MObI: Multimodal Object Inpainting Using Diffusion Models

Supplementary Material

A. Extended Related Work

Multimodal data is crucial for ensuring safety in autonomous driving, and most state-of-the-art perception systems employ a sensor fusion approach, particularly for tasks like 3D object detection [15, 28, 33]. However, testing and developing such safety-critical systems requires vast amounts of data, which is costly and time-consuming to obtain in the real world. Consequently, there is a growing need for simulated data, enabling models to be tested efficiently without requiring on-road vehicle testing.

Copy-and-paste Early efforts in synthetic data generation relied on copy-and-paste methods. For example, [12] used depth maps for accurate scaling and positioning when inserting objects, while later approaches like [8] focused on achieving patch-level realism through blending, improving 2D object detection. A more straightforward approach, presented by [13], naively pastes objects into images without blending and demonstrates its efficacy in improving image segmentation. In autonomous driving, PointAugmenting [51] extends this copy-and-paste approach to both camera and lidar data to enhance 3D object detection. Building on the lidar GT-Paste method [63], it incorporates ideas from CutMix augmentation [69] while ensuring multimodal consistency. This method addresses scale mismatches and occlusions by utilising the lidar point cloud for guidance during the insertion process. Similarly, MoCa [72] employs a segmentation network to extract source objects before insertion, instead of directly pasting entire patches. Geometric consistency in monocular 3D object detection has also been explored in [27]. While these methods improve object detection and mitigate class imbalance, their compositing strategy leads to unrealistic blending, especially in image space. Furthermore, they lack controllability, such as the ability to adjust the position and orientation of inserted objects, limiting their utility for testing.

Image compositing In this work, we aim to improve upon these approaches by drawing inspiration from recent advancements in image inpainting. Early efforts like ST-GAN [30] tackled the challenge of unrealistic foreground blending by using GANs [14] with spatial transformer networks to recursively predict and apply corrections, achieving natural blending via warp composition. ObjectStitch [48] leverages diffusion within an edit mask for smooth patch-level blending. Methods like Paint-by-Example [64] and AnyDoor [5] extend this capability by generating entire images conditioned on scene context and an edit mask, achieving greater semantic coherence. AnyDoor achieves fine-grained object inpainting by using SAM [22] for reference segmentation and more advanced feature extraction techniques. Other notable works include Magic Insert [45], which enables drag-and-drop object insertion between images with differing styles, and [24], which adjusts object pose to respect scene affordances. ObjectDrop [57] trains on counterfactual examples to enhance object insertion. Although these methods improve seamless and context-aware image compositing, they do not control the 3D position and orientation of objects in the real world, a critical requirement for training and testing, nor do they consider multimodal extensions.

Full scene generation Recent advancements in conditional full-scene generation have yielded impressive results. BEVControl [65] uses a two-stage method (controller and coordinator) to generate scenes conditioned on sketches, ensuring accurate foreground and background content. Text2Street [49] combines bounding box encoding with text conditions, employing a ControlNet-like [70] architecture for guidance. DrivingDiffusion [26] represents bounding boxes as layout images passed as an extra channel in the U-Net [43]. MagicDrive [10] incorporates bounding boxes and camera parameters alongside text conditions for full-scene generation, with a cross-view attention module leveraging BEV layouts. SubjectDrive [20] generates camera videos conditioned on the appearance of foreground objects. LiDM [41] focuses on lidar scene generation conditioned on semantic maps, text, and bounding boxes. DriveScape [58] introduces a method to generate multi-view camera videos conditioned on 3D bounding boxes and maps using a bi-directional modulated transformer for spatial and temporal consistency.

Synthetic lidar data generation has also advanced significantly. LidarGen [74] and LiDM [41] employ diffusion for lidar generation, with the latter also incorporating semantic maps, bounding boxes, and text. UltraLidar [62] densifies sparse lidar point clouds, while RangeLDM [19] accelerates lidar data generation by converting point clouds into range images using Hough sampling and enhancing reconstruction through a range-guided discriminator. DynamicCity [2] generates lidar sequences conditioned on dynamic scene layouts, and [60] generates object-level lidar data, demonstrating its benefits for object detection. However, these works do not jointly generate camera and lidar data, and full-scene generation can result in a large domain gap, particularly for downstream tasks like object detection, making it challenging to create realistic counterfactuals.

Multimodal object inpainting GenMM [46] represents a new direction in multimodal object inpainting using a multi-stage pipeline that ensures temporal consistency. However, it remains limited in controllability, requiring the reference to closely align with the insertion angle. Furthermore, it does not generate lidar and camera modalities jointly, instead focusing on geometric alignment while excluding lidar intensity values. We take a similar approach, but propose an end-to-end method that jointly generates camera and lidar data for reference-guided multimodal object inpainting. Our method achieves realistic and consistent multimodal outputs across diverse object angles.

