AAHLS Final Report

T1 電子所 陳奕達 戴妤珊 蔡岳峰

I. Background

In autonomous driving, the most important part is to observe the surroundings of the car. As an example, the vehicle, pedestrian, and cyclist are required to be observed by obtaining their position and size. Therefore, object detection techniques are required in such autonomous driving scenario. An example of such object detection is shown in Fig. 1.

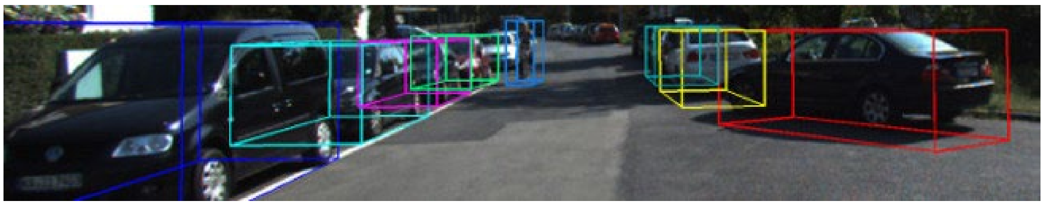


Fig. 1. Object detection visualization with 2D images.

In the object detection, there are two main categories, which are methods based on 2D camera image and 3D LiDAR, respectively as shown in Fig. 2. Compared with 2D cameras, 3D LiDAR has low sensitivity in different light condition and more accurate 3D mapping capability, which is favorable for autonomous driving. However, as the 3D point clouds from the 3D LiDAR do not have a regular structure, processing them requires higher computational complexity than the 2D images from regular camera.

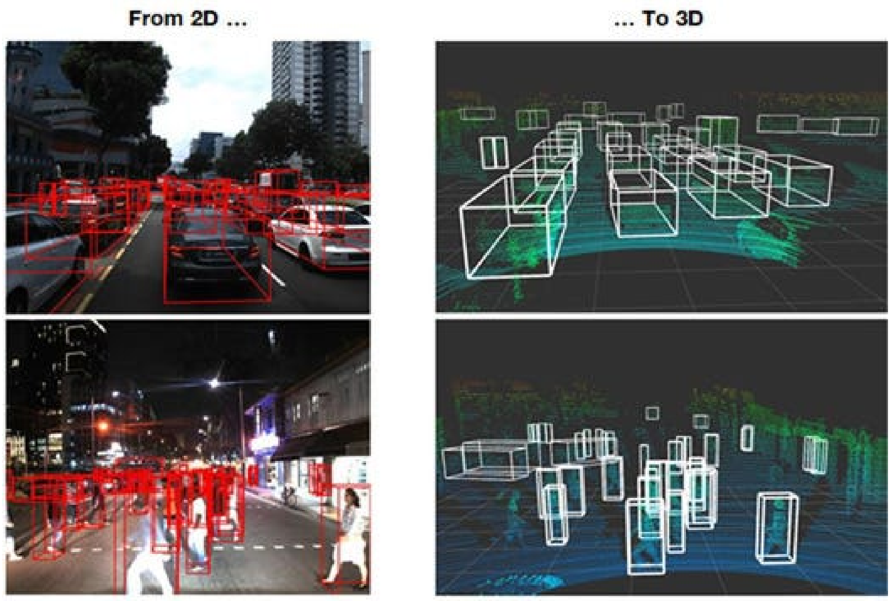


Fig. 2. Images from 2D cameras vs. point clouds from 3D LiDAR.

In addition, in the specification of autonomous driving, it can be categorized into 6 levels as shown in Fig. 3. We can see that in the level 3-5, the autonomous system requires to monitor the driving environment. The continuous monitoring of the environment requires real time processing, which its latency and throughput are highly related to the safety of the autonomous driving system.

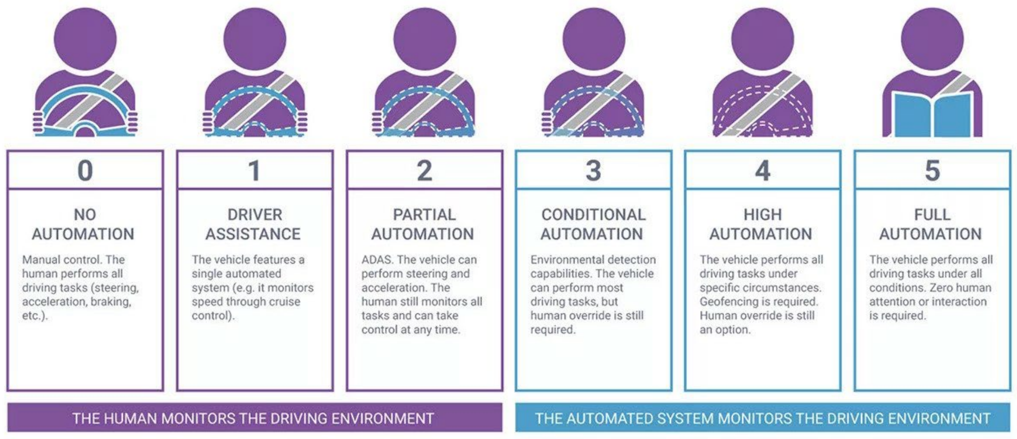

0
NO AUTOMATION
Manual control. The human performs all driving tasks (steering, acceleration, braking, etc.).
1
DRIVER ASSISTANCE
The vehicle features a single automated system (e.g. it monitors speed through cruise control).
2
PARTIAL AUTOMATION
ADAS. The vehicle can perform steering and acceleration. The human still monitors all tasks and can take control at any time.
3
CONDITIONAL AUTOMATION
Environmental detection capabilities. The vehicle can perform most driving tasks, but human override is still required.
4
HIGH AUTOMATION
The vehicle performs all driving tasks under specific circumstances. Geofencing is required. Human override is still an option.
5
FULL AUTOMATION
The vehicle performs all driving tasks under all conditions. Zero human attention or interaction is required.
THE HUMAN MONITORS THE DRIVING ENVIRONMENT
THE AUTOMATED SYSTEM MONITORS THE DRIVING ENVIRONMENT

