NVIDIA.

Scaling Neural Reconstruction for Closed-Loop Autonomous Vehicle Simulation

Best Practices

August 2026

NVIDIA.

Table of Contents

August 2026

Note: This article is a co-authored best-practices report from the NVIDIA Omniverse NuRec team and Li Auto. It describes practices and product behavior as of the time of writing and is not the canonical reference for configuration or API details. For current parameter names and defaults, see Configuration. For current gRPC messages and methods, see Use the NuRec gRPC API.

Introduction

The neural reconstruction-based closed-loop simulation systems are becoming important for modern autonomous driving development. By transforming vehicle driving logs into high-fidelity 3D environments, these systems provide a safe and controlled environment to cover long-tail scenarios, a robust framework for algorithm evaluation and training, thereby accelerating development cycles and providing users with a safer autonomous driving experience.

NVIDIA Omniverse NuRec is NVIDIA's neural scene reconstruction platform for autonomous vehicle development. It converts real-world driving logs into photorealistic, simulation-ready 3D Gaussian Splat scenes for high-fidelity closed-loop simulation on real-world data.

NuRec-based simulation system handles 4,000,000 daily clip evaluations, 4,000 daily scene reconstructions, and 200,000 safety-critical assessments per week within NVIDIA. Furthermore, selected NuRec capabilities have been integrated into Li Auto's closed-loop simulation framework to support its continued development and optimization. As a global leader in the autonomous vehicle industry, Li Auto has successfully achieved mass-production scale with its simulation platform consisting of millions clips library, executing hundreds of thousands of clip evaluation and reinforcement learning training on a daily basis. (All driving logs and sensor data used for simulation of Li Auto's system are strictly anonymized, processed locally on Li Auto's infrastructure, and in full compliance with local data privacy and security regulations)

This article details the foundational best practices for scaling neural reconstruction-based simulation architectures. Our discussion is structured around two core components: the neural reconstruction engine itself, and the broader simulation system that leverages it.

Neural Reconstruction

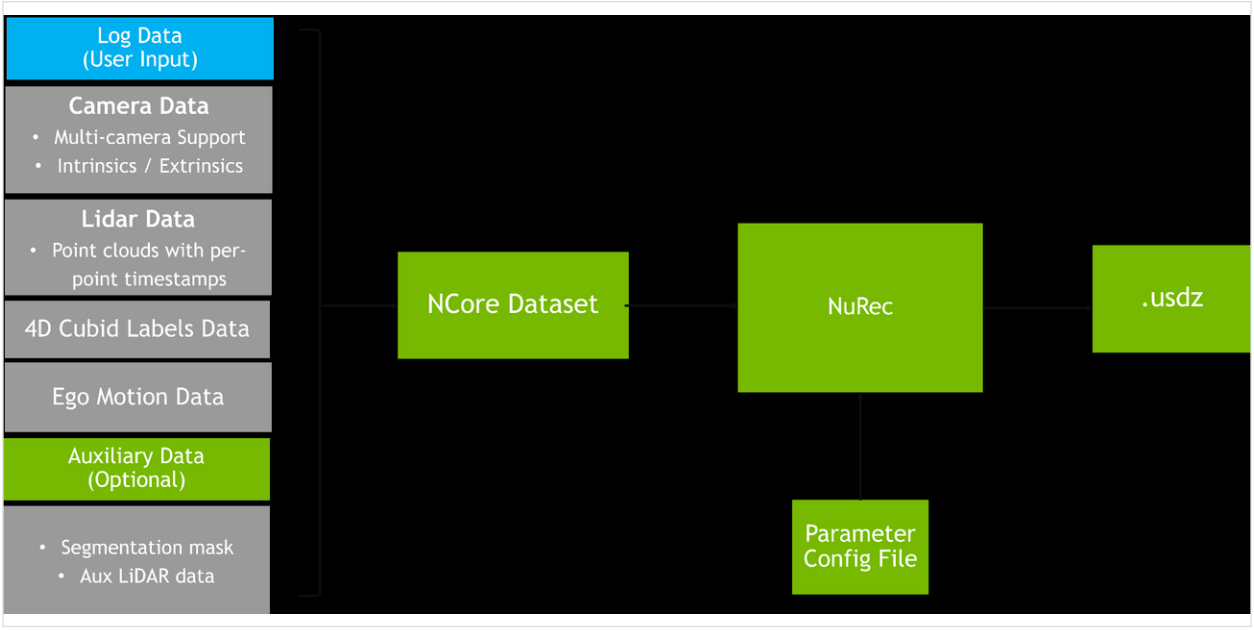
Within NVIDIA Omniverse NuRec, scenes are generally structured into four distinct Gaussian layers:

- `background` layer: represents the static environment
- `dynamic_rigid` layer: accommodates moving rigid objects such as trucks and vehicles
- `dynamic_deformables` layer: accounts for pedestrians
- `road` layer: handles the road surface

Furthermore, a separate sky layer is incorporated; rather than being represented by Gaussians, it utilizes a cubic environment map. The categorization of these layers relies on cuboid track labels and per-point semantic labels.

NVIDIA Omniverse NuRec provides a full-stack neural reconstruction solution for autonomous driving scenario reconstruction and sensor rendering.

Figure 1: Overview of the NuRec pipeline



The workflow diagram illustrates how NuRec operates, beginning with the conversion of raw driving log data into NCore format. NuRec then consumes this NCore dataset to generate reconstructed 3D scenes in USDZ format.

Li Auto's neural reconstruction workflow consumes 7-camera frames, LiDAR point clouds, and odometry as inputs. Following labeling and preparatory phases—including scene decoupling, pose refinement, and Gaussian initialization—the system trains a cohesive GS model for

every unique scene.

Prior to the reconstruction phase, scenes also undergo decoupling into distinct layers—including vehicles, pedestrians, traffic signals, road surfaces, and backgrounds—through labeling and auxiliary data preparation. The core methodology employs LiAutoScaffoldGS, a customized version of ScaffoldGS and 3DGUT designed to support unified vision and LiDAR modalities within a single GS representation.

Scaling up neural reconstruction for mass production-level simulation presents various challenges across functionality, efficiency, and evaluation standards. The following sections explore these challenges in detail.

Functionality and Quality

This section explores the essential features and core methodologies required to achieve superior reconstruction quality when utilizing autonomous driving data.

Handling Noisy Input Data

Achieving a high-quality final reconstruction generally relies on the precision of the raw data. It is essential to validate data accuracy prior to starting the reconstruction process and address any identified issues as thoroughly as possible.

NVIDIA Omniverse NuRec features a data sanity check steps notebook at https://nvidia.github.io/ncore/tutorial/data_sanity_check.html to visually inspect the quality of data saved in the NCore format. Additionally, a variety of tools are accessible at https://github.com/NVIDIA/ncore/tree/main/tools to help you examine the NCore data:

