Appendix

A. RF-Loc architecture details

A.1. Network Architecture Details

The RF-Loc system is built upon a dual-branch encoder architecture that concurrently processes geometric and RF signal modalities. Each modality is handled by a dedicated backbone, enabling modality-specific feature extraction while maintaining architectural consistency. In the following paragraphs, we elaborate on the core components and building blocks that constitute the overall design.

Structural encoder. Table 3 summarizes the structural backbone of RF-Loc, which follows a KPConv-FPN design [53, 54]. We operate on a structural point cloud that is first voxelized at 10 cm to reduce memory. The encoder comprises 5 hierarchical stages with voxel sizes $0.5, 1, 2, 4$, and 8 m. At stage l, we maintain a superpoint set $\mathcal{S}^{(l)} = \{\mathbf{s}_j^{(l)}\}_{j=1}^{N^{(l)}}$ with positions $\mathbf{s}_j^{(l)} \in \mathbb{R}^3$ and associated features $\mathcal{F}^{(l)} = \{\mathbf{f}_j^{(l)}\}_{j=1}^{N^{(l)}}$. KPConv uses the positions $\mathcal{S}^{(l)}$ to define local neighborhoods, with radius $r^{(l)} = 2v^{(l)}$, where $v^{(l)}$ is the voxel size at that stage.

The network is built from *Unary Blocks* and *Residual Bottleneck Blocks*. A Unary Block applies a pointwise linear projection followed by GroupNorm [63] and LeakyReLU, keeping the positions fixed:

$$\mathrm{Unary}\big(\mathcal{S}^{(l)}, \mathcal{F}^{(l)}\big) = \big(\mathcal{S}^{(l)}, \mathcal{F}'^{(l)}\big), \tag{4}$$

$$\mathbf{f}_j'^{(l)} = \delta\big(\mathrm{GN}(\phi(\mathbf{f}_j^{(l)}))\big), \tag{5}$$

where $\phi : \mathbb{R}^{C_{\text{in}}} \to \mathbb{R}^{C_{\text{out}}}$ is a linear layer, GN is GroupNorm with 32 groups, and δ is LeakyReLU with slope 0.1.

Each encoder stage consists of Residual Bottleneck Blocks that combine channel reduction, KPConv, and a shortcut. Given $(\mathcal{S}^{(l)}, \mathcal{F}^{(l)})$ with features in $\mathbb{R}^{C_{\text{in}}}$, a bottleneck block first reduces channels to $C_{\text{mid}} = C_{\text{out}}/4$ via a Unary Block:

$$\mathcal{F}_1^{(l)} = \mathrm{Unary}_{C_{\text{in}} \to C_{\text{mid}}}\big(\mathcal{F}^{(l)}\big). \tag{6}$$

The reduced features are then processed by KPConv, which aggregates over neighboring superpoints according to their positions:

$$\mathcal{F}_2^{(l)} = \mathrm{KPConv}\big(\mathcal{S}^{(l)}, \mathcal{F}_1^{(l)}; r^{(l)}, K\big), \tag{7}$$

where K is the number of kernel points and neighborhoods are defined in $\mathbb{R}^3$ around each $\mathbf{s}_j^{(l)}$. We then expand back to C_{out} channels with a second Unary Block:

$$\mathcal{F}_3^{(l)} = \mathrm{Unary}_{C_{\text{mid}} \to C_{\text{out}}}\big(\mathcal{F}_2^{(l)}\big). \tag{8}$$

The shortcut path $\mathcal{F}_{\text{skip}}^{(l)}$ adapts to changes in resolution and channel dimensionality using a single linear map $\phi_s : \mathbb{R}^{C_{\text{in}}} \to \mathbb{R}^{C_{\text{out}}}$:

$$\mathbf{f}_{\text{skip},k}^{(l)} = \begin{cases} \mathbf{f}_k^{(l)}, & \text{non-strided, } C_{\text{in}} = C_{\text{out}}, \\ \phi_s\big(\mathrm{GN}(\mathbf{f}_k^{(l)})\big), & \text{non-strided, } C_{\text{in}} \neq C_{\text{out}}, \\ \phi_s\big(\max\limits_{j \in \mathcal{N}_k} \mathbf{f}_j^{(l)}\big), & \text{strided}, \end{cases} \tag{9}$$

where $\mathcal{N}_k$ is the set of input superpoints assigned (e.g. by voxel downsampling) to the k-th output superpoint, and the max is taken elementwise over features. The final output of the Residual Bottleneck Block is

$$\mathcal{F}_{\text{out}}^{(l)} = \delta\big(\mathcal{F}_3^{(l)} + \mathcal{F}_{\text{skip}}^{(l)}\big). \tag{10}$$

Superpoint confidence and position heads. The last encoder layer provides coarse superpoints $\hat{\mathcal{P}} = \mathcal{S}^{(L)} = \{\hat{\mathbf{s}}_u\}_{u=1}^{N^{(L)}}$ with features $\hat{\mathcal{F}} = \mathcal{F}^{(L)} = \{\hat{\mathbf{f}}_u\}_{u=1}^{N^{(L)}}$, and each $\hat{\mathbf{s}}_u$ indexes a fine patch $G(\hat{\mathbf{s}}_u)$ via Eq. (3). The final decoder layer recovers fine superpoints $\tilde{\mathcal{P}} = \mathcal{S}^{(1)} = \{\tilde{\mathbf{p}}_i\}_{i=1}^{N^{(1)}}$ with features $\tilde{\mathcal{F}} = \mathcal{F}^{(2L-1)} = \{\tilde{\mathbf{f}}_i\}_{i=1}^{N^{(1)}}$.

Coarse superpoint confidence. We predict a scalar confidence $c_u \in [0, 1]$ for each coarse superpoint $\hat{\mathbf{s}}_u$. Two Residual Bottleneck Blocks are applied to $(\hat{\mathcal{P}}, \hat{\mathcal{F}})$,

$$\{\mathbf{z}_u^{\text{conf}}\}_{u=1}^{N^{(L)}} = \phi_{\text{conf}}\big(\hat{\mathcal{P}}, \hat{\mathcal{F}}\big), \tag{11}$$

followed by a small MLP with sigmoid activation:

$$c_u = \sigma\big(\psi_{\text{conf}}(\mathbf{z}_u^{\text{conf}})\big), \tag{12}$$

where ψ_{conf} maps each $\mathbf{z}_u^{\text{conf}}$ to a scalar logit and $\sigma(\cdot)$ denotes the logistic sigmoid.

