Flux4D: Flow-based Unsupervised 4D Reconstruction

Jingkang Wang[1,2*] **Henry Che**[1,3*†] **Yun Chen**[1,2*] **Ze Yang**[1,2]
Lily Goli[1,2†] **Sivabalan Manivasagam**[1,2] **Raquel Urtasun**[1,2]
Waabi[1] University of Toronto[2] UIUC[3]
https://waabi.ai/flux4d

Abstract

In the supplementary material, we provide detailed information on our method and the baselines, additional experimental results, and a discussion of limitations and future work. We begin by describing the implementation details of *Flux4D* in Sec. A. After that, in Sec. B we present how the baseline models were adapted to our experimental settings, which are described in Sec. C. Next, we show additional experimental results in Sec. D, including future prediction with *Flux4D*, ablation studies, and more results for controllable simulation. We also include additional discussions and comparisons of *Flux4D* with closely related works in Sec. E. Moreover, we provide the limitations of *Flux4D* and future directions in Sec. F. Finally, we discuss *Flux4D*'s broader impact (Sec. G), computation resources (Sec. H), and asset's licenses (Sec. I). Additionally, we include a supplementary video, **supplementary.mp4**, which offers an overview of our methodology and showcases video results of *Flux4D* for scalable reconstruction and simulation.

A *Flux4D* Details

A.1 Motion Enhancement Details

Velocity reweighting: To further enhance the flow estimation for dynamic actors, we employ a pixel-wise reweighting scheme in loss computation based on the predicted velocity magnitude. This is used in place of $\mathcal{L}_{\text{rgb}}$ during training, and is defined as:

$$\mathcal{L}_{\text{rgb\_vw}} = (1 + ||sg(\mathbf{v}_\mathbf{r})||) \cdot \mathcal{L}_{\text{rgb}}, \tag{1}$$

where $sg(\cdot)$ denotes the stop gradient operation, $\mathbf{v}_\mathbf{r}$ is the rendered velocity in image space, and $\mathcal{L}_{\text{rgb}}$ is the photometric loss. For stability, we clip the values of $\mathbf{v}_\mathbf{r}$ to $[0, 10]$.

Polynomial motion modeling: In addition to the constant velocity assumption, we also consider a polynomial motion model to better handle more complex actor behaviors. The velocity is parameterized as a polynomial function of degree ℓ. Specifically, rather than predicting a single constant velocity, we predict polynomial velocity parameters $\mathbf{v} = [\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_\ell]$, such that

$$\mathbf{p}_i^{t'} = \mathbf{p}_i^{t_i} + \sum_{j=0}^{\ell} \mathbf{v}_j \frac{(t'^{j+1} - t_i^{j+1})}{j+1}. \tag{2}$$

In our experiments, we set $\ell = 1$, which corresponds to the constant acceleration assumption. We also explored higher-order terms with $\ell = 2, 3$, but did not observe noticeable gains.

A.2 Model Architecture

Initialization: Given a sequence of input images and corresponding LiDAR point clouds, we initialize the Gaussian parameters as follows:

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

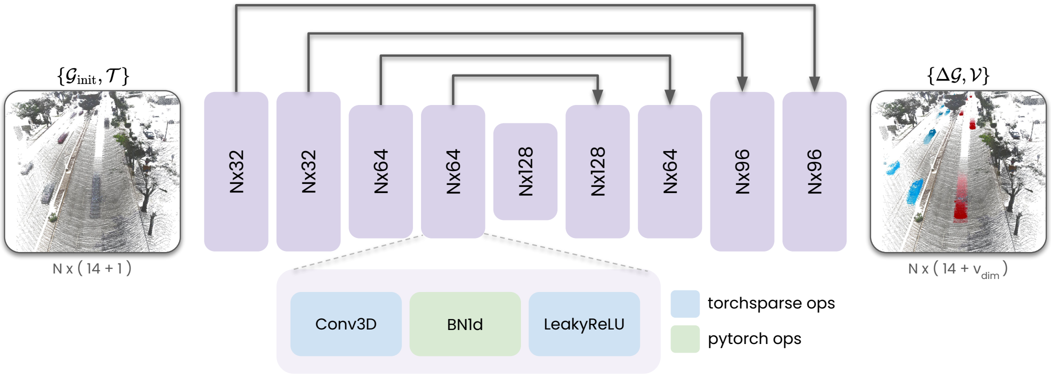

Figure A1: ***Flux4D-base* model architecture.**

- **Position:** Each Gaussian is initialized at the location of its corresponding LiDAR point.
- **Color:** For each image at a given frame, we project the LiDAR points at that frame onto the image and sample their colors from the image.
- **Scale:** The scale of each Gaussian is set to the average distance to its three nearest neighbors. Following [4, 19], we apply a logarithmic transformation and `softplus` activation to stabilize training.
- **Orientation:** The orientation of each Gaussian is randomly initialized and then normalized.
- **Opacity:** The opacity of each Gaussian is initialized to 0.5.
- **Time:** Each Gaussian inherits the timestamp of its corresponding LiDAR point, which is then normalized to the range $[0, 1]$.

To model the sky, we compute the axis-aligned bounding box (AABB) of the initialized Gaussians above, and randomly sample 1 million sky points in the upper-half of the sphere centered at the center of the AABB, with radius set to 4 times the length of the AABB.

Flux4D-base: We employ a Sparse 3D UNet (Fig. A1) adopted from [15] for our Flux4D-base reconstruction model f_θ. The input to f_θ are $\mathcal{G}_{\text{init}} = \{\mathbf{g}_{\text{init},i} : \mathbf{g}_{\text{init},i} \in \mathbb{R}^{14}\}$ and $\mathcal{T} = \{t_i : t_i \in \mathbb{R}^1\}$. The first 3 channels of each $\mathbf{g}_i$ are the 3D Gaussian location ($\mathbf{p}_i \in \mathbb{R}^3$), following by 4 channels for quaternions ($\mathbf{q}_i \in \mathbb{R}^4$), 3 channels for scales ($\mathbf{s}_i \in \mathbb{R}^3$), 1 channel for opacity ($\mathbf{o}_i \in \mathbb{R}^1$), and the last 3 for colors ($\mathbf{c}_i \in \mathbb{R}^3$). All input are concatenate on the last dimension, resulting in the input of tensor of shape $N \times 14$. Although we observe a marginal improvement in reconstruction quality with spherical harmonics, we omit them from our representation for simplicity, with no compromise in motion quality. Each block contains a `spnn.Conv3D` layer follows by `nn.BatchNorm1d` and `spnn.LeakyReLU`, where `nn` is the standard `PyTorch` [10] operation and `spnn` is the `Torchsparse` [15] operation. The skip connections are applied by concatenating down features with up features channel-wise. The output are the per-gaussian residues $\Delta\mathcal{G} = \{\Delta\mathbf{g}_i : \Delta\mathbf{g}_i \in \mathbb{R}^{14}\}$ and velocity $\mathcal{V} = \{\mathbf{v}_i : \mathbf{v}_i \in \mathbb{R}^{v_{\text{dim}}}\}$. Here, $v_{\text{dim}} = 3(\ell + 1)$ following the convention from Sec. A.1, where $\ell = 0$ for constant velocity assumption and $\ell = 1$ for constant acceleration.

Rendering: To render the scene at a given time t, for each inital Gaussian $\mathbf{g}_{\text{init},i}$, we first apply the residue $\Delta\mathbf{g}_i$ to obtained the refined gaussian $\mathbf{g}_i$:

$$\mathbf{g}_i = \mathbf{g}_{\text{init},i} + \Delta\mathbf{g}_i = \{\mathbf{p}_{\text{init},i} + \Delta\mathbf{p}_i, \mathbf{q}_{\text{init},i} + \Delta\mathbf{q}_i, \mathbf{s}_{\text{init},i} + \Delta\mathbf{s}_i, \mathbf{o}_{\text{init},i} + \Delta\mathbf{o}_i, \mathbf{c}_{\text{init},i} + \Delta\mathbf{c}_i, \},$$

Afterwards, we apply the velocity $\mathbf{v}_i$ to the Gaussian position p_i to obtain the position at time t following Eqn. (2), obtaining the final $\mathcal{G}_t$. We use `gsplat` [30] for Gaussian rendering. Before passing $\mathcal{G}_t$ to `gsplat`, we activate the Gausisan parameters by appling sigmoid function to the opacity and color of each Gaussian. We also clamp the scale to $[0m, 1m]$ to stabilize the training.

