Multimodal Input-Dependent Object Query Initialization for LiDAR-Camera 3D Object Detection with Transformers

MSc Thesis

by

M. R. van Geerenstein

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday September 4, 2023 at 14:00.

Student number:	4598660	
Project duration:	November 14, 2022 – September 4, 2023	
Thesis committee:	Prof. Dr. Dariu M. Gavrila,	TU Delft, supervisor
	Dr. Holger Caesar,	TU Delft
	Dr. Jan C. van Gemert,	TU Delft
	M.Sc. Felicia Ruppel,	Bosch Research, daily supervisor

This thesis is confidential and cannot be made public until September 4, 2023.

TU Delft

Acknowledgement

First of all, I would like to say a big thank you to Prof. Dr. Dariu Gavrila for being my academic supervisor during this thesis, specifically for bringing me in touch with the relevant contacts at Bosch Research, for helping shape the research topic, and for providing valuable feedback to early versions of this document and related presentations.

Second, I'm extremely grateful to M.Sc. Felicia Ruppel at Bosch Research in Stuttgart, who was my daily supervisor during this project and the internship prior to it. She gave me the freedom to shape my own path in this project. The weekly meetings always offered plenty of room for discussion and enabled me to make the most of my academic journey in Stuttgart.

Third, I would like to extend my sincere thanks to Dr. Holger Caesar and Dr. Jan van Gemert, for taking part in the committee that assesses the quality of my research.

Finally, I am also thankful to M.Sc. Folkert de Ronde and Laura Schellekens for proofreading my thesis and helping me push the quality of my work forwards.

M. R. van Geerenstein
Delft, August 23, 2023

Multimodal Input-Dependent Object Query Initialization for LiDAR-Camera 3D Object Detection with Transformers

Mathijs R. van Geerenstein[1,2]
[1]Delft University of Technology [2]Bosch Corporate Research

Abstract

3D object detection models that exploit both LiDAR and camera sensor features are top performers in large-scale autonomous driving benchmarks. A transformer is a popular network architecture used for this task, in which so-called object queries act as candidate objects. Initializing these object queries based on current sensor inputs leads to state-of-the-art performance. Existing methods rely strongly on LiDAR data however, and do not fully exploit image features. Besides, they introduce significant latency.

To overcome these limitations we propose EfficientQ3M, an efficient, modular, and multimodal solution for object query initialization for transformer-based 3D object detection models. Using both the LiDAR and camera modalities as input, we use efficient grid sampling and a lightweight detection head to predict a set of initial object query locations and corresponding query feature vectors. The proposed initialization method is combined with a "modality-balanced" transformer decoder where the queries can access all sensor modalities throughout the decoder.

We achieve state-of-the-art performance for both LiDAR-camera and LiDAR-only sensor setups on the competitive nuScenes benchmark while being up to 15 times more efficient than the closest related method. The proposed initialization can be applied with any combination of sensor modalities as input, demonstrating its modularity.

1. Introduction

3D object detection is a vital part of autonomous driving systems, and the resulting detections serve as a starting point for downstream tasks such as tracking or trajectory prediction. In automotive scenes, we try to predict multi-class 3D bounding boxes for all road users and other important objects around the ego vehicle. Models that exploit multimodal sensor data are currently top performers in popular benchmarks [2, 42] for 3D object detection. The sensor suite typically consists of a roof-mounted LiDAR and a set of monocular cameras, the former providing a sparse 3D point cloud and the latter high-resolution dense images. These two sensor types are complementary: LiDAR brings accurate depth information and the cameras offer texture information and higher resolution for small, far-away objects.

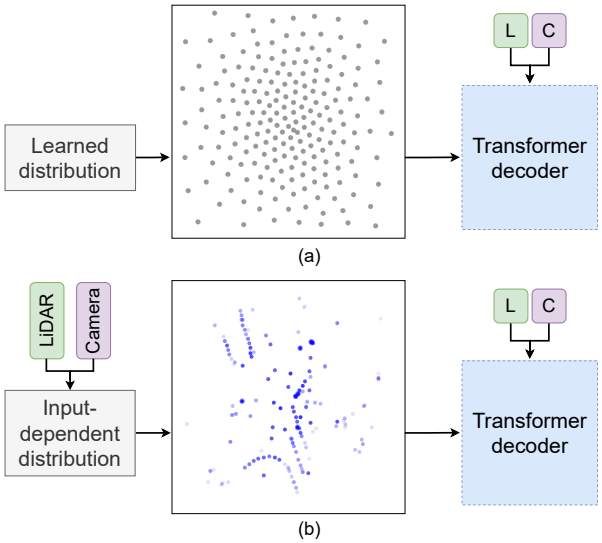

Figure 1. Different query initialization approaches in transformer-based LiDAR-camera object detection. We show the initial object query locations in grey/blue from the bird's eye view perspective. (a) is input-agnostic initialization as in FUTR3D [7] and (b) is the proposed feature-informed initialization.

In recent years, the transformer [44] architecture has successfully been applied to 2D [3, 55, 63] and 3D [1, 7, 33, 39] object detection tasks. Besides performance improvements, their architecture provides a natural extension to the object tracking task [32, 38], and enables end-to-end training because it does not require non-maximum suppression (NMS) post-processing to remove duplicate predictions [3]. Transformers rely on *object queries* to detect objects, where each query is an object candidate that can detect at most one object. A query is essentially a feature vector in a latent space that encodes all information needed to predict a classified bounding box. Usually, each query is accompanied by a reference or anchor point, with respect to which the bounding box is predicted [30]. The initialization of the query feature vectors and their locations is an active research topic, and we distinguish two approaches: learning a fixed distribution for the object queries during training, or initializing them based on the current sensor inputs. In Fig. 1, we show a learned distribution of initial query locations from input-agnostic method FUTR3D [7] (a), and input-dependent initialization using our proposed method (b).

Methods with learned object queries need many queries to sufficiently cover the large grid. Even then, they still may miss objects if many are located in a small area, because there will not be enough queries close by to cover the objects. Input-dependent initialization solves this by placing the queries at locations where we expect to find objects after predictions from a first-stage network. The concept of input-dependent query initialization in itself is not novel: it has been applied in TransFusion [1] for LiDAR-camera 3D detection. This state-of-the-art method has also been used as the query initialization approach for other models like DeepInteraction [53].

We however find two limitations with TransFusion's initialization that lead us to propose a novel method. First, TransFusion takes a sequential approach to sensor fusion, where camera features are only used to refine a set of initial LiDAR-only predictions (Fig. 2a). Although their query locations are predicted from LiDAR and camera features, the corresponding object query feature vectors are initialized only with LiDAR features and their model does therefore not fully exploit the camera features. Second, TransFusion uses an elaborate transformer network solely to fuse LiDAR and camera features into a shared bird's eye view (BEV) space, from which the initial query locations are predicted. This approach adds significant overhead to the model.

To overcome these drawbacks, we propose a new input-aware initialization approach to initialize object queries with both LiDAR and camera features (Fig. 2b) while introducing only minimal computational overhead.

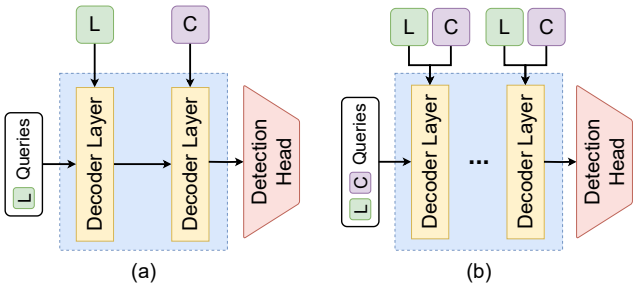

Figure 2. Different approaches to sensor fusion within a transformer decoder. We call (a) sequential fusion, found in TransFusion [1] and (b) modality-balanced fusion in our proposed method.

Our main contributions are as follows. We propose a new input-dependent object query initialization strategy to initialize the query feature vectors with both LiDAR and camera features, which is also able to work with any other combination of sensor modalities. We combine the proposed initialization with a modality-balanced transformer decoder to achieve state-of-the-art performance on the nuScenes [2] benchmark. The proposed initialization outperforms alternative solutions while being up to 15 times more efficient than state-of-the-art method TransFusion [1].

2. Related Work

Our work relates to two main disciplines in 3D object detection: LiDAR-only and LiDAR-camera detection. In addition, we also elaborate specifically on object query initialization for transformer-based object detection models.

2.1. LiDAR-based 3D Detection

For unimodal models, LiDAR-based detectors top the tables of popular benchmarks like nuScenes [2] and Waymo [42]. Most of these detectors quantize the LiDAR point cloud into a regular grid of 3D voxels [50, 60], 2D pillars [20] or perspective range views [4, 12, 43] using convolutional backbones. Others operate directly on the unordered, irregular point cloud [5, 24, 33, 37, 54]. Many detectors adopt anchor boxes in their detection heads [9, 20, 60], where objects are predicted as offsets to these anchor boxes. Alternatively, center-based detectors [56, 61] capture objects as points and predict the 3D bounding box from this center point representation.

After the pioneering work of DETR [3] that applied transformers to 2D detection, transformer models have also found their way to 3D object detection. Some works apply transformers only in the LiDAR backbone for improved feature extraction from any one point cloud representation [11, 31, 35, 40] or to fuse multiple representations [13, 51]. This is challenging from a memory viewpoint as the self-attention operation in the encoder grows quadratically with the number of input tokens (e.g. points, pillars or voxels), the number of which can be in the order of millions for the outdoor scenes in autonomous driving. Because of this, many works adopt only a transformer decoder, with a more traditional convolutional backbone for feature extraction [1, 7, 38, 61]. Contrary to DETR [3], where the model directly predicts the bounding box location in global image coordinates, most transformers for 3D detection predict boxes relative to an *anchor point*. Note that these anchor points are different from anchor boxes discussed above: only the bounding box location is predicted as an offset to the anchor, contrary to both the location and box size. Anchor locations are either fixed and independent of the current input [7], or computed using the current point cloud by a sampling method [33, 38] or center heatmap approach [1, 56, 61].