Gated superpoint repositioning (coarse and fine). To refine the superpoint positions at both coarse and fine scales, we apply a shared head architecture, instantiated at levels $\ell \in \{L, 1\}$ with superpoints $\mathcal{S}^{(\ell)} = \{\mathbf{s}_k^{(\ell)}\}$ and features $\mathcal{F}^{(\ell)} = \{\mathbf{f}_k^{(\ell)}\}$. Two Residual Bottleneck Blocks with KPConv take the positions and features as input and produce

$$\{\mathbf{z}_k^{(\ell)}\}_k = \phi_{\text{off}}^{(\ell)}\big(\mathcal{S}^{(\ell)}, \mathcal{F}^{(\ell)}\big), \tag{13}$$

which are fed to a linear head to regress bounded 3D offsets

$$\boldsymbol{\delta}_k^{(\ell)} = \tanh\big(\psi_{\text{off}}^{(\ell)}(\mathbf{z}_k^{(\ell)})\big)\, \alpha_{\max}^{(\ell)}, \tag{14}$$

where $\psi_{\text{off}}^{(\ell)} : \mathbb{R}^D \to \mathbb{R}^3$ and $\alpha_{\max}^{(\ell)}$ is a scale-dependent maximum shift (8 m at the coarse level $\ell = L$, 1 m at the fine level $\ell = 1$). In parallel, a small MLP predicts a scalar gate $\beta_k^{(\ell)} \in [0, 1]$:

$$r_k^{(\ell)} = \|\boldsymbol{\delta}_k^{(\ell)}\|_2, \tag{15}$$

$$\beta_k^{(\ell)} = \sigma\big(\psi_{\text{gate}}^{(\ell)}([\mathbf{z}_k^{(\ell)}, r_k^{(\ell)}])\big), \tag{16}$$

with $[\cdot,\cdot]$ denoting concatenation. The effective offset and refined position at level ℓ are

$$\tilde{\boldsymbol{\delta}}_k^{(\ell)} = \beta_k^{(\ell)}\,\boldsymbol{\delta}_k^{(\ell)}, \qquad \tilde{\mathbf{s}}_k^{(\ell)} = \mathbf{s}_k^{(\ell)} + \tilde{\boldsymbol{\delta}}_k^{(\ell)}. \tag{17}$$

We instantiate this head at $\ell = L$ (coarse superpoints) and $\ell = 1$ (fine superpoints) and use the refined positions $\tilde{\mathbf{s}}_k^{(L)}$ and $\tilde{\mathbf{s}}_k^{(1)}$ as geometric points for the subsequent matching and registration stages.

Structural decoder. The decoder mirrors the encoder with three KPConv-based upsampling stages. Starting from the coarsest features with 2048 channels, we successively upsample and refine them at voxel sizes $4, 2$, and $1\,\mathrm{m}$ using a KPConv layer followed by a Unary Block at each stage, reducing the feature dimensionality from $2048 \to 1024 \to 512 \to 256$. A final linear layer maps the finest-level features to a 256-dimensional structural descriptor at $1\,\mathrm{m}$ resolution, which is used for coarse-to-fine matching and fusion with RF.

RF encoder. The *RF encoder* maps sparse K-dimensional RF fingerprints to D-dimensional descriptors at RF superpoints and consists of: (i) a fingerprint encoder, (ii) a neighborhood aggregator with multihead attention, and (iii) a projection head. Unless stated otherwise, we use $D_f = 256$, $D_{\text{tok}} = 256$ and $D = 256$.

(1) Fingerprint encoder. Each RF point i has a normalized RSSI vector $\mathbf{x}_i \in \mathbb{R}^K$ and a binary mask $\mathbf{m}_i \in \{0,1\}^K$. We apply per-endpoint IDF weights $\mathrm{IDF} \in \mathbb{R}^K$, computed once per map as $\mathrm{IDF}_k = \log\big((N+1)/(1+n_k)\big)$ with n_k the number of points where endpoint k is visible. The masked input is

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i \odot \mathbf{m}_i \odot \mathrm{IDF}. \tag{18}$$

It is passed through a three-layer MLP

$$\tilde{\mathbf{h}}_i = \phi_{\mathrm{RF}}(\tilde{\mathbf{x}}_i), \tag{19}$$

where

$$\begin{aligned}\phi_{\mathrm{RF}} : \mathrm{Linear}(K, 512) &\to \mathrm{LeakyReLU} \to \mathrm{LayerNorm} \\ &\to \mathrm{Linear}(512, 256) \to \mathrm{LeakyReLU} \to \mathrm{LayerNorm} \\ &\to \mathrm{Linear}(256, D_f),\end{aligned} \tag{20}$$

with LeakyReLU slope 0.1. To stabilize very sparse inputs, we use a mask-dependent gate

$$\mathbf{g}_i = \sigma(W_g \mathbf{m}_i), \tag{21}$$

$$\mathbf{e}_i = \tilde{\mathbf{h}}_i \odot \mathbf{g}_i, \tag{22}$$

where $W_g \in \mathbb{R}^{D_f \times K}$ and $\mathbf{e}_i \in \mathbb{R}^{D_f}$ is the RF point descriptor.

(2) RF superpoints and fingerprint aggregator. We define RF superpoints as voxel centers of a regular $0.5 \times 0.5 \times 0.5\,\mathrm{m}$ grid over the scene. Let $\{\mathbf{q}_j\}_{j=1}^M$ be the positions of these RF superpoints and $\{\mathbf{p}_i^{\mathrm{RF}}, \mathbf{e}_i\}_{i=1}^N$ the RF points and their descriptors. For each RF superpoint j we precompute a fixed-length neighbor list $\mathcal{N}(j)$ of size T, containing RF points within a radius $r_{\max}$ (we use $r_{\max} = 2.5\,\mathrm{m}$).

For each neighbor (j, i) we compute the distance

$$d_{ji} = \big\|\mathbf{p}_i^{\mathrm{RF}} - \mathbf{q}_j\big\|_2, \tag{23}$$

and embed it with R Gaussian radial basis functions:

$$\boldsymbol{\psi}(d_{ji})_r = \exp\!\big(-\beta_r (d_{ji} - \mu_r)^2\big), \quad r = 1, \ldots, R. \tag{24}$$

We use $R = 6$ bins, with centers

$$\mu_r = \frac{r-1}{R-1}\, r_{\max}, \tag{25}$$

initialized uniformly on $[0, r_{\max}]$ and learned during training. The scales β_r are also learned and constrained to be positive via a softplus parameterization.

Table 3. The Structural Backbone Architecture

Structural Backbone		
Stage	**Encoder**	**Voxel (m)**
1	KPConv(1→64) ResBlock(64→128)	0.5
2	ResBlock(128→128, strided) ResBlock(128→256)	1
3	ResBlock(256→256, strided) ResBlock(256→256)	2
4	ResBlock(256→512, strided) ResBlock(512→512) ResBlock(512→1024)	4
5	ResBlock(1024→1024, strided) ResBlock(1024→2048) ResBlock(2048→2048)	8
Stage	**Decoder**	**Voxel (m)**
1	Linear(2048→2048)	
2	KPConv(2048 → 1024) UnaryBlock(1024→1024)	4
3	KPConv(1024 → 512) UnaryBlock(512→512)	2
4	KPConv(512 → 256) UnaryBlock(256→256)	1
5	Linear(256→256)	

The token for neighbor (j, i) is obtained by concatenation

$$\mathbf{u}_{ji} = \begin{bmatrix} \mathbf{e}_i \\ \boldsymbol{\psi}(d_{ji}) \end{bmatrix} \in \mathbb{R}^{D_f+R}, \tag{26}$$

and projecting with a two-layer MLP

$$\mathbf{t}_{ji}^{(0)} = \phi_{\text{tok}}(\mathbf{u}_{ji}), \tag{27}$$

$$\phi_{\text{tok}} : \text{Linear}(D_f + R, D_{\text{tok}}) \to \text{LeakyReLU} \tag{28}$$

$$\to \text{Linear}(D_{\text{tok}}, D_{\text{tok}}), \tag{29}$$

with $D_{\text{tok}} = 256$ and LeakyReLU slope 0.1.

