Evaluating Performance of LiDAR and Camera Fused 3D Object Detection Using BEVFusion

Team 17
Project Option 1

Kaushika Uppu	Sravani Neelapala	Loi Nguyen	Hau Nguyen
014756859	016801486	010384543	017530084
kaushika.uppu@sjsu.edu	sravani.neelapala@sjsu.edu	loi.p.nguyen@sjsu.edu	hau.v.nguyen@sjsu.edu

***Abstract*—Autonomous driving significantly relies on the ability to detect and localize 3D objects within a complex environment. Two inputs typically used for this problem are LiDAR (Light Detection and Ranging) and cameras. With both technologies having pros and cons, using a fusion of LiDAR and cameras could lead to a more comprehensive 3D perception system with higher accuracy and robustness. This project aimed to integrate BEVFusion, a state-of-the-art multi-sensor fusion framework, into a ROS-2 environment to evaluate 3D object detection on the NuScenes dataset. We found that in terms of latency, the Jetson Orin Nano and P100 GPU outperformed the other systems with higher FPS and inference times. In terms of performance, we obtained extremely low mAP scores across the board due to what we think were architecture mismatch issues.**

***Keywords*—3D object detection, BEVFusion, LiDAR, ROS-2.**

I. Introduction

Object detection, and more specifically 3D object detection, has numerous applications, but it is especially important for autonomous vehicles. These vehicles need to carefully scan the environment around them in order to safely navigate through trajectories on their routes. However, this is not a simple problem as these environments are typically very complex and have many simultaneously moving parts. Therefore, for the safety of everyone in and around the vehicle, it is absolutely critical that autonomous vehicles are able to localize the objects and terrain around them accurately.

Vehicles with autonomous systems tend to have main kinds of inputs to take in the world around them: camera and LiDAR (Light Detection and Ranging). LiDAR is a remote sensing mechanism that utilizes pulsed laser light to construct a 3-dimensional model of the surrounding environment [1]. These two data collection systems both aid in the solution towards safe autonomous navigation, but each of them have their strengths and weaknesses.

LiDAR, given its laser-based sensing, is less affected by changing environments and poor weather conditions. Conversely, cameras struggle in low-light and foggy environments. LiDAR sensors are also particularly adept at capturing geometric precision and depth given their 3D spatial data outputs. On the other hand, cameras are specifically effective in capturing color, texture, and other contextual information. LiDAR, however, is not as successful in catching the rich texture of the environment [2].

With both of these technologies having their pros and cons, using a fusion of LiDAR and cameras could help systems leverage the strengths of each one while mitigating their weaknesses. This would thus result in a more comprehensive 3D perception and detection system with higher accuracy and robustness. In this project, we therefore aim to use a state-of-the-art multi-sensor fusion framework that combines LiDAR and camera, BEVFusion, to evaluate 3D object detection on the NuScenes dataset. We intend to integrate this into a ROS-2 environment, and analyze the model's overall system-level performance, latency, and accuracy.

II. Related Work

With advancements in technology, particularly related to computer vision, 3D object detection in autonomous vehicles has evolved with deep learning approaches. One type of model in this area is LiDAR-based. Point-based detectors such as Qi et al.'s PointNet++ [3] and Shi et al.'s PointRCNN [4] utilize point cloud inputs for sampling, which are then used to learn features downstream. In addition, another kind of LiDAR detectors are grid-based detectors that represent a point cloud as a voxel or pillar. Some of the prominent frameworks in this area include VoxelNet [5], PointPillars [6], and SECOND [7].

In contrast to these approaches, camera-only pipelines have also been developed. Stereo-based methods utilize pairs of stereo images to capture depth information for detection,

such as Stereo R-CNN [8] and IDA-3D [9]. Enhancing these frameworks are multi-view object detection techniques, which construct a unified bird's eye view (BEV) space from multiple different cameras to carry out detection. Models that leverage this type of 3D detection are BEVDet [10], Lift-Splat-Shoot (LSS) [11], DETR3D [12], and BEVFormer [13].

However, given the limitations that both LiDAR-only and camera-only detection pipelines have, multi-modal object detection pipelines have been proposed to try and adopt the strengths of each. Early fusion-based techniques typically integrated image information into the LiDAR point clouds, and then used a LiDAR detection framework. These include models such as F-PointNet [14], and are often a very sequential pipeline. These were elevated with methods that fused LiDAR and camera data at more intermediate stages, such as in the backbone or during RoI refinement. The models in this category include DeepFusion [15] and FUTR3D [16].

BEVFusion [17] is a model of the last category, fusing camera and LiDAR data into a shared BEV representation. The fusion is conducted with the output feature maps of the backbones, making it a framework with intermediate stage fusion. It also specifically utilizes BEV pooling to combine LiDAR and camera information and features.