Iterative refinement: For iterative refinement module r_θ, we use the same UNet architecture as above, but take the gradient of the previous step's residual as additional input. We denote $\nabla\mathcal{G}^{(i)}$ as the gradient of the Gaussian parameters after i^{th} iteration ($\mathcal{G}^{(i)}$), and is obtained by storing the detached gradients of $\Delta\mathcal{G}^{(i)}$ after the backward pass. Here, $\Delta\mathcal{G}^{(i)}$ is the output residue of r_θ after the i^{th} iteration. For the *Flux4D* model with N iteration, the final refined Gaussian $\mathcal{G}$ is obtained by:

$$\mathcal{G} = \mathcal{G}_{\text{init}} + \sum_{j=1}^{N-1} \Delta\mathcal{G}_i^{(j)} = \{\mathbf{g}_i : \mathbf{g}_i = \mathbf{g}_{\text{init},i} + \sum_{j=1}^{N-1} \Delta\mathbf{g}_i^{(j)}\},$$

where

$$\Delta\mathcal{G}_i^{(j)} = \begin{cases} f_\theta(\mathcal{G}_{\text{init}}, \mathcal{T}), & \text{if } j = 0, \\ r_\theta(\mathcal{G}^{(j-1)}, \mathcal{T}, \nabla\mathcal{G}^{(j-1)}), & \text{if } j > 0. \end{cases}$$

In all experiments, we set $N = 3$. We observe that increasing N (*e.g.*, $N = 10$) can improve reconstruction quality but comes at the cost of reduced inference speed.

A.3 Pseudocode

***Flux4D*:** We provide the pytorch-style pseudocode of *Flux4D* in Algorithm 1.

Algorithm 1 Pseudocode of *Flux4D* in PyTorch style.

```
# Network: neural network for predicting Gaussians
# G_init: Gaussians initialized from LiDAR+image, [N x 14]
# T: per-Gaussian normalized time captured, [N x 1]
# G_delta: predicted Gaussians residue, [N x 14]
# V: velocity of each Gaussian, [N x v_dim]

# Predict updated Gaussians
G_delta, V = Network(G_init, T)

# Apply residue to Gaussians
G = G_init + G_delta

L_recon = 0
for t_target in range(F): # Iterate over each frame
    # Move Gaussians to time t using velocities
    G_t = G.clone()
    G_t.means = G.means + V * (t_target - G.t)

    # Render scene with updated Gaussians
    I_rendered_t = render_gaussians(G_t) # Nx(3+1)

    # Compute losses
    L_recon += F.l1_loss(I_rendered_t, I_gt_t)

# velocity regularization
L_vel = torch.mean(torch.norm(V, p=2, dim=-1))

# Backpropagation and optimization
loss = L_vel + L_recon
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Iterative refinement: The pseudocode for iterative refinement as described in Sec. A.2 can be found in Algorithm 2.

Camera simulation: Additionally, we provide the pseudocode for dynamic clustering as seen in Fig. A8, A9 in Algorithm 3.

B Baseline Implementation Details

NeuRAD [16]: NeuRAD models dynamic scenes using compositional neural radiance fields while introducing techniques to handle complex sensor effects such as rolling shutter, ray dropping, and beam divergence. We use the publicly available implementation[†] in `neurad` mode and train the

[†] https://github.com/georghess/neurad-studio

Algorithm 2 **Pseudocode for Iterative Refinement in Pytorch style.**

```
# F_Network: Flux4D-base neural network (f_theta)
# R_Network: FLux4D refinement network (r_theta)
# N: number of iterations, we fix N = 3

G = G_init
G_grad, V = None, None

# Iterative refinement
for step in range(N):
    # Predict first iteration's residue
    if step == 0:
        G_delta, V = F_Network(G, T)

    # Predict subsequent iterations' residue
    else:
        G_delta = R_Network(G, T, G_grad)

    # Apply residue to Gaussians
    G = G + G_delta

    # Compute Loss following Flux4D Pseudocode
    L = compute_loss(G, T, V)

    # Backpropagation
    L.backward()

    # Store gradients for next iteration
    G_grad = G.grad.clone()
```

Algorithm 3 **Pseudocode for Dynamic Clustering in Python style.**

```
# G: the refined Gaussian set (i.e., after applying the residue), [N x 14]
# V: per-Gaussian velocity, [N x 3]
# V_thresh: threshold for dynamic filtering, we fix V_thresh=5

# Compute the velocity norm of each Gaussian
V_normed = torch.norm(V, dim=1)

# Obtain dynamic Gaussians
dynamic_G = G[V_normed > V_thresh]

# Perform clustering on dynamic_G's positions obtain moving instances
instances = DBSCAN(eps=0.4, min_samples=10).fit_predict(G.means) # NxC
unique_instances = torch.unique(instances)