B. Method Details

B.1. Details on image processing

Bounding Box Projection: The bounding boxes from the source and destination scenes, $\text{box}_s, \text{box}_d \in \mathbb{R}^{8\times 3}$, are projected onto the image space using the respective camera transformations:

$$\text{box}_s^{(\text{C})} = \mathbf{T}_s^{(\text{C})} \cdot \text{box}_s \in \mathbb{R}^{8\times 2}, \quad \text{box}_d^{(\text{C})} = \mathbf{T}_d^{(\text{C})} \cdot \text{box}_d \in \mathbb{R}^{8\times 2}.$$

We randomly crop the source image around the corresponding bounding box in such a way that the projected bounding box covers at least 20% of the area. We apply the corresponding viewport transformation to box_d.

Edit Mask: The edit region is defined by a binary mask $\mathbf{m}^{(\text{C})} \in \{0,1\}^{D\times D}$, created by inpainting $\text{box}_d^{(\text{C})}$ onto an initially all-zero matrix, where the inpainted region is assigned values of 1. The complement of this mask is defined as:

$$\bar{\mathbf{m}}^{(\text{C})} = \mathbf{J} - \mathbf{m}^{(\text{C})}, \quad \mathbf{J} \in \{1\}^{D\times D}.$$

B.2. Details on lidar processing and encoding

We consider the lidar point cloud of the destination scene, $P_d \in \mathbb{R}^{N\times 4}$, where N represents the number of points and the four channels correspond to the x, y, z coordinates and intensity values. The lidar points are projected onto a range view $R_d \in \mathbb{R}^{32\times 1096\times 2}$ using the transformation described below. This transformation is loss-less, except for points near the end of the lidar sweep that overlap with the beginning due to motion compensation.

Point cloud to range view transformation We consider the point cloud for a single sweep of the destination scene, $P_d \in \mathbb{R}^{N\times 4}$, where N represents the number of points, and the four channels correspond to the x, y, z coordinates and intensity values. The lidar points are projected onto a range view $R_d \in \mathbb{R}^{32\times 1096\times 2}$ using the transformation described below.

For each point in P_d, the depth (Euclidean distance from the sensor) is calculated as:

$$d_i = \sqrt{x_i^2 + y_i^2 + z_i^2}.$$

Points with depths outside the predefined range $[1.4, 54]$ are filtered out. The yaw and pitch angles are then computed as:

$$\text{yaw}_i = -\arctan 2(y_i, x_i), \quad \text{pitch}_i = \arcsin\left(\frac{z_i}{d_i}\right).$$

The beam pitch angles $\{\theta_k\}_{k=1}^{H}$ are chosen as $\theta_k = 0.0232 \cdot x_k$, where $x_k \in \{-23, -22, \ldots, 8\}$, to best match the binning of the nuScenes [3] lidar sensor's vertical beams and its field of view. Each point is assigned to the closest vertical beam based on its pitch angle, determining its y_i vertical coordinate, an integer in the range $[0, 31]$.

The yaw angle is mapped to the horizontal coordinate x of the range view grid as:

$$x_i = \left\lfloor \frac{\text{yaw}_i}{\pi} \cdot \frac{W}{2} + \frac{W}{2} \right\rfloor,$$

The final range view representation R_d of the destination scene encodes depth and intensity for each point projected onto the $H \times W$ grid, where $H = 32$ denotes the number of vertical beams, and $W = 1096$ represents the horizontal resolution. Unassigned pixels in the range view are set to a default value. Each point is mapped to a specific pixel coordinate in the range view.

Note that the transformation is not injective, as some points overlap at the start and end of the lidar sweep due to motion compensation; however, this overlap has minimal impact. We additionally store the original pitch and yaw values for each point assigned to a range view pixel in matrices $R_d^{\text{yaw}} \in \mathbb{R}^{H\times W}$ and $R_d^{\text{pitch}} \in \mathbb{R}^{H\times W}$, respectively. These matrices enhance the inverse transformation from range view to point cloud by preserving the unrasterized angular information.

Range view to point cloud Transformation To reconstruct the point cloud from the range view, we leverage the stored unrasterized pitch and yaw matrices, $R_d^{\text{pitch}} \in \mathbb{R}^{H\times W}$ and $R_d^{\text{yaw}} \in \mathbb{R}^{H\times W}$, which preserve the original angular information for each pixel.

The depth values $R_d^{\text{depth}} \in \mathbb{R}^{H\times W}$ are flattened to the vector $\mathbf{d} \in \mathbb{R}^N$, where $N = H \times W$. Similarly, the pitch and yaw matrices are flattened to the vectors $\boldsymbol{\theta} \in \mathbb{R}^N$ and $\boldsymbol{\phi} \in \mathbb{R}^N$, representing the pitch and yaw angles for each pixel in the range view. Using these angular and depth values, the point cloud $P_d \in \mathbb{R}^{N\times 3}$ is reconstructed as:

$$\begin{aligned}
\mathbf{p}_x &= \mathbf{d} \cdot \cos(\boldsymbol{\phi}) \cdot \cos(\boldsymbol{\theta}) \\
\mathbf{p}_y &= -\mathbf{d} \cdot \sin(\boldsymbol{\phi}) \cdot \cos(\boldsymbol{\theta}) \\
\mathbf{p}_z &= \mathbf{d} \cdot \sin(\boldsymbol{\theta}),
\end{aligned}$$

where $\mathbf{p}_x, \mathbf{p}_y, \mathbf{p}_z \in \mathbb{R}^N$ are the vectors of reconstructed x, y, and z coordinates, respectively. The reconstructed point cloud P_d is then given by stacking these coordinate vectors as $P_d = [\mathbf{p}_x, \mathbf{p}_y, \mathbf{p}_z]$.

