# The Architectural Implications of Multi-modal Detection Models for Autonomous Driving Systems

Yunge Li
*Computer Science & Engineering*
*Oakland University*
Rochester Hills, USA
yungeli@oakland.edu

Shaibal Saha
*Computer Science & Engineering*
*Oakland University*
Rochester Hills, USA
shaibalsaha@oakland.edu

Lanyu Xu
*Computer Science & Engineering*
*Oakland University*
Rochester Hills, USA
lxu@oakland.edu

***Abstract*—Object detection plays a pivotal in autonomous driving by enabling the vehicles to perceive and comprehend their environment, thereby making informed decisions for safe navigation. Camera data provides rich visual context and object recognition, while LiDAR data offers precise distance measurements and 3D mapping. Multi-modal object detection models are gaining prominence in incorporating these data types, which is essential for the comprehensive perception and situational awareness needed in autonomous vehicles. Although graphics processing units (GPUs) and field-programmable gate arrays (FPGAs) are promising hardware options for this application, the complex knowledge required to efficiently adapt and optimize multi-modal detection models for FPGAs presents a significant barrier to their utilization on this versatile and efficient platform. In this work, we evaluate the performance of camera and LiDAR-based detection models on GPU and FPGA hardware, aiming to provide a specialized understanding for translating multi-modal detection models to suit the unique architecture of heterogeneous hardware platforms in autonomous driving systems. We focus on critical metrics from both system and model performance aspects. Based on our quantitative implications, we propose foundational insights and guidance for the design of camera and LiDAR-based multi-modal detection models on diverse hardware platforms.**

***Index Terms*—autonomous driving, multi-modality, heterogeneous hardware, object detection, GPU, FPGA**

## I. Introduction

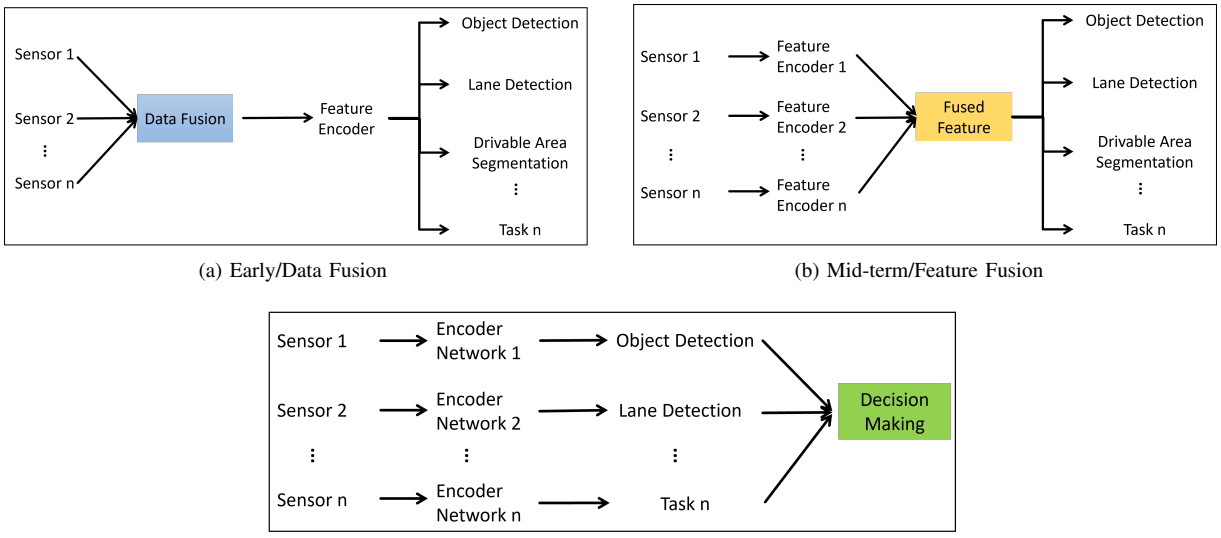
As autonomous driving technology continues to evolve, the complexity of its perception systems is steadily increasing. As a fundamental and critical task in autonomous driving, object detection has seen rapid progress. Initially focusing on 2D object detection, the applications of detection models have now expanded to 3D object detection. This evolution has not only enhanced the capabilities of autonomous driving but also broadened their potential applications.

In object detection for autonomous driving, models are typically categorized based on the modality of input data into two types: *camera-based* and *LiDAR-based* models. Camera-based models, primarily using multi-view images as input, estimate depth information through specific algorithms, combining it with image features to achieve 3D object detection. These models are rich in image information but may need more accuracy in depth estimation. On the other hand, LiDAR-based models address these depth estimation shortcomings, as point clouds inherently contain depth information. These models extract point cloud features to facilitate 3D object detection by employing specialized feature extraction networks for point clouds. However, the sparsity of object feature information in point clouds can impact the performance of these models.

To provide a more comprehensive and robust understanding of the environment, multi-modal object detection models have been proposed to integrate the rich visual context from cameras and the precise depth information from LiDAR. Multi-modal models show significant advantages over single-modal models in performance, efficiency, and energy consumption [1]. To integrate camera and LiDAR data for multi-modal models, there are three fusion methods that can be classified into *early*, *mid-term*, and *late fusion*. Early fusion, also known as data fusion (Figure 1a), refers to the fusion of data from different sensors in the early stage of a multi-modal model to facilitate subsequent feature extraction and task completion. In mid-term fusion, also known as feature fusion (Figure 1b), data from different sensors are processed by different encoders to form features, which are then fused to complete various tasks. Late fusion, also known as decision-level fusion (Figure 1c), is to synthesize these decisions through a fusion mechanism after different task networks obtain results or make decisions.

In the perception systems of autonomous driving, mid-term fusion methods, particularly the construction of Bird's Eye View (BEV) feature maps, are widely adopted. The construction of BEV feature maps, which is a typical example of mid-term or feature fusion, is the inspiration for this work. In this method, features from LiDAR and cameras are combined into a BEV feature map for enhanced object detection. Before integration, each modality undergoes an independent feature extraction process. To understand and analyze these multi-modal models with improved accuracy and granularity, we divide the processing into camera and LiDAR streams. Further, we deconstruct and evaluate two specific object detection models, one camera-based and the other LiDAR-based, to investigate the various modules within these two streams.