2.2. LiDAR-Camera 3D Detection

LiDAR-camera sensor fusion is widely used in multi-modal 3D detectors because of the complementary nature of irregular, sparse 3D point clouds and dense, high-resolution, textured 2D images. Only in recent years were such models able to outperform LiDAR-only detectors [45]. Early works mainly adopt proposal-level fusion [6, 18], where proposals are generated in both modalities individually and

then shared to the other(s) by projection. Such methods are unable to exploit the full potential of the two modalities because of the information bottleneck imposed by the low-dimensional proposals. Feature-level approaches show more potential. Following PointPainting [45], other works similarly apply semantic segmentation networks on images to augment point clouds with richer features [14, 46, 48]. These methods are better able to exploit the multimodal features but are more sensitive to feature alignment issues from suboptimal sensor calibration because of the hard association between points and pixels. Finally, there are methods that fuse both modalities into a shared BEV space, either with a direct BEV projection (view transform) of the image pixels [10, 22, 25, 29, 58, 59], or by explicitly *lifting* image pixels into 3D space using projected LiDAR depth information [15, 16, 26, 53, 57].

Within transformer-based models we see two main approaches in LiDAR-camera fusion: those that deploy transformers only as the fusion mechanism for multimodal features [17, 48, 52, 59] and those that use transformers for both sensor fusion and the actual object detection [1, 7, 49, 53].

2.3. Object Query Initialization

Since our work encompasses object query initialization in transformer-based models, we present an overview of related methods in this section. A more general introduction to transformer models can be found in Appendix A.

Many advances in query initialization originate from 2D object detectors in the image domain. In DETR's [3] initial implementation, the object queries are a small set of $M = 100$ learned embeddings. Each query is an abstract feature vector with length $d = 256$, thus we have the object queries $\{\mathbf{q}_i\}_{i=1}^{M} \in \mathbb{R}^d$. Deformable DETR [63] introduces deformable attention and adds positional information to the object queries to improve convergence speed and detection performance. The object queries $\{\mathbf{q}_i\}_{i=1}^{M} \in \mathbb{R}^d$ are now accompanied by their 2D locations $\{\mathbf{c}_i\}_{i=1}^{M} \in \mathbb{R}^2$. In either case, the object queries (and their locations, in deformable DETR) are learned and independent of the current sensor inputs at test time. Efficient DETR [55] applies input-dependent initialization with a region proposal network (RPN) to place object queries in locations with a high likelihood of finding objects.

In 3D object detection, the total area spanned by the scene is very large relative to the size of objects. FUTR3D [7] operates with learned object queries and reference points similar to deformable DETR [63], but uses 3D locations and increases the number of object queries to $M = 900$ to get sufficient coverage of the large space. Other works [33, 38] use farthest point sampling [37] on the input point cloud to evenly spread query locations based on the current input. Here, 3D query locations $\{\mathbf{c}_i\}_{i=1}^{M} \in \mathbb{R}^3$ are sampled from the point cloud, and the corresponding feature vectors $\{\mathbf{q}_i\}_{i=1}^{M} \in \mathbb{R}^d$ are d-dimensional positional embeddings computed from the respective locations. Finally, there are methods [1, 53, 61] that initialize object queries from a predicted BEV center heatmap. In these works, the 2D BEV query locations $\{\mathbf{c}_i\}_{i=1}^{M} \in \mathbb{R}^2$ are taken as the top-M peaks in a predicted heatmap, and the feature vectors $\{\mathbf{q}_i\}_{i=1}^{M} \in \mathbb{R}^d$ are initialized with LiDAR BEV features sampled at the locations. Notably, none of these methods initialize object queries $\mathbf{q}$ with camera features.

3. Methodology

In this work, we propose Efficient3QM, a new multimodal method for input-dependent object query initialization where queries are initialized with both LiDAR and camera features sampled at predicted 3D locations. The proposed initialization method is combined with a modality-balanced transformer decoder (Fig. 2b) following FUTR3D [7]. Our implementation is explained below, and an overview of our proposed model is presented in Fig. 3.

3.1. Multimodal Input-Dependent Initialization

In essence, we adopt a lightweight network to predict bounding boxes from a large set of M_{dense} proposal object queries, and base the initial object queries for the transformer decoder on the top-M proposals. We choose M_{dense} to be much larger than M. In this way, we have a sufficient number of proposals to not miss any objects and still keep the number of queries M in the decoder manageable. The proposed method is explained below in more detail.

We create a dense grid of query proposal locations $\mathcal{C} \in \mathbb{R}^{X \times Y \times 1}$, spread out uniformly over the detection range in the (x, y) direction. Each 2D location in grid $\mathcal{C}$ is assigned the same fixed height to get 3D query locations. We now have a set of $M_{dense} = X \cdot Y$ initial query proposal locations, which are not dependent on the current sensor inputs. For all M_{dense} proposals, we sample sensor features at instance level from the given sensor feature maps after computing the respective view projections (Fig. 3 ①) using available intrinsic and extrinsic sensor parameters. The result is the corresponding query feature vectors. Queries are then made location-aware by adding a positional embedding based on their location using sine encoding following DETR [3]. The query feature vectors now contain the information needed to predict a 3D bounding box relative to their respective locations. For all M_{dense} query proposals we predict a classified bounding box, using the same regression head Φ_{reg} and classification head Φ_{cls} found later in the transformer decoder layers. From all proposal bounding boxes, we select the top-M with the highest confidence scores, and let the queries from which they are predicted be our set of initial object queries.

For each query $\mathbf{q}_i$ out of the M selected queries, the location $\mathbf{c}_i$ is updated with the predicted 3D offset $\Delta\mathbf{x}_i$

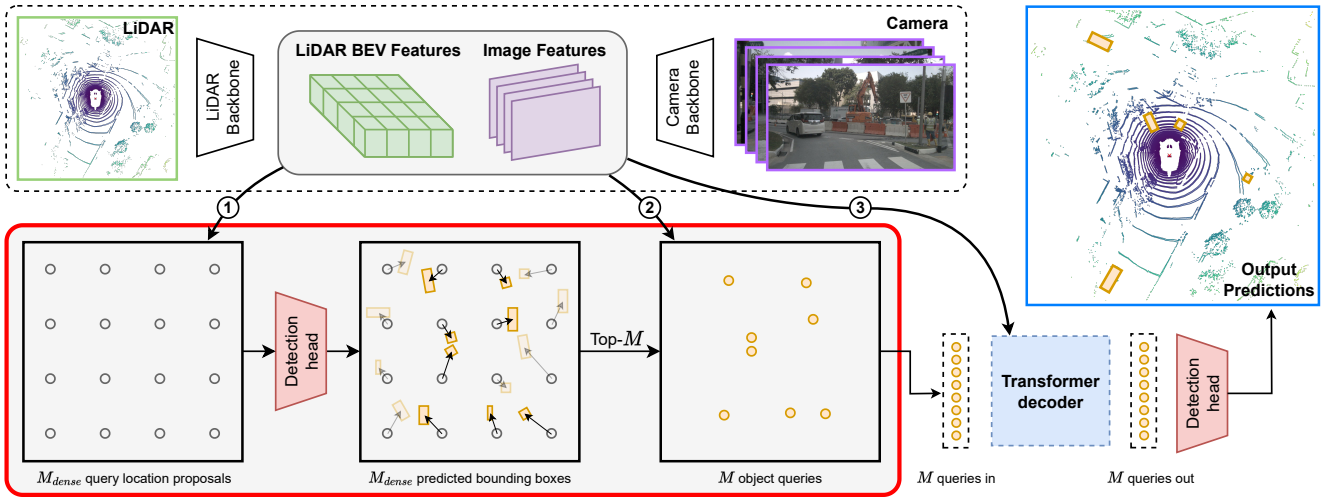

Figure 3. Overview of EfficientQ3M, with the main contribution framed in **red**. We start with a fixed grid $\mathcal{C}$ of M_{dense} query location proposals. We sample LiDAR and camera features at instance level for each proposal ①, and predict a bounding box relative to the grid location. The 3D xyz centers of the top-M bounding boxes with the highest confidence scores are selected as the set of initial object query locations. We re-sample LiDAR and camera features for these M object queries ② and pass them to the modality-balanced decoder, where the queries have access to both sensor modalities in each layer of the decoder ③. A shared regression and classification head is used to produce the final detections from the object queries at the output of the decoder.

from the regression head like $\mathbf{c}'_i = \mathbf{c}_i + \Delta\mathbf{x}_i$. We finally generate new query feature vectors from the updated locations, where the query feature vector $\mathbf{q}_i$ is generated by re-sampling features at the new location $\mathbf{c}'_i$ (Fig. 3 ②). This results in the object query feature vectors $\{\mathbf{q}_i\}_{i=1}^{M} \in \mathbb{R}^d$ and their 3D locations $\{\mathbf{c}'_i\}_{i=1}^{M} \in \mathbb{R}^3$, which we pass to the decoder. In the decoder, the object queries interact through self-attention, and have access to the sensor features from all modalities in cross-attention (Fig. 3 ③).

Because we predict a 3D offset $\Delta\mathbf{x}_i$ with Φ_{reg}, we deem it not necessary to initialize a 3D dense grid $\mathcal{C} \in \mathbb{R}^{X\times Y\times Z}$ with multiple heights as proposals.

Modular and Multimodal Our initialization method is modular because it can work with any sensor combination, like camera-only, camera-RADAR, LiDAR-only and LiDAR-camera, similar to the modality-balanced decoder in FUTR3D [7]. The initial grid $\mathcal{C}$ with the proposal locations is identical in each case, but the modalities of features sampled at the locations will differ.