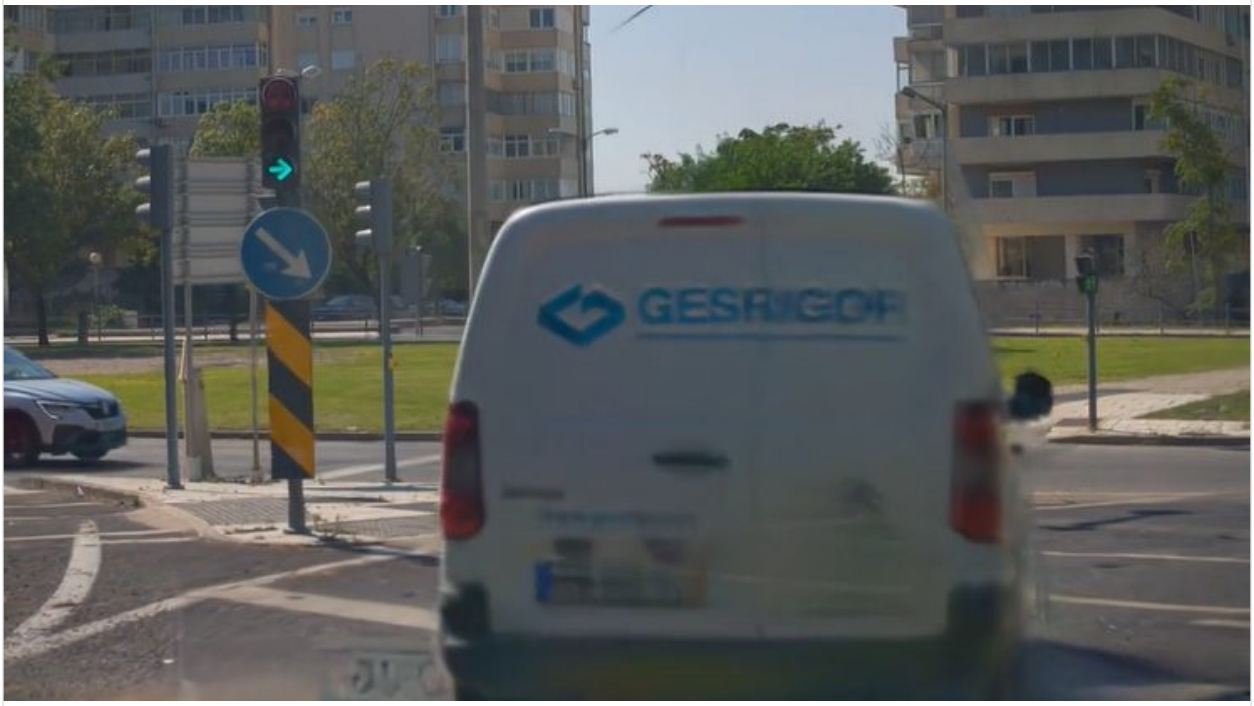
- `ncore-vis`: NCore Interactive 3D Viewer
- `ncore_export_ply`: export point clouds to world coordinate system with motion compensation. Help you check the poses and LiDAR point clouds quality. And usually, we will remove the LiDAR points within the ego vehicle areas to avoid floaters in the training and invalid semantic labels in the auxiliary data generation phase.
- `ncore_project_pc_to_img`: Help you check the camera and LiDAR alignment (calibration & timestamp)

However, the data in real production is not always perfect: there could be jittering in ego poses data and misalignment calibration parameters of sensors in a specific session. Additionally, track labels are not always stable and accurate in every frame — especially if the labels are from an automatic labelling system. Some of them can be examined following the sanity check steps, but it is not easy to fix them all manually.

To address this challenge, NVIDIA Omniverse NuRec provides online calibration for sensor poses and track poses. By enabling the online calibration, sensor poses and track poses will also be optimized during the training process. These features are enabled by default in `configs/apps/prod/Hyperion-8.1/car2sim.yaml`. Default sensor poses calibration will optimize only camera poses with initial `lr= 0.0001`. It is worth enabling LiDAR pose calibration and tune the learning rate for specific cases. Default track calibration is applied to `dynamic_rigids` and `dynamic_deformables`. Usually, the track calibration will not introduce quality regression. However, if you observe significant tracks jittering in the reconstruction, consider tuning the hyperparameters or disabling it.

Figure 2a: The reconstruction without online calibration

Figure 2b: The reconstruction with online calibration



In production at Li Auto, Li Auto applies several data refining steps to improve the data quality:

- **Annotation refinement:** filter the frames with low quality 4D cuboid labels.
- **Pose optimization:** Collected vehicle data often exhibits ego-pose inaccuracies due to GNSS/IMU drift. Li Auto addresses this by utilizing odometry constraints to refine ego-poses and employing a rigid body alignment (rig bundle adjustment) formulation for camera extrinsic optimization. This approach significantly minimizes pose errors and mitigates "ghosting" artifacts resulting from misaligned multi-frame observations.
- **Online track refinement:** Introduce learnable frame-specific track position offsets to mitigate discrepancies stemming from annotation and multimodal data integration, effectively enhancing the clarity of foreground reconstruction.

Road Constraint

High-quality road reconstruction is vital for synthesizing novel views with lateral trajectory shifts. The road surface possesses distinct geometric and photometric characteristics compared to the general background, featuring a nearly planar structure and strong normal consistency.

NVIDIA Omniverse NuRec provides several optimizations to enhance road reconstruction quality in both original view and novel view. We introduce geometry prior knowledge in the

Gaussians' initialization and supervision. By default, the following techniques are enabled in `configs/apps/prod/Hyperion-8.1/car2sim.yaml`:

- **Ground mesh initialization for the road:** We generate a road ground mesh from the LiDAR points and use the vertices of the mesh to initialize the Gaussians in the road layer. Note that the ground mesh will be placed according to the nominal ground point which is by default the ego/rig coordinate system's origin point — so if your ego/rig coordinate system's origin point is not on the ground (e.g., it could be at the center of ego vehicle's rear axle), change the `model.layers.road.initialization.z_offset` to the ground height in rig frame accordingly.
- **Road loss:** We enable road distortion loss to constrain the height and rotation variance of gaussians in the road layer and z-scale loss to penalize z-scale values above a given threshold for the road layer. We also deploy the semantic-based Gaussians identification in the training, so that we can avoid the Gaussians not in the road layer to represent the road or Gaussians that do not represent the road within the road layer from being incorrectly penalized by the road loss.

Figure 3a: The novel rendering view (3 meters left translation) in the reconstruction without road techniques

Figure 3b: The same view as Figure 3a in the reconstruction with road techniques

In Li Auto's solution, Li Auto also created RoadLoss — a custom loss term built upon LiAutoScaffoldGS that leverages ground attributes (planar priors and surface normal regularization) to provide auxiliary normal supervision. This methodology enhances reconstruction quality across both original and novel perspectives, effectively mitigating "floater" artifacts and tiling-style inconsistencies typically encountered with vanilla GS on flat road surfaces.

Fisheye Camera Reconstruction and Rendering

Vanilla 3DGS implementations typically assume a pinhole camera model. Applying this to fisheye cameras—essential for surround-view and automated parking coverage—presents a difficult trade-off: either utilize distorted RGB data as ground truth, which introduces significant radial inconsistencies, or perform undistortion, which discards substantial pixel data, particularly at the boundary area of images. By natively incorporating the fisheye distortion model directly into the reconstruction workflow, the system significantly expands the volume of pixels available for RGB supervision. This approach markedly enhances reconstruction fidelity for surround-view and parking sensors without compromising field-of-view coverage.

NVIDIA Omniverse NuRec provides the functionality to train and render with opencv conventional fisheye camera model (Kannala-Brandt distortion model). NuRec deploys the

NVIDIA's 3DGRUT under the hood — unlike conventional 3D Gaussian Splatting, which will require undistorted images in the rendering, 3DGRUT natively supports images with distortion and motion effects like rolling-shutter. To use this feature, store the images with distortion and corresponding camera model parameters in the NCore dataset: https://nvidia.github.io/ncore/data/sensor_models.html#opencv-fisheye-camera-model. For fisheye camera model, notice that when you don't have `max_angle` from the calibration tool, you could use the API provided by NCore: https://nvidia.github.io/ncore/apis/data.html#ncore.data.OpenCVFisheyeCameraModelParameters.compute_max_angle

Dynamic Traffic Signal (Traffic Light, Vehicle's Dynamic Signals)

In autonomous driving contexts, the state of traffic lights and lead-vehicle tail lights (braking or signaling) serves as a critical input influencing ego-vehicle trajectory planning. While these signals change temporally, standard GS models typically assume static Gaussian appearance, often resulting in baked-in single states or blurred transitions.