# Obtain per-instance Gaussians
instance_G = []
for instance_id in unique_instances:
    instance_G.append(G[instances == instance_id])
```

models for 10,000 iterations to ensure convergence in 1.5s snippets. Training for only 5,000 iterations results in noticeable color artifacts, likely due to insufficient adaptation of the convolutional neural network. For 8s full-log reconstruction, we train for 20,000 iterations.

Street Gaussian [24]: Street Gaussian replaces NeRF-based representations [28, 16] with compositional 3D Gaussian Splatting (3DGS), enabling real-time camera simulation. We adopt `drivestudio`'s offical implementation[‡] with default hyperparameters, including density control and the learning rate schedule. The 3D Gaussians are initialized with 800,000 downsampled aggregated LiDAR points and a randomly sampled subset of 200,000 points. The model is trained for 10,000 iterations for 1.5s snippet reconstruction and 20,000 iterations for 8s full-log reconstruction.

EmerNeRF [26]: EmerNeRF is a self-supervised method for reconstructing 4D neural scene representations. It decomposes static and dynamic components while learning 3D scene flows without relying on ground truth object annotations. We adopt the public implementation[§] and follow the configuration incorporating a dynamic encoder and flow encoder. Following OmniRe [4], we use

[‡] https://github.com/ziyc/drivestudio

[§] https://github.com/NVlabs/EmerNeRF

SegFormer [22] to extract sky masks for training. The model is trained for 10,000 iterations for 1.5s snippet reconstruction and 20,000 iterations for 8s full-log reconstruction.

DeSiRe-GS [11]: DeSiRe-GS is a 3DGS-based representation designed for self-supervised static-dynamic decomposition and high-quality surface reconstruction in driving scenes. It follows a two-stage pipeline: the first stage extracts 2D motion masks by computing feature differences between rendered and ground-truth images, while the second stage distills this 2D motion information into Gaussian space using PVG [3]. We use the public implementation[¶] and train for 5,000 iterations per stage for 1.5s snippet reconstruction. For 8s full-log reconstruction, we train for 20,000 iterations in the first stage and 30,000 iterations in the second stage.

L4GM [13]: We fine-tune the officially released L4GM `big` model on PandaSet [21] for novel view synthesis (interpolation). Each training sample consists of 11 consecutive frames with rescaled cameras and scene depth, ensuring that the furthest camera from the scene center matches L4GM's default camera radius of 1.5. Our modified pipeline uses a single camera view per frame, taking six even-numbered frames as input to predict all frames in the sequence. The network is supervised with photometric and LPIPS losses, along with a 0.1-weighted depth loss aligning ray termination with ground truth LiDAR depth. While depth supervision slightly improves reconstruction quality, higher weights lead to catastrophic forgetting. Despite this, the predicted depth remains low-range, mostly capturing forward-facing scenes with flat geometry.

DepthSplat [23]: We implement a feed-forward, multi-view generalizable 3DGS baseline inspired by [23]. Our model employs a U-Net with residual blocks to process six input frames, each containing RGB, depth, and 3D point information. To enhance geometric accuracy in outdoor scenes, we incorporate LiDAR data as a depth prior by aggregating LiDAR points across multiple frames and rendering them as depth maps for additional input features. The network predicts Gaussian parameters per pixel. For dynamic scenes, we extend the model with motion modeling by predicting velocity. The input is augmented with a binary dynamic mask to identify moving objects, while the output is expanded to a velocity-augmented Gaussian representation, including a 3D velocity vector. During rendering, point positions are updated similarly to *Flux4D*. We constrain velocities using a `tanh` activation and introduce the same "as static as possible" regularization for velocity.

