Aria-NBV: Quality-Driven Next-Best-View Planning with Egocentric Foundation Models

Jan Duchscherer
Department of Computer Science & Mathematics
Munich University of Applied Sciences
Munich, Germany
j.duchscherer@hm.edu

Abstract—Next-Best-View (NBV) planning addresses the fundamental challenge of autonomous viewpoint selection in active 3D reconstruction, aiming to maximize acquisition quality under a limited capture budget. Classical NBV methods rely on hand-crafted criteria, limited action spaces, or per-scene optimized representations. Learning-based NBV methods improve generalization but still optimize geometric coverage as a proxy for reconstruction quality, which can fail in cluttered scenes with occlusions and fine details. Diretly optimizing reconstruction quality, as pioneered by VIN-NBV [1], improves candidate ranking via Relative Reconstruction Improvement (RRI) but remains limited to object-centric scenarios without pre-trained foundation-model priors.

We introduce Aria-VIN-NBV, an oracle labeling pipeline for quality-driven RRI based NBV on egocentric indoor trajectories in Aria Synthetic Environments (ASE). We compute oracle RRI labels by rendering candidate depths from ASE ground-truth meshes and scoring their RRI relative to a previously captured SLAM point clud. Utilizing these labels we train an RRI prediction model leveraging an egocentric foundation model (EFM3D) backbone to capture rich priors from large-scale pre-training.

Index Terms—next-best-view, relative reconstruction improvement, egocentric foundation models, EFM3D, Aria Synthetic Environments, ordinal regression

I. Introduction

Active 3D reconstruction systems must decide where to move the sensor next in order to maximize reconstruction quality under limited capture budgets [1]. This next-best-view (NBV) problem is especially challenging for egocentric platforms where scenes are large, cluttered, and partially observed. Many classical NBV methods rely on proxy criteria such as coverage or information gain, which can miss occluded or geometrically complex regions [1]. Recent methods such as VIN-NBV instead optimize reconstruction quality directly by predicting Relative Reconstruction Improvement (RRI) for candidate views in object-centric capture settings [1]. Our goal is to bring this quality-driven paradigm to the Aria ecosystem by combining the Aria Synthetic Environments (ASE) dataset, the EFM3D/EVL foundation model stack, and an oracle RRI computation pipeline tailored to egocentric trajectories [2], [3]. Project Aria provides an egocentric multi-modal recording platform and toolchain, including calibrated, time-aligned sensor streams and machine perception services that recover accurate trajectories and semi-dense reconstructions from raw recordings [4]. Importantly, the ecosystem spans both scalable synthetic data (ASE) and real-world Aria datasets (e.g., AEO in the EFM3D release), which share the same geometric primitives (trajectory, cameras, semi-dense SLAM points) and conventions [2]. This makes sim-to-real transfer directly testable: we can iterate on expensive supervision (oracle RRI) in simulation while keeping the learned scoring model compatible with real Aria recordings.

Concurrently, reinforcement-learning approaches such as GenNBV learn continuous 5-DoF free-space policies, using multi-source state embeddings and coverage-gain rewards to improve cross-dataset generalization [5]. However, coverage remains a surrogate for reconstruction quality and can under-prioritize occlusions and fine details [1]. VIN-NBV instead leverages an oracle RRI signal to train a lightweight candidate-ranking network via imitation learning, making the objective directly reconstruction-quality aligned [1].

We do not yet learn an end-to-end action policy. Instead, we first establish a high-fidelity supervision signal and train a model that predicts it, which can later serve as the backbone for an NBV policy. This focus is practical: a single candidate evaluation requires depth rendering and point↔ mesh distance computations across many candidate views. Following VIN-NBV, we discretize the action space into a candidate set and compute per-candidate oracle RRI labels [1]. We then train a VIN v3

candidate scorer on these labels using a frozen EVL backbone and report training diagnostics (Section Section X and Appendix Section XVIII). Learning a fully integrated NBV policy remains future work.

We target the following setting. Each ASE scene provides a prerecorded Aria trajectory with RGB and SLAM cameras, semi-dense SLAM points, and (for a 100-scene validation subset) a watertight ground-truth mesh. We sample candidate camera poses around the current rig pose, render candidate depth maps from the mesh, and compute oracle RRI scores by scoring the relative reconstruction improvement that the backprojected candidate depths would provide when fused with the existing semi-dense SLAM points.

Contributions. This paper documents an oracle supervision pipeline and a trained VIN v3 baseline for quality-driven NBV on Aria Synthetic Environments:

1) Oracle RRI labeling pipeline for ASE that samples candidate views, renders metric depth from ground-truth meshes, backprojects candidate point clouds, and evaluates reconstruction quality against the ground-truth mesh.
2) Reproducible immutable VIN offline store of oracle samples (candidates, renders/point clouds, labels, and frozen EVL features) to enable batched training without re-running the expensive oracle computations.
3) Trained VIN v3 candidate scorer on frozen EVL voxel features with view-conditioned evidence (pose encoding, voxel coverage proxies, semi-dense projection cues) and an ordinal CORAL head; we report training/validation diagnostics.
4) Diagnostics and convenience first research framework around PyTorch Lightning (modular components, declarative experiment configs, CLI utilities for data download/cache building/training/analysis, logging + experiment tracking + HParam sweeps) to inspect candidate validity/coverage, projection maps, loss/metric behavior and other relevant signals during development.

The remainder of the paper covers related work, problem formulation, dataset and oracle computation, an implementation sketch for learning-based scoring, the evaluation protocol, and diagnostic findings, followed by limitations and future directions.

II. Related Work

A. Next-best-view planning

Early NBV systems optimize coverage or information-gain utilities, while learning-based methods largely fall into (i) continuous-action policies trained with reinforcement learning and (ii) discrete candidate-ranking approaches [1]. GenNBV learns a continuous 5-DoF free-space policy with PPO and coverage-gain rewards [5]. VIN-NBV instead samples candidate views and predicts Relative Reconstruction Improvement (RRI) via imitation learning on oracle labels [1].

B. Egocentric foundation models

EFM3D introduces a benchmark for egocentric 3D perception with two core tasks: 3D OBB detection and surface regression, and proposes EVL, which lifts multi-stream RGB + SLAM snippets into a local, gravity-aligned voxel grid using frozen 2D foundation features plus semi-dense point and free-space masks [2]. The grid is anchored to the last RGB pose (local 4 m cube; in the voxel frame the extent is $[-2, 2] \times [0, 4] \times [-2, 2]$) and processed by a 3D U-Net [6] before dense heads predict occupancy, centerness, box parameters, and class logits; post-processing yields OBB detections [2]. The release also adds ASE OBB annotations and GT meshes for the validation subset, as well as a small real-world Aria Everyday Objects (AEO) set to support sim-to-real evaluation [2]. We treat EVL as a frozen backbone and build a lightweight NBV head on top of its voxel features; the local extent motivates semi-dense projection cues for out-of-bounds candidates.

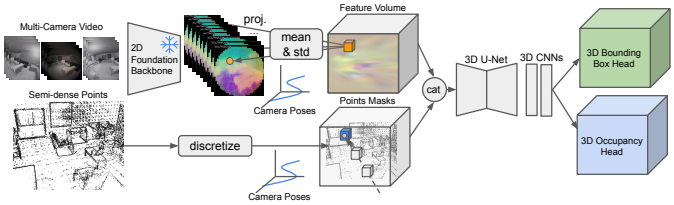


Fig. 1. EFM3D/EVL architecture overview (from the EFM3D release) [2].

C. Ordinal regression for continuous targets

Oracle RRI is a continuous target, but NBV ultimately requires a robust *ranking* of candidate views. Directly regressing RRI is challenging and can hurt generalization; heavily skewed RRI distributions with large outliers and stage-dependent scaling make absolute-value regression brittle [1]. A common remedy is to discretize RRI into K ordered bins and pose prediction as *ordinal* classification: the label $y \in \{0, ..., K-1\}$ has a natural order, and misclassifying a candidate by many bins should be penalized more than confusing nearby bins. Unlike nominal K -way classification, this setting can exploit label ordering instead of treating bins as unrelated categories [7].

Among ordinal losses, CORAL is attractive because it is *rank-consistent* and efficient [7]. It converts the K-class ordinal problem into $K-1$ binary threshold tasks (predicting whether y exceeds each rank) with shared classifier weights, which avoids contradictory non-monotone outputs that can arise from independent one-vs-rest reductions. CORAL therefore yields well-structured cumulative probabilities that can be mapped back to a scalar score (e.g., via an expected bin value) for candidate ranking while reducing large misclassifications.

D. Feature-wise conditioning (FiLM)

Feature-wise Linear Modulation (FiLM) applies a learned per-channel scale and shift to condition intermediate features on an auxiliary signal [8]. We use FiLM-style conditioning to modulate voxel-derived candidate features with view-dependent semi-dense projection statistics, providing a lightweight mechanism for late fusion when reliability of the modulated signal varies heavily across candidates.

III. Problem Formulation

We consider an egocentric reconstruction episode with a sequence of captured frames and poses. Let $\mathcal{P}_t$ be the current reconstruction point set at step t, and let $\mathcal{M}_{\mathrm{GT}}$ denote the ground-truth surface mesh for the scene. At each step we sample a finite set of N_q candidate camera poses $\boldsymbol{q} \in \mathcal{Q} \subset \mathrm{SE}(3)$ (with optional roll constraints), render a candidate point set $\mathcal{P}_q$ by rasterized depth-rendering $\mathcal{M}_{\mathrm{GT}}$ from pose $\boldsymbol{q}$, converting pixel centers to PyTorch3D's NDC screen coordinates for unprojection, and score candidates by their expected improvement in reconstruction quality. In practice, candidates are sampled within a constrained shell (radius/elevation/azimuth), then pruned by collision and free-space checks.

Our work focuses on two pieces: (i) computing these oracle per-candidate scores (continuous RRI values and their corresponding ordinal bin labels), and (ii) training a lightweight candidate scorer (VIN v3) that predicts the ordinal labels from egocentric observations and the candidate pose. The resulting model implements a one-step ranking policy (select the candidate with the highest predicted expected RRI). Learning a multi-step NBV policy and/or continuous action policy on top of this scorer remains future work.

A. Chamfer distance and RRI

We measure reconstruction quality using a Chamfer-style point $\leftrightarrow$ mesh distance between a point set $\mathcal{P}$ and a mesh surface $\mathcal{M}_{\mathrm{GT}}$. We represent $\mathcal{M}_{\mathrm{GT}}$ by its triangular faces $\mathcal{F}_{\mathrm{GT}}$ and evaluate both directional terms using squared point-to-triangle and triangle-to-point distances.

$$\mathrm{CD}(\mathcal{P}, \mathcal{M}_{\mathrm{GT}}) = \mathcal{A}(\mathcal{P}, \mathcal{M}_{\mathrm{GT}}) + \mathcal{C}(\mathcal{P}, \mathcal{M}_{\mathrm{GT}}) \tag{1}$$

$$\mathcal{A}(\mathcal{P}, \mathcal{M}_{\mathrm{GT}}) = \frac{1}{\|\mathcal{P}\|} \sum_{\boldsymbol{p} \in \mathcal{P}} \min_{\boldsymbol{f} \in \mathcal{F}_{\mathrm{GT}}} d(\boldsymbol{p}, \boldsymbol{f})^2 \tag{2}$$

$$\mathcal{C}(\mathcal{P}, \mathcal{M}_{\mathrm{GT}}) = \frac{1}{\|\mathcal{F}_{\mathrm{GT}}\|} \sum_{\boldsymbol{f} \in \mathcal{F}_{\mathrm{GT}}} \min_{\boldsymbol{p} \in \mathcal{P}} d(\boldsymbol{p}, \boldsymbol{f})^2 \tag{3}$$

The Relative Reconstruction Improvement for candidate $\boldsymbol{q}$ is then

$$\mathrm{RRI}(q) = \frac{\mathrm{CD}(\mathcal{P}_t, \mathcal{M}_{\mathrm{GT}}) - \mathrm{CD}\big(\mathcal{P}_t \cup \mathcal{P}_q, \mathcal{M}_{\mathrm{GT}}\big)}{\mathrm{CD}(\mathcal{P}_t, \mathcal{M}_{\mathrm{GT}}) + \varepsilon} \tag{4}$$

Here ε is a small stabilizer. A positive RRI means that adding the candidate view decreases the Chamfer distance, thereby improving reconstruction quality. A greedy one-step oracle policy would select

$$q_\star = \operatorname*{argmax}_{q \in \mathcal{Q}} \mathrm{RRI}(q) \tag{5}$$

This one-step selection rule is inherently myopic: it optimizes immediate surface error reduction and ignores longer planning horizons. As a simple thought experiment, consider a corridor with a doorway into an unseen room. A candidate view that moves toward the doorway might yield little immediate RRI (it still sees mostly the corridor), yet it enables a subsequent view inside the room with a large gain. A greedy one-step rule can therefore prefer refinements of already-observed surfaces over actions that open new regions. Addressing this requires explicit lookahead or learning a multi-step policy, which is outside the scope of this paper.

B. Ordinal binning

Direct regression on RRI is sensitive to outliers and stage-dependent scaling (early stages often yield larger gains). Following VIN-NBV, we discretize RRI into K ordered bins and solve an ordinal classification problem [1]. The continuous prediction is recovered by taking the expectation over the estimated ordinal distribution.