Graphics processing units (GPUs) are currently the dominant hardware used to process object detection for autonomous driving due to their high parallel processing power. Meanwhile, the demands for high performance, low latency, resource utilization, and low energy consumption in autonomous driving make field-programmable gate arrays (FPGAs) flexible




(a) Early/Data Fusion

(b) Mid-term/Feature Fusion

(c) Late/Decision-level Fusion

Fig. 1: Three fusion methods for camera and LiDAR data.

and efficient alternatives. However, the specialized and complex nature of FPGA technology presents a significant barrier to the utilization of FPGAs in autonomous driving for the following reasons. Firstly, providing tailored design to accelerate specific tasks on FPGAs requires in-depth knowledge of hardware description languages, which is a significant shift from the high-level programming languages used in AI model (such as detection models) development. Secondly, optimizing an AI model on FPGAs requires a deep understanding of both the model's computational needs and the FPGA's architecture. Thirdly, AI models are typically resource-intensive, while FPGAs have limited resources (e.g., logic blocks, memory). It is challenging to plan and utilize the limited resources for AI tasks carefully. Last but not least, the development tools and community support for FPGA applications are not as well as other hardware platforms like GPUs. This makes the utilization of FPGA in autonomous driving more complex and less accessible to those without specialized training. To overcome this barrier and achieve the optimal collaboration between diverse hardware platforms, it is essential to understand the performance characteristics of different modules of detection models on various platforms.

In this work, we deeply explore two perception models: camera-based BEVDet [2] and LiDAR-based PointPillars [3]. We focus on evaluating the performance of key components of these models, including *image encoder*, *BEV encoder with head*, *LiDAR encoder*, and *detection head*, across both GPU and FPGA platforms. Specifically, we focus on resource utilization, latency, power, and energy consumption to investigate the unique advantages of each platform and ensure optimal hardware allocation. The goal is to understand the strengths of two promising autonomous driving hardware, guide the integration of software and hardware in multi-modal object detection models, and enhance the performance and efficiency of autonomous driving systems.

The main contributions of this work can be summarized as the following three points:

- We analyze prevalent detection models and deconstruct them into several key modules as the primary focus of our evaluation across platforms. Specifically, we identify an image encoder and a BEV encoder with a detection head from the camera-based model. We identify a LiDAR encoder and a detection head from the LiDAR-based model. Such deconstruction can be applied to general detection models.
- We perform a comprehensive evaluation of the identified modules from detection models on both GPU and FPGA platforms, aiming to understand the strengths and limitations of each hardware. This analysis covers a range of system-specific metrics, including resource utilization, latency, power, and energy consumption.
- We present insights into the efficiency of different hardware platforms in processing multi-modality data for detection models in autonomous driving systems. Additionally, we discuss the potential of software and hardware co-design and explore the implications for the development of future heterogenous multi-modal perception models.

## II. Related Work

The perception system is an essential part of autonomous driving. It determines whether the autonomous vehicle fully understands the surrounding environment and can support subsequent system modules. Object detection is a critical task of the perception system. With the rise of machine learning, more and more object detection networks based on deep neural networks (DNN) are constantly being proposed.

### A. Object Detection Models

R-CNN [4], YOLO [5], and CenterNet [6], a series of single-modal models that take images as input, mainly focusing on solving 2D object detection tasks. Image encoders exist in these networks to obtain rich image features, but different networks have different processing methods. For example, R-CNN utilizes the Region Proposal Network (RPN) to generate region proposals and extract features for classification and regression. YOLO completes object classification and localization in one step, using predefined anchor boxes to generate bounding boxes. CenterNet uses keypoint detection to locate an object's center point and predict the object's size and offset.

More complex 3D object detection models can be divided into three types based on input data, namely based on LiDAR point cloud, based on multi-view image, and the fusion of the two, which is a multi-modal model. Models such as PointPillars [3] and CenterPoint [7] use LiDAR as a sensor to solve 3D object detection. Since the point cloud can provide depth information, this type of network can obtain point cloud features through the LiDAR backbone and then locate and classify the object. However, due to the sparsity of point cloud features, rich image information is often lacking. The model that uses multi-view images as input estimates depth information through multiple visions of the vehicle itself. A standard method is LSS [8], which estimates depth information for image features and fuses depth features and image features to build BEV features. BEVDet [2] uses such a method, but in the process of building BEV features, it optimizes the generation of BEV feature maps to improve efficiency. Although the model of multi-view images can estimate depth information, there is often geometric loss. The multi-modal perception model uses both cameras and LiDAR as sensors, which seamlessly integrates these two different data sources: LiDAR for depth and spatial information and RGB images for texture, color, and finer object details. This fusion mode ensures robust and comprehensive 3D object detection. BEVFusion [1] is a typical multi-modal perception model. The network takes multi-view images and LiDAR point cloud as input and sets different encoders for these two types of data. In the camera stream, BEVFusion uses a processing method similar to BEVDet for image feature extraction, depth information estimation, and BEV feature transformer. For the LiDAR stream, BEVFusion mainly extracts the LiDAR feature from the LiDAR encoder and flattens it so that it can be converted into a BEV feature and fused with the BEV feature of the camera. Moreover, BEVFusion can not only support 3D object detection but also solve BEV Segmentation. Investigating the architecture implications of these networks provides insights into the model deconstruction and analysis of different modules.

### B. Heterogeneous Hardware Design

The collaboration of heterogeneous hardware platforms is a trend that leverages the unique advantages of each platform to alleviate the limitations inherent in relying on a single type of hardware to perform complex computing tasks. For example, GPU is good at parallel processing of a large number of basic operations based on deep learning to accelerate calculations. Still, it comes with high energy consumption and a lack of flexibility. FPGAs offer flexibility and low energy consumption for data processing but do not perform as well as GPUs at low latency. The integration of FPGA and GPU in autonomous driving systems has attracted attention due to their complementary advantages. There are many examples of heterogeneous hardware platform computation. Murad Qasaimeh et al. [9] examined the energy efficiency of central processing units (CPUs), GPUs, and FPGAs in vision kernel implementations and explored the performance of hardware accelerators for embedded vision applications, specifically comparing multi-core CPUs, GPUs, and FPGA performance. Lin et al. [10] tried different combinations of CPUs, GPUs, FPGAs, and application-specific integrated circuits (ASICs) to perform tasks such as autonomous driving detection, tracking, and localization to observe the performance of different combinations in terms of energy consumption and latency. Hao et al. [11] utilized FPGA as a backup system, enabling it to temporarily assume control and manage various tasks when the primary computing platforms for autonomous driving, such as GPU and CPU, fail, thereby prevent potential hazardous incidents from happening. Another study [12] proposed a hardware/software co-design architecture for autonomous driving systems, where different hardware platforms are responsible for different task modules within the entire autonomous driving system. However, further details were not elaborated upon. These researches on heterogeneous platform computing offer valuable insights that inform our subsequent analysis and deployment of hardware platforms.

