

# Sensor2Sensor: Cross-Embodiment Sensor Conversion for Autonomous Driving

## Supplementary Material

### A. Extended Qualitative Results

In this section, we provide an in-depth visual analysis to complement the quantitative results presented in the main paper. These figures are specifically designed to highlight the efficacy and generalization capabilities of our Sensor2Sensor pipeline across different output modalities. We present additional qualitative results covering image generation (Figure 11 and Figure 12), LiDAR generation (Figure 13), and image–LiDAR alignment (Figure 14). Finally, to best illustrate the temporal coherence and realism of our full pipeline, more video generation results are presented in the accompanying **supplementary video**.

### B. Implementation Details

#### B.1. Training Pipeline

Our model is trained in a four-step pipeline to progressively incorporate increasingly complex conditioning information.

- **Step 1: Base Conditioning Single Frame Generation.** The model is first trained on single frame generation, given conditional dashcam images.
- **Step 2: Previous Frame Conditioning.** The model is then fine-tuned with dense conditioning signals, including the latent representations of the previous frame’s camera and LiDAR data, as well as an additional dashcam view.
- **Step 3: DAgger Data Generation.** We use the model from Step 2 to generate a new dataset in a Dataset Aggregation (DAgger) fashion. The model is unrolled for multiple steps to create long-term simulations, which may include drifted data.
- **Step 4: DAgger Fine-tuning.** Finally, the model is fine-tuned on the DAgger-generated dataset from Step 3. This step involves training with augmented latent representations from the previous frame, which helps the model learn to correct its own errors and improves long-term simulation stability.

#### B.2. Model Architecture

The core of our generative model is a conditional diffusion model with a multi-stream UNet backbone designed for multi-modal sensor data.

**Backbone.** We employ a UNet architecture with temporal attention connections. It features separate processing streams for camera and LiDAR data, allowing the model to learn modality-specific representations while fusing information through shared attention layers. The UNet processes inputs from 8 surrounding camera views and one dashcam

view, along with the top-mounted LiDAR. The architecture uses a block structure with output channels of (320, 640, 1280, 1280).

**Variational Autoencoders (VAEs) [24]** We use separate, pre-trained VAEs to encode the raw sensor data into a compact latent space.

- **Image VAE:** A VAE [10] is used to encode the camera views into 8-channel latent representations.
- **LiDAR VAE:** A dedicated VAE encodes the raw LiDAR spin image into a 16-dimensional latent space. The UNet’s LiDAR stream is configured with 16 input and output channels to match this latent space. More details about LiDAR VAE training is shown in Section B.7.

**Conditioning Mechanisms** The generation process is guided by conditioning inputs.

- **Dashcam Conditions:** Current frame of dashcam is conditioned into the diffusion blocks by concatenate the feature with the denoising latents in the view dimension. During training, we incorporate random spatial masking (with a probability of 0.2) on the conditional dashcam frames. At inference time, we can leverage this capability to apply targeted masks over distractor elements (e.g., dashcam watermarks or the ego-vehicle hood), ensuring the model focuses solely on the relevant scene context.
- **Previous frame Conditions:** To ensure temporal consistency, the model is conditioned on the latent representations of the previous frame’s camera images and LiDAR scan. We achieve this by concatenating the latents from the previous timestep to the current frame’s latents along the channel dimension. Additionally, during training, we randomly drop this temporal conditioning with a probability of 0.5 to facilitate the learning of initial frame generation and improve robustness.

#### B.3. Training Details

The model is trained on 128 TPUs. We use the AdamW optimizer with a learning rate of  $5e-5$ . We clip the global norm of gradients at 1.0. For regularization, we randomly drop conditioning signals during training. For evaluation, we use an exponential moving average (EMA) of the model weights with a decay of 0.999. The training follows the multi-step pipeline described above, with each step fine-tuning from the checkpoint of the previous step. For step 1, 2, and 4, we train with 80k, 40k, and 20k steps, respectively. The number of model parameters is around 250M.