NVIDIA Omniverse NuRec provides the temporal appearance to model the dynamic appearance (color) changing in the real video, such as traffic light changing (even numbers counting down), vehicles' braking light, turning signal and emergency light. In `configs/apps/prod/Hyperion-8.1/car2sim.yaml`, we enable temporal appearance for vehicles by default. For traffic light, we provide two different usages:

- If you don't have traffic light cuboid labels, NVIDIA Omniverse NuRec provides the functionality to infer the traffic cuboids from the point cloud with semantic labels. The point cloud could be the LiDAR point cloud in NCore. And we will apply semantic labels to the points through the process of auxiliary data preparation: Generate Auxiliary Data. To enable this feature, Add the following section in your config:

```
defaults:
  # Traffic light layer with step-function time embedding for abrupt state changes
  - /model/gaussians/models@model.layers.traffic_light: rigid_gaussians
  - /model/gaussians/extra_signal@model.layers.traffic_light.extra_signal: [semantic_logits]
  - /model/gaussians/initialization@model.layers.traffic_light.initialization: camera_dynamic_tracks
  - /model/input_embedding@model.layers.traffic_light.time_embed: individual_step_time_gaussian
  - /model/tracks_calib@model.layers.traffic_light.tracks_calib: skip
dataset:
  generate_static_rigid_cuboid_tracks:
    enabled: true
    visibility_check_camera_ids: ${dataset.camera_ids}
    rigid_classes:
      - "traffic light"
model:
  layers:
    background:
      ignore_classes_from_layers: ["road", "traffic_light"]
    # Traffic light layer with step-function time embedding for abrupt color transitions
    traffic_light:
```

```
      class_labels: ["traffic light"]
      is_static: true
      fourier_features_dim: 5
      tracks:
        label_classes:
          ["generated traffic light"]
      initialization:
        keep_all_track_poses: true
        symmetric_axis: null
```

- If you already have traffic light cuboid labels stored in NCore dataset, you could just change above config like:

```
model:
  layers:
    traffic_light:
      tracks:
        label_classes:
          ["<Your own traffic light label>"]
```

In the config, `fourier_features_dim` is to control the capability of the temporal modeling. We set it to 5 by default, but if you want to capture the changing in high frequency, it is worth experimenting to increase this value.

Figure 4a: The turning signals of the vehicle within the reconstruction

Figure 4b: The traffic light changing in the reconstruction

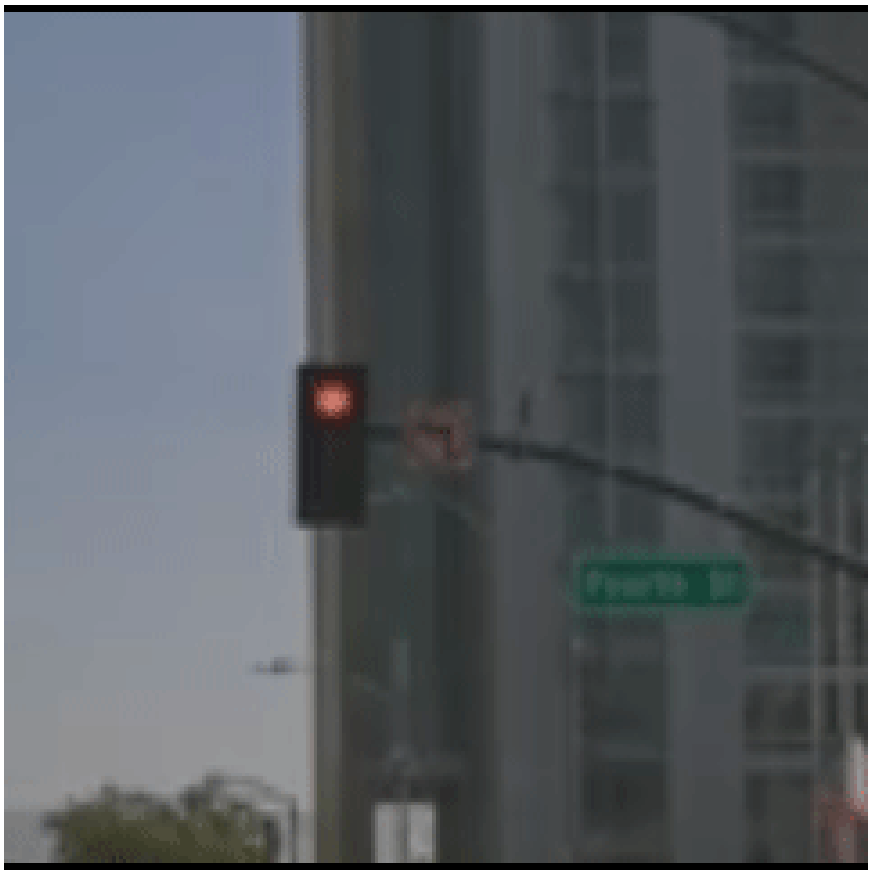
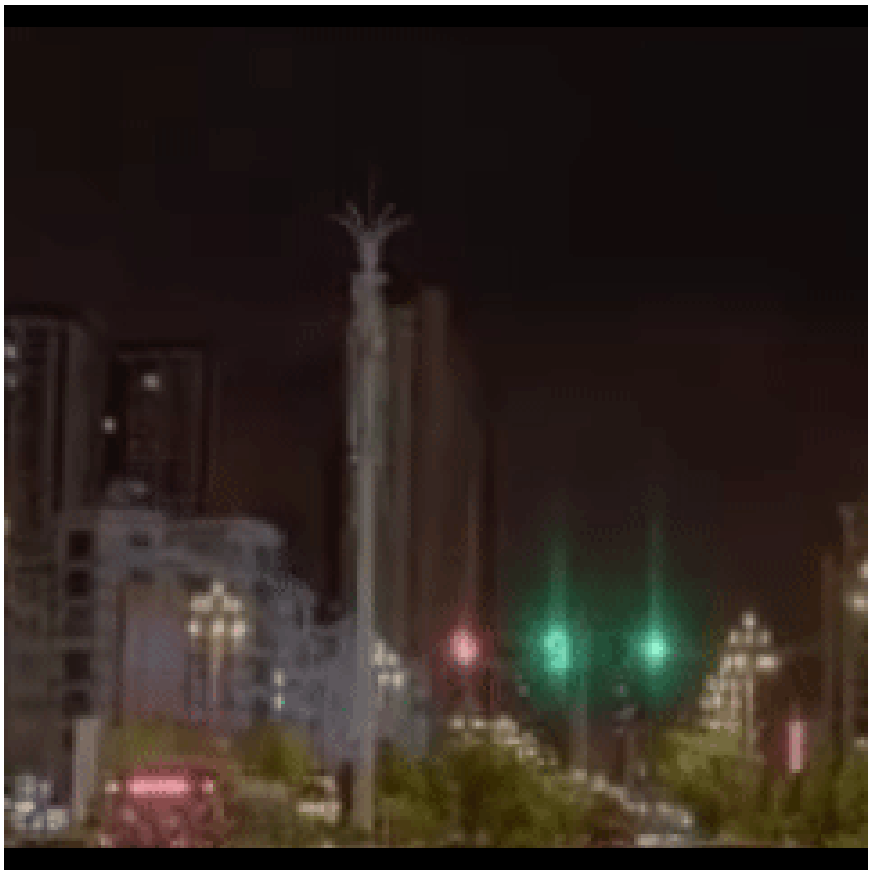