G3R [2]: Gradient guided generalizable reconstruction (G3R) is a framework that enables efficient and high-quality 3D scene reconstruction for large-scale scenes using iterative gradient feedback. In our implementation of G3R, we adhere to a source-novel split: six frames serve as the source to compute gradient input, four frames for interpolation, and five frames for extrapolation during supervised training. We utilize SparseUNet from `torchsparse`, training the model for 24,000 (1000×24) iterations. Gaussians are initialized by aggregating LiDAR points, utilizing labels for both static backgrounds and dynamic actors. We use labels to transform actor gaussians during rendering. For full 8-second reconstructions, we follow the same setting as [2]. For 1-second reconstructions, we limit the number of points to 800,000 to better reflect real-world conditions.

STORM [25]: STORM is a generalizable, self-supervised method for dynamic scene reconstruction from multi-view RGB. It employs a feed-forward transformer to jointly predict per-pixel 3D Gaussian primitives and scene flow. We use the authors' public implementation[‖] with default hyperparameters unless noted, and fine-tune for 100,000 iterations to reconstruct 1.5 s snippets on 8×H100 GPUs with a global batch size of 64 (image resolution: 270×480). We also fine-tune the released pre-trained checkpoint on PandaSet. Both training regimes yield comparable performance.

C Experiment Details

C.1 Dataset Splits

Pandaset: For all experiments with Pandaset [21], namely interpolation (Tab. 1), full-log reconstruction (Tab. 2), future prediction (Tab. 4), scaling law (Fig. 8 top), and ablation (Tab. 5, 6), we use logs `001,011,016,065,084,090,106,115,123,158` for testing and the remaining for training following [29, 16].

[¶] https://github.com/chengweialan/desire-gs

[‖] https://github.com/NVlabs/GaussianSTORM

Waymo Open Datset: We conduct experiment with DrivingRecon [8] on WOD [14] in Tab. 3. We adopt their data splits, combining the NOTR *dynamic32* split and *static32* split. These splits are curated by [26] and are publicly available at `https://github.com/NVlabs/EmerNeRF/blob/main/docs/NOTR.md`.

Argoverse2: We use Argoverse2 Sensor Dataset [1] to showcase our ability to handle multiple datasets in Fig. 7. We process the Argoverse data using Omnire's open-source script at `https://github.com/ziyc/drivestudio/blob/main/datasets/argoverse/argoverse_preprocess.py`, and select the first 100 logs for testing and the rest for training.

C.2 Experiment Settings Details

Interpolation: We split the training sequences into 1-second snippets (11 frames), ensuring a 5-frame overlap between consecutive snippets. Frames at indices [0, 2, 4, 6, 8, 10] are used as input frames, while frames at indices [1, 3, 5, 7, 9] serve as target frames.

Future prediction: We split the training sequences into 1.5-second snippets (16 frames), ensuring a 5-frame overlap between consecutive snippets. Frames at indices [0, 2, 4, 6, 8, 10] are used as input frames, while frames at indices [1, 3, 5, 7, 9, 11, 12, 13, 14, 15] serve as target frames.

C.3 Training Details

We train both models for 30,000 steps for approximately 2 days on $4\times$ NVIDIA L40S GPUs. We use batch size 1 per GPU, achieving an effective batch size of 4. We use the Adam optimizer with a learning rate of 1e-3, with an exponential-decay annealing scheduler and a warmup phase of 1000 steps. We use the full-resolution camera images on all datasets: PandaSet (1920 $\times$ 1080), Waymo Open Dataset (1920 $\times$ 1280), Argoverse 2 (1550 $\times$ 2048). For the scaling analysis, all models are trained on PandaSet for 10,000 steps and on WOD for 30,000 steps.

D Additional Experiments

D.1 Scene Flow Evaluation

Table A1 and A2 evaluate the scene flow performance of FastNSF [7] and *Flux4D* on WOD. Although not designed for scene flow estimation, *Flux4D* achieves superior performance across all scene flow metrics. We leave comparisons to state-of-the-art dedicated scene flow methods (*e.g.*, EulerFlow [17]) in the future work. These findings unveil a promising path to unifying state-of-the-art flow and reconstruction within a single framework.

Table A1: **Comparison with scene flow estimation methods on WOD.**

Method	EPE3D ↓	Acc_5 ↑	Acc_{10} ↑	θ_ϵ ↓	EPE-BS ↓	EPE-FS ↓	EPE-FD ↓	EPE-3way ↓	Inference time ↓
FastNSF [7]	0.162	0.734	0.805	0.908	0.131	0.076	0.650	0.203	∼0.28 s/frame
Flux4D	**0.048**	**0.901**	**0.929**	**0.540**	**0.011**	**0.012**	**0.440**	**0.114**	**∼0.20 s/frame**

Table A2: **Bucketed scene flow error on WOD.** Normalized EPE3D (↓) per class, split into static (S) and dynamic (D) regions. Mean S/D are averages across all buckets. Abbrev.: BG = Background, VEH = Vehicle, PED = Pedestrian, CYC = Cyclist.

Method	BG-S↓	VEH-S↓	VEH-D↓	PED-S↓	PED-D↓	CYC-S↓	CYC-D↓	Mean S↓	Mean D↓
FastNSF [7]	0.144	0.043	0.653	0.044	1.026	**0.026**	0.744	0.064	0.807
