# Supplementary Material: Can we make NeRF based visual localization privacy-preserving?

The supplementary material is structured as follows. In Sec. 1 we first provide comprehensive details regarding the ppNeSF architecture, including a detailed description of the image-based encoder $\mathbf{\Phi}$, the feature field $\mathbf{\Gamma}$, and the underlying neural implicit field. In addition we give more details regarding the hierarchical scheme and the uncertainty computation. In Sec. 2 first we detail how the model is trained, then, we provide further details on the pose refinement, the relation between training/runtime cost vs. performance, and study the impact of the number of segmentation classes considered on the localization accuracy. Finally, in Sec. 3 we expand upon the privacy aspect. First, we describe the inversion model used for the privacy attack and describe our training and evaluation protocol for the privacy attack. Then, in Sec. 3.1 we provide additional privacy baseline experiments and in Sec. 3.2 an analysis on why using segmentations allows high level privacy preservation.

## 1. Details on the ppNeSF architecture

**Image-based encoder $\mathbf{\Phi}$.** The image-based encoder is composed of a feature backbone and a decoder. The backbone is a SWIN-t transformer [12] (patch size: 2, embed dim: 96, depths: $\{2, 2, 6, 2\}$, num heads = $\{3, 6, 12, 24\}$) which outputs a set of 4 feature maps with strides (2, 4, 8, 16). No pre-trained weights are used to initialize the encoder.

The feature map with stride 2 has a dimension of 96. It is processed through a feature head consisting of three conv2D layers (internal dimension 128) with ReLU activation and one upsampling layer such that the feature map is pixel aligned. This resulting feature map has a dimension of 96 and is used for the joint embedding space learning.

The multi-scale feature maps are processed with a convolutional feature pyramid type of network [11] to output two final feature maps of stride 4 and 2. They are processed through segmentation heads (three conv2D layers with ReLU activation and internal dimension of 128, one upsampling layer of scale 4 or 2) which yield the pixel aligned coarse and fine segmentation maps with respectively $K$ and $K_f$ classes.

The final feature maps are processed through uncertainty heads (three conv2D layers with ReLU activation and internal dimension of 128, one upsampling layer of scale 4 or 2) to obtain the pixel aligned coarse and fine uncertainty maps with respectively $K$ and $K_f$ dimensions. The gradients are detached from the input of the feature head, coarse and fine uncertainty heads as to not destabilize the segmentation representation learning.

**Feature field $\mathbf{\Gamma}$.** The feature field consists of hash tables with 10 levels with resolution ranging from 16 to 16,384. The feature dimension per level is set to 4 with a log hashmap size of 21. Following [2], a conical section of a ray consists of 6 points and is encoded through the hash tables. The encoded features are weighted and averaged. This encoding makes the 3D feature scale-aware which is beneficial for scenes with large viewpoint changes. The resulting feature is then concatenated with the scale feature (see [2] for more details) and processed by an MLP consisting of 3 linear layers (internal dimension 128) with ReLU activations. The output feature has a dimension of 96 to match the encoder-based feature dimension. Feature rendering uses the opacity weight from the main neural implicit field. Gradients are detached from these weights.

**ppNeSF.** A neural implicit field is a representation of a 3D scene whose underlying geometry is encoded through MLPs. Additional information such as radiance, semantics, or features may also be stored and encoded. NeRFs [13] are a special form of neural implicit fields which store radiance and use volumetric rendering [7] to optimize the neural fields solely using RGB images with known intrinsics and camera poses. Follow-up work tackle training efficiency by introducing 3D structures such as hash tables, octrees, or grids [4–6, 14, 21, 22], reducing the number of required training views by adding additional regularization or geometric priors [16, 19, 20, 23], and improving the quality by reducing aliasing [1, 2, 15].

PPNeSF uses the ZipNerf architecture as the background neural implicit field.[1] Concretely, given an image with known pose, each emitted ray is modelled as a conical frustum whose sections can be decomposed into a set of 6

[1]Our implementation is based on https://github.com/SuLvXiangXin/zipnerf-pytorch.git.

isotropic Gaussians such that they approximate the shape of the conical frustum. For each level $V_l$ of the grid in a a conical section, a feature is obtained by trilinear interpolation applied at the Gaussian's mean location $x_j$. The six resulting features are re-weighted $w_{j,l}$ based on how each Gaussian fits into the grid cell and the averaged feature is

$$f_l = \text{mean}_j(w_{j,l} trilerp(x_j; V_l)$$

where $trilerp$ is the trilinear interpolation operation. The resulting level features are then concatenated and fed into a shallow MLP to obtain the density and a geometric feature $d_i, g_i = h_{geo}(cat[f_l]_l)$. The geometric feature is fed along with the encoded point position to a shallow MLP to obtain the 3D segmentation

$$s_i^{3D} = h_{seg}(cat[g, enc(pos)]).$$

This combination of multi-sampling and weighting effectively reduces spatial-aliasing and handles scale variations.