III. Model Design

As mentioned above, the model we chose to use in this project is BEVFusion (Bird's Eye View Fusion). Other fusion-based approaches typically use camera features to augment LiDAR's point cloud data, but this method often causes the loss of semantic information present in camera data [17]. BEVFusion addresses this issue by unifying multi-modal data into a shared BEV space instead. Therefore, it keeps both geometric structure from LiDAR inputs and semantic density from camera inputs to provide a more comprehensive object detection framework.

The model architecture consists of four main stages: LiDAR feature transformation, camera feature transformation, multimodal fusion, and object detection. The diagrams below show a more detailed visual representation of this pipeline with inputs, outputs, and their dimensions (Fig. 1, 2).

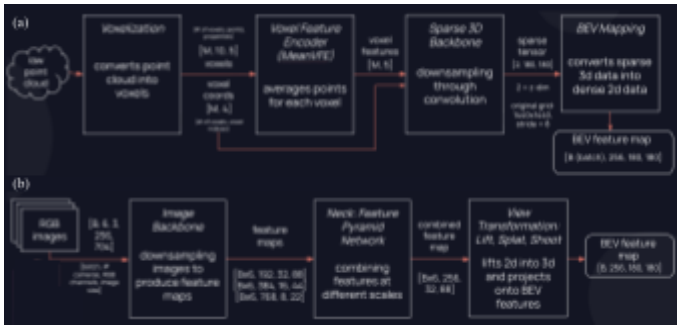

Fig. 1. The first two stages of the BEVFusion model architecture: (a) LiDAR feature transformation and (b) camera feature transformation.

A. Stage 1: LiDAR Feature Transformation

This stage takes in LiDAR input data and transforms it into the BEV shared representation space. The first step is voxelization, which takes in a raw point cloud and discretizes it into voxels and voxel coordinates. Then, the voxel feature encoder, MeanVFE, averages the points for each voxel and outputs aggregated voxel features.

Next, the data goes through the sparse 3-dimensional backbone. In this step, the voxels are downsampled through convolutional layers to extract multi-scale features from the voxel data. The backbone outputs a sparse tensor, reduced in size to a grid of 180x180 from an original 1440x1440. Finally, the features are mapped to the BEV space. The LiDAR 3D data is projected into the 2D BEV plane, converting the sparse features into a dense 2D feature map.

B. Stage 2: Camera Feature Transformation

This is a parallel step to the one before, which instead transforms the camera data into the BEV representation space. First, RGB images go through the image backbone. This step downsamples the images to extract multi-scale 2D image features. Through multiple layers, the backbone captures features at different spatial resolutions. It then outputs numerous feature maps at decreasing resolutions and increasing channel counts.

Moving on, the feature maps are inputted into the neck, or the feature pyramid network (FPN). The FPN fuses the multi-scale features from the backbone through upsampling and lateral connections to combine deeper, semantically rich and shallower, high-resolution features. This creates a map of a robust semantic representation of the camera features.

Finally, the last step is the view transformation, which puts the camera features in the BEV plane through the Lift, Splat, Shoot (LSS) method. First, a depth distribution is predicted for each pixel, lifting 2D features into 3D space. Then, these are aggregated onto a common 3D grid in the camera's frame. Next, the height dimension is collapsed to create the final 2D BEV map.

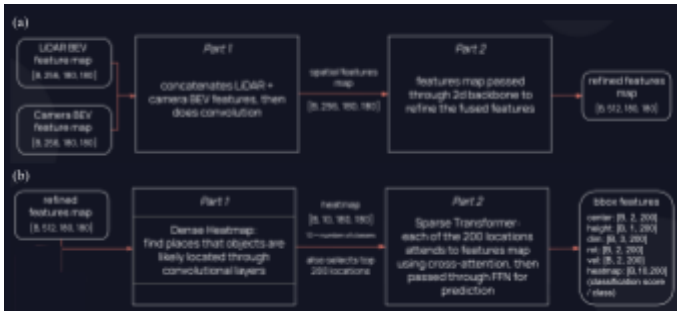

Fig. 2. The last two stages of the BEVFusion model architecture: (a) multimodal fusion and (b) object detection.

C. Stage 3: Multimodal Fusion

Now that both LiDAR and camera raw data have been mapped onto the BEV representation space, they can be fused together. The first part of this consists of combining the information into a unified feature space through concatenation along the channel dimension. This is followed by convolutional layers to blend the features and reduce the channel count, outputting a spatial features map.

The second part of this stage is to refine the fused features. A 2D backbone designed for BEV features is used for this step, and increases the channel count of the spatial features map. This network continues to aggregate local and global context of the features for a better representation of the data.