We refine tokens using $L_{\text{tok}} = 2$ residual feed-forward blocks:

$$\mathbf{t}_{ji}^{(\ell+1)} = \mathbf{t}_{ji}^{(\ell)} + \text{FFN}\Big(\text{LayerNorm}(\mathbf{t}_{ji}^{(\ell)})\Big), \quad \ell = 0, 1, \tag{30}$$

where $\text{FFN} : \mathbb{R}^{D_{\text{tok}}} \to \mathbb{R}^{D_{\text{tok}}}$ is a two-layer MLP

$$\text{FFN}(\mathbf{v}) = W_4\, \delta\big(W_3 \mathbf{v}\big), \tag{31}$$

$$W_3 \in \mathbb{R}^{D_{\text{tok}} \times D_{\text{tok}}}, \quad W_4 \in \mathbb{R}^{D_{\text{tok}} \times D_{\text{tok}}}, \tag{32}$$

with LeakyReLU activation δ (we use the same structure for all occurrences of FFN, with independent parameters).

Pooling over neighbors at each RF superpoint is performed by a single-query multihead attention module with $H = 4$ heads. We initialize the pooled RF state with a randomly initialized trainable vector $\mathbf{p}_{\text{init}} \in \mathbb{R}^{D_{\text{tok}}}$, shared across all superpoints, and set $\mathbf{p}_j^{(0)} = \mathbf{p}_{\text{init}}$ for all j. We then apply $L_{\text{loop}} = 2$ attention–FFN layers:

$$\mathbf{q}_j^{(\ell)} = \text{LayerNorm}\big(\mathbf{p}_j^{(\ell)}\big), \tag{33}$$

$$\Delta\mathbf{p}_j^{(\ell)} = \text{MHA}\Big(\mathbf{q}_j^{(\ell)}, \{\mathbf{t}_{ji}\}_{i \in \mathcal{N}(j)}\Big), \tag{34}$$

$$\hat{\mathbf{p}}_j^{(\ell)} = \mathbf{p}_j^{(\ell)} + \Delta\mathbf{p}_j^{(\ell)}, \tag{35}$$

$$\mathbf{p}_j^{(\ell+1)} = \hat{\mathbf{p}}_j^{(\ell)} + \text{FFN}\Big(\text{LayerNorm}(\hat{\mathbf{p}}_j^{(\ell)})\Big), \tag{36}$$

for $\ell = 0, \ldots, L_{\text{loop}} - 1$. This yields a pooled state $\mathbf{p}_j^{(L_{\text{loop}})} \in \mathbb{R}^{D_{\text{tok}}}$ for each RF superpoint j.

The final RF descriptor for superpoint j is obtained via

$$\mathbf{z}_j = W_{\text{out},2}\, \delta\Big(W_{\text{out},1}\, \text{LayerNorm}\big(\mathbf{p}_j^{(L_{\text{loop}})}\big)\Big), \tag{37}$$

with $W_{\text{out},1} \in \mathbb{R}^{2D_{\text{tok}} \times D_{\text{tok}}}$, $W_{\text{out},2} \in \mathbb{R}^{D \times 2D_{\text{tok}}}$ and $D = 256$.

Local RF–Structural Fusion. Given the coarse structural $\mathcal{P}^S$ and RF superpoints $\mathcal{P}^{\text{RF}}$, the *Local RF–Structural Fusion* layer fuses RF descriptors into the structural branch using local RF neighborhoods and single-query multihead attention.

(1) Local RF neighborhood encoding. For each structural point p_i^S and RF neighbor $p_j^{\text{RF}} \in \mathcal{N}_i^{\text{RF}}$ (with at most $K = 128$ neighbors within radius $r_{\max}$), we compute

$$\mathbf{u}_{ij} = p_j^{\text{RF}} - p_i^S, \quad d_{ij} = \big\|\mathbf{u}_{ij}\big\|_2, \quad \hat{\mathbf{u}}_{ij} = \frac{\mathbf{u}_{ij}}{\|\mathbf{u}_{ij}\|_2}. \tag{38}$$

Distances are embedded with a Gaussian RBF bank

$$\boldsymbol{\phi}_{\text{rbf}}(d_{ij})_r = \exp\big(-\beta_r (d_{ij} - \mu_r)^2\big), \quad r = 1, \ldots, R, \tag{39}$$

where we use $R = 8$ bins and learnable centers μ_r and scales β_r (positivity enforced via softplus). The geometric encoding is

$$\mathbf{g}_{ij} = \Big[\hat{\mathbf{u}}_{ij},\ \log(1 + d_{ij}),\ \boldsymbol{\phi}_{\text{rbf}}(d_{ij})\Big], \tag{40}$$

and we concatenate RF and geometry features

$$\mathbf{x}_{ij} = \Big[\mathbf{z}_j^{\text{RF}},\ \mathbf{g}_{ij}\Big]. \tag{41}$$

A small MLP maps $\mathbf{x}_{ij}$ to 256-d RF tokens:

$$\tilde{\mathbf{x}}_{ij} = \phi_{\text{loc}}(\mathbf{x}_{ij}), \tag{42}$$

$$\phi_{\text{loc}} : \text{Linear}(d_{\text{rf}} + d_{\text{pos}}, 256) \to \text{LayerNorm} \to \text{LeakyReLU}$$

$$\to \text{Linear}(256, 256) \to \text{LeakyReLU}, \tag{43}$$

with $d_{\text{pos}} = 3 + 1 + R$ and LeakyReLU slope 0.1.

(2) RF-to-structural attention pooling. Each structural descriptor $\mathbf{z}_i^S$ provides a query

$$\mathbf{q}_i = W_q\, \mathbf{z}_i^S, \quad W_q \in \mathbb{R}^{256 \times d_{\text{str}}}, \tag{44}$$

while RF tokens are projected to keys and values

$$\mathbf{k}_{ij} = W_{\text{kv}}\, \tilde{\mathbf{x}}_{ij}, \qquad \mathbf{v}_{ij} = \mathbf{k}_{ij}, \tag{45}$$

with $W_{\text{kv}} \in \mathbb{R}^{256 \times 256}$. A single multihead attention layer (4 heads, masked on invalid neighbors) is applied per i, using $\mathbf{q}_i$ as query and $\{\mathbf{k}_{ij}, \mathbf{v}_{ij}\}_{j \in \mathcal{N}_i^{\text{RF}}}$ as keys/values, yielding an attention output $\mathbf{h}_i^{\text{att}} \in \mathbb{R}^{256}$. The aggregated RF descriptor in the fusion space is

$$\mathbf{h}_i^{\text{RF}} = W_{\text{out}}\, \mathbf{h}_i^{\text{att}}, \quad W_{\text{out}} \in \mathbb{R}^{d_{\text{out}} \times 256}, \tag{46}$$

with $d_{\text{out}} = 256$ in our experiments.

