Aria-NBV: Oracle RRI + VIN v3 Candidate Scoring

Offline oracle supervision, diagnostics, and learned NBV baseline

Jan Duchscherer

VCML Project WS24/25

› Munich University of Applied Sciences

Data and Oracle Pipeline

Candidate generation + depth rendering + RRI computation

Figure 1: First/last RGB, SLAM-L/R, and depth frames (example snippet).

ASE ATEK Dataset

ASE ATEK overview (what we need)

- Scale (full ASE): 100k scenes, 58M+ RGB frames, 67 days.
- Per snippet (ATEK/EFM view):
  1. Trajectory $\boldsymbol{T}_{\text{rig}}^{\text{w}}(t)$ (20 frames @10 Hz; 2 s window).
  2. Semi-dense SLAM points $\mathcal{P}_t$.
  3. 3M OBBs (43 classes).
- Local shard snapshot (`.data/ase_efm`): 100 scenes, 576 shards, 4,608 snippets (49 GB).

ATEK-EFM snippet format (key facts)

- WebDataset shards: per-scene folders with `shards-*.tar` (streamed, then windowed).
- Window stride: 10 frames (1 s overlap between consecutive 2 s snippets).
- Resized streams: RGB 240x240, SLAM 320x240 (calibration + rig poses preserved).
- GT meshes: $\mathcal{M}_{\text{GT}}$ for 100 validation scenes (watertight).

ASE ATEK Dataset

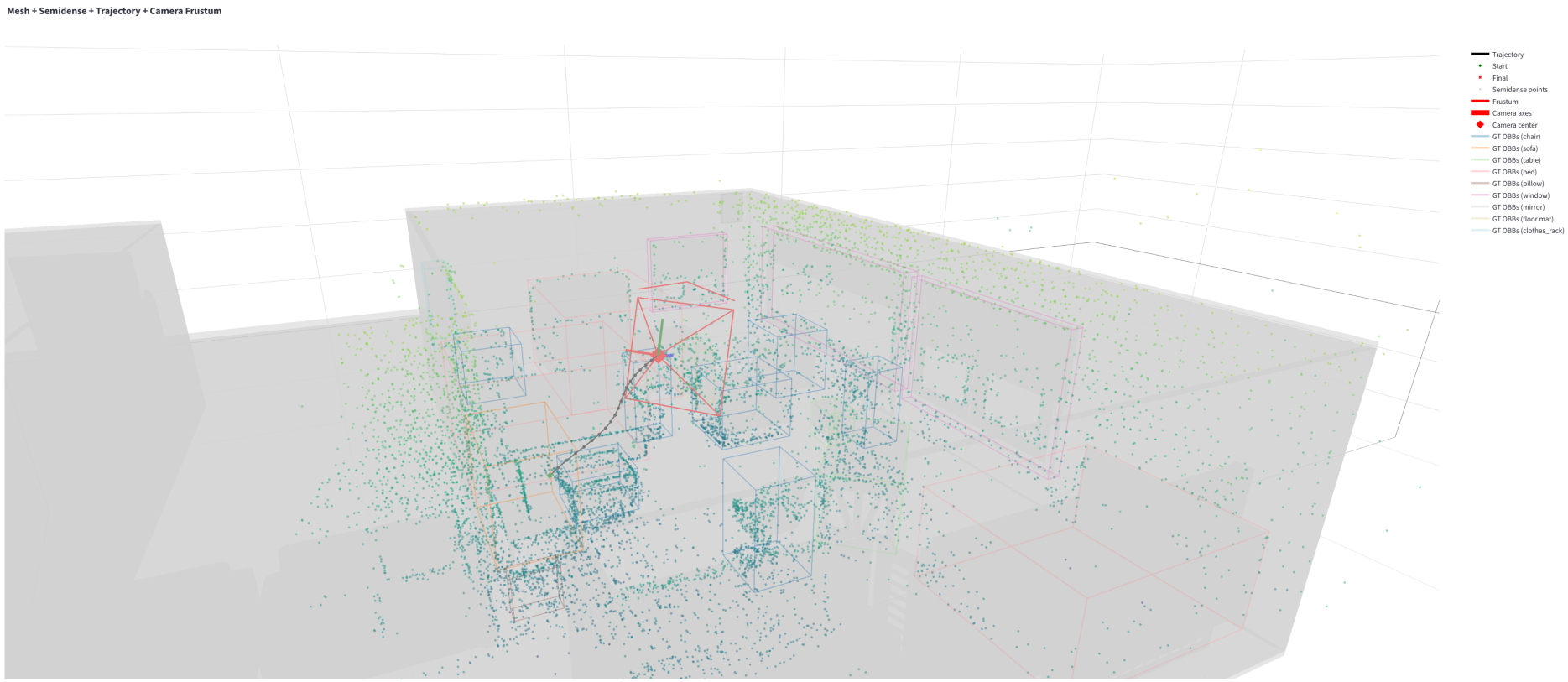


Figure 2: One snippet: $\mathcal{M}_{\mathrm{GT}}$ + $\mathcal{P}_t$ + $\boldsymbol{T}_{\mathrm{rig}}^{\mathrm{w}}(t)$ + camera frustum.

Oracle RRI Pipeline

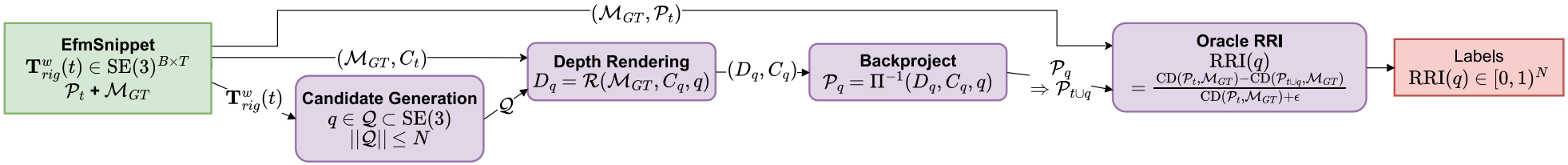


Figure 3: Oracle RRI Pipeline.

Stages

1. Sample candidate poses $\mathcal{Q}$ around the reference rig pose.
2. Render candidate depth maps $\boldsymbol{D}_q$ (metric z-buffer from $\mathcal{M}_{\mathrm{GT}}$).
3. Backproject valid depths $\longrightarrow$ candidate point clouds $\mathcal{P}_q$.
4. Evaluate point-to-mesh quality before/after and compute RRI.

```python
class OracleRriSample:
    sample: EfmSnippetView
    candidates: CandidateSamplingResult
    depths: CandidateDepths
    candidate_pcs: CandidatePointClouds
    rri: RriResult
```

Figure 4: Oracle RRI Sample.

Candidate Generation

Input

- Trajectory $\boldsymbol{T}_{\text{rig}}^{\text{w}}(t)$
- GT Mesh $\mathcal{M}_{\text{GT}}$

Output

- Candidate views $\mathcal{Q} \subset \text{SE}(3)^{N_q}$ + cameras $\boldsymbol{C}_q$
- Reference pose $\boldsymbol{T}_{\text{rig}}^{\text{w}}(T)$
- Valid + debug masks

Summary

- Sample candidate centers on a constrained shell around the reference pose.
- Assign view directions (radial-away or forward) and optional jitter.
- Prune using collision + free-space rules and min-distance checks.

Setting	Value
N_q	60
r_{min}	0.5
r_{max}	1.8
θ_{min}	−20
θ_{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).

Candidate Generation: Position sampling

Position sampling

- Sample directions on the unit sphere $\mathbb{S}^2 \subset \mathbb{R}^3$, then rescale into (ψ, θ) caps.
- Gravity-aligned sampling uses $\boldsymbol{T}_s^w$ (roll/pitch removed).
- $\boldsymbol{T}_r^w$ is the reference pose rotation; (ψ, θ) only parameterize the direction $\boldsymbol{s}_q \in \mathbb{S}^2$.

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

$$(\psi, \theta) = \text{angles}(\boldsymbol{s}_q)$$

$$(\psi, \theta) \leftarrow \text{lin}([\psi_{\min}, \psi_{\max}] \times [\theta_{\min}, \theta_{\max}]; (\psi, \theta))$$

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

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

[Pap24, p.43]

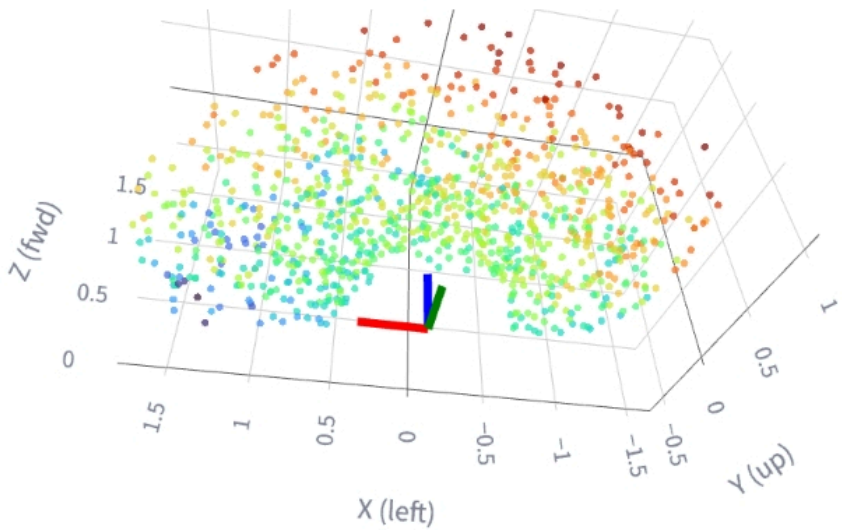


Figure 5: Candidate centers in the reference frame.