In our implementation, we fit empirical **quantile edges** (equal-mass bins) and assign the ordinal label by edge counting (Section Section VI):

$$e_k = \mathrm{Quantile}\left(\{r_i\}_{i=1}^{N}, \frac{k}{K}\right), \quad k \in \{1, ..., K-1\} \tag{6}$$

$$y(r) = \sum_{k=1}^{K-1} \mathbb{1}[r > e_k], \quad y(r) \in \{0, ..., K-1\} \tag{7}$$

IV. Dataset and Inputs

We use Aria Synthetic Environments (ASE), a large-scale synthetic dataset of 100,000 procedurally generated indoor scenes with realistic Aria sensor simulations [3], [9].

A. ASE: synthetic egocentric trajectories

ASE scenes are recorded as single egocentric walk-through trajectories with synchronized RGB + SLAM camera streams, inertial signals, and SLAM products (trajectory and semi-dense points) in a gravity-aligned world frame [4]. The native dataset release also includes dense per-frame supervision (e.g., depth maps, instance masks, and scene language), which we primarily use for visualization and qualitative debugging.

B. ATEK-EFM variant: WebDataset snippets (EFM3D / EVL format)

We consume ASE through the ATEK Data Store WebDataset export in the EFM3D/EVL training format [2], [10]. In this representation, each sample is a fixed-length snippet of 20 frames at 10 Hz (2 s), with calibration and rig poses attached per frame, and a stride of 10 frames (1 s) between consecutive snippets. The standard preprocessing resizes RGB frames to 240 × 240 and SLAM grayscale frames to roughly 320 × 240 while preserving geometry

(intrinsics/extrinsics) and coordinate conventions (LUF camera frames; gravity-aligned world frame for rig/voxel quantities).

An example snippet with synchronized RGB/SLAM frames is shown in Fig. 2.

Fig. 2. ASE snippet overview: synchronized RGB/SLAM frames.

C. Modalities and supervision

The key modalities for Aria-VIN-NBV are:

- RGB and SLAM camera streams with per-frame intrinsics and extrinsics.
- Optional per-frame ray distance / depth supervision in the EFM schema.
- Rig trajectory and gravity, providing world-frame poses for all streams (trajectory notation: $\boldsymbol{T}^{\mathrm{w}}_{\mathrm{rig}}(t)$).
- Semi-dense SLAM points ($\boldsymbol{\mathcal{P}}_t$) with per-point inverse-distance uncertainty (σ_ρ) and optional observation counts (n_{obs}). Points are provided per snippet frame and are padded/clipped to a fixed budget (50k) for batching; we collapse them across the snippet by concatenating valid points, deduplicating when observation counts are requested, and (optionally) subsampling to a fixed maximum for projection features. Observation counts (track length) are appended when enabled and serve as a reliability cue.
- Ground-truth meshes for a subset of scenes, used to compute Chamfer-based reconstruction quality.
- Ground-truth 3D boxes and semantic categories (used by the entity-aware extension; see Section XIII).

Any mesh preprocessing that is specific to oracle labeling (e.g., optional snippet-bound cropping, simplification, and caching) is described in the oracle section Section VI to avoid repeating implementation details here.

D. Ground-truth mesh subset

The public ASE release includes GT meshes for a subset of scenes. In the EFM3D/ASE release, 100 scenes include GT meshes that we use for oracle RRI labels [2]. The corresponding ATEK-EFM export provides 4,608 snippet windows. For efficient iteration, we build an offline dataset of oracle labels for 883 snippets from 80 of the 100 mesh scenes and use an 80/20 train/val split (706/177).

E. Optional visibility metadata

ASE also provides per-point observation metadata (e.g., visibility tables for semi-dense points). We treat these signals as optional accelerators or analysis tools and do not rely on them for the oracle label computation described in this paper.

V. Coordinate Conventions and Geometry

We follow the EFM3D/ATEK coordinate conventions throughout the pipeline. The world frame is gravity-aligned, the rig frame moves with the headset, and each camera frame is expressed in left-up-forward (LUF) coordinates. All poses are represented as SE(3) transforms, and cameras are represented by calibrated camera objects that bundle intrinsics and extrinsics.

LUF is defined as $+x$ left, $+y$ up, and $+z$ forward (along the optical axis). Image-plane pixel coordinates follow the standard convention: origin at the top-left corner, u increasing to the right, and v increasing downward.

For the fixed CW90 rig-basis correction (`rotate_yaw_cw90`) used in candidate generation and VIN inputs, see Appendix Section XVI.

A. Transformation notation

We write $\boldsymbol{T}^{\mathrm{A}}_{\mathrm{B}}$ for the transform from frame B to frame A. Such transforms lie in SE(3); composing poses follows standard multiplication, and inversion yields the reverse transform. For example, a world point $\boldsymbol{x}$ can be expressed in a (candidate) camera frame via

$$\boldsymbol{x}^{c_q} = \boldsymbol{T}^{c_q}_{w} \boldsymbol{x}^{w} \tag{8}$$

These conventions are critical when projecting semi-dense points into candidate views and when mapping EVL voxel coordinates back to the rig frame.

B. EVL voxel grid contract

EVL represents the scene in a local, fixed-size voxel grid aligned to a gravity-aware frame. The grid is specified by a rigid transform between voxel and world coordinates and a metric extent (grid bounds in metres). Candidates can fall partially or fully outside this extent; we therefore track per-candidate validity signals so the NBV head can down-weight unreliable voxel context.

C. Camera model

EFM3D represents Aria cameras as batched `CameraTW` calibration objects (fisheye model parameters plus per-frame extrinsics) and provides utilities such as valid-radius masking for diagnostics [2]. For oracle rendering, backprojection, and view-conditioned projections, we convert these calibrations to a PyTorch3D `PerspectiveCameras` instance so that rasterization and unprojection share a single camera convention [11].

We use PyTorch3D's `PerspectiveCameras` for the oracle renderer because it enables fast batched mesh rasterization on GPU and provides a consistent projection/unprojection interface that we reuse for view-conditioned features [11]. Concretely, we convert Aria calibrations to a batched PyTorch3D camera instance and render metric depth maps with `in_ndc=false`. Using the same camera model for (i) depth rendering, (ii) backprojection, and

(iii) screen-space projection of semi-dense points prevents convention bugs (pixel vs NDC, principal point ordering, and axis directions) and keeps the oracle labels and VIN features geometrically consistent.

VI. Oracle RRI Computation

Oracle RRI labels are computed offline by rendering candidate depth maps from ground-truth meshes and fusing these points with the current reconstruction. The pipeline is implemented with PyTorch3D rasterization and EFM3D utilities for unprojection, point fusion, and Chamfer evaluation.

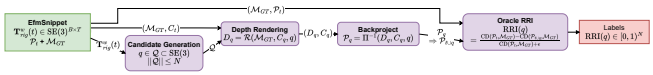


Fig. 3. Oracle RRI pipeline (offline labeling).

A. Inputs and outputs

The oracle labeler consumes one ASE-EFM snippet (see Section IV) and produces per-candidate oracle scores and auxiliary diagnostics. Concretely, the input provides:

- the trajectory $\boldsymbol{T}_{\text{rig}}^{\text{w}}(t)$ and final reference pose $\boldsymbol{T}_{\text{rig}}^{\text{w}}(T)$,
- the semi-dense reconstruction point set $\boldsymbol{\mathcal{P}}_t$ derived from $\boldsymbol{\mathcal{P}}_t$ as described in Section IV,
- the ground-truth mesh surface $\boldsymbol{\mathcal{M}}_{\text{GT}}$ (triangles $\boldsymbol{\mathcal{F}}_{\text{GT}}$) for the mesh subset described in Section IV,
- camera intrinsics/extrinsics (used to build candidate cameras).

The output is a set of candidate poses $\boldsymbol{\mathcal{Q}}$ and oracle labels (`RRI`, plus accuracy/completeness components), optionally followed by ordinal binning for CORAL training.

For efficiency, we preprocess the GT mesh $\boldsymbol{\mathcal{M}}_{\text{GT}}$ at most once per snippet: we optionally crop it to snippet bounds and simplify it via quadric decimation, then cache the processed mesh for reuse across collision checks and depth rendering (see `mesh_cache.py`).

The end-to-end oracle labeler is implemented in `oracle_rri_labeler.py`.

B. Candidate generation

Given the reference rig pose $\boldsymbol{T}_r^w$ at the final trajectory time step, we sample N_q candidate camera poses in a constrained shell around the reference and prune invalid proposals.

Setting	Value
N_q	60
$r_{\min}$	0.5
$r_{\max}$	1.8
$\theta_{\min}$	-20
$\theta_{\max}$	25
ψ_{span}	170
ψ_δ	60
θ_δ	30
φ_δ	0
`align_to_gravity`	`true`
`min_dist_to_mesh`	0.2

Table 1. Key parameters (candidate generation).

a) *Candidate center sampling (position sampling):* Candidate centers are sampled by drawing a direction $\boldsymbol{s}_q$ on the unit sphere $\mathbb{S}^2 \subset \mathbb{R}^3$ and a radius $r_q \in (r_{\min}, r_{\max})$, then rescaling the direction into (ψ, θ) caps **without rejection** (see `positional_sampling.py`).

Here $\boldsymbol{s}_q \in \mathbb{S}^2$ is a unit direction parameterized in spherical coordinates by azimuth ψ and elevation θ.

$$\boldsymbol{s}_q \sim \mathcal{U}(\mathbb{S}^2), \quad r_q \sim \mathcal{U}(r_{\min}, r_{\max}) \tag{9}$$

We draw directions either uniformly, or from a forward-biased Power Spherical distribution centered at the device forward axis $\boldsymbol{s}_q \sim \text{PS}(\boldsymbol{\mu}, \kappa)$ [12].

Gravity-aligned sampling uses $\boldsymbol{T}_s^w$ (roll/pitch removed). $\boldsymbol{T}_r^w$ is the reference pose rotation; the angles (ψ, θ) only parameterize the direction $\boldsymbol{s}_q \in \mathbb{S}^2$:

$$\begin{gathered}(\psi, \theta) = \text{angles}(\boldsymbol{s}_q) \\ (\psi, \theta) \leftarrow \text{lin}([\psi_{\min}, \psi_{\max}] \times [\theta_{\min}, \theta_{\max}]; (\psi, \theta))\end{gathered} \tag{10}$$

$$\boldsymbol{s}_{q'} = (\cos\theta \sin\psi, \sin\theta, \cos\theta \cos\psi) \tag{11}$$

$$\boldsymbol{c}_q = \boldsymbol{T}_r^w(r_q \boldsymbol{s}_{q'}) \tag{12}$$

We use standard spherical coordinate relations for the angle/vector conversions [13].

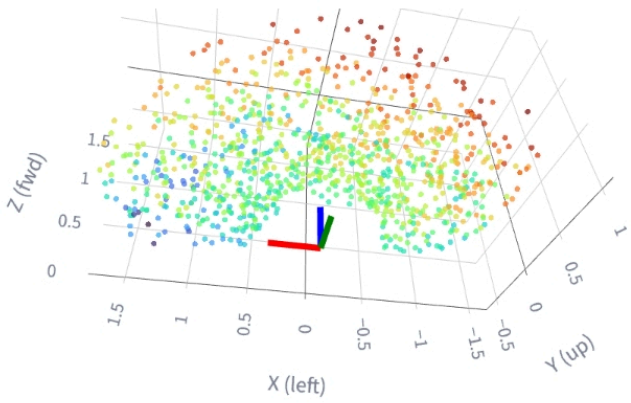


Fig. 4. Candidate centers in the reference frame.

b) *Candidate orientations (view directions):*
For each sampled center $\boldsymbol{c}_q$, we construct a base camera orientation according to the configured `ViewDirectionMode` (`orientations.py`). The most common mode is radial "look-away", which points the camera optical axis away from the reference rig translation.

$$\begin{aligned} \boldsymbol{R}_{\text{base}} &= \text{look-away}\big(\boldsymbol{c}_q, \boldsymbol{T}^{\text{w}}_{\text{rig}}(T)\big) \\ \boldsymbol{T}^{w}_{c_q} &= \big(\boldsymbol{R}_{\text{base}} \circ \boldsymbol{R}_{\text{delta}}, \boldsymbol{c}_q\big) \end{aligned} \tag{13}$$

The (ψ, θ) caps are applied to the **jitter delta** (box-uniform), not the base view. See the next subsection for the definition of $\boldsymbol{R}_{\text{delta}}$.

Radial look-away. Let $\boldsymbol{u}_{\text{wup}}$ be the world up vector and $\boldsymbol{t}_{\text{ref}}$ the reference translation (from $\boldsymbol{T}^{\text{w}}_{\text{rig}}(T)$). Define the forward axis

$$\boldsymbol{z}_q = \frac{\boldsymbol{c}_q - \boldsymbol{t}_{\text{ref}}}{\| \boldsymbol{c}_q - \boldsymbol{t}_{\text{ref}} \|_2 + \varepsilon} \tag{14}$$

and construct a roll-stable orthonormal basis

$$\boldsymbol{y}_q = \frac{\boldsymbol{u}_{\text{wup}} - (\boldsymbol{u}_{\text{wup}} \cdot \boldsymbol{z}_q)\boldsymbol{z}_q}{\| \boldsymbol{u}_{\text{wup}} - (\boldsymbol{u}_{\text{wup}} \cdot \boldsymbol{z}_q)\boldsymbol{z}_q \|_2 + \varepsilon}, \quad \boldsymbol{x}_q = \boldsymbol{y}_q \times \boldsymbol{z}_q \tag{15}$$

The resulting rotation is $\boldsymbol{R}_{\text{base}} = [\boldsymbol{x}_q, \boldsymbol{y}_q, \boldsymbol{z}_q]$, which avoids pathological roll behavior for near-vertical directions.