By leveraging the stored pitch and yaw matrices, the process accurately restores the point cloud while avoiding misalignments introduced by motion compensation. This ensures that the reconstructed point cloud aligns perfectly with the original input, except for the overlapping points we previously mentioned, which do not get regenerated.

Range view to range image processing We project the bounding box box_d onto R_d using the coordinate-to-range transformation, resulting in $\text{box}_d^{(\text{R})} \in \mathbb{R}^{8\times 3}$, while preserving the depth of each bounding box point. To enhance the region of interest, we employ a zoom-in strategy analogous to that used in the image processing, by cropping the range view width-wise around $\text{box}_d^{(\text{R})}$, resulting in a $32 \times W^{(\text{R})} \times 2$ object-centric range view, and resizing it to obtain the range image $\mathbf{x}^{(\text{R})} \in \mathbb{R}^{D\times D\times 2}$. We apply the same viewport transformation to the bounding box $\text{box}_d^{(\text{R})}$. The edit region is defined by a mask $\mathbf{m}^{(\text{R})} \in \{0,1\}^{D\times D}$, which is created by inpainting the bounding box box_d onto an initially all-zero matrix, where the inpainted region has values of 1. The complement of this mask is $\bar{\mathbf{m}}^{(\text{R})} = \left(\mathbf{J} - \mathbf{m}^{(\text{R})}\right)$.

Range image reconstruction metrics An important step towards achieving realistic lidar inpainting is ensuring the autoencoder can reconstruct the input point cloud with high fidelity. Since the point cloud to range view transformation is loss-less, we can focus our attention on evaluating the quality of reconstructed range views. We restrict our evaluation to the region within the edit mask $\mathbf{m}^{(\text{R})}$ and the object points from the target range view, selected using the 3D bounding box, see Fig. S8 for examples. For each input range view $\mathbf{X}^{(\text{R})}$ and its reconstruction, $\mathcal{D}^{(\text{R})}(\mathcal{E}^{(\text{R})}(\mathbf{X}^{(\text{R})}))$, we compute the median depth error and the mean squared error (MSE) of the intensity values, restricted on the object points and the edit mask.

Range image encoding We adapt the pre-trained image VAE [21] of StableDiffusion [42] to the lidar modality through a series of training-free adaptations and a fine-tuning step, ablated in Tab. 1.

As a naive solution to encode the lidar modality, we take the preprocessed range view $\mathbf{x}^{(\text{R})} \in \mathbb{R}^{D\times D\times 2}$, duplicate the depth channel, and pass the resulting 3-channel representation through the image VAE [21]. After discarding one depth channel and resizing back to $32 \times W^{(\text{R})} \times 2$ using nearest neighbour interpolation, we compute reconstruction errors using the metrics described in Sec. B.2. This naive approach results in unsatisfactory reconstruction errors.

To address this, we propose three cumulative adaptations that improve depth and intensity reconstruction for object points and the extended edit mask. First, we leverage the higher resolution of $\mathbf{x}^{(\text{R})}$ by applying average pooling when downsizing, which serves as an error correction mechanism.

Next, we observe that the reconstruction error of range pixel values is proportional to the interval size of their distribution. Since intensity values follow an exponential distribution, we normalize intensity $i \in [0, 255]$ using the cumulative distribution function (CDF) of the exponential distribution, choosing $\lambda = 4$ experimentally:

$$i' = 2e^{-\lambda \frac{i}{255}} - 1 \in [-1, 1]$$

To enhance object-level depth reconstruction, we apply depth normalization based on the minimum and maximum depth of $\text{box}_d^{(\text{R})}$, scaling the bounding box by 0.1, which extends the interval the object depth values are distributed on and, in turn, improves object reconstruction error:

$$d' = \begin{cases} -\alpha + 2\alpha \cdot \frac{d-\min_d}{\max_d - \min_d} & \text{if } \min_d \le d \le \max_d \\ -1 + (-(\alpha-1)) \cdot \frac{d+1}{\min_d + 1} & \text{if } -1 \le d < \min_d \\ \alpha + (1-\alpha) \cdot \frac{d-\max_d}{1-\max_d} & \text{if } \max_d < d \le 1 \end{cases}$$

where d is the depth value, α controls range scaling, and $\min_d$, $\max_d$ define normalization boundaries within $[-1, 1]$. Depth values are originally between $[1.4, 54]$, but are linearly normalized to $[-1, 1]$.

Thirdly, we replace the input and output convolution of the pre-trained image encoder and decoder, with two residual blocks, respectively. We now have 2 input and output channels. We fine-tune the VAE [21] with an additional discriminant [9]. The same normalization and resizing strategies are applied, yielding the best reconstruction metrics for $\tilde{\mathbf{x}}^{(R)} = \text{resize}(\mathcal{D}^{(\text{R})}(\mathcal{E}^{(\text{R})}(\text{norm}(\mathbf{x}^{(\text{R})}))))$.