Figure 4c: Reconstructed dynamic rendering of a countdown traffic timer



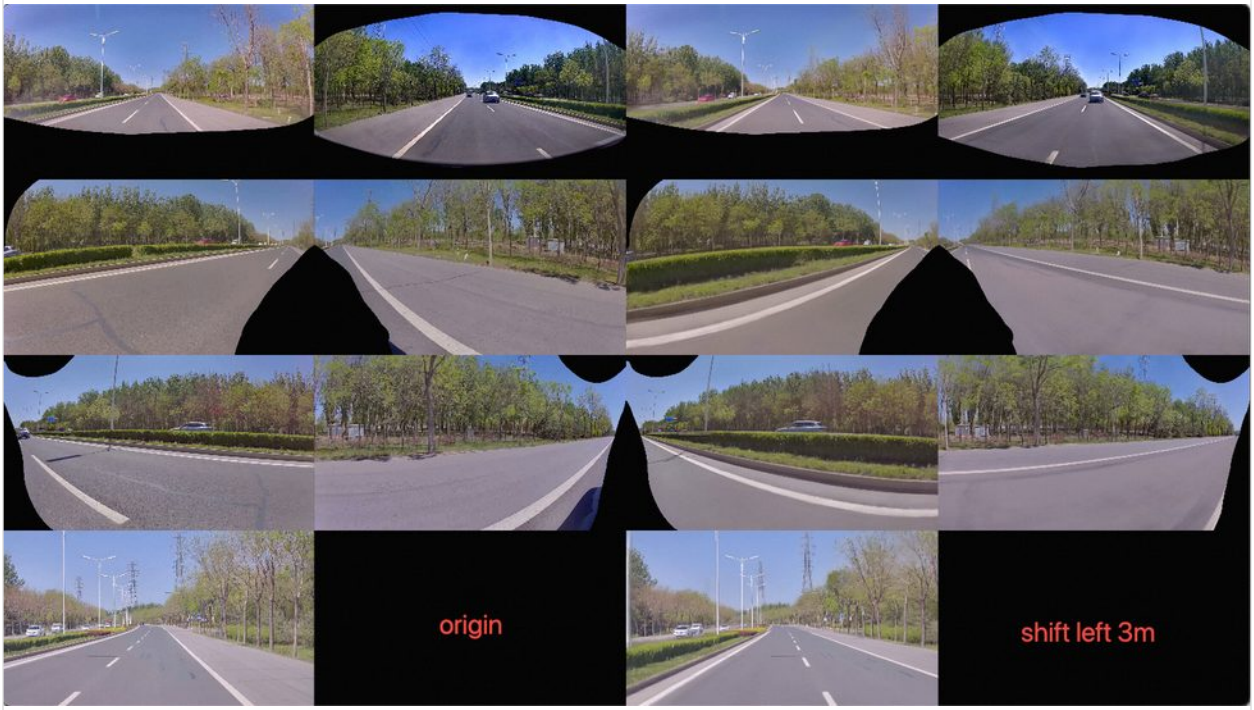
Li Auto addresses this by implementing a ColorOffset module within LiAutoScaffoldGS. This system assigns learnable, time-dependent color offsets to each signal's Gaussian, conditioned on observed state transitions. This approach ensures reconstruction of color changes with precise temporal alignment—accurately capturing transitions for traffic signals and vehicle lighting—which is fundamental for high-fidelity closed-loop simulation in intersection and car-following scenarios.

Virtual View Sampling

Supervision based solely on original perspectives is insufficient to ensure superior novel view synthesis, particularly regarding background elements and road surfaces shifted laterally or vertically from the ego-position. Without explicit training signals for novel views, 3D GS models often degrade into view-dependent memorization rather than authentic 3D reconstruction. Li Auto addresses this by projecting LiDAR point clouds and camera frames into synthetic offset perspectives to create virtual training samples. Utilizing these rendered pseudo-ground truth images as auxiliary supervision significantly enhances reconstruction

fidelity for off-axis perspectives, ensuring highly credible simulations for lane-changing maneuvers, lateral offsets, and alternative trajectory planning.

Figure 5: The effect of virtual view sampling. The left two columns are original views in the reconstruction. The right two columns are 3 meters left translation views in the reconstruction.

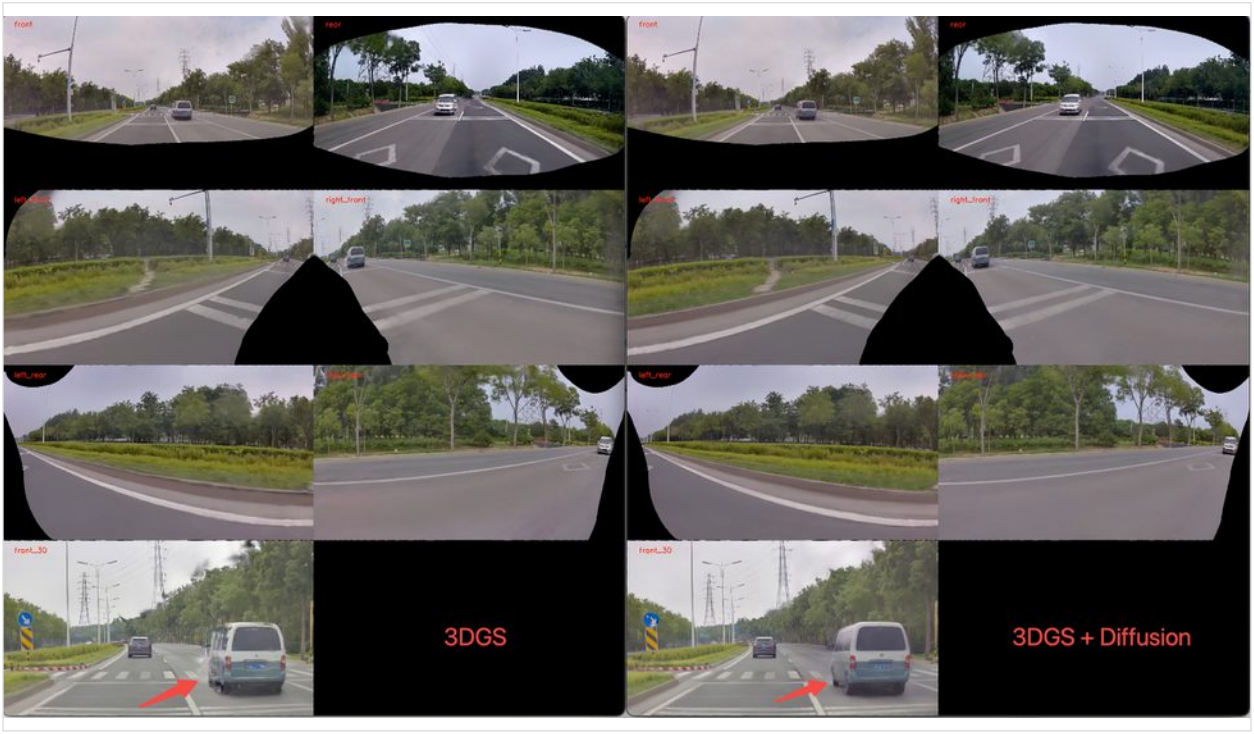


Enhance Reconstruction with GenAI

Generative AI such as image diffusion models plays an important role in novel view synthesis in both Li Auto solution and NuRec.

Despite the implementation of virtual view sampling, foreground actors such as VRUs and vehicles frequently suffer from occlusion-induced gaps and sparse point cloud density in novel perspectives, resulting in blurred or fragmentary reconstructions. To address these limitations, Li Auto incorporates a diffusion-based enhancement phase during the training cycle for rendered RGB frames. This generative approach repairs incomplete or occluded foreground regions within the synthetic views, producing high-fidelity samples that serve as auxiliary supervision. This methodology substantially improves the fidelity of novel-view foreground reconstruction, particularly for dynamic objects that are only partially visible from a single observation point in the original driving logs.