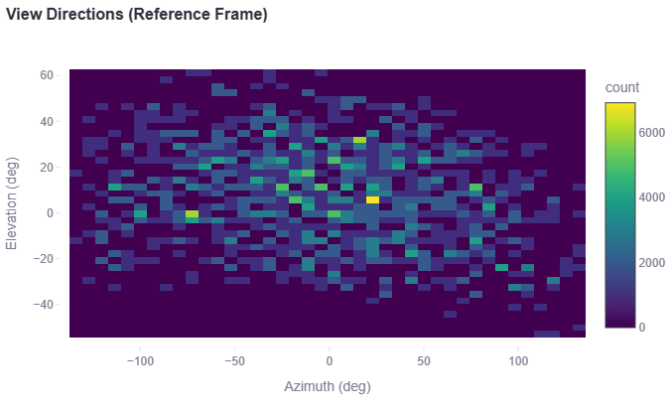


Fig. 5. View direction density (azimuth/elevation).

c) *Candidate orientations (view jitter):*
After base pose construction, we optionally apply bounded yaw/pitch/roll perturbations. Box-uniform caps apply to yaw/pitch, with optional roll:

$$\begin{aligned} \psi &\sim \mathcal{U}\left(-\frac{\psi_\delta}{2}, \frac{\psi_\delta}{2}\right) \\ \theta &\sim \mathcal{U}\left(-\frac{\theta_\delta}{2}, \frac{\theta_\delta}{2}\right) \\ \varphi &\sim \mathcal{U}(-\varphi_\delta, \varphi_\delta) \end{aligned} \tag{16}$$

$$\begin{aligned} \boldsymbol{R}_{\text{delta}} &= \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\varphi) \\ \boldsymbol{T}^{w}_{c_q} &= \boldsymbol{T}^{w}_{c_q} \circ \begin{pmatrix} \boldsymbol{R}_{\text{delta}} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{pmatrix} \end{aligned} \tag{17}$$

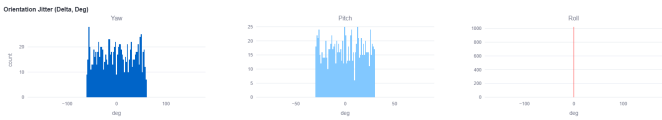


Fig. 6. Orientation jitter distribution (delta yaw/pitch/roll, deg).

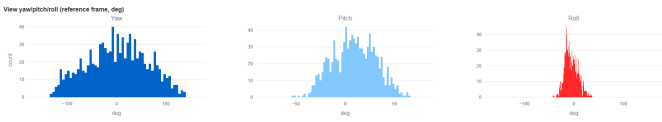


Fig. 7. Reference-frame yaw/pitch/roll distribution after look-away + jitter (deg).

Fig. 6 verifies that the sampled view jitter stays within the configured caps. Fig. 7 shows the resulting yaw, pitch, and roll distribution in the reference frame after composing `look-away` with the jitter rotation.

d) *Candidate pruning rules:*
Candidates are oversampled and then filtered by modular rule objects (`candidate_generation_rules.py`). Let $\mathcal{M}_{\text{GT}}$ be the GT mesh and $\mathcal{B}$ be the snippet occupancy AABB.

Parameter	Value
min_distance_to_mesh	0.2 m
ensure_collision_free	true
collision_backend	pytorch3d
ray_subsample	32
step_clearance	0.1 m
ensure_free_space	true

Table 2. Candidate pruning configuration (effective).

Mesh clearance. Reject candidates too close to the mesh:

$$d_i = \min_{\boldsymbol{m} \in \mathcal{M}_{\mathrm{GT}}} \| \boldsymbol{c}_i - \boldsymbol{m} \|_2, \quad d_i > d_{\min} \tag{18}$$

Path collision. Reject candidates whose straight segment from the reference to the candidate center intersects the mesh. In the discretized variant, we sample points along the segment and require a minimum clearance:

$$\begin{gathered} \boldsymbol{s}_{i,k} = \boldsymbol{t}_{\mathrm{ref}} + \alpha_k (\boldsymbol{c}_i - \boldsymbol{t}_{\mathrm{ref}}), \quad \alpha_k \in [0,1], \\ \min_{\boldsymbol{m} \in \mathcal{M}_{\mathrm{GT}}} \| \boldsymbol{s}_{i,k} - \boldsymbol{m} \|_2 > \delta \end{gathered} \tag{19}$$

Free space. Reject candidates outside the occupancy bounds: $(\boldsymbol{c})_i \in \mathcal{B}$.

C. Candidate depth rendering

For each candidate pose q, we render a depth map from the GT mesh using a metric z-buffer (PyTorch3D rasterizer with `in_ndc=false`). We backproject valid depth pixels into world coordinates to obtain $\mathcal{P}_q$. We treat hits as valid if they correspond to a rasterized face and the resulting depth lies inside configured bounds ($z_{\mathrm{near}} < z < z_{\mathrm{far}}$); invalid/missing pixels are masked out and do not contribute to the fused point cloud. This ensures that the oracle computation respects the same geometric visibility constraints as the candidate camera.

Depth rendering is implemented in `pytorch3d_depth_renderer.py` using PyTorch3D's `PerspectiveCameras`, `RasterizationSettings`, and `MeshRasterizer` [11]. We use hard z-buffering with `faces_per_pixel=1` and `blur_radius=0`, clip triangles closer than `znear`, and expose performance knobs such as `bin_size` via the renderer config.

Parameter	Value
max_candidates_final	60
resolution_scale	0.5
znear	0.001 m
zfar	20 m
cull_backfaces	true
blur_radius	0
bin_size	0
dtype	float32

Table 3. Depth rendering configuration (effective).

Pose conventions. Our poses are stored as world ← camera transforms. PyTorch3D expects a world → view mapping using row vectors: $\boldsymbol{x}_{\mathrm{cam}} = \boldsymbol{x}_{\mathrm{world}} \boldsymbol{R} + \boldsymbol{T}$. We therefore invert the pose and transpose the rotation before passing it to PyTorch3D (see in-code comments in the renderer).

The rasterizer outputs a z-buffer $\boldsymbol{D}_q$ and a face-index buffer `pix_to_face`. Valid pixels satisfy `pix_to_face >= 0` and are further clipped to `znear < depth < zfar`.

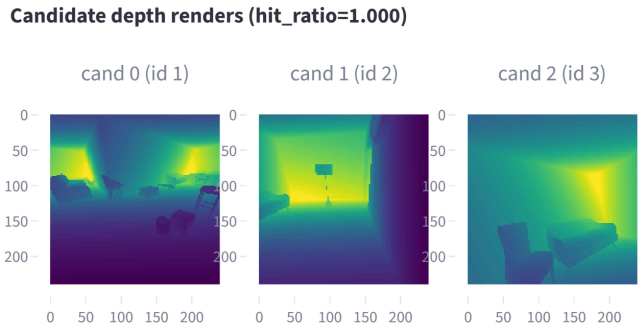


Fig. 8. Candidate depth renders (1x3).

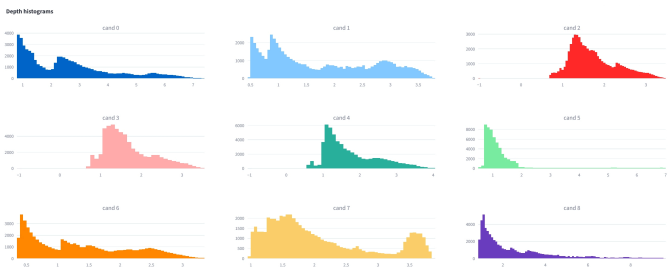


Fig. 9. Depth histograms across candidates (3x3).

The depth renders in Fig. 8 are used to sanity-check geometric conventions (camera poses, handedness, and near/far clipping), while the histograms in Fig. 9 reveal depth discontinuities and spurious spikes that can arise from misconfigured rasterization or invalid triangles.

D. Backprojection and NDC alignment

We convert each depth image to a point cloud via `backproject_depths_p3d_batch` in `unproject.py`. For pixel centers $\left(u+\frac{1}{2}, v+\frac{1}{2}\right)$ we first map to PyTorch3D's **normalized device coordinates** (NDC) using the convention (+X left, +Y up) and the scale $s = \min(H, W)$ [11]:

$$\begin{aligned} x_{\text{ndc}} &= -\left(u+\frac{1}{2}-\frac{W'}{2}\right)\left(\frac{2}{s}\right), \\ y_{\text{ndc}} &= -\left(v+\frac{1}{2}-\frac{H'}{2}\right)\left(\frac{2}{s}\right) \end{aligned} \tag{20}$$

We then unproject in NDC space:

$$\boldsymbol{p}_{\text{world}} = \Pi^{-1}(x_{\text{ndc}}, y_{\text{ndc}}, d_q, \boldsymbol{C}_q), \quad d_q = \boldsymbol{D}_q(u, v) \tag{21}$$

Why NDC? With `PerspectiveCameras(in_ndc=false)` and non-square images, PyTorch3D internally uses an NDC-like normalization tied to $\min(H, W)$. Backprojecting from pixel coordinates (`from_ndc=false`) yields points in a different screen convention than the rasterizer, which empirically leads to backprojected points that do not lie on the rendered mesh. Converting pixels to the same NDC convention used by the rasterizer makes depth unprojection and rasterization consistent.

The per-candidate point set is then $\left(\mathcal{P}_{q_i} = \{\boldsymbol{p}_{\text{world}} : (u, v) \text{ valid}\}\right)$, optionally subsampled by a stride to control point count.

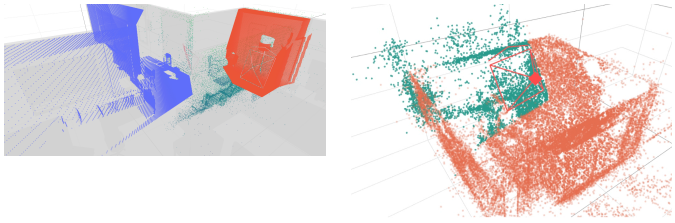


Fig. 10. Oracle fusion diagnostics: backprojected candidate points align with the GT mesh (left) and candidate visibility over the semi-dense reconstruction (right).

E. Oracle RRI computation

We measure reconstruction quality using a Chamfer-style point $\leftrightarrow$ mesh distance between a point set $\mathcal{P}$ and a mesh surface $\mathcal{M}_{\text{GT}}$ (triangles $\mathcal{F}_{\text{GT}}$). We evaluate both directional terms using squared point-to-triangle and triangle-to-point distances:

$$\text{CD}(\mathcal{P}, \mathcal{M}_{\text{GT}}) = \mathcal{A}(\mathcal{P}, \mathcal{M}_{\text{GT}}) + \mathcal{C}(\mathcal{P}, \mathcal{M}_{\text{GT}}) \tag{22}$$

$$\mathcal{A}(\mathcal{P}, \mathcal{M}_{\text{GT}}) = \frac{1}{\|\mathcal{P}\|} \sum_{\boldsymbol{p}\in\mathcal{P}} \min_{\boldsymbol{f}\in\mathcal{F}_{\text{GT}}} d(\boldsymbol{p}, \boldsymbol{f})^2 \tag{23}$$

$$\mathcal{C}(\mathcal{P}, \mathcal{M}_{\text{GT}}) = \frac{1}{\|\mathcal{F}_{\text{GT}}\|} \sum_{\boldsymbol{f}\in\mathcal{F}_{\text{GT}}} \min_{\boldsymbol{p}\in\mathcal{P}} d(\boldsymbol{p}, \boldsymbol{f})^2 \tag{24}$$

For each candidate, we fuse the semi-dense reconstruction with the candidate point cloud:

$$\mathcal{P}_{t\cup q} = \mathcal{P}_t \cup \mathcal{P}_q \tag{25}$$

The Relative Reconstruction Improvement for candidate $\boldsymbol{q}$ is then

$$\text{RRI}(q) = \frac{\text{CD}(\mathcal{P}_t, \mathcal{M}_{\text{GT}}) - \text{CD}(\mathcal{P}_t \cup \mathcal{P}_q, \mathcal{M}_{\text{GT}})}{\text{CD}(\mathcal{P}_t, \mathcal{M}_{\text{GT}}) + \varepsilon} \tag{26}$$

Here ε is a small stabilizer. A positive RRI means that adding the candidate view decreases the Chamfer distance, thereby improving reconstruction quality.

In practice, candidate scoring is batched on GPU: the ground-truth mesh is cropped to an occupancy-aligned bounding box shared across candidates and the point$\leftrightarrow$mesh distances are evaluated for all candidates in a single forward pass whenever memory permits. A crop that contains no mesh faces is treated as an invalid oracle input rather than falling back to full-scene scoring.