Sensor features are sampled from the corresponding feature map around the projected query location. If there are multiple sensor modalities available, we concatenate the sampled features from both and fuse them as in Eq. (1) for LiDAR-camera fusion. Here, $\mathcal{SF}^i_{\text{lid}}$ are the sampled LiDAR features, $\mathcal{SF}^i_{\text{cam}}$ the camera features, Φ_{fus} the fusion multi-layer perceptron (MLP) and $\mathcal{SF}^i_{\text{fus}}$ is the fused feature vector for query $\mathbf{q}_i$. When there is only one sensor modality available (e.g. LiDAR-only), Φ_{fus} is simply a linear projection since the dimension of the sampled features is already equal to that of the object query feature vector $\mathbf{q}_i$.

$$\mathcal{SF}^i_{\text{fus}} = \Phi_{\text{fus}}\left(\mathcal{SF}^i_{\text{lid}} \oplus \mathcal{SF}^i_{\text{cam}}\right) \tag{1}$$

In contrast to TransFusion [1], where an elaborate transformer network is used to create a shared LiDAR-camera heatmap to initialize the object queries, our implementation is lightweight and straightforward. On top of that, it is not specific to any sensor suite and may be used with a variety of sensors as input.

Model Details The implementation of the backbones, transformer decoder, and final detection head –i.e. all components outside of the red frame in Fig. 3– follows FUTR3D [7]. Details on the complete model, outside of the object query initialization, can be found in Appendix B. Our initialization method may be applied to other decoder implementations with any combination of sensor modalities, as long as there exist transformations from global 3D coordinates to the respective sensor feature map coordinates.

3.2. Losses

We supervise the model in three locations, starting with the object query initialization. As explained in Sec. 3.1, we predict a large set of M_{dense} bounding boxes and initialize our object queries from the top-M with the highest confidence score. To supervise the M_{dense} bounding boxes, we perform bipartite matching between the ground truth objects and all M_{dense} predicted boxes to get a set of

one-to-one matches. The Hungarian algorithm [19] is used to produce these matches. The associated matching cost is a weighted sum of classification and regression costs:

$$C_{match} = \lambda_1 L_{cls}(p, \hat{p}) + \lambda_2 L_{reg}(b, \hat{b}) \quad (2)$$

where (p, b) are the predicted class confidence scores and bounding box parameters and $(\hat{p}, \hat{b})$ the corresponding supervisory signals, L_{cls} is the focal loss [28] and L_{reg} the L1 regression loss, and λ_1, λ_2 are the corresponding weights. From the set of matched predictions, we compute the classification loss (focal loss) and regression loss (L1 loss).

Additionally, we supervise all M_{dense} predictions with a dense heatmap to improve convergence, because the number of matched predictions is much smaller than the number of proposals M_{dense}. For this, we take the class confidence scores from all predictions as a class-specific dense heatmap $\mathcal{S} \in \mathbb{R}^{X \times Y \times K}$, which is supervised by a ground truth heatmap $\hat{\mathcal{S}} \in \mathbb{R}^{X \times Y \times K}$ with the penalty reduced focal loss. Here, $X \times Y$ defines the spatial dimension of the heatmap, which matches our dense grid of query locations $\mathcal{C} \in \mathbb{R}^{X \times Y}$. We take the K confidence scores for each prediction from grid $\mathcal{C}$ to obtain heatmap $\mathcal{S}$. The ground truth heatmap is computed following CenterPoint [56]. We find that without this dense loss term, our initialization method does not converge.

Next, the predicted bounding boxes at the output of each transformer decoder layer are supervised as in FUTR3D [7]. The classification and regression loss are computed after each decoder layer. Finally, because sparse supervision can hinder learning in transformer-based models [64], we follow FUTR3D and implement an auxiliary detection head parallel to the transformer decoder for improved supervision of the LiDAR backbone. This CenterPoint [56] head is only used to help the LiDAR backbone learn better features during training, and is removed at test time.

4. Experiments

In this section we elaborate on the chosen dataset, performance metrics, and hyperparameter settings before presenting the main results and ablation studies.

4.1. Dataset and Metrics

We use nuScenes [2] to evaluate our model. nuScenes is a large-scale autonomous driving dataset with 3D object annotations. Its sensor suite consists of a single 360-degree 32-beam LiDAR and 6 cameras outputting multi-view images, each with a resolution of 1600×900. RADAR is also available, but not used in this work. The dataset is a collection of 1000 *scenes*, each 20 seconds long and annotated at 2 Hz. Objects are annotated with 3D bounding boxes with a location (x, y, z), size (w, h, l), yaw rotation (φ), velocity (v_x, v_y) and a class label out of $K = 10$ object classes. Ground truth annotations of the train and validation set are publicly available. To evaluate on the test set, predictions have to be submitted to a test server.

Performance is measured with the popular mean average precision (mAP) metric and the custom nuScenes detection score (NDS). nuScenes uses the BEV center distance between a prediction and ground truth object as the threshold for matching, instead of IoU. The AP is calculated for each distance threshold for all ten classes, and then averaged to obtain the mAP score. The NDS additionally measures the quality of true positives. It is a weighted average of mAP and other metrics that measure translation, scale, orientation, velocity and box attribute errors of true positives.

4.2. Implementation Details

Our implementation is written using PyTorch [36] in the open-source MMDetection3D [34] framework. The code is built on top of the public release of FUTR3D [7], and we follow their model settings for all hyperparameters. The most important model settings are listed below. Other implementation details like the training schedule and the augmentation strategy are presented in Appendix C.1.

Model Settings The LiDAR backbone is a VoxelNet [50, 60] with a voxel size of $(0.075\,\text{m}, 0.075\,\text{m}, 0.2\,\text{m})$. The image backbone is a VoVNET [21] and is pretrained on nuScenes [2] using the camera-only version of FUTR3D. The detection range is $[-54\,\text{m}, 54\,\text{m}]$ for the X, Y grid and $[-5\,\text{m}, 3\,\text{m}]$ for the Z axis. The number of levels in the feature pyramid networks of the LiDAR and camera backbones is $m = 4$. The channel dimension of the sensor feature maps and the object queries is $d = 256$. The number of sampling offsets in the deformable attention is $V = 4$. Like in nuScenes, the number of camera images is $N = 6$ and the number of object classes is $K = 10$. The number of object query proposals is $M_{dense} = 3600$, spread out uniformly over the X, Y detection range defined above. We report results for both $M = 200$ and $M = 900$, with M the number of object queries that are input to the decoder.

4.3. Main Results

The detection performance is evaluated on nuScenes and the results are presented in Tab. 1. The proposed method outperforms the FUTR3D [7] baseline for both LiDAR-only detection and LiDAR-camera fusion, and achieves state-of-the-art performance on the nuScenes *val* set.

Concerning the test set, we differentiate between two different scores obtained by the FUTR3D baseline: those stated in their paper are listed first, while those obtainable with their publicly available model weights are marked with an asterisk. This discrepancy is likely caused by a difference in training strategy. FUTR3D's top results are obtained after training on the nuScenes training *and* validation set,

Method	Modality	Backbone		*validation*		*test*	
		Camera	LiDAR	mAP ↑	NDS ↑	mAP ↑	NDS ↑
CenterPoint [56]	L	-	VoxelNet	59.6	66.8	60.3	67.3
TransFusion-L [1]	L	-	VoxelNet	**65.1**	**69.9**	65.5	70.2
FUTR3D [7]	L	-	VoxelNet	63.7	69.0	65.3 / 64.6*	69.9 / 69.5*
EfficientQ3M (ours)	L	-	VoxelNet	**65.3**	**69.6**	65.4	69.7
PointAugmenting [46]	L+C	DLA34	VoxelNet	-	-	66.8	71.0
TransFusion [1]	L+C	DLA34	VoxelNet	67.3	70.9	68.9	71.6
DeepInteraction [53]	L+C	R50	VoxelNet	69.9	72.6	70.8	73.4
FUTR3D [7]	L+C	VoVNet	VoxelNet	**70.3**	**73.1**	69.4 / 68.0*	72.1 / 71.1*
EfficientQ3M (ours)	L+C	VoVNet	VoxelNet	**71.2**	**73.5**	69.2	71.7

Table 1. Comparison to the state of the art on the nuScenes *validation* and *test* set. The best and second best scores for each sensor modality are in **red** and **blue** respectively. All LiDAR backbones use the same voxel size of (0.075, 0.075, 0.2) meters. We highlight the FUTR3D baseline and the proposed method in gray. None of the listed methods use test-time augmentation or model ensemble.

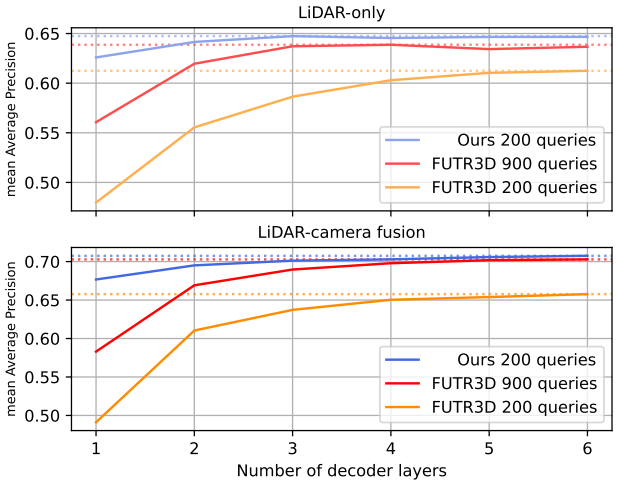

Figure 4. Detection performance over the number of decoder layers on the nuScenes *val* set. The top figure compares LiDAR-only versions of our proposed method with 200 queries to the FUTR3D [7] baseline with both 900 and 200 queries. The bottom shows the same comparison but then for LiDAR-camera fusion.

while those marked with an asterisk are not. Our proposed method is trained only on the training set, and is able to outperform the corresponding* FUTR3D scores.

We find that the proposed input-dependent object query initialization not only produces superior detection scores, but also enables the use of fewer object queries and decoder layers. In Fig. 4, we compare our model to FUTR3D with a varying number of decoder layers. Where FUTR3D needs multiple passes through the decoder to achieve high performance, the input-dependent initialization already shows good performance for a single-layer model. The proposed method with 200 object queries outperforms the baseline with 900 queries for any number of decoder layers. For LiDAR-only detection (Fig. 4, top), the improvement on the baseline is $+1.0$ mAP for 6 decoder layers and increases to $+6.5$ mAP for 1 decoder layer. For LiDAR-camera fusion we see similar results (Fig. 4, bottom). Improvements on FUTR3D are $+0.5$ mAP for the 6-layer model and $+9.4$ mAP for the single-layer model.