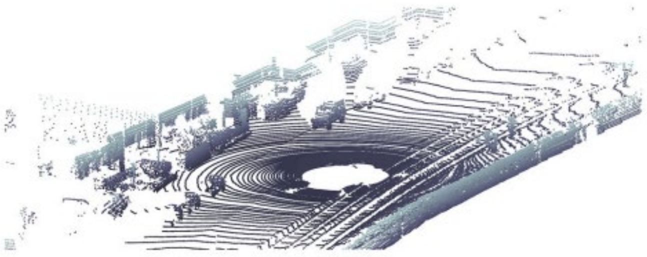
Fig. 3. The six levels of autonomous vehicles as described by the Society of Automobile Engineers (SAE) [1], their definitions, and the features in each level.

Therefore, hardware acceleration to approach real-time object detection is critical. However, there are several hardware platforms exist. Graphical processing unit (GPU) is a most common one and is also easy to deploy such application on to. However, GPU is power hungry and costs hundred watts when executing DNN models. On the contrast, application-specific integrated circuit (ASIC) is a more compact design which consumes less power and achieves high energy efficiency. However, the overall development life cycle of the ASIC costs more than one year. Moreover, ASIC is not as flexible as the GPUs. If the object detection system changes, it requires to redesign the circuits, which is costly. Therefore, we select a compromise, that is the FPGAs. FPGAs have higher energy efficiency then GPUs and are more flexible than the ASICs. In addition, the FPGA manufactures also provide several frameworks to assist the development of different applications, such as the Vitis AI and FINN from Xilinx.

In this project, we aim to observe the differences from different platforms and different implementation methods. We will implement the 3D object detection on the GPUs, FPGAs with Vitis AI, and FPGAs with FINN, and compare the performance difference among them.

II. Dataset

To do the object detection of the autonomous driving, the dataset is critical and required. We select the KITTI Vision Benchmark Suite [2] as the dataset. The KITTI dataset provides 7481 data for different scene on the road. It also provides three modalities, including the point clouds from the LiDAR sensor, the images from four surrounding cameras, and the information of the GPS/IMU navigation system for various development of different systems with different modalities. The following figure is to illustrate the LiDAR point cloud and the camera image from the same scene in the KITTI dataset.



(a) LiDAR point cloud [16]

(b) Camera image [16]

Fig. 4. (a) The visualization of the point cloud data and (b) the camera data from the KITTI dataset.

III. Model

In this section, we will describe the selected model for the object detection system with the 3D point clouds data. We select the PointPillars [3] as the target model as shown in Fig 5. The PointPillars model contains three parts: The Pillar Feature Net (PFN), the Backbone, and the detection head.

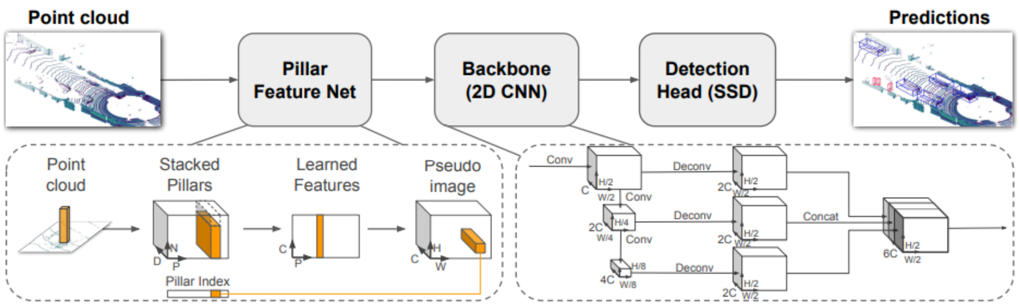
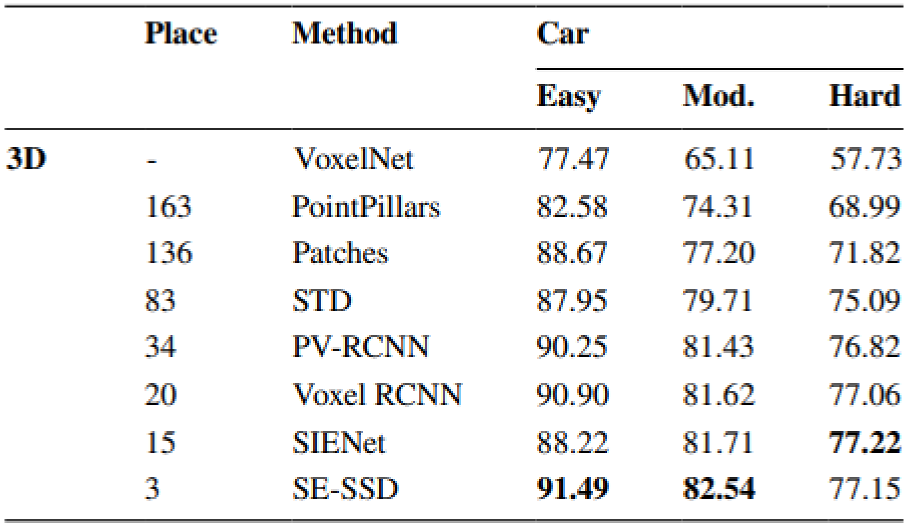


Fig. 5. The overall architecture of the PointPillars model.