Conceptually, the **accuracy** term $\mathcal{A}$ measures how well the point set $\mathcal{P}$ lies on the GT surface: it averages the squared distance from each point $\boldsymbol{p} \in \mathcal{P}$ to its nearest mesh triangle $\boldsymbol{f} \in \mathcal{F}_{\text{GT}}$. Thus, $\mathcal{A}$ penalizes noisy or misregistered points that do not agree with the mesh.

The **completeness** term $\mathcal{C}$ measures how much of the GT surface is **covered** by the points: it averages the squared distance from each triangle $\boldsymbol{f} \in \mathcal{F}_{\text{GT}}$ to its nearest point $\boldsymbol{p} \in \mathcal{P}$. Thus, $\mathcal{C}$ penalizes missing surface coverage (holes / unobserved regions).

We report the Chamfer-style distance as the sum $\text{CD}(\mathcal{P}, \mathcal{M}_{\text{GT}}) = \mathcal{A} + \mathcal{C}$ and use an ε stabilizer in the RRI denominator.

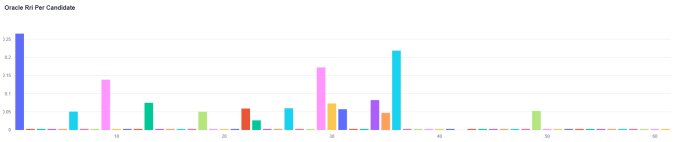


Fig. 11. Oracle RRI values across candidates for a representative snippet (skewed distribution).

Inspecting this single-snippet candidate RRI bar plot reaveals a highly skewed distribution with many extremely small values and a few strong candidates. This motivates our choice of quantile-based binning for CORAL (next section).

F. Ordinal binning for CORAL (label post-processing)

To train with CORAL, we discretize continuous RRIs into K ordered bins using empirical quantile edges (`rri_binning.py`). Given a stream of oracle RRIs $\{r_n\}_{\{n=1\}}^{N}$, we compute $K-1$ edges at the quantiles $\frac{k}{K}$:

$$e_k = \text{Quantile}\left(\{r_i\}_{i=1}^{N}, \frac{k}{K}\right), \quad k \in \{1, ..., K-1\} \tag{27}$$

The ordinal label is then the number of edges below the value (implemented via `torch.bucketize`):

$$y(r) = \sum_{k=1}^{K-1} \mathbb{1}[r > e_k], \quad y(r) \in \{0, ..., K-1\} \tag{28}$$

CORAL represents the label y as binary level targets [7]:

$$t_k = \mathbb{1}[y > k], \quad k \in \{0, ..., K-2\} \tag{29}$$

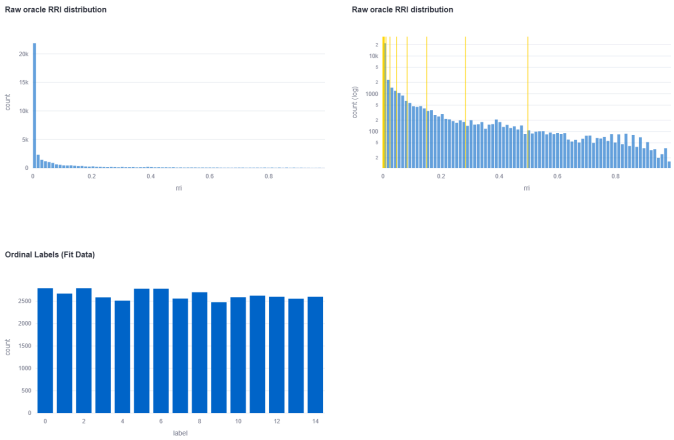


Fig. 12. Oracle RRI distribution and fitted quantile binning (fit data). The skewed distribution motivates discretization; quantile edges yield approximately uniform ordinal class counts.

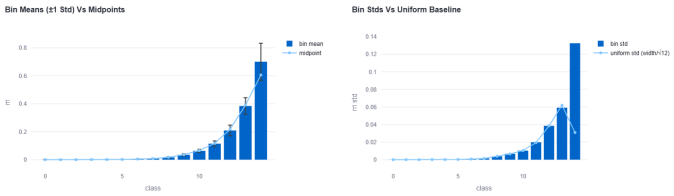


Fig. 13. Per-bin statistics for ordinal bins (fit data). Quantile bins have uneven widths; we initialize bin representatives from bin means and monitor calibration and variance across bins.

VII. Aria-VIN-NBV Architecture

This section describes the currently implemented VINv3 candidate scorer that we train against oracle RRI labels. The design emphasizes explicit *view-conditioned* evidence and a lightweight CORAL head, with EVL providing the voxel backbone [2].

A. Core design and optional ablations

To avoid ambiguity between what is implemented in VINv3 and what is exploratory, we separate the model into (i) a stable core that is always present and (ii) optional modules evaluated via ablations.

- **Core**: EVL voxel evidence, pose encoding, pose-conditioned global context, per-candidate semi-dense projection statistics (optionally a projection-grid CNN), and a final MLP + CORAL head.
- **View-conditioned evidence**: candidate-specific cues obtained by projecting both semi-dense points and pooled voxel centers into each candidate view.
- **Optional ablations**: trajectory-context encoder and alternative feature fusion mechanisms are not part of this section, even though they might have been part of our "baseline" model.

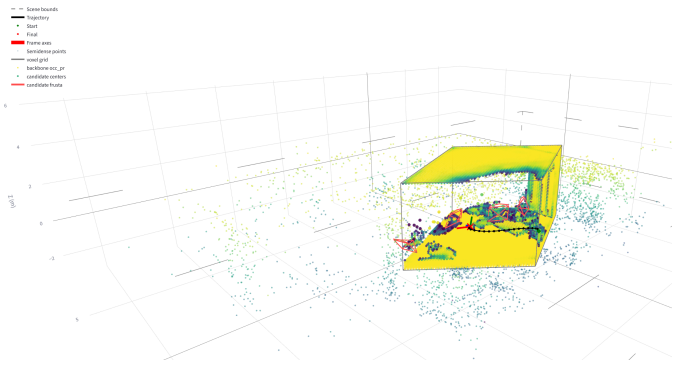


Fig. 14. Superposition of VINv3 inputs: EVL occupancy prior, candidate frusta, and semi-dense points.

We denote the core *per-candidate* features as the pose embedding $\boldsymbol{E}_q$, global context $\boldsymbol{g}_q$, and semidense projection evidence $\boldsymbol{s}_{\text{proj}_q}$ augmented by $\boldsymbol{s}_{\text{grid}_q}$, fused by an MLP + CORAL head (Section **Feature fusion and CORAL head**).

B. Backbone and scene field

EVL lifts multi-view observations into a local voxel grid centered near the latest rig pose and outputs multiple dense heads (occupancy probability, centerness, free-space evidence, and observation counts) [2]. We assemble an input voxel field $\boldsymbol{F}_v^{\text{in}} \in \mathbb{R}^{B\times F_{\text{in}}\times V\times V\times V}$ by concatenating selected EVL head channels plus derived cues. In the current VINv3 baseline, the default channel set is $(\boldsymbol{V}_{\text{occ}}^{\text{pr}}, \boldsymbol{V}_{\text{surf}}^{\text{in}}, \boldsymbol{V}_{\text{count}}^{\text{norm}}, \boldsymbol{V}_{\text{cent}}^{\text{pr}})$, with optional inclusion of $\boldsymbol{V}_{\text{free}}^{\text{in}}$, $\boldsymbol{V}_{\text{unk}}$, and $\boldsymbol{V}_{\text{new}}$ via configuration. In our EVL configuration the voxel grid uses $V = 48$ (roughly a 4 m cube), and the observation evidence channels $\boldsymbol{V}_{\text{surf}}^{\text{in}}$ and $\boldsymbol{V}_{\text{count}}^{\text{in}}$ are derived from the semi-dense SLAM points.

We normalize observation counts via a per-snippet log1p normalization:

$$\boldsymbol{V}_{\text{count}}^{\text{norm}} = \frac{\log(1 + \boldsymbol{V}_{\text{count}}^{\text{in}})}{\log(1 + \max(\boldsymbol{V}_{\text{count}}^{\text{in}}))} \tag{30}$$

where the maximum is taken over the voxel grid (per batch element). We then derive the unknown mask and a new-surface prior as

$$\boldsymbol{V}_{\text{unk}} = 1 - \boldsymbol{V}_{\text{count}}^{\text{norm}}, \quad \boldsymbol{V}_{\text{new}} = \boldsymbol{V}_{\text{unk}} \cdot \boldsymbol{V}_{\text{occ}}^{\text{pr}} \tag{31}$$

When EVL does not provide free-space evidence, we derive it as

$$\boldsymbol{V}_{\text{free}}^{\text{in}} = \boldsymbol{V}_{\text{obs}} \odot (1 - \boldsymbol{V}_{\text{surf}}^{\text{in}}) \tag{32}$$

(observed mask times empty occupancy). The final scene field is then $\boldsymbol{F}_v = \varphi(\text{Conv}_{1\times1\times1}(\boldsymbol{F}_v^{\text{in}})) \in \mathbb{R}^{B\times F_{\text{field}}\times V\times V\times V}$, implemented as `Conv3d + GroupNorm + GELU` [14]. The EVL output summary used to form the scene field is shown in Fig. 24, and representative input-channel slices are provided in Fig. 15.

C. Pose encoding

Each candidate pose is expressed in the reference rig frame via $\boldsymbol{T}_{c_q}^r = \boldsymbol{T}_r^{w-1} \cdot \boldsymbol{T}_{c_q}^w$. We encode translation and rotation using a 6D rotation representation [15] and

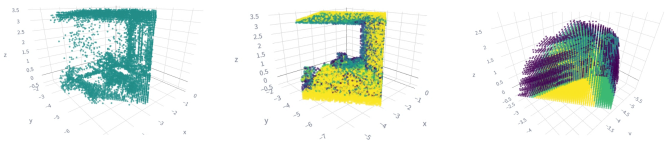


Fig. 15. Scene-field input channels: occ_input, occ_pr, and normalized observation counts.

Learnable Fourier Features (LFF) [16]. For each candidate i the pose encoder yields:

- a low-dimensional pose vector in $\mathbb{R}^9$ (translation + rotation-6D),
- an embedding $\boldsymbol{E}_{q_q} \in \mathbb{R}^{F_{\text{pose}}}$,
- the candidate center $\boldsymbol{c}_q \in \mathbb{R}^3$ in the rig frame.

Figure Fig. 16 summarizes the pose-encoding branch and its output contract.

D. Global context via pose-conditioned attention

We pool $\boldsymbol{F}_v$ onto a coarse grid of size $G_{\text{pool}} \times G_{\text{pool}} \times G_{\text{pool}}$, yielding G_{pool}^3 voxel tokens $\boldsymbol{t}$. For each voxel j, the voxel-center position $\boldsymbol{p}_j$ is used both to compute positional keys and to parameterize the voxel-projection branch. Each token is augmented with an LFF positional encoding of its voxel center expressed in the rig frame.

Candidate pose embeddings act as queries in a multi-head cross-attention block [17]. Let $\boldsymbol{t}_j$ denote the pooled voxel token feature at voxel center $\boldsymbol{p}_j$ (both defined in the reference rig frame). Cross-attention then produces candidate-conditioned mixtures of voxel tokens:

$$\boldsymbol{Q}_q = \boldsymbol{W}_Q \cdot \boldsymbol{E}_q,$$
$$\boldsymbol{K}_j = \boldsymbol{W}_K \cdot \boldsymbol{t}_j + \varphi(\boldsymbol{p}_j), \tag{33}$$
$$\boldsymbol{V}_j = \boldsymbol{W}_V \cdot \boldsymbol{t}_j$$
$$\boldsymbol{g}_q = \text{MHCA}(\boldsymbol{Q}_q, \boldsymbol{K}, \boldsymbol{V}) \tag{34}$$

The attention output yields a global context vector $\boldsymbol{g}_q \in \mathbb{R}^{F_g}$ for each candidate. We apply residual connections and an MLP block for stability.

Figure Fig. 17 illustrates how the pose-conditioned pooling and the voxel-projection FiLM modulation combine into the final global feature $\boldsymbol{g}_q$.

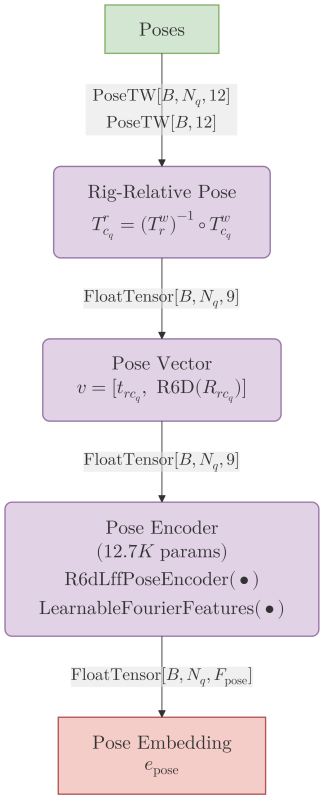


Fig. 16. Pose encoding branch: rig-relative candidate pose $\boldsymbol{T}^r_{c_q}$ mapped to an embedding $(\boldsymbol{E}_q)_q$.