Segmentations can finally be rendered through alpha composition $s^{3D} = \sum_{i=1}^{n} T_i \alpha_i s_i^{3D}$, where $T_i$ is the accumulated transmittance and $\alpha$ the discrete opacity value at sample $i$. Furthermore, Z-aliasing induced by the proposal sampling is handled by deriving an inter-level loss which is smooth with regard to translation along the ray. For more details please refer to [2].

Except for the segmentation part, PPNeSF's geometric field $\mathbf{\Psi}$ architecture is similar to ZipNeRF [2]. Two levels of proposal networks are used, their hash tables' maximum resolutions are 512 and 2048. The main network uses a hash grid with 10 levels with resolution ranging from 16 to 16,384. The feature dimension per level for all hash tables is set to 4 with a log hashmap size of 21. The geometric MLP contains two linear layers and ReLU activations (internal dimension 64), scale featurization is used. Coarse and fine semantic heads consist of a three layer MLP with ReLU activations (internal dimension 128).

The number of samples per proposal network is set to 48 while the number of samples for the main network is set to 24. Sampling is handled by the proposal networks (see [2] for more details).

### 1.1. Fine segmentation level

In Sec. 4 of the main paper, we introduced a hierarchical approach to provide a finer supervision to ppNeSF and increase the convergence basin during visual localization. We further explain how the fine segmentation targets are derived. For each coarse class $k \in [1, K]$, we want to establish a mapping between the assigned features $A(k)$ to coarse class $k$ and the $n$ fine prototypes associated to class $k$. Let $K_f = n * K$ be the total number of fine prototypes and $U$ the number of pixels in the training batch. We first compute softmax scores from the similarities between 3D features and 3D fine prototypes $\{p_{fk}^{3D}\}_{k=1}^{n}$ and the similarity between 2D features and 2D fine prototypes $\{p_{fk}^{2D}\}_{k=1}^{n}$. Subsequently, for a coarse class $k$, we solve Eq. 2 (main paper) with $S^f \in \mathbb{R}^{|A(k)|n}$ defined as $S_{ik}^f = exp(F_i^{2D} p_{fk}^{2D}/\tau_2) exp(F_i^{3D} p_{fk}^{3D}/\tau_2)/Z$ yielding $Q_k^f \in \mathbb{R}^{|A(k)|n}$, Z being a normalization factor. The $K$ resulting mappings $Q_k^f$ are combined into $Q^f \in \mathbb{R}^{UK_f}$, which is used as segmentation target to optimize the overall training objective Eq. 1 (main paper) with regard to the image encoder $\mathbf{\Phi}$ and the segmentation/geometry fields $\mathbf{\Omega}$/$\mathbf{\Psi}$. Note that with this hierarchy constraint, $Q^f$ is not a minimizer of Eq. 1 (main paper) for the fine feature/prototypes similarity softmax score but still provides coherent and valid target labels. Fine assignments are defined as $A_f(k) = \{i | h(i) = k\}$ with $h = argmax_k(Q^f)$. The fine prototypes are updated by EMA based on fine assignments, similar to the update of the coarse prototypes.

We want to enforce, in the embedding space, the hierarchy that was defined through the aforementioned hierarchical clustering algorithm. To that end we introduce the following hierarchical prototype loss.

$$L_{hierar} = \frac{1}{2} \sum_{k=1}^{K} \frac{1}{|A(k)|} \sum_{i \in A(k)} L_{ick}$$

$$+ \sum_{k=1}^{K_f} \frac{1}{|A_f(k)|} \sum_{i \in A_f(k)} max(L_{ifk}, max_{j \in A(k^c)}(L_{jck^c}))$$

where $L_{ick}$ and $L_{ifk}$ are the pixel to prototype contrastive losses at coarse ($c$) and fine level ($f$), with

$$L_{ick} = -\log\left(\frac{\exp(F_{u_i} p_{ck}/\tau)}{\sum_{j=1}^{N} \exp(F_{u_i} p_{ck}/\tau)}\right)$$

and $L_{ifk}$ defined similarly. For simplicity, we omitted the 2D/3D notations, but the loss is independently applied on 2D feature/prototypes and 3D feature/prototypes. Note that the $k^c$ coarse prototypes are associated to the fine prototypes $k$ and minimizing these distances between pixels and their assigned prototypes ensures that the loss between each pixel and its corresponding fine prototype remains below the loss with the associated coarse prototype.

### 1.2. Uncertainty

We model uncertainty by adding an uncertainty prediction head so that the 2D encoder predicts both segmentation logits $l \in \mathbb{R}^{KxWxH}$ and class-wise uncertainties $\rho \in \mathbb{R}^{KxWxH}$ for each image pixel. In this formulation, the network predicts a per-pixel Gaussian distribution over the segmentation classes, where the vector of logits $l_i$ is the mean and the vector of uncertainties $\rho_i$ is the diagonal elements of the covariance matrix for the pixel $u_i$ The cross entropy objective in Eq. 1 (main paper) is therefore modified by sampling the Gaussian distribution before applying