D. Stage 4: Object Detection

The final stage is to use the refined spatial features map to predict the 3D bounding boxes and properties of the detected objects through a detection head. The first step is a dense heatmap prediction. Through convolutional layers, this generates a dense map indicating the likelihood of an object center being present at each BEV grid location. In addition to the dense heatmap, this step also selects the top 200 locations as initial object center candidates.

Finally, a sparse transformer is used to refine the initial proposals. This step uses cross-attention, where each of the 200 queries attends to the entire BEV feature map. This allows the model to globally select the most relevant features to make an accurate prediction for an object. The resulting features are then passed through feed forward networks to predict the final bounding box attributes.

The bounding box feature outputs of this stage are as follows:

- Center (2D BEV coordinates)
- Height (3D object height)
- Dimension (3D object size: length, width, height)
- Rotation (Encoded rotation: sin(θ), cos(θ))
- Velocity (Predicted velocity in BEV)
- Heatmap (classification score per class)

IV. Implementation

A. High Performance Computing (HPC)

For one of the baselines, we chose to use SJSU's High Performance Computing (HPC) server. With this, we ran BEVFusion on a NVIDIA A40 GPU, with a CUDA version of 12.6, and memory of approximately 48 GB. In terms of setup, first we utilized Conda to create an environment. We then installed the necessary packages, such as setuptools and openmim. Next, we moved onto the builds that required more custom implementations, as pre-built wheels resulted in kernel errors when trying to run inference. We cloned mmcv from Github and compiled and built it with CUDA from scratch. We followed the same procedure for mmdetection3d and mmdetection. We also had to reinstall numpy to a version below 2.0 to account for any dependency errors. The detailed commands and instructions for exactly how the environment was set up for this portion of the project can be found in the Github repository under "a40gpu_setup.md."

In terms of getting BEVFusion, we downloaded the LIDAR-Cam model configuration and checkpoint from the BEVFusion Github [18] and uploaded them to the HPC server. For the dataset, we first attempted to use the full NuScenes dataset with a combined train and validation of 850 scenes. However, while creating this dataset on the server, we kept running into errors that prevented this creation. After trying to fix it for quite some time, we fell back to using the NuScenes mini dataset instead. For this, we downloaded the .tgz file from the NuScenes website [19], and uploaded and unpackaged it onto the server.

For testing, we had to modify the configuration file that we downloaded to adjust for using the mini dataset, which can be found as "bevfusion_mini.py" in our Github repository. Additionally, the original inference script utilized NuScenes evaluators, which led to errors due to the mismatch between the full and mini datasets. Therefore, we created dummy evaluators and bypassed those and instead computed metrics after inference was complete. The script used for this can be found as "get_nusc_metrics.py" in the repository. We also slightly modified the "test.py" script to output the latency metrics, which can also be found in the repository.

B. Google Colab

In this project, another experimentation was establishing a reproducible environment setup for BEVFusion and nuScenes using Google Colab, where established a new environment in addition to a pre-existing environment with Python 3.12, setup a new PyTorch 2.3.0 environment with CUDA 12.1 and a compatible torchvision, and tested the usage of a Tesla T4 in these environments. Further, we set up mmdet and mmengine, two OpenMMLab projects providing a 3D detection toolkit with dependencies like mmdet3D, and devoted a considerable amount of time to resolving mismatches in versions and extensions not being compiled. To make this foolproof, we wrote a short diagnostic tool to print information such as the path for Python, versions for Torch and CUDA, and checked if mmdet3D [20] and basic functions of it can be imported successfully in a Python 3.10 environment.

On the data side, we added the nuScenes v1.0-mini dataset [19] to the mmdetection3d project. This step involved mounting Google Drive in Colab, unpacking the dataset into /content/nuscenes, establishing the correct symlink folders (data/nuscenes -> /content/nuscenes), and setting up the necessary libraries such as the nuScenes devkit, geometry, and vision libraries. We used the official tools/create_data.py Nuscenes preprocessing script to prep the info files for train and val datasets and also prep the 3D-ground-truth database. Additionally, we ensured that all statistics related to cars, pedestrians, trucks, motorcycles, and other statistics were in

order, ensuring all datasets were appropriately set up for BEVFusion.

On the modeling and inference part, we configured the BEVFusion model with the official lidar+camera configuration and checkpoint from nuScenes. We downloaded this checkpoint into the project's directory and constructed project-specific CUDA extensions used in efficient inference. We also used functions to set up an initialization of the BEVFusion model [17], suppress non-critical differences in state_dict size during initialization, and check if the entire 3D detection pipeline functioned end-to-end. To gain a clearer insight into data processing, we began to examine individual samples of the nuScenes dataset by ensuring each batch consisted of samples, had a proper point cloud dimensionality, and processed a grid of multi-camera images properly. The last step was the latency bench-marking scripts for BEVFusion on nuScenes in Colab. The latency bench-marking scripts assemble a testing/validation dataset of nuScenes using the OpenMMLab registry, conduct warm-up loops, and proceed to record latency on a per-sample basis via model.predict with single-sample mini-batches and correct GPU synchronization.