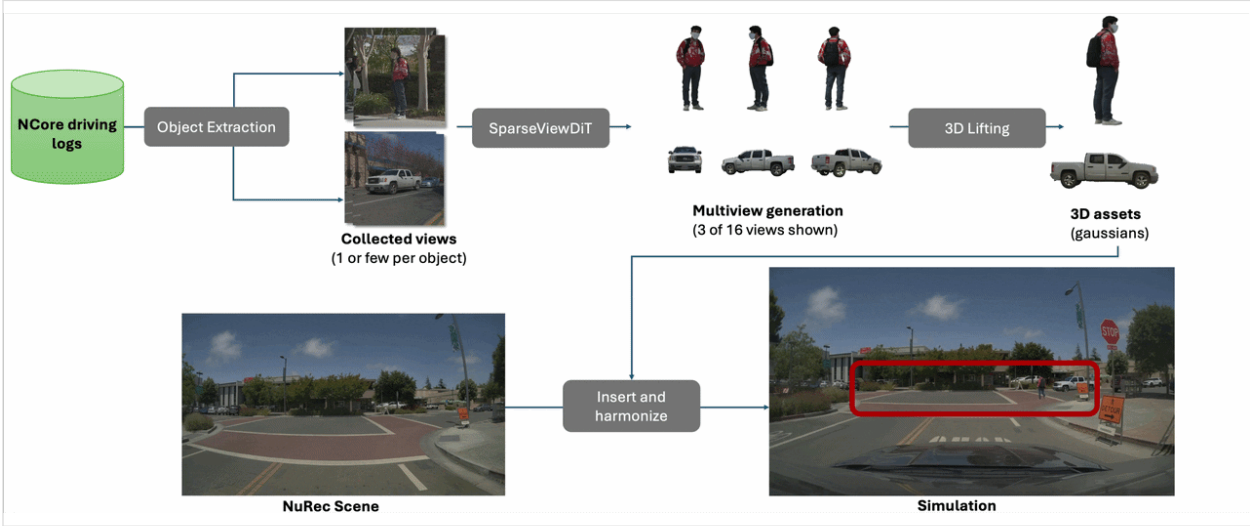
Figure 6: The effect of GenAI enhanced reconstruction. The left two columns are novel views in the reconstruction without Diffusion model. The right two columns are novel views with a Diffusion model.



NVIDIA Omniverse NuRec provides two components based on generative AI to enhance the novel view quality of reconstruction: Asset Harvester and Harmonizer.

Asset Harvester is an image-to-3D model and end-to-end system that converts sparse, in-the-wild object observations from real driving logs into complete, simulation-ready assets. The model generates 3D assets from a single image or multiple images of vehicles, VRUs or other road objects extracted from autonomous driving sessions. Check the model card for more details: https://huggingface.co/NVIDIA/asset-harvester

Figure 7: The overall pipeline of Asset Harvester



We recommend running the pipeline with NCore format data — through this way, Asset Harvester could leverage the camera poses from driving logs when doing the asset generation. However, Asset Harvester also supports starting from a single image — we will estimate the camera poses from the image. The estimation might not be as precise as the recorded data from NCore and potentially lead to tilted assets in its local canonical coordinates system.

Usually, the assets from Asset Harvester are not harmonious to the original scene — There are the mismatched color (appearance) and missing shadow. We could disable the learnable post-processing in reconstruction, e.g., PPISP to relieve color mismatch. Then apply Harmonizer to the rendered frames to further enhance the appearance and shadow of the inserted assets.

Harmonizer is a diffusion-based enhancement model designed to bridge neural reconstruction and photorealistic simulation. This main functionalities of this model are:

- Fix (remove) the artifacts in novel view synthesis.
- Harmonize the inserted asset with the NuRec reconstructed background.

Its backbone is CosmosPredict2 0.6B, fine-tuned using a custom data curation pipeline for autonomous driving and large-scale robotics simulation. For more information, check https://huggingface.co/NVIDIA/Harmonizer

Figure 8: The influence of Harmonizer. This model adds shadow to the inserted asset and removes the artifacts in novel view

While Harmonizer can bring obvious visual benefit, it usually introduces regression on image quality KPI (and downstream task KPI). One of the possible reasons for this regression is the domain gap (distortion, ISP, etc.) between original cameras frames used in training Harmonizer and the target cameras frames in the deployment. A way to reduce such regression is to do camera domain adaption. The recipes are:

- Build camera domain adaption dataset
  - Use regular reconstruction frames to form the training image pairs.
  - Use frames from all target cameras.
  - Usually, A small number (>= 3K) image pairs should be sufficient for the camera domain adaption. But you can increase the number of frames for a better adaptation.
- Do the post-training on 10X smaller learning rate and 10 epochs

The camera domain adaption could resolve the quality regression issue. For more details of post-training, please check: https://github.com/NVIDIA/harmonizer/tree/main

Li Auto's solution integrates the Harmonizer, which helps enhance the appearance of the generated assets.

Multi-Camera Color Match

Autonomous driving logs usually contain several cameras capturing a wide range of views. Besides the different calibration parameters (and distortion models), these cameras might also have variations in image signal processing (ISP) which will lead to photometric inconsistency in reconstruction. NVIDIA Omniverse NuRec integrates Physically-Plausible ISP (PPISP) to address this issue. The PPISP is enabled by default in `configs/apps/prod/Hyperion-8.1/car2sim.yaml`.

Figure 9a: The rendering result without per-camera PPISP

Figure 9b: The rendering result with per-camera PPISP

Efficiency

Enhancing the efficiency of the reconstruction engine is vital to handle extensive reconstruction and simulation rendering demands. This section reviews methodologies for optimizing the performance of the reconstruction solution.

Unified Representation of LiDAR and Camera

LiDAR point clouds serve as a critical input modality for many autonomous vehicle (AV) algorithms, complementing traditional camera-based data. Consequently, the capability to render LiDAR point clouds from reconstructed scenes is essential for comprehensive simulation. However, conventional 3D Gaussian Splatting (3DGS) is not inherently optimized for LiDAR data. This typically necessitates the maintenance of separate representational models for LiDAR and cameras, effectively doubling the engineering effort required to support both modalities in simulation.

Built upon 3DGUT's 3D Gaussian representations, NVIDIA Omniverse NuRec offers native support for a unified LiDAR and camera representation, providing several key benefits:

- A single set of Gaussians enables highly efficient rendering of both LiDAR point clouds and image frames.
- Novel view synthesis for cameras is enhanced via geometry supervised by LiDAR, though a slight trade-off in original view quality may occur because camera pixels and LiDAR points are not perfectly aligned.

Utilizing this capability requires saving a LiDAR model that details the ray shooting pattern (azimuths and elevations) alongside LiDAR data frames containing per-point timestamps and point positions. For further details, see the references below:

- LiDAR model: https://nvidia.github.io/ncore/data/sensor_models.html#lidar-models
- LiDAR sensor frames: https://nvidia.github.io/ncore/data/formats.html#sensor-components

In addition to representation, NVIDIA Omniverse NuRec coordinates the combined supervision of camera and LiDAR systems for dynamic objects within the AV scene. By rendering LiDAR point clouds and camera frames independently, NVIDIA Omniverse NuRec automatically optimizes the positioning of dynamic tracks based on the specific timestamps of individual sensor frames.

Figure 10a: Without LiDAR and camera orchestration, the vehicle of high speed is broken in the reconstruction

Figure 10b: with LiDAR and camera orchestration, the vehicle of high speed maintains its shape and appearance

Li Auto's solution also integrates 3DGUT's LiDAR rendering capability, achieving approximately 50% cost savings in the reconstruction process.

Optimization in Reconstruction and Rendering Implementation

To scale neural reconstruction to industrial-grade throughput, Li Auto implemented two pivotal efficiency enhancements:

- **Batch Optimization:** Transforms sequential per-frame optimization into batch operations, maximizing CPU/GPU utilization and boosting training throughput for large-scale environments.
- **FasterGS:** Integrates high-performance CUDA rendering kernels from FasterGS into LiAutoScaffoldGS, significantly accelerating both training and inference phases.

Furthermore, the system incorporates advanced GPU memory optimizations:

- **Early Opacity Pruning:** Refines the opacity masking logic for ScaffoldGS-based models by removing low-opacity Gaussians prematurely, which substantially lowers peak VRAM requirements.

- **Conditional Concatenation:** Optimizes the allocation of `cat_local_view` features by performing concatenation on-demand, rather than via pre-allocation, drastically reducing memory overhead during training.