(3) Proximity-based confidence and residual gating. RF proximity around p_i^S is summarized as

$$w_{ij}^{\text{prox}} = \exp(-\gamma_{\text{prox}} d_{ij}), \tag{47}$$

$$\bar{w}_i^{\text{prox}} = \frac{1}{|\mathcal{N}_i^{\text{RF}}|} \sum_{j \in \mathcal{N}_i^{\text{RF}}} w_{ij}^{\text{prox}}, \tag{48}$$

with $\gamma_{\text{prox}} = 0.25$. A scalar confidence is predicted from $\bar{w}_i^{\text{prox}}$ via a two-layer MLP (64 hidden units, sigmoid output),

and set to zero if fewer than min_valid $= 8$ RF neighbors are present. We then learn a residual gate

$$g_i^{\text{res}} = \sigma\Big(W_{g,2}\,\delta\big(W_{g,1}\,[\mathbf{z}_i^S, \mathbf{h}_i^{\text{RF}}]\big)\Big), \tag{49}$$

with a 64-dimensional hidden layer and LeakyReLU(0.1), and define the effective scalar gate $g_i = c_i\, g_i^{\text{res}}$.

Finally, structural descriptors are projected to the fusion space and updated:

$$\tilde{\mathbf{z}}_i^S = W_{\text{fuse}}\,\mathbf{z}_i^S, \quad W_{\text{fuse}} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{str}}}, \tag{50}$$

$$\mathbf{z}_i^{\text{fuse}} = \text{LayerNorm}\big(\tilde{\mathbf{z}}_i^S + g_i\,\mathbf{h}_i^{\text{RF}}\big). \tag{51}$$

The layer outputs fused descriptors $\mathbf{z}_i^{\text{fuse}} \in \mathbb{R}^{d_{\text{out}}}$.

Superpoint Matching Module. The *Superpoint Matching Module* presented in Table 4, receives the fused feature vectors along with their corresponding superpoints and adopts the Geometric Transformer architecture proposed in [39]. Within the self-attention stages, sinusoidal embeddings are used to encode both positional and angular relationships between superpoints. An initial linear projection reduces the input feature dimensionality to 128, followed by six Geometric Transformer layers—comprising three with relative positional embeddings (RPE) and three standard (vanilla) layers. A final linear projection produces a 256-dimensional descriptor for matching.

Fine Matching Module. Matching is refined using a learnable optimal transport [8, 26, 38, 45] solver with 100 iterations. To enhance robustness, dustbin masking and mutual filtering are applied. The resulting correspondences are further refined through a weighted Procrustes alignment [19], performed over 5 iterations to estimate the final transformation between point clouds. The matching scores are used as weights during the alignment process.

A.2. Training details

Implementation details. RF-Loc is implemented and evaluated in PyTorch [36]. Training follows a four-stage strategy that combines progressive structural pretraining, separate RF encoder learning, cross-modal fusion, and final end-to-end fine-tuning.

Stage 1: structural backbone and matching. We first train the structural KPConv-FPN backbone jointly with the coarse superpoint matching and fine point matching modules, using only structural input. To make full-map training feasible, we progressively increase the global map context. For each local map, we compute the bounding box of the structural points, obtain its center, transform this center to the global frame using the ground-truth pose, and extract a cubic crop from the global structural map. We start with a $60 \times 60 \times 60$ m crop, then expand to $100 \times 100 \times 100$ m, and finally train on the full global map. This curriculum allows the network to gradually learn to handle large spatial context while keeping memory usage manageable. The first two context phases are trained on NVIDIA L40S GPUs, whereas full-map training requires NVIDIA H100 GPUs due to the increased memory footprint.

Local structural maps are precomputed offline and segmented into overlapping trajectory chunks of lengths 5, 10, 15, and 20 m, with an overlap of one third of the chunk length. During structural training, we apply the following augmentations to both local and global structural point clouds: independent Gaussian noise $\mathcal{N}(0, 0.05)$ m on 3D coordinates; random scaling with a factor sampled uniformly from $[0.8, 1.2]$; random shift of the cropping center by a vector drawn from $\mathcal{N}(0, 5)$ m; independent point dropout with probability 0.2; and a consistent random 3D rotation of local maps and their ground-truth poses, with yaw $\theta_z \sim \mathcal{U}(-180°, 180°)$, pitch $\theta_y \sim \mathcal{U}(-30°, 30°)$, and roll $\theta_x \sim \mathcal{U}(-10°, 10°)$. To mimic realistic sensing conditions, we corrupt the rendered ground-truth depth images with a stereo noise model $\sigma_z(z) = \alpha z^2 + \beta$ (with $\alpha = 0.001$ and $\beta = 0.01$) and reconstruct the structural point clouds from the noisy depths. The structural backbone is trained on the three context sizes for 100, 100, and 150 epochs, respectively, with a batch size of 1 per GPU. We use Adam [28] with an initial learning rate of 1×10^{-4}, exponential decay by a factor of 0.95 every 4 epochs, and weight decay 1×10^{-6}.

Stage 2: RF encoder pre-training. In the second stage, the RF encoder is trained *entirely separately* from the structural network. Global RF points are defined in the same coordinate frame as the structural points, and we train directly on the full global RF maps without cropping or incremental context expansion; the RF training data fits on a single NVIDIA L40S GPU. Local RF maps are segmented into trajectory chunks in the same way as the structural data, but the RF-specific augmentations operate only on the fingerprint vectors. Concretely, we add element-wise Gaussian noise $\mathcal{N}(0, 4)$ dBm to the RF fingerprints and randomly drop individual access points from the fingerprint vectors to simulate missing or unstable RF measurements. No geometric scaling or crop jitter is applied in this stage. The RF encoder is trained such that superpoints that are close to each other in 3D space obtain similar RF descriptors, while distant superpoints are mapped to more distinct embeddings.

Stage 3: fusion and superpoint matching. In the third stage, we freeze both the structural backbone and the RF encoder and train the localized cross-modal fusion layer together with the superpoint matching module. On each structural–RF superpoint, the fusion module aggregates information from both modalities and produces fused descriptors that are optimized for robust superpoint matching and pose estimation, while keeping the underlying encoders fixed. Training uses the same trajectory chunking and global context strategy as in Stage 1 (including full-map crops), but without modifying the frozen backbones.

Table 4. RF-loc Architecture

Stage	Superpoint Matching Module		
1	Input Projection: Linear(2048→128)		
2	RPESelfAttention(Local) FeatCrossAttention(Local→Global)	8	RPESelfAttention(Global) FeatCrossAttention(Global→Local)
3	RPESelfAttention(Local) FeatCrossAttention(Local→Global)	8	RPESelfAttention(Global) FeatCrossAttention(Global→Local)
4	RPESelfAttention(Local) FeatCrossAttention(Local→Global)	8	RPESelfAttention(Global) FeatCrossAttention(Global→Local)
5	Output Projection: Linear(128→256)		
6	Gaussian Correlation → Dual-Normalization → Top-k Retrieval		

Stage 4: joint fine-tuning. Finally, we fine-tune the *entire* RF-Loc network end-to-end, including the structural backbone, RF encoder, fusion layer, and the coarse-to-fine matching components. This stage is performed on full global maps with the same structural and RF augmentations as in the previous stages, reusing the Adam optimizer and learning-rate schedule with a reduced effective learning rate. End-to-end fine-tuning allows all components to co-adapt to the final localization objective on large-scale scenes.

All training experiments are run on a compute cluster equipped with $8\times$ NVIDIA L40S and $4\times$ NVIDIA H100 GPUs, while inference and runtime evaluation are performed on a single NVIDIA RTX 3090 GPU.

