Lai, Z. Liu, S. Han, and J. Jia, "LongLoRA: Efficient Fine-Tuning of Long-Context Large Language Models," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

[352] Y. Lin, D. Hafdi, K. Wang, Z. Liu, and S. Han, "Neural-Hardware Architecture Search," in *Proceedings of the NeurIPS Workshop on "ML for Systems"*, 2019.