Init. Method	Mod.	mAP ↑	Lat. (ms) ↓	# (M)
Learned distr.	L	63.7	208.6	7.29
w/ TransFusion	L	64.5 (+0.8)	212.7 (+2.0%)	7.89
w/ ours	L	64.7 (+1.0)	209.5 (+0.4%)	7.51
Learned distr.	L+C	70.3	645.4	9.32
w/ TransFusion	L+C	70.3 (+0.0)	751.1 (+16.4%)	11.79
w/ ours	L+C	70.8 (+0.5)	652.3 (+1.1%)	9.80

Table 2. Comparisson of different query initialization methods paired with the modality-balanced decoder on the nuScenes *val* set. We take FUTR3D's [7] learned distribution with 900 queries as the baseline and compare it to TransFusion [1] and our proposed method with 200 queries. The latency is the average duration of one complete forward pass, measured on a V100 GPU. # is the number of parameters in millions, excluding the backbones.

In Tab. 2 we compare the proposed method to the initialization of the closest related work, TransFusion's [1], with FUTR3D's [7] input-agnostic approach included for reference. Model settings such as the backbones, transformer decoder, and detection head are equal, only the object query initialization method is varied. For TransFusion, we implement their 2D query initialization and naively append a fixed z coordinate to make it 3D, to be able to initialize the query feature vectors with both LiDAR and camera features.

We find that both strategies achieve similar improvements on the baseline in the LiDAR-only setting, but that the proposed method is superior for LiDAR-camera fusion. We contribute this to the benefit of directly predicting a 3D query location, with which the relevant image features can be sampled already at initialization. Additionally, we find that the proposed method is lightweight and efficient compared to TransFusion. Especially for LiDAR-camera

Figure 5. Two examples of output predictions with the proposed model on the nuScenes *val* set. We highlight difficult true positives in the top sample and false positives in the bottom sample. The LiDAR BEV shows ground truth objects in dark green. Best viewed zoomed in.

fusion, our method remains efficient with only $+6.9$ ms of added latency while TransFusion's introduces $+105.7$ ms of overhead compared to a model with learned queries (i.e. a $15\times$ difference, highlighted in gray). This is explained by the elaborate transformer network used in their query initialization, the size of which also shows when looking at the number of model parameters.

Qualitative Analysis Fig. 5 shows two examples of detections made with the proposed method. The top sample highlights three successful detections in a difficult setting (1.A-1.C), i.e. with rain and strong occlusions. The bottom sample includes mistakes, with a duplicate detection (2.A), and two false positives where traffic signs are detected as pedestrians (2.B and 2.C). The false positive of 2.A is a rare example of a duplicate prediction that would have been prevented with NMS post-processing. We provide additional qualitative analysis in Appendix D.1.

The benefits of the proposed method are summarized as follows. Our proposed input-dependent query initialization combined with a modality-balanced decoder achieves strong performance on the nuScenes benchmark, outperforming related works like TransFusion [1] and FUTR3D [7]. The initialization method adds minimal overhead compared to employing a set of learned object queries, as shown in Tab. 2. Because our method places the initial query locations already close to the objects, we are able to decrease the number of queries and decoder layers and still achieve better performance than FUTR3D.

4.4. Ablation Studies

We evaluate our design choices by performing ablation studies on the nuScenes validation set. First, we consider the initialization of the query location and the query feature vector separately, before conducting experiments on the number of object queries. Additional experiments can be found in Appendix C.2 and C.3.

Query Initialization We test if it suffices to initialize the query feature vectors with positional embeddings based on their location, or if we instead need to initialize them with sensor features sampled at their location. For this, we train model variants on a reduced schedule for 6 epochs on half of the training set with $M = 200$ object queries, see Tab. 3.

We find that the best performance is achieved with the proposed initialization (**w/ refs + feats**) with 6 decoder layers. This brings a meaningful improvement of $+3.4$ mAP compared to the input-agnostic initialization. When we de-

	R.	F.	#L.	mAP ↑	NDS ↑
Learned distr.			6	60.9 ± 0.2	67.1 ± 0.1
w/ refs	✓		6	64.3 ± 0.1	**69.1** ± 0.1
w/ refs + feats	✓	✓	6	**64.3** ± 0.1	69.0 ± 0.0
Learned distr.			1	48.2 ± 0.2	56.8 ± 0.2
w/ refs	✓		1	61.8 ± 0.2	67.0 ± 0.2
w/ refs + feats	✓	✓	1	**62.2** ± 0.1	**67.3** ± 0.1

Table 3. Ablation on object query initialization components on the nuScenes *val* set, for LiDAR-only. We run the training for each experiment three times with different random seeds and report the mean and standard deviation. **#L.** is the number of decoder layers. **R.** is whether the query locations are predicted with the proposed method, or are a learned distribution. **F.** is whether the query vectors are initialized with LiDAR features at their location, or with a positional embedding based on their location. Without either, we have learned input-agnostic object queries like in FUTR3D [7].

crease the number of decoder layers to 1, we see the improvement on the baseline increase. The baseline performance drops heavily because it needs multiple layers to iteratively update the query location if it is not located close to an object at initialization. Notably, our approach only shows a relatively small drop of 2.1 mAP. With only a single decoder layer, we now also see the benefit of initializing the query vectors with sensor features. This results in a small performance increase compared to using the positional embedding, for the single-layer model. Consequently, our proposed initialization method uses sensor features as the query feature vectors.

Number of Queries One key hyperparameter of transformer-based models in object detection is the number of object queries M. FUTR3D [7] uses $M = 900$ to get sufficient coverage of the large 3D space. With our input-dependent query initialization, we are able to use fewer queries and still have them located close to the objects in the scene. We compare our method with FUTR3D for 200 and 900 queries, where we fine-tune FUTR3D's pretrained model to learn a new distribution with $M = 200$. The results are shown in Tab. 4.

	Mod.	# 200	# 900
FUTR3D	L	61.3 (-2.4)	63.7
FUTR3D	L+C	65.8 (-4.5)	70.3
EfficientQ3M (ours)	L	64.7 (-0.6)	65.3
EfficientQ3M (ours)	L+C	70.8 (-0.4)	71.2

Table 4. Detection performance (mAP ↑) on the nuScenes *val* set for the number of object queries $M = 200$ and 900. The proposed method with 200 queries already outperforms the FUTR3D [7] baseline with 900 queries for both sensor setups.

We find that the proposed method can still achieve strong performance even with many fewer queries, thanks to the input-dependent initialization. Specifically, our method with 200 queries outperforms FUTR3D with 900 queries.

We still see a marginal benefit of using 900 queries with the proposed input-dependent initialization method, even though 200 queries are enough to cover the maximum number of objects in nuScenes. We hypothesize this to be caused by the larger self-attention operation in the decoder which allows for querying more information, and by the additional queries compensating for imperfect initialization.

5. Discussion and Conclusion

We introduced EfficientQ3M, a novel and efficient approach for initializing object queries in transformer-based 3D object detection models from any sensor modality. Existing methods primarily rely on a LiDAR-only first stage, which limited the model's recall to that of the LiDAR-only stage. EfficientQ3M overcomes this limitation by initializing object queries with features from any combination of sensor modalities. The proposed method, when combined with a modality-balanced transformer decoder, achieves state-of-the-art performance on both LiDAR-only and LiDAR-camera sensor setups in the nuScenes benchmark. Additionally, EfficientQ3M demonstrates significant efficiency gains, being up to 15 times more efficient than the closest related method. Furthermore, the modularity of the proposed method allows its application to other transformer decoders as well.

One point of discussion is the general benefit of input-dependent initialization. It is expected that such initialization enables a reduction of the number of object queries and the number of transformer decoder layers while achieving similar performance to a model with input-agnostic object queries (i.e. improving efficiency). We however find that there is also a performance improvement, even though the input-agnostic queries in FUTR3D [7] should be able to detect the same objects. We hypothesize that this advantage is caused by densely populated areas. The learned queries in FUTR3D cover 13 m^2 on average. If there are many objects in a small area, there will not be enough queries around to detect all of them. The proposed method is less sensitive to this problem.

For future work, it would be interesting to extend the scope of the experiments to obtain results for more sensor setups such as RADAR-camera and camera-only. Additionally, the proposed initialization can be applied to different decoder designs like DeepInteraction [53]. Finally, an interesting direction for research would be to investigate the use of a soft-association sensor fusion mechanism for the query feature vectors. The current implementation concatenates the sampled features and fuses them with an MLP, but a form of attention may also be a good option.