Candidate Generation: View directions

Direction + pose

- Base orientation from `view_direction_mode` (e.g., radial-away).
- (ψ, θ) caps apply to the **jitter delta** (box-uniform), not the base view.
- Compose candidate pose $\boldsymbol{T}_{c_q}^{w}$ from center + base + delta.

$$\boldsymbol{R}_{\text{base}} = \text{look-away}\left(c_q, \boldsymbol{T}_{\text{rig}}^{\text{w}}(T)\right)$$

$$\psi \sim \mathcal{U}\left(-\frac{\psi_\delta}{2}, \frac{\psi_\delta}{2}\right), \theta \sim \mathcal{U}\left(-\frac{\theta_\delta}{2}, \frac{\theta_\delta}{2}\right)$$

$$\boldsymbol{T}_{c_q}^{w} = \left(\boldsymbol{R}_{\text{base}} \circ \boldsymbol{R}_{\text{delta}}, c_q\right)$$

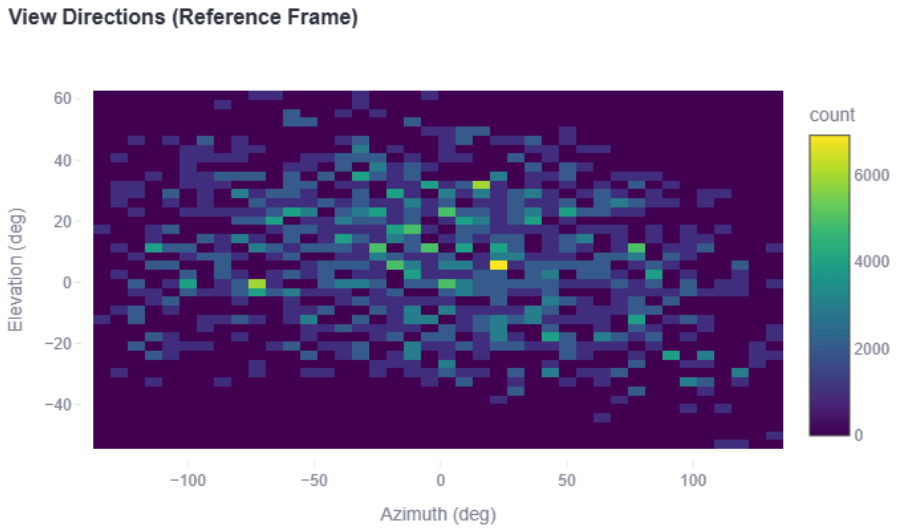


Figure 6: View direction density (azimuth/elevation).

Candidate Generation: Jitter + rules

View jitter

- Box-uniform jitter in yaw/pitch within caps:

$$\psi \sim \mathcal{U}\left(-\frac{\psi_\delta}{2}, \frac{\psi_\delta}{2}\right)$$

$$\theta \sim \mathcal{U}\left(-\frac{\theta_\delta}{2}, \frac{\theta_\delta}{2}\right).$$

- Optional roll:

$$\varphi \sim \mathcal{U}(-\varphi_\delta, \varphi_\delta)$$

$$\boldsymbol{R}_\delta = \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\varphi)$$

$$\boldsymbol{T}_{c_q}^{w} = \boldsymbol{T}_{c_q}^{w} \circ \begin{pmatrix} \boldsymbol{R}_\delta & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}$$

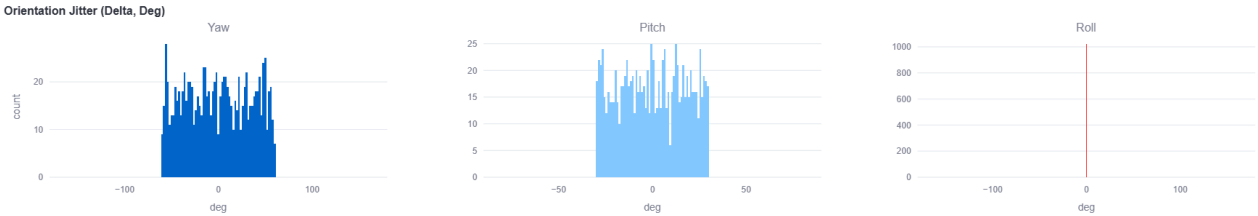


Figure 7: Orientation jitter distribution (delta yaw/pitch/roll, deg).

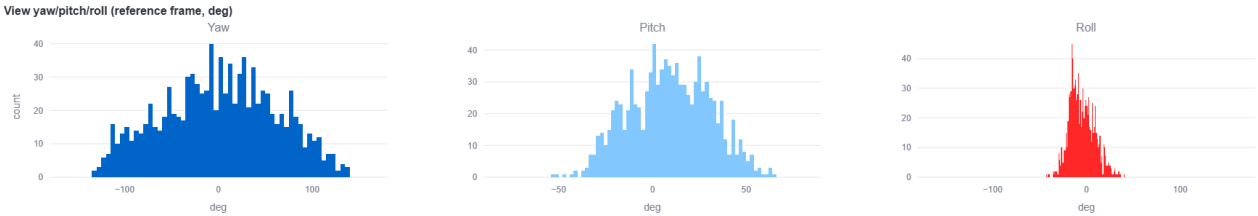


Figure 8: Reference-frame yaw/pitch/roll distribution (deg).

Pruning Rules

- **Rules**: min distance to mesh, collision-free ray, free-space AABB.

Candidate Depth Rendering

Input

- Candidate views $\mathcal{Q}$, C_q
- GT mesh $\mathcal{M}_{\text{GT}}$, $\mathcal{F}_{\text{GT}}$

Output

- Depth maps D_q
- Valid mask M_q
- C_q' (P3D cameras)

Transforms + ops

- Build PyTorch3D C_q' with extrinsics $T_{c_q}^{w}$ and $\left(\frac{W}{2}, \frac{H}{2}\right)$ from `CameraTW`.
- Rasterize $\mathcal{M}_{\text{GT}}$ to depth $D_q \subset R^{N_q \times H' \times W'}$.
- Valid mask: `pix_to_face` ≥ 0 and `znear` $< D_q <$ `zfar`
  - $M_q \subset \{0,1\}^{N_q' \times H' \times W}$

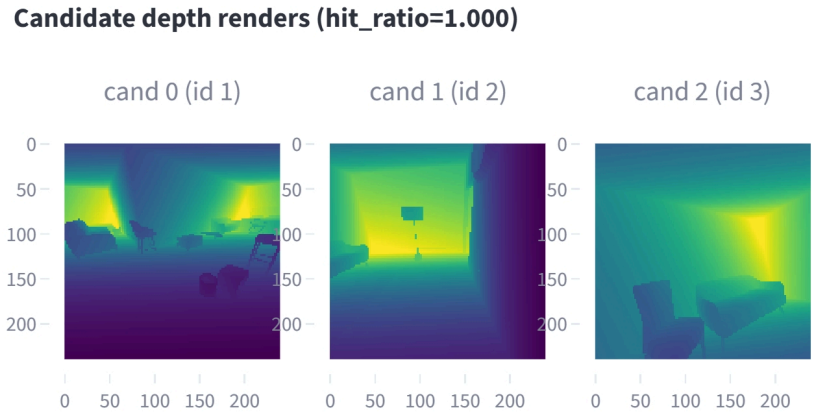


Figure 9: Candidate depth renders (1x3).

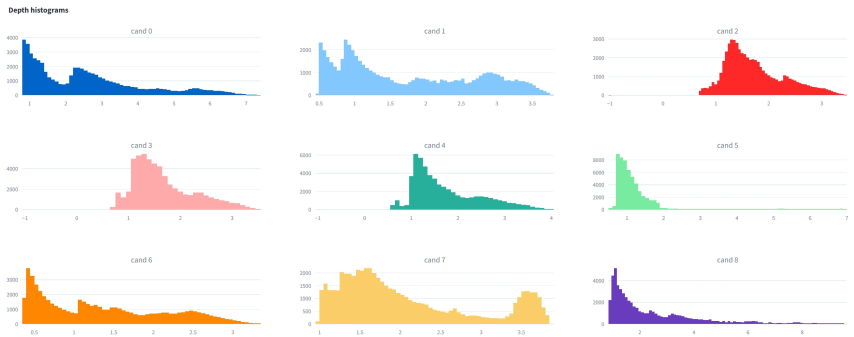


Figure 10: Depth histograms across candidates (3x3).

Backprojection

Input

- Depth maps $\boldsymbol{D}_q$ + $\boldsymbol{C}_q$
- Semi-dense PC $\mathcal{P}_t$

Output

- Candidate PCs $\mathcal{P}_q \subset R^{N_q \times P \times 3}$
- $\ell_q \subset \{0, ..., H \cdot W\}^{N_q}$
- AABBs $\boldsymbol{b}_{\{\text{aabb}\}} \subset R^6$

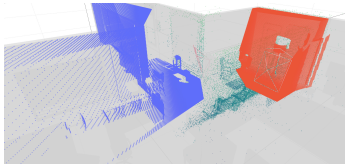


Figure 11: Backprojected candidate points + semi-dense SLAM points.

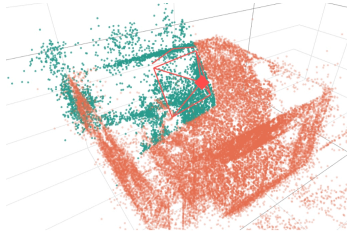


Figure 12: Candidate visibility in semi-dense point cloud.