Flux4D	**0.011**	**0.009**	**0.502**	**0.014**	**0.680**	**0.026**	**0.732**	**0.015**	**0.638**

D.2 Ablation Study

Ablation on *Flux4D* components: We conduct an ablation study on the key components of *Flux4D* in Fig. A2. The results clearly show that both iterative refinement and polynomial motion modeling enhance the sharpness and overall rendering quality.

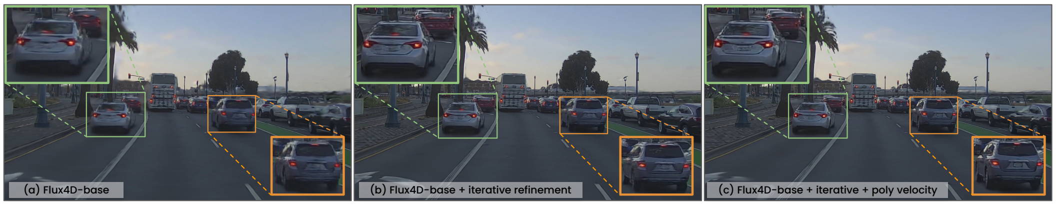
(a) Flux4D-base
(b) Flux4D-base + iterative refinement
(c) Flux4D-base + iterative + poly velocity

Figure A2: **Ablation study on *Flux4D* components.** Iterative refinement (2 iterations) and polynomial motion modeling (constant acceleration, $\ell = 1$) enhance fine-grained reconstruction details.

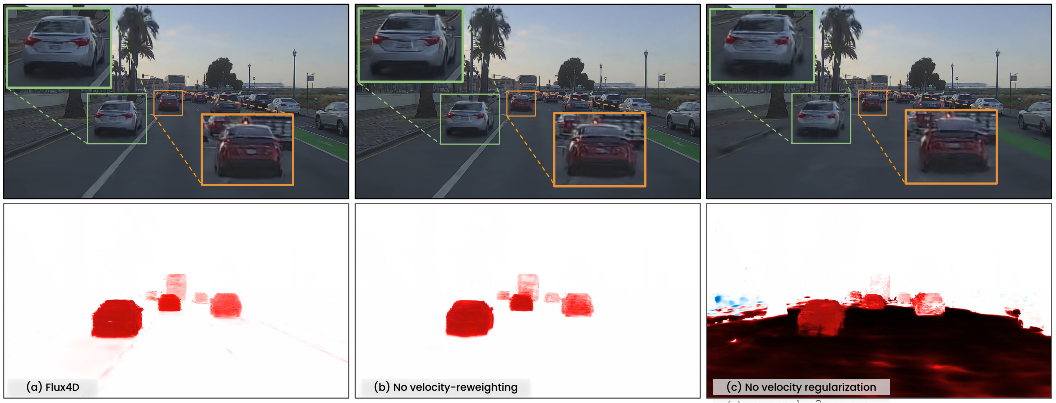

Figure A3: **Ablation study on training strategies.** Velocity re-weighting and static-preference regularization are crucial for capturing fine-grained details in dynamic actors and ensuring accurate, consistent motion dynamics.

Ablation on training strategies: We further conduct an ablation study on the training strategies of *Flux4D* in Fig. A3. Without velocity regularization, the model predicts false-positive velocities in static regions, leading to artifacts in the rendered frames. Without velocity re-weighting, the rendering quality degrades, resulting in blurry and inconsistent renderings.

D.3 Qualitative Comparison to DrivingRecon [8]

We confirm that our evaluation setup aligns with the NVS setting in [8] and sincerely thank the authors for in-depth discussions and for providing visual examples for comparison. We provide qualitative comparisons with DrivingRecon [8] in Fig. A4. Overall, our method achieves superior reconstruction quality, operating at a higher resolution while avoiding artifacts present in DrivingRecon. However, we acknowledge that DrivingRecon produces better background reconstruction (*e.g.*, sky, buildings), potentially due to its pixel-aligned Gaussian prediction. For simplicity, we currently sample points in a spherical image and use the same network for both foreground and background, which may lead to suboptimal results. We leave more advanced background modeling for future work (Sec. F).

D.4 Future Prediction with *Flux4D*

Flux4D achieves state-of-the-art performance on future prediction compared to other unsupervised reconstruction methods. We share additional qualitative results in Fig. A6. Additionally, Fig. A7 showcases *Flux4D*'s competitive qualitative results even when compared with methods that rely on labels for rendering extrapolation views.

D.5 Additional Qualitative Examples

Controllable simulation: We provide additional examples of realistic and controllable simulation with *Flux4D* in Fig. A8 and Fig. A9. Moving objects are clustered into instances using DBSCAN [5], which can be manipulated to create diverse and counterfactual scenarios. We additionally show *Flux4D*'s ability to handle various camera movement and actor manipulation.

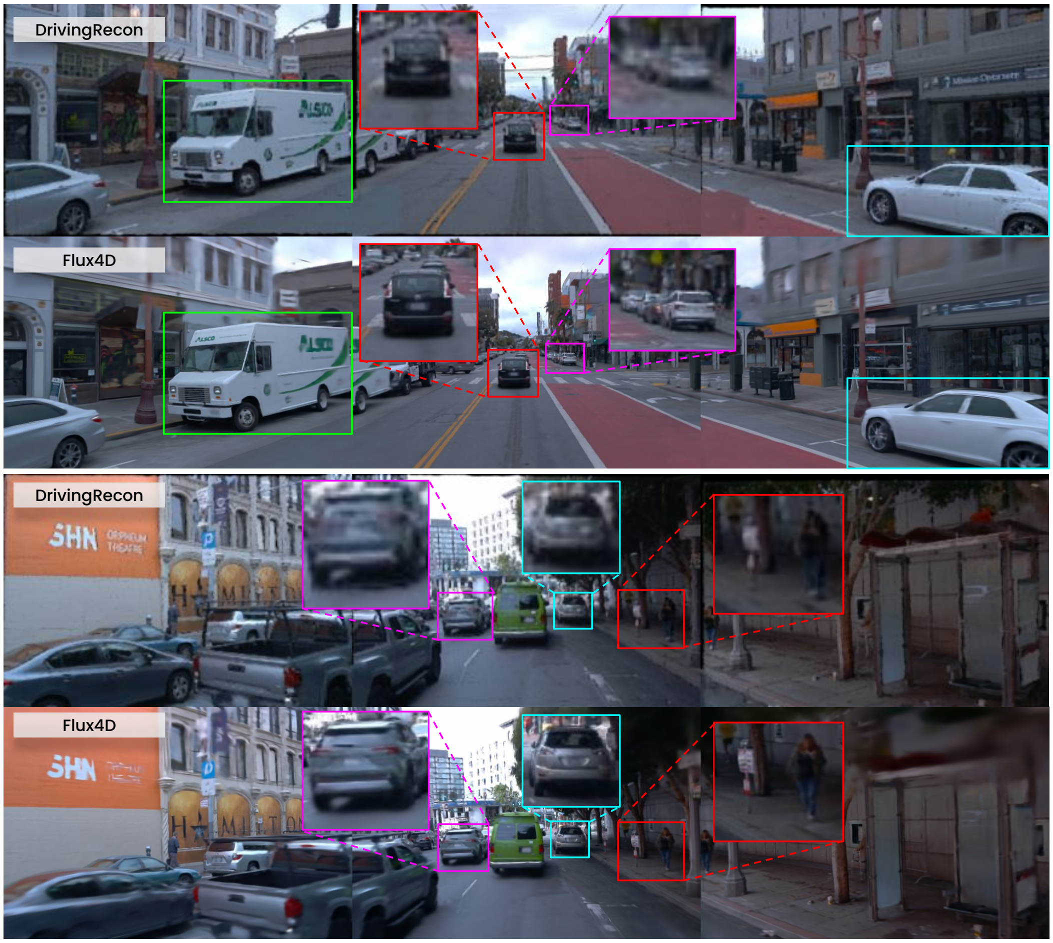

Figure A4: **Qualitative comparison with DrivingRecon [8] on WOD.**

Generalization to diverse scenes: As an unsupervised reconstruction method at heart, *Flux4D* excels at reconstructing diverse 4D urban driving scenes at scale without relying on any annotations or pretrained vision models. We showcase additional examples in Fig. A10 and Fig. A11.

E Discussions

We would like to highlight the main differences between our method and several related works. We hope this disccusion can help clarify the novelty and contirbutions of our *Flux4D*.

E.1 Differences to G3R

We bring the iterative refinement idea in G3R [2] to further improve the reconstruction realism of *Flux4D-base*. Our approach differs in several key aspects. We omit the neural Gaussian representation and the flatness regularization term. Thanks to our scalable unsupervised training and high-quality motion estimation, we achieve superior performance as indicated in Table 1 (interpolation setting), 2 (full sequence setting), and 4 (extrapolation setting) of the main paper. Moreover, our method only require $N = 3$ iteration compared to G3R's $N = 24$ iterations to achieve competitive rendering quality, resulting in significantly faster reconstruction than G3R (3.9s v.s. 17s, Table 1).

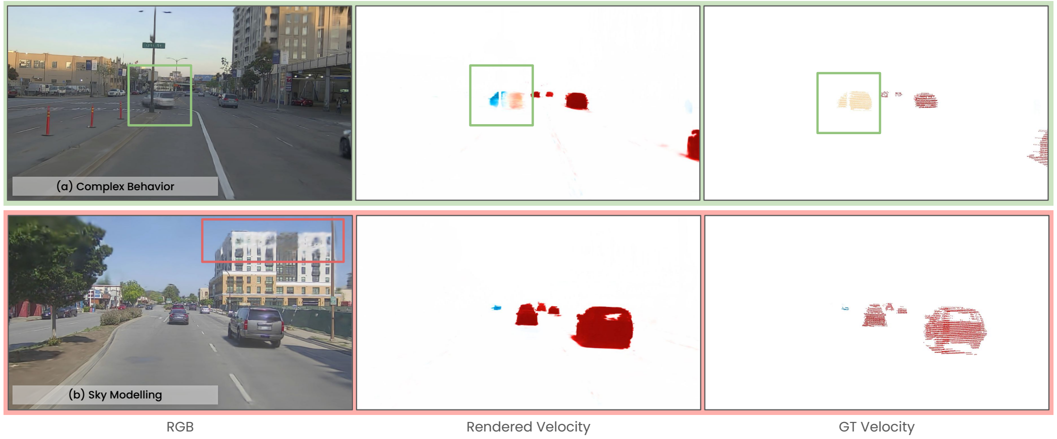