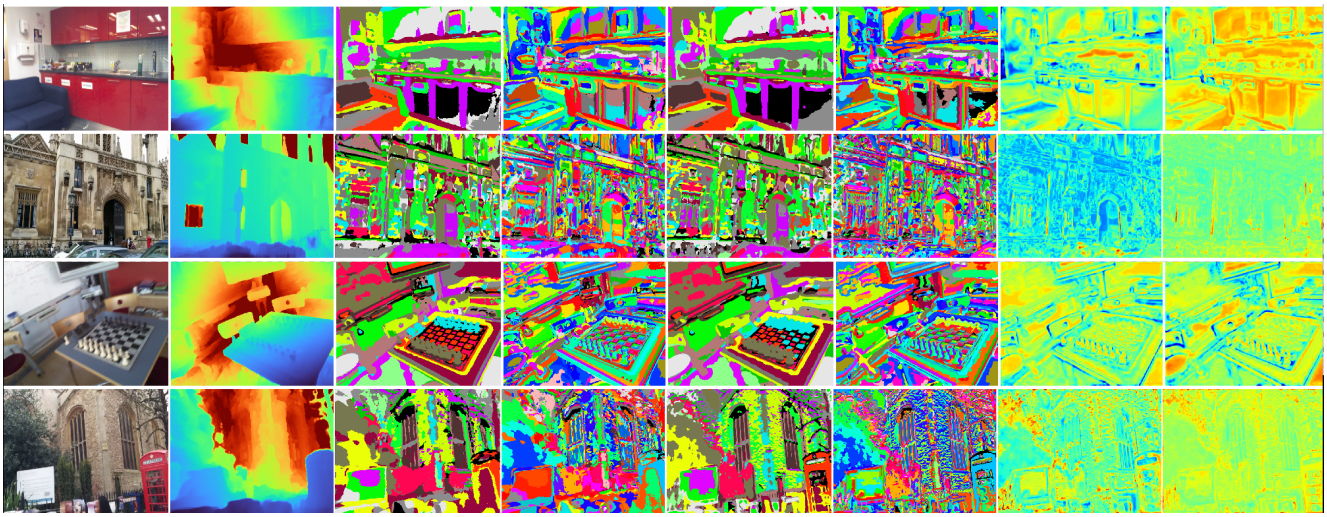

Figure 1. From left to right: original image, rendered depth, coarse image-based segmentation $s_c^{2D}$, fine image-based segmentation $s_f^{2D}$, coarse rendered segmentation $s_c^{3D}$, fine rendered segmentation $s_f^{3D}$, coarse uncertainty map $u_c^{2D}$ and fine uncertainty map $u_f^{2D}$.

softmax. This yields:

$$\bar{s}^{2D}(k|u_i) = \text{softmax}(\text{mean}_t(\{l_i^{2D} + \rho_i * \epsilon_t\}_1^{N_S}))$$
$$\bar{s}^{3D}(k|u_i) = \text{softmax}(\text{mean}_t(\{l_i^{3D} + \rho_i * \epsilon_t\}_1^{N_S}))$$

where $\epsilon_t \sim N(0, I)$ is jointly sampled $N_S$ times for the 2D and 3D logits so that both modalities benefit from the uncertainty modelling. This allows for better stability during training and we use uncertainty to filter out ambiguous pixels during visual localization in Eq. 3 (main paper).

We provide visual examples of coarse/fine image-based and rendered segmentations, as well as coarse/fine uncertainty maps in Fig. 1.

## 2. Details on training and localization

**Training.** ppNeSF and the feature field $\mathbf{\Gamma}$ are trained with an initial learning rate of 1e-2 exponentially decaying to 1e-4 with the Adam optimizer [10]. The encoder is trained with an initial learning rate of 1e-3 exponentially decaying to 1e-4 with the Adam optimizer [10].

All models are trained for 50k iterations on a single Nvidia V100 32Gb GPU. We sample 4096 rays per training iteration. A training iteration takes approximately 800ms. 50,000 training iterations are done per scene. Proposal weights are annealed during 1,000 iterations. The coarse and fine prototypes are randomly initialized at the beginning of the training. Similar to [2], distortion (with a weight of 0.5) and anti-interlevel (with a weight of 0.1) losses are used during training.

Depth supervision with the depth loss from [23] (with a weight of 2) is used to give coarse geometry priors to our models. We used rendered depth maps from DSAC*[2] on 7Scenes, Marigold [8] (model "`marigold-lcm-v1-0`") to estimate depth maps on Cambridge Landmarks, and ZoeDepth [3] ("`zoe nk`" model on torch hub) on Indoor6.

To resume these details, we provide the pseudo-code describing the training process of PPNeSF in Algorithm 1.

### 2.1. Visual localization

During the pose refinement, 4096 rays are sampled per iteration on 7Scenes and Indoor6, while 8192 rays are sampled per iteration on Cambridge Landmarks. 64 proposal samples per ray and 32 final samples per ray are used. The pose is refined for 150 coarse iterations and 150 fine iterations. While most of the queries converge in a much lower number of iterations, a few queries require more iterations when the optimization landscape is not smooth. The pose is optimized on SE(3) with an Adam optimizer, 0.33 decay rate where the initial learning rate is set to 2e-2 for all datasets.