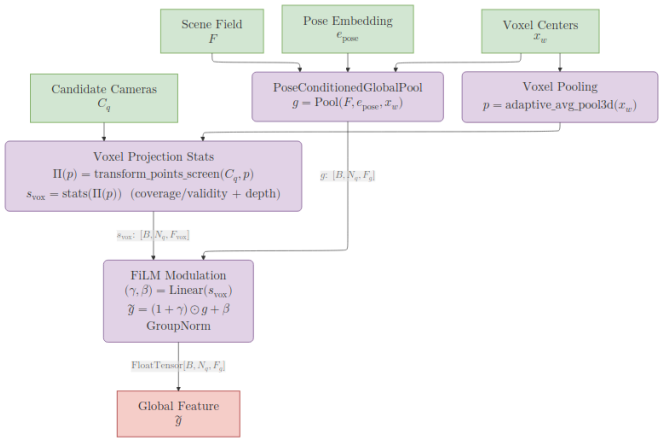


Fig. 17. Pose-conditioned global pooling with voxel-projection FiLM: pooled voxel tokens attend to pose queries, then $(\boldsymbol{g})_q$ is modulated by candidate-specific voxel projection statistics.

E. Coverage proxies and candidate validity

Candidate shells can place views outside EVL's local voxel extent. We therefore compute a per-candidate voxel-evidence fraction $(v)_q$ by sampling the normalized observation-count field $\boldsymbol{V}^{\text{norm}}_{\text{count}}$ at the candidate camera center and applying in-bounds and finiteness checks. Together with the semi-dense visibility fraction $(v^{\text{sem}})_q$, we define the conservative candidate-validity mask m as:

$$m_i = \mathbb{1}[\text{finite}] \cdot \mathbb{1}[v_i > 0] \cdot \mathbb{1}[v_i^{\text{sem}} > 0] \tag{35}$$

F. Voxel projection FiLM

To modulate global features with candidate-dependent evidence, we pool voxel centers to the same coarse grid used in global attention ($(G_{\text{pool}})^3$ points) via adaptive average pooling over the voxel-center grid (with symmetric center-cropping if the grid is padded), project those centers into each candidate view, and summarize their screen-space coverage and depth statistics. Because voxel centers have no per-point reliability metadata, the projection statistics use uniform weights. A linear FiLM head predicts per-channel $(\boldsymbol{\gamma}_q, \boldsymbol{\beta}_q)$ from these projection statistics and applies

$$\boldsymbol{g}_i^{\text{film}} = (1 + \boldsymbol{\gamma}_i) \cdot \boldsymbol{g}_i + \boldsymbol{\beta}_i \tag{36}$$

This provides lightweight view-conditioned modulation without introducing additional token-level attention [8].

G. Semi-dense projection statistics

To inject view-dependent evidence beyond the EVL voxel extent, we project the semi-dense SLAM points into each candidate view using the same screen-space camera model as the depth renderer. A projected point is valid if it is finite, in front of the camera ($z > 0$), and within the image bounds. For each candidate we compute:

- coverage ratio and empty fraction using a coarse $G_{\text{sem}} \times G_{\text{sem}}$ image grid (fraction of bins hit by at least one valid point),
- visibility fraction $(v^{\text{sem}})_q$ as the (optionally reliability-weighted) fraction of valid projections among finite projections,
- depth mean and depth standard deviation over valid points.

Reliability weights combine normalized observation count n_{obs} and inverse-distance uncertainty σ_ρ when available and clamp the normalized values to $[0, 1]$. The resulting projection statistics $\boldsymbol{s}_{\text{proj}}$ are concatenated to the head input. We subsample the padded semi-dense point cloud using the per-snippet `lengths` field to avoid including invalid padding in the projection.

Figure Fig. 18 summarizes the semidense projection-statistics branch, while Fig. 19 shows example diagnostic maps for three of the per-bin statistics.

Concretely, we treat $\boldsymbol{s}_{\text{proj}}{}_q$ as a compact scalar feature vector of the form

$$\left[\nu_{\text{cov}}, \nu_{\text{empty}}, v^{\text{sem}}, \mu_s, \sigma_\rho\right]_q \tag{37}$$

capturing coverage/emptiness, visibility, and depth moments.

We use the per-point validity mask

$$m_{i,j} = \mathbb{1}[\text{finite}] \cdot \mathbb{1}\left[z_{i,j} > 0\right] \cdot \mathbb{1}\left[0 \le u_{i,j} < W_i\right] \cdot \mathbb{1}\left[0 \le v_{i,j} < H_i\right] \tag{38}$$

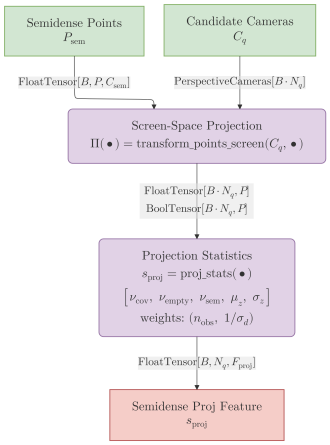


Fig. 18. Semidense projection statistics branch: project semidense points into each candidate camera and summarize coverage/visibility/depth moments into $(\boldsymbol{s}_{\text{proj}})_q$.

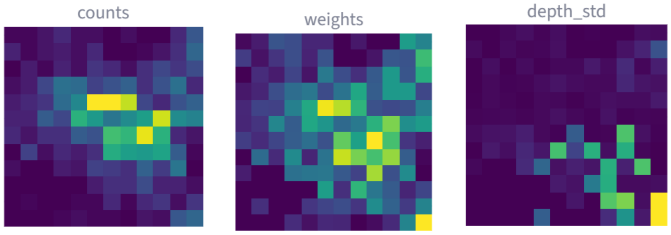


Fig. 19. Semi-dense projection statistics (example): per-bin counts, reliability weights, and depth standard deviation.

and define the (optionally reliability-weighted) visibility fraction $(v^{\text{sem}})_q = v_q^{\text{sem}}$ as the valid fraction among finite projections:

$$v_i^{\text{sem}} = \frac{\sum_j w_{i,j} m_{i,j}}{\sum_j w_{i,j} f_{i,j}} \tag{39}$$

Depth mean and depth standard deviation are computed over the valid set.

H. Semi-dense projection grid CNN

We also build a per-candidate $G_{\text{sem}} \times G_{\text{sem}}$ grid from the projected semi-dense points, storing occupancy, depth mean, and depth standard deviation per bin. A tiny 2D CNN encodes this grid into a compact feature vector that is appended to the head input. This adds local view-plane structure while remaining far lighter than a full image encoder.

The resulting grid is encoded into $(\boldsymbol{s}_{\text{grid}})_q$. In the current implementation, the grid CNN uses the hard validity mask but does not apply per-point reliability weights within bins.

In particular, we form a per-candidate grid tensor $\boldsymbol{H}_q \in \mathbb{R}^{3 \times G_{\text{sem}} \times G_{\text{sem}}}$ with channels $[O, \mu_z, \sigma_z]_q$ (occupancy and depth moments), and map it to $\boldsymbol{s}_{\text{grid}}{}_q$ via a tiny CNN.

Figure Fig. 20 summarizes the grid-CNN encoder.

I. Feature fusion and CORAL head

The final per-candidate feature vector concatenates multiple candidate-specific and snippet-level signals, e.g.

$$\boldsymbol{h} = \left[\boldsymbol{E}_q; \boldsymbol{g}; \boldsymbol{s}_{\text{proj}}; \boldsymbol{s}_{\text{grid}}\right] \tag{40}$$

The head MLP produces $K - 1$ logits per candidate. CORAL interprets these as cumulative probabilities and yields class probabilities and an expected ordinal score $\hat{r}$ (used as a ranking proxy) [7]. When we need a continuous RRI estimate, we combine the class probabilities with representative bin values (Section **Training Objective**).

Figure Fig. 21 summarizes the head computation and the `VinPrediction` outputs produced by our implementation.

This architecture preserves VIN-NBV's inductive bias (view-conditioned projection features) while leveraging EVL's egocentric voxel representation and providing explicit candidate conditioning throughout the pipeline.

Current VINv3 baseline (Jan 2026 run). The trajectory encoder is dis-

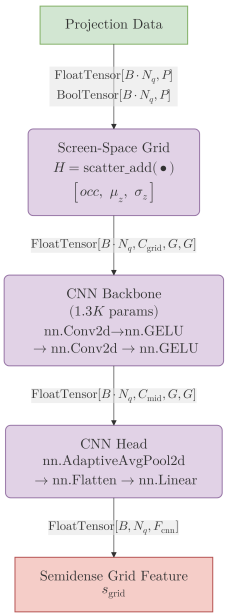


Fig. 20. Semi-dense projection grid CNN branch (tiny CNN on a coarse $G_{\text{sem}} \times G_{\text{sem}}$ grid).

abled (`use_traj_encoder=false`), the semi-dense projection CNN is enabled (`semidense_cnn_enabled=true`), and voxel-projection FiLM is enabled by default. The effective configuration used in the offline-cache training run is: field dim $F_{\text{field}} = 24$, global pool grid $G_{\text{pool}} = 5$, semidense projection grid $G_{\text{sem}} = 12$, max semidense points $P_{\text{max}} = 16384$, head

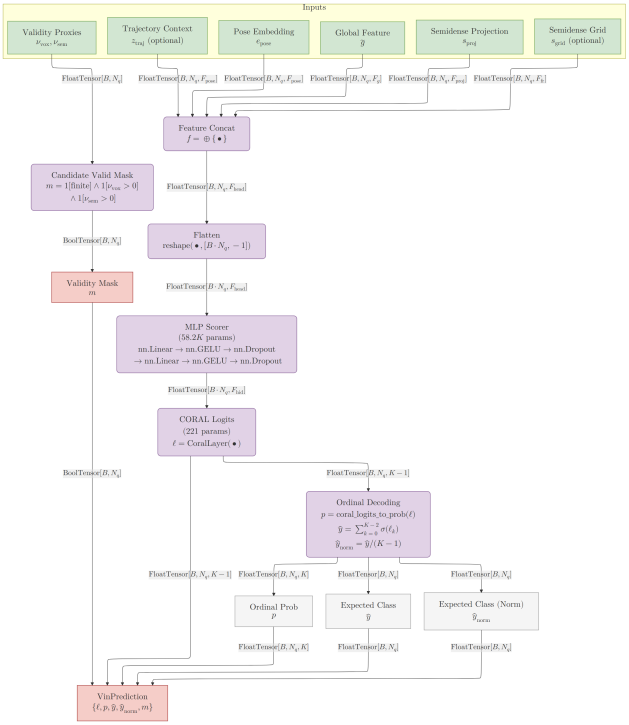


Fig. 21. Scoring head + CORAL decoding: concatenate per-candidate features and predict threshold logits for ordinal RRI labels.

hidden dim $F_{\text{hid}} = 192$ with 2 MLP layers, and 15 ordinal bins. The VINv3 head has 74,104 trainable parameters; the EVL backbone remains frozen and is excluded from this count.

VIII. Training Objective

To enable training a our candidate scorer on oracle RRI labels, we describe an ordinal regression objective. RRI values are binned into K ordered classes. CORAL models the cumulative probability that the class exceeds each threshold, yielding $K-1$ logits ℓ_k (collectively $\ell \in \mathbb{R}^{K-1}$) and cumulative probabilities $\boldsymbol{p} = \sigma(\ell)$ [7]. These are **cumulative** probabilities, not class marginals.

A. CORAL loss

Let r be the continuous RRI and y the ordinal bin index. The CORAL targets are binary levels $t_k = \mathbb{1}[y > k]$. The per-sample loss is

$$\mathcal{L}_{\text{coral}}(y, \boldsymbol{p}) = -\sum_{k=0}^{K-2} (t_k \log(p_k) + (1 - t_k) \log(1 - p_k)) \tag{41}$$

To recover a scalar prediction, we convert cumulative probabilities into marginal class probabilities:

$$\pi_k = p_{k-1} - p_k, \quad p_{-1} = 1, \quad p_{K-1} = 0 \tag{42}$$

The expected RRI is then

$$\hat{r} = \sum_{k=0}^{K-1} \pi_k \cdot u_k \tag{43}$$

where u_k is a representative value for bin k (initialized from bin means and optionally learned). This matches the CORAL paper's ordinal semantics and avoids treating cumulative probabilities as class posteriors.

When u_k is learned, we constrain it to be monotone (e.g., via a cumulative softplus parameterization) to preserve ordinal ordering.

We additionally monitor whether CORAL's cumulative probabilities are rank consistent. Since $p_k = P(y > k)$ should be non-increasing in k, we define a monotonicity violation rate to monitor the correctness of our modifications to the CORAL setup:

$$v = \frac{1}{K-2} \sum_{k=0}^{K-3} \mathbb{1}[p_{k+1} > p_k] \tag{44}$$

and report its mean over valid candidates as a diagnostic.

B. Implementation deltas vs. coral-pytorch

We build on the reference implementation in `coral-pytorch` [18] and extend it for RRI regression:

- **Label handling**: convert ordinal labels to level targets internally (no external `levels_from_labelbatch` step).
- **Class marginals**: derive π_k from cumulative probabilities, then compute expected RRI with bin representatives (Eq. above).
- **Learnable bin values**: treat u_k as monotone learnable scalars via softplus deltas,

$$u_0 \in \mathbb{R}, \quad u_k = u_0 + \sum_{j=1}^{k} \text{softplus}(\delta_j) \tag{45}$$

- **Bias initialization from priors**: optional initialization of per-threshold biases from fitted class priors (instead of fixed descending bias values).
- **Loss variants**: support balanced BCE and focal threshold losses to mitigate imbalance (definitions in Appendix Section XVIII).
- **Diagnostics**: log monotonicity violations and a relative-to-random baseline (

$$\mathcal{L}_{\text{rel}} = \frac{\mathcal{L}_{\text{coral}}}{(K-1)\,\log(2)} \tag{46}$$

) for calibration tracking.