The NuScenes mini dataset had a total of 81 samples for 10 scenes, and annotations for a variety of different objects, including cars, pedestrians, etc. In terms of our model behavior, the BEVFusion was successfully initialized, and the LIDAR + camera fusion layers were executed. Additionally, the CUDA operators, such as bev_pool and voxel, compiled correctly.

C. Optimization of BEVFusion on Jetson Orin Nano

The project implements a ROS 2 node for real-time 3D object detection using NVIDIA's CUDA-BEVFusion [21] on NVIDIA Jetson Orin Nano, processing camera images and LiDAR point clouds. The project attempts to address specific challenges for the Orin Nano, including compiling custom CUDA kernels for sm_87 (Ampere), and wrapping the C++ inference engine in a ROS 2 Python node.

We built the CUDA-BEVFusion from source because CUDA kernels must be compiled for the specific GPU architecture (SM version). The build process creates platform-optimized shared libraries that enable the ROS 2 Python node to interface with the C++ inference engine. NVIDIA's CUDA-BEVFusion include 3 different pretrained BEVFusion models with different camera backbones: Swin-Tiny (FP16), ResNet50 (FP16), ResNet50-PTQ (post-training quantization, INT8). The models use the same SparseConvNet for the LIDAR backbone. We built specific TensorRT engines for each configuration to optimize inference performance.

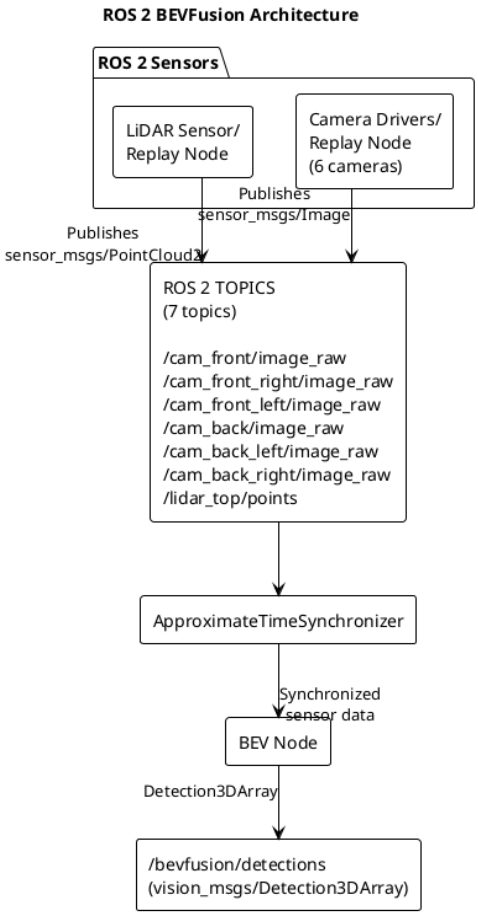

Fig. 3. ROS 2 BEVFusion architecture

We developed a custom ROS 2 package containing a dedicated inference node that utilizes this publish-subscribe architecture to process sensor data and publish detection results. ROS 2 package is an organizational unit for ROS 2 code that helps sharing, installing and running implementation of BEVFusion in ROS 2 easier. Inside the package, nodes are responsible for modular functions such as replaying sensor data, doing real-time 3D object detection inference, or logging detections. The nodes exchange information by subscribing or publishing to topics.

The ROS 2 BEVFusion architecture uses a publish-subscribe pipeline: The replay node publishes images to 6 camera topics and LIDAR pointclouds to a LIDAR topic. An ApproximateTimeSynchronizer synchronizes these 7 topics and forwards aligned sensor data to the BEV Node, which processes the fused sensor inputs and publishes 3D detections as vision_msgs/Detection3DArray on a detections topic.

The BEV node uses a layered architecture: a Python orchestration component handles ROS 2 message parsing and

data preparation, while a C++ inference component executes optimized TensorRT models. Synchronized sensor data (six camera images and one LiDAR point cloud) enters the Data Preparation component, which decodes and resizes images to 1600×900, converts BGR to RGB, extracts point cloud coordinates (x, y, z, intensity), and manages zero-copy buffers. The prepared data is converted to NumPy arrays (float16) and passed via pybind11 bindings (libpybev.so) to the C++ inference engine through load_bevfusion() and forward() calls.