Comparatively, we run a simple baseline performing pose refinement by aligning RGB images rendered with ZipNeRF to the query images. It yields an average 19cm/4.8°/47% on 7Scenes and 5.1m/31° on Cambridge Landmarks compared to 5cm/0.35°/87% on 7Scenes and 14cm/0.4° on Cambridge Landmarks obtained with ppNeSF. The poor localization results obtained with RGB alignment is mainly due to to illumination variations and lack of texture, which is compensated by the segmentation based representations of ppNeSF, which is robust to such challenges.

### 2.2. Training/runtime cost *vs*. performance

ppNeSF provides higher localization accuracy and a better degree of privacy than concurrent methods thanks to its better representation learning/clustering approach and regular-

[2] https://github.com/vislearn/dsacstar

**Data:** Set of posed training images with associated depth
Randomly initialize the coarse and fine prototypes.
**for** *iteration in range(N_iterations)* **do**
  Sample a random image $I$
  Extract 2D image-based features, fine/coarse segmentation maps and uncertainty $F^{2D}, s_c^{2D}, s_f^{2D}, u_c, u_f$
  Sample 4096 rays $\{r_n\}_{n=1}^{4096}$ in $I$, through two rounds of proposal networks sample 24 final samples per ray
  Bilinearly interpolate $F^{2D}$ at ray origin pixels which yields $\{F_i^{2D}\}_{n=1}^{4096}$
  Query the feature field $\mathbf{\Gamma}$ and ppNeSF's $\mathbf{\Psi}$-$\mathbf{\Omega}$ to a obtain feature, opacity, coarse and fine segmentations per 3D point $f_i^{3D}, o_i, s_{ic}, s_{if}$
  Integrate along rays to obtain rendered feature, opacity and segmentation maps $F^{3D}, d, s_c^{3D}, s_f^{3D}$
  Compute depth loss $L_{depth}$, distortion loss $L_{dist}$, anti inter-level loss $L_{interl}$
  Compute feature contrastive loss $L_{NCE}$ (see Sec. 4.2.1) with $\{F_i^{3D}, F_i^{2D}\}$
  Solve Eq. 2 (main paper) to obtain coarse segmentation targets $Q$, find assignments per class A(k)
  Compute coarse cross entropy loss (Eq. 1 main paper) with coarse assignment $Q$ and coarse segmentations $s_c^{3D}, s_c^{2D}$
  **for** *coarse class k in range(K)* **do**
    With fine prototypes associated to class k and feature assigned to class k, solve Eq. 2 (main paper) to obtain partial fine segmentation targets $Q_k f$, find assignments per fine class $A_f(k)$
  **end**
  Combine partial fine partial into assignments into fine assignments $Q_f$, compute fine cross entropy loss (Eq. 1 main paper)
  EMA update on coarse and fine prototypes based on coarse and fine assignments
  With coarse/fine assignments $A, A_f$, coarse/fine prototypes and $\{F_i^{3D}, F_i^{2D}\}$ compute $L_{hierar}$
  Jointly update the feature field, ppNeSF's segmentation-geometry fields $\mathbf{\Omega}$-$\mathbf{\Psi}$ and ppNeSF's encoder $\mathbf{\Phi}$ by minimizing the overall loss
  $L = 2 * L_{depth} + 0.5 * L_{dist} + 0.1 * L_{interl} + 0.2 * L_{NCE} + 0.2 * L_{CEc} + 0.2 * L_{CEf} + 0.05 * L_{hierar}$
**end**

**Algorithm 1:** Pseudo algorithm describing the training process of PPNeSF.

| Runtimes | 7Scenes | Indoor6 | 360 | CL |
|---|---|---|---|---|
| Training | 9h23 | 9h05 | 8h38 | 10h44 |
| Refinement | 15s | 15s | X | 15s |

Table 1. Average training and average per-image inference times per dataset.

izations. But this comes at a higher training cost. In Tab. 1, we report for each dataset the average training time, as well as average inference time for an image. Note that efficiency during visual localization could be improved through diverse optimization techniques, however working on this aspect was out of the scope of this paper.

### 2.3. The impact of the number of classes

To evaluate the impact of the numbers of latent segmentation classes, we train the model with different number of classes on the following two scenes: *OldHospital* and *KingsCollege* from the Cambridge Landmarks dataset. We show median translation errors and rotation errors against number of classes in Fig. 2. On *KingsCollege*, at least 100 classes are required to accurately refine the pose while on *OldHospital* 50 classes are enough (*OldHospital* is less complex than *KingsCollege* so most information within the scene can be captured with fewer classes). Above these minimum number of classes, accuracy increases at a marginal rate. For a small number of classes the localization accuracy is poor, meaning that a minimum number of classes is required to provide sufficient discriminativeness. Overall, the optimal number of classes is scene-dependent and depends on the complexity of the scene.