C. Auxiliary regression

We optionally add a Huber loss on $\hat{r}$ to stabilize early training and improve calibration. The final objective is

$$\mathcal{L} = \mathcal{L}_{\text{coral}} + \lambda \cdot \mathcal{L}_{\text{reg}} \tag{47}$$

The auxiliary weight λ is decayed over training to encourage sharper ordinal separation while retaining a meaningful continuous prediction. In the current training config we use $\lambda_0 = 10$, $\gamma = 0.99$, $\lambda_{\min} = 0.1$, and apply the decay once per epoch.

D. Threshold imbalance and balancing

Ordinal thresholds are inherently imbalanced because early thresholds have high positive rates. To mitigate collapse toward constant predictions, we support balanced BCE or focal variants over thresholds, using priors from the fitted binner. This retains CORAL semantics while improving gradients for rare thresholds [19]. We also log monotonicity violations to ensure ordinal consistency.

IX. Evaluation Protocol: Metrics

- Bidirectional surface error: overall reconstruction quality (accuracy + completeness).
- Accuracy and completeness: directional components for diagnosis.
- Oracle RRI: normalized improvement per candidate.
- Ranking and ordinal metrics: Spearman correlation, top-k bin accuracy, confusion matrices, and label histograms for RRI bins.

When training a candidate scorer, these metrics are logged per epoch and per interval to detect collapse and to validate the effect of imbalance-aware ordinal losses. In this paper, we provide the definitions and use them for oracle-label diagnostics.

A. Ranking and ordinal metrics

Let $s_i \in [0, 1]$ be the predicted score for candidate i (the normalized expected CORAL bin) and r_i the oracle RRI. We measure ranking agreement with Spearman's rank correlation [20]:

$$\rho = \text{corr}\Big(\text{rank}\,\{s_i\}_{i\in\mathcal{J}}, \text{rank}\,\{r_i\}_{i\in\mathcal{J}}\Big) \tag{48}$$

Current runs report all metrics defined in `VinLightningModule`: CORAL losses, auxiliary regression diagnostics, Spearman correlation, top-3 accuracy, confusion matrices, label histograms, monotonicity violations, and coverage fractions (voxel and semidense). Results in this paper use the ASE-EFM GT split from the offline cache (80 scenes, 883 snippets; train/val split 706/177).

where $\mathcal{J}$ contains candidates with finite oracle RRI and finite model outputs. We compute ρ over all valid candidates accumulated across an epoch (flattened over snippets and candidates).

For ordinal supervision, let $y_i \in \{0, ..., K-1\}$ be the binned oracle label and $\boldsymbol{p}_i \in [0, 1]^K$ the predicted class probabilities. We report top-k bin accuracy

$$\text{Acc@k} = \frac{1}{|\mathcal{J}|} \sum_{i\in\mathcal{J}} \mathbb{1}[y_i \in \text{TopK}(\boldsymbol{p}_i, k)]. \tag{49}$$

Here, $\text{TopK}(\boldsymbol{p}_i, k)$ denotes the set of the k most probable bins under $\boldsymbol{p}_i$.

In our logs we use $k = 3$. Confusion matrices and label histograms are computed from $(\hat{y}_i, y_i)$, where $\hat{y}_i$ is the decoded ordinal class from CORAL logits, to expose class imbalance and mode collapse:

$$\begin{aligned} C_{a,b} &= |\{i \in \mathcal{J} \mid \hat{y}_i = a, y_i = b\}| \\ h_b &= |\{i \in \mathcal{J} \mid y_i = b\}|. \end{aligned} \tag{50}$$

X. Training Dynamics

This section summarizes training dynamics of the current best run and visualizes how validation ordinal performance evolves over training.

The run shown here is `rtjvfyyp` (`v03-best`).

We focus on the **within-run** improvement (start → finish) rather than attributing changes to specific architectural or optimization choices. A comparison against another near-identical run is discussed in Section XI.

In particular, `rtjvfyyp` improves from validation relative CORAL loss $\mathcal{L}_{\text{rel}} = 0.743$ to 0.666 and from Spearman $\rho = 0.254$ to 0.501 over training.

Metric	Start → finish (delta)
$\mathcal{L}_{\text{rel}}^{\text{train}}$	0.777 → 0.659 (-15.2%)
$\mathcal{L}_{\text{rel}}^{\text{val}}$	0.743 → 0.666 (-10.4%)
$\rho(r, \hat{r})$	0.254 → 0.501 (+96.9%)
Acc@3 (val)	0.248 → 0.329 (+32.8%)

Table 4. Start → finish improvements for `rtjvfyyp` (first and last logged points).

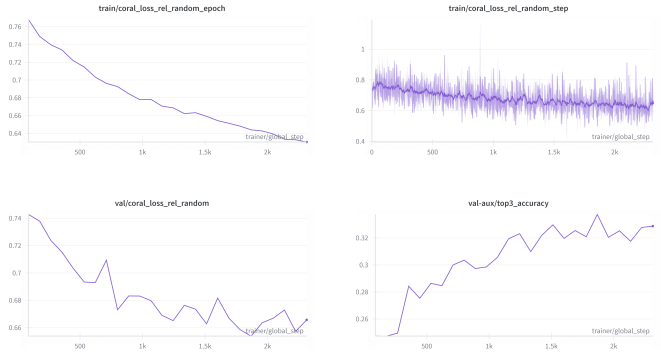


Fig. 22. CORAL loss (relative-to-random) and validation top-3 bin accuracy (shown for `rtjvfyyp`).

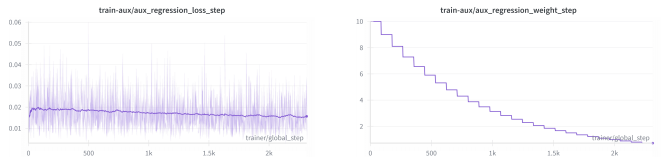
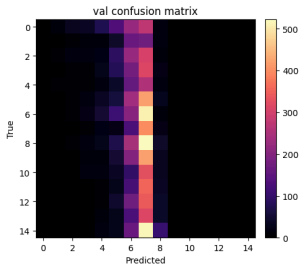
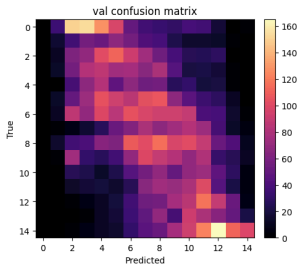


Fig. 23. Auxiliary regression loss and weight schedule (shown for `rtjvfyyp`).

text(weight: “bold”) [`rtjvfyyp` (start)]



text(weight: “bold”) [`rtjvfyyp` (finish)]



Listing 1. Validation confusion matrices for `rtjvfyyp` (first and last logged).

A. Observations

- The run improves validation ordinal performance substantially over training (Spearman and top-3 accuracy increase, and `val/coral_loss_rel_random` decreases).
- The **early** validation confusion matrices are highly collapsed (nearly constant predictions), while the **final** matrices show a much richer structure with fewer massed columns.

XI. Ablation Plan and Open Experiments

We tested various archtectural features, but could not establsih significant results due to the lack time and computational ressources as well as too many DoFs in our previous Optuna sweeps. Hence, we refer to these potential ablations as planned experiments with hypotheses about their impact; the expected outcomes are summarized in Table 5.

Ablation	Change	Hypothesis
Semi-dense point encoder	on/off (PointNeXt)	Does a global semi-dense embedding help beyond view-conditioned cues?
Semi-dense frustum MHCA	on/off	Does token-level candidate conditioning improve ranking vs. projection stats alone?
Visibility token embedding	on/off	Token-type embedding should help keep invalid points informative without masking.
Mask invalid tokens	on/off	Masking may stabilize attention but can erase evidence about **missing** visibility.
Observation counts	on/off + normalization choice	Track-length features should correlate with point reliability and improve frustum aggregation.
Trajectory context	on/off	History-aware features should reduce ambiguity between candidates with similar geometry.
Voxel reliability gating	on/off	Gating should reduce noise when candidates fall outside EVL voxel extent.
Voxel reliability feature	on/off	Explicitly conditioning the head on evidence quality should improve calibration.
Global pool resolution	grid size G in $\{4, 5, 6, 7, 8\}$	Higher resolution may help in clutter but risks overfitting / compute overhead.
CORAL imbalance handling	loss variant: CORAL vs. balanced vs. focal	Better gradients for rare thresholds; less median-bin collapse.
Coverage-weight curriculum	on/off (annealed weighting)	Early emphasis on high-evidence candidates should prevent collapse while still learning all candidates.

Table 5. Planned ablations and hypotheses.

We will report per-epoch Spearman correlation, confusion matrices, and calibration curves for each ablation, and use these results to guide the transition to entity-aware NBV.

A. Current Evidence: Two Baselines

The current top-2 training runs runs provide an **uncontrolled** comparison of trajectory conditioning:

- `hq1how1j` (`R2026-01-27_13-08-02`): trajectory encoder disabled, ReduceLROnPlateau, no

auxiliary regression loss, batch size of 16 instead of 8. Trained for 22 epochs (early stopped at 21).

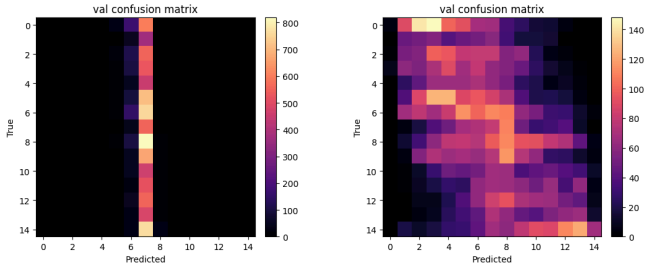
- `rtjvfyyp` (`v03-best`): trajectory encoder enabled, OneCycleLR. Trained for 35 epochs (early stopped at 22)

`rtjvfyyp` achieves better validation metrics, but the evidence is inconclusive: the two runs differ in more than the trajectory encoder and require a controlled ablation to attribute gains to trajectory features.

Run	$\mathcal{L}_{\text{rel}}$	ρ	TopKAcc(3)
`hq1how1j`	0.677	0.469	0.314
`rtjvfyyp`	0.666	0.501	0.329

Table 6. Final validation metrics for the current top-2 runs (last logged points).

text(weight: "bold") [`hq1how1j` (start)] text(weight: "bold") [`hq1how1j` (finish)]



Listing 2. Validation confusion matrices for `hq1how1j` (first and last logged). The corresponding matrices for `rtjvfyyp` are shown in Listing 1.

Status. The current VINv3 baseline (Jan 2026 run) already realizes several ablations by design: PointNeXt is off, frustum MHCA is off, and trajectory context is disabled. The delta to `rtjvfyyp` confirms that trajectory features are neither harmful nor informative, that the choice of LR schedule doesn't matter significantly, and that auxiliary regression loss is not essential to training performance. It shoudl be noted that the confusion matrix of the ablation run looks somewhat better calibrated than the baseline. Futhermore, both runs use coverage-weight curriculum, balanced CORAL loss, and voxel reliability gating and features which should be ablated in future controlled experiments.

XII. Discussion and Limitations

The current Aria-VIN-NBV system demonstrates a practical path to quality-driven NBV for egocentric scenes, but several limitations remain.

A. Local voxel extent

EVL produces a local voxel grid centered on the latest pose. Candidates outside this extent receive limited voxel evidence. Semidense projection features (VIN v3) and, in VIN v2 ablations, frustum attention mitigate the issue, yet fully global representations remain an open challenge.

B. Stage dependence and label scaling

RRI distributions shift over the course of a trajectory. Early views typically yield larger gains than late-stage refinements. Stage-aware features or trajectory-conditioned binning may be required for consistent calibration.

C. Computational cost of oracle labels

Oracle RRI computation requires depth rendering and point-to-mesh distances for all candidates, which is expensive and limits dataset size. This cost also makes on-policy training of continuous 5-DoF action policies impractical in our current system: each policy step would require multiple oracle evaluations, and our labeler is not yet optimized for large-scale multiprocessing. Discretizing the action space into a candidate set amortizes oracle cost and provides dense supervision (RRI labels for all candidates), enabling stable offline training in future work. A learned RRI predictor can then act as a fast surrogate objective for continuous pose search or eventual on-policy fine-tuning.

D. Entity-aware objectives

The current pipeline optimizes global reconstruction quality. ASE provides object-level annotations and OBBs that enable entity-aware RRI. Incorporating per-entity objectives is a promising direction for task-driven NBV.

XIII. Future Extensions

This project currently establishes oracle supervision for egocentric NBV and a learned candidate scorer baseline. The following extensions are enabled by the existing pipeline and represent the main research and engineering directions we have identified.