B. Pseudo ground-truth pose generation for Hololens sessions

While the Lamar dataset offers large-scale, multi-modal data, several Hololens sequences lack transformation poses that align them with the global reference map constructed using the NavVis device. To address this, we compute pseudo ground-truth transformations by applying the sequence alignment algorithm proposed in [47]. For completeness, we briefly summarize this method below. The alignment procedure registers each Hololens sequence individually to the reference model. Given a sequence of n frames, the algorithm estimates the absolute poses $\{{}^{w}\mathbf{T}_i\}_{1\le i\le n}$ of each frame in the global coordinate system. Each frame corresponds to an image captured at a specific time, or in the case of a multi-camera rig such as the Hololens, a set of simultaneously captured images. The method assumes access to per-frame trajectories ${}^{0}\mathbf{T}_i^{\text{track}}$ estimated by a visual-inertial tracking system.

Pre-processing: During the alignment process, we observed that several Hololens sessions exhibited significant drift between adjacent frames. These errors prevented the optimizer from successfully converging, resulting in poor alignment. To mitigate this issue, we cropped out segments of the session where high drift was detected. Specifically, a crop was initiated if the time difference between consecutive frames exceeded 0.025 seconds and either the rotational speed exceeded 4 rad/s or the translational speed exceeded 3 m/s. Cropping continued until the motion conditions returned to acceptable levels. This preprocessing step effectively splits each session into multiple subsessions, on which we subsequently perform sequence alignment individually.

Initial Localization: For each frame $\{I_i^{\text{query}}\}$, a fixed number r of relevant reference images $(I_j^{\text{ref}})_{1\le j\le r}$ is retrieved using global image descriptors. Sparse local features [10, 41] are extracted from the query frame and matched to the reference images to form 2D-2D correspondences $\{\mathbf{p}_{i,k}^{q}, \mathbf{p}_{j,k}^{\text{ref}}\}_k$. These keypoints of the reference images are lifted to 3D using the reference mesh, resulting in 2D-3D correspondences $\mathcal{M}_{i,j} := \{\mathbf{p}_{i,k}^{q}, \mathbf{P}_{j,k}^{\text{ref}}\}_k$. All matches for a given frame are aggregated into $\mathcal{M}_i = \bigcup_{j=1}^{r} \mathcal{M}_{i,j}$. An initial absolute pose ${}^{w}\mathbf{T}_i^{\text{loc}}$ is estimated using a generalized P3P algorithm [29] within a LO-RANSAC framework [7]. Frames lacking sufficient inlier correspondences are excluded from further processing.

Rigid Alignment: A coarse initial pose $\{{}^{w}\mathbf{T}_i^{\text{init}}\}$ is recovered for all frames, including those without successful localization. Assuming drift-free tracking, the algorithm determines a rigid transformation ${}^{w}\mathbf{T}_0^{\text{init}}$ that maximizes consensus among localization poses:

$$ {}^{w}\mathbf{T}_0^{i} = {}^{w}\mathbf{T}_i^{\text{loc}} \left({}^{0}\mathbf{T}_i^{\text{track}}\right)^{-1} $$

This candidate is selected via a voting scheme:

$$ {}^{w}\mathbf{T}_0^{\text{init}} = \arg \max_{\mathbf{T}\in\{{}^{w}\mathbf{T}_0^{i}\}_{1\le i\le n}} \sum_{1\le j\le n} \mathbb{1}\left[\text{dist}\left({}^{w}\mathbf{T}_j^{\text{loc}}, \mathbf{T}\cdot{}^{0}\mathbf{T}_j^{\text{track}}\right) < \tau_{\text{rigid}}\right] $$

Subsequently, per-frame initial poses are computed as:

$$ {}^{w}\mathbf{T}_i^{\text{init}} := {}^{w}\mathbf{T}_0^{\text{init}} \cdot {}^{0}\mathbf{T}_i^{\text{track}} $$

Pose Graph Optimization: Initial absolute poses are refined $\{{}^{w}\mathbf{T}_i^{\text{PGO}}\}$ by maximizing the consistency between tracking and localization cues. The optimization minimizes the energy:

$$E(\{{}^{w}\mathbf{T}_i\}) = \sum_{i=1}^{n-1} C_{\text{PGO}}\left({}^{w}\mathbf{T}_{i+1}^{-1}\,{}^{w}\mathbf{T}_i,\, {}^{i+1}\mathbf{T}_i^{\text{track}}\right) + \quad (52)$$

$$\sum_{i=1}^{n} C_{\text{PGO}}\left({}^{w}\mathbf{T}_i,\, {}^{w}\mathbf{T}_i^{\text{loc}}\right) \quad (53)$$

where

$$C_{\text{PGO}}(\mathbf{T}_1, \mathbf{T}_2) := \|\text{Log}(\mathbf{T}_1\mathbf{T}_2^{-1})\|_{\Sigma,\gamma}^2$$

The operator Log maps elements from the Lie group $\text{SE}(3)$ to its Lie algebra $\mathfrak{se}(3)$ [50].

Guided Localization via Visual Overlap: To improve pose accuracy, the refined poses $\{{}^{w}\mathbf{T}_i^{\text{PGO}}\}$ are used to identify additional reference images based on visual overlap. Visual overlap between a source image i and a reference image j is defined as the fraction of pixels in i that are visible in j. This is determined by projecting pixels from image i into the coordinate frame of image j using known depth and camera poses, and checking whether they fall within the image bounds. The score is given by:

$$O(i \to j) = \frac{1}{W \cdot H} \sum_{k \in (W,H)} \mathbb{1}[\Pi_j\left({}^{w}\mathbf{T}_j, \quad (54)$$

$$\Pi_i^{-1}\left({}^{w}\mathbf{T}_i, \mathbf{p}_k^i, z_k\right) \in (W,H)]\alpha_k, \quad (55)$$

where Π_i denotes the projection of a 3D point into image i, and Π_i^{-1} is its inverse that backprojects a pixel $\mathbf{p}_k^i$ with depth z_k into 3D space. The indicator function $\mathbb{1}[\cdot]$ evaluates to 1 if the projected 3D point lies within the bounds of image j. Each pixel's contribution is weighted by the cosine of the angle between the surface normals observed in both views, given by $\alpha_k = \cos(\mathbf{n}_{i,k}, \mathbf{n}_{j,k})$.

For each frame, the top r overlapping reference images are selected, local features are matched, and new absolute poses are estimated. These new priors are incorporated in a subsequent pose graph optimization.

Bundle Adjustment: For each frame i, the set of 2D-3D correspondences $\mathcal{M}_i$ is recovered used by the guided re-localization and used to jointly refine the poses $\{{}^{w}\mathbf{T}_i^{\text{BA}}\}$ by minimizing:

$$E(\{{}^{w}\mathbf{T}_i\}) = \sum_{i=1}^{n-1} C_{\text{PGO}}\left({}^{w}\mathbf{T}_{i+1}^{-1}\,{}^{w}\mathbf{T}_i,\, {}^{i+1}\mathbf{T}_i^{\text{track}}\right) +$$

$$+ \quad \sum_{i=1}^{n} \sum_{\mathcal{M}_{i,j} \in \mathcal{M}_i} \sum_{(\mathbf{p}_{i,k}^q, \mathbf{P}_{j,k}^{\text{ref}}) \in \mathcal{M}_{i,j}} \left\| \Pi({}^{w}\mathbf{T}_i, \mathbf{P}_{j,k}^{\text{ref}}) - \mathbf{p}_{i,k}^q \right\|_{\sigma^2}^2$$

The second term evaluates the reprojection error of 3D points. Correspondences with reprojection error above the threshold τ_{reproj} are filtered out. Optimization is performed using the Ceres Solver, with noise priors aligned to the lidar specifications.