References

[1] Xuyang Bai, Zeyu Hu, Xinge Zhu, Qingqiu Huang, Yilun Chen, Hangbo Fu, and Chiew-Lan Tai. TransFusion: Robust LiDAR-Camera Fusion for 3D Object Detection with Transformers. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1080–1089, New Orleans, LA, USA, June 2022. IEEE. 1, 2, 3, 4, 6, 7, 12, 13, 14, 15, 16

[2] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A Multimodal Dataset for Autonomous Driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, Seattle, WA, USA, June 2020. IEEE. 1, 2, 5

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-End Object Detection with Transformers. In *Computer Vision – ECCV 2020*, volume 12346, pages 213–229. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science. 1, 2, 3, 12

[4] Yuning Chai, Pei Sun, Jiquan Ngiam, Weiyue Wang, Benjamin Caine, Vijay Vasudevan, Xiao Zhang, and Dragomir Anguelov. To the Point: Efficient 3D Object Detection in the Range Image with Graph Convolution Kernels. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15995–16004, Nashville, TN, USA, June 2021. IEEE. 2

[5] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, Honolulu, HI, July 2017. IEEE. 2

[6] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3D Object Detection Network for Autonomous Driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6526–6534, Honolulu, HI, July 2017. IEEE. 2

[7] Xuanyao Chen, Tianyuan Zhang, Yue Wang, Yilun Wang, and Hang Zhao. FUTR3D: A Unified Sensor Fusion Framework for 3D Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 172–181, 2023. 1, 2, 3, 4, 5, 6, 7, 8, 12, 13, 16

[8] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, Dec. 2022. 16

[9] Jiajun Deng, Shaoshuai Shi, Peiwei Li, Wengang Zhou, Yanyong Zhang, and Houqiang Li. Voxel R-CNN: Towards High Performance Voxel-based 3D Object Detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(2):1201–1209, May 2021. Number: 2. 2

[10] Florian Drews, Di Feng, Florian Faion, Lars Rosenbaum, Michael Ulrich, and Claudius Gläser. DeepFusion: A Robust and Modular 3D Object Detector for Lidars, Cameras and Radars, Sept. 2022. Issue: arXiv:2209.12729 arXiv:2209.12729 [cs]. 3

[11] Lue Fan, Ziqi Pang, Tianyuan Zhang, Yu-Xiong Wang, Hang Zhao, Feng Wang, Naiyan Wang, and Zhaoxiang Zhang. Embracing Single Stride 3D Object Detector with Sparse Transformer. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8448–8458, New Orleans, LA, USA, June 2022. IEEE. 2

[12] Lue Fan, Xuan Xiong, Feng Wang, Naiyan Wang, and Zhaoxiang Zhang. RangeDet: In Defense of Range View for LiDAR-based 3D Object Detection. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2898–2907, Montreal, QC, Canada, Oct. 2021. IEEE. 2

[13] Tianrui Guan, Jun Wang, Shiyi Lan, Rohan Chandra, Zuxuan Wu, Larry Davis, and Dinesh Manocha. M3DETR: Multi-Representation, Multi-Scale, Mutual-Relation 3D Object Detection With Transformers. pages 772–782, 2022. 2, 12

[14] Tengteng Huang, Zhe Liu, Xiwu Chen, and Xiang Bai. EPNet: Enhancing Point Features with Image Semantics for 3D Object Detection. In *Computer Vision – ECCV 2020*, volume 12360, pages 35–52. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science. 3

[15] Philip Jacobson, Yiyang Zhou, Wei Zhan, Masayoshi Tomizuka, and Ming C. Wu. Center Feature Fusion: Selective Multi-Sensor Fusion of Center-based Objects, Sept. 2022. Issue: arXiv:2209.12880 arXiv:2209.12880 [cs]. 3

[16] Yang Jiao, Zequn Jie, Shaoxiang Chen, Jingjing Chen, Lin Ma, and Yu-Gang Jiang. MSMDFusion: A Gated Multi-Scale LiDAR-Camera Fusion Framework with Multi-Depth Seeds for 3D Object Detection, Nov. 2022. Issue: arXiv:2209.03102 arXiv:2209.03102 [cs]. 3

[17] Yecheol Kim, Konyul Park, Minwook Kim, Dongsuk Kum, and Jun Won Choi. 3D Dual-Fusion: Dual-Domain Dual-Query Camera-LiDAR Fusion for 3D Object Detection, Nov. 2022. Issue: arXiv:2211.13529 arXiv:2211.13529 [cs]. 3, 12

[18] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L. Waslander. Joint 3D Proposal Generation and Object Detection from View Aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8, Oct. 2018. ISSN: 2153-0866. 2

[19] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. 5

[20] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. PointPillars: Fast Encoders for Object Detection From Point Clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12689–12697, Long Beach, CA, USA, June 2019. IEEE. 2

[21] Youngwan Lee, Joong-won Hwang, Sangrok Lee, Yuseok Bae, and Jongyoul Park. An Energy and GPU-Computation Efficient Backbone Network for Real-Time Object Detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 752–760, Long Beach, CA, USA, June 2019. IEEE. 5, 13

[22] Yanwei Li, Yilun Chen, Xiaojuan Qi, Zeming Li, Jian Sun, and Jiaya Jia. Unifying Voxel-based Representation with

Transformer for 3D Object Detection, Oct. 2022. Issue: arXiv:2206.00630 arXiv:2206.00630 [cs]. 3, 12

[23] Yingwei Li, Adams Wei Yu, Tianjian Meng, Ben Caine, Jiquan Ngiam, Daiyi Peng, Junyang Shen, Yifeng Lu, Denny Zhou, Quoc V. Le, Alan Yuille, and Mingxing Tan. DeepFusion: Lidar-Camera Deep Fusion for Multi-Modal 3D Object Detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17161–17170, New Orleans, LA, USA, June 2022. IEEE. 12

[24] Zhichao Li, Feng Wang, and Naiyan Wang. LiDAR R-CNN: An Efficient and Universal 3D Object Detector. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7542–7551, Nashville, TN, USA, June 2021. IEEE. 2

[25] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep Continuous Fusion for Multi-sensor 3D Object Detection. In *Computer Vision – ECCV 2018*, volume 11220, pages 663–678. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science. 3

[26] Tingting Liang, Hongwei Xie, Kaicheng Yu, Zhongyu Xia, Zhiwei Lin, Yongtao Wang, Tao Tang, Bing Wang, and Zhi Tang. BEVFusion: A Simple and Robust LiDAR-Camera Fusion Framework, Nov. 2022. Issue: arXiv:2205.13790 arXiv:2205.13790 [cs]. 3

[27] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, Honolulu, HI, July 2017. IEEE. 12, 13

[28] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal Loss for Dense Object Detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2980–2988, 2017. 5

[29] Zhijian Liu, Haotian Tang, Alexander Amini, Xinyu Yang, Huizi Mao, Daniela Rus, and Song Han. BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation, June 2022. Issue: arXiv:2205.13542 arXiv:2205.13542 [cs]. 3

[30] Jiageng Mao, Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. 3D Object Detection for Autonomous Driving: A Comprehensive Survey. *International Journal of Computer Vision*, 131(8):1909–1963, Aug. 2023. 1

[31] Jiageng Mao, Yujing Xue, Minzhe Niu, Haoyue Bai, Jiashi Feng, Xiaodan Liang, Hang Xu, and Chunjing Xu. Voxel transformer for 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3164–3173. IEEE, 2021. 2

[32] Tim Meinhardt, Alexander Kirillov, Laura Leal-Taixé, and Christoph Feichtenhofer. TrackFormer: Multi-Object Tracking With Transformers. pages 8844–8854, 2022. 1

[33] Ishan Misra, Rohit Girdhar, and Armand Joulin. An End-to-End Transformer Model for 3D Object Detection. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2886–2897, Montreal, QC, Canada, Oct. 2021. IEEE. 1, 2, 3, 12

[34] MMDetection3D Contributors. OpenMMLab's Next-generation Platform for General 3D Object Detection, July 2020. Programmers: _:n1491. 5

[35] Xuran Pan, Zhuofan Xia, Shiji Song, Li Erran Li, and Gao Huang. 3D Object Detection With Pointformer. pages 7463–7472, 2021. 2

[36] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. *31st Conference on Neural Information Processing Systems (NIPS)*, (NeurIPS-W), 2017. Number: NeurIPS-W. 5, 16

[37] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 2, 3

[38] Felicia Ruppel, Florian Faion, Claudius Gläser, and Klaus Dietmayer. Transformers for Multi-Object Tracking on Point Clouds. In *2022 IEEE Intelligent Vehicles Symposium (IV)*, pages 852–859, June 2022. arXiv:2205.15730 [cs]. 1, 2, 3, 12

[39] Felicia Ruppel, Florian Faion, Claudius Gläser, and Klaus Dietmayer. Transformers for Object Detection in Large Point Clouds. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 832–838, Oct. 2022. 1, 13

[40] Hualian Shenga, Sijia Cai, Yuan Liu, Bing Deng, Jianqiang Huang, Xian-Sheng Hua, and Min-Jian Zhao. Improving 3D Object Detection with Channel-wise Transformer. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2723–2732, Montreal, QC, Canada, Oct. 2021. IEEE. 2

[41] Leslie N. Smith. Cyclical Learning Rates for Training Neural Networks, Apr. 2017. Issue: arXiv:1506.01186 arXiv:1506.01186 [cs]. 13

[42] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2443–2451, Seattle, WA, USA, June 2020. IEEE. 1, 2

[43] Pei Sun, Weiyue Wang, Yuning Chai, Gamaleldin Elsayed, Alex Bewley, Xiao Zhang, Cristian Sminchisescu, and Dragomir Anguelov. RSN: Range Sparse Net for Efficient, Accurate LiDAR 3D Object Detection. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5721–5730, Nashville, TN, USA, June 2021. IEEE. 2

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 1, 12

[45] Sourabh Vora, Alex H. Lang, Bassam Helou, and Oscar Beijbom. PointPainting: Sequential Fusion for 3D Object Detection. In *2020 IEEE/CVF Conference on Computer Vision*

and Pattern Recognition (CVPR), pages 4603–4611, Seattle, WA, USA, June 2020. IEEE. 2, 3

[46] Chunwei Wang, Chao Ma, Ming Zhu, and Xiaokang Yang. PointAugmenting: Cross-Modal Augmentation for 3D Object Detection. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11789–11798, Nashville, TN, USA, June 2021. IEEE. 3, 6, 13

[47] Yue Wang, Tianyuan Zhang, Hang Zhao, Vitor Guizilini, Yilun Wang, and Justin Solomon. DETR3D: 3D Object Detection from Multi-view Images via 3D-to-2D Queries. 13

[48] Shaoqing Xu, Dingfu Zhou, Jin Fang, Junbo Yin, Zhou Bin, and Liangjun Zhang. FusionPainting: Multimodal Fusion with Adaptive Attention for 3D Object Detection. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3047–3054, Sept. 2021. 3, 12

[49] Xinli Xu, Shaocong Dong, Lihe Ding, Jie Wang, Tingfa Xu, and Jianan Li. FusionRCNN: LiDAR-Camera Fusion for Two-stage 3D Object Detection, Sept. 2022. Issue: arXiv:2209.10733 arXiv:2209.10733 [cs]. 3