## 3. Evaluating privacy preservation

**Architecture and optimization.** We call "internal representation" of a NIF the feature output of the MLP predicting geometric information. In NeRFs this feature further serves as the input to the MLPs predicting color. We use a UNET [18] architecture, which takes as input feature maps and outputs one channel grayscale reconstructed images. It consists of six encoding blocks (each composed of the following layers: Reflectionpad2D, conv2D, Batchnorm2D and ReLU activation), six decoding blocks (each composed of the following layers: Upsample, Reflectionpad2D, conv2D, Batchnorm2D and ReLU activation) of four refinement blocks (each composed of the following layers: Reflectionpad2D, conv2D, Batchnorm2D and ReLU activation or sigmoid activation and no batchnorm for the

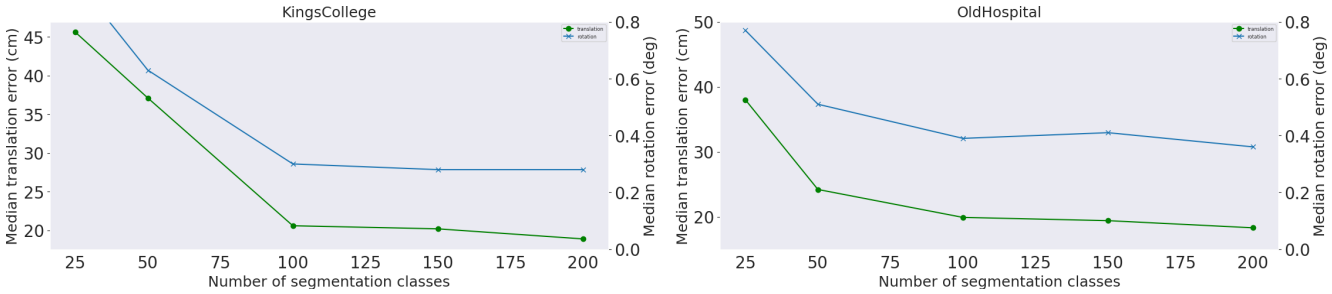


Figure 2. Refinement with different number $K_f = n * K$ of fine classes (varying number $K$ of coarse classes, $n = 5$ fine classes per coarse class). The green curve corresponds to the translation and the blue curve to the rotation error.

last block). Internal dimensions of encoder blocks are (256, 256, 256, 512, 512, 512), internal dimensions of decoder blocks are (512, 512, 512, 256, 256, 256), internal dimensions are refinement blocks are (128, 64, 32, 1), the encoder blocks conv2d kernel size is 4, the encoder blocks conv2d kernel size is 3. Skip connections are used for the decoder blocks. The inversion model is trained with an Adam optimizer [9], the initial learning rate set to 1e-3 and weight decay set to 1e-4. It is trained for 50 epochs and each epoch containing 100 rendering iterations with a batch size of 2. The scene is changed (and the associated implicit model trained on the same scene) once every epoch and the learning rate decays once every epoch. The set of scenes used for training one inversion model belong to the same dataset.

**Training protocol.** For each training epoch of the inversion model, we randomly select a scene along with a Neural Implicit Field (NIF) trained on that scene (note that the NIF is then frozen as we train only the inversion model). Then for each iteration, we select a viewpoint with an associated image. On the rays emerging from this camera viewpoint, points are sampled and internal representations from the implicit model are extracted. These representations are then rendered by alpha composition using the opacity weights of the implicit model. The rendered internal representations are fed to the inversion network which attempts to reconstruct the grayscale GT image by minimizing a combination of L1 loss and perceptual LPIPS loss. We learn to reconstruct the grayscale images – instead of RGB – to increase the generalization power of the model across datasets and to make the model robust to color variations of certain objects.

**Evaluation protocol.** We use the FID implementation of `torcheval` metrics and the LPIPS implementation of `https://github.com/richzhang/PerceptualSimilarity.git`. These metrics are applied on grayscale images with 3 replicated channels. Overall the more visually similar are the reconstructed images to the original (grayscale) ones, the higher is the risk that the NIF contain fine grained details, hence sensitive information about the scene, implying lower privacy. We use the publicly available LLaVa mode "`liuhaotian/llava-v1.5-7b`" to describe the grayscale original and reconstructed images. Max new tokens is set to 1024 and num beams to 4. The KeyBert implementation used is from `https://github.com/MaartenGr/KeyBERT.git` is used.

### 3.1. Additional privacy baseline

**Training ppNeSF with an additional photometric loss.** In addition to ZipNeRF-wo-RGB and ppNeSF, we add another baseline called RGB-ppNeSF in which we train ppNeSF with an additional photometric loss. Note that, similar to ZipNeRF-wo-RGB, we remove the color head of RGB-ppNeSF after training. Following the privacy attack and evaluation protocol from Sec. 3 of the main paper, we train inversion models and evaluate the privacy of this new baseline on three datasets (7Scenes, Indoor6, Mip360). The reconstruction results are displayed in Tab. 3. Adding a photometric loss during training subsequently allows the inversion to recover more detailed and accurate images, indicating a degraded degree of privacy compared to ppNeSF. This further validates our hypothesis according to which using RGB supervision induces a privacy breach. This conclusion can be extended to any Nerf method using RGB supervision as they use similar architectures and optimization processes as our baselines. This further confirms that traditional NeRF-based localization methods are not privacy preserving.