As shown in Fig. 6, although PointPillars model is not the model with the highest mIoU, it is suitable for hardware implementations compared with other models with better mIoU. The main reason is that instead of using the 3D CNN or the transformer block, PointPillars first transforms the 3D points to 2D pseudo images, and the 2D pseudo images can be processed with the efficient 2D CNN backbone. Finally, different feature scale of the 2D CNN features are used to capture different size of the object in the 3D point cloud. The backbone architecture of the PointPillars is shown in Fig. 7. We can see that most of the calculations are from the T-Net (2D CNN) and the MLP (FCN). Therefore, it is suitable for FPGA acceleration.



	Place	Method	Car		
			Easy	Mod.	Hard
3D	-	VoxelNet	77.47	65.11	57.73
	163	PointPillars	82.58	74.31	68.99
	136	Patches	88.67	77.20	71.82
	83	STD	87.95	79.71	75.09
	34	PV-RCNN	90.25	81.43	76.82
	20	Voxel RCNN	90.90	81.62	77.06
	15	SIENet	88.22	81.71	**77.22**
	3	SE-SSD	**91.49**	**82.54**	77.15

Fig. 6. The mIoU of different models on the KITTI dataset.

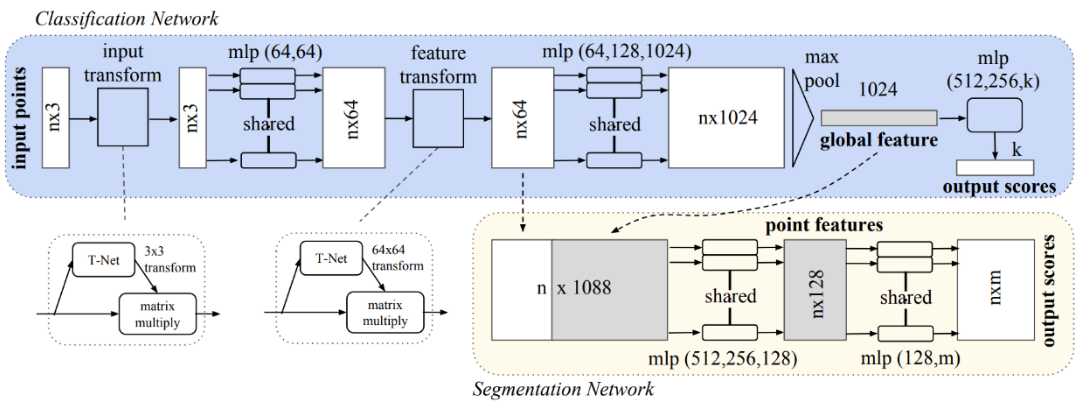


Fig. 7. The backbone of the PointPillars model.

IV. PointPillars on GPU

First, we execute the pretrained PointPillars model on the A5000 GPU as shown in Fig. 8. We first show the model architecture of the PointPillars on the GPU and check whether it is the same as the original paper as shown in Fig. 9. We reproduce the inferencing process of the PointPillars on the GPU. We execute 1000 batches of input data with 1 data per batch. We count the total execution time and divide it by the total number of 3D point cloud images. The latency can be calculated in this manner. After then, 1 second is divided by the latency and we can acquire the FPS of the PointPillars executed on the A5000 GPU. The GPU power is also observed with the nvidia-smi command as shown in Fig. 10. Finally, the FPS and the power of the PointPillars executed on the A5000 GPU are summarized in Table I.

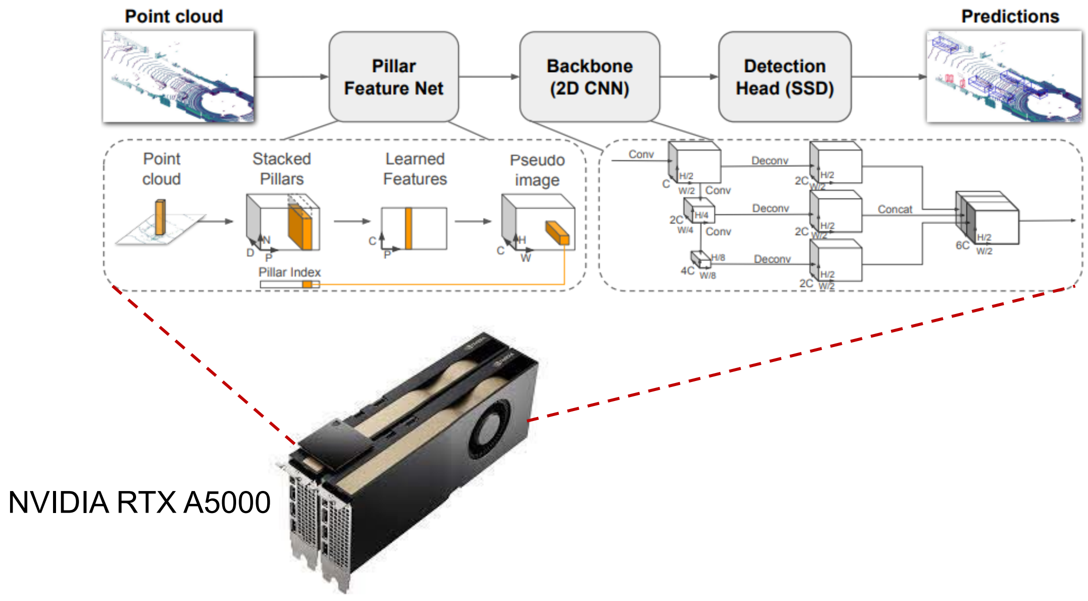
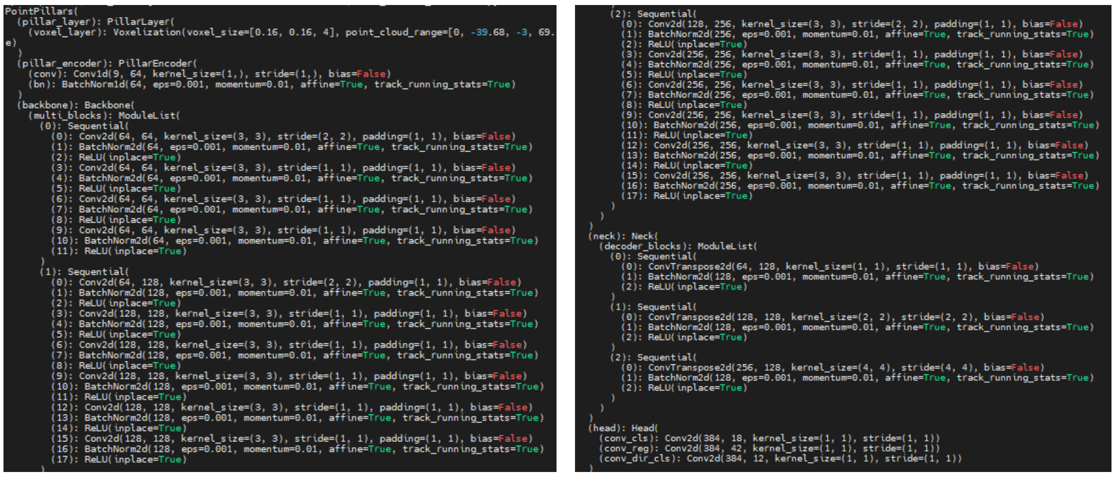