A. Toward Entity-Aware NBV

ASE provides object-level annotations and EVL predicts 3D OBBs, enabling task-driven NBV. Instead of optimizing only scene-level reconstruction quality, we can optimize a weighted combination of global and entity-specific improvement:

$$\text{RRI}_{\text{total}(q)} = \sum_{e \in \mathcal{E}} \boldsymbol{w}_e \cdot \text{RRI}_e + \lambda_{\text{scene}} \cdot \text{RRI} \qquad (51)$$

Here, $(\boldsymbol{w}_e)$ encodes entity importance (user- or task-defined), and λ_{scene} trades off between entity-centric and global quality.

How to define the entity-specific term. A practical definition reuses the existing oracle and changes only the **evaluation subset**:

- **OBB-cropped mesh + points:** define an entity region from a GT OBB (or EVL prediction), crop $\boldsymbol{\mathcal{M}}_{\text{GT}}$ and filter both the current semidense points $\boldsymbol{\mathcal{P}}_{t_t}$ and candidate points $\boldsymbol{\mathcal{P}}_q$ to that region (plus a

margin), then compute $\mathrm{RRI}_{e(q)}$ with the same point $\leftrightarrow$ mesh evaluation.
- **OBB surface proxy:** approximate the entity surface by sampling points on the OBB (or an OBB-derived SDF shell) and compute a completeness-style proxy that rewards observing previously missing regions of that proxy surface.

B. Scaling supervision and data products

Oracle labels are expensive; scaling coverage is the primary lever for stronger training and more reliable conclusions.

- Address dataset constraints explicitly: each ASE scene provides only one prerecorded trajectory (no arbitrary novel viewpoints), GT meshes are limited to a subset of scenes, and beyond that we must rely on pseudo-GT or reduced
- Extend the immutable VIN offline store to cover the full mesh-supervised ASE subset.
- While the number of scenes with GT meshes is limited to 4608 (of which we have been using 19.2%), we can increase variability by:
  - Choosing subsequences of the pre-recorded trajectories
  - Generating multiple candidate sets per snippet and altering the candidate sampling strategy (e.g., more diverse orientations, wider spatial coverage, allow backward-facing views, roll jitter, …)

C. From discrete candidates to continuous planning

The current system ranks a discrete candidate set; planning and continuous action spaces remain open.

- Learn continuous pose proposal distributions whose samples are scored by the learned RRI predictor, using free-space/occupancy for collision avoidance.
- Evaluate the design choice of **filtering** invalid candidates versus including them with strong penalties; this affects the RRI distribution and therefore CORAL binning and calibration. There are various caveats to consider.
- Explore differentiable refinement of candidate poses by querying voxel fields with differentiable sampling primitives, enabling gradient-based local pose improvement around promising candidates.

D. Oracle throughput and robustness

Several improvements target correctness and scalability of oracle supervision.

E. Upgrades to the View Introspection Network

The main goal is to strengthen candidate-specific signal, improve calibration, and reduce mode collapse while preserving interpretability.

- Apply candidate shuffling in the datamodule after batching.
- Validate learnable CORAL bin shifts/centers against baseline fixed binning.

F. Evaluation, deployment, and human-in-the-loop systems

Beyond per-snippet ranking, we need end-to-end evaluation and real-world integration.

- Extend toward real-device deployment (Aria, Quest 3, iPhone LiDAR) and sim-to-real evaluation using the same pose/camera/point primitives.
- Compare alternative backbones for NBV scoring (e.g., EVL vs. SceneScript) and assess whether fine-tuning an EFM on NBV objectives improves performance on target platforms.
- Build an interactive human-in-the-loop NBV guidance system: entity selection UI, real-time scoring with streaming updates, and AR overlays for viewpoint guidance.
- Explore LLM/VLA integration for natural-language explanations and high-level task guidance layered on top of entity-aware objectives to allow specifying objects of interest via language.

G. Experiment management and reporting

- Run stationary Optuna sweeps focused on architectural toggles (avoid confounding schedule/width changes) and tag trials by sweep phase for clean analysis.

XIV. Conclusion

We presented an oracle supervision and diagnostics pipeline for quality-driven NBV research in egocentric indoor scenes. The approach computes per-candidate oracle RRI labels from ASE ground-truth meshes and semi-dense reconstructions, and provides tooling to inspect candidate generation, depth rendering, and surface-error behavior. We also outlined an ordinal label representation (CORAL) and trained a preliminary VIN v3 candidate scorer on frozen EVL features. Learning a full next-best-view policy on top of these labels remains future work, including entity-aware objectives and learning more expressive view-conditioned representations. Furthermore, we have implemented a rich ML experiment management and development suite with immutable VIN offline stores, W&B + Optuna integration, various helpful CLI utilities, and a comprehensive Streamlit app for data inspection and debugging.

Quantitatively, our current VIN offline store covers 80 ASE-EFM GT scenes with 883 oracle-labelled snippets (train/val 706/177) drawn from 4608 snippet windows. The best VIN v3 run (`v03-best`, W&B id `rtjvfyyp`) achieves non-trivial ranking performance (val Spearman ≈ 0.501, val top-3 accuracy ≈ 0.329), indicating that the oracle labels and feature set can support learning while leaving room for further improvements and controlled ablations.

XV. Appendix: OracleRriLabeler Pipeline

This appendix is intentionally kept minimal.

All pipeline details (candidate generation, PyTorch3D depth rendering, NDC-aligned backprojection, point↔mesh scoring, and optional ordinal binning) are consolidated in Section 5 (**Oracle RRI Computation**), together with the paper figures and the effective module parameters loaded from `/typst/shared/data/paper_figures_oracle_labeler.toml`.

XVI. Appendix: VIN Pose Frames and Consistency Checks

This appendix summarizes the SE(3) frames provided to `VinModelV3.forward`, using the notation from the coordinate conventions section. We verify the conventions with offline-cache data loaded via `.configs/offline_only.toml` (the debug config in `.vscode/launch.json`).

A. CW90 rig-basis correction (`rotate_yaw_cw90`)

`rotate_yaw_cw90` is a fixed 90° twist about the pose-local $+z$ (forward) axis (a roll); the name is historical. In the current pipeline, candidates are generated about $\boldsymbol{T}^{\mathrm{w}}_{\mathrm{rig}}(T)$ after applying `rotate_yaw_cw90` to that reference pose. Without this one-time rig-basis correction, the candidate sampling azimuth and elevation effectively swap when interpreted in a right-handed LUF camera frame.

This correction must be applied **at most once** along any path. Plotting code may apply the same twist purely for display; do not apply it again to already-corrected poses. If the rotation is undone for learning or diagnostics (e.g., `apply_cw90_correction=True`), the same undo must be reflected in the associated PyTorch3D cameras; otherwise pose encoding and projection features desynchronize (see checks below).

B. Inputs to `forward`

Frame glossary (consistent with the main notation):

- w: global world frame (gravity-aligned).
- r: reference rig frame for the snippet (physical headset pose).
- c_q: candidate camera frame in LUF coordinates (left-up-forward).
- v: EVL voxel grid frame (local, gravity-aware).

Input	Notation	SE(3) meaning	Notes
`candidate_poses_world_cam`	$\boldsymbol{T}^{w}_{c_q}$	world → camera (PoseTW)	Pose of each candidate camera in world coordinates.
`reference_pose_world_rig`	$\boldsymbol{T}^{w}_{r}$	world → rig reference (PoseTW)	Physical rig pose; not gravity-aligned in cache.
`p3d_cameras`	$\boldsymbol{T}^{c_q}_{w}$	camera → world (PerspectiveCameras)	PyTorch3D row-vector convention, with $\boldsymbol{T}^{c_q}_{w} = \boldsymbol{T}^{w-1}_{c_q}$.
`backbone_out.t_world_voxel`	$\boldsymbol{T}^{w}_{v}$	world → voxel (PoseTW)	EVL voxel grid pose for sampling voxel-aligned fields.
`backbone_out.pts_world`	$\boldsymbol{p}^{w}$	points in world frame	Voxel-center points used for positional encoding and pooling.

Table 7. SE(3) inputs passed into `VinModelV3.forward` and their frame meaning.

Derived frames inside the model. The pose encoder builds candidate poses in the reference rig frame $\boldsymbol{T}^{r}_{c_q} = \boldsymbol{T}^{w-1}_{r}\boldsymbol{T}^{w}_{c_q}$. Semidense and voxel projections are evaluated in camera frames derived from `p3d_cameras` and must be consistent with the above pose encoding.

C. Offline-cache consistency checks

Using cached batches (same config as the debug launcher), we verified:

- `candidate_poses_world_cam` and `p3d_cameras` are consistent: $\boldsymbol{T}^{c_q}_{w}$ reconstructed from `candidate_poses_world_cam` matches `p3d_cameras` with max abs error $\leq 1e-6$.
- Undoing the CW90 display correction on poses **without** rotating `p3d_cameras` breaks that alignment (max abs error approx 1.2 in R, approx 10 in t), which corrupts semidense projection features.

D. Current issues

- **CW90 correction mismatch**: `apply_cw90_correction=True` in `VinModelV3` only adjusts poses unless the caller pre-corrects `p3d_cameras`. This can desynchronize pose encoding and projection features.
- **Missing gravity-aligned reference in cache**: offline cache stores only `reference_pose_world_rig` (physical rig). The gravity-aligned sampling pose is not available, so cached datasets cannot be retrofitted without recompute.
- **Mixed display vs. physical frames**: display-only rotations (CW90) should not leak into model inputs.

E. Recommended streamlining

1) **Single canonical frame per batch**: treat all cached inputs as physical rig frames; disable CW90 inside the model for cached datasets.
2) **Batch-level normalization helper**: if we want to undo CW90, apply it consistently to `candidate_poses_world_cam`, `reference_pose_world_rig`, and `p3d_cameras` (plus a `cw90_corrected` tag) before entering `forward`.

The CW90 consistency guard is implemented in `VinModelV3.forward`: it raises if `apply_cw90_correction=True` without a `p3d_cameras.cw90_corrected` tag.

XVII. VIN offline store and batching

This appendix summarizes why we materialize oracle outputs, what is stored, and the approximate storage footprint per snippet and for the full ASE mesh subset.

A. Motivation

The oracle pipeline combines candidate sampling, depth rendering, backprojection, and point-to-mesh scoring.

Each step is GPU-heavy and hard to parallelize inside PyTorch. In practice, this results in per-snippet runtimes on the order of tens of seconds, while the EVL backbone alone can consume multiple GB of GPU memory per forward pass and hence limits us to batch sizes of one. The immutable VIN offline store makes training a standard supervised learning problem and enables larger batch sizes for noisy ordinal supervision.

B. Storage footprint

Let S_{cur} denote the size of the cached subset and f_{cover} the scene coverage fraction. We estimate the full-coverage size as:

$$S_{\text{full}} \approx \frac{S_{\text{cur}}}{f_{\text{cover}}} \tag{52}$$

.

For the current materialized subset we observe:

- $S_{\text{cur}} = 263$ GB for materialized oracle/backbone payloads.
- $S_{\text{vin}} = 0.809$ GB for minimal VIN snippet tensors.
- $S_{\text{full}} \approx 1372$ GB for 100 mesh scenes.

Field	MB
backbone_out	304.9 MB
candidate_pcs	11.2 MB
depths	4.6 MB
candidates	0.03 MB
rri	0.002 MB
vin_snippet	1 MB
total + backbone	320.8 MB
min train payload	1.1 MB

Table 8. Representative per-snippet tensor footprint (CPU, float32).

C. Minimal training payload

The bare minimum for VIN training (with cached candidates) is:

- Candidate poses and PyTorch3D camera parameters.
- Oracle targets (RRI + point-to-mesh components).
- `VinSnippetView` (semi-dense points, lengths, trajectory).

This minimal payload is approximately 1.1 MB per snippet in the current configuration. Caching the full EVL backbone output adds 304.9 MB per snippet, dominating storage when enabled.

XVIII. Appendix: Additional Diagnostics

This appendix collects additional diagnostics and logging definitions referenced from the main text.

A. Current Training Configuration (VIN v3 Ablation)

This table records the configuration used in the `R2026-01-27_13-08-02` VINv3 baseline training run.

Parameter	Value
Number of classes	15
Head hidden dim	192
Head layers	2
Head dropout	0.031317082333434144
Field dim	24
Global pool grid	5×5×5
Semidense proj CNN	enabled
Semidense obs. count	off
Trajectory encoder	off
Point encoder (PointNeXt)	off
Voxel valid-frac gate	off
Optimizer	AdamW
Learning rate	0.0008
Weight decay	0.013244782071249196
Scheduler	ReduceLROnPlateau
LR factor / patience	0.4 / 2
Gradient clip	4
Coverage weighting	voxel (anneal 0.6→0 over 6 epochs)
Aux loss	none
Aux weight gamma	0.9
CORAL bias init	prior logits

Table 9. Training configuration for the VIN v3 baseline (Jan 2026 run).

Note: auxiliary regression can be disabled; the curves in Fig. 23 correspond to a run where it was enabled.

B. Logged training metrics (VIN Lightning)

We log scalars and diagnostic figures in a few namespaces:

- `stage/...` for main loss scalars, where `stage` is one of `train`, `val`, or `test`.
- `stage-aux/...` for auxiliary/diagnostic scalars (ranking, validity, and loss variants).
- `stage-figures/...` for confusion matrices and label histograms (logged as images).
- `train-gradnorms/grad_norm_*` for per-module gradient norms (training only).