Transforms + ops

1. Subsample pixel centers (u, v) by stride k.
2. Map to NDC with $s = \min(H, W)$ [Met25]:

$$x_{\text{ndc}} = -\left(u + \frac{1}{2} - \frac{W'}{2}\right)\left(\frac{2}{s}\right) \quad y_{\text{ndc}} = -\left(v + \frac{1}{2} - \frac{H'}{2}\right)\left(\frac{2}{s}\right)$$

3. Unproject: $\boldsymbol{p}_{\text{world}} = \Pi^{-1}(x_{\text{ndc}}, y_{\text{ndc}}, d_q)$, where d_q is sampled from $\boldsymbol{D}_q$.
4. Pad per-candidate PCs; fuse with $\mathcal{P}_t$ for AABB cropping.

Oracle RRI: Accuracy + Completeness

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

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

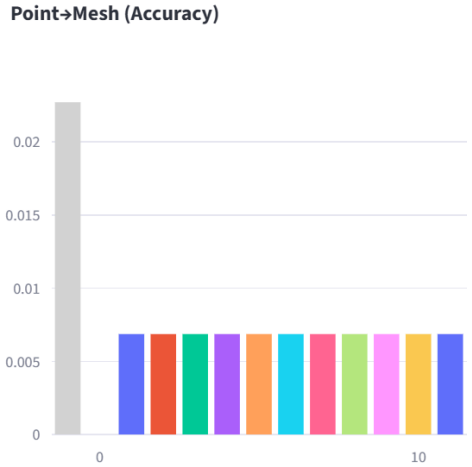


Figure 13: Candidates by accuracy (lower is better).

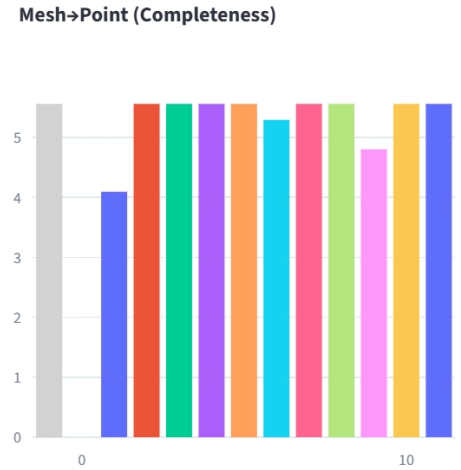


Figure 14: Candidate completeness (lower is better).

- Crop $\mathcal{M}_{\mathrm{GT}}$ to AABB, then compute $\mathcal{P}_\bullet \leftrightarrow \mathcal{M}_{\mathrm{GT}}$ distances with PyTorch3D.
- $\mathcal{A}$ dominated by $\mathcal{P}_t \longrightarrow$ not discriminative w.r.t. candidates

Oracle RRI: Relative improvement

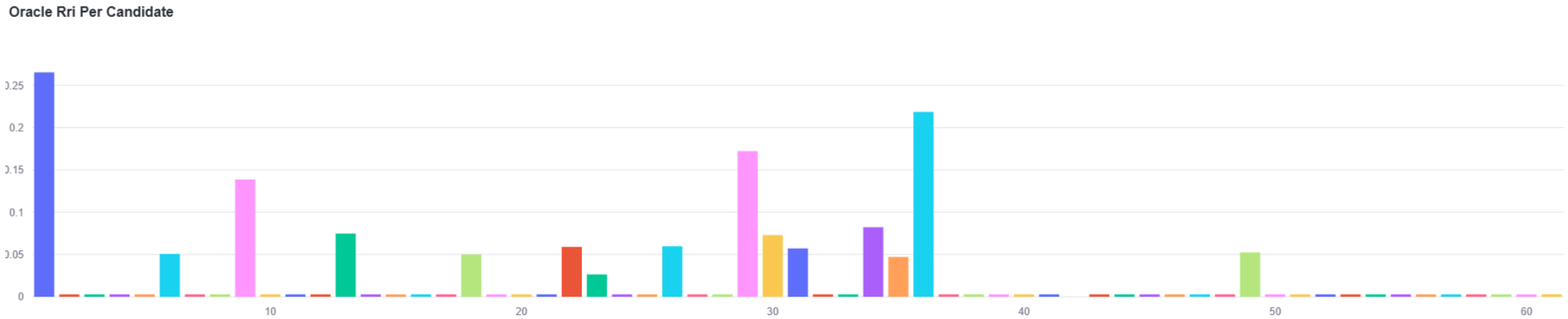


Figure 15: Per-candidate oracle RRI (bar chart).

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

Offline Dataset and Batching

Fast training iterations without recomputing the oracle

Offline cache: Motivation

Why offline?

- Oracle pipeline uses P3D rendering + backprojection mesh ops.
  1. >30s for $N_q = 60$
  2. cannot be parallelized
- Single EVL forward-pass requires:
  1. 8+ GB VRAM
  2. >60s per sample
- VIN offline store enables:
  1. batching (∇ stablization)
  2. reproducible splits & validation monitoring
  3. >180x speedup (8 min vs >24 h)
  4. enables HParam sweeps

VIN offline store: coverage + footprint

Coverage numbers

- Stored scenes: 80 / 100 (80%)
- Stored samples: 883 / 4608 (19.2%)
- Split: (train: 706, val: 177)

Storage snapshot

- EFM ATEK: 49 GB
- Oracle/backbone payloads: 263 GB (current subset).
- Minimal VIN snippets: 0.809 GB.
- Full coverage estimate: 1.37 TB

> Memory footprint dominated by voxel grid features in EVL backbone (> 90%).

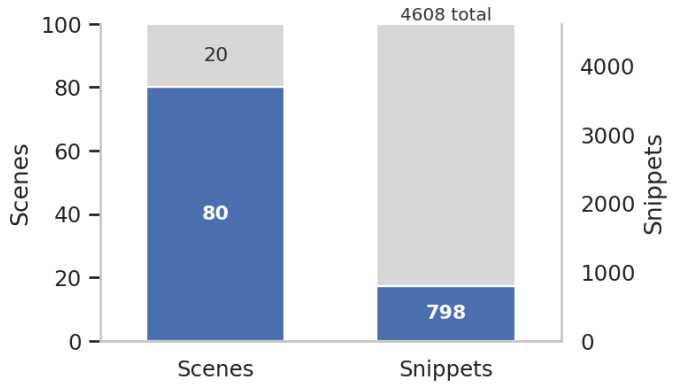


Figure 16: Coverage snapshot for the cached subset.

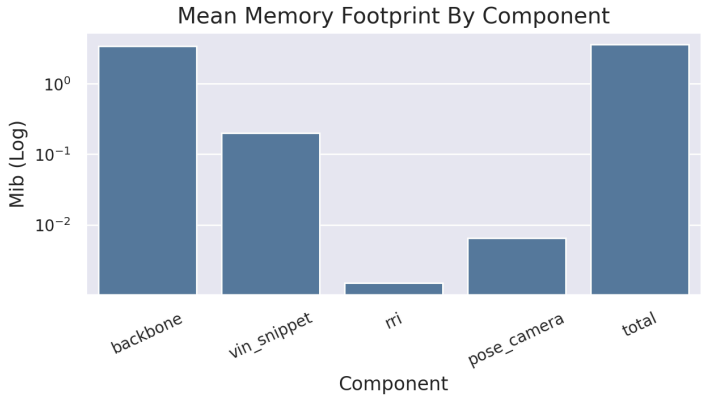


Figure 17: Per-sample memory footprint (mean).

VIN offline store: what is stored?

Store structure

- Full Oracle pipeline outputs:
  1. Candidates views $\mathcal{Q}$ + $\boldsymbol{C}_q$
  2. Depth renders $\boldsymbol{D}_q$ + valid mask
  3. Candidate PCs $\mathcal{P}_q$
  4. RRI Labels: RRI, $\mathcal{A}$, $\mathcal{C}$
  5. EVL backbone outputs
  6. VinSnippetView: $\boldsymbol{T}_{\text{rig}}^{\text{w}}(t)$ + $\mathcal{P}_t$

Training only requires RRI labels + $\mathcal{Q}$ + $\boldsymbol{C}_q$ + $\mathcal{P}_q$ + $\mathcal{P}_t$ & EVL head features.

```python
class OracleRriCacheSample:
    key: str
    scene_id: str
    snippet_id: str
    candidates: CandidateSamplingResult
    depths: CandidateDepths
    candidate_pcs: CandidatePointClouds
    rri: RriResult
    backbone_out: EvlBackboneOutput
    efm_snippet_view: EfmSnippetView
```

Figure 18: VIN offline-store sample: candidates, depths, points, and RRI.

VinOracleBatch + VinSnippetView

Key typed tensors (padded + batched)

- Candidate poses: `PoseTW[`B`,` N_q`, 12]`.
- Reference pose: `PoseTW[`B`, 12]`.
- Labels: `rri[`B`,` N_q`]` + $(\mathcal{A}, \mathcal{C})$ + lenghts.
- `PerspectiveCameras[`B`,` N_q`]`.
- VinSnippetView: `points_world[`B`,` P`, 3 +` C_{sem}`]` + `lengths[`B`]`.

```python
class VinOracleBatch:
    efm_snippet_view: EfmSnippetView | VinSnippetView
    candidate_poses_world_cam: PoseTW
    rri: Tensor
    p3d_cameras: PerspectiveCameras
    backbone_out: EvlBackboneOutput | None
```

Figure 19: VinOracleBatch Sample.

Data Flow: VinDataModule + VinOfflineDataset

Pipeline (immutable VIN offline store)

- `VinDataModule` builds a map-style `VinOfflineDataset`.
- `manifest.json` + `sample_index.jsonl` route train/val rows into immutable shards.
- Zarr blocks provide fixed tensor reads; indexed MessagePack blocks preserve diagnostics on demand.
- Each row reconstructs a `VinOracleBatch` with padded candidates, oracle RRI, cameras, and a minimal `VinSnippetView`.
- Legacy oracle/VIN caches are removed; rebuild immutable stores with `VinOfflineWriter` when data is available.