Input image:



A white car and a black car in front



Input image:



Two cars in front



Input image:



A red and a white car in front left



Figure 11. Additional qualitative results for image generation. Our proposed method demonstrates superior fidelity compared to the input dashcam image, accurately preserving the correct shape and color of objects, especially vehicles, which is challenging for the baselines.



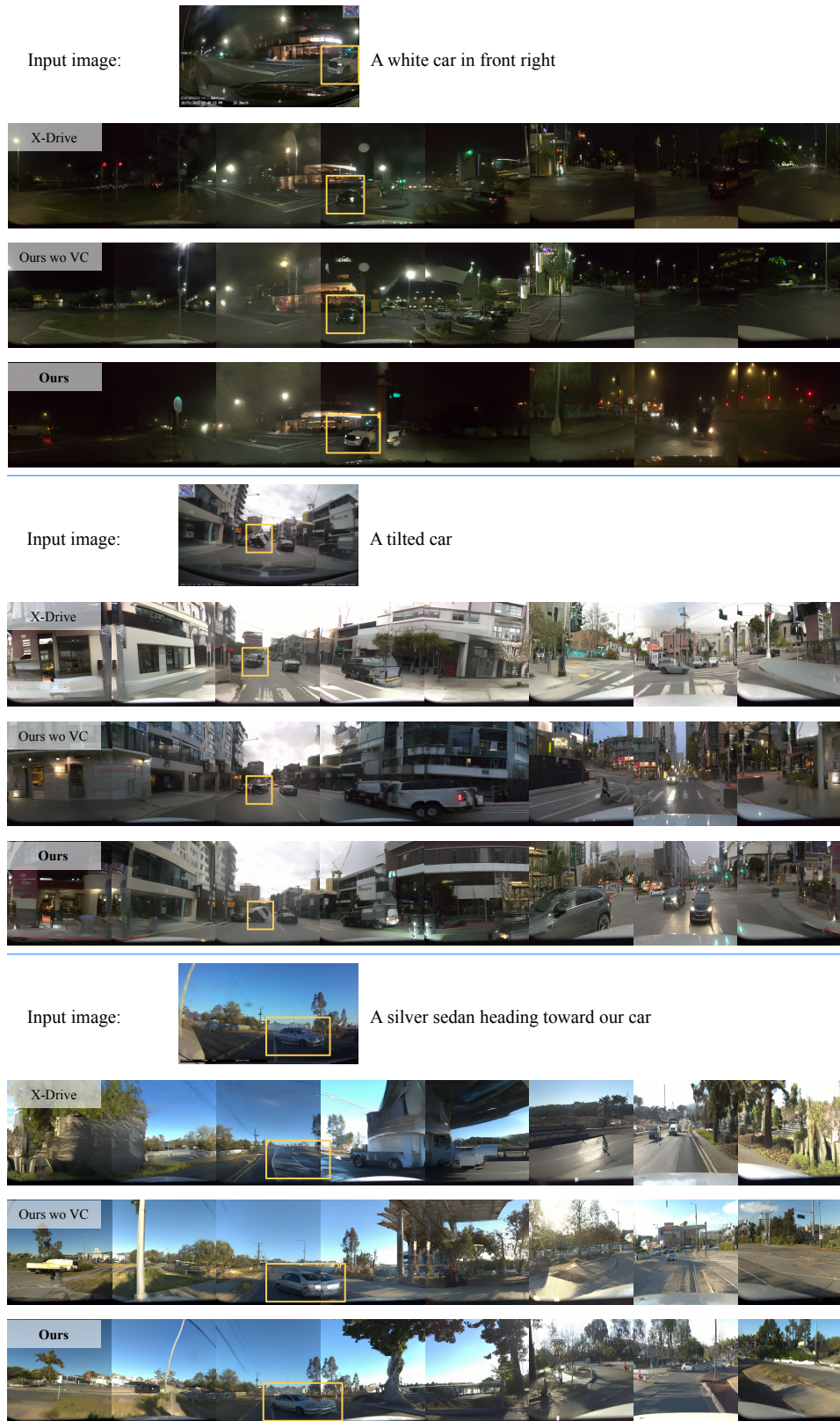


Figure 12. Additional qualitative results for image generation. Our proposed method demonstrates superior fidelity compared to the input dashcam image, accurately preserving the correct shape and color of objects, especially vehicles, which is challenging for the baselines.