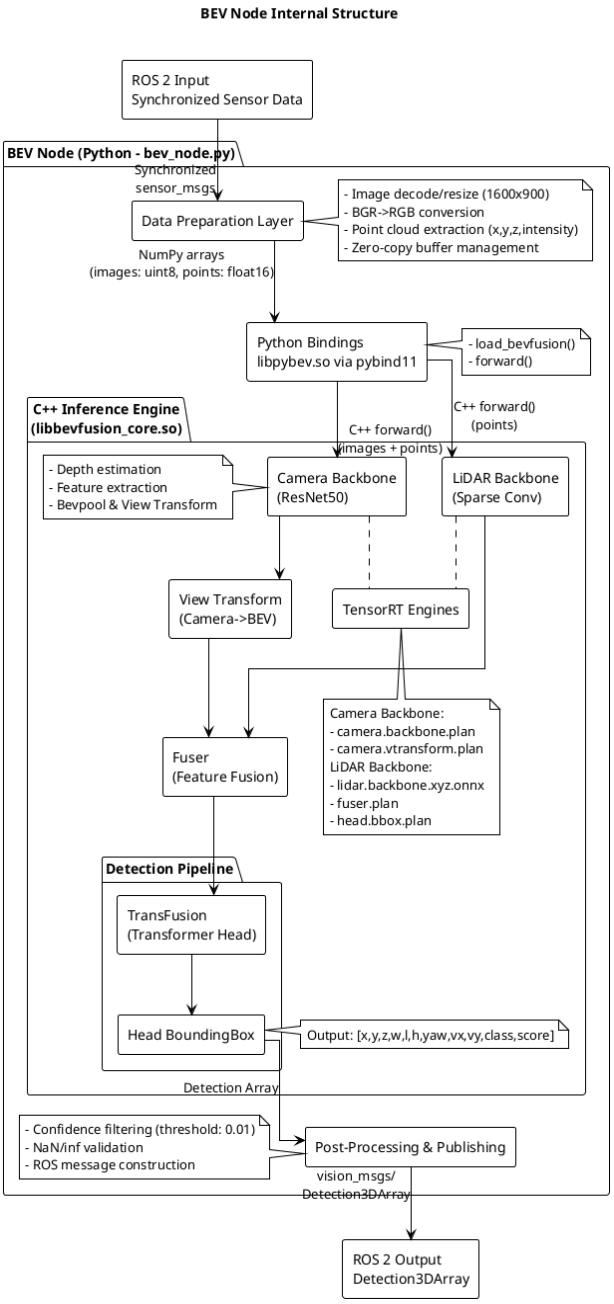

Fig. 4. BEV Node internal structure

The C++ Inference Engine (libbevfusion_core.so) runs a multi-stage pipeline using TensorRT-optimized models: the Camera Backbone extracts features, the View Transform projects them to Bird's-Eye View, the LiDAR Backbone processes sparse point cloud features, the Fuser fuses camera and LiDAR features, and the Detection Head (TransFusion, 3.0 MB) outputs 3D bounding boxes with [x, y, z, w, l, h, yaw, class, score]. Results return to Python for Post-Processing & Publishing, which applies confidence filtering (threshold 0.01), validates for NaN/inf, constructs ROS messages, and publishes detections. This design keeps compute-intensive operations in optimized C++/CUDA while maintaining Python flexibility for integration and message handling.

D. Tesla P100

The project also tries to implement a foxy ROS2 node on the TeslaP100 inside a custom Docker environment. The similar challenges were faced with the Jetson like compiling custom CUDA kernels for sm60. This system rebuilt the core components, integrating TensorRT FP16 engines, and adding Python bindings through libpybev. We used a ResNet50 (FP16) camera backbone. Because the P100 does not support Ampere-compiled SparseConvNet kernels, the pipeline functions primarily as a high-performance camera-centric BEVFusion system, achieving ~10 FPS inference while maintaining ROS 2 compatibility.

V. Task Distribution and Contributions

Sravani: BEVFusion + NuScenes setup on Google Colab, debugging, custom CUDA kernel builds, CUDA latency testing+ inference, testing/metrics, project setup, experimentation on working with jetson, report + presentation work.

Kaushika: BEVFusion + NuScenes setup on HPC server, custom CUDA kernel builds, latency + inference testing/metrics, visualizations, debugging, project setup, report + presentation work

Loi: BEVFusion + NuScenes setup on TeslaP100 with ros2, custom CUDA kernel builds, latency + inference testing/metrics, visualizations, debugging, project setup, report + presentation work

Hau: ROS2 development, implement BEVFusion on Jetson Orin Nano, testing/metrics, visualizations, debugging, project setup, report + presentation work

VI. Evaluation and Results

A. Challenge

The main challenge throughout the project is that the LIDAR branch of BEVFusion fails due to GPU architecture mismatch. The LIDAR processing uses 3rd party 3DSparseConvolution spconv.so, provided in NVIDIA's repo.