[50] Yan Yan, Yuxing Mao, and Bo Li. SECOND: Sparsely Embedded Convolutional Detection. *Sensors*, 18(10):3337, Oct. 2018. Number: 10 Publisher: Multidisciplinary Digital Publishing Institute. 2, 5, 12, 13

[51] Honghui Yang, Wenxiao Wang, Minghao Chen, Binbin Lin, Tong He, Hua Chen, Xiaofei He, and Wanli Ouyang. PVT-SSD: Single-Stage 3D Object Detector With Point-Voxel Transformer. 2

[52] Yanlong Yang, Jianan Liu, Tao Huang, Qing-Long Han, Gang Ma, and Bing Zhu. RaLiBEV: Radar and LiDAR BEV Fusion Learning for Anchor Box Free Object Detection System, Nov. 2022. Issue: arXiv:2211.06108 arXiv:2211.06108 [cs]. 3, 12

[53] Zeyu Yang, Jiaqi Chen, Zhenwei Miao, Wei Li, Xiatian Zhu, and Li Zhang. DeepInteraction: 3D Object Detection via Modality Interaction. *Advances in Neural Information Processing Systems*, 35:1992–2005, Dec. 2022. 2, 3, 6, 8

[54] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. 3DSSD: Point-Based 3D Single Stage Object Detector. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11037–11045, Seattle, WA, USA, June 2020. IEEE. 2

[55] Zhuyu Yao, Jiangbo Ai, Boxun Li, and Chi Zhang. Efficient DETR: Improving End-to-End Object Detector with Dense Prior, Apr. 2021. Issue: arXiv:2104.01318 arXiv:2104.01318 [cs]. 1, 3

[56] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3D Object Detection and Tracking. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11779–11788, Nashville, TN, USA, June 2021. IEEE. 2, 5, 6

[57] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Multimodal Virtual Point 3D Detection. In *Advances in Neural Information Processing Systems*, volume 34, pages 16494–16507. Curran Associates, Inc., 2021. 3

[58] Jin Hyeok Yoo, Yecheol Kim, Jisong Kim, and Jun Won Choi. 3D-CVF: Generating Joint Camera and LiDAR Features Using Cross-view Spatial Feature Fusion for 3D Object Detection. In *Computer Vision – ECCV 2020*, volume 12372, pages 720–736. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science. 3

[59] Yihan Zeng, Da Zhang, Chunwei Wang, Zhenwei Miao, Ting Liu, Xin Zhan, Dayang Hao, and Chao Ma. LIFT: Learning 4D LiDAR Image Fusion Transformer for 3D Object Detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17151–17160, New Orleans, LA, USA, June 2022. IEEE. 3, 12

[60] Yin Zhou and Oncel 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*, pages 4490–4499, 2018. 2, 5, 12

[61] Zixiang Zhou, Xiangchen Zhao, Yu Wang, Panqu Wang, and Hassan Foroosh. CenterFormer: Center-Based Transformer for 3D Object Detection. *Computer Vision – ECCV 2022*, 13698:496–513, 2022. Series Title: Lecture Notes in Computer Science. 2, 3, 12

[62] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced Grouping and Sampling for Point Cloud 3D Object Detection, Aug. 2019. Issue: arXiv:1908.09492 arXiv:1908.09492 [cs]. 13

[63] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable Transformers for End-to-End Object Detection. In *ICLR 2021*, Mar. 2021. arXiv:2010.04159 [cs]. 1, 3, 13

[64] Zhuofan Zong, Guanglu Song, and Yu Liu. DETRs with Collaborative Hybrid Assignments Training, Mar. 2023. Issue: arXiv:2211.12860 arXiv:2211.12860 [cs]. 5

Supplementary Material

The supplementary material is organized as follows:

- Appendix A introduces the concept of transformer models and their application to object detection.
- Appendix B describes the sensor backbones and explains the modality-balanced decoder.
- Appendix C contains details on the training strategy, as well as additional ablation studies. It also includes an experiment on evaluation with grouped object classes to provide insight into how image features improve detection scores compared to the LiDAR-only setting.
- Appendix D finally presents additional results in the form of the distribution for multiple training runs, as well as class-specific detection scores.

A. Background on Transformers

Transformer [44] models consist of an encoder and a decoder, both of which contain *attention* mechanisms. Attention lets the model selectively focus on relevant data. We differentiate between *self-attention* and *cross-attention*. In self-attention, the tokens of an input sequence can interact with each other. We denote an input sequence $\mathbf{x} = \{\mathbf{x}_i\}_{i=1}^{N}$ of N tokens, where each token is a feature vector. In object detection, this input sequence would for example represent the N pixels of an image or the N points of a point cloud. Since each token can interact with any other token, there exists a quadratic memory relation to N. Attention works as follows. From each token $\mathbf{x}_i$, a query $\mathbf{q_i}$, key $\mathbf{k_i}$ and value $\mathbf{v_i}$ vector are computed. To obtain the attention weight between two tokens $\mathbf{x}_a$ and $\mathbf{x}_b$ with $a, b \in \{1, \ldots, N\}$ we take the dot product of their respective query and key vectors: $\mathbf{q}_a \cdot \mathbf{k}_b$. Repeating this for $\mathbf{q}_a$ with all other keys in $\mathbf{k}$ gives us N attention weights from token $\mathbf{x}_a$ to all tokens in $\mathbf{x}$. After normalization and a softmax operation, these weights are used to query information from all value vectors in $\mathbf{v}$ onto $\mathbf{q}_a$, completing the attention operation. This process is repeated for each query $\mathbf{q}_i$. The operation is typically computed efficiently with the query, key, and value vectors stacked in matrix form as defined in 3, where d is the dimensionality of the query, key, and value vectors.

$$\text{Attention}\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}\right) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d}}\right)\mathbf{V} \quad (3)$$

In cross-attention, the tokens of two sequences $\mathbf{x}_{i=1}^{N}$, and $\mathbf{y}_{j=1}^{M}$ can interact with each other, where N and M do not have to be equal. The attention formulation is equal to that of self-attention in 3, but the queries $\mathbf{q}$ are now computed from the second sequence $\mathbf{y}$, whereas the keys $\mathbf{k}$ and values $\mathbf{v}$ are from $\mathbf{x}$. Cross-attention enables the tokens in $\mathbf{y}$ to query information from sequence $\mathbf{x}$, by selectively spreading attention over it.

As for the complete transformer model, self-attention is applied to sequence $\mathbf{x}$ in the encoder. In the decoder, both self- and cross-attention are used: self-attention between the tokens of $\mathbf{y}$, and cross-attention each query $\mathbf{q}_j$ originating from $\mathbf{y}$, and keys $\mathbf{k}$ and values $\mathbf{v}$ from the encoded sequence $\mathbf{x}$. Most implementations stack the encoder and decoder blocks multiple times, where a typical number of layers is $L = 6$. As stated in Sec. 2.1, most models for 3D object detection in automotive scenes omit the encoder due to the size of the self-attention operation, and replace it with a convolutional backbone instead.

What exactly is represented in the decoder input $\mathbf{y}$ depends on the task at hand. In object detection, the decoder is used as either a detection- or fusion mechanism. In the former case, each token in $\mathbf{y}$ represents a candidate object [1, 7, 22, 33, 38, 61], where the number of tokens M is equal to the maximum number of objects that can be detected, typically in the order of hundreds. The tokens in $\mathbf{x}$ are sensor features, e.g. camera or LiDAR features. Each *object query* $\mathbf{q}_j$ interacts with all other queries in the decoder's self-attention, and with the input sequence $\mathbf{x}$ in cross-attention to query information from the sensor features. The object queries are finally passed to a separate regression and classification head to generate a bounding box from each query. In case of sensor feature fusion, both $\mathbf{x}$ and $\mathbf{y}$ are sensor features. Cross-attention is then used as a soft association fusion mechanism to let the tokens in $\mathbf{y}$ query information from $\mathbf{x}$. Examples include fusing LiDAR-camera features [17, 23, 59], LiDAR-RADAR features [52] and multiple LiDAR representations [13, 48].

In this work, we consider transformer implementations that use the decoder as the detection mechanism for 3D object detection in automotive scenes. One benefit of using a transformer decoder for object detection is the ability to do end-to-end training, because NMS post-processing can be left out since this task is carried out in the self-attention operation between object queries in the decoder [3]. Additionally, object queries naturally extend to tracking tasks, because their high-dimensional representations serve as better candidate objects for the following frame than the low-dimensional bounding boxes typically used [38].

B. Model Details

B.1. Backbones

Following FUTR3D [7], the LiDAR backbone is a VoxelNet [50, 60] followed by a feature pyramid network (FPN) [27] to obtain multi-scale BEV feature maps $\mathcal{F}_{\text{lid}}^{j} \in \mathbb{R}^{C \times H_j \times W_j}$, with C the channel dimension, and $H_j \times W_j$ the size of the j-th feature map for all $j \in \{1, 2, \ldots, m\}$ with m the number of levels in the FPN. The camera back-

bone is a VoVNet [21], again followed by an FPN [27]. With N surrounding cameras as input, the backbone outputs multi-scale features for each image, resulting in $\mathcal{F}_{\text{cam}}^{kj} \in \mathbb{R}^{C \times H_j \times W_j}$ for all $k \in \{1, 2, \ldots, N\}$ and for all $j \in \{1, 2, \ldots, m\}$.

B.2. Modality-Balanced Decoder

Following FUTR3D [7] we employ a *modality-balanced* decoder. In contrast to TransFusion [1], the object queries can access features from all sensor modalities in every decoder layer. The conceptual difference between these approaches can be seen in Fig. 2. In the decoder, the query's 3D location is projected onto the corresponding camera image(s) and the BEV LiDAR features to allow querying information from both modalities, after which the selected features are fused and used to update the query.

We now briefly explain the essence of this decoder. Our implementation follows that of FUTR3D [7] unless stated otherwise, and we refer the reader to their work for more details.