## III. Design constraints

Deconstructing the neural network models into modules for analysis in heterogeneous platforms yields multiple design constraints. In particular, constraints(e.g., latency, power, thermal) must be dealt with in the autonomous driving industry to get an accurate and safe driving experience. These constraints can significantly influence the deployed solution's performance, efficiency, and overall feasibility.

### A. Latency constraint

Latency is critical in designing and deploying deep learning models, especially on heterogeneous computing platforms. Real-time applications such as autonomous driving and augmented reality systems demand low-latency implementations. At the inference level, latency depends on the model's complexity and the computational capacity of the hardware. Higher degrees of complexity in neural networks with deep layers generally correspond to high latency experienced during inference tasks. Furthermore, there is potential for high latency in heterogeneous platforms where data transmission occurs between different types of processors (such as GPU or FPGA) [13]. Moreover, fetching data from memory also contributes to latency, especially if the data is not readily available in the cache or RAM [14]. In the context of BEV perception, the

swift and precise identification of objects, such as vehicles and pedestrians, from sensor inputs like LiDAR and cameras is crucial for safe autonomous navigation with low latency. So, It is essential to consider the latency efficiency during the deconstruction of the models and analyze the latency of respective modules in respective platforms.

### *B. Power constraint*

Power consumption is another significant constraint when deploying deep learning models on heterogeneous computing platforms. High power consumption has a broader environmental impact, making energy efficiency a priority for sustainable computing practices. Generally, GPUs consume more power than FPGAs, especially in applications where the computational workload can be optimized to take benefit of the FPGA's reconfigurable architecture. The required power generally increased due to frequent data sharing between processor and memory [14]. Additionally, higher power consumption in GPUs translates to more heat generation, which causes more thermal issues compared to FPGA. These thermal issues can have overall effects on the autonomous driving experience. In a nutshell, offloading the modules into FPGA can reduce significant power usage. So, power is a significant constraint but a complex procedure to ensure minimum power consumption with maximum accuracy during the deconstruction of the modules in heterogeneous platforms.

### *C. Memory constraint*

Each processor type has its memory limitations. For instance, FPGAs typically have less onboard memory compared to GPUs. The memory affects how large a model is or how much data can be processed simultaneously. The different memory sizes in different hardware are the biggest challenge to fitting the models without losing accuracy. For example, FPGAs typically have limited on-chip memory (such as Block RAM or BRAM). This memory is fast and efficient but needs to be more significant to store large models or datasets. So, maximizing the use of on-chip memory for critical data and using memory swapping can significantly enhance performance. There is some ongoing research focused on memory issues during the heterogeneous platforms. One of the recent studies by Miao and Lin [15] proposed a framework where they utilized memory-swapping techniques to avoid memory overhead issues. In the BEV perception, datasets like nuScenes, Kitti, and Waymo need memory to process in the heterogeneous platforms. Power consumption is also rational, along with the excess use of memory. So, it is challenging to design the hardware to optimize memory usage, especially in the FPGA, where memory is limited.

### *D. Development tools constraint*

Software and tooling constraints in deploying deep learning models on heterogeneous platforms surround various issues, from development tool availability and compatibility to framework and library support challenges across different processors. Tools are needed to keep the diverse architectures in a heterogeneous system, including compilers and development environments. The lack of a unified IDE that supports all components of a heterogeneous system can complicate the development process. Developers need multiple tools for different system parts, increasing complexity. For instance, we need to use VITIS-AI 3.0 [16] as a software platform to perform model compression and xmodel generation for FPGA deployment, which requires multiple platforms and knowledge. Additionally, different FPGA providers have different hardware platform software (e.g., Quartus for Intel-designed FPGA [17], whereas Vivado for AMD Xilinx-designed FPGA [18]). Profiling tools must provide insights into the performance of each system component, but there are voids in FPGA compared to GPU. The need for sophisticated debugging and profiling tools for specific processors, like FPGAs, can restrict development. For example, customizing the modules individually in the complex models in devices like FPGA requires knowledge of both hardware and software for each provider. So, knowledge about software and hardware tools for individual providers usually restricts the successful analysis of the complex models into FPGA.

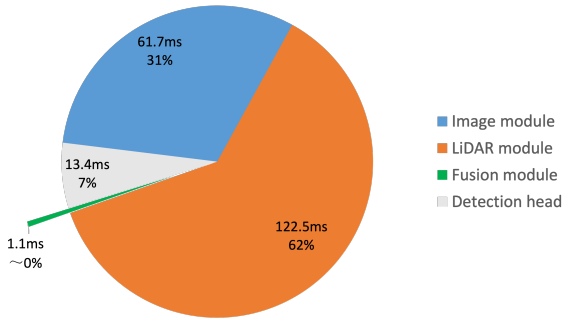



Fig. 2: Execution time of different modules.

## IV. Model Deconstruction and Deployment

In this section, we first deconstruct the object detection models into discrete modules and then systematically allocate these modules across diverse hardware platforms, focusing on FPGA and GPU in this work.