Finally we encode the range image $\mathbf{x}^{(\text{R})}$ to obtain a latent representation $\mathbf{z}_0^{(\text{R})} = \mathcal{E}^{(\text{R})}(\text{norm}(\mathbf{x}^{(\text{R})}))$. Similarly, we encode the lidar environment context $\mathbf{x}^{(\text{R})} \odot \bar{\mathbf{m}}^{(\text{R})}$ to obtain a latent conditioning representation $\mathbf{c}_{\text{env}}^{(\text{R})} = \mathcal{E}^{(\text{R})}(\text{norm}(\mathbf{x}^{(\text{R})} \odot \bar{\mathbf{m}}^{(\text{R})}))$.

B.3. Additional training details

We start by training the newly added input and output adapters of the range autoencoder while keeping the rest of the image VAE [21] from Stable Diffusion [42] frozen. This training phase spans 8 epochs (15k steps) with a learning rate of 4.5×10^{-5}, selecting the checkpoint with the lowest reconstruction loss.

During fine-tuning of the diffusion model, the autoencoders and all layers from the PbE [64] framework remain frozen. Only the bounding box encoder, bounding box adaptation layer, and cross-modal attention layers are trained over 30 epochs (approximately 90k steps) with a

	Reinsertion						Replacement					
	same ref			tracked ref			in-domain ref			cross-domain ref		
Method	FID↓	LPIPS↓	CLIP-I↑	FID↓	LPIPS↓	CLIP-I↑	FID↓	LPIPS↓	CLIP-I↑	FID↓	LPIPS↓	CLIP-I↑
copy&paste	n/a						13.50	0.196	n/a	17.08	0.213	n/a
PbE [64]	7.34	0.131	84.50	7.58	0.135	83.31	9.62	0.148	77.44	10.54	0.150	**77.06**
MObI (ours)	**6.50**	**0.114**	**84.94**	**6.70**	**0.115**	**83.50**	**8.95**	**0.127**	**77.50**	**9.05**	**0.130**	76.00

Table S1. Comparison with image inpainting methods at $D = 512$ resolution in terms of camera realism.

constant learning rate of 8×10^{-5} and a batch size of 2 multimodal samples.

Training takes approximately 20 hours on 8x 24GB NVIDIA A10G or 2x 80GB NVIDIA A100 GPUs. Inference throughput is about 8 camera+lidar samples per minute on a single A100.

Sampling Empty Boxes for Augmentation To enhance augmentation, we sample empty bounding boxes to train the model to reconstruct missing details. A dedicated database of 10,000 such boxes is created. For a given scene, an object from a different scene is selected, ensuring that teleporting the bounding box into the current scene does not result in 3D overlap or a total 2D IoU overlap exceeding 50% with other objects. During training, 30% of the samples are drawn from this database. Black images and boxes with zero coordinates are used for these samples, enabling the model to learn how to fill in background details, as shown in Fig. S7.

Tracked reference sampling Rather than reinserting objects into the scene using the same reference, we utilize the temporal structure of the nuScenes dataset [3]. References for the current object are sampled from a different timestamp following the distribution shown in Fig. S6.