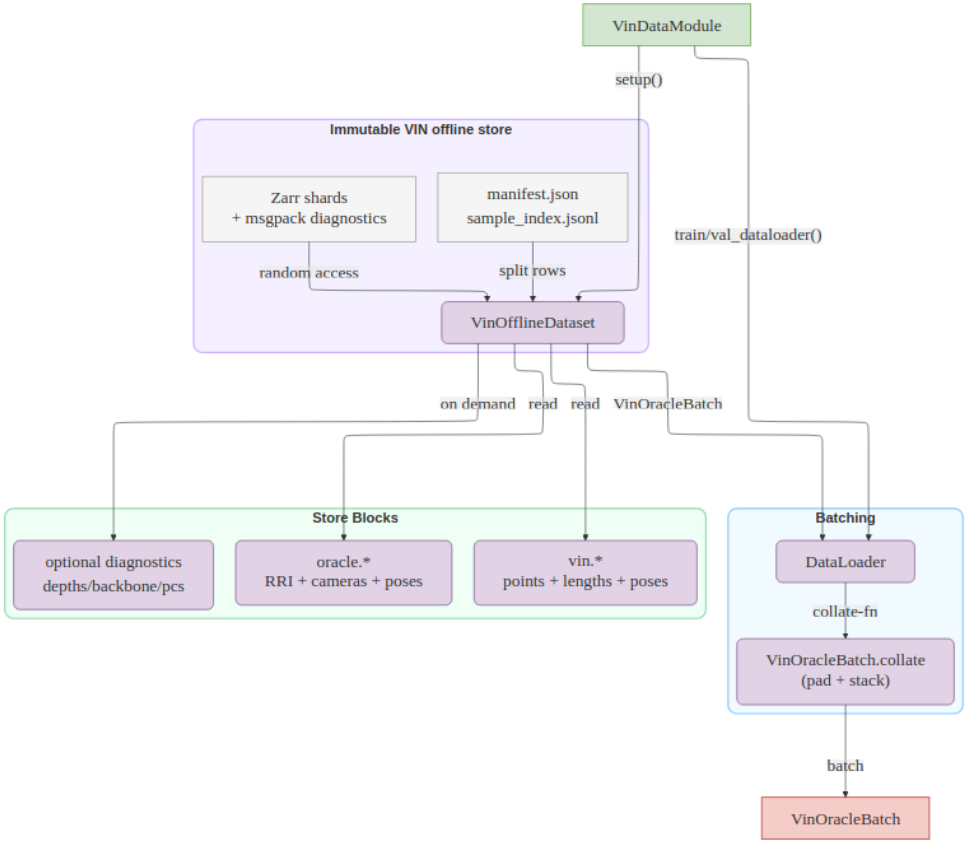


Figure 20: Data Flow: immutable VIN offline store ⟶ VinOracleBatch.

Data Flow: VinDataModule + VinOfflineDataset

Streamline, simplify, keep only necessary components

CORAL & Ordinal Binning

Skewed RRI → Quantile Bins → CORAL

Ordinal Binning

Motivation (binning + CORAL)

- Oracle RRI is heavy-tailed / right-skewed (many near-zero RRI, few large gains).
- Direct regression is sensitive to outliers and scene effects. [Fra+25]
- Use $K = 15$ ordered quantile bins + CORAL thresholds. [CMR19]
- Random Classifier:
  $\mathcal{L}_{\text{rnd}} \approx (K-1) \cdot \log(2)$?

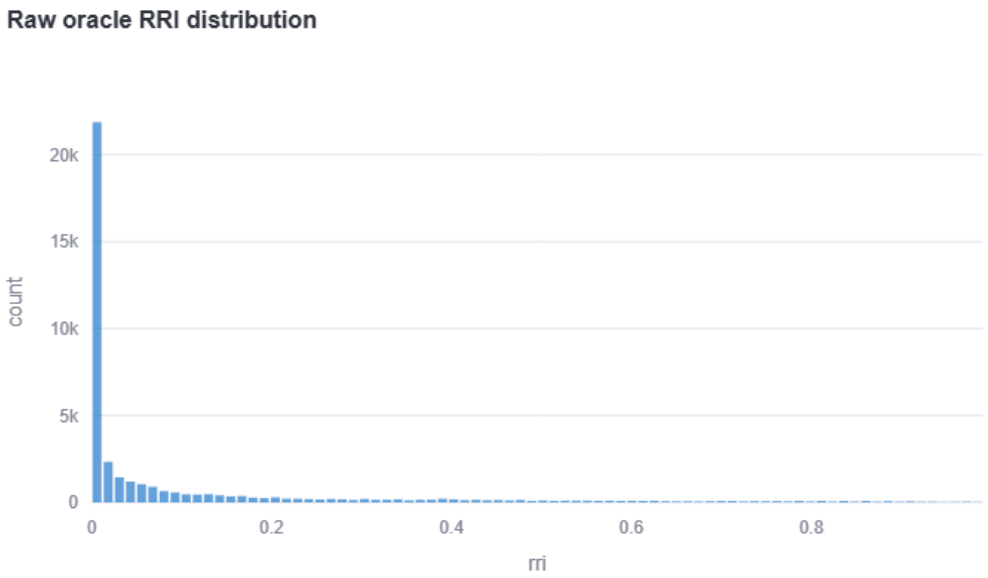


Figure 21: Raw oracle RRI distribution (linear counts).

Quantile binning (equal-mass ordinal classes)

Our Binner

- Bins define the target label $y \in \{0, ..., K-1\}$.
- Fit empirical quantiles on oracle RRIs (equal-mass bins):

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

- Assign ordinal label via edge counting (`torch.bucketize`):

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

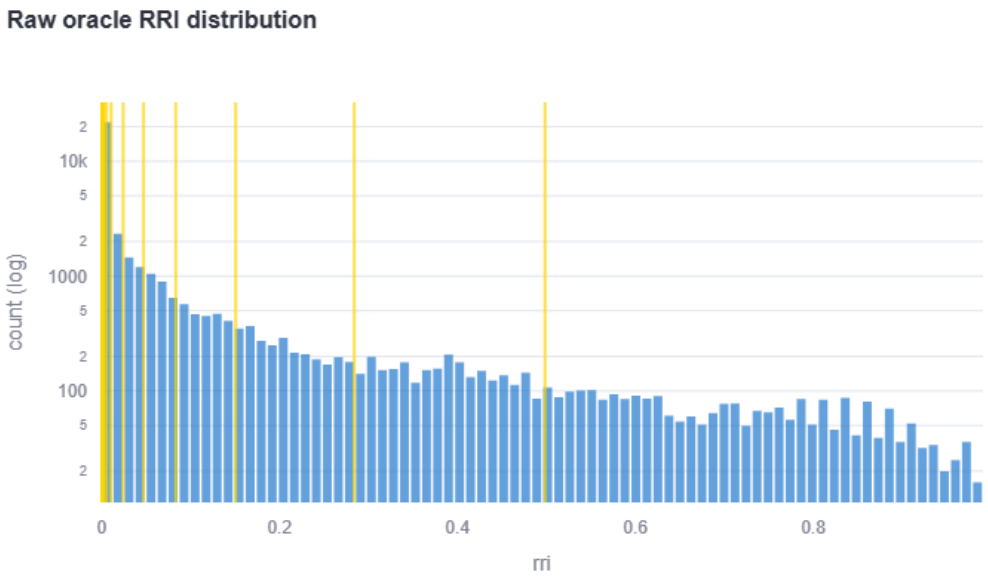


Figure 22: Log-count RRI distribution with fitted quantile edges (vertical lines).

Ordinal labels + per-bin statistics (fit data)

From labels to CORAL levels

- CORAL converts each label r into $K-1$ binary targets y:
  $t_k = \mathbb{1}[y > k], \quad k \in \{0, \ldots, K-2\}$
- Penalizes far mis-rankings more than near ones.
- Enables monotonicity diagnostics on $p_k = P(y > k)$.
- Non-uniform bin widths $\longrightarrow$ near-uniform class counts.

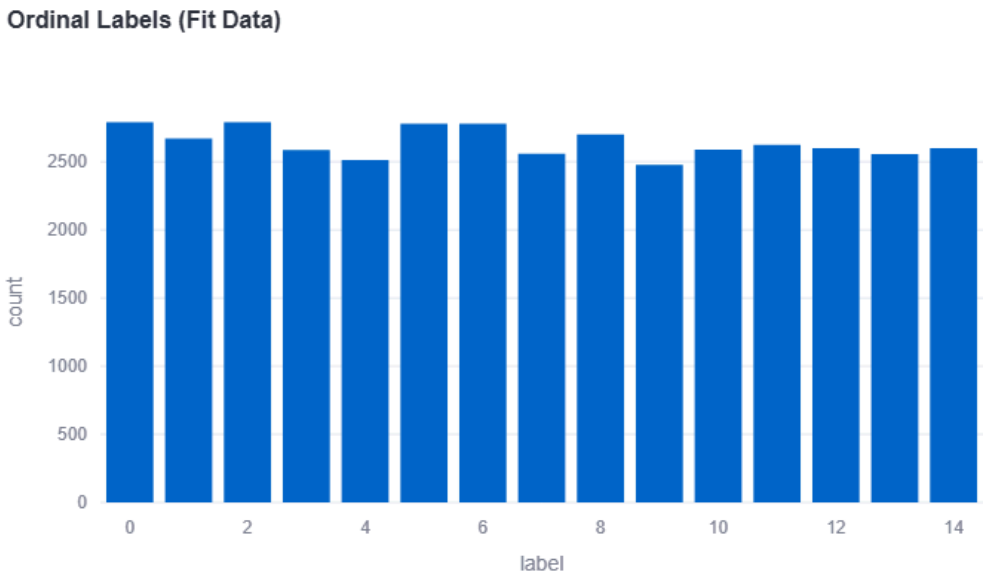


Figure 23: Label histogram after fitting K=15 quantile bins.