In Fig. 3 we also display additional comparisons between reconstructed images from ZipNeRF-wo-RGBand ppNeSF, along with the associated LLaVa description for each image in the figure with the prompt "Precisely list the objects and details which can be seen in the image". Our approach obfuscates most of the high frequency and texture details compared to models using RGB as supervision.

**Privacy experiments with 8bit images.** In this section we explore an alternative option where instead of training NeRF models with continuous RGB images, we use discretized 8bit RGB images containing obviously much less

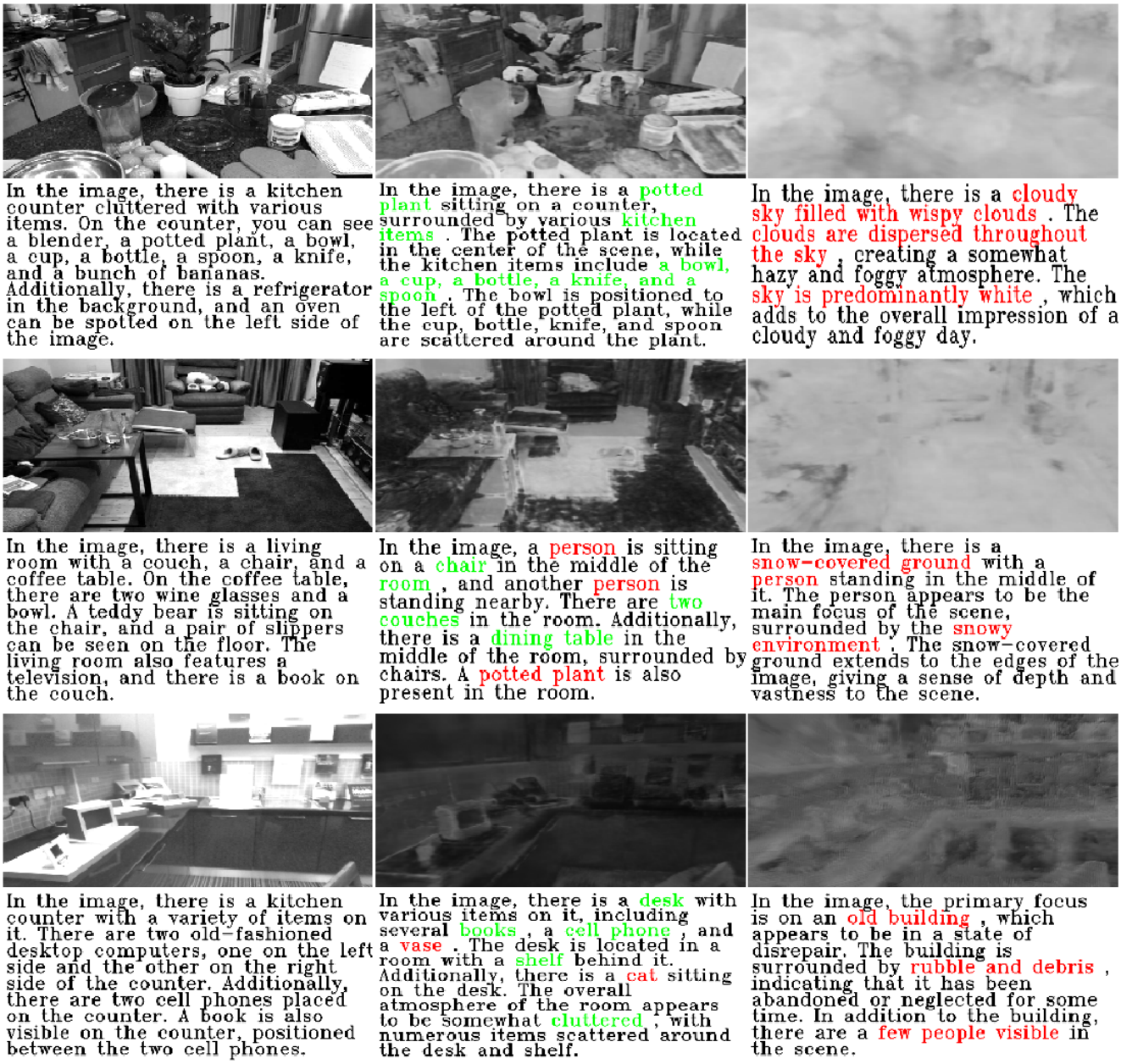


Figure 3. From left to right: Ground truth image, image reconstructed from ZipNeRF-wo-RGB, image reconstructed from ppNeSF with the inversion attack. Below each image we show the associated LlaVa's descriptions highlighting in green objects correctly identified by the VLM and in red hallucinated objects.

| Metric | ZipNeRF-wo-RGB | ZipNeRF-wo-RGB 8bits | RGB-ppNeSF | RGB-ppNeSF 8 bits | ppNeSF |
|---|---|---|---|---|---|
| LPIPS(↑) | 0.53 | 0.59 | 0.55 | 0.54 | **0.60** |
| FID(↑) | 233 | 241 | 296 | 262 | **313** |