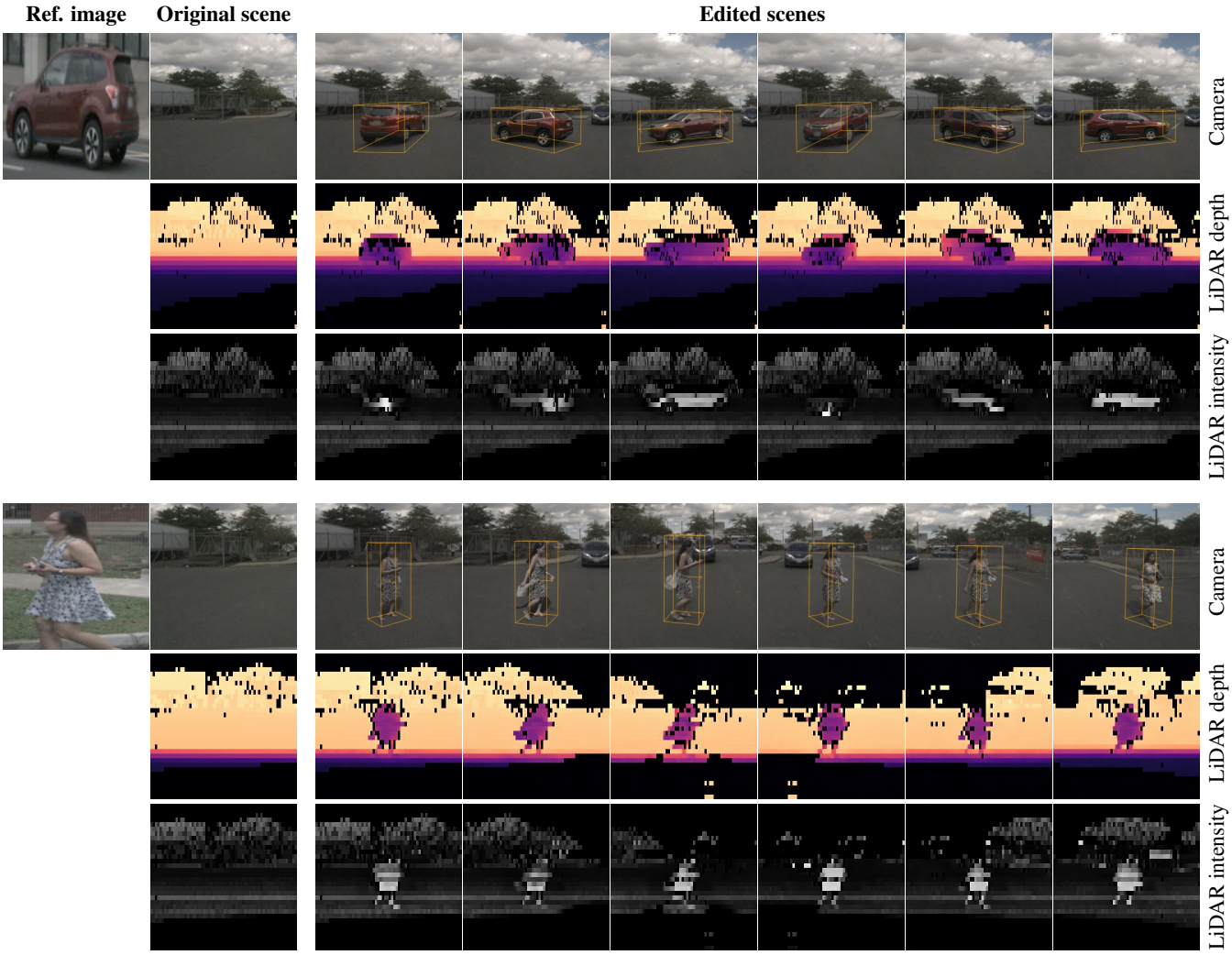


Figure S1. Additional examples showcasing our method's controllability. From left to right: reference image $\mathbf{x}_{\text{ref}}$ extracted from a seperate source scene, original destination scene (original RGB image $\mathbf{x}^{(\text{C})}$, LiDAR range depth $\mathbf{x}_0^{(\text{R})}$ and intensity $\mathbf{x}_1^{(\text{R})}$), and edited scenes.