However, the library for aarch64_cuda12.8 only contains kernels for sm_101a, while Jetson Orin Nano requires sm_87. We attempted to run the library for aarch64_cuda11.4 in a JetPack 5 container but faced mismatch kernel issues since Jetson uses Jetpack 6 12.6.

Because of incompatible CUDA kernels, the spconv.so library causes a runtime error that leads to binary incompatibility during the Lidar backbone execution. As a result, it may dump garbage values, degrading performance of the BEVFusion pipeline. In addition, the inference time does not reflect the correct time for LIDAR branch, thereby affecting the inference time of the whole pipeline.

Despite this limitation, the system still functions and generates valid detections. However, in order to capture detections that would typically be filtered out, the confidence level must be dropped to 0.01. The fix acknowledges decreased confidence calibration while preserving system operation; despite lower scores, detections are still geometrically accurate. In this section, we attempt to evaluate inference time of running BEVFusion on different hardware. Note that these results underestimate the true latency, as the architecture mismatch in spconv.so prevented full execution of the LIDAR branch.

B. HPC NVIDIA A40 GPU

We ran the BEVFusion model on SJSU's HPC server as a baseline for comparison, using a NVIDIA A40 GPU, with a CUDA version of 12.6, and memory of approximately 48GB. However, the results we got after evaluation were extremely low, to the point where we believe they are inaccurate. We spent a lot of time trying to figure out what was going on and why the inference results were not properly working, but even after trying multiple ways to improve scores, nothing really changed. We believe it may be due to a mismatch between the model configuration and checkpoint loaded, but we were unable to find any other checkpoints to use for inference and therefore had to stick to the one we had. The table below shows the inference and latency results from the A40 HPC inference run (Table 1).

TABLE I
PERFORMANCE OF NUSCENES ON HPC A40 GPU

	Performance
mAP	0.0080
NDS	0.0394
Mean Latency	437.57 ms/frame
FPS	2.29

The figure below is an example of the inference run on the A40 GPU, highlighting some of the predictions that were made (Fig. 3). In addition, Fig. 4 shows some bounding boxes placed on top of a scene. However, it is clear that the predictions were not very accurate, as can be seen in the picture. Nevertheless, the bottom left image reveals a correctly predicted bounding box on top of a car, revealing that the model was able to predict some objects accurately.

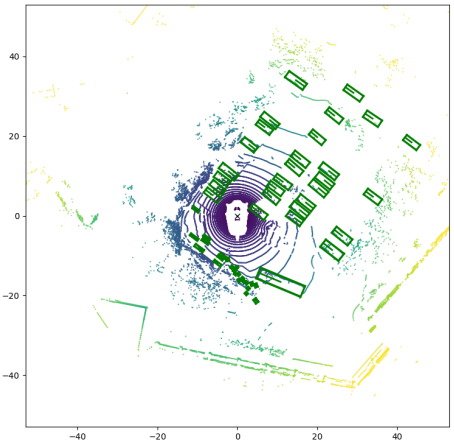

Fig. 5. Example of object predictions from inference on HPC A40 GPU.

Fig. 6. Scene from NuScenes with bounding box predictions from A40 GPU.

C. Google Colab T4 GPU

For the Google Colab run on the Tesla T4 GPU, we got performance metrics on MM3D detection and captured system latency and resource utilization on the server.

We ran the BEVFusion 3D detection model on Google Colab to see how fast and stable it is in practice. Using the small nuScenes-mini dataset, we let the model warm up and then measured how long each frame took to process, how many frames per second it could handle, and how much GPU memory it used. During this process, we also checked that the LiDAR+camera fusion worked correctly, that the custom CUDA operators (like BEV pooling and voxelization) compiled and ran without errors, and that the whole MMDetection3D pipeline could run end-to-end on our setup. We have successfully obtained accurate latency performance metrics for BEVFusion on T4 with a mean latency of approximately 296.94 ms/frame and a median latency of

approximately 297.11 ms over a total of 30 samples, where FPS was 3.36 FPS.

TABLE 2
PERFORMANCE OF NUSCENES ON GOOGLE COLAB T4 GPU

	Performance
Mean Latency	296.94 ms/frame
Median Latency	297.11 ms/frame
FPS	3.36

D. Jetson Orin Nano

We evaluated the BEVFusion inference latency on the Jetson Orin Nano. Preprocess time (time spent preparing sensor data for inference) and publish time (time spent constructing and publishing ROS 2 detection messages, from receiving raw detections to message publication.) are consistent across models. Inference time is the dominant bottleneck and varies significantly by model, with ResNet50int8 (quantized) achieving the lowest inference latency. The average total latency ranges from 130-200ms, achieving 5-8FPS, which are not very high for real time inference.