C. Train and validation sequence selection for Lamar

Trajectory data is inherently embedded in a 3D physical space, and random splits partitioning methods often fail to account for critical spatial properties like coverage imbalance, redundancy, and the availability of ground truth. In this work, we address this challenge in the context of a dataset comprising trajectories from two distinct sources: NavVis, a system that provides high-accuracy ground truth trajectories for select portions of the environment, and Hololens, which contributes a larger and less structured set of trajectories. Our aim is to construct a validation set from Hololens subsessions that enables robust evaluation against NavVis ground truth, while ensuring spatial diversity, avoiding redundancy, and maintaining broad coverage of the reference environment.

To avoid the pitfalls of random selection, we develop a greedy spatial selection strategy that operates on the following core principles: (i.) maximize overlap between the selected validation trajectories and the NavVis-covered regions; (ii.) minimize internal redundancy within the validation set; and (iii.) avoid including trajectories that traverse regions not covered by NavVis, which cannot be used for reliable evaluation.

The environment is discretized into uniform 3D voxels, each measuring 3 meters per side. Let $\mathcal{S}_{\text{hl}} = \{s_1, \ldots, s_N\}$ denote the set of all Hololens subsessions. Each subsession $s \in \mathcal{S}_{\text{hl}}$ is associated with a set of visited 3D voxels $\mathcal{C}(s) \subset \mathbb{R}^3$. Let $\mathcal{C}_{\text{nav}} \subset \mathbb{R}^3$ be the union of voxels visited by NavVis trajectories, representing the spatial domain where ground-truth supervision is available.

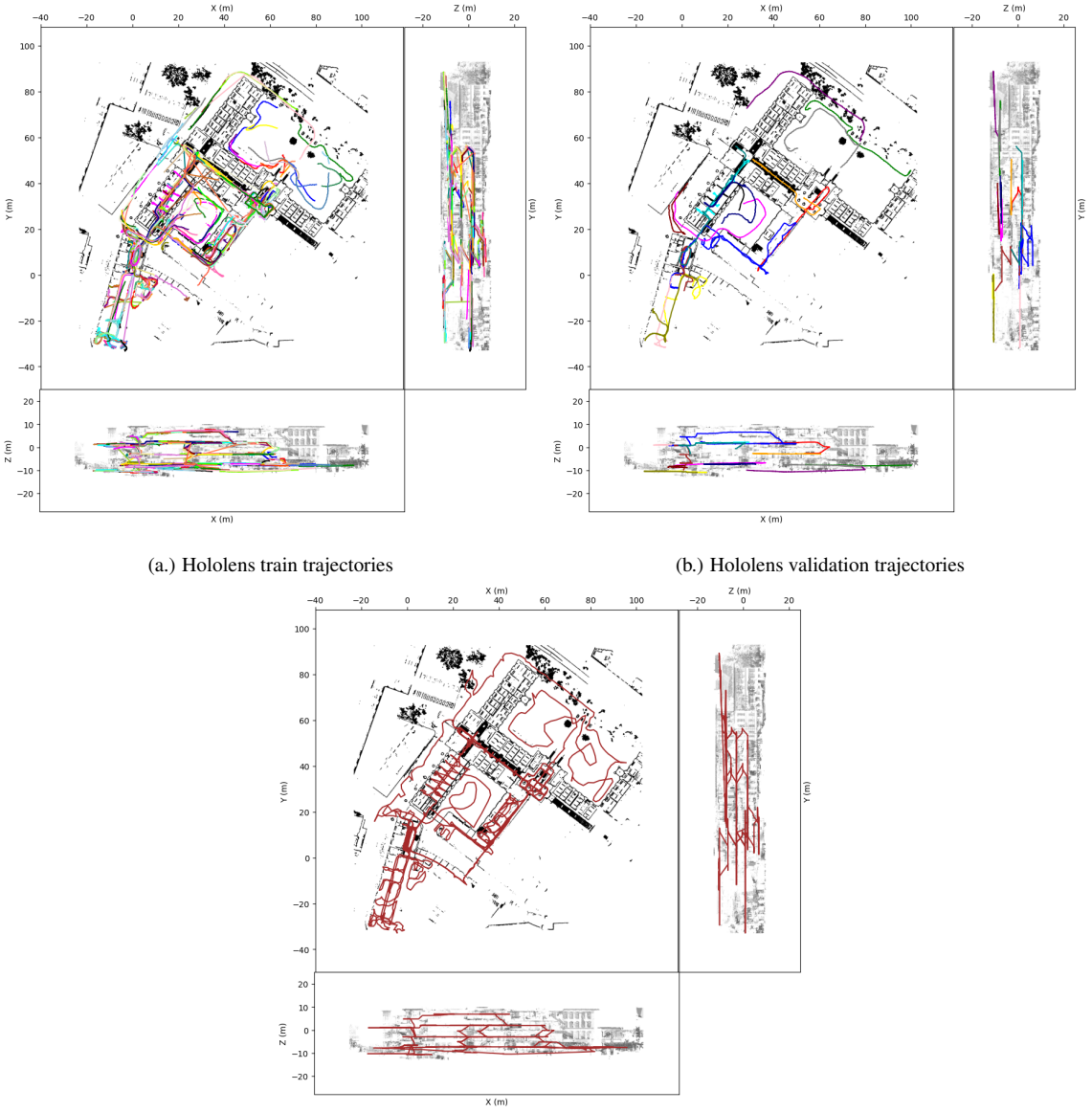
The NavVis coverage of a candidate validation set $\mathcal{V} \subset \mathcal{S}_{\text{hl}}$ is defined as:

$$\text{Coverage}(\mathcal{V}) = \frac{\left|\bigcup_{s \in \mathcal{V}} \mathcal{C}(s) \cap \mathcal{C}_{\text{nav}}\right|}{|\mathcal{C}_{\text{nav}}|},$$

i.e., the fraction of NavVis-covered space that is jointly visited by the selected validation trajectories.

The selection process begins with an empty validation set $\mathcal{V} = \emptyset$ and proceeds greedily. At each step, the algorithm selects the subsession $s^* \in \mathcal{S}_{\text{hl}} \setminus \mathcal{V}$ that yields the highest marginal increase in NavVis coverage:

$$\Delta_{\text{coverage}}(s \mid \mathcal{V}) = \frac{\left|\mathcal{C}(s) \setminus \bigcup_{s' \in \mathcal{V}} \mathcal{C}(s') \cap \mathcal{C}_{\text{nav}}\right|}{|\mathcal{C}_{\text{nav}}|}.$$



(a.) Hololens train trajectories

(b.) Hololens validation trajectories

(c.) NavVis trajectories

Figure 2. NavVis and Hololens trajectories for the CAB scene in the Lamar dataset.