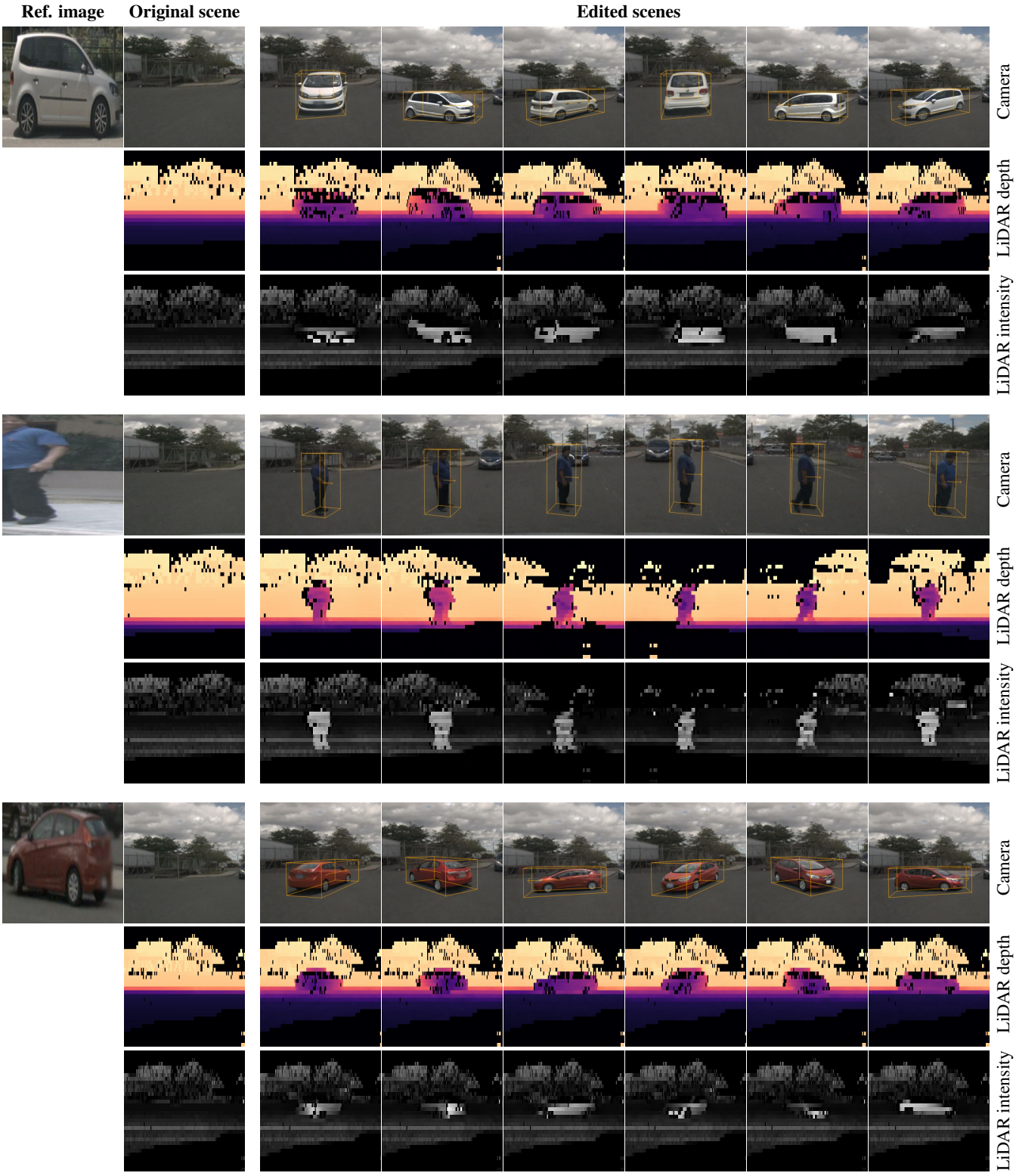


Figure S2. Additional examples showcasing our method's controllability. From left to right: reference image $\mathbf{x}_{\text{ref}}$ extracted from a seperate source scene, original destination scene (original RGB image $\mathbf{x}^{(\text{C})}$, LiDAR range depth $\mathbf{x}_0^{(\text{R})}$ and intensity $\mathbf{x}_1^{(\text{R})}$), and edited scenes.

Figure S3. Comparison of detection results between the original scene and the same scene with the object shown in red replaced. BEVFusion [33] achieves good detection performance on the object reinserted using our method, while leaving the boxes of the other objects undisturbed. Interestingly, even though the aspect of the car behind the reinserted object in the third column is changed slightly, it does not seem to affect detection much. We hypothesise that this is due to the fact that while the camera view is sensitive to occlusions, the range view is much less so since we reinsert only the points that are in the box used for conditioning, see Sec. 2.3. All detections are filtered using a score threshold of 0.08.