Built upon the widely used gsplat differential rendering library, NVIDIA Omniverse NuRec integrates numerous rendering optimizations contributed by NVIDIA:

- Accelerated rasterization techniques.
- FP16 precision in spherical-harmonics computation.
- HiGS implementation for inference-only rendering processes.
- An optimized kernel tailored for the MCMC strategy.
- …

In addition to rendering optimizations, the reconstruction implementation within NuRec has been significantly enhanced by NVIDIA:

- Fusing various small kernels that are executed during the Gaussian parameter collection phase.
- Removing nearly all CPU-GPU synchronization points throughout the reconstruction lifecycle.

In summary, utilizing NVIDIA's benchmark clip configured with six cameras via `car2sim_6cam.yaml` on an RTX6000Pro, the system achieves 3.33 reconstructions per hour alongside a rendering speed of 100 FPS at 1080P resolution. For additional insights, explore the technical blog: https://developer.nvidia.com/blog/optimizing-a-neural-reconstruction-pipeline-using-nvidia-nsight-developer-tools/

Multi-GPU Reconstruction

By default, NVIDIA Omniverse NuRec runs reconstruction with one single GPU. Multi-gpus and multi-nodes mode can be enabled to get faster training reconstruction in some emergency situations. Multi-GPU and multi-node training for NuRec using a hybrid solution that combines Distributed Data Parallel (DDP) and Fully Sharded Data Parallel (FSDP) methods for optimization. Our approach distributes 3D Gaussian particles across GPUs, and with a unique focus on the reconstruction of dynamic and deformable objects, which are essential for autonomous-driving applications.

Performance results indicate that NuRec achieves an approximate 4x to 5x acceleration using 8 GPUs, varying with specific devices, datasets, and configurations. While it can scale past 64 GPUs for enhanced speed-up ratios, this scaling is non-linear. Furthermore, there is a risk of image quality regression if configurations are not adjusted from single-GPU baselines. For more information, check Run on Multi-GPU Systems.

Li Auto's solution integrates the multi-GPU scheme of NuRec, enabling them to achieve a 4x speed-up using 8 GPUs.

InstantNuRec Initialization

NVIDIA Omniverse NuRec allows for the initialization of Gaussians from a variety of source point clouds. In addition to the LiDAR point clouds typically utilized in conventional approaches, NuRec is capable of reconstructing scenes using predicted Gaussians derived from InstantNuRec, a feedforward 3DGS model.

Reconstructing dynamic outdoor scenes from autonomous-vehicle driving logs traditionally requires lengthy per-scene optimization. InstantNuRec takes a different route: a feed-forward transformer directly infers a dynamic 3D-Gaussian scene representation in a single forward pass. Given a short window of multi-camera observations from an AV log, the model predicts a Gaussian primitive per pixel — covering geometry, appearance, and per-Gaussian motion. NuRec uses the Gaussians from InstantNuRec for initializing the background Gaussians layers — with these Gaussians already representing the background environment well, NuRec significantly reduces required training iterations without sacrificing reconstruction quality and compensates areas that are not scanned by the LiDAR. For more information, check Launch Reconstruction Model Training.

Evaluation Metrics

Evaluation metrics at reconstruction engine side aim to guide the iteration of the software. Li Auto's reconstruction evaluation framework incorporates three primary dimensions:

- **Sensor modality metrics:** encompassing sensor fidelity standards such as PSNR/FID and LiDAR-specific MAE.
- **Downstream perception benchmarks:** including performance indicators for occupancy, as well as static detection (lane detection) and dynamic object detection.
- **Closed-loop evaluation:** comparative analysis against product-version of reconstruction software, utilizing identical closed-loop trajectories or synthesizing new trajectories through re-simulation.

Each successive reconstruction iteration undergoes baseline benchmarking against its predecessor across these domains. A release is approved for promotion once it demonstrates tangible gains in key metrics without introducing unexpected performance regressions.

Sensor Modality Metrics

Li Auto deploys the following metrics for camera image quality:

- **Peak Signal-to-Noise Ratio (PSNR):** Serves as the primary metric for pixel-level fidelity, quantifying the quality of reconstruction for the original perspectives.
- **Fréchet Inception Distance (FID):** Evaluates novel view synthesis at the distributional level; in the absence of ground-truth frames for novel perspectives, this acts as a proxy for perceptual quality by comparing against real driving data distributions.

Li Auto deploys the following metrics for LiDAR point cloud quality:

- **Mean Absolute Error (MAE):** For the LiDAR reconstruction, this metric quantifies geometric precision by measuring the absolute discrepancy between synthetic point clouds—or depth maps—and the corresponding ground-truth data, where lower values signify superior fidelity.

Downstream Task

The effectiveness of the engine is further validated by its performance on downstream perception tasks. Li Auto conducts rigorous quantitative benchmarking of perception models—focusing on occupancy prediction, camera/LiDAR-based 3D object detection, and multimodal lane detection—across original driving logs and successive reconstruction iterations. This comparative analysis ensures that every new release maintains or exceeds the functional benchmarks established by previous versions, preventing any degradation in simulation fidelity.

Pre-Simulation Evaluation

Before executing full closed-loop simulations, Li Auto performs two distinct levels of pre-simulation validation to ensure reconstruction integrity:

- **Visual comparison (Legacy closed-loop):** The system replays the original ego-trajectory within the newly reconstructed environment, comparing the output against previous visual baselines. This stage is designed to identify significant rendering regressions—such as artifacts, temporal flickering, or color errors—prior to comprehensive simulation.
- **Behavioral verification (Re-simulation):** The closed-loop simulation is re-executed using the updated reconstruction to compare the resulting synthetic ego-trajectory with recorded ground truth. If the reconstructed scene faithfully mirrors the original driving environment, the simulated vehicle should exhibit analogous behaviors; any significant divergence indicates that the reconstruction has introduced a simulation-to-reality gap.

Simulation System

Li Auto's simulation platform serves as a comprehensive, full-stack, in-house developed multi-sensor closed-loop environment tailored for autonomous driving development. The system integrates proprietary modules—including a neural reconstruction engine, physics-based ego-controllers, dynamic object editing, and traffic agent simulation—within a unified orchestration framework across all primary sensor modalities. With a plug-and-play design and a self-developed deterministic middleware scheduling strategy, it can flexibly combine most Li Auto-developed AD production and pre-research algorithm modules, such as end-to-end/VLA, campus roaming, automated parking, and active safety. This section explores Li Auto's practical methodology in full-stack simulation system development together with an introduction to NVIDIA Omniverse NuRec APIs designed for constructing simulation systems.

Functionality

Sensor Simulation

Li Auto's simulation framework supports 11+ camera IDs, providing comprehensive coverage for the entire autonomous driving perception stack:

- **Driving:** includes front-view and surround-view cameras
- **Parking:** accommodates panoramic fisheye cameras for APA and AVP scenarios

The system features the following core capabilities:

- **GPU IPC (Inter-Process Communication):** facilitates real-time frame sharing across multiple cameras and processes.
- Low-latency multi-stream synchronization across all camera channels.
- Unified camera and LiDAR reconstruction via a shared world model instance and rendering interfaces, orchestrating multi-sensor alignment strategies online. It enables sensor frame-level alignment with real-world source scenes and supports multiple pixel layout formats according to raw outputs from production vehicles
- Millimeter-wave radar integration for fusion with other perception sensors.
- A complete sensor playback pipeline with GPU acceleration.
- Flexible deployment options:
  - Option 1: Asynchronous gRPC service deployment, decoupling the C++ simulation engine from Python world model inference to support independent scaling;

- Option 2: Single-machine deployment, utilizing GPU shared memory for communication between AD algorithm services and the rendering engine to achieve maximum efficiency.