Bin calibration: midpoints, means, and variance

How we recover a scalar

- CORAL predicts cumulative probabilities $p_k = P(y > k)$.
- Convert to class marginals π_k and compute expectation:

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

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

- We initialize u_k (bin representatives) from fitted **bin means**.

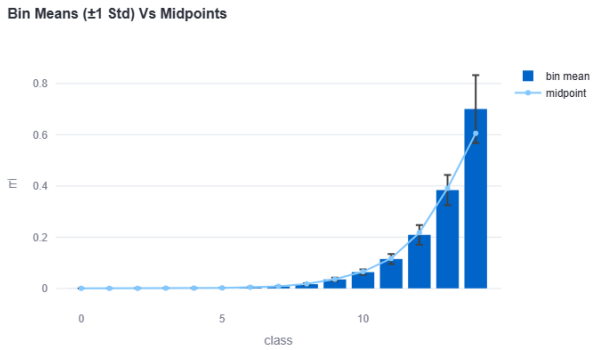


Figure 24: Bin means vs midpoints.

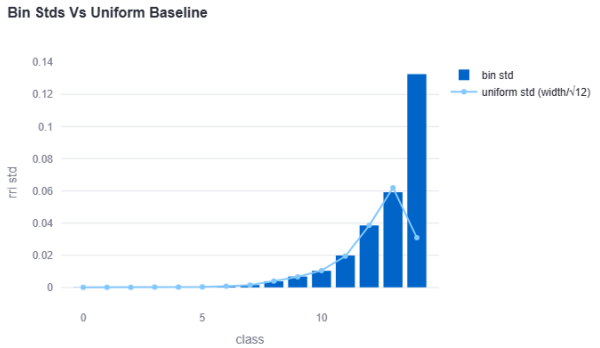


Figure 25: Per-bin std vs uniform baseline (width/12).

CORAL Implementation Deltas

What we add on top of coral-pytorch

- Reference layer + loss from [Ras25, coral-pytorch].
- Learnable monotone bin values u_k (softplus deltas) for learned expectation of RRI.
- Softplus enforces positive increments, keeping u_k ordered.
- Diagnostics: monotonicity violations and relative-to-random loss.
- Learned params: CORAL (w, b_k) and bin values (u_0, δ_j).

Key equations

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

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

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

TODO: Test against baseline!

VIN Scoring Architecture

EVL voxel context → per-candidate evidence → ordinal RRI

VIN Pipeline

Data Flow & Branches

- **Inputs**: Candidate Poses, EVL voxel field, semidense points, optional trajectory.
- **Pose branch** Candidate Poses → PoseEncodings $\boldsymbol{E}_q$.
- **Scene branch**: EVL Voxel Fields → global conditioned features.
- **Semidense branch**: $\mathcal{P}_t$ → `semidense_proj` (+ grid CNN).
- Concat → MLP → CORAL logits → expected class score (ranking proxy).
- Continuous expected RRI uses bin reps u_k (paper: Training Objective).

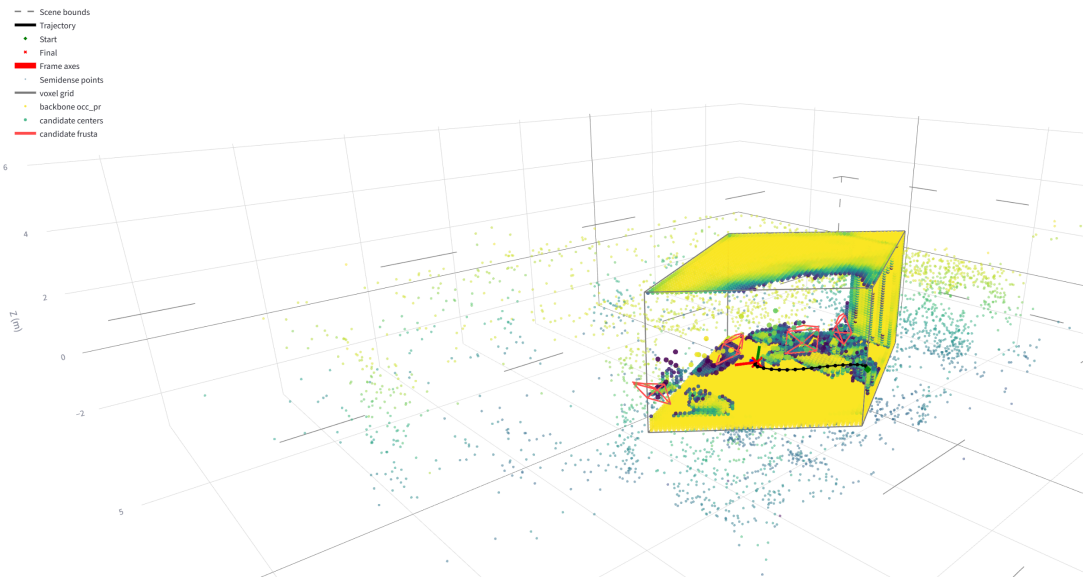


Figure 26: Superposition of all VINv3 inputs.

Candidate Pose Encoding

Concept (R6D + LFF)

- Express each candidate in the reference rig frame r: $T_{r,c_q} = T_{w,r}^{-1} \cdot T_{w,c_q}$.
- **R6D SO(3) encoding** $\longrightarrow$ stable pose vector $\left[\boldsymbol{t}_{c_q}^{r}, \mathrm{R6D}\left(\boldsymbol{R}_{c_q}^{r}\right)\right]$.
- **LFF+MLP** map pose vector to a learned embedding $\boldsymbol{E}_q$.
- $\boldsymbol{E}_q$ conditions the global scene context $\boldsymbol{g}$ and head.

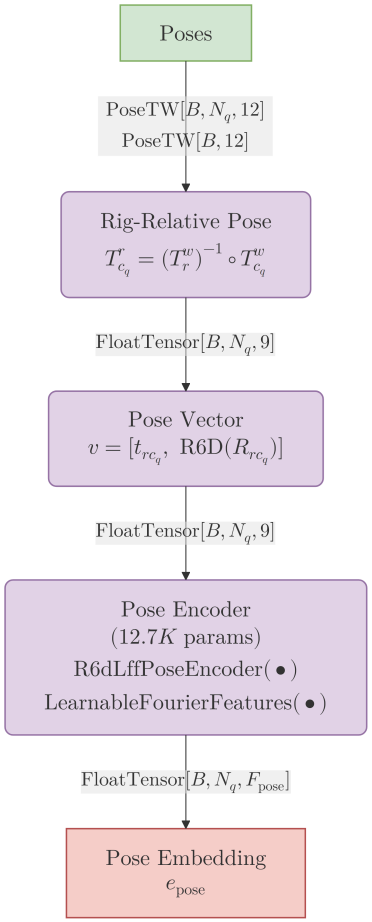


Figure 27: Candidate Pose Encoding.

Scene branch: FieldBundle (EVL voxel field)

Concept (EVL scene field)

- EVL evidence heads are stacked into the scene-field input $(\boldsymbol{F}_v^{\text{in}})$: $(\boldsymbol{V}_{\text{surf}}^{\text{in}}, \boldsymbol{V}_{\text{count}}^{\text{norm}}, \boldsymbol{V}_{\text{occ}}^{\text{pr}}, \boldsymbol{V}_{\text{cent}}^{\text{pr}})$.
- Optional derived channels augment $(\boldsymbol{F}_v^{\text{in}})$: $\boldsymbol{V}_{\text{free}}^{\text{in}}, \boldsymbol{V}_{\text{unk}}, \boldsymbol{V}_{\text{new}}$.
- Derived channel definitions:

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

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

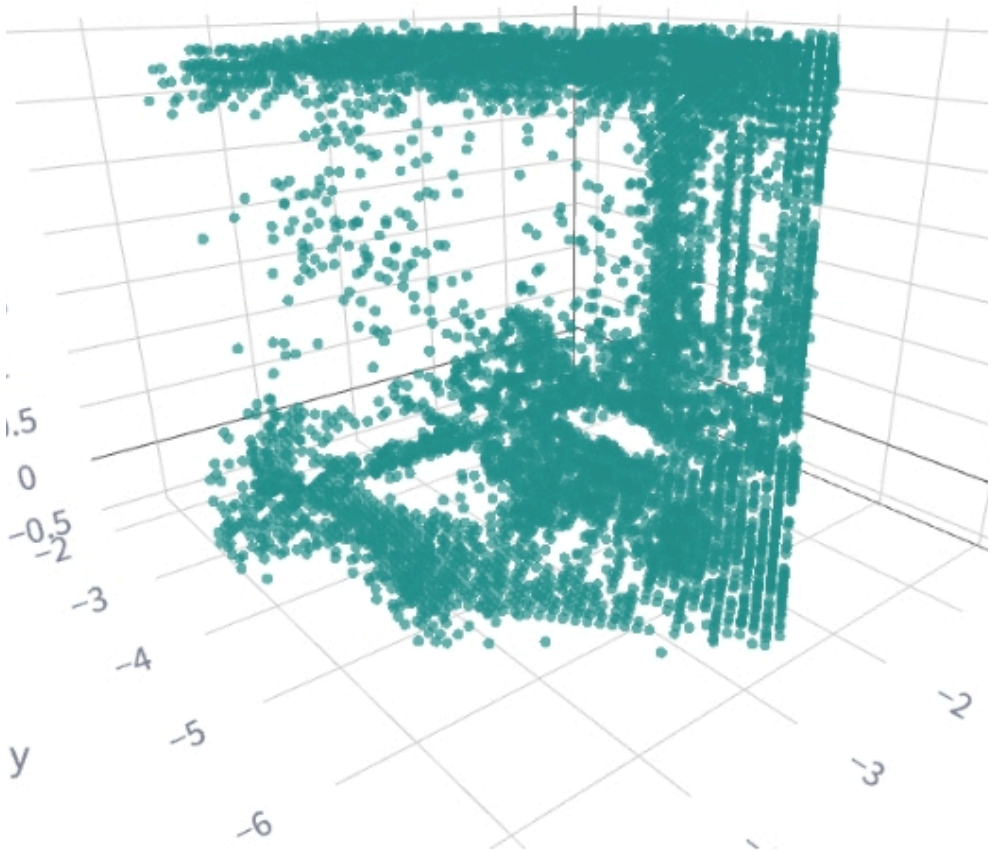


Figure 28: EVL occupancy evidence slice (scene-field input).