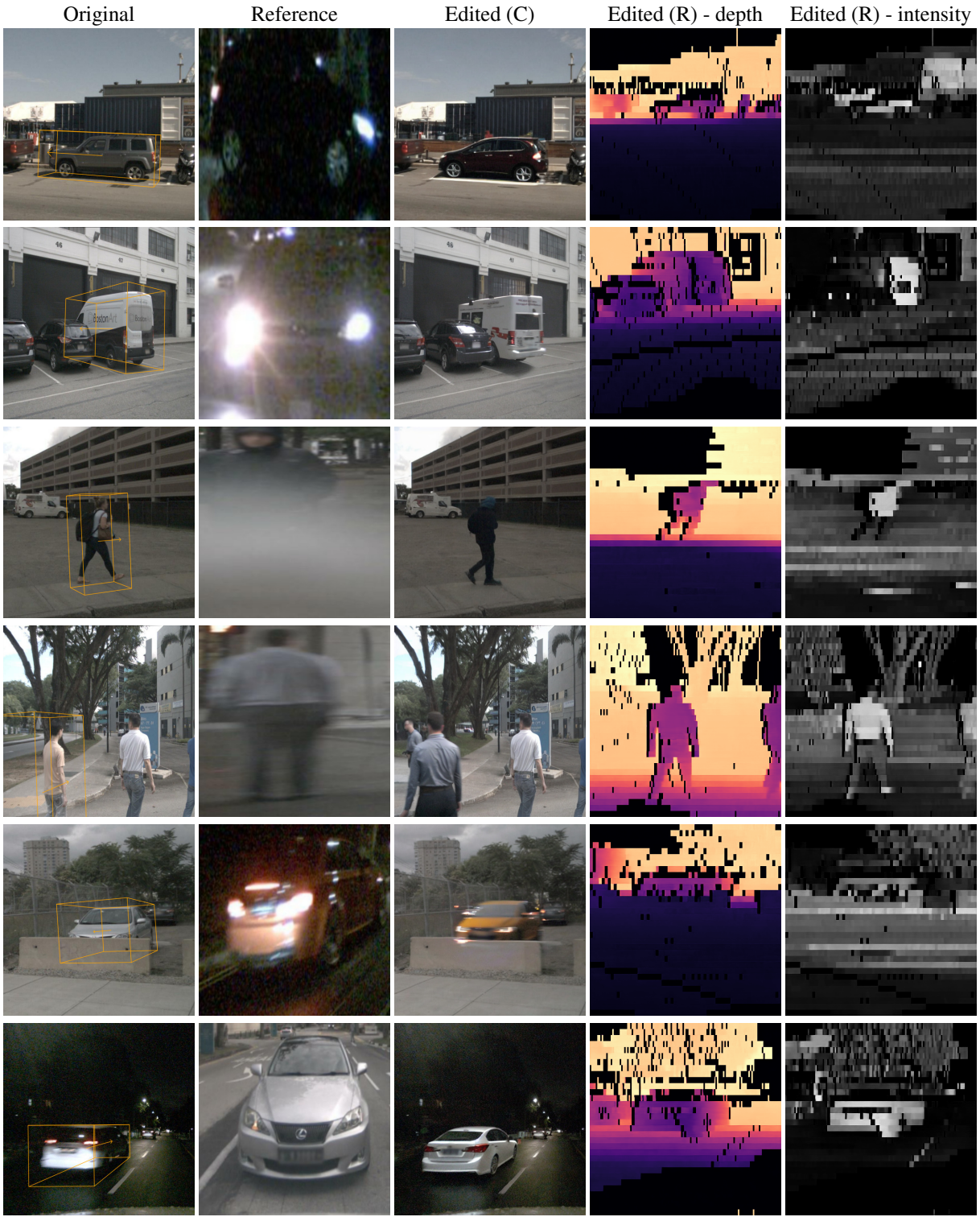
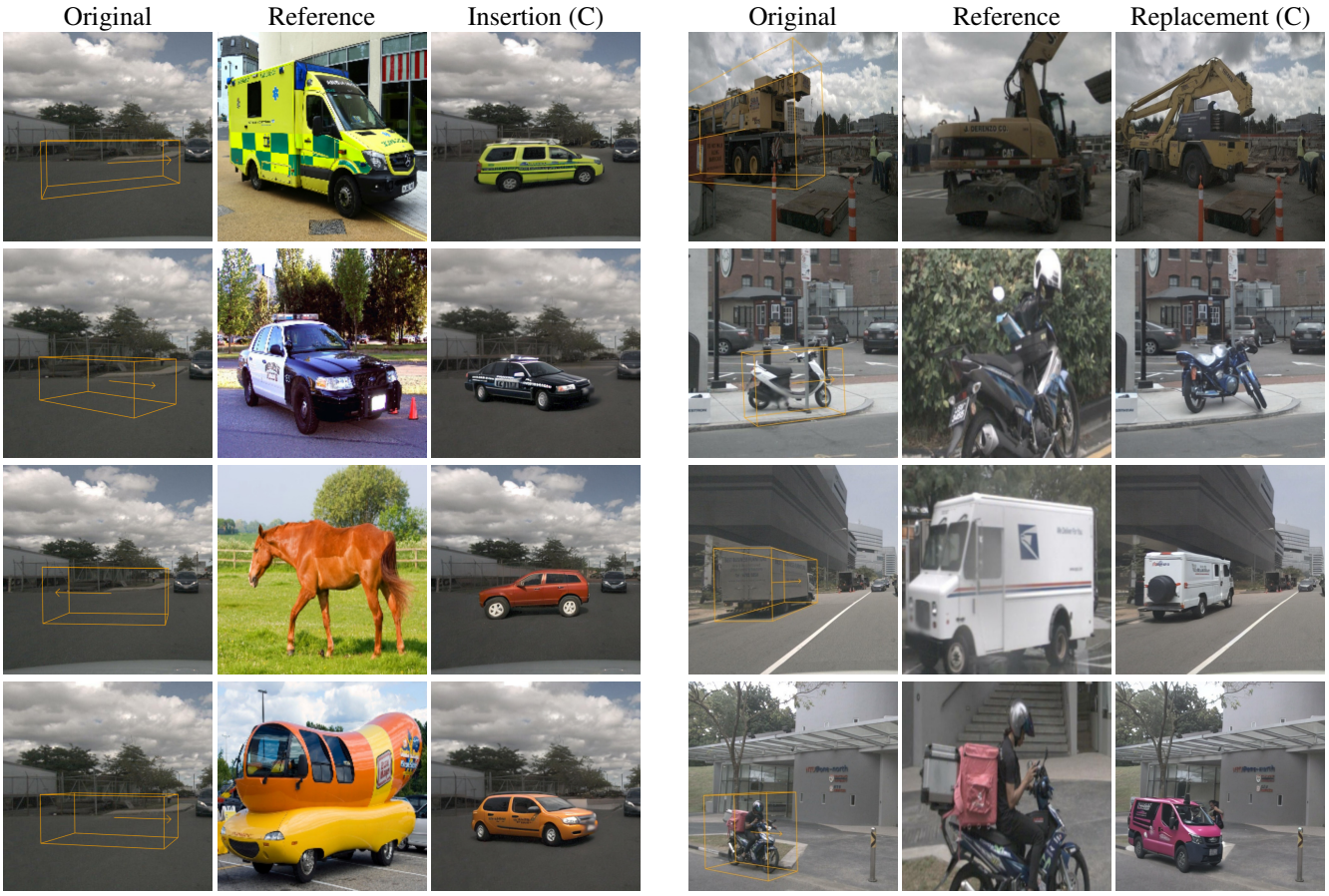


Figure S4. Object replacement results using hard references (different weather conditions or time of day, occlusions, etc.). Top three rows: MObI is able to insert these hard references in the target bounding box successfully while preserving the overall scene consistency. Bottom three rows: some examples of failure cases (a new pedestrian is hallucinated, the inserted car shows too much motion blur, the lightning is not coherent with the overall scene).