Decoder Overview The decoder follows a typical implementation of transformers for object detection. Object queries interact with each other in multi-head self-attention, and with sensor features in multi-head cross-attention. Multiple of these decoder layers are stacked, where the current layer is $l \in \{1, 2, \ldots, L\}$. Each decoder layer is accompanied by a regression- and classification head Φ_{reg}^{l} and Φ_{cls}^{l}, which are shared among the object queries. In the l-th layer, the object queries are defined as $\{\mathbf{q}_i^l\}_{i=1}^{M} \in \mathbb{R}^C$, and their locations as $\{\mathbf{c}_i^l\}_{i=1}^{M} \in \mathbb{R}^3$. For each object query $\mathbf{q}_i^l$, the regression head Φ_{reg}^{l} outputs the offset $\Delta\mathbf{x}_i^l \in \mathbb{R}^3$ from the query's location to the predicted object's center coordinate, together with other bounding box regression targets. The classification head Φ_{cls}^{l} predicts a categorical label $\hat{y}_i^l$.

The predicted offset $\Delta\mathbf{x}_i^l$ is used to iteratively update the query's location before the next decoder layer as shown in 4, following other works [39, 47, 63].

$$\mathbf{c}_i^{l+1} = \mathbf{c}_i^l + \Delta\mathbf{x}_i^l \tag{4}$$

Cross-Attention The LiDAR-camera fusion is done at instance-level for each object query individually in the cross-attention of the decoder. LiDAR features are sampled using deformable attention [63]. From query $\mathbf{q}_i$, we predict $V \times m$ sampling offsets and corresponding attention weights to sample V LiDAR features around the query's BEV location $\mathcal{P}(\mathbf{c}_i)$, for all m multi-scale BEV feature maps. Here, $\mathcal{P}(\mathbf{c}_i)$ is the projection of the 3D query location $\mathbf{c}_i$ onto the 2D BEV grid. The result is a LiDAR feature vector $\mathcal{SF}_{lid}^{i} \in \mathbb{R}^C$ for query $\mathbf{q}_i$.

Camera features are sampled around the query's projected image location after computing a projection $\mathcal{T}_k(\mathbf{c}_i)$ for all $k \in \{1, 2, \ldots, N\}$, with N the number of cameras. Features are sampled using bilinear sampling between the features in $\mathcal{F}_{\text{cam}}^{kj}$ corresponding to $\mathcal{T}_k(\mathbf{c}_i)$, for the j-th feature map of the k-th image. Attention weights are predicted from the query once again to perform a weighted sum over all sampled features, resulting in a single camera feature vector $\mathcal{SF}_{\text{cam}}^{i} \in \mathbb{R}^C$ for query $\mathbf{q}_i$.

The sampled LiDAR and camera features for each query are then concatenated and fused with an MLP Φ_{fus} as in 5, with $\mathcal{SF}_{\text{fus}}^{i} \in \mathbb{R}^C$, the same channel dimension as query q_i.

$$\mathcal{SF}_{\text{fus}}^{i} = \Phi_{\text{fus}}\left(\mathcal{SF}_{\text{lid}}^{i} \oplus \mathcal{SF}_{\text{cam}}^{i}\right) \tag{5}$$

Before updating the query, the fused features are made location aware by adding a positional embedding $\text{PE}(\mathbf{c}_i)$ based on the location of the query. At last, the query is updated as $\mathbf{q}_i' = \mathbf{q}_i + \Delta\mathbf{q}_i$, where $\Delta\mathbf{q}_i = \mathcal{SF}_{\text{fus}}^{i} + \text{PE}(\mathbf{c}_i)$.

C. Experiments

C.1. Implementation Details

Data Augmentation nuScenes is annotated at 2 Hz but LiDAR data is captured at 20 Hz, so we follow the common practise of combining the annotated sample with the previous 9 sweeps to get a denser point cloud. We additionally adopt a common augmentation pipeline for the LiDAR data, where we use random rotation with $r \in [\pi/4, \pi/4]$, random scaling with $s \in [0.9, 1.1]$, random xyz translation with a standard deviation 0.5, and random horizontal and vertical flipping. We use CBGS [62] class-balanced sampling, which repeats samples that contain uncommon classes to improve the class balance in nuScenes. Finally, we use ground truth copy-paste augmentation [50], and disable it for the final epochs to match the real data distribution again [46]. We do not adopt any augmentation at test time.

Training Schedule We first train the LiDAR branch of our model, using a pretrained LiDAR backbone. The schedule is set to 6 epochs, with GT copy-paste augmentation disabled in the final 3 epochs. From there, the pre-trained image backbone is added, and the LiDAR-camera model is trained for another 4 epochs with both backbones frozen. Such a sequential approach has shown to yield better performance than joint training from the start, because it allows for better augmentation in the LiDAR-only stage of training [1, 7]. We use an initial learning rate of 1.0×10^{-4} for both LiDAR-only and LiDAR-camera training, with a cyclic learning rate policy [41]. We train the proposed model multiple times and report the mean and standard deviation of the resulting scores in Appendix D.2.

C.2. Ablation

LiDAR-Camera Fusion We ablate on the LiDAR-camera fusion components, specifically on the locations in

the model where we include image features. We can use image features in the proposed initialization to find better query locations, and in the decoder to let the queries sample image features based on their location. The results are shown in Tab. 5, where use the LiDAR-only version of the proposed method as the baseline.

	mAP ↑	NDS ↑
EfficientQ3M-L	64.3 ± 0.1	69.0 ± 0.0
w/ cam init	64.4 ± 0.1	69.1 ± 0.1
w/ cam decoder	69.6 ± 0.0	72.4 ± 0.1
EfficientQ3M	**70.1** ± 0.1	**72.6** ± 0.1

Table 5. Ablation on the fusion components on the nuScenes *val* set. **w/ cam init** is whether we use image features to predict initial query locations. **/w cam decoder** denotes if image features are available to the queries in the decoder's cross-attention. The proposed model **EfficientQ3M** has both.

Using both LiDAR and camera features to find initial query locations (**w/ cam init**) on an otherwise LiDAR-only model does not bring any significant gain. Even if the initialization produces better queries, the LiDAR-only decoder is not able to turn these into better predictions. Allowing the queries to access camera features in the decoder's cross-attention (denoted as **w/ cam decoder**) brings the majority of the performance improvement on the LiDAR-only model. Using camera features in *both* the initialization of query locations and the decoder's cross-attention results in the best performance.

	mAP ↑		
	[0m,15m]	[15m,30m]	[30m, +inf]
EfficientQ3M-L	78.8	67.6	45.1
EfficientQ3M	82.0 (+3.2)	74.1 (+6.5)	55.5 (+10.4)

Table 6. Performance breakdown for different object distances to the ego vehicle on the nuScenes *val* set.

Next, to show where the camera modality adds detection performance, we compare the LiDAR-only version of the proposed method to the LiDAR-camera version by dividing detections into different groups based on distance thresholds to the ego vehicle. In Tab. 6, we find that image features improve detections the most on far-away objects when compared to the LiDAR-only model. This can be explained by the fact that such objects are typically more difficult to detect from LiDAR data alone, due to the limited density of point clouds at longer distances.

Ground Truth Query Initialization To gain insight into the upper performance limits of input-dependent object query initialization, we initialize object queries based on the locations of ground truth objects. The object query locations are taken as the top-M peaks in ground truth heatmap $\hat{\mathcal{S}} \in \mathbb{R}^{X \times Y \times K}$, introduced in Sec. 3.2. The corresponding object query feature vectors are then initialized with features at these locations. Next to our proposed method, we also conduct the experiment for TransFusion [1] using the public source code and model weights. Neither model is trained specifically to receive these "perfect" object queries.

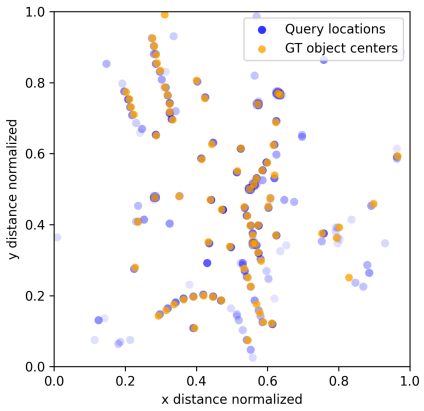

Figure 6. Example of predicted query locations with the proposed method, seen from the bird's eye view. The opacity of the blue query locations indicates the confidence of the respective prediction. Ground truth object locations are overlayed in orange.

	Mod.	mAP ↑	GT mAP ↑
TransFusion	L	65.1	69.2 (+4.1)
TransFusion	L+C	67.3	69.9 (+2.6)
EfficientQ3M	L	64.7	68.7 (+4.0)
EfficientQ3M	L+C	70.8	70.5 (-0.3)

Table 7. Detection performance on the nuScenes *val* set for various models with input-dependent object query initialization. We also initialize the object queries using ground truth annotations, and denote the resulting scores as **GT mAP**. We use the number of queries $M = 200$ for all models.

As seen in Tab. 7, we find that, in general, the ground truth (GT) initialization boosts performance significantly. Only the LiDAR-camera version of the proposed method does not improve with this initialization. It may be that at some point, the decoder cannot produce better detections from GT object queries because the sensor features are not sufficient to do so. As an example of the query distribution generated by the proposed method, we see in Fig. 6 that the predicted locations already cover the objects well.

We also find that the performance gap between LiDAR-only models and LiDAR-camera models shrinks when applying ground truth initialization. One explanation is that since the LiDAR-camera mAP scores are already higher to begin with, it is more difficult to improve upon those.

Method	LiDAR-only			LiDAR-camera			Extra # TP w/ Camera	
	# TP	# TP_G	# Miscl.	# TP	# TP_G	# Miscl.	Normal	Grouped
TransFusion [1]	114799	115847	1048	115132	116134	1002	333	287
EfficientQ3M (ours)	116819	117456	637	118103	118601	498	1284	1145

Table 8. Grouped evaluation experiment on the nuScenes *val* set. # TP is the total number of true positives over all classes, # TP_G is the total number of true positives when evaluating with grouped classes, and # Miscl. is the difference between the two, representing the number of misclassified objects in normal evaluation. When comparing the number of true positives between the LiDAR-only setting and LiDAR-camera setting, we additionally find the number of newly-found objects from having the camera modality available, both for normal evaluation and evaluation with grouped classes. The contributing components of the latter metric are highlighted for each model.