Scene branch: voxel_valid_frac (coverage proxy)

Concept (coverage proxy)

- Project the input field $(\boldsymbol{F}_v^{\text{in}})$ with `Conv3d + GroupNorm + GELU` to the scene field $\boldsymbol{F}_v$.
- The same voxel context drives global pooling and voxel-projection statistics (for FiLM).
- Sample $\boldsymbol{V}_{\text{count}}^{\text{norm}}$ at each candidate center $\boldsymbol{c}$ to get per-candidate voxel validity v in $[0, 1]$.
- Used for candidate validity m and coverage scheduling.

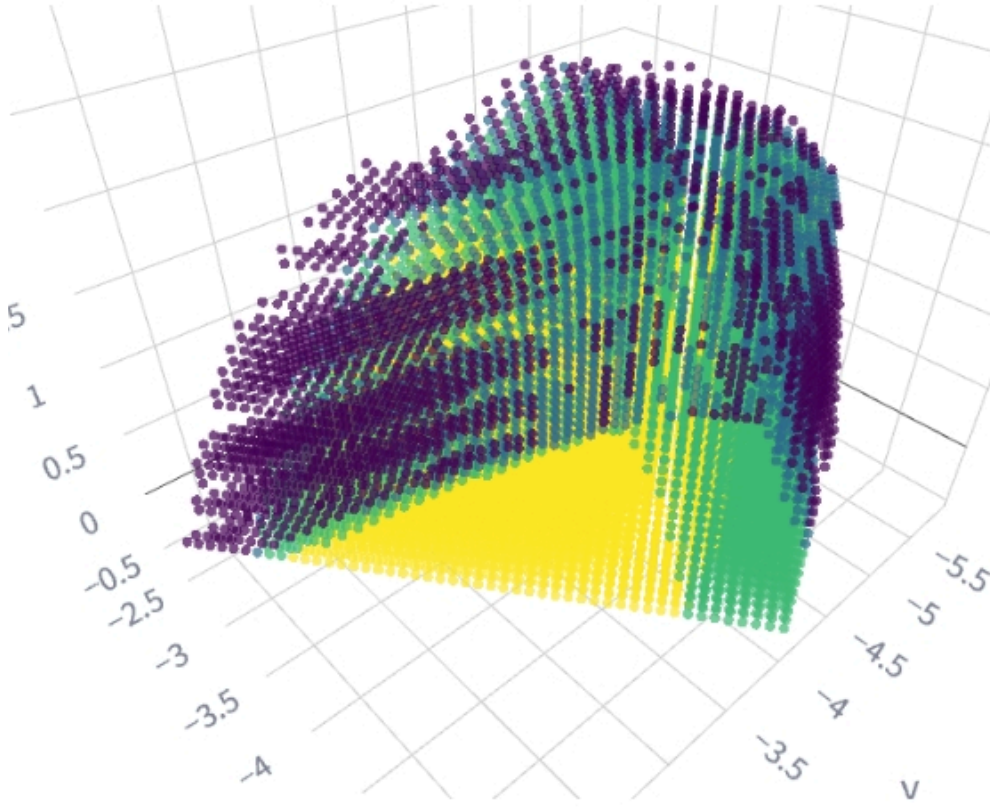


Figure 29: Normalized observation counts (coverage proxy).

Scene Branch: Global Context + FiLM

Concept (pose-conditioned pooling)

- From $\boldsymbol{F}_v$, build voxel tokens $\boldsymbol{t}$ and pool with pose queries $\boldsymbol{E}_q$ to get per-candidate global context $\boldsymbol{g}$:

$$\boldsymbol{g}_i = \mathrm{MHCA}\big(q = \boldsymbol{E}_q, k = \boldsymbol{t} + \varphi(\boldsymbol{p}), v = \boldsymbol{t}\big)$$

- In parallel, pool voxel centers and project into candidate cameras to get per-candidate projection stats (coverage/validity (visibility) + depth moments).
- FiLM uses these stats to modulate $\boldsymbol{g}$:

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

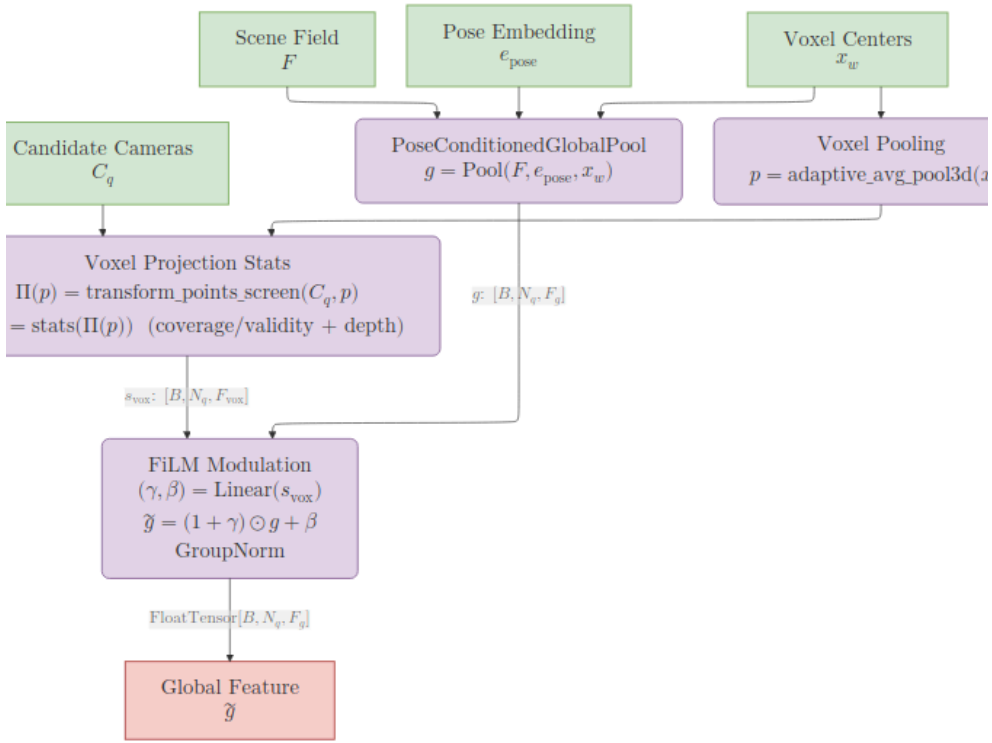


Figure 30: Global context + voxel-projection FiLM.

Semidense Branch: Scalar Stats

Concept (projection stats)

- Project $\mathcal{P}_t$ into each candidate view.
- Compute coverage, visibility, and depth moments from valid projections.
- Reliability weights combine n_{obs} and σ_ρ (inverse-distance std).
- Yields per-candidate scalar evidence $\boldsymbol{s}_{\text{proj}}$ for the head.

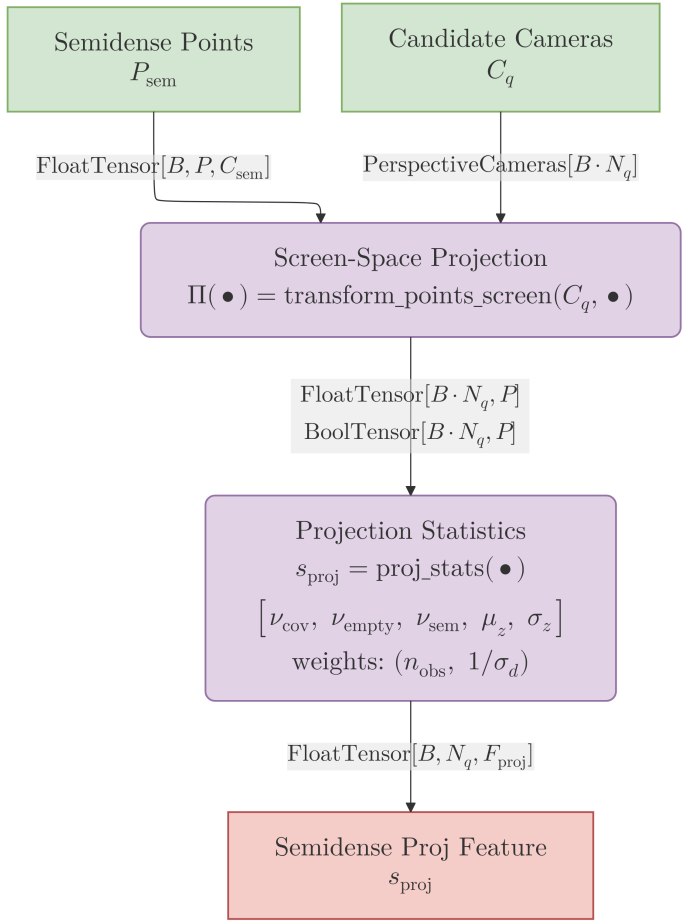


Figure 31: Semidense projection statistics branch.

Semidense Projections

Projection

- Use `transform_points_screen` to project points into the candidate camera.
- Valid points are finite, in front of the camera, and inside image bounds.