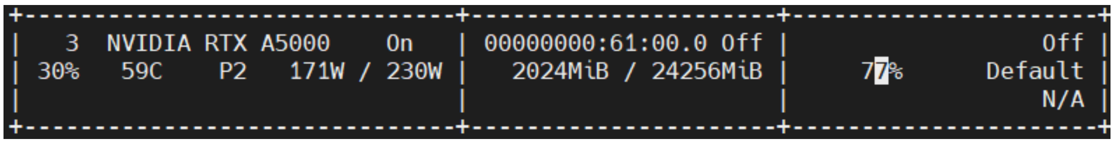
Fig. 8. The illustration of executing PointPillars on the A5000 GPU.



```
PointPillars(
  (pillar_layer): PillarLayer(
    (voxel_layer): Voxelization(voxel_size=[0.16, 0.16, 4], point_cloud_range=[0, -39.68, -3, 69.
e)
  )
  (pillar_encoder): PillarEncoder(
    (conv): Conv1d(9, 64, kernel_size=(1,), stride=(1,), bias=False)
    (bn): BatchNorm1d(64, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
  )
  (backbone): Backbone(
    (multi_blocks): ModuleList(
      (0): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(64, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (7): BatchNorm2d(64, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (8): ReLU(inplace=True)
        (9): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (10): BatchNorm2d(64, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (11): ReLU(inplace=True)
      )
      (1): Sequential(
        (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (7): BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (8): ReLU(inplace=True)
        (9): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (10): BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (11): ReLU(inplace=True)
        (12): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (13): BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (14): ReLU(inplace=True)
        (15): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (16): BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (17): ReLU(inplace=True)
      )
```

```
      (2): Sequential(
        (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (5): ReLU(inplace=True)
        (6): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (7): BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (8): ReLU(inplace=True)
        (9): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (10): BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (11): ReLU(inplace=True)
        (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (13): BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (14): ReLU(inplace=True)
        (15): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (16): BatchNorm2d(256, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (17): ReLU(inplace=True)
      )
    )
  )
  (neck): Neck(
    (decoder_blocks): ModuleList(
      (0): Sequential(
        (0): ConvTranspose2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
      )
      (1): Sequential(
        (0): ConvTranspose2d(128, 128, kernel_size=(2, 2), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
      )
      (2): Sequential(
        (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(4, 4), bias=False)
        (1): BatchNorm2d(128, eps=0.001, momentum=0.01, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
      )
    )
  )
  (head): Head(
    (conv_cls): Conv2d(384, 18, kernel_size=(1, 1), stride=(1, 1))
    (conv_reg): Conv2d(384, 42, kernel_size=(1, 1), stride=(1, 1))
    (conv_dir_cls): Conv2d(384, 12, kernel_size=(1, 1), stride=(1, 1))
  )
```