When the same scalar is logged both per-step and per-epoch, Lightning emits two keys with suffixes `_step` and `_epoch`. Some diagnostics are explicitly step-only (e.g. `train-aux/spearman_step`).

Logged loss scalars.

- Main losses (`stage/...`):
  - `stage/loss`: Combined objective (Eq. (53)).
  - `stage/coral_loss`: Mean ordinal loss (CORAL thresholds; Eq. (54)).
  - `stage/coral_loss_rel_random`: Relative-to-random baseline (diagnostic; Eq. (55)).
- Auxiliary / diagnostic losses (`stage-aux/...`):
  - `stage-aux/aux_regression_loss`: Optional regression on expected RRI $\hat{r}$ (Huber: Eq. (56); MSE: Eq. (57)).
  - `stage-aux/coral_loss_balanced_bce`: Balanced-BCE threshold loss (diagnostic; Eq. (58), weights Eq. (59)).
  - `stage-aux/coral_loss_focal`: Focal threshold loss (diagnostic; Eq. (60), defs Eq. (61)).

Logged diagnostic scalars + figures.

- Ranking + calibration (`stage-aux/...`):
  - `stage-aux/rri_mean`: Mean oracle r (Eq. (62)).
  - `stage-aux/pred_rri_mean`: Mean predicted $\hat{r}$ (Eq. (63)).
  - `stage-aux/spearman`: Spearman correlation (Eq. (64)).
  - `train-aux/spearman_step`: Step-interval Spearman (every `log_interval_steps`).
  - `stage-aux/top3_accuracy`: Top-3 bin accuracy (k=3; Eq. (65)).
  - `val-aux/pred_rri_bias2`: Validation bias^2 (diagnostic).
  - `val-aux/pred_rri_variance`: Validation variance (diagnostic).
- Validity + coverage (`stage-aux/...`):
  - `stage-aux/candidate_valid_frac`: Candidate validity fraction (Eq. (69)).
  - `stage-aux/voxel_valid_frac_mean` / `stage-aux/voxel_valid_frac_std`: Voxel proxy mean/std (mean/std over candidates).
  - `stage-aux/semidense_candidate_vis_frac_mean` / `stage-aux/semidense_candidate_vis_frac_std`: Semidense visibility mean/std (over candidates).
  - `train-aux/coverage_weight_strength`: Coverage-weight strength λ (train only; Eq. (72) / (73)).
  - `train-aux/coverage_weight_mean`: Mean weight w (train only; Eq. (70)).
  - `stage-aux/aux_regression_weight`: Aux-loss weight schedule λ_{reg} (Eq. (71)).
- CORAL sanity (`stage-aux/...`):
  - `stage-aux/coral_monotonicity_violation_rate`: Monotonicity violation rate (Eq. (74)).
- Robustness flags (`stage/...`):
  - `stage/drop_nonfinite_logits_frac`: Non-finite logits among finite RRIs (Eq. (75)).
  - `stage/skip_nonfinite_logits`: Skip indicator when logits are non-finite (Eq. (76)).
  - `stage/skip_no_valid`: Skip indicator when no finite RRIs exist (Eq. (77)).
- Figures (`stage-figures/...`):
  - `stage-figures/confusion_matrix`: Confusion matrix image (Eq. (66)).

- `stage-figures/label_histogram`: Label histogram image (Eq. (67)).
- `train-figures/confusion_matrix_step` / `train-figures/label_histogram_step`: Step-interval variants (train only).

- Optimization diagnostics:
  - `train-gradnorms/grad_norm_*`: Gradient norms over selected `vin` submodules (train only; Eq. (78)).

a) *Definitions (selected):*
Losses:

$$\mathcal{L} = \mathcal{L}_{\text{coral}} + \lambda \cdot \mathcal{L}_{\text{reg}} \tag{53}$$

$$\mathcal{L}_{\text{coral}}(y, \boldsymbol{p}) = -\sum_{k=0}^{K-2} (t_k \log(p_k) + (1 - t_k) \log(1 - p_k)) \tag{54}$$

$$\mathcal{L}_{\text{rel}} = \frac{\mathcal{L}_{\text{coral}}}{(K-1)\log(2)} \tag{55}$$

$$\mathcal{L}_{\text{reg}} = \frac{1}{N} \sum_i \text{Huber}_1(\hat{r}_i - r_i) \tag{56}$$

$$\mathcal{L}_{\text{reg}} = \frac{1}{N} \sum_i (\hat{r}_i - r_i)^2 \tag{57}$$

$$\mathcal{L}_{\text{bal}} = -\frac{1}{K-1} \sum_{k=0}^{K-2} (w_k t_k \log(p_k) + (1 - t_k) \log(1 - p_k)) \tag{58}$$

$$w_k = \frac{1 - \pi_k^{\text{th}}}{\pi_k^{\text{th}}} \tag{59}$$

$$\mathcal{L}_{\text{focal}} = -\frac{1}{K-1} \sum_{k=0}^{K-2} \alpha_{t,k} (1 - p_{t,k})^{\gamma} \log(p_{t,k}) \tag{60}$$

$$p_{t,k} = p_k t_k + (1 - p_k)(1 - t_k), \quad \alpha_{t,k} = \alpha t_k + (1 - \alpha)(1 - t_k) \tag{61}$$

Metrics and schedules:

$$|(r) = \frac{1}{N} \sum_i r_i \tag{62}$$

$$|(\hat{r}) = \frac{1}{N} \sum_i \hat{r}_i \tag{63}$$

$$\rho = \text{corr}(\text{rank}(\hat{r}_i), \text{rank}(r_i)) \tag{64}$$

$$\text{TopKAcc}(k) = \frac{1}{N} \sum_i \mathbb{1}[y_i \in \text{TopK}(\boldsymbol{\pi}_i, k)] \tag{65}$$

$$C_{a,b} = |\{i : y_i = a, \hat{y}_i = b\}| \tag{66}$$

$$h_k = |\{i : y_i = k\}| \tag{67}$$

$$m_i = \mathbb{1}[\text{finite}] \cdot \mathbb{1}[v_i > 0] \cdot \mathbb{1}[v_i^{\text{sem}} > 0] \tag{68}$$

$$\frac{1}{N} \sum_i m_i \tag{69}$$

$$|(w) = \frac{1}{N} \sum_i w_i \tag{70}$$

$$\lambda_{\text{reg}}(t) = \max(\lambda_0 \cdot \gamma^t, \lambda_{\min}) \tag{71}$$

$$\lambda_t = \lambda_0 + (\lambda_T - \lambda_0) \cdot \left(\frac{t}{T}\right) \tag{72}$$

$$\lambda_t = \lambda_T + (\lambda_0 - \lambda_T) \cdot \frac{1 + \cos(\pi \frac{t}{T})}{2} \tag{73}$$

$$v = \frac{1}{K-2} \sum_{k=0}^{K-3} \mathbb{1}[p_{k+1} > p_k] \tag{74}$$

$$\frac{\sum_i \mathbb{1}[\text{finite}(r_i)] \cdot \mathbb{1}[\text{nonfinite}(\ell_i)]}{\sum_i \mathbb{1}[\text{finite}(r_i)]} \tag{75}$$

$$\mathbb{1}\left[\sum_i \mathbb{1}[\text{finite}(r_i)] > 0 \cdot \sum_i m_i = 0\right] \tag{76}$$

$$\mathbb{1}\left[\sum_i \mathbb{1}[\text{finite}(r_i)] = 0\right] \tag{77}$$

$$\|\nabla_\theta \mathcal{L}\|_2 \tag{78}$$

Gradient norms (`train-gradnorms/grad_norm_*`) are logged in `VinLightningModule.on_after_backward` and computed over config-selected `vin` submodules. Targets are collected by walking `vin.named_modules()` and selecting either modules at a fixed `group_depth` or explicit `include` glob patterns (with optional `exclude`), capped by `max_items` `norm_type` selects `L1`/`L2`/`Linf`.

C. Training dynamics

We summarize the most salient training-dynamics signals from the logged `train-aux/*` metrics (see Fig. 22 and Fig. 23):

- **Coverage weighting anneals as intended:** the coverage-weight strength decays to zero, and the mean per-sample weight rises toward one, indicating the curriculum transitions to uniform weighting.
- **Coverage/validity distributions are stable:** voxel-valid and semidense visibility means/standard deviations remain roughly constant over training, suggesting the schedule, not data drift, drives weight changes.
- **Aux-loss behaves as a stabilizer:** the auxiliary regression loss drops smoothly while its weight decays; predicted RRI mean converges near the oracle mean, indicating the aux task is quickly satisfied and then faded out.
- **Ranking improves after aux decay:** Spearman correlation and top-3 accuracy increase steadily even after coverage weighting diminishes, supporting the idea that the curriculum reduces early noise without preventing later ordinal learning.

D. EVL output summary

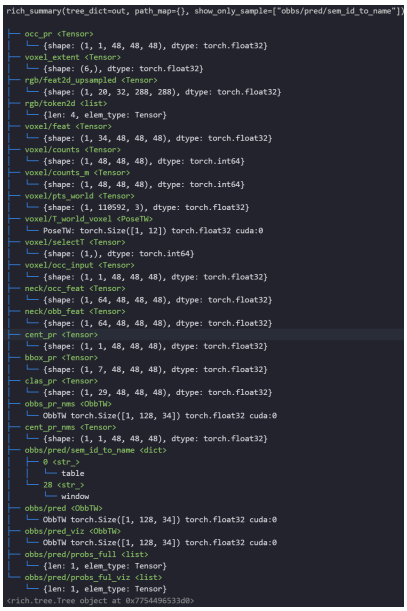


Fig. 24. EVL output summary used to construct the VIN scene field.

References

[1] N. Frahm et al., "VIN-NBV: A View Introspection Network for Next-Best-View Selection." [Online]. Available: https://arxiv.org/abs/2505.06219

[2] J. Straub, D. DeTone, T. Shen, N. Yang, C. Sweeney, and R. Newcombe, "EFM3D: A Benchmark for Measuring Progress Towards 3D Egocentric Foundation Models." [Online]. Available: https://arxiv.org/abs/2406.10224

[3] Meta Platforms Inc., "Aria Synthetic Environments Dataset." [Online]. Available: https://facebookresearch.github.io/projectaria_tools/docs/open_datasets/aria_synthetic_environments_dataset

[4] J. Engel et al., "Project Aria: A New Tool for Egocentric Multi-Modal AI Research." [Online]. Available: https://arxiv.org/abs/2308.13561

[5] X. Chen, Q. Li, T. Wang, T. Xue, and J. Pang, "GenNBV: Generalizable Next-Best-View Policy for Active 3D Reconstruction." [Online]. Available: https://arxiv.org/abs/2402.16174

[6] O. Cicek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation." [Online]. Available: https://arxiv.org/abs/1606.06650

[7] W. Cao, V. Mirjalili, and S. Raschka, "Rank consistent ordinal regression for neural networks with application to age estimation." [Online]. Available: https://arxiv.org/abs/1901.07884

[8] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. Courville, "FiLM: Visual Reasoning with a General Conditioning Layer." [Online]. Available: https://arxiv.org/abs/1709.07871

[9] A. Avetisyan et al., "SceneScript: Reconstructing Scenes With An Autoregressive Structured Language Model." [Online]. Available: https://arxiv.org/abs/2403.13064

[10] Meta Platforms Inc., "ATEK Data Store documentation." [Online]. Available: https://github.com/facebookresearch/ATEK/blob/main/docs/ATEK_Data_Store.md

[11] Meta Platforms Inc., "PyTorch3D Cameras + Rendering Documentation, Tutorial: Render a Textured Mesh." [Online]. Available: https://pytorch3d.readthedocs.io/en/latest/modules/renderer/cameras.html,%20https://pytorch3d.org/docs/renderer,%20https://pytorch3d.org/docs/renderer_getting_started,%20https://pytorch3d.org/docs/cameras,%20https://pytorch3d.org/tutorials/render_textured_meshes

[12] N. De Cao, W. Aziz, and I. Titov, "The Power Spherical distribution." [Online]. Available: https://arxiv.org/abs/2006.04437

[13] L. Papula, Mathematische Formelsammlung: Für Ingenieure und Naturwissenschaftler, 13th ed. Springer Fachmedien Wiesbaden, 2024. doi: 10.1007/978-3-658-45806-5.

[14] Y. Wu and K. He, "Group Normalization." [Online]. Available: https://arxiv.org/abs/1803.08494

[15] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the Continuity of Rotation Representations in Neural Networks," arXiv preprint arXiv:1812.07035, 2019, [Online]. Available: https://arxiv.org/abs/1812.07035

[16] Y. Li, S. Si, G. Li, C.-J. Hsieh, and S. Bengio, "Learnable Fourier Features for Multi-Dimensional Spatial Positional Encoding." [Online]. Available: https://arxiv.org/abs/2106.02795

[17] A. Vaswani et al., "Attention Is All You Need." [Online]. Available: https://arxiv.org/abs/1706.03762

[18] Raschka Research Group, "coral-pytorch documentation." [Online]. Available: https://raschka-research-group.github.io/coral-pytorch/

[19] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection." [Online]. Available: https://arxiv.org/abs/1708.02002

[20] C. Spearman, "The proof and measurement of association between two things," The American Journal of Psychology, vol. 15, no. 1, pp. 72–101, 1904, doi: 10.2307/1412159.