Grid binning

- Bin valid projections into a $G_{\text{sem}} \times G_{\text{sem}}$ screen grid.
- Compute per-bin visibility, mean depth, and depth std.
  $\longrightarrow$ CNN inputs.

Grid

- If H=W=120, G=12 $\longrightarrow$ each cell covers 10x10 pixels.

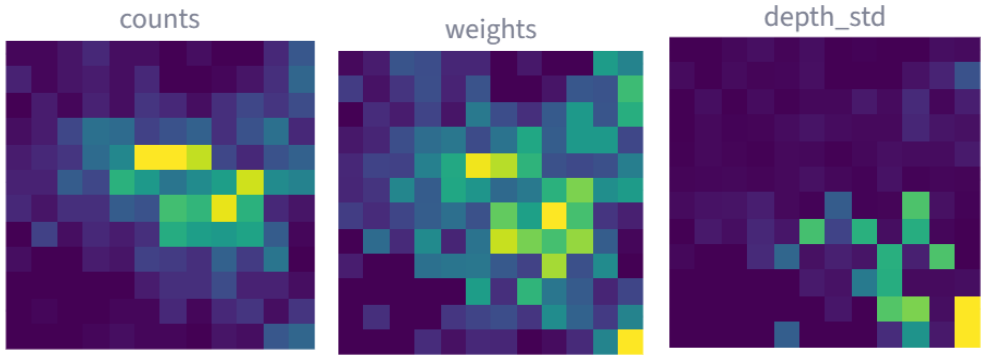


Figure 32: Semidense projection maps (counts / weights / depth std).

> Why so coarse !?

Semidense Branch: Grid CNN

Concept (projection grid)

- Keep coarse view-plane structure that scalar stats discard.
- Bin valid projections into a $G_{\text{sem}} \times G_{\text{sem}}$ grid.
- Tiny CNN encodes occupancy + depth moments into $\boldsymbol{s}_{\text{grid}}$.
- Appended to the head with $\boldsymbol{s}_{\text{proj}}$.
- Grid channels: $\boldsymbol{H}_i \in \mathbb{R}^{3 \times G_{\text{sem}} \times G_{\text{sem}}}$ with $\left[O_i, \mu_{z_i}, \sigma_{z_i}\right]$.

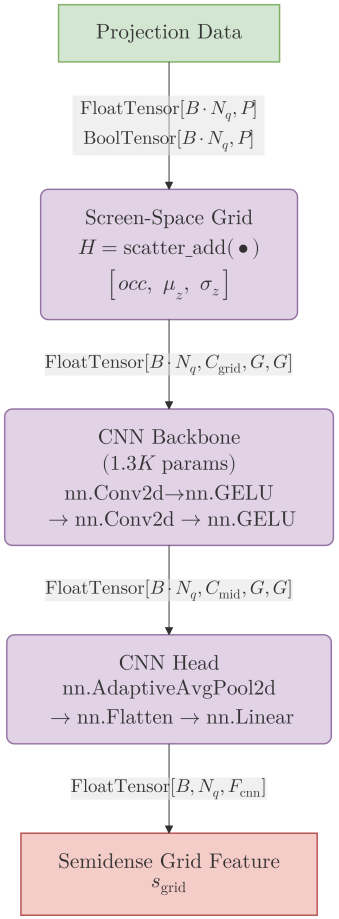


Figure 33: Semidense grid CNN branch.

MLP & Coral Head

Concept (fusion + CORAL head)

- Concatenate features:

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

- MLP scorer $\longrightarrow$ CORAL logits $\longrightarrow$ expected class score $\hat{r}$ (regression proxy).
- Continuous expected RRI uses class probs with bin reps u_k from learned u_0, δ_k.
- Naive candidate validity:

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

Candidate validity should be soft!

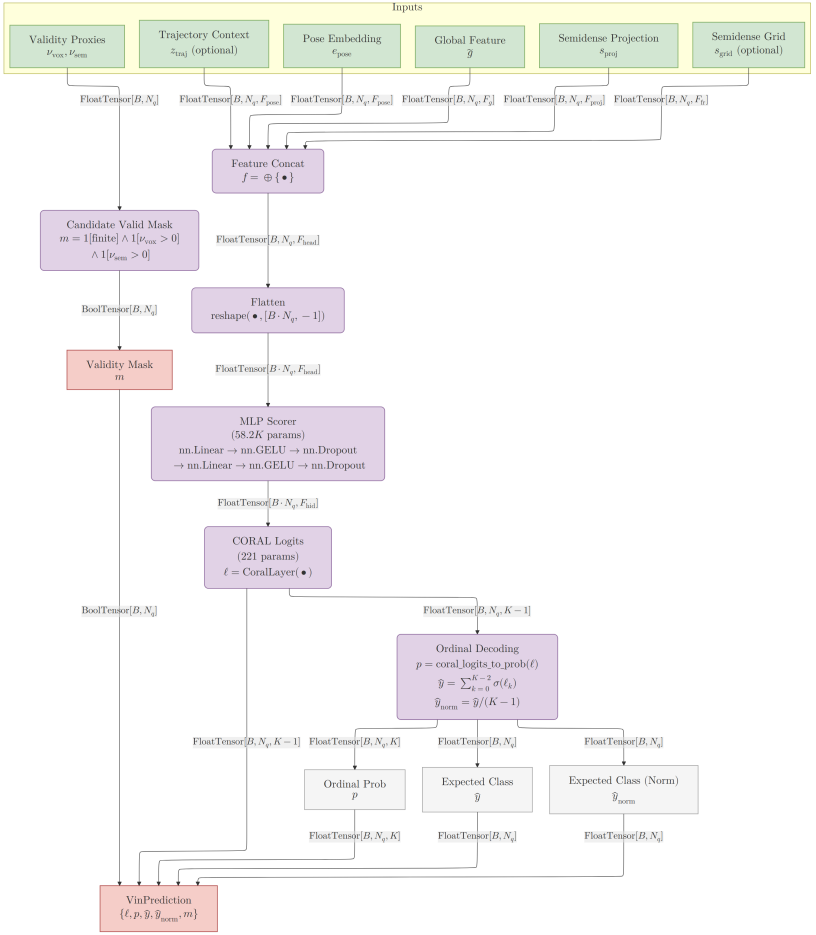


Figure 34: Feature fusion + CORAL head.

Training: Objective, Metrics & Diagnostics

Ordinal supervision + diagnostics-first monitoring

Training objective (ordinal supervision)

Objective

- CORAL loss + aux. regression loss $\mathcal{L}_{\text{reg}}$

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

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

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

- Coverage/visibility curriculum: reweight per-candidate loss by evidence $w_{i(t)} = (1 - \lambda_t) + \lambda_t(v_i \cdot v_i^{\text{sem}})$ and anneal $\lambda_t \to 0$ over training.

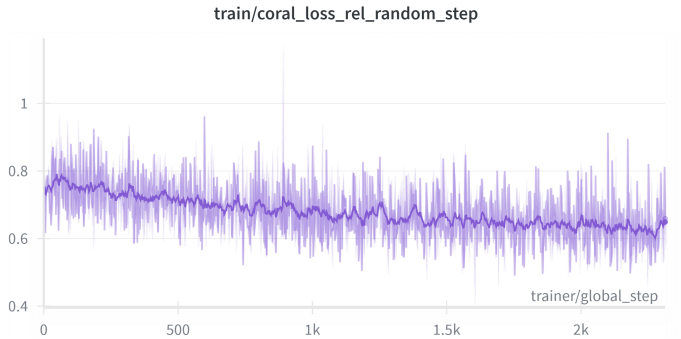


Figure 35: Training CORAL relative loss.

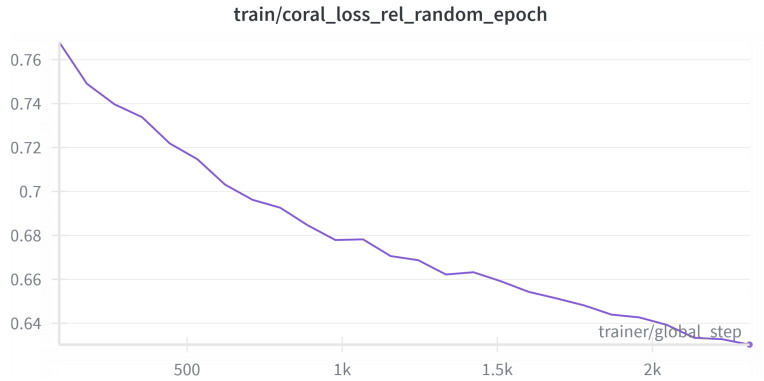


Figure 36: Epoch-level training CORAL relative loss.

Best-run curves + Metrics

Metrics (definition + intuition)

- Relative CORAL loss:

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

- Ranking agreement:

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

- Top-3 bin accuracy:

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

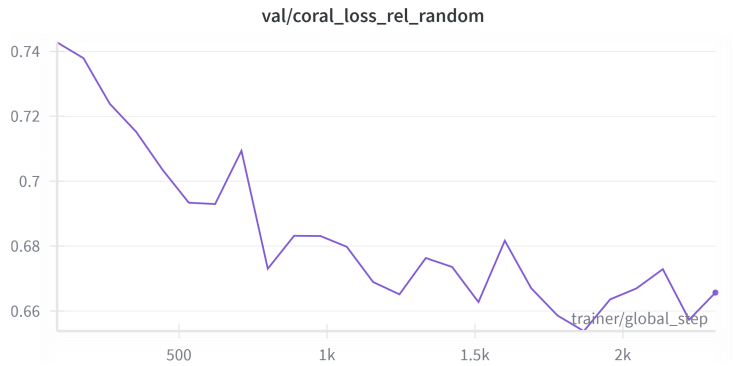


Figure 37: Relative CORAL loss.