Figure A5: **Artifacts and limitations of *Flux4D*.** (a) *Flux4D* may struggle to accurately decompose the motion of actors exhibiting complex behaviors, such as abrupt left turns, particularly in occluded scenarios. Scaling to more diverse and larger-scale scenarios could help mitigate this issue. (b) Distant regions may appear blurry due to missing LiDAR points and our simplified sky modeling approach, which represents the sky using randomly placed 3D Gaussians on a single spherical plane. Incorporating more advanced sky modeling techniques [28] and denser point initialization (*e.g.*, structure from motion) could further improve performance.

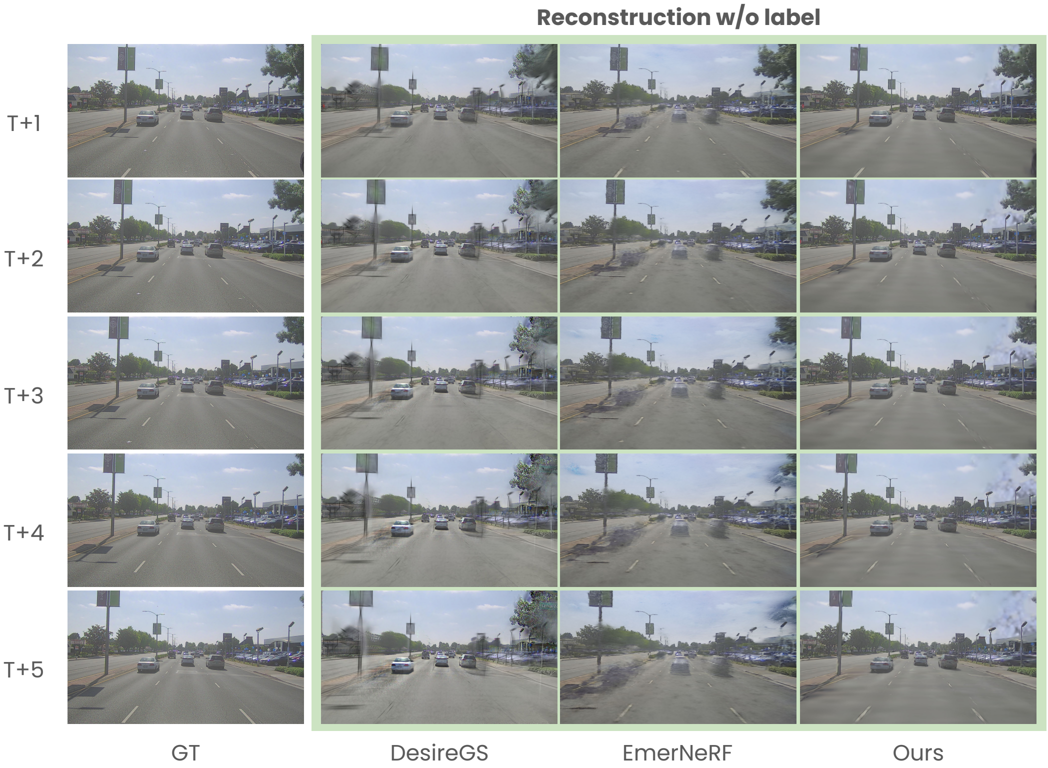

Figure A6: **Comparison against unsupervised reconstruction methods on future prediction.**

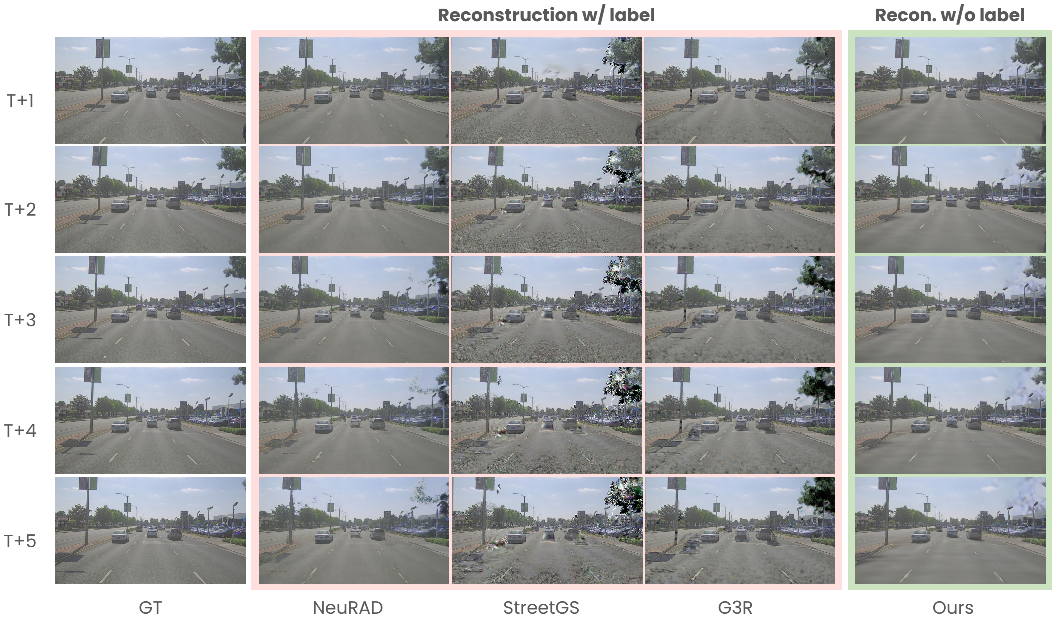

Figure A7: ***Flux4D* achieves competitive performance on future frames, even compared to methods that reconstruct with labels.** Unlike these methods, which rely on labels to render extrapolation views, *Flux4D* learns to predict future dynamics in an unsupervised manner, highlighting its potential for scene understanding and forecasting.

E.2 Comparison to DrivingRecon and STORM

Most recently, DrivingRecon [8] and STORM [25] explore similar problem of unsupervised generalizable 4D reconstruction for driving scenes, using feed-forward networks to predict the velocities of 3D Gaussians. We highlight three important distinctions as follows:

Model dependency: DrivingRecon and STORM rely on pre-trained vision models (*i.e.*, DeepLabv3+, SAM, ViT-Adapter). In contrast, Flux4D is fully unsupervised and does not depend on any pretrained vision models or foundational priors. Despite its simpler design, *Flux4D* achieves superior performance against DrivingRecon (Table 3 in the main paper).

Resolution and number of views: While DrivingRecon and STORM operate on low-resolution inputs (256 × 512 and 160 × 240 respectively) with limited views (3-4 frames), *Flux4D* supports high-resolution (1080x1920) inputs with 20+ frames, enabling significantly more detailed reconstructions.

Efficiency: *Flux4D* is more computationally efficient and scalable thanks to our minimalist design. In contrast to DrivingRecon and STORM, which require approximately 480 and 768 A100 GPU hours (confirmed by authors) respectively, *Flux4D* trains with only 140 L40S GPU hours.

We provide a detailed summary of the differences between *Flux4D* and these two methods in Table A3.

Table A3: **High-level comparison of *Flux4D* with DrivingRecon [8] and STORM [25].**

Method	**DrivingRecon**	**STORM**	***Flux4D***
Image resolution	256 × 512	160 × 240	$\geq$ 1080 × 1920
Training GPU hours	∼480 (A100 80GB)	∼768 (A100 80GB)	∼140 (L40S 48GB)
Support LiDAR input	No	No	Yes
Usage of Pre-trained vision model	Yes	Yes	No
Number of frames (3 cameras)	3	4	20+

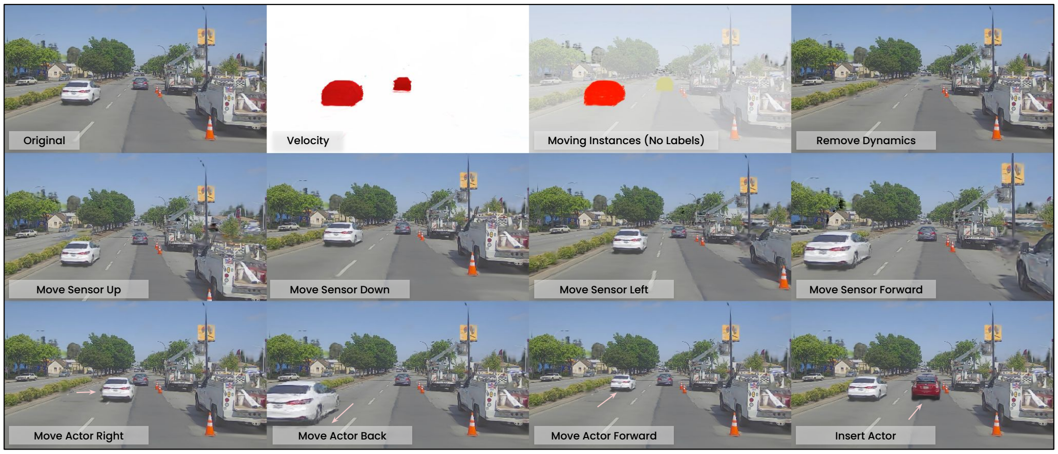

Figure A8: **Realistic and controllable simulation with *Flux4D* (Part 1/2).** Our approach decomposes 4D dynamic scenes with accurate motion flows, enabling precise control over both sensor viewpoints and dynamic actor placements. We demonstrate various scene modifications, including novel sensor placements, actor removal, insertion and manipulation. Importantly, we achieve this without reliance on any labels or annotations.

F Limitations

F.1 Artifacts and Potential Enhancements

Complex behaviors: In Fig. A5 (a), *Flux4D* fails to accurately predict the flow of the left-turning vehicle due to occlusions from the pole. *Flux4D* arrives at the nonrigid solution for the vehicle by predicting 2 clusters of opposite flows. Potential enhancements include enforcing rigidity for moving instances or augmenting *Flux4D* with memory.

Faraway regions: *Flux4D* initializes from LiDAR point cloud. However, due to the sparsity of LiDAR data, some faraway regions such as trees and sky are often not covered as shown in Fig. A5 (b). Potential enhancements include learning multi-layer high-resolution skyboxes or panorama images or adding density control strategy.

F.2 Limitations and Future Work

Complex motion pattern: Despite *Flux4D*'s significant performance improvements over prior work, accurately reconstructing flow for highly dynamic actors or those with complex motion patterns remains challenging. Scaling to larger, more diverse datasets and adopting more advanced velocity models or implicit flow representations could help mitigate this limitation.

Requirement of poses: Besides, *Flux4D* requires posed cameras and LiDAR initialization, limiting its applicability to scenarios with unknown camera parameters. Integrating pose-free methods like DUSt3R [18] would eliminate the dependency on camera poses, enabling reconstruction from casual video captures and increasing Flux4D's versatility to uncalibrated or partially calibrated sensor setups.