Fig. 9. The model architecture of the PointPillars executed on the GPU.



```
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA RTX A5000     On  | 00000000:61:00.0 Off |                  Off |
| 30%   59C    P2   171W / 230W |   2024MiB / 24256MiB |     77%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Fig. 10. The power of the GPU while executing the PointPillars model.

Table I: The power consumption and FPS of PointPillars on A5000

	NVIDIA RTX A5000
Power consumption (W)	171
Frames per second	54.45

V. PointPillars on FPGA with Vitis AI

In this section, we will describe the details of the PointPillars model executed on the FPGA with Vitis AI and FINN, respectively.

In the original settings, we are asked to implement our model on the PYNQ board or the KV260 board. However, the logic gates on both boards are too small to fit the PointPillars model into them. The PYNQ board only contains 85k logic cells and 220 DSPs, while the KV260 only contains 256K logic cells and 1.2K DSPs. Therefore, we implement our PointPillars model on the ZCU104 MPSoC development board. There are 504K logic cells, 38Mb on chip memory, and 1728 DSPs on the ZCU104 board. We setup the board with a power meter attached on the socket to measure the power draw of the entire board as shown in Fig. 11. Finally, the PetaLinux operating system is installed in the SD card and insert to the ZCU104 to run the OS on the CPUs.

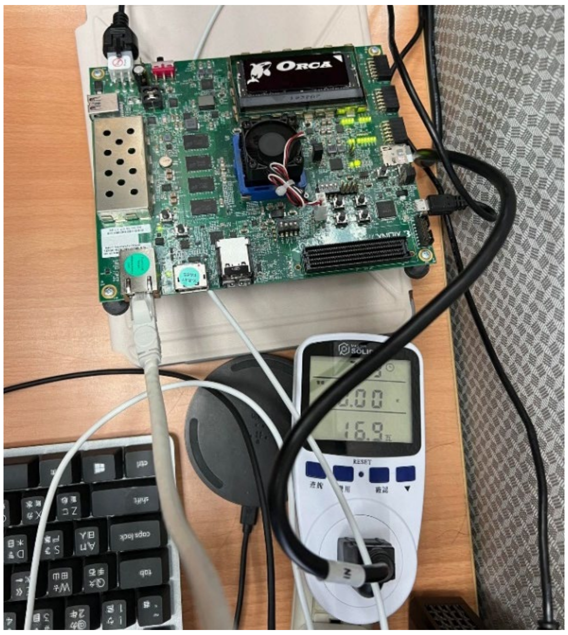


Fig. 11. The setup of the ZCU104 board and the power meter.

1. PointPillars with Vitis-AI

First, we build the PointPillars with the Vitis-AI. The core execution unit in the Vitis-AI is the deep processing unit (DPU) of the PL. The DPU is an accelerated which is controlled by its instruction set. Therefore, a DPU can execute various models with different instructions, which is suitable for reconfigurable design. The DPU on the ZCU104 is combined with 2048 MACs. The operating frequency of the DPU is 325MHz and the DSP unit inside the DPU is executed with 650MHz.

The overall executing process of the PointPillars on the FPGA with Vitis-AI is shown in Fig. 12. First the point cloud image is read from the SD card on the PS. The data preprocessing step is also executed on the PS. After preprocessed, the 3D image is sent to the PL and inputted into the PFN. The output of the PFN is then sent back to the PS for executing the scatter operation. The backbone and the detection head are then executed on the PL. Finally, postprocessing such as the Non-Maximum Suppression (NMS) process is executed on the PS. The final output is then store back to the SD card and can be further visualized.

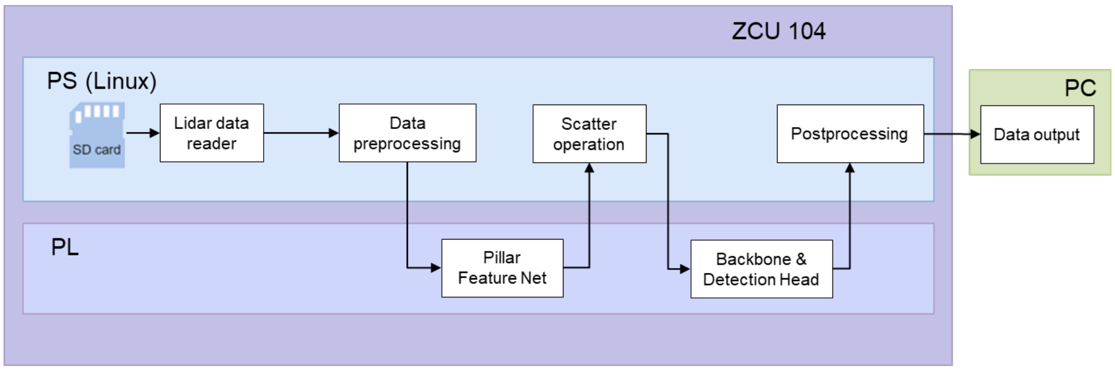
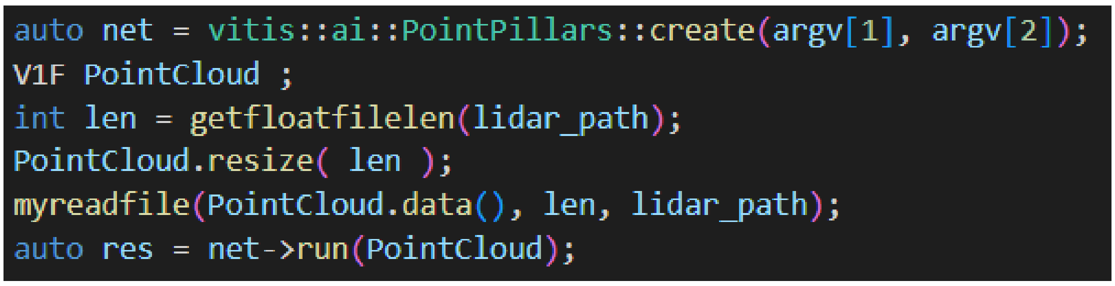


Fig. 12. The overall execution flow of PointPillars on the FPGA with Vitis-AI.

The host program of the Vitis-AI is shown in Fig. 13. First the vitis::ai::PointPillars create the nets with the given xmodel files. Then the 3D point cloud data is read from the SD card. Finally, the 3D point cloud data is inputted into the PointPillars network and get the final result.