C.3. Grouped Evaluation for Sensor Fusion

To provide context to our hypothesis that a sequential fusion approach like in TransFusion [1] does not make optimal use of image features, we design an experiment where we group certain object classes during evaluation. The philosophy behind the experiment is as follows. There are three ways in which adding the camera modality can improve detection scores compared to a LiDAR-only model:

1. Increase the confidence scores of true positives to move them 'higher' in the PR-curve relative to false positives, thus increasing the gap between them.
2. Correctly classify a previously misclassified prediction to turn a false positive into a true positive.
3. Create new true positives by finding new objects.

Our expectation is that TransFusion [1] mainly improves from the first and second point, because it only samples image features that fall within the projected bounding box of existing LiDAR-only predictions. The recall of the full model is thus limited to that of the LiDAR-only predictions. We expect that the modality-balanced decoder with the proposed initialization method should be able to exploit camera features better, and would see improvements from all three points above because of it.

In this experiment, we measure how many new objects (new true positives) are found for a given model using LiDAR-camera fusion, relative to the LiDAR-only version of the same model. To remove the effect of better classification (point 2), we group object classes to reduce the number of classes in evaluation from 10 to 5. We group the `[car, truck, construction_vehicle, bus, trailer]` classes into a single `vehicle` class, and similarly group the `[bicycle, motorcycle]` classes into a single `bike` class. We leave the `pedestrian`, `traffic cone` and `barrier` classes as is, because we assume negligible misclassification to occur between those. The model still assigns each prediction with one out of $K = 10$ class labels, but we no longer count a false positive if the model would for example confuse a bicycle and a motorcycle.

We conduct this experiment for TransFusion [1] and the proposed method, and present the results in Tab. 8. We evaluate both models twice on the nuScenes validation set, once with normal evaluation and once with the grouped classes. For the latter, we assume to have mostly removed any effects of misclassification. When we compare the total number of true positives between a LiDAR-only and LiDAR-camera model using the grouped evaluation, we should measure only the effect of camera features helping to find new objects (right-most column in Tab. 8).

Since these metrics are measured as integers, we do not base them on the average of multiple distance matching thresholds like nuScenes does in the mAP metric, but instead only consider a single threshold of 2.0 m.

Grouped Evaluation Results We find that the proposed method is able to make better use of the camera modality to improve the model's recall (i.e. finding more true positives) compared to TransFusion. We see that the recall of TransFusion is indeed somewhat limited by LiDAR-only stage of the model, since the LiDAR-camera model does not find many new objects (287 objects, compared to 1145 in the proposed method).

We note that the relative increase in true positives is small in either case (0.2% for TransFusion and 1.0% for the proposed method). Considering that we find a performance improvement of +6.1 mAP with the LiDAR-camera version of the proposed model compared to the LiDAR-only version, it appears that most of the benefit of adding the camera modality comes not from finding new objects, but rather from assigning better confidence scores to existing predictions (i.e. point 1, defined above).

Another observation from this experiment is that the number of misclassifications does not drop significantly after adding the camera modality to the LiDAR-only version of each model. We find a reduction in misclassified objects of only 46 objects with TransFusion and 139 with the proposed method. It should be noted that these results are also affected by a general increase in detected objects with LiDAR-camera fusion, which may cause an increase in misclassified objects. The exact reduction of misclassified ob-

jects may therefore be higher than reported here.

C.4. Memory Requirements

One advantage of reducing the number of object queries is that less GPU memory may be required to run the model. This enables us to increase the batch size in training or use more accessible GPU cards with less memory. The self-attention operation has a quadratic memory requirement $\mathcal{O}(M^2)$ to the number of queries M. With our proposed model we can reduce M from 900 to 200, which would reduce the memory requirement for this operation by a factor of ≈ 20. We measure the memory usage for various values of M and report the results in Tab. 9.

# queries	200	900	1800	3600
GPU memory	775	801	1071	2335

Table 9. GPU memory usage in MegaByte as a function of the number of object queries for the LiDAR-only version of the proposed model. Memory is measured at test time using `torch.cuda.max_memory_allocated()`, which returns the maximum allocated memory over the entire run time.

We find that the influence of M on the maximum memory usage is not significant for the order of magnitude that we are considering. For reference, we also report results for 1800 and 3600 object queries. We do start see the quadratic relation then, but only for M much larger than required for our use case. We conclude that we need not to limit our number of object queries for the purpose of managing GPU memory usage. On top of our findings, the latest versions of PyTorch [36] offer memory-efficient implementations of attention like FlashAttention [8], which reduces the memory requirement to $\mathcal{O}(M)$. Although the latency is now quadratically dependent on M, the memory usage is no issue anymore.

Init. Method	Mod.	Train Mem. ↓	Test Mem. ↓
Learned distr.	L	11032	1426
w/ TransFusion	L	11424	1473
w/ ours	L	10636	1427
Learned distr.	L+C	23682	23282
w/ TransFusion	L+C	51006	23306
w/ ours	L+C	23698	23286

Table 10. GPU memory usage for a batch size of 4, with the number of queries $M = 200$. Memory usage is measured in MegaByte using `torch.cuda.max_memory_allocated()`, which returns the maximum allocated memory over the entire run time.

Tab. 10 additionally shows the GPU memory usage for different query initialization strategies for both LiDAR-only and LiDAR-camera settings. We find that the proposed method does not result in increased maximum memory usage compared to the learned queries used in FUTR3D [7]. Surprisingly, we find that the transformer network used to fuse LiDAR and camera features in TransFusion's [1] initialization method consumes more than double the memory of the FUTR3D baseline during training. This may restrict training because it only allows the use of half the batch size of the baseline for a given GPU.

D. Additional nuScenes Results

D.1. Qualitative Analysis

Below, we provide additional qualitative analysis of the predictions made by the proposed model. Fig. 7 shows a succesful sample from a dense scene at a busy intersection. Looking at the LiDAR BEV window, we do see a number of missed objects but these however entail either heavily occluded objects or objects outside of the detection range.

Fig. 8 shows two samples from an interesting traffic scene, with the top sample containing false positives. The bottom sample occurs 2 seconds later than the top sample. We highlight the same region of interest in each sample, both in the relevant camera image and the LiDAR BEV. We first consider the top sample. We see one duplicate detection resulting in a false positive in the front-left camera view (1.A). Additionally, there are three cars detected in the parking lot (1.B, 1.C and 1.D, in the LiDAR BEV) that do not have a matching ground truth annotation. When looking at the bottom sample however, we see that these objects actually do exist and are now also annotated (2.B, 2.C and 2.D). The top sample thus includes three detections that are labeled a false positive during evaluation, even though they should be true positives.

D.2. Distribution of Detection Scores

We run the training of our main LiDAR-only and LiDAR-camera models three times each with different random seeds, and report the resulting mAP and NDS scores in Tab. 11 together with the mean and standard deviation of the scores. We find that the deviation between runs is small, and that any run outperforms the FUTR3D [7] baseline.

D.3. Class-Specific Results

Since FUTR3D [7] does not report class-specific AP scores on the nuScenes *test* set, we compare our results on the nuScenes *val* set in Tab. 12. TransFusion [1] is also included for reference. We see improvements on the baseline for all classes except the `Bus` class. Notable results for the LiDAR-only model are large improvements for the `Truck` (+3.3 mAP), `Barrier` (+4.7 mAP) and `Bike` class (+2.5 mAP). For the LiDAR-camera model, we find strong improvements for the `C.V.` (+2.1 mAP) and `Trailer` class (+3.7 mAP).

Figure 7. Example of a densely populated scene with correct detections. Best viewed zoomed in.

Figure 8. Two consecutive samples from the same scene where the top sample occurs 2 seconds before the bottom sample. We highlight the same region of interest in bright green in both the camera view and the LiDAR BEV for both samples. Best viewed zoomed in.

Method	Mod.	mAP ↑				NDS ↑			
		Run 1	Run 2	Run 3	Mean + std	Run 1	Run 2	Run 3	Mean + std
EfficientQ3M	L	65.08	64.93	**65.34**	65.11 ± 0.17	69.45	69.38	**69.60**	69.48 ± 0.09
EfficientQ3M	L+C	71.10	**71.23**	71.13	71.15 ± 0.05	73.47	**73.52**	73.43	73.47 ± 0.04

Table 11. Detection scores on the nuScenes *val* set for multiple training runs of both the LiDAR-only version and the LiDAR-camera version of the proposed model with $M = 900$ queries. The high scores for each modality are reported as the main results in Tab. 1.

Method	Mod.	mAP ↑	NDS ↑	Car	Truck	C.V.	Bus	Trailer	Barrier	Motor.	Bike	Ped.	T.C.
FUTR3D	L	63.7	69.0	85.9	55.7	27.1	**75.7**	44.6	64.2	72.7	54.9	84.3	71.5
TransFusion	L	65.1	**69.9**	**87.0**	**61.8**	27.9	72.9	43.1	69.6	70.8	56.3	**87.0**	**74.0**
EfficientQ3M	L	**65.3**	69.6	86.9	59.0	**28.6**	74.1	**45.0**	**69.9**	**73.2**	**57.4**	85.7	73.5
FUTR3D	L+C	70.3	73.1	88.4	67.1	33.7	**78.5**	46.7	71.4	79.6	**72.3**	86.7	78.5
TransFusion	L+C	67.3	70.9	87.9	64.0	29.8	74.1	43.5	70.1	74.3	63.5	**88.3**	77.1
EfficientQ3M	L+C	**71.1**	**73.5**	**89.1**	**67.2**	**35.8**	77.8	**50.4**	**72.6**	**80.5**	72.1	87.9	**78.9**

Table 12. Class-specific mAP scores on the nuScenes *validation* set. The best results for each modality are highlighted in bold text. **C.V.** is construction vehicle, **Motor.** is motorcycle, **Ped.** is pedestrian and **T.C.** is traffic cone.