### *A. Model Deconstruction*

In the construction of multi-modal object detection networks, we refer to the typical model BEVFusion to obtain the architecture in Figure 3. This architecture acquires and processes data from two sensors, a camera and LiDAR. Camera features and LiDAR features are fused in the BEV space to obtain BEV features, which are then connected to different task head modules to complete the perception task. We analyze the execution time of the image module, LiDAR module, fusion module, and detection head through the evaluation of BEVFusion. The evaluation results in Figure 2 show that the computational bottleneck of the multi-modal model is actually the image module, LiDAR module, and detection head, while

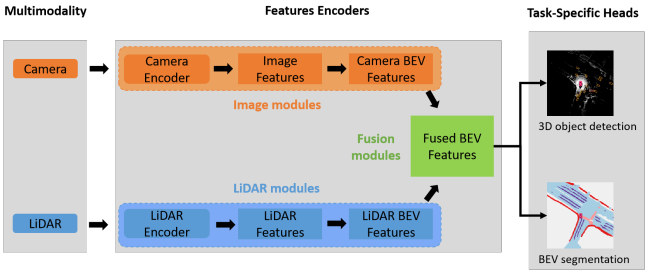



Fig. 3: The architecture of BEVFusion [1].

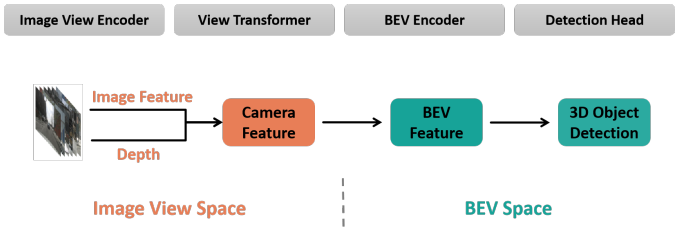



Fig. 4: The architecture of BEVDet [2].

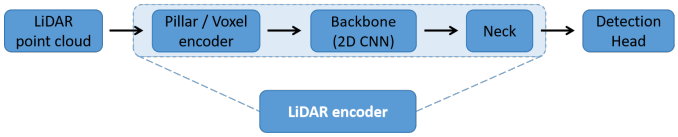



Fig. 5: The architecture of PointPillars [3].

the impact of fusion modules is minimal. Overall, we can divide the model into two processing streams: the camera and LiDAR streams. This division makes our analysis and evaluation more specific.

In order to analyze the camera stream, we select the BEVDet model for discussion. It is similar to the BEVFusion model in processing the camera stream. When processing RGB image data, the pre-processing link is indispensable. Typical pre-processing steps, such as image normalization, play a key role in improving the model's training efficiency, performance, and stability. Regarding feature extraction, popular CNN architectures include ResNet [19], EfficientNet [20] , and VGG16 [21], while Transformer-based models such as ViT [22] and Swin-transformer [23] are also receiving increasing attention. BEVDet emphasizes the feature extraction capabilities of ResNet and Swin-Transformer models. ResNet [19] has been used as the backbone to meet the deployment requirements of FPGA. Additionally, FPN [24] as the neck is crucial in effectively enhancing the image features. In the conversion of camera features to BEV features, it is essential to obtain the depth information. Both BEVDet and BEVFusion use the LSS [8] method to convert depth prediction from a regression problem to a classification problem and predict the depth of each image feature. It is noticeable that these two models have optimized the efficiency of LSS by introducing the bevpooling technology and significantly accelerating the speed of depth prediction. When generating BEV features, image and depth features are fused and mapped onto the BEV feature map using both intrinsic and extrinsic of the camera.

The architecture of BEVDet, as shown in Figure 4, divides the entire model into two spaces, which are image space and BEV space. Image-space includes the image backbone, neck, and view transformer; BEV space includes the BEV encoder and detection head. According to this architecture, in order to be consistent with the model structure of FPGA, we use the image backbone and neck as one module and the remaining view transformer, BEV encoder, and head as another module. When working with LiDAR streams, PointNet [25], VoxelNet [26], and PointPillars [3] are the most commonly used networks to extract point cloud features. This study focuses on the PointPillars network as it is specifically optimized for 3D object detection, and its architecture is represented in Figure 5. We adopt the PointPillars configuration in openmmlab [27], which uses hardVFE and PointPillarsScatter as voxel encoders, with SECOND [28] as the backbone. FPN is also used as the neck structure to enhance features. It uses a 3D object detection method based on two-stage anchors for detection heads. Since point cloud data has 3D information, it does not require depth prediction processing, thus making the processing of LiDAR data streams more straightforward. We deconstruct these components into two main modules: the LiDAR encoder module integrating the pillar or voxel encoder, backbone, and neck, and the subsequent detection head module. This helps us better understand and analyze LiDAR stream processing in multi-modal detection networks.

In this study, we explore BEVFusion [1], a typical multi-modal perception model in autonomous driving, and divide the camera flow and LiDAR flow according to its architecture. Next, for these two data streams, we select two representative models, BEVDet and PointPillars, as research objects to evaluate the multi-modal model's critical modules in detail. Through this analysis, we identify four core modules as the focus of evaluation: image encoder, BEV encoder with detection head, LiDAR encoder, and LiDAR-based detection head. These modules are comprehensively evaluated on heterogeneous hardware platforms to better understand their performance and optimization space in multi-modal perception systems.

### B. Deployment of Deconstruction Modules

*1) GPU Implementation:* For the GPU implementation, we leverage optimized machine-learning software libraries. We implement BEVDet and PointPillars, both using the Pytorch library. In this implementation, we only focus on pre-trained models provided by BEVDet official GitHub [2] for BEVDet and use pre-trained models from openmmlab detection library [27] for our inference tasks. We deconstruct all the modules for both models and calculate the power usage using NVML [29] libraries. Additionally, we calculate each module's latency and memory usage for the nuScenes mini dataset [30], which is elaborately discussed in the section V.