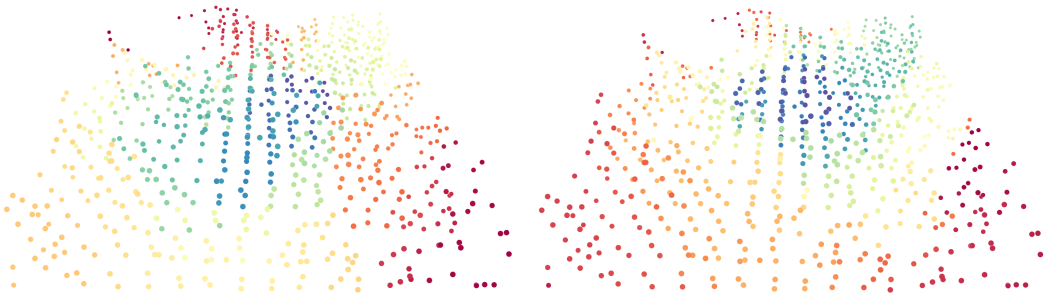
Only sessions with $\Delta_{\text{coverage}} \geq \tau$ (where $\tau = 0.01$) are considered for inclusion, ensuring that each addition provides meaningful new coverage of the ground-truth region. If multiple candidates yield similar marginal coverage gains, a tie-breaking strategy is applied. The first criterion is spatial redundancy, measured as the fraction of a trajectory's voxels that are visited more than once:

$$\text{Redundancy}(s) = \frac{\sum_{v \in \mathcal{C}(s)} \mathcal{F}_s(v)}{|\mathcal{C}(s)|},$$

where $\mathcal{F}_s(v)$ denotes the number of times voxel v is visited in subsession s. This expression captures the average number of visits per voxel, thus quantifying how often the trajectory revisits the same space. If redundancy is also similar (within a small tolerance), ties are broken by minimizing the proportion of the trajectory that lies outside the NavVis-covered region:

$$\text{NonNavVis}(s) = \frac{|\mathcal{C}(s) \setminus \mathcal{C}_{\text{nav}}|}{|\mathcal{C}(s)|}.$$

This hierarchical selection rule ensures that, at each iteration, the most spatially informative, diverse, and evaluable trajectory is added to the validation set. The process terminates when no further session can increase NavVis coverage beyond the threshold τ. The result is a validation set $\mathcal{V} \subset \mathcal{S}_{\text{hl}}$



(a) Without RF fusion (b) With RF fusion

Figure 3. t-SNE projection of coarse (high level) feature descriptors with and without RF signal integration.

(a) Overlay without RF fusion (b) Overlay with RF fusion

Figure 4. Overlay of coarse descriptors on a map. With RF, descriptors exhibit stronger spatial separation.

that maximizes spatial coverage over ground-truthed regions, with minimized internal redundancy and minimal inclusion of unevaluable space. The training set is defined as the disjoint complement $\mathcal{T} = \mathcal{S}_{\text{hl}} \setminus \mathcal{V}$.

Figs. 2a and 2b illustrate the outcome of the greedy selection strategy applied to partition the Hololens sessions into training and validation trajectory sets, with different colors indicating distinct sessions. Fig. 2c depicts the NavVis trajectory used to construct the global reference map. By ensuring that the validation set is well covered, we make sure that evaluation metrics are meaningful and grounded. At the same time, the training set retains broad spatial diversity, including areas not visited by NavVis, thus supporting generalization to unseen regions during deployment. This approach provides a principled, reproducible framework for constructing training and validation sets in spatial trajectory datasets.

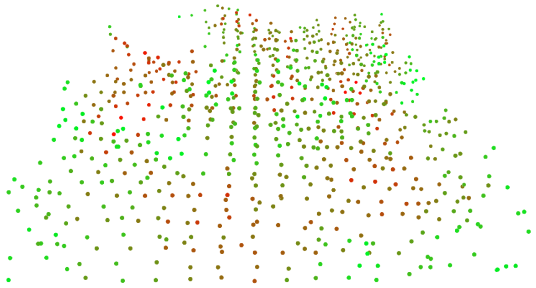
D. Detailed runtime analysis

We further dissect the runtime of the single-image pipeline into local feature extraction, pair selection, feature matching, pose estimation, and pose graph optimization, and study how these depend on the choice of local features, global descriptors, and matcher, as well as on the trajectory length.

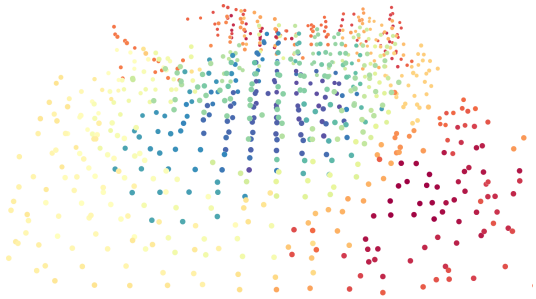
Feature extraction and pair selection. The per-image GPU cost of local feature extraction is almost identical for SuperPoint and ALIKED: SuperPoint requires roughly $6-7$ ms per image, while ALIKED-N16 increases this to only $11-12$ ms. Thus, the choice of local detector/descriptor has only a minor impact on the overall runtime compared to downstream stages. In contrast, pair selection—which includes the computation of global image descriptors and nearest-neighbor search—shows a clear dependency on the global descriptor backbone. OpenIBL yields the lowest cost with about $0.018-0.019$ s per image, whereas Megaloc, SALAD, and the fusion model all lie in a similar but noticeably slower range ($\approx 0.046-0.058$ s per image). This confirms that, among the retrieval backbones we consider, OpenIBL is clearly preferable when targeting low latency.

Feature matching. As expected, the choice of matcher strongly affects the GPU runtime. Across all global descriptor backbones, LightGlue is consistently faster than SuperGlue. For SuperPoint, LightGlue requires about 0.22 s per image, whereas SuperGlue needs around 0.29 s. This gap persists when switching to ALIKED, confirming that LightGlue yields a $20-25\%$ reduction in matching time while keeping the rest of the pipeline unchanged.

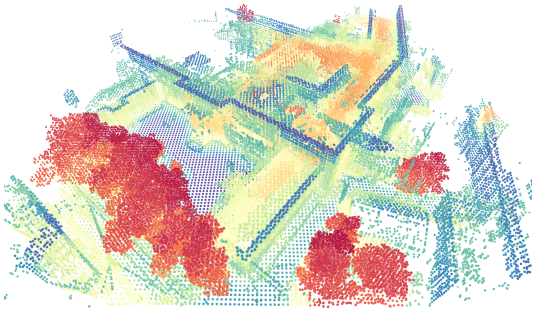
(a) RGB colored point cloud of the CAB scene.



(b) Confidence scores of the coarse superpoints of the global map.



(c) t-SNE visualization of the fused coarse features of the global map.



(d) t-SNE visualization of the fine features of the global map.

Figure 5. Visualization of a the CAB scene, together with the computed coarse confidence scores, coarse and fine extracted feature representation visualized in a lower-dimensional space using t-SNE.

Pose estimation. The CPU-side pose estimation cost per image varies most strongly with the detector -matcher combination. For SuperPoint, pose estimation takes approximately 0.26–0.40 s per image, with SuperGlue configurations being slightly faster than LightGlue on average because fewer valid correspondences survive matching. ALIKED combined with LightGlue is consistently slower, with pose estimation times in the range of $0.50-0.64$ s per image. This is in line with the intuition that ALIKED produces more keypoints and LightGlue preserves more valid matches, which increases the number of 2D–3D correspondences that have to be processed by PnP+RANSAC and therefore slows down the CPU stage. In contrast, the SuperPoint+SuperGlue configuration yields the smallest number of inliers, which keeps PnP+RANSAC cheapest and explains why its pose estimation runtime is the best among all evaluated combinations.