NVIDIA Omniverse NuRec offers a comprehensive set of gRPC APIs to support sensor simulation. A standard request for rendering camera frames is shown below:

```
   request = CameraRenderRequest(
        scene_id="your_scene_id",
        resolution_h=300,
        resolution_w=400,
        camera_intrinsics=front_wide_camera.intrinsics,
        frame_start_us=middle_timestamp,
        frame_end_us=middle_timestamp + 1,
        sensor_pose=PosePair(
            start_pose=se3_to_grpc_pose(pose),
            end_pose=se3_to_grpc_pose(pose)
        ),
        image_format=ImageFormat.JPEG,
        image_quality=95,
        dynamic_objects=[]
    )
```

While most parameters are straightforward, several key aspects deserve special attention:

- `resolution_h` and `resolution_w` controls the resolution of the rendered camera frame. When these two parameters specify a setting that alters the aspect ratio of the camera model, only the resolution with the higher scale ratio will take effect.
- `camera_intrinsics` configures the camera model (intrinsic parameters and distortion model) used during rendering. Users have the option to specify a new camera model with custom parameters or reuse the cameras from the reconstruction by configuring this protobuf message:

```
message CameraSpec {
    string temporary_camera_spec = 1 [deprecated = true];
    oneof camera_param {
        FthetaCameraParam ftheta_param = 2;
        OpenCVPinholeCameraParam opencv_pinhole_param = 3;
        OpenCVFisheyeCameraParam opencv_fisheye_param = 4;
    }
    // these two are for book-keeping by the user
    string logical_id = 5;
    // DEPRECATED: This field is no longer used in alpasim runtime code and
    // NRE no longer supports multiple trajectories
    uint32 trajectory_idx = 6 [deprecated = true];
    uint32 resolution_h = 7;
    uint32 resolution_w = 8;
    ShutterType shutter_type = 9;
    oneof external_distortion {
        BivariateWindshieldModelParameters bivariate_windshield_model_param = 10;
```

```
    }
}
```

- Set `logical_id` only and set it to the camera string name used in the reconstruction to leverage that camera directly.
- Set `camera_param` to enable a new camera model. NuRec gRPC API supports opencv-pinhole, opencv-fisheye and FTheta camera model as in the reconstruction. For more information about camera parameters in protobuf message, check https://docs.nvidia.com/nurec/api/nre.grpc.protos.sensorsim.html#message-nre.grpc.protos.sensorsim.CameraSpec
- Set `shutter_type` to enable rolling shutter effect in the rendering.
- To activate the rolling shutter effect, users must configure the frame capture duration using `frame_start_us` and `frame_end_us`, and the corresponding positions at these timestamps through `sensor_pose`, in addition to defining the type within `camera_intrinsics`.
- Timestamp `frame_start_us` and `frame_end_us` should be set within the time range of the original reconstruction:
  - To replay the driving log, the timestamp should be set as the camera frame in the original NCore data. And `dynamic_objects` can be set to an empty list.
  - To deal with novel view synthesis at the timestamp out of original reconstruction's range (This is quite common if the ego vehicle slows down in the closed-loop simulation compared to the original trajectory for reconstruction):
    > For a pure static scene, use a fixed valid timestamp. This will not influence the reconstruction results.
    > For a scene with dynamic objects, besides the fixed timestamp, explicitly control the dynamic objects position through the argument `dynamic_objects` in the render request.

NVIDIA Omniverse NuRec also supports rendering LiDAR point clouds through gRPC API. Below is a typical LiDAR rendering request:

```
request_data = LidarRenderRequest(
    scene_id="your scene id",
    frame_start_us=LiDAR_start_timestamp_us,
    frame_end_us=LiDAR_end_timestamp_us,
    sensor_pose=PosePair(
            start_pose=LiDAR_start_pose,
            end_pose=LiDAR_end_pose
            ),
    dynamic_objects=dynamic_objects,
    render_filter=render_filter,
)
```

There are similar parameters to the camera rendering request like `frame_start_us`, `frame_end_us`, `sensor_pose`, `dynamic_objects`. Besides the common parts, there are several things to be noticed:

- At present, configuring an arbitrary LiDAR model is not supported, unlike the flexible options available for camera rendering in NuRec; only the original LiDAR device utilized during the reconstruction phase can be used.
- The positions of the rendered points are transformed into the sensor coordinate system at the frame-end timestamp.
- To eliminate noise points in the final rendering, three distinct point filters are available. The thresholds for these filters can be adjusted using the `render_filter` argument:
  - **Per-ray opacity filter:** This filter removes rays with an accumulated opacity below a default threshold of 0.8, effectively clearing numerous floaters at the ego vehicle region.
  - **Raydrop filter:** This option filters out rays exceeding a default drop probability of 0.5.
  - **Distance-based filter:** This mechanism excludes rays intersecting with Gaussians across highly variable distances, aiding in the removal of noise along the boundaries between foreground objects and the background.

Dynamic Objects Editing

Li Auto has implemented a log2sim framework, enabling versatile editing of dynamic actors within simulation environments:

- **Multi-level object description:** Each dynamic object is characterized by its spatial coordinates, orientation, velocity, and dimensions (utilizing both tight and full box representations), alongside expanded semantic labels and real-time signaling states such as turn indicators and braking status.
- **Dual-trajectory architecture:** The system maintains two parallel paths for every dynamic actor — `logTrajectory` (derived from original data logs) and `simTrajectory` (modified via simulation) — facilitating a seamless integration of recorded behaviors with synthetic behaviors.
- **Driving model engagement:** Specialized interfaces allow new driving model's behaviors to override the recorded log trajectory at user-defined configuration points.

NVIDIA Omniverse NuRec provides a comprehensive set of API to edit the dynamic objects including the native reconstructed objects and the assets generated from Asset Harvester. The typical steps to edit native reconstructed objects are:

- Start the grpc-server with the option `--enable-editing-actors`.
  - It is set to false by default and the gRPC server will just do replay for all the dynamic objects in a scene.

- Perform editing through `dynamic_objects` in `CameraRenderRequest` and `LidarRenderRequest`. The `dynamic_objects` is a list of message like:

```
message DynamicObject {
    string track_id = 1;
    PosePair pose_pair = 2;
}
```

- **Replay:** Explicitly specify each track using its ID alongside the pair of positions at both the start and end timestamps of the sensor frame.
- **Remove:** Delete a given `DynamicObject` from the replay list.
- **Move:** Adjust the corresponding `pose_pair` details.

Editing assets from Asset Harvester follows a similar process, but requires some additional considerations and steps:

- Before the gRPC server processes the assets, they must be included in the `.usdz` file. See Add Asset Harvester Output to a Reconstruction.
- Use the asset editing request `EditAssetsRequest` prior to the sensor render request when replacing a native track or inserting a new asset with a new track ID.

```
message DynamicObjectTrack {
    string id = 1;
    string semantic_class = 2;
    nre.grpc.protos.common.Trajectory trajectory = 3;
    nre.grpc.protos.common.AABB object_size = 4;
    string asset_id = 5;
}
message ReplaceAssetAction {
    string original_id = 1;
    string replacement_id = 2;
    nre.grpc.protos.common.AABB object_size = 3;
}
message EditAssetsRequest {
    string scene_id = 1;
    repeated ReplaceAssetAction replace = 2;
    repeated DynamicObjectTrack insert = 3;
}
```

- **Replace:** Substitute the original reconstructed track specified by `original_id` with the asset designated by `replacement_id`.
- **Insert:** Introduce a new track using the identifier specified by `id` and populate it with Gaussians from the asset defined by `asset_id`. NuRec references the `semantic_class` to identify the specific Gaussian layer where the new track will be added.
- Start the gRPC server with option `--enable-harmonizer` to harmonize the assets with the original reconstructed background.

- NuRec gRPC server provides API `restore_model_parameters` to restore all the assets insertion and replacement.