```
auto net = vitis::ai::PointPillars::create(argv[1], argv[2]);
V1F PointCloud ;
int len = getfloatfilelen(lidar_path);
PointCloud.resize( len );
myreadfile(PointCloud.data(), len, lidar_path);
auto res = net->run(PointCloud);
```

Fig. 13. Host program (Only the most critical part) of the Vitis-AI.

In our experiment, we input a 3D image into the Vitis-AI PointPillars model. We observe the latency of the data and the power consumption of the board. We reported that the FPS of the Vitis-AI PointPillars is 18.472. The end-to-end latency is 5.4ms, while the latency of the DPU part is 3.4ms. As shown in Fig. 14, the idle power of ZCU104 is 16.4w, and the power during the Vitis-AI PointPillars execution is 18.3w. Therefore, the execution of the Vitis-AI PointPillars model is 1.9w. The summary of the FPS and the power consumption of the Vitis-AI PointPillars model is shown in Table II.

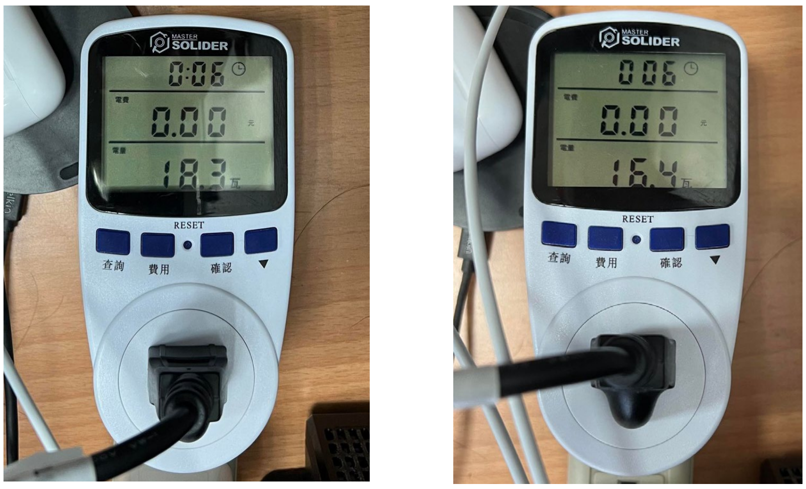


Fig. 14. Left: power consumption of ZCU104 under load; Right: power consumption of ZCU104 in idle.

Table II: The power consumption and FPS of Vitis-AI PointPillars on ZCU104

	ZCU104
Power consumption (W)	1.9
Frames per second	18.472

2. PointPillars with FINN

In this section, we will describe the details of the FINN-based PointPillars. Different from the Vitis-AI PointPillars, FINN-based PointPillars only offloads the backbone and SSD onto the FPGA as shown in the Fig. 15. The PFN is not offloaded due to the compatibility of the layers inside the PFN. Moreover, the FINN-based PointPillars are implemented in a dataflow manner, where each layer has its dedicated hardware to execute the operations. The compiled hardware is stored into a pp.xclbin file and is read in the host program. The bitstream of the pp.xclbin is then downloaded into the FPGA to execute the offloaded model of the FINN-based PointPillars. Moreover, different from the Vitis-AI PointPillars, the model information is stored in a pipeline.config file in the FINN-based PointPillars and is read into the host program during execution.

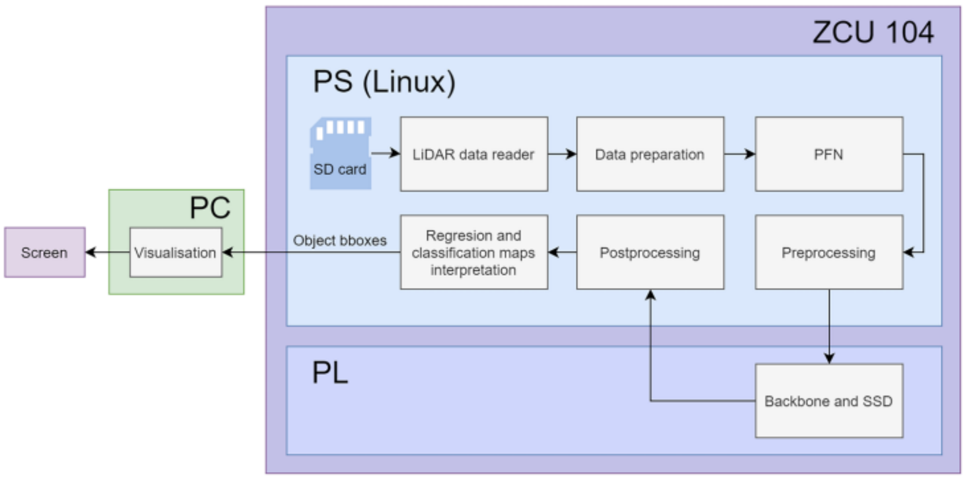
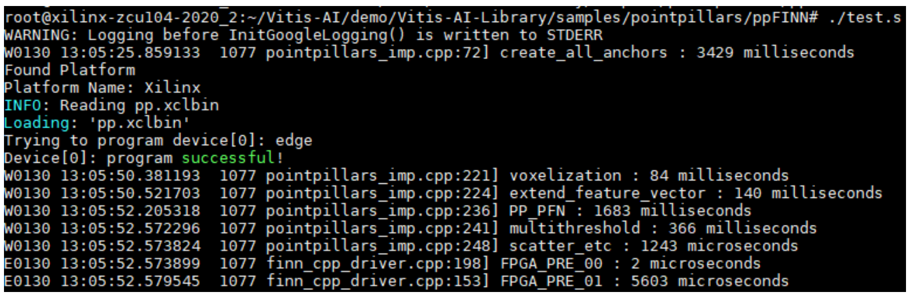


Fig. 15. The overall execution flow of PointPillars on the FPGA with FINN.

The host program of the FINN-based PointPillars model is based on the Vitis-AI PointPillars model. The preprocessing step and the post processing step of both the models are executed on the PS. Therefore, to lower the difficulty of developing the host program, FINN-based PointPillars reuse part of the code from the Vitis-AI PointPillars. The execution result is shown in Fig. 16. We have compared the execution time of our reproduced FINN-based PointPillars with the result from the paper [4]. The voxelization is increased from 7.13ms to 84ms. The extend feature vector process is increased from 1.65ms to 140ms. The PFN is increased from 62.43ms to 1683ms. The scatter process is increased from 0.72ms to 1.24ms. The preprocessing process is increased from 3.1ms to 5.6ms. We can see that most of the result get from the reproduced FINN-based PointPillars is far longer than that mentioned in the paper. We also measured the power consumption during the execution of the FINN-based PointPillars as shown in Fig. 17. The reproduced FINN-based PointPillars consumes 0.2w. The summary of the FINN-based PointPillars is shown in Table III.