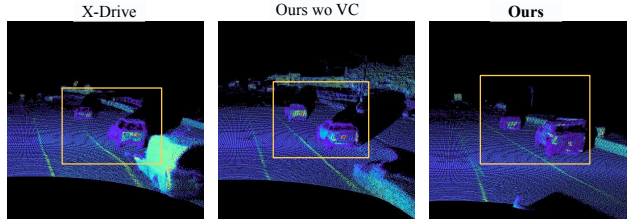
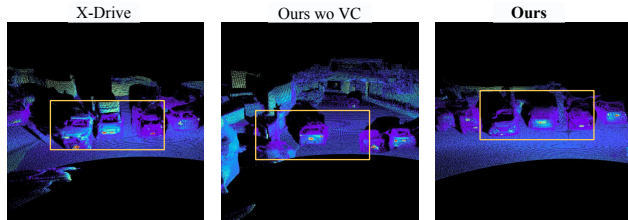
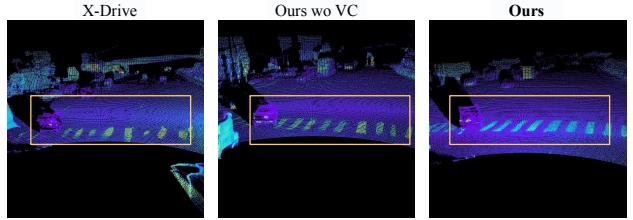
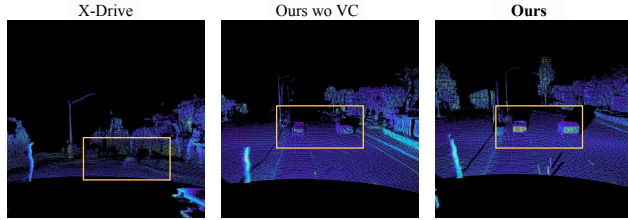
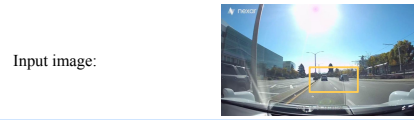
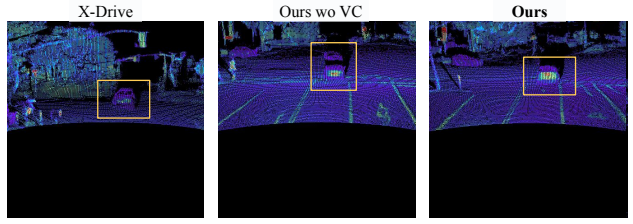
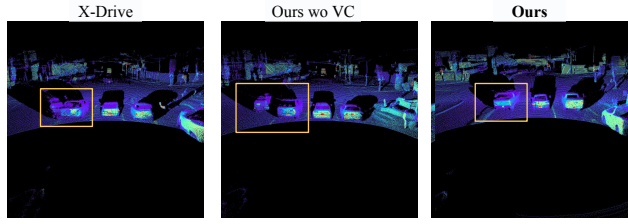


Figure 13. Additional qualitative results for LiDAR generation. Our method yields more accurate geometry in the synthesized point clouds, resulting in a less noisy output and a better correspondence with the accompanying image data. This improved fidelity allows for a more accurate preservation of the underlying spatial relationships of the scene.