(a) (b)

Figure S5. Object insertion and replacement with out-of-domain and open-world references for MObI trained only on the pedestrian and car classes of nuScenes. (a) In the first two examples (top left), MObI inserts the correct object successfully but loses fine appearance details. In the last two examples (bottom left), MObI inserts a car instead of the object depicted by the reference. (b) In the first three examples (top right), MObI correctly replaces objects from classes outside of its training set, yet quality degrades. In the last example (bottom right), the model replaces the motorcycle with a small vehicle, reverting to a familiar class. Note that all examples have been correctly inserted in the target bounding box with the correct orientation.

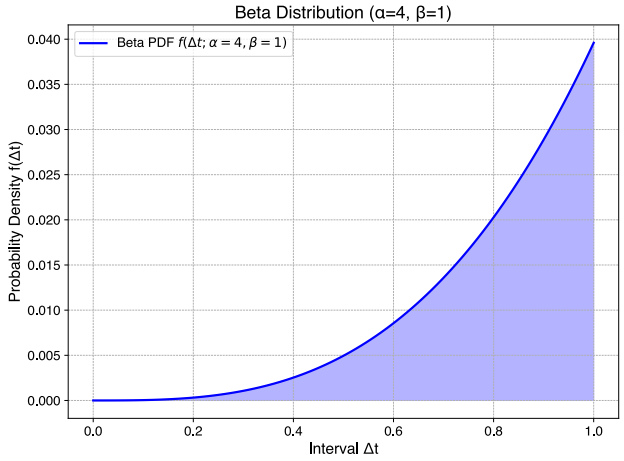


Figure S6. The probability density function of the Beta distribution with parameters $\alpha = 4$ and $\beta = 1$, used to sample reference patches of an object based on the normalized timestamp difference Δt between tracked instances. Patches from further time points are sampled with higher frequency.

Figure S7. Empty boxes are sampled during training for data augmentation, with the reference conditioning set to a black image and the bounding box coordinates set to zero.

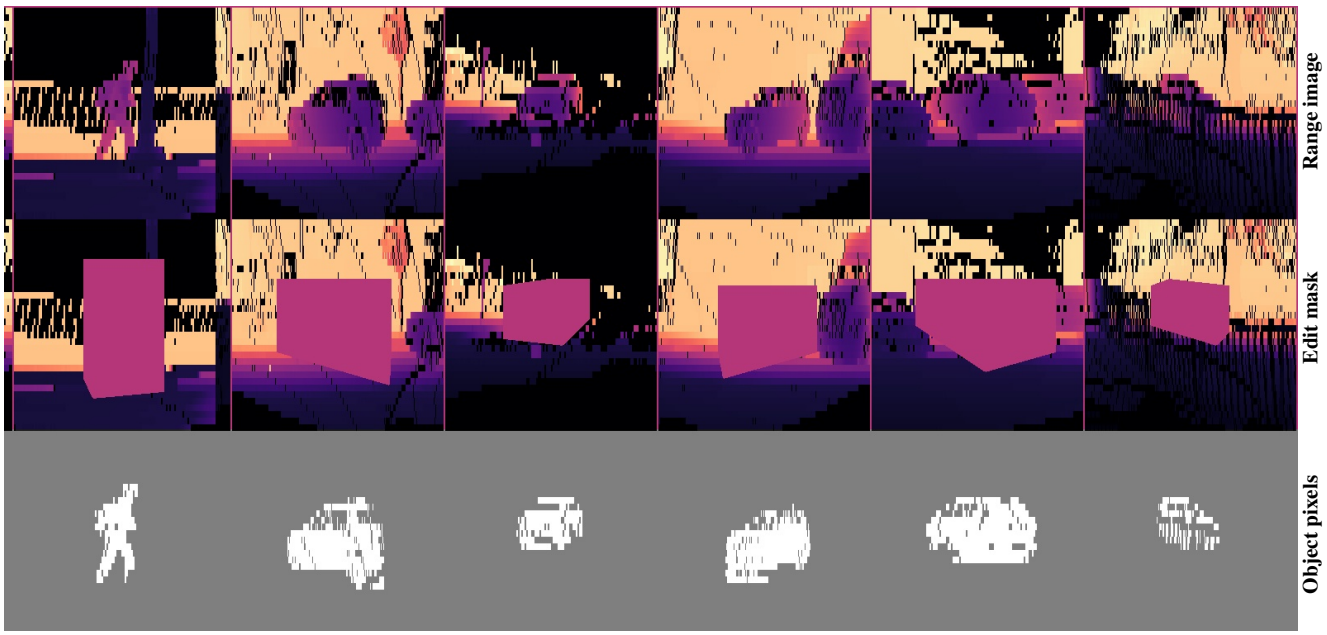


Figure S8. From top to bottom: (i) object-centric range depth image, (ii) range depth context with an edit mask, generated by projecting the object bounding box onto the range view, and (iii) object mask highlighting pixels corresponding to points within the 3D bounding box.