Table 2. **Comparison to additional privacy baselines.** The inversion model is trained on the mip360 dataset and evaluated on the Chess scene from the 7Scenes dataset. We evaluate image reconstruction quality through the LPIPS(↑)/ FID (↑) metrics. Higher LPIPS/FID on images reconstructed from our privacy attack indicates a more privacy-preserving neural field model.

| | Model | Chess | Fire | Heads | Office | Pumpkin | Redkitchen | Stairs | Average |
|---|---|---|---|---|---|---|---|---|---|
| Tr360 | ZipNeRF-wo-RGB | 0.53/233 | 0.55/347 | 0.56/323 | 0.55/230 | 0.54/217 | 0.57/232 | 0.56/173 | 0.55/250 |
| | RGB-ppNeSF | 0.55/296 | 0.59/405 | 0.57/337 | 0.57/321 | 0.51/217 | 0.55/260 | 0.51/193 | 0.54/289 |
| | ppNeSF | 0.60/313 | 0.62/425 | 0.60/389 | 0.62/282 | 0.58/284 | 0.60/304 | 0.57/257 | **0.59/322** |
| | | Bicycle | Bonsai | Counter | Garden | Kitchen | Room | Stump | Average |
| Tr7S | ZipNeRF-wo-RGB | 0.69/321 | 0.49/193 | 0.53/240 | 0.64/150 | 0.62/321 | 0.57/301 | 0.66/328 | 0.6/264 |
| | RGB-ppNeSF | 0.72/241 | 0.63/362 | 0.61/391 | 0.70/269 | 0.62/306 | 0.60/284 | 0.73/413 | 0.65/323 |
| | ppNeSF | 0.81/297 | 0.71/446 | 0.72/470 | 0.81/308 | 0.74/381 | 0.69/386 | 0.81/448 | **0.76/390** |
| | | scene1 | scene2a | scene3 | scene4a | scene5 | scene6 | | Average |
| Tr 7S | ZipNeRF-wo-RGB | 0.51/262 | 0.53/372 | 0.52/334 | 0.55/267 | 0.50/253 | 0.53/302 | | 0.52/298 |
| | RGB-ppNeSF | 0.52/245 | 0.57/292 | 0.56/268 | 0.60/275 | 0.50/208 | 0.56/0.53/303 | | 0.55/265 |
| | ppNeSF | 0.56/311 | 0.61/364 | 0.57/324 | 0.61/334 | 0.56/279 | 0.57/316 | | **0.58/321** |

Table 3. We evaluate image reconstruction quality through the LPIPS(↑)/ FID (↑). We train an inversion model on one dataset (7S: Tr7S, 360: Tr360) and reconstruct images from another unseen dataset. Note that lower reconstruction quality implies better privacy.

fine-grained information. On the *Chess* scene (7Scenes dataset), we re-train our privacy baselines (ZipNeRF-wo-RGB and RGB-ppNeSF) using 8bit RGB images instead of the original RGB images. On the mip360 dataset, we train our privacy baselines (ZipNeRF-wo-RGB and RGB-ppNeSF) using 8bit RGB images and train inversion models (see Sec. 3 of the main paper for more details) to recover images on the same mip360 dataset. Finally, we apply the inversion model on the privacy baselines trained on *Chess* and evaluate the quality of the reconstructed images in Tab. 2. Replacing RGB with 8bit images slightly increases the level of privacy, but it remains much lower than ppNeSF, which does not use image level supervision. Indeed, 8bit color images still contain much more information than our segmentation maps that are based on 100 scene specific clusters. It also shows that training neural implicit fields with discretized information only still yields a coherent and accurate neural field.

### 3.2. Privacy of the segmentations

This paper tackles the privacy of NeRF models by analysing the information contained in the internal space of the neural fields and accordingly designing a solution to remove privacy sensitive content from these fields. Regarding the privacy of segmentations (either rendered or extracted images), we refer to [17], which shows that segmentations increase privacy by preventing inversion through non-injective RGB→classes mappings. Note that contrarily to [17], our segmentations are trained per scene and correspond to a quantization of PPNeSF's already privacy-preserving internal features which further increases the degree of privacy. To illustrate this, we train, on 7Scenes, inversion models taking as input query segmentation maps (Q-Segs), rendered segmentation maps (R-Segs) and rendered segmentation maps combined with internal ppNeSF features. For each modality we then evaluate the inversion model on Indoor6. We report LPIPS/FID and captions similarity between the original and recovered images in Tab. 4. We observe that segmentation still provide a much higher degree of privacy than the privacy baseline ZipNeRF-wo-RGB. Adding rendered segmentations to internal ppNeSF features only degrades very slightly the level of privacy, which further validates our assumption about the privacy of segmentations. Example visualizations are provided in Fig. 4.

## References


[1] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. In *ICCV*, 2021. 1

[2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. In *ICCV*, 2023. 1, 2, 3

[3] Shariq Farooq Bhat, Reiner Birkl, Diana Wofk, Peter Wonka, and Matthias Müller. Zoedepth: Zero-shot transfer by combining relative and metric depth. *arXiv preprint arXiv:2302.12288*, 2023. 3

[4] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial Radiance Fields. In *ECCV*, 2022. 1

[5] Anpei Chen, Zexiang Xu, Xinyue Wei, Siyu Tang, Hao Su, and Andreas Geiger. Factor Fields: A Unified Framework for Neural Fields and Beyond. arXiv:2302.01226, 2023.