TABLE 3
PERFORMANCE OF NUSCENES ON JETSON ORIN NANO

Image backbone	Preprocess (ms)	Inference (ms)	Publish (ms)	Total (ms)	FPS
ResNet50	19.05	142.44	4.53	166.01	6.02
Swin Tiny	19.36	173.03	4.23	196.62	5.09
ResNet50int8	18.24	108.03	4.52	130.79	7.65

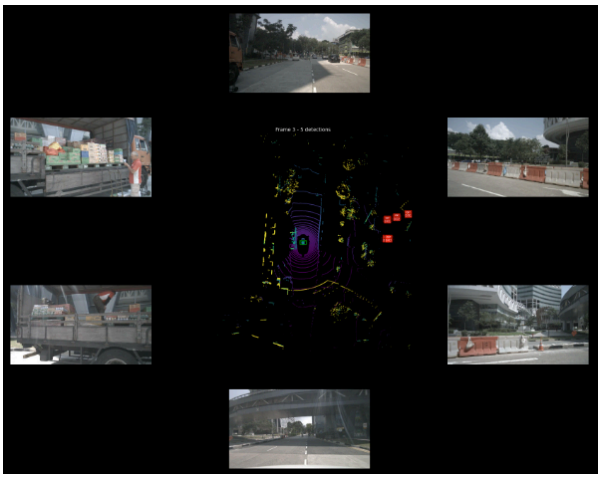

Fig. 7. Visualizing detections on Jetson.

Visualizing the detections of the whole ROS2 pipeline in Jetson, the detection bounding boxes are geometrically accurate, but the detection performance is very poor and inaccurate. This is as expected due to the kernel mismatch issue.

E. Tesla P100

We deployed the CUDA-BEVFusion ROS 2 pipeline inside a custom TensorRT-enabled container on an NVIDIA Tesla P100 GPU to benchmark end-to-end perception performance using the nuScenes-mini dataset. We recorded detailed per-frame latency across the entire pipeline including image preprocessing, LiDAR preparation, TensorRT inference, and ROS 2 message publishing. BEVFusion achieved an average of 103.48 ms per frame (9.66 FPS), with 30.93 ms spent in preprocessing, 69.14 ms in inference, and 3.41 ms in publishing. These results show that, despite lacking tensor cores and being two architectural generations behind Ampere, the Tesla P100 can still sustain near–real-time camera-only BEVFusion throughput within a ROS 2 deployment.

However, detection confidence values were lower compared to runs on Ampere GPUs (e.g., T4, A10, Orin), resulting in few or no 3D bounding boxes meeting the confidence threshold. The low confidence values stem from architectural incompatibilities between the P100 (SM 6.0) and the SparseConvNet (spconv) kernels used for BEVFusion's LiDAR backbone. The spconv build packaged with CUDA-BEVFusion is compiled exclusively for SM $\geq$ 80 while for the P100 was SM = 60, making its CUDA kernels incompatible with Pascal hardware. Consequently, LiDAR voxelization and sparse convolution fail at runtime with errors such as "no kernel image is available for execution on the device," forcing the pipeline to fall back to camera-only feature extraction with incomplete 3D spatial information. This produces extremely low detection scores (≈0.01), causing post-processing to filter out nearly all predicted boxes.

TABLE 4
PERFORMANCE OF NUSCENES ON TESLA P100 GPU

	Performance
Mean Latency	69.14 ms/frame
Avg Total Latency	103.48 ms/frame
FPS	9.66

VII. CONCLUSION

In this project, we successfully created and implemented a BEVFusion pipeline based on ROS 2 on the NVIDIA Jetson Orin Nano. We created a working end-to-end system that can ingest multi-camera and LiDAR data to generate 3D object detections by wrapping the model within a modular ROS 2 package and utilizing TensorRT optimizations.

Our project faced a major architecture mismatch in the spconv library, where CUDA kernels for the LiDAR backbone were not created for SM87, despite the pipeline integration being successful. As a result, the LiDAR feature extraction and overall detection accuracy were very low.

The project shows it is possible to run state of the art models like BEVFusion on edge devices like Jetson Orin Nano. Future developments should focus on addressing the spconv issue to restore full functional accuracy and exploring further optimization techniques to improve real-time performance.

Appendix

All of our code and implementations for this project can be found in our Github Repository at the following link: Evaluation of BEVFusion for 3D Object Detection Repo.