Pose graph optimization and trajectory length. The runtime of the final pose graph optimization step scales with the number of keyframes in a trajectory chunk. For shorter trajectories (e.g. 5 m chunks), there are fewer keyframes and therefore fewer constraints in the optimization, resulting in pose-graph times of roughly 0.08 s per chunk. For longer trajectories (20 m chunks), the number of keyframes and loop constraints increases and the PGO cost grows to about $0.30-0.35$ s per chunk. This behaviour is largely independent of the chosen detector and matcher, since PGO operates only on the selected keyframes and their poses, not on individual correspondences.

E. Evaluation Metrics

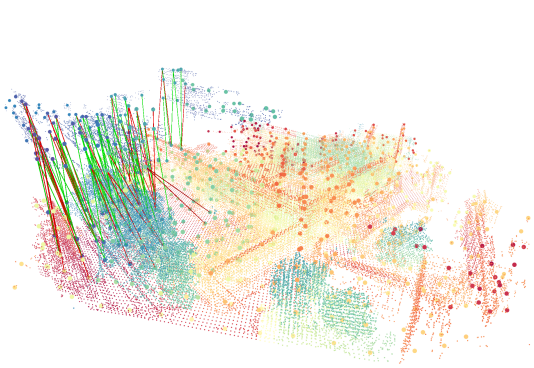
We evaluate the quality of point cloud registration using three commonly adopted metrics: Relative Rotation Error (RRE), Relative Translation Error (RTE), and Registration Recall (RR).

Relative Rotation Error (RRE) RRE quantifies the angular discrepancy between the estimated rotation matrix $R \in \mathrm{SO}(3)$ and the ground-truth rotation matrix $\bar{R} \in \mathrm{SO}(3)$. It is computed as the geodesic distance on the rotation group:

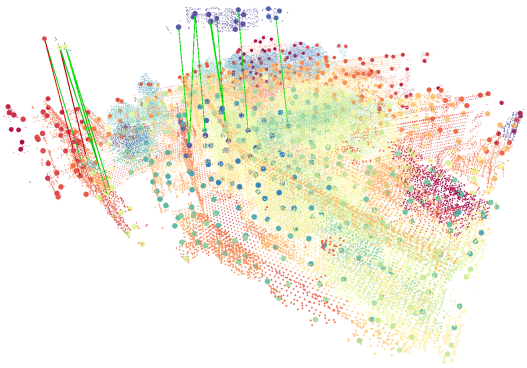
$$\mathrm{RRE} = \arccos\left(\frac{\mathrm{trace}(R^\top \bar{R}) - 1}{2}\right).$$

Relative Translation Error (RTE) RTE measures the Euclidean distance between the predicted translation vector $t \in \mathbb{R}^3$ and the ground-truth translation vector $\bar{t} \in \mathbb{R}^3$:

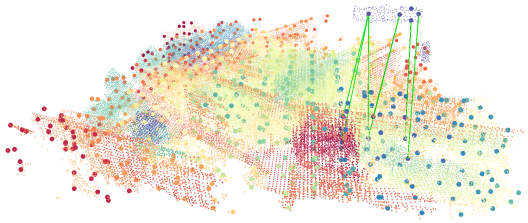
$$\mathrm{RTE} = \|t - \bar{t}\|_2 .$$



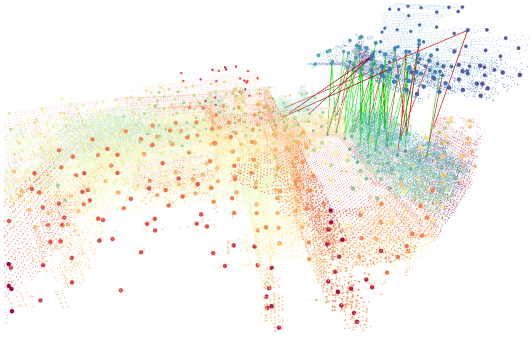
(a) Sidewalk area with tree clusters and partial building wall



(b) Indoor layout capturing some distant outdoor points



(c) Narrow indoor corridor with dense wall and ceiling points



(d) Outdoor area facing a tall building wall

Figure 6. RF-assisted coarse superpoint matching visualizations. All examples use t-SNE to visualize the fused descriptors. Green lines denote correct matches with spatial overlap; red lines indicate incorrect correspondences.

Registration Recall (RR) RR evaluates the fraction of point cloud pairs correctly registered, defined as those for which both RRE and RTE fall below specified thresholds. For the Lamar benchmark, we use thresholds of 5° for RRE and 1 m for RTE. Given M test samples, RR is defined as:

$$\mathrm{RR} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}\left[\mathrm{RRE}_i < 5^\circ \ \wedge \ \mathrm{RTE}_i < 1\,\mathrm{m}\right],$$

where $\mathbb{1}[\cdot]$ denotes the indicator function.

F. Impact of RF Fusion on Feature Distribution

To better understand how RF signals affect the learned representations, we compare the spatial distribution of high-level features extracted with and without RF input. Figure 3 shows a t-SNE projection of the descriptors obtained in both cases. When RF signals are included during feature extraction, the resulting descriptors exhibit more distinct clusters and greater separation in the embedding space. This separation leads to a more discriminative representation, which is beneficial for localization. In the no-RF case, features from geometrically similar but spatially distant areas (e.g., outdoor flat regions) often result in similar descriptors. While these areas may appear structurally alike, assigning them nearly identical embeddings hinders precise localization, as distinct locations become indistinguishable. With RF fusion, the signal differences across floors and building sections help the model disambiguate these similar-looking regions. As shown in Figure 4, different parts of the building—such as separate wings or levels—are mapped to clearly distinct descriptor spaces, allowing more accurate and robust localization.

G. Limitations

The proposed model has several limitations that arise from constraints in memory efficiency and system scalability.

To fit input point clouds into GPU memory, a voxel grid subsampling with 10 cm resolution is applied. While this enables practical training and inference, it comes at the cost of reduced spatial fidelity. The loss of fine-grained geometric detail can negatively impact localization accuracy, particularly in environments where structural precision is essential. Although finer resolutions—such as 5 cm voxel grids—could improve performance, they exceed the memory capacity of the NVIDIA L40S GPUs.

Training on large-scale environments, such as the full CAB map, poses an additional challenge. This process requires GPUs with very high memory capacity, such as the NVIDIA H100 with 80 GB of VRAM, limiting the method's accessibility to users with more conventional hardware se-

tups.

Moreover, scaling the model to even larger environments necessitates distribution across multiple GPUs to accommodate both input data and intermediate feature representations. This multi-GPU setup increases engineering complexity, introduces additional synchronization overhead, and complicates deployment in resource-constrained or real-time systems.

H. Qualitative Results of Encoding and Registration

Figure 5 shows the point cloud of the CAB scene from the LaMAR dataset, along with its corresponding fine and coarse feature representations. High-dimensional feature vectors are projected into a lower-dimensional space using t-SNE [57] to visualize the fused coarse and decoded fine features.

To support the quantitative findings, in Figure 6 we show examples of registration behavior in diverse environments. These include a sidewalk area with vegetation and wall structures, an indoor layout partially capturing outdoor regions, a narrow corridor, and an open area facing a tall building wall. The model successfully identifies discriminative features across varied scenes and viewpoints.