*2) FPGA Implementation:* Implementing a neural network on an FPGA involves steps and considerations to ensure

TABLE I: Specification of accelerating platforms.

| Specification | Zynq UltraScale+ MPSoC ZCU104 | Nvidia GeForce RTX 3080 |
|---|---|---|
| **CPU/GPU Cores** | Quad-core ARM Cortex-A53 & Dual-core ARM Cortex-R5 | 8,704 CUDA Cores |
| **Frequency** | Cortex-A53: Up to 1.5 GHz Cortex-R5: Up to 600 MHz | 1.44 GHz Base, 1.71 GHz Boost Clock |
| **Memory** | 4 GB DDR4 SDRAM | 10 GB GDDR6X |
| **Memory Bandwidth** | Depends on DDR4 configuration tens of GB/s | 760 GB/s |
| **FPGA Fabric (if any)** | Integrated FPGA fabric for custom hardware acceleration | Not applicable |
| **Additional Features** | Flexible I/O, RF-ADC/DAC support, FPGA programmability | Ray Tracing Cores, Tensor Cores, DLSS, PCIe 4.0 |

efficient and effective deployment. Neural networks typically use floating-point arithmetic, which is resource-intensive on FPGAs. Quantization involves converting these floating-point numbers into fixed-point numbers, which are more FPGA-friendly. We utilize the Vitis-AI 3.0 [16] to apply the model compression and set up the environment. We use the pre-built data processing units (DPU) image downloaded from the official Xilnix website [31] and Petalinux version 22.2 as the operating system in FPGA. As our goal is to analyze the module's performance in heterogeneous platforms, we divide both models into two subgraphs to document the experiment result in the FPGA. Subgraphs in FPGA deployment mean deconstructing specific modules to run on the DPU. To import the algorithmic components to FPGA, we focus on the image encoder and BEV Encoder with detection head modules for BEVDet, LiDAR encoder, and LiDAR-based detection head for PointPillars. As a result, in our FPGA deployment, we need to divide the graph into two subgraphs, where each subgraph utilizes one DPU to perform the inference task. We transfer the compressed model into FPGA and use the existing hardware design provided by Xilinx to deploy the model. The deployment Later, we utilize the Vaitrace library [32] from Vitis-AI to document the results in section V. To calculate the power consumption, we impose a delay of 10ms at the start of every module so that FPGA can cool down to the idle state.

## V. Evaluation

In this section, we delve into the implementation specifics and provide an extensive result analysis of the four core modules of BEVDet and PointPillars, focusing on hardware analysis. To investigate the architectural implications of BEVDet and PointPillars, the result analysis section considers resource utilization, latency, and power/energy consumption to substantiate the excellent trade-off between FPGA and GPU.

### *A. Implementation*

In this evaluation, we use two heterogeneous hardware platforms: Nvidia RTX 3080 GPU and Xilinx Zynq UltraScale+ MPSoC ZCU104 FPGA. Their detailed configurations are shown in Table I. Although the GPU runs cooperatively with the CPU, the CPU only undertakes the work of reading and writing data and does not directly participate in the runtime of the four modules, so the performance of the CPU is not considered in this study. We select nuScenes [30] and the KITTI dataset [33], famous datasets for multi-modal perception tasks in autonomous driving. The nuScenes dataset has 1000 scenes and can provide image data from 6 different perspectives and corresponding LiDAR and Radar data. We use the mini nuScenes dataset, which contains only one percent of the scenes of the full dataset but is sufficient for our evaluation. The KITTI dataset provides front-view images consisting of 7481 samples, split into 7481 training data and 7518 test images, as well as the corresponding point clouds. We use 3769 samples from the test samples to evaluate the model. The original model is deployed on the GPU, but only the quantized model can be deployed on FPGA due to resource limitations. In order to achieve fair evaluation, the same quantized (INT8) model is also deployed on GPU.

TABLE II: Performance metrics for BEVDet and PointPillars models on GPU With and Without Quantization. The table presents the mAP and NDS metrics for BEVDet on the nuScenes dataset, along with mAP (BEV) and mAP (3D) for PointPillars on the KITTI dataset. Performance degradation is observed for both models when quantization is applied.

| | **BEVDet** | | **PointPillars** | |
|---|---|---|---|---|
| | **mAP** | **NDS** | **mAP (BEV)** | **mAP (3D)** |
| **GPU (w/o quantization)** | 0.34 | 0.34 | 70.90 | 65.25 |
| **GPU (w/ quantization)** | 0.31 | 0.31 | 66.58 | 57.94 |

### *B. Results Analysis*

We evaluate critical metrics on GPU and FPGA, including resource utilization, latency, and power/energy consumption of the four modules of the two 3D object detection models (BEVDet and PointPillars). Although our primary interest lies in the efficiency of these multi-modal perception models, we measure the mean Average Precision (mAP) and nuScenes detection score (NDS) for BEVDet on nuScenes datasets and mAP for PointPillars on KITTI datasets. Table II illustrates the accuracy for quantized and without quantized models for both models on GPU. The mAP and NDS of the quantized BEVDet on the nuScenes dataset have both decreased, and the mAP of the quantized PointPillars on the KITTI dataset has also significantly decreased on BEV detection and 3D detection. Later, we deploy the quantized model to analyze latency, resource utilization, and power usage on FPGA.

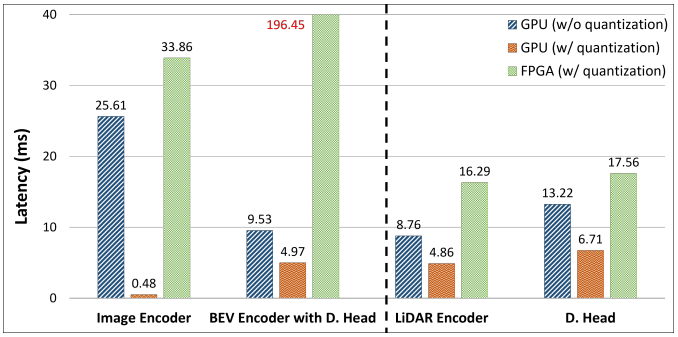



Fig. 6: Latency Result. 196.48 in red font indicates that the latency of BEV encoder with detection head on FPGA is much greater than other latency results. D. head means detection head.

*1) Resource Utilization:* For GPUs, we measure how much memory each module of BEVDet and PointPillars uses. The GPU we use has a max of 10 GB (10240 MB) memory (see details in Table I) and is utilized to calculate the memory usage percentage for each of the four modules. In BEVDet on GPU, the image encoder utilizes 192.19 MB memory, and the BEV Encoder with detection head requires 204.67 MB memory to execute the operations. The LiDAR encoder is less memory-intensive, using 18.75 MB, with the LiDAR-based detection head needing 186.41 MB memory.