References

[1] C. Teague, "What Lidar Is and Why It's Important for Autonomous Vehicles," *Autoweek*, Apr. 2021. https://www.autoweek.com/news/a36190274/what-lidar-is/

[2] A. Bollu, "LiDAR vs. Camera: Which is the better Security Technology?," *Blickfeld*, Jun. 2025. https://www.blickfeld.com/blog/lidar-vs-camera/

[3] C. R. Qi, L. Yi, H. Su, and Guibas, Leonidas J, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," *arXiv.org*, 2017. https://arxiv.org/abs/1706.02413

[4] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud," *IEEE Xplore*, Jun. 2019. https://ieeexplore.ieee.org/document/8954080

[5] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," *arXiv:1711.06396 [cs]*, Nov. 2017. https://arxiv.org/abs/1711.06396

[6] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast Encoders for Object Detection from Point Clouds," *arXiv:1812.05784 [cs, stat]*, May 2019, Available: https://arxiv.org/abs/1812.05784

[7] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely Embedded Convolutional Detection," *Sensors*, vol. 18, no. 10, p. 3337, Oct. 2018, doi: https://doi.org/10.3390/s18103337

[8] P. Li, X. Chen, and S. Shen, "Stereo R-CNN based 3D Object Detection for Autonomous Driving," *arXiv.org*, 2019. https://arxiv.org/abs/1902.09738

[9] W. Peng, H. Pan, H. Liu, and Y. Sun, "IDA-3D: Instance-Depth-Aware 3D Object Detection From Stereo Vision for Autonomous Driving," *Thecvf.com*, pp. 13015–13024, 2020, Accessed: Dec. 17, 2025. [Online]. doi: 10.1109/CVPR42600.2020.01303

[10] J. Huang, G. Huang, Z. Zhu, Y. Ye, and D. Du, "BEVDet: High-performance Multi-camera 3D Object Detection in Bird-Eye-View," *arXiv.org*, 2021. https://arxiv.org/abs/2112.11790

[11] J. Philion and S. Fidler, "Lift, Splat, Shoot: Encoding Images From Arbitrary Camera Rigs by Implicitly Unprojecting to 3D," *arXiv.org*, 2020. https://arxiv.org/abs/2008.05711

[12] Y. Wang, V. Guizilini, T. Zhang, Y. Wang, H. Zhao, and J. Solomon, "DETR3D: 3D Object Detection from Multi-view Images via 3D-to-2D Queries," *arXiv.org*, 2021. https://arxiv.org/abs/2110.06922

[13] Z. Li *et al.*, "BEVFormer: Learning Bird's-Eye-View Representation from Multi-Camera Images via Spatiotemporal Transformers," *arXiv.org*, 2022, doi: https://doi.org/10.48550/arXiv.2203.17270.

[14] C. R. Qi, W. Liu, C. Wu, H. Su, and Guibas, Leonidas J, "Frustum PointNets for 3D Object Detection from RGB-D Data," *arXiv.org*, 2017. https://arxiv.org/abs/1711.08488

[15] Y. Li *et al.*, "DeepFusion: Lidar-Camera Deep Fusion for Multi-Modal 3D Object Detection," *arXiv:2203.08195 [cs]*, Mar. 2022, Available: https://arxiv.org/abs/2203.08195

[16] X. Chen, T. Zhang, Y. Wang, Y. Wang, and H. Zhao, "FUTR3D: A Unified Sensor Fusion Framework for 3D Detection," *arXiv.org*, 2022. https://arxiv.org/abs/2203.10642

[17] Z. Liu *et al.*, "BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation," *arXiv (Cornell University)*, May 2022, doi: https://doi.org/10.48550/arxiv.2205.13542

[18] open-mmlab, "mmdetection3d/projects/BEVFusion at main · open-mmlab/mmdetection3d," *GitHub*, 2025. https://github.com/open-mmlab/mmdetection3d/tree/main/projects/BEVFusion

[19] Motional, *nuScenes: A multimodal dataset for autonomous driving*, 2025. [Online]. Available: https://nuscenes.org/

[20] OpenMMLab, *MMDetection3D: OpenMMLab Next-Generation Toolbox for 3D Detection*, GitHub repository, 2020. [Online]. Available: https://github.com/open-mmlab/mmdetection3d

[21] NVIDIA-AI-IOT, "GitHub - NVIDIA-AI-IOT/Lidar_AI_Solution: A project demonstrating Lidar related AI solutions, including three GPU accelerated Lidar/camera DL networks (PointPillars, CenterPoint, BEVFusion) and the related libs (cuPCL, 3D SparseConvolution, YUV2RGB, cuOSD,).," *GitHub*, 2025. https://github.com/NVIDIA-AI-IOT/Lidar_AI_Solution