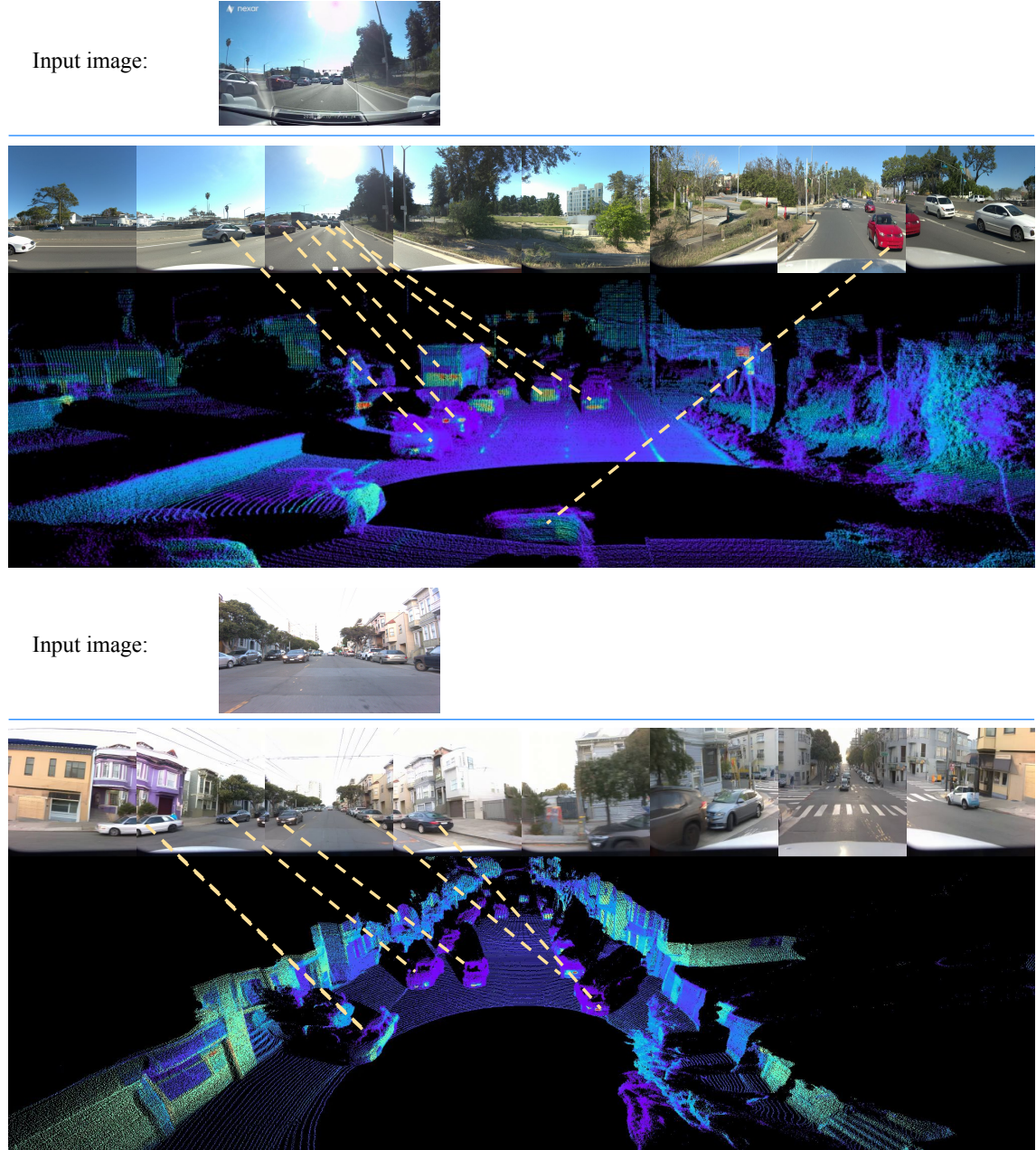


Figure 14. Additional qualitative results showcasing the Image-LiDAR alignment and cross-modal consistency achieved by our method. These visualizations confirm that the generated LiDAR point cloud accurately reflects the geometric details observed in the synthesized image view, demonstrating precise spatial registration between the two modalities.