However, to calculate the FPGA performance, we document the resource utilization parameters such as Look-Up Tables (LUT), Digital Signal Processors (DSP), Registers, and Ultra RAM (URAM) for BEVDet and PointPillars. Both models utilize approximately 93% of the available resources for the Xilinx ultrascale+ ZCU104 mpsoc device (details in Table III). Table III also illustrates performance and memory bandwidth consumption for BEVDet and PointPillars. As we set out the DPU core for FPGA board sets to 2, we call it DPU1 and DPU2. The image encoder runs on DPU1, and the BEV encoder with detection head runs on DPU2 for BEVDet, whereas the LiDAR encoder runs on DPU1, and the LiDAR-based detection head runs on DPU2 in PointPillars. The reason for consuming high memory bandwidth in BEVDet is that the image encoder often requires substantial memory bandwidth due to the need to transfer immense amounts of image data and intermediate results between memory and the processing units. From PointPillar's point of view, LiDAR data processing is inherently computation-heavy. The encoding process typically involves transforming sparse 3D point clouds into a structured 2D grid, which is dense and demands significant memory bandwidth to manage the high volume of data. However, the Giga operations per second (GOP/s) measure indicates the computational operations performed per second. The higher GOP/s of DPU2 in both models indicates that the detection head tasks are computationally intensive, involving more arithmetic operations (e.g., multiplications, additions) within the neural network layers than the encoding tasks on DPU1.

*2) Latency:* In evaluating latency on GPUs, we set timers before and after different modules of BEVDet and PointPillars to complete the measurement. However, for the quantized BEVDet, because the timer cannot be set inside the TensorRT engine, we use the Nvidia Nsight Systems [34] to analyze the latency of the image encoder and BEV encoder with detection head modules. For latency assessment on FPGAs, we analyze two corresponding subgraphs representing these two modules to determine their latencies. The results, as shown in Figure 6, indicate that the GPU achieves lower latency across all modules, which aligns with expectations given the superior computational power of the RTX 3080 compared to the ZCU104 device. Specifically, the image encoder on the GPU is accelerated by about 1.3 times compared to the FPGA, and the latency of the BEV encoder with detection head on the GPU is accelerated by about 20.6 times. Furthermore, the application of quantization and TensorRT accelerates BEVDet. The processing speed for the image encoder and the BEV encoder with detection head on the GPU is approximately 70.5 and 39.5 times faster than on the FPGA, respectively. It is worth noting that the detection results here are post-processed. In the quantized BEVDet, post-processing is not included in the TensorRT engine, so most of the time is spent in post-processing, which is about 4.8ms. The latency for PointPillars aligns with this trend. The quantized model on the GPU has the lowest latency. Compared to the FPGA, the latency for the LiDAR encoder and the detection head on the GPU is reduced by factors of 3.4 and 2.6, respectively.

*3) Power Usage:* In the assessment of power usage, we measure the maximum and average power utilization of each module during operation, excluding idle power usage. To obtain stable power readings on the GPU platform for both quantized and non-quantized models, we extend the runtime by executing each module 5000 times. Running multiple times is necessary to ensure the accuracy of our power measurements. In contrast, the Vaitrace library on FPGA allows for capturing power usage over a specific period. However, we need to adjust the Vaitrace library to capture the power for each module, eliminating the need for multiple runs. We set a 5 ms waiting time between the two modules for the FPGA, while on the GPU, we set the same waiting time as the interval for the same module to start again. This approach is more in line with the operating rules of different modules in the model and ensures the accurate measurement of the power of each module.

Table IV illustrates the average and maximum power usage for GPU (w/ & w/o quantization) and FPGA on all modules. After quantization, we observe that the average power usage on GPU drops for all modules except the BEV encoder with detection head. The high power usage for the BEV encoder with detection head may be because of the estimated power usage. Due to the limitation of registering custom operator during onnx inference, the two modules in the quantized BEVDet model are not able to be disassembled completely. Therefore, instead of obtaining accurate power usage, the estimated power usage of the image encoder module is calculated from the whole model power usage, and the estimated power usage of

TABLE III: Resource utilization for BEVDet and PointPillars on ZCU104 mpsoc.

| | Resource utilization | | | | Performance (GOP/s) | | Memory Bandwidth (MB/s) | |
|---|---|---|---|---|---|---|---|---|
| | LUT | Register | DSP | URAM | DPU1 | DPU2 | DPU1 | DPU2 |
| **Available resource on board** | 52161 | 98249 | 710 | 68 | - | - | - | - |
| **BEVDet** | 50951 | 97923 | 710 | 46 | 74.169 | 92.76 | 3413.031 | 1692.983 |
| **PointPillars** | 49281 | 97100 | 690 | 46 | 1.903 | 92.156 | 3890.906 | 1785.417 |

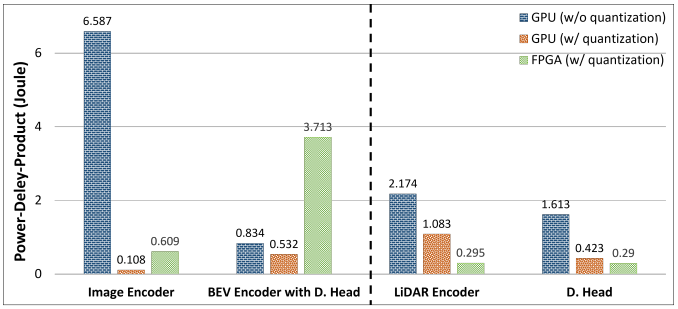



Fig. 7: Power-Delay-Product (PDP). It is the product of average power and latency.

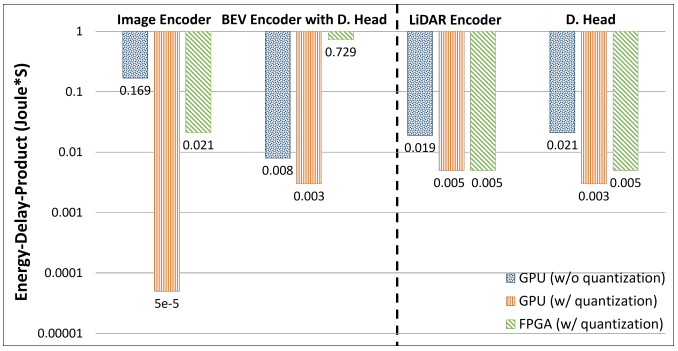



Fig. 8: Energy-Delay-Product (EDP). It is the product of average power and the square of latency. Due to the large difference in values, the logarithmic scale is used here. The data labels represent actual value without logarithmic scale.

the BEV encoder with detection head is calculated from the post-processing power usage.