```
root@xilinx-zcu104-2020_2:~/Vitis-AI/demo/Vitis-AI-Library/samples/pointpillars/ppFINN# ./test.s
WARNING: Logging before InitGoogleLogging() is written to STDERR
W0130 13:05:25.859133  1077 pointpillars_imp.cpp:72] create_all_anchors : 3429 milliseconds
Found Platform
Platform Name: Xilinx
INFO: Reading pp.xclbin
Loading: 'pp.xclbin'
Trying to program device[0]: edge
Device[0]: program successful!
W0130 13:05:50.381193  1077 pointpillars_imp.cpp:221] voxelization : 84 milliseconds
W0130 13:05:50.521703  1077 pointpillars_imp.cpp:224] extend_feature_vector : 140 milliseconds
W0130 13:05:52.205318  1077 pointpillars_imp.cpp:236] PP_PFN : 1683 milliseconds
W0130 13:05:52.572296  1077 pointpillars_imp.cpp:241] multithreshold : 366 milliseconds
W0130 13:05:52.573824  1077 pointpillars_imp.cpp:248] scatter_etc : 1243 microseconds
E0130 13:05:52.573899  1077 finn_cpp_driver.cpp:198] FPGA_PRE_00 : 2 microseconds
E0130 13:05:52.579545  1077 finn_cpp_driver.cpp:153] FPGA_PRE_01 : 5603 microseconds
```

Fig. 16. The execution result of FINN-based PointPillars on the FPGA.

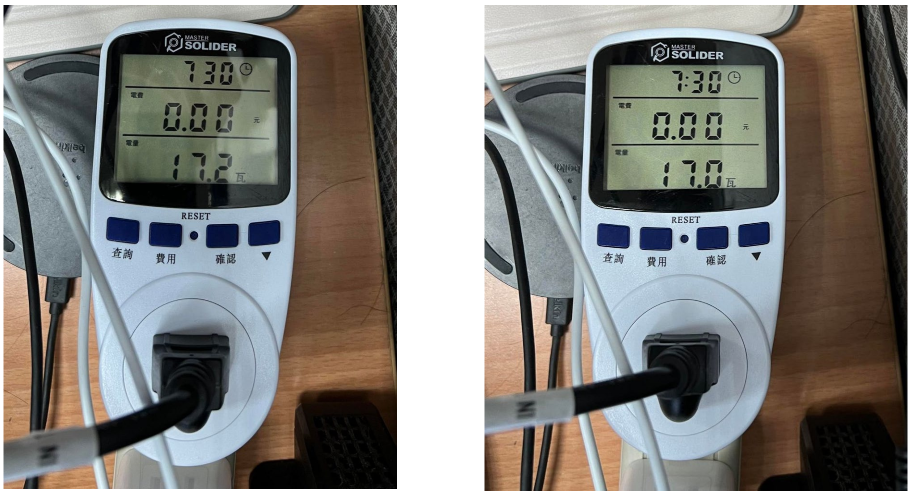


Fig. 17. Left: power consumption of ZCU104 under load; Right: power consumption of ZCU104 in idle.

Table III: The power consumption and FPS of FINN-based PointPillars on ZCU104

	ZCU104
Power consumption (W)	0.2
Frames per second (Paper)	2.67
Frames per second (Transplant)	5.417

As mentioned in their paper, to fit the PointPillars model into the FINN process, they have modified the model architecture. Therefore, we suspect that this modification indeed increases the model complexity and causes longer execution of the model. We apply the modified FINN-based PointPillars model and transplant it onto the Vitis-AI based PointPillars to observe the execution latency. After implementation the modified FINN-based PointPillars model consumes 18.54ms and the FPS is 5.41. It is 3.41x slower than the model of the Vitis-AI PointPillars. Therefore, we think that this trade-off is not a good solution for such FINN-based PointPillars and thus cannot observe the advantages of using the FINN to implement the DNN model.

VI. Visualization of different PointPillars model

In this section, we will show several results from the GPU-based PointPillars model and the Vitis-AI PointPillars model. The first result is shown in Fig. 18. In this case, both models perform similar, which most of the vehicles are captured. We show another example in the Fig. 19. There are some vehicles that is not circled by the Vitis-AI

PointPillars model. We infer that this error is caused from the quantization of the original model.