[6] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance Fields without Neural Networks. In *CVPR*, 2022. 1

[7] James T. Kajiya and Brian P. Von Herzen. Ray Tracing Volume Densities. *ACM SIGGRAPH Computer Graphics*, 18 (3):0097–8930, 1984. 1

[8] Bingxin Ke, Anton Obukhov, Shengyu Huang, Nando Metzger, Rodrigo Caye Daudt, and Konrad Schindler. Repurposing Diffusion-Based Image Generators for Monocular Depth Estimation. In *CVPR*, 2024. 3

[9] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015. 5

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for

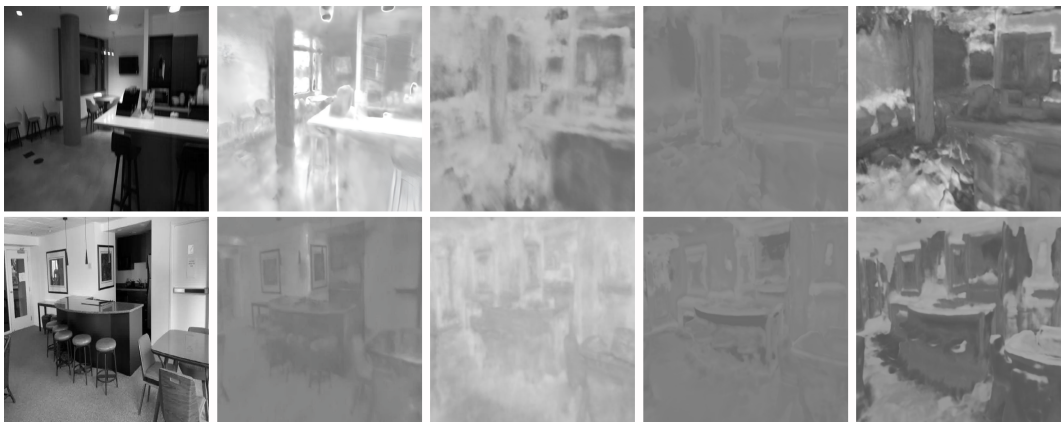

Figure 4. **Left to right**: GT image, images reconstructed from ZipNeRF-wo-RGB/ ppNeSF/ Q-Segs / ppNeSF+R-Segs. Segmentations provide little additional information.

| | LPIPS(↑)/ FID (↑) / Captions similarity (↓) | | | |
|---|---|---|---|---|
| Model | ZipNeRF-wo-RGB | PPNeSF | Q-Segs | PPNeSF+R-Segs |
| Metrics | 0.52 / 298 / 0.55 | **0.58** / 321 / 0.43 | 0.55 / **324** / **0.42** | 0.57/ 306 / 0.48 |

Table 4. **Privacy of the segmentations.** Indoor6 inversions results (trained on 7Scenes). Higher LPIPS/FID and lower captions similarity implies better privacy.


stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3

[11] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 1

[12] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In *ICCV*, 2021. 1

[13] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*, 2020. 1

[14] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Transactions on Graphics*, 41(4):1–15, 2022. 1

[15] Seungtae Nam, Daniel Rho, Jong Hwan Ko, and Eunbyung Park. Mip-Grid: Anti-aliased Grid Representations for Neural Radiance Fields. In *NeurIPS*, 2023. 1

[16] Michael Niemeyer, Jonathan T Barron, Ben Mildenhall, Mehdi SM Sajjadi, Andreas Geiger, and Noha Radwan. RegNeRF: Regularizing Neural Radiance Fields for View Synthesis from Sparse Inputs. In *CVPR*, 2022. 1

[17] Maxime Pietrantoni, Martin Humenberger, Torsten Sattler, and Gabriela Csurka. SegLoc: Learning Segmentation-Based Representations for Privacy-Preserving Visual Localization. In *CVPR*, 2023. 7

[18] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *MICCAI*, 2015. 4

[19] Prune Truong, Marie-Julie Rakotosaona, Fabian Manhardt, and Federico Tombari. SPARF: Neural Radiance Fields from Sparse and Noisy Poses. In *CVPR*, 2023. 1

[20] Guangcong Wang, Zhaoxi Chen, Chen Change Loy, and Ziwei Liu. SparseNeRF: Distilling Depth Ranking for Few-shot Novel View Synthesis. In *ICCV*, 2023. 1

[21] Liao Wang, Jiakai Zhang, Xinhang Liu, Fuqiang Zhao, Yanshun Zhang, Yingliang Zhang, Minye Wu, Jingyi Yu, and Lan Xu. Fourier PlenOctrees for Dynamic Radiance Field Rendering in Real-time. In *CVPR*, 2022. 1

[22] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *ICCV*, 2021. 1

[23] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. MonoSDF: Exploring Monocular Geometric Cues for Neural Implicit Surface Reconstruction. In *NeurIPS*, 2022. 1, 3