Additionally, we observe that the maximum power usage of the quantized model on GPU is significantly higher than FPGA. It is indicated in Table IV that the power usage of both modules in BEVDet on GPU is approximately 14 times higher than the FPGA. Moreover, the FPGA's maximum power consumption for the LiDAR encoder in PonitPillars is approximately 12 times lower than that of its quantized model on GPU. Similarly, when focusing on the LiDAR-based detection head, the data reveals a reduction in power usage – approximately 3.5 times less on the FPGA compared to the GPU. Overall, we can conclude that the FPGA has a significant advantage in terms of power, with the power usage of each module being approximately one-tenth of the GPU.

*4) PDP and EDP:* To comprehensively evaluate efficiency, we incorporate latency and power into our analysis using Product-Delay-Product (PDP) and Energy-Delay-Product (EDP) metrics. Lower PDP values indicate a system's ability to operate quickly while using less power, and lower EDP values show efficient operation with minimal energy use. Our findings, illustrated in Figure 7, reveal that the quantized BEVDet on the GPU demonstrates superior PDP performance, particularly for the image encoder and BEV encoder with detection head, which exhibit the lower PDP. For the LiDAR encoder and detection head, the lower PDP is achieved when these components are deployed on the FPGA. The results of the EDP analysis are displayed in Figure 8. The analysis shows that the quantized image encoder and BEV encoder with detection head perform better on the GPU. On the other hand, the quantized LiDAR encoder and detection head have similar efficiencies on both the GPU and FPGA platforms. Thus, our results suggest that quantization and deployment on the GPU enhance the efficiency of the image encoder and BEV encoder with detection head. For the LiDAR encoder and detection head, FPGA deployment after quantization is more efficient. PDP and EDP are two methods that consider both latency and power consumption, and their use often requires adjustments based on specific requirements and constraints in different real-world scenarios. When the system prioritizes low latency and has no strict power or energy consumption constraints, deploying all four quantized modules on the GPU is better. On the other hand, if the system requires minimal energy consumption and can tolerate higher latency, placing LiDAR-related modules on an FPGA will be a more efficient choice.

## VI. Discussion

In this study, we aim to explore the performance of different modules of multi-modal detection models on heterogeneous hardware platforms, thereby providing guidance and insights for the cooperative design of hardware and software in multi-modal object detection systems. Initially, we select a typical multi-modal perception model, the BEVFusion, as our benchmark for analysis and evaluation. However, we soon realize that deploying BEVFusion on FPGAs posed numerous challenges. FPGAs' unique architecture and resource constraints mean that deploying complex models requires thorough optimization and adjustment. One of the main challenges is that bridging the hardware design and software design in the FPGA for large models requires detailed knowledge of both sectors. Because of the characteristics of FPGA, the hardware design is different from provider to provider and model to model, which shrinks the unified concept of FPGA architecture for neural networks. Considering these challenges, we decide to deconstruct BEVFusion into two specific mod-

TABLE IV: Comparison of average and maximum power consumption (in WATTS) for BevDet and PointPillars across GPU and FPGA, including GPUs with and without quantization and FPGAs with quantization. The asterisks (*) denote the power of the image encoder, which is estimated by the power of the overall model, while the power of the BEV encoder with D. head is estimated by the power of post-processing.

| | **BEVDet** | | | | **PointPillars** | | | |
|---|---|---|---|---|---|---|---|---|
| | **Image Encoder** | | **BEV Encoder with D. Head** | | **Lidar Encoder** | | **D. Head** | |
| | **Avg. Power (W)** | **Max Power (W)** | **Avg. Power (W)** | **Max Power (W)** | **Avg. Power (W)** | **Avg. Power (W)** | **Avg. Power (W)** | **Max Power (W)** |
| **GPU (w/o quantization)** | 257.2 | 267.3 | 87.5 | 88.2 | 248.1 | 251.6 | 122.3 | 126.5 |
| **GPU (w/ quantization)** | 224.5* | 228.8* | 107.0* | 111.0* | 222.9 | 244.2 | 63.1 | 66.4 |
| **FPGA (w/ quantization)** | 18.0 | 21.2 | 18.9 | 21.4 | 18.0 | 20.5 | 16.5 | 19.5 |

els, corresponding to the camera stream and LiDAR stream. This approach simplifies the deployment process and enables effective analysis of each module within multi-modal models. The BEVDet and PointPillar models we select use methods similar to BEVFusion for processing data from the camera and LiDAR, allowing us to evaluate the respective modules efficiently.

### *A. Fusion Module*

Even though the fusion module has a very small impact on latency as shown in Figure 2, it is still a very important module in the multi-modal model because features from different sensors are fused here. For example, in BEVFusion, as shown in Figure 3, the BEV features from the camera and LiDAR will be fused in the fusion module. Although in BEVDet, the camera features are transformed into BEV features for 3D object detection, PointPillar does not undergo such a transformation with LiDAR features, so we lack the analysis of LiDAR BEV features. However, obtaining BEV features from LiDAR is relatively straightforward in reality. Since point cloud data inherently contains depth information, there is no need for depth estimation as with camera data. All that is needed is to flatten the point cloud data along the Z-axis.

Nevertheless, analyzing the fusion module on an FPGA remains a complex task. This is not only because the inputs and outputs of the fusion module are not merely simple image or point cloud data, but it also requires handling BEV features from two different types of sensors. In future work, we plan to comprehensively explore how to deploy and evaluate the fusion module on FPGAs.

### *B. Multi-task Model*

BEVFusion is not just a multi-modal model but also a multi-task model. Apart from performing object detection tasks, it can also handle BEV segmentation. Since multi-task models typically share a backbone and neck, it makes sense to consider the segmentation head as another module for analysis. However, we must acknowledge the difficulty in deploying and evaluating this module separately on FPGAs. Therefore, in our current research, our primary focus is limited to the detection head. Considering the trend of multi-modal multi-task perception models, we plan to continue our research and analysis into these complex models' performance and potential.

### *C. Heterogeneous Hardware*

Communication between heterogeneous hardware is a critical factor to consider. Data transfer inevitably affects the model's inference speed and contributes to energy consumption. Therefore, achieving low-latency and low-power communication between heterogeneous platforms is key. As our study only analyzed the performance of different modules of a multi-modal perception system on GPUs and FPGAs, the design of hardware communication becomes an essential part when researchers design a software-hardware collaborative multi-modal model based on our evaluation results.