Physics Modeling: Ego

Li Auto utilizes a Model Predictive Control (MPC) framework as the primary ego-vehicle physics architecture within its simulation environment, while providing support for:

- Production-vehicle controller-in-the-loop integration
- Idealized control models for upper-bound performance benchmarking
- Pure Pursuit algorithms tailored for simplified operational scenarios

The system employs co-optimization of dynamic modeling and MPC controllers to ensure stable ego-vehicle trajectory. Evaluation criteria for performance include:

- Maximum lateral error: < 0.2 m
- Maximum longitudinal error: < 0.5 m

The core philosophy emphasizes the decoupling of autonomous driving logic from control algorithms. For AV algorithm validation, high-fidelity dynamic modeling is rarely the primary constraint; the critical requirement is a reliable, predictable controller capable of consistently following the trajectory predicted by the AV algorithm stack. This ensures that simulation variances are attributable to perception or planning iterations rather than the noise from physical modeling.

Traffic Sim

Li Auto incorporates two distinct traffic simulation agent paradigms:

- **IDM Agent:** utilizes the traditional Intelligent Driver Model (IDM) to manage deterministic, rule-based NPC traffic flow within standard operational contexts.
- **Sim Agent:** a neural network-driven behavioral model for NPCs. It supports reactive traffic flow scenarios.

The core philosophy is that traffic simulation agents are primarily reserved for highly complex traffic interactions or rare corner cases where recorded logs are insufficient (e.g., cut-ins, near-misses, or emergency braking maneuvers). For standard generalization benchmarking, the system prioritizes replay-based traffic flow to ensure maximum reproducibility.

Efficiency

Li Auto achieves high-throughput simulation by optimizing the framework across both individual instance performance and large-scale multi-instance orchestration:

Single-instance optimization:

- Deep integration between simulation scheduling and AD algorithm pipelines.
- Incorporation of E2E, VLA, and safety redundancy modules into the core orchestration interface.
- DAG-based task scheduling for graph-driven execution.
- Modular application architecture utilizing shared memory for inter-process communication.
- Asynchronous CPU/GPU execution, enabling overlap between CPU orchestration and GPU-side rendering or inference.

High-concurrency multi-instance scaling:

- GPU virtualization (partitioning into 1:2 or 1:4 ratios) to maximize hardware utilization across parallel simulation threads.
- Comprehensive caching of artifacts and assets: unified access to reconstructed assets and scene data reduces redundant I/O overhead during mass deployment.

Simulation Usage

Build the Clips Library for Simulation

Li Auto organizes its comprehensive simulation task library around five primary operational domains:

- **General Driving:** Integrating multiple sensors to reconstruct complex urban roads and highway scenes, verifying NOA functionality.
- **Parking:** Utilizes fisheye reconstruction and panoramic simulation to ensure full coverage for APA and AVP functional scenarios.
- **Traffic Signals:** High-fidelity reconstruction of dynamic signal transitions, specifically targeting complex intersections, unprotected turning, and critical yellow-light decision-making.
- **Must-Solve Set (High-Value Use Cases):** Focuses on complex edge-case scenarios requiring system intervention and safety-critical boundary cases identified from authentic fleet data logs.

- **Targeted Tasks:** Construction of specific capability gap evaluations through large-scale data mining and manual scene editing methodologies.

Li Auto's clips library is constructed based on the following standards:

- **MPI from Closed-loop Validation System and Scene Analytics:** Li Auto utilizes fleet data gathered through shadow mode to establish comprehensive scene distribution statistics. Miles Per Intervention (MPI) in the real world is regarded as the foundational benchmark or reference for building the clips library.
- **Clips Multidimensional Annotation:** Automatically annotating the environment (such as weather and road conditions) and abnormal behaviors (such as sudden braking, line pressure, and cutting) for videos, aiming to achieve efficient and focused analysis of specific scenes.
- **General Driving Library Construction:** Li Auto has developed a generalization set comprising 400,000 clips, designed to balance general label distributions while significantly amplifying the weight of rare, high-value anomalous events. This methodology ensures precise distribution alignment, allowing the simulation library to faithfully mirror real-world environmental complexities and the actual frequency of edge-case scenarios.

Reproduce Rate in the Simulation

Li Auto evaluates simulation reproducibility at the event cluster level, prioritizing the positive correlation between synthetic environments and specific real-world behaviors (highlighted events or identified clusters).

Individual clip-level metrics lack preamble information, and segment level metrics often suffer from excessive noise. By mining similar scenes and combining them with scene labels — for instance, "unprotected left turns at complex intersections" or "cut-ins during highway maneuvers exceeding 80 km/h"—the system calculates simulation consistency across each event cluster. This methodology enables targeted diagnostics to identify scenarios with low simulation correlation, guiding enhancements in reconstruction and simulation fidelity.

Consistency in Simulation

Simulation consistency encompasses two vital dimensions:

- **Simulation-to-Reality Consistency:**
  - Accurately reflecting real vehicle status with high fidelity.
- **Internal System Consistency:**
  - Ensuring deterministic execution outcomes when using identical autonomous driving software stacks and reconstructions.

Li Auto pursues Simulation-to-Reality Consistency through a six-layered engineering framework:

- **Deterministic Scheduling:** Incorporates flow-controlled playback, simulation orchestration, and topological replication to ensure that the identical data path from sensors to the autonomous driving stack is faithfully reproduced within simulation.
- **Control Layer:** Features co-optimization of simulation MPC controllers and vehicle dynamics models to match authentic vehicle trajectories.
- **Perception Input Layer:** Focuses on high-frequency visual information restoration and noise modeling to reconstruct comprehensive sensor input signals, including precise frame rates and sensor-specific noise patterns.
- **State Layer:** Implements prior state injection and temporal feature replay (historical open-loop feature playback) to inject previously computed perception or prediction features, thereby reproducing stateful algorithm behaviors.
- **Preprocessing Layer:** Guarantees vehicle-to-cloud consistency for sensor data processing, ensuring that all preprocessing transformations—such as undistortion, synchronization, and coordinate transforms—applied in simulation align perfectly with the real vehicle.
- **Reconstruction Layer:** Optimize the neural reconstruction to maintain photometric, textural, and geometric consistency with real sensor characteristics.

For Internal System Consistency for closed-loop simulation, in addition to deterministic scheduling and preprocessing, Li Auto invests substantial engineering effort into mitigating stochasticity within both the rendering pipeline and autonomous driving software stacks to ensure determinism in the valid numerical range.

Strategy to Iterate the Simulation System with the Neural Reconstruction

Li Auto's philosophy for iteratively advancing the simulation system alongside the neural reconstruction is built upon a foundation of progressive validation and a hybrid simulation paradigm.

Rather than postponing deployment until the reconstruction achieves total photorealism, Li Auto operates a mixed environment that integrates closed-loop simulation with strategic open-loop components:

- **Closed-loop simulation:** The ego-vehicle algorithm actively dictates the trajectory, with the neural reconstruction providing real-time rendering for novel perspectives. While this offers a rigorous test of generalization capabilities, it necessitates superior quality in novel view synthesis.

- **Open-to-closed transition:** The ego-vehicle initially follows the recorded log trajectory (or a slightly perturbed version) to accumulate authentic state data via raw imagery. Upon reaching a designated keypoint, the system transitions to closed-loop control. Because rendering occurs at or near original perspectives, this phase maintains high fidelity while verifying closed-loop behavioral responses within the reconstructed environment.

By unifying these methodologies, Li Auto effectively achieves the following:

- Leverages open-loop signals as a pre-warm and constraint mechanism, ensuring synthetic behaviors more closely mirror real-world dynamics.
- Scales the proportion of closed-loop testing in tandem with reconstruction quality enhancements, ensuring continuous utility throughout the development cycle.
- Employs benchmarking as a gatekeeper for version deployment—prioritizing releases that boost closed-loop reproducibility over those that merely improve original-view PSNR.

This hybrid iteration strategy facilitates a collaborative evolution between the simulation framework and the reconstruction, where the constraints of each component serve to guide the development priorities of the other.

Acknowledgments

This article is co-authored by Tyler Zhu and Phoebe Li of NVIDIA, and Mofan Zhou, Yimeng Li, and Chen Liu of Li Auto, with substantial support from the NVIDIA Omniverse NuRec product team and Li Auto reconstruction and simulation team.