Overall, we hope that our simple and scalable design serves as a foundation for the community to build upon, enabling further advancements in 4D reconstruction.

G Broader Impact

Flux4D is an unsupervised 4D reconstruction method that can be applied to various applications, including autonomous driving, robotics, and augmented reality. Especially for autonomous driving, its ability to accurately model the highly dynamic nature of the urban environemnt, coupling with its unsupervised nature, allow it to be a powerful pretraining method for various downstream tasks such as motion forecasting, object labelings, and simulator training. Additionally, with more expressive

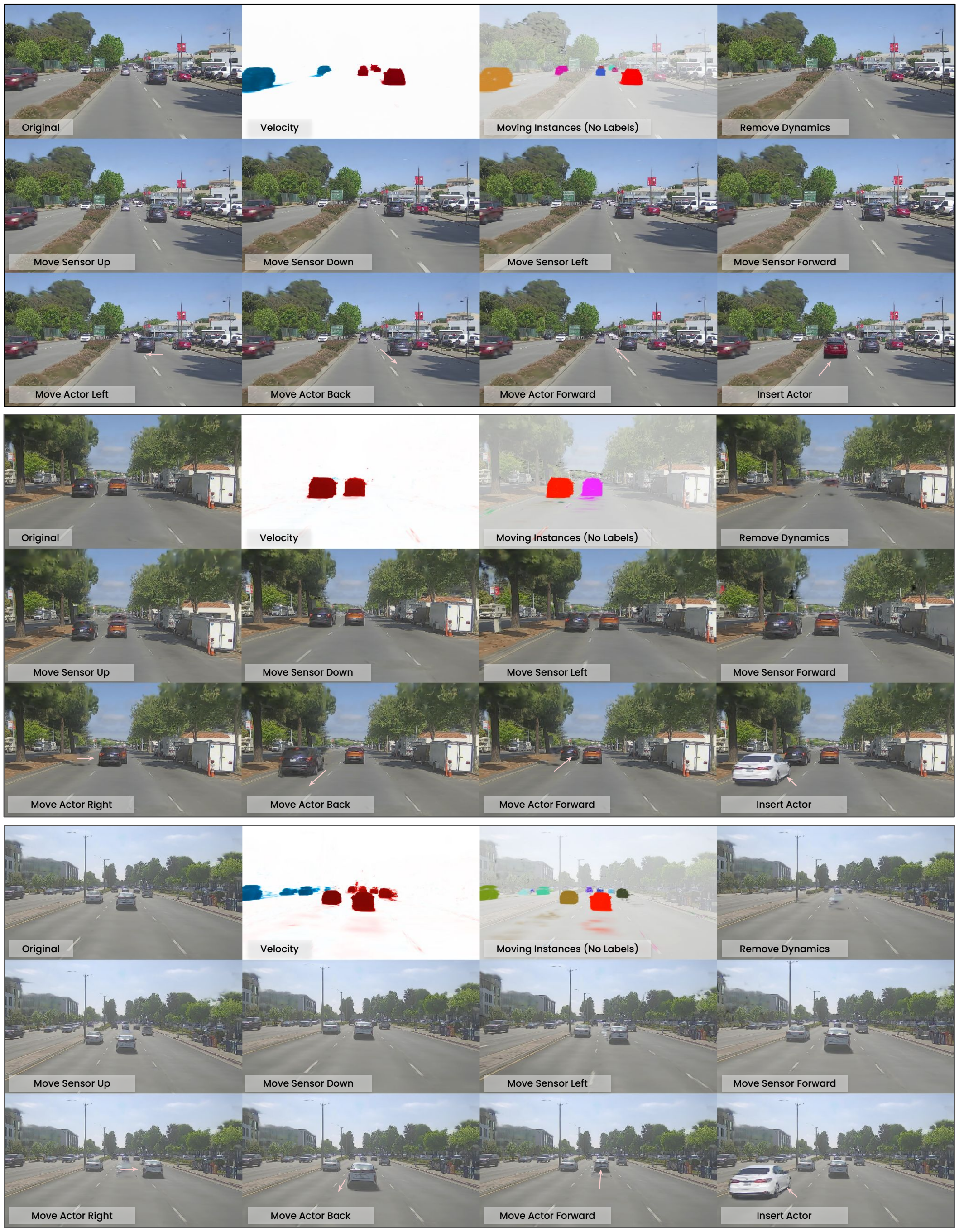

Figure A9: **Realistic and controllable simulation with *Flux4D* (Part 2/2).**

motion modeling, *Flux4D* could be useful to other domain such as augmented reality or robotics, where accurate 4D reconstruction is crucial for scene understanding and interaction. Creating digital twins of real-world locations may raise privacy concerns. Additionally, our system may exhibit unstable performance or unintended behavior on different datasets, particularly when sensory data are sparse or noisy.

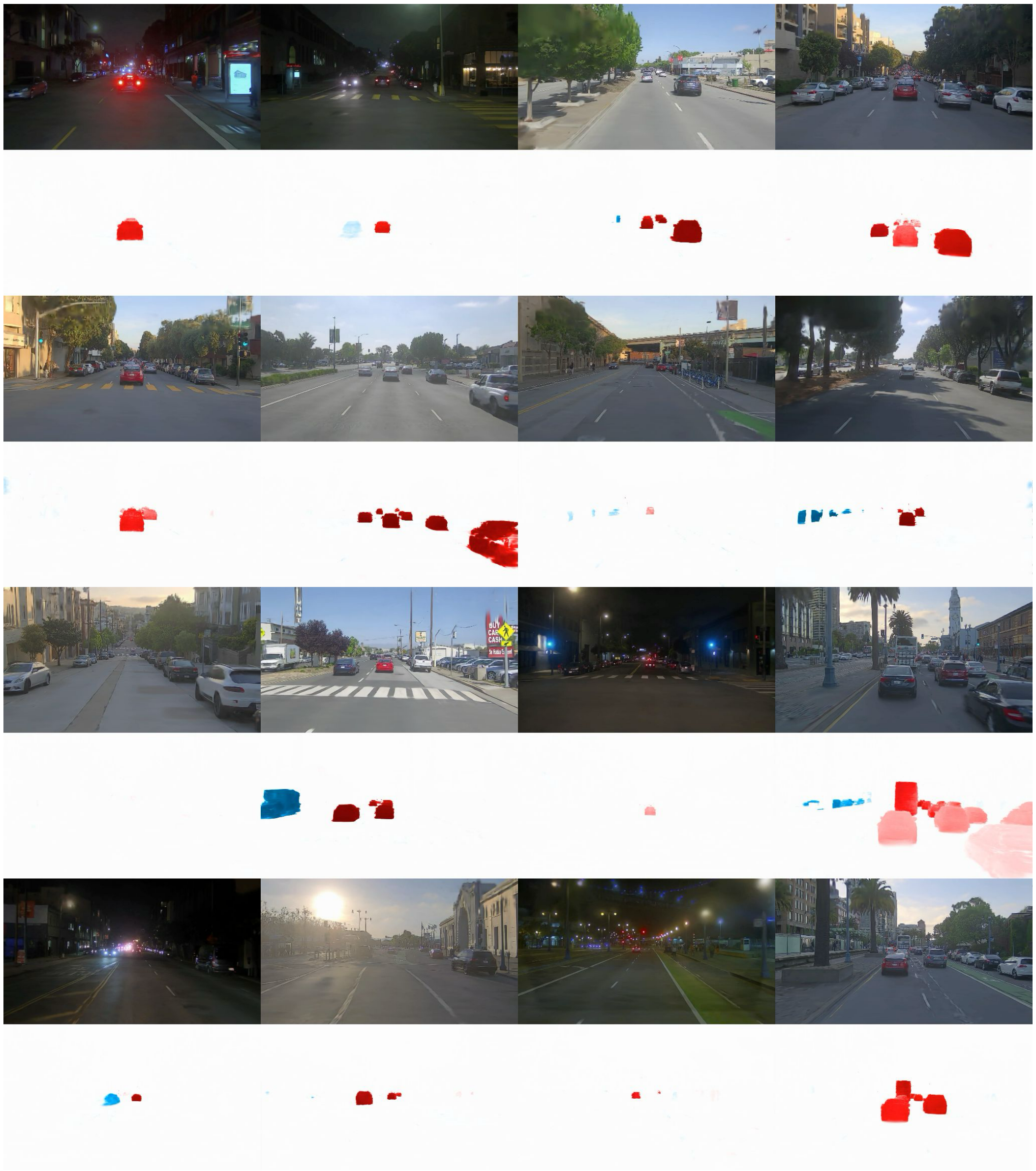

Figure A10: **Reconstructing diverse 4D urban driving scenes at scale with *Flux4D* (Part 1/2).**

Experiment	**L40S Hours**	**Comments**
Table 1 & 2 (Interpolation & Full Seq)	∼ 400	We use the same *Flux4D* model for Tab.1 and 2
Table 3 (DrivingRecon)	∼ 150	No baseline training needed
Table 4 (Future Prediction)	∼ 300	No training for Depthsplat, L4GM
Table 5 & 6 (Ablation)	∼ 900	This requires 5 more *Flux4D*'s training jobs
Fig. 7 (Argoverse2 & WOD)	∼ 300	WOD: 150hr, Argoverse2: 150hr
Fig. 8 (Scaling Analysis)	∼ 1200	Pandaset: 750hr, WOD: 450hr
Others (data generation & demos)	∼ 10	No data generation needed, estimated 10hr for all demos

Table A4: Summary of GPU hours used for the final experiments.

H Computation Resources

We provide the estimated GPU usage for the final experiments in Table A4, where all hours are converted to 1xL40S hours.

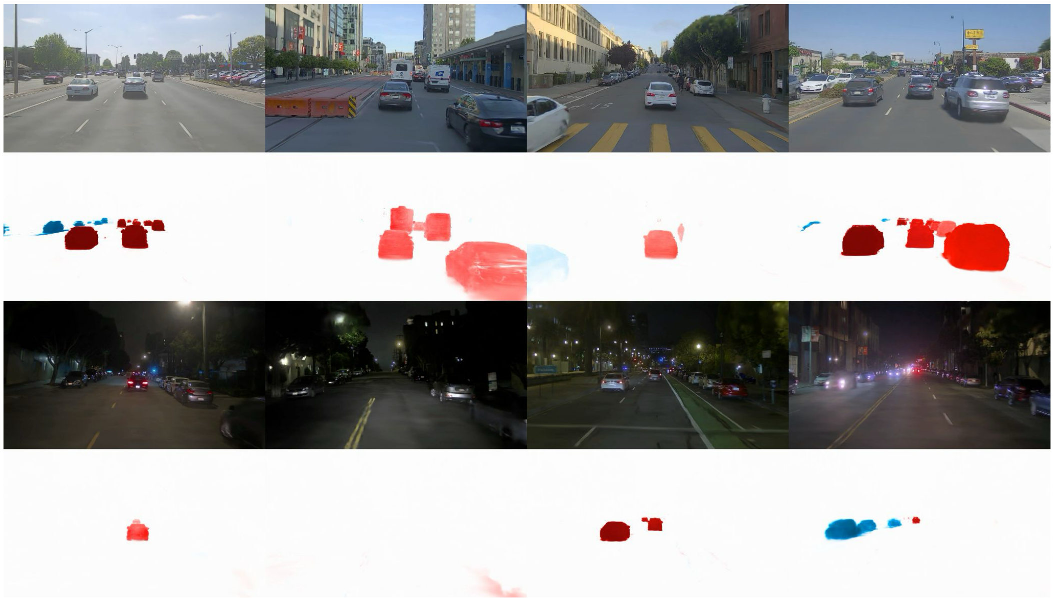

Figure A11: **Reconstructing diverse 4D urban driving scenes at scale with *Flux4D* (Part 2/2).**

I Licenses of Assets

We summarize the licenses and terms of use for all assets (datasets, software, code) in Table A5. We use the Waymo Open Dataset (WOD) only for research purposes and solely for final benchmarking, in accordance with its license requirements.

Assets	**License**	**URL**
Datasets		
PandaSet [21]	CC BY 4.0	https://scale.com/open-av-datasets/pandaset
WOD [14]	Custom**	https://waymo.com/open/
ArgoVerse [20]	CC BY-NC-SA 4.0	https://www.argoverse.org/av2.html
Codebases		
PyTorch [10]	Custom††	https://github.com/pytorch/pytorch
Torchsparse [15]	MIT	https://github.com/mit-han-lab/torchsparse
gsplat [30]	Apache-2.0	https://github.com/nerfstudio-project/gsplat
Omnire [4]	MIT	https://github.com/ziyc/drivestudio
Baseline Codebases (Comparisons purposes only)		
NeuRAD [16]	Apache-2.0	https://github.com/georghess/neurad-studio
Depthsplat [23]	MIT	https://github.com/cvg/depthsplat
EmerNeRF [26]	Custom‡‡	https://github.com/NVlabs/EmerNeRF
L4GM [13]	Apache-2.0	https://github.com/nv-tlabs/L4GM-official
DeSiRe-GS [11]	N/A	https://github.com/chengweialan/desire-gs

Table A5: Summary of the licenses of assets.

** https://waymo.com/open/terms

†† https://github.com/pytorch/pytorch/blob/main/LICENSE

‡‡ https://github.com/NVlabs/EmerNeRF/blob/main/LICENSE

References

[1] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. *CVPR*, 2019. 6

[2] Yun Chen, Jingkang Wang, Ze Yang, Sivabalan Manivasagam, and Raquel Urtasun. G3R: Gradient guided generalizable reconstruction. In *ECCV*, 2025. 5, 8

[3] Yurui Chen, Chun Gu, Junzhe Jiang, Xiatian Zhu, and Li Zhang. Periodic vibration gaussian: Dynamic urban scene reconstruction and real-time rendering. *arXiv preprint arXiv:2311.18561*, 2023. 5

[4] Ziyu Chen, Jiawei Yang, Jiahui Huang, Riccardo de Lutio, Janick Martinez Esturo, Boris Ivanovic, Or Litany, Zan Gojcic, Sanja Fidler, Marco Pavone, et al. Omnire: Omni urban scene reconstruction. *arXiv preprint arXiv:2408.16760*, 2024. 2, 4, 14

[5] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996. 7