#### B.4. Dataset Details

For training, we use a proprietary dataset of 100k 10s clips (8 cameras + top LiDAR) for 4DGS reconstruction. The resulting rendered images and synchronized sensor logs constitute the paired training data for diffusion models. For evaluation, quantitative analysis uses 1K paired 3s sequences from proprietary Fixed-Camera-to-AV logs. For in-the-wild evaluation, we collect diverse unconstrained in-

puts: internet videos, ADAS logs, and manually captured dashcam (e.g., Nexar) and smartphone footage.

#### B.5. Dashcam Parameter Distribution

Camera parameters are sampled via a two-stage process: (1) Extrinsics: We first select a vehicle category (e.g., Sedan, SUV), then sample 6-DoF poses from category-specific distributions (e.g., for Sedans: height 1.1–1.3m, forward translation 2.0–2.5m, pitch  $\pm 10^\circ$ ). (2) Intrinsic: Parameters are

drawn from a set of calibrated real-world dashcams (e.g., Nexar, VIOFO) and augmented with uniform noise (e.g.,  $\pm 5\%$  focal length). Final outputs undergo exposure compensation and gamma correction for lighting normalization.

## B.6. Different Target Camera Configurations

*Sensor2Sensor* is designed for multi-sensor flexibility via its raymap-conditioning architecture, which encodes camera intrinsics and extrinsics into the generation process. While the current results focus on our large-scale proprietary platform, the raymap ensures the model is not limited to a single configuration, as it learns the fundamental mapping between 3D rays and pixel intensities. To adapt to new platforms, our paradigm simply requires 4DGS-based paired data generation for the target sensor configurations.

## B.7. LiDAR VAE Training

We introduce a VAE architecture for generating LiDAR spin images, jointly encoding depth, intensity, and elongation. The encoder and decoder are both convolutional, and its latent space are regularized with a KL divergence loss. The normalized range, intensity, and elongation use an L1 reconstruction loss, while the validity reconstruction loss uses cross entropy. In addition, we add an LPIPS loss on surface normals (derived from predicted point cloud), intensity, elongation, and validity. The total loss, which we seek to minimize, is a weighted sum of all components, shown in Equation (1), with terms:  $\mathcal{L}_{\text{range}}^{\text{L1}} + \mathcal{L}_{\text{elongation}}^{\text{L1}} + \mathcal{L}_{\text{intensity}}^{\text{L1}} + \mathcal{L}_{\text{validity}}^{\text{BCE}} + \mathcal{L}_{\text{normals}}^{\text{LPIPS}} + \mathcal{L}_{\text{elongation}}^{\text{LPIPS}} + \mathcal{L}_{\text{intensity}}^{\text{LPIPS}} + \mathcal{L}_{\text{validity}}^{\text{LPIPS}} + \mathcal{L}^{\text{KL}}$ . In this formulation, the  $\mathcal{L}_{\text{normals}}^{\text{LPIPS}}$  term uses normals  $\mathbf{f}_{\text{normals}}^L = \text{ComputeNormals}(\mathbf{f}_{\text{range}}^L)$  that are computed based on finite differences using the projected 3D lidar points. We now define each loss term individually.

**L1 Reconstruction Loss.** For the signal components using L1 loss (range, elongation, and intensity), the loss is defined as:

$$\mathcal{L}_{\text{signal}}^{\text{L1}} = \lambda_{\text{signal}} \|\mathbf{f}_{\text{signal}}^L - \hat{\mathbf{f}}_{\text{signal}}^L\|_1 \quad (3)$$

where “signal” represents range, elongation, or intensity. In this equation,  $\mathbf{f}_{\text{signal}}^L$  is the ground truth LiDAR feature map and  $\hat{\mathbf{f}}_{\text{signal}}^L$  is its corresponding reconstruction from the VAE. The term  $\lambda_{\text{signal}}$  is a scalar hyperparameter that weights the contribution of this specific loss component.