### *D. Model Quantization*

In this study, we realize it is unfair to directly compare original models on GPU with quantized models deployed on FPGA. Therefore, we also deploy quantized models on GPUs for evaluation. The quantized PointPillars is obtained from the official Vitis AI repository [31], while the quantized model of BEVDet, according to the Vitis AI documentation, came from BEVDet's official repository [2]. We successfully deploy and evaluate the quantized PointPillars on the GPU. However, for BEVDet, we can only obtain a full ONNX model from BEVDet's official repository. The BEVDet requires custom registration of the bevpoolv2 [35] operator in the kernel during onnx runtime inference. Unfortunately, this custom registration creates a limitation that prevents the quantized BEVDet from being disassembled on GPU. As a result, we have to use external tools to evaluate the model's runtime performance. We analyzed its latency using the Nvidia Nsight Systems, but this tool does not include power analysis capabilities. Therefore, we can not accurately analyze the power of each module within the quantized BEVDet. In this research, we use the power of the overall model to estimate the power usage of the image encoder module, while the power of the BEV encoder with detection head is obtained by measuring the post-process. However, a better solution to this problem involves obtaining two independent ONNX models for the image encoder and BEV encoder with detection head during the quantization of BEVDet. These can then be deployed

on GPU and FPGA for evaluation. This approach requires a broader understanding of quantization and a solid foundation in hardware design for deployment on FPGA. In this process, it is also necessary to optimize the model to improve accuracy. Some approaches such as quantization-awared training will help to avoid sacrificing the model accuracy.

## VII. Conclusion

Developing a multi-modal perception model is crucial in the evolving landscape of autonomous driving technology. Our work aims to explore the performance of multi-modal object detection model modules on heterogeneous hardware platforms, especially on GPU and FPGA. For this purpose, we conduct an in-depth analysis of the image-based BEVDet model (including image encoder, BEV encoder with detection head) and the PointPillars model based on LiDAR (including LiDAR encoder and detection head). The evaluation metrics cover resource utilization, latency, power, and energy consumption. In FPGA, the resource utilization is approximately 93% of the available resources, which is much better than that of GPU. Evaluation results also indicate that the latency for all four modules is markedly lower for GPU implementation compared to FPGA. However, FPGA outperforms in PDP and EDP in LiDAR-related processing. Moving forward, this research needs to be expanded to a broader range of models to provide insights that are crucial for designing multi-modal perception models with optimal hardware configurations, and for balancing resource utilization, latency, and power efficiency in panoptic perception tasks.

## Acknowledgment

This work was partly supported by the U.S. National Science Foundation under Grants CNS-2245729 and Michigan Space Grant Consortium 80NSSC20M0124.

## References

[1] 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 *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[2] J. Huang, G. Huang, Z. Zhu, Y. Yun, and D. Du, "Bevdet: High-performance multi-camera 3d object detection in bird-eye-view," *arXiv preprint arXiv:2112.11790*, 2021.

[3] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 12 697–12 705.

[4] P. Sun, R. Zhang, Y. Jiang, T. Kong, C. Xu, W. Zhan, M. Tomizuka, L. Li, Z. Yuan, C. Wang *et al.*, "Sparse r-cnn: End-to-end object detection with learnable proposals," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 14 454–14 463.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[6] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6569–6578.

[7] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3d object detection and tracking," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 11 784–11 793.

[8] J. Philion and S. Fidler, "Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*. Springer, 2020, pp. 194–210.

[9] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels," in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, 2019, pp. 1–8.

[10] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 751–766.

[11] C. Hao, A. Sarwari, Z. Jin, H. Abu-Haimed, D. Sew, Y. Li, X. Liu, B. Wu, D. Fu, J. Gu, and D. Chen, "A hybrid gpu + fpga system design for autonomous driving cars," in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, 2019, pp. 121–126.

[12] C. Hao and D. Chen, "Software/hardware co-design for multi-modal multi-task learning in autonomous systems," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2021, pp. 1–5.

[13] Z. Wang, Z. Jiang, Z. Wang, X. Tang, C. Liu, S. Yin, and Y. Hu, "Enabling latency-aware data initialization for integrated cpu/gpu heterogeneous platform," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3433–3444, 2020.

[14] Z. Wang, Z. Wang, C. Liu, and Y. Hu, "Understanding and tackling the hidden memory latency for edge-based heterogeneous platform," in *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*, 2020.

[15] H. Miao and F. X. Lin, "Towards out-of-core neural networks on microcontrollers," in *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, 2022, pp. 1–13.

[16] V. Kathail, "Xilinx vitis unified software platform," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 173–174.

[17] R. Snider, "Chapter 6: Introduction to intel quartus prime," in *Advanced Digital System Design using SoC FPGAs: An Integrated Hardware/Software Approach*. Springer, 2022, pp. 55–86.

[18] T. Feist, "Vivado design suite," *White Paper*, vol. 5, p. 30, 2012.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[20] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[22] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[23] 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*, 2021, pp. 10 012–10 022.

[24] 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 conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[25] 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 conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[26] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4490–4499.

[27] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin, "MMDetection: Open mmlab detection toolbox and benchmark," *arXiv preprint arXiv:1906.07155*, 2019.

[28] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[29] "Nvidia Management Library (nvml)," https://developer.nvidia.com/nvidia-management-library-nvml, accessed 2024-02-05.
[30] 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*, 2020, pp. 11 621–11 631.
[31] "Vitis AI-3.0 Documentation," https://xilinx.github.io/Vitis-AI/3.0/html/docs/quickstart/mpsoc.html, accessed 2024-02-05.
[32] "AMD Adaptive Computing Documentation Portal," https://docs.xilinx.com/r/3.0-English/ug1414-vitis-ai/vaitrace-Usage, accessed 2024-01-05.
[33] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3354–3361.
[34] "Nvidia Nsight Systems," https://developer.nvidia.com/nsight-systems, accessed 2024-02-05.
[35] J. Huang and G. Huang, "Bevpoolv2: A cutting-edge implementation of bevdet toward deployment," *arXiv preprint arXiv:2211.17111*, 2022.