[6] Jing He, Haodong Li, Wei Yin, Yixun Liang, Leheng Li, Kaiqiang Zhou, Hongbo Zhang, Bingbing Liu, and Ying-Cong Chen. Lotus: Diffusion-based visual foundation model for high-quality dense prediction. *arXiv preprint arXiv:2409.18124*, 2024.

[7] Xueqian Li, Jianqiao Zheng, Francesco Ferroni, Jhony Kaesemodel Pontes, and Simon Lucey. Fast neural scene flow. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9878–9890, 2023. 6

[8] Hao Lu, Tianshuo Xu, Wenzhao Zheng, Yunpeng Zhang, Wei Zhan, Dalong Du, Masayoshi Tomizuka, Kurt Keutzer, and Yingcong Chen. Drivingrecon: Large 4d gaussian reconstruction model for autonomous driving. *arXiv preprint arXiv:2412.09043*, 2024. 6, 7, 8, 10

[9] Seungtae Nam, Xiangyu Sun, Gyeongjin Kang, Younggeun Lee, Seungjun Oh, and Eunbyung Park. Generative densification: Learning to densify gaussians for high-fidelity generalizable 3d reconstruction. *arXiv preprint arXiv:2412.06234*, 2024.

[10] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 2, 14

[11] Chensheng Peng, Chengwei Zhang, Yixiao Wang, Chenfeng Xu, Yichen Xie, Wenzhao Zheng, Kurt Keutzer, Masayoshi Tomizuka, and Wei Zhan. Desire-gs: 4d street gaussians for static-dynamic decomposition and surface reconstruction for urban driving scenes. *arXiv preprint arXiv:2411.11921*, 2024. 5, 14

[12] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3), 2022.

[13] Jiawei Ren, Cheng Xie, Ashkan Mirzaei, Karsten Kreis, Ziwei Liu, Antonio Torralba, Sanja Fidler, Seung Wook Kim, Huan Ling, et al. L4gm: Large 4d gaussian reconstruction model. In *NeurIPS*, 2025. 5, 14

[14] 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 *CVPR*, 2020. 6, 14

[15] Haotian Tang, Shang Yang, Zhijian Liu, Ke Hong, Zhongming Yu, Xiuyu Li, Guohao Dai, Yu Wang, and Song Han. Torchsparse++: Efficient training and inference framework for sparse convolution on gpus. In *MICRO*, 2023. 2, 14

[16] Adam Tonderski, Carl Lindström, Georg Hess, William Ljungbergh, Lennart Svensson, and Christoffer Petersson. NeuRAD: Neural rendering for autonomous driving. In *CVPR*, 2024. 3, 4, 5, 14

[17] Kyle Vedder, Neehar Peri, Ishan Khatri, Siyi Li, Eric Eaton, Mehmet Kemal Kocamaz, Yue Wang, Zhiding Yu, Deva Ramanan, and Joachim Pehserl. Neural eulerian scene flow fields. In *ICLR*, 2025. 6

[18] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. Dust3r: Geometric 3d vision made easy. In *CVPR*, 2024. 11

[19] Xinyue Wei, Kai Zhang, Sai Bi, Hao Tan, Fujun Luan, Valentin Deschaintre, Kalyan Sunkavalli, Hao Su, and Zexiang Xu. Meshlrm: Large reconstruction model for high-quality meshes. *arXiv preprint arXiv:2404.12385*, 2024. 2

[20] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, et al. Argoverse 2: Next generation datasets for self-driving perception and forecasting. *arXiv preprint arXiv:2301.00493*, 2023. 14

[21] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, et al. Pandaset: Advanced sensor suite dataset for autonomous driving. In *ITSC*, 2021. 5, 14

[22] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *NeurIPS*, 34:12077–12090, 2021. 5

[23] Haofei Xu, Songyou Peng, Fangjinhua Wang, Hermann Blum, Daniel Barath, Andreas Geiger, and Marc Pollefeys. Depthsplat: Connecting gaussian splatting and depth. *arXiv preprint arXiv:2410.13862*, 2024. 5, 14

[24] Yunzhi Yan, Haotong Lin, Chenxu Zhou, Weijie Wang, Haiyang Sun, Kun Zhan, Xianpeng Lang, Xiaowei Zhou, and Sida Peng. Street gaussians for modeling dynamic urban scenes. In *ECCV*, 2024. 4

[25] Jiawei Yang, Jiahui Huang, Yuxiao Chen, Yan Wang, Boyi Li, Yurong You, Maximilian Igl, Apoorva Sharma, Peter Karkus, Danfei Xu, Boris Ivanovic, Yue Wang, and Marco Pavone. Storm: Spatio-temporal reconstruction model for large-scale outdoor scenes. *arXiv preprint arXiv:2501.00602*, 2025. 5, 10

[26] Jiawei Yang, Boris Ivanovic, Or Litany, Xinshuo Weng, Seung Wook Kim, Boyi Li, Tong Che, Danfei Xu, Sanja Fidler, Marco Pavone, and Yue Wang. Emernerf: Emergent spatial-temporal scene decomposition via self-supervision. *arXiv preprint arXiv:2311.02077*, 2023. 4, 6, 14

[27] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. *arXiv:2406.09414*, 2024.

[28] Ze Yang, Yun Chen, Jingkang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator. In *CVPR*, 2023. 4, 9

[29] Ze Yang, Yun Chen, Jingkang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator. In *CVPR*, 2023. 5

[30] Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, and Angjoo Kanazawa. gsplat: An open-source library for Gaussian splatting. *arXiv preprint arXiv:2409.06765*, 2024. 2, 14