**Binary Cross-Entropy Loss.** The cross-entropy loss on the validity mask is calculated by:

$$\begin{aligned} \mathcal{L}_{\text{validity}}^{\text{BCE}} = & -\lambda_{\text{BCE}} [\mathbf{f}_{\text{valid}}^L \log(\hat{\mathbf{f}}_{\text{valid}}^L) \\ & + (1 - \mathbf{f}_{\text{valid}}^L) \log(1 - \hat{\mathbf{f}}_{\text{valid}}^L)] \end{aligned} \quad (4)$$

where  $\mathbf{f}_{\text{valid}}^L$  is the ground truth binary validity mask (with values 1 for valid returns and 0 otherwise) and  $\hat{\mathbf{f}}_{\text{valid}}^L$  is the predicted validity probability map output by the decoder. The  $\lambda_{\text{BCE}}$  is its corresponding loss weight.

**LPIPS Perceptual Loss.** The LPIPS (Learned Perceptual Image Patch Similarity) [54] loss measures the perceptual distance between a reference image  $x$  and a distorted image  $\hat{x}$ . Unlike traditional metrics like L1 or MSE, LPIPS leverages features extracted from a pre-trained deep neural network (e.g., VGG [36]). The loss, presented in the equation

$$\mathcal{L}_{\text{LPIPS}}(x, \hat{x}) = \sum_i \frac{1}{H_i W_i} \sum_{h,w} \|w_i \odot (y_{hw}^i - \hat{y}_{hw}^i)\|_2^2, \quad (5)$$

is computed by feeding both images through the network and calculating a weighted distance between their internal activations. In this formulation,  $i$  indexes the network layers used for the comparison. At a given layer  $i$ , the terms  $\hat{y}_{hw}^i$  and  $y_{0,hw}^i$  represent the feature activation vectors at spatial position  $(h, w)$  for images  $x$  and  $x_0$ , respectively, which have been unit-normalized along the channel dimension. The total height and width of the feature map at this layer are given by  $H_i$  and  $W_i$ , allowing the  $\frac{1}{H_i W_i} \sum_{h,w}$  operation to compute the spatial average of the distances. The difference between activations is scaled by  $w_i$ , a learned channel-wise weight vector optimized to match human perceptual judgments, via the element-wise product ( $\odot$ ). The squared L2 norm ( $\|\cdot\|_2^2$ ) is then used to compute the distance between these weighted vectors. Finally, the total  $\mathcal{L}_{\text{LPIPS}}$  is the sum of these spatially-averaged distances across all included layers  $i$ .

The LPIPS loss on the signals (normals, elongation, intensity, and validity) is calculated by:

$$\mathcal{L}_{\text{signal}}^{\text{LPIPS}} = \lambda_{\text{signal}} \mathcal{L}_{\text{LPIPS}}(\mathbf{f}_{\text{signal}}^L, \hat{\mathbf{f}}_{\text{signal}}^L) \quad (6)$$

Here,  $\lambda_{\text{signal}}$  is the corresponding weighting factor for each specific signal type.

**KL Divergence Regularization.** The KL divergence loss, which regularizes the latent space to follow a standard normal distribution, is calculated by:

$$\mathcal{L}^{\text{KL}} = \frac{1}{2} \lambda_{\text{KL}} \sum_{j=1}^D (\mu_j^2 + \sigma_j^2 - \log(\sigma_j^2) - 1) \quad (7)$$

This term represents the Kullback-Leibler divergence between the encoder’s output distribution,  $\mathcal{N}(\mu, \sigma^2)$ , and the prior,  $\mathcal{N}(\mathbf{0}, \mathbf{I})$ . Here,  $D$  is the dimensionality of the VAE’s latent space. For each latent dimension  $j$ , the encoder outputs a mean  $\mu_j$  and a variance  $\sigma_j^2$ . Finally,  $\lambda_{\text{KL}}$  is the hyperparameter that balances this regularization term against the reconstruction losses.