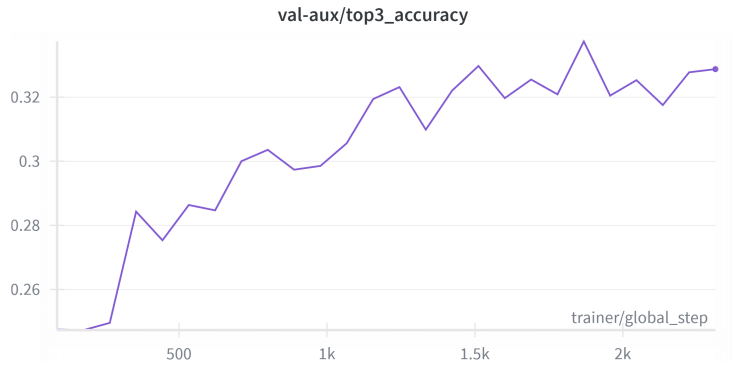


Figure 38: Top-3 bin accuracy.

Best run: start → finish summary

Metric	Start → finish
Training $\mathcal{L}_{\text{rel}}$	0.777 → 0.659 (−15.2%)
Validation $\mathcal{L}_{\text{rel}}$	0.743 → 0.666 (−10.4%)
Spearman ρ	0.254 → 0.501 (96.9%)
Top-3 TopKAcc(3)	0.248 → 0.329 (32.8%)

Calibration

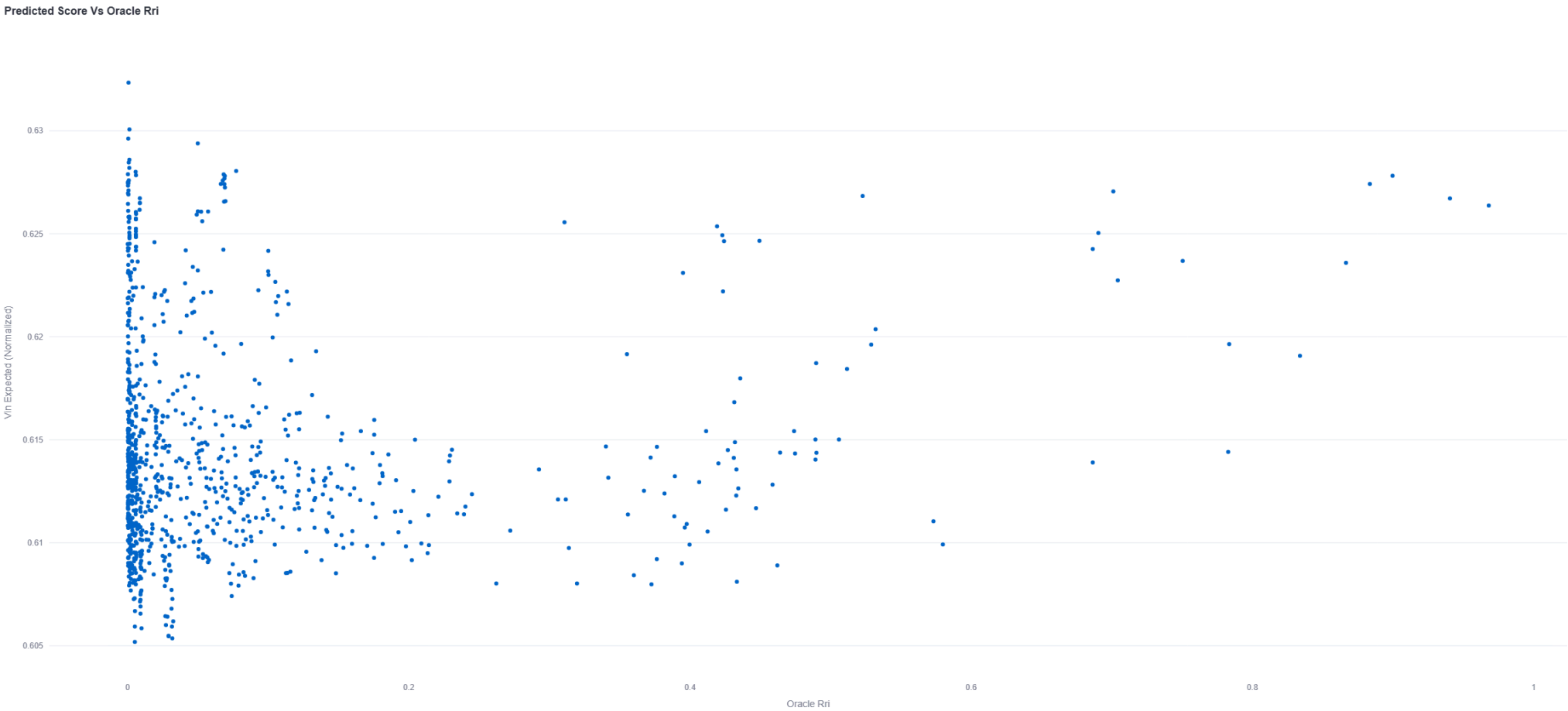


Figure 39: Predicted vs Oracle RRI scatter.

Comparison: best run vs baseline run

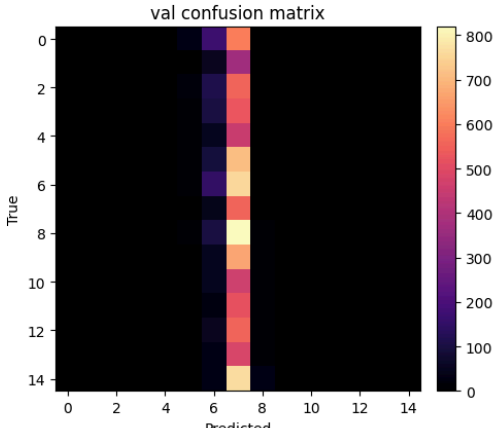
Ablation

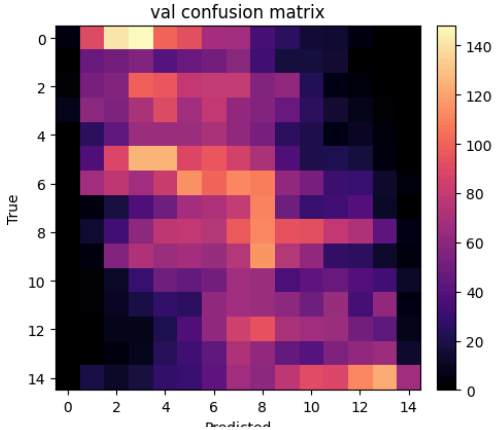
1. `OneCycleLR` ⟶ `ReduceOnPlateau`
2. No trajectory encoder
3. No auxiliary regression loss
4. Batch size 8 ⟶ 16

Final validation metrics (last logged)

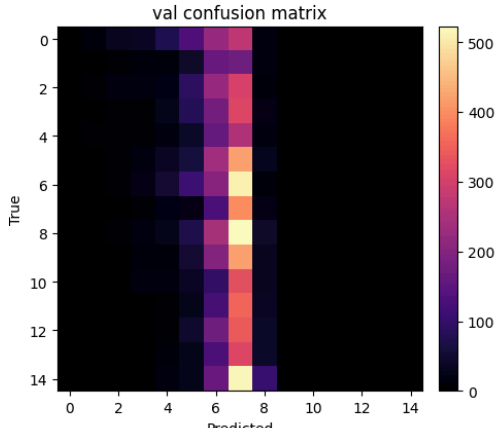
Run	$\mathcal{L}_{\text{rel}}$	ρ	TopKAcc(3)
`base`	0.677	0.469	0.314
`ablation`	0.666	0.501	0.329

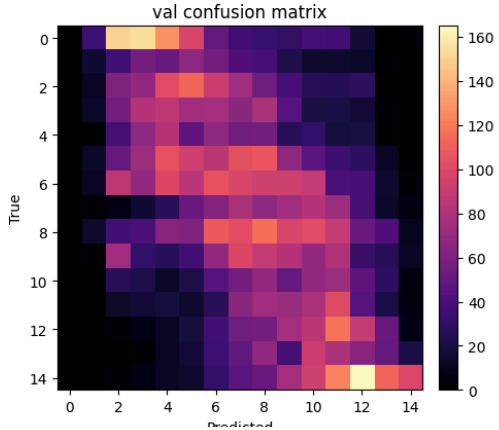
`ablation`





`base`





Key takeaways

What is solid now

- Oracle RRI pipeline is implemented end-to-end and fully functional.
- Offline cache enables fast training/debug loops with a typed batching.
- Rich training and post-hoc diagnostics.

Main limitations

- Data scale: Trained on 80 scenes and 706 / 4608 snippets (0.153%).
- Weak component-level signals: Too many DoFs have been changed at once!
- EVL backbone's Voxel Fields are too narrow (4x4x4)m

Key takeaways

Master thesis next steps

- Scale dataset (more scenes, more snippets, increase variety, **more compute**).
- Streamline OfflineCacheDataset (it's a mess)
- Run clean Optuna sweep (stationary regime; architecture toggles only).
- Strengthen evidences on different architectural choices.
- Entity-wise RRI should should be feasible now.

References

[Pap24] L. Papula, Mathematische Formelsammlung: Für Ingenieure und Naturwissenschaftler, 13th ed. Springer Fachmedien Wiesbaden, 2024. doi: 10.1007/978-3-658-45806-5.

[Met25] 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

[Fra+25] N. Frahm et al., "VIN-NBV: A View Introspection Network for Next-Best-View Selection." [Online]. Available: https://arxiv.org/abs/2505.06219

[CMR19] 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

[Ras25] Raschka Research Group, "coral-pytorch documentation." [Online]. Available: https://raschka-research-group.github.io/coral-pytorch/