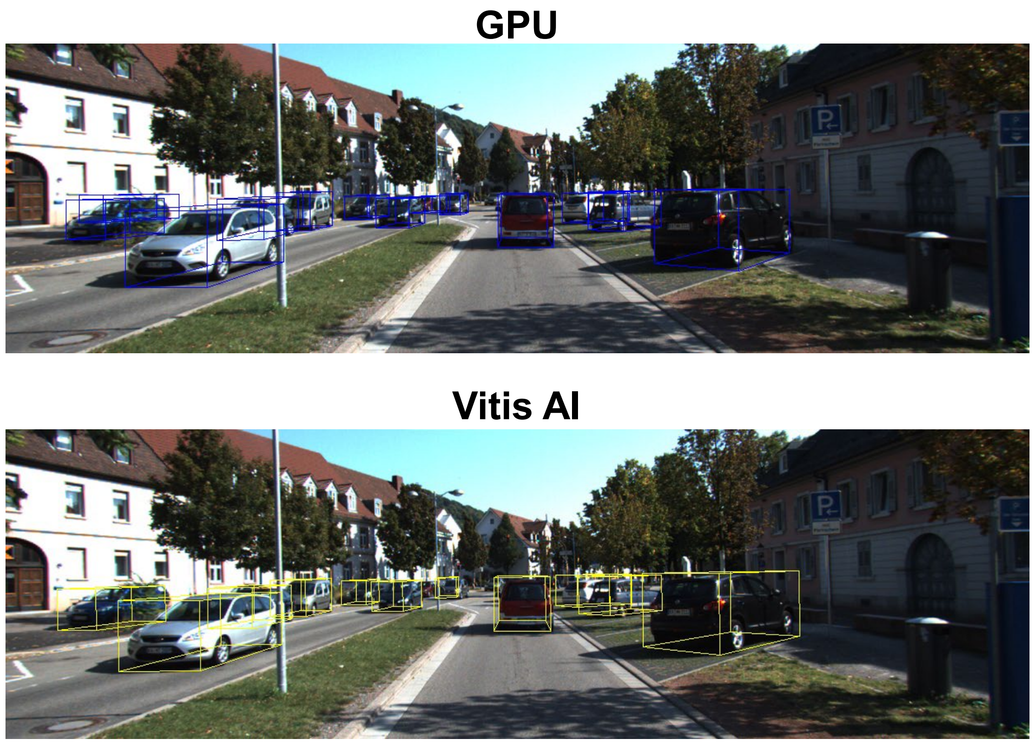


Fig. 18. The first example of the PointPillars from the GPU-based and the Vitis-AI PointPillars model.

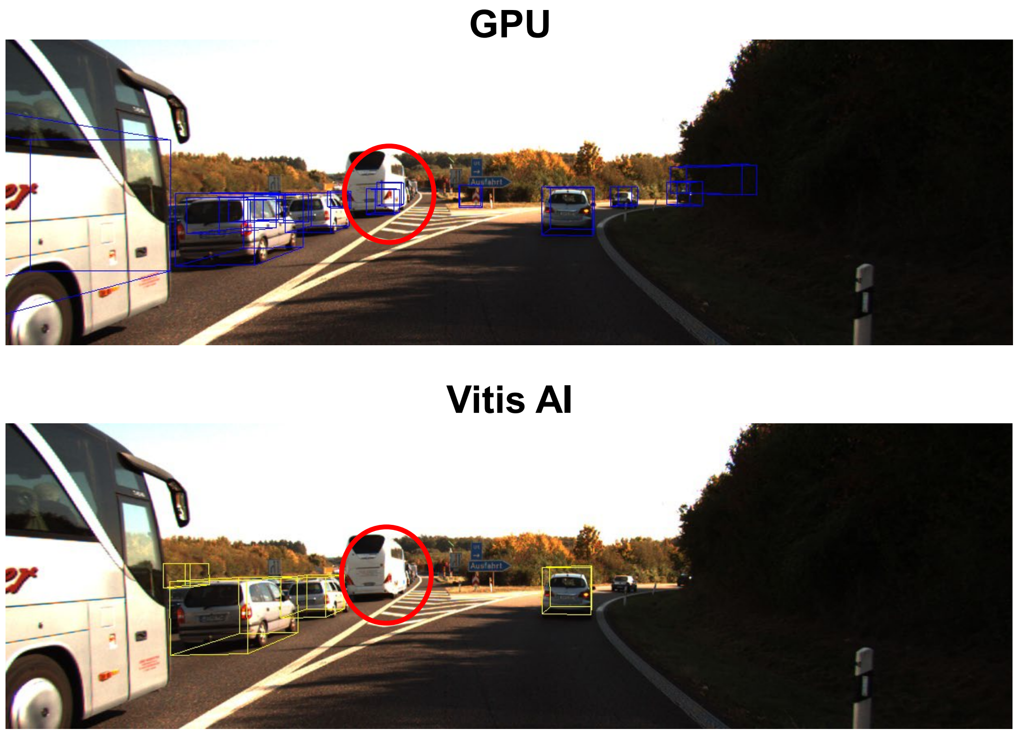


Fig. 19. The second example of the PointPillars from the GPU-based and the Vitis-AI

PointPillars model.

VII. Conclusion and Discussion

The comparison of three models are shown in Table IV. Although FINN-based PointPillars has the lowest frame per joule, the FPS is not enough for real-time processing. The Vitis-AI PointPillars has the best trade-off between the energy efficiency and the FPS, which is suitable for object detection in the autonomous driving.

Table IV: The power consumption and FPS of FINN-based PointPillars on ZCU104

	GPU	**Vitis AI**	**FINN**
Device	RTX A5000	ZCU104	ZCU104
FPS	**54.45**	18.472	2.67
Power (watt)	171	1.9	**0.2**
Frame per Joule	0.32	9.722 (**30x**)	**13.35 (41.74x)**

In summary, FPGA still shows superior energy efficiency compared with GPUs. However, in our experience, developing the application in FPGAs are more difficult than that of GPUs. Vitis-AI is easy to deploy. However, there exist small space to finetune the performance of the DNN model. Therefore, we suggest that Vitis-AI is suitable for demo of specific application. On the other hand, FINN has the possibility to approach higher energy efficiency than Vitis-AI. However, if sacrifice is needed due to the compatibility of the FINN framework itself, it may be not a good choice to use FINN as the development tool. Finally, we met several issues that is due to the compatibility between different versions of the Vitis AI and the PetaLinux itself, causing segmentation fault, and we can barely debug. We hope the future version of FINN and Vitis AI can be better backward compatible to older versions of PetaLinux.

VIII. Reference

[1] Shadrin, Sergej S., and Anastasiia A. Ivanova. "Analytical review of standard Sae J3016 «taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles» with latest updates." Avtomobil'. Doroga. Infrastruktura. 3 (21) (2019): 10.

[2] Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. International Journal of Robotics Research (IJRR).

[3] Lang, Alex H., et al. "Pointpillars: Fast encoders for object detection from point clouds." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.

[4] Stanisz, Joanna, Konrad Lis, and Marek Gorgon. "Implementation of the pointpillars network for 3D object detection in reprogrammable heterogeneous devices using FINN." *Journal of Signal Processing Systems* 94.7 (2022): 659-674.