## B.8. DAgger Training

Dataset Aggregation (DAgger) [34] is originally an imitation learning algorithm designed to mitigate the compounding errors of behavioral cloning. It iteratively collects and

aggregates data by querying an expert  $\pi^*$  for optimal actions  $a^*$  on states  $s$  visited by the current policy  $\pi_i$ . A new policy  $\pi_{i+1}$  is then trained on this aggregated dataset. We adapt DAGger to autoregressive video generation, treating it as a sequential decision-making process to combat temporal inconsistency.

In our case, we introduce DAGger for Video Generation. We map the components as follows. (a) Policy  $\pi$ : the video generation model, which predicts the next frame. (b) State  $s_t$ : the sequence of previously generated frames,  $s_t = \{f_1, f_2, \dots, f_t\}$ . (c) Action  $a_t$ : the generated next frame,  $a_t = f_{t+1}$ . (d) Expert  $\pi^*$ : a mechanism (e.g., human evaluator, critic model, or ground-truth data) that provides a “correct” next frame  $a_t^*$  given a policy-generated state  $s_t$ .

First, we train a base model to generate the current frame conditioned on the ground-truth previous frame. We then utilize the base model to auto-regressively generate rollout frames for all segments in the training set. These generated frames serve as a “degraded” dataset for augmentation. We train an improved model by randomly substituting the ground-truth history with these generated frames during training. This exposes the model to its own accumulation errors, making this model significantly more robust than the base model. While this process can be repeated iteratively, we find that a single iteration yields satisfactory rollout quality. For the DAGger training phase, we set the rollout horizon to 6 steps. Although each training segment contains approximately 35 frames, we find that training on this shorter rollout window is empirically sufficient to achieve robust performance, avoiding the computational cost of full-sequence training.

### C. Limitations and Potential Solutions

Our approach achieves state-of-the-art per-frame generative quality, with our multi-modal diffusion model serving as a high-fidelity backbone for static scenes. We then leverage this powerful single-frame model for video synthesis by extending it auto-regressively, conditioning each new frame on the previously generated one. A limitation is, while our DAGger finetuning strategy effectively mitigates short-term error accumulation, temporal drift remains a known challenge for long-horizon sequences (e.g.,  $> 30$  seconds). Over extended rollouts, minor prediction errors such as small geometric drifts in LiDAR or slight visual inconsistencies, can compound. This may lead to a gradual loss of long-range temporal coherence or a perceived drift in sensor calibration. However, this limitation could be addressed by incorporating a more robust longer video generative backbone designed for long-range consistency. A complementary, and more immediate, solution would be to expand the auto-regressive conditioning window. Instead of conditioning only on the single prior frame ( $t - 1$ ), the model could attend to a richer temporal context (e.g.,  $t - k, \dots, t - 1$ ).



Figure 15. Visualization of synthetic dashcam images rendered from 4DGS across diverse camera settings. The renders demonstrate high visual fidelity and realism, effectively simulating the characteristics of in-the-wild footage used for training.

This would provide stronger priors for maintaining object-level and scene-level consistency over time. While outside the primary scope of this work, we leave these promising directions for future exploration.

### D. Synthetic Cameras from 4DGS

One important component of our pipeline is the utilization of 4D Gaussian Splatting (4DGS) to synthesize paired training data by simulating third-party camera views. As illustrated in Figure 15, the synthetic dashcam images rendered via our 4DGS pipeline exhibit high photorealism, faithfully mimicking the optical characteristics and environmental complexity of real-world dashcam footage.

Crucially, our diffusion model is trained to map these synthetic inputs ( $I_{synth}$ ), which may contain minor reconstruction artifacts such as floaters or slight blur, to pristine, ground-truth real sensor data ( $O_{real}$ ). This training objective effectively functions as a denoising task, forcing the network to learn robust spatial and semantic mappings between the monocular view and the target sensor suite, rather than overfitting to low-level input artifacts. Consequently, at inference time, the model demonstrates significant robustness when presented with sub-optimal or noisy real-world dashcam inputs, successfully generating coherent and geometrically consistent AV logs.