# Unified Spherical Frontend: Learning Rotation-Equivariant Representations of Spherical Images from Any Camera

Supplementary Material

## A. Conventions

In this paper, *location*, *coordinate*, or *geometry* refers to a point on the unit sphere, represented either by Cartesian coordinates in a right-handed system:

$$\mathbf{p} = (x, y, z) \in \mathbb{R}^3, \quad \|\mathbf{p}\| = 1 \tag{1}$$

or by polar coordinates:

$$(\theta, \phi) \in \mathbb{S}^2, \quad \theta \in [-\frac{\pi}{2}, \frac{\pi}{2}], \quad \phi \in [-\pi, \pi] \tag{2}$$

where $\theta$ and $\phi$ are latitudes and longitudes. $(x, y, z) = (1, 0, 0)$ corresponds to $(\theta, \phi) = (0, 0)$, and the bidirectional mapping between Cartesian and polar coordinates is given below:

$$\begin{cases} x = \cos\theta\cos\phi \\ y = \cos\theta\sin\phi \\ z = \sin\theta \end{cases}, \begin{cases} \theta = \arctan\left(\frac{z}{\sqrt{x^2+y^2}}\right) \\ \phi = \arctan\left(\frac{y}{x}\right) \end{cases} \tag{3}$$

Both Cartesian and polar coordinates are *2D* parameterizations of the same manifold. However, the spherical parameterization exhibits singularities at the poles: all $(\frac{\pi}{2}, \phi)$ map to the north pole, and all $(-\frac{\pi}{2}, \phi)$ to the south pole, regardless of $\phi$.

## B. Location Sampling Method

This section details point generation methods.

### B.1. Goldberg Polyhedron

Polyhedral sampling generates spherical points by subdividing each polygon face of a base convex polyhedron (e.g., tetrahedron, hexahedron, dodecahedron) with generalized barycentric coordinates. Let a polygon face have $m$ vertices $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{S}^2$. For a subdivision level $n_{\text{side}}$, we enumerate all integer tuples:

$$(i_1, \ldots, i_m),\ i_k \geq 0,\ \sum_{k=1}^{m} i_k = n_{\text{side}}, \tag{4}$$

Each tuple defines a generalized barycentric combination inside the face:

$$\mathbf{p} = \sum_{k=1}^{m} \frac{i_k}{n_{\text{side}}} \mathbf{v}_k. \tag{5}$$

After normalization to unit norm, this yields a point on the sphere. Applying this process to all faces yields a near-uniform spherical point set. Different polyhedra use the same subdivision rule but differ in how $N_{\text{side}}$ relates to the target average pixel area.

In this work, "Goldberg polyhedron" refers only to the *spherical Voronoi* tessellation induced by the sampled points, which typically manifests as a mixture of pentagons and hexagons. The sampling itself is purely barycentric.

### B.2. HEALPix

Refer to the HEALPix Primer[6].

### B.3. Fibonacci Lattice

$$\mathbf{p}_i = (\sin\phi_i\cos\theta_i,\ \sin\phi_i\sin\theta_i,\ \cos\phi_i), \tag{6}$$

$$\phi_i = \arccos\left(1 - \frac{2i}{N}\right), \tag{7}$$

$$\theta_i = \frac{2\pi \cdot i}{\varphi},\ \varphi = \frac{1+\sqrt{5}}{2}. \tag{8}$$

Where $\varphi$ is the golden ratio, $N$ is the total number of points. Note that $\phi$ and $\theta$ here are different from the definition in Sec. A.

### B.4. Quasi-Random Sampling

Quasi-random sampling generates low-discrepancy sequences similar to Sobol or Halton sequences.

First, generate evenly distributed points in $[0, 1]^2$ with irrational-ratio recurrence. Specifically, the plastic constant $\psi \approx 1.32471795724474602596$, the unique real root of $\psi^3 - \psi - 1 = 0$, is adopted as the irrational number. Define two incommensurate step sizes $\begin{cases} \alpha_u &= \psi^{-1} \approx 0.7549, \\ \alpha_v &= \psi^{-2} \approx 0.5698 \end{cases}$ together with starting offsets $s_{0,u} = s_{0,v} = 0.5$, 2D quasi-random sequence can be generated by:

$$u_i = (s_{0,u} + i \cdot \alpha_u) \bmod 1, \tag{9}$$

$$v_i = (s_{0,v} + i \cdot \alpha_v) \bmod 1, \tag{10}$$

for $i = 0, \ldots, N-1$.

Then, each point $(u_i, v_i) \in [0, 1]^2$ is mapped to the unit sphere with the Lambert equal-area projection:

$$\mathbf{p}_i = (r_i\cos\phi_i,\ r_i\sin\phi_i, z_i)\,, \tag{11}$$

$$z_i = 1 - 2u_i, \tag{12}$$

$$r_i = \sqrt{1 - z_i^2}, \tag{13}$$

$$\phi_i = 2\pi \cdot v_i. \tag{14}$$

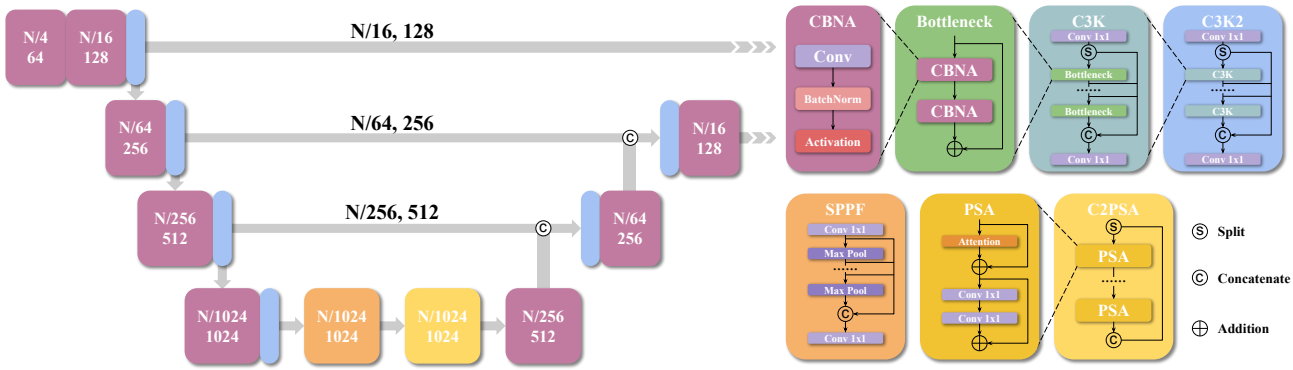



Figure 1. **YOLOv11.** Core components include: **CBNA**: convolution, batch normalization, activation; **SPPF**: spatial pyramid pooling-fast; **PSA**: Partial Spatial Attention.

## C. Value Interpolation RBF Kernel

For an output location $\mathbf{p}_o$, let $\mathcal{N}(p_o)$ be the collection of neighbor input points and $\omega_k$ be the weight applied to input value $x_k$ at input location $\mathbf{p}_k$. $d(\mathbf{p}_i, \mathbf{p}_j)$ is the geodesic distance between point $\mathbf{p}_i$ and $\mathbf{p}_j$.
The interpolated value $x_o$ at a new output location $\mathbf{p}_o$ is expressed as:

$$x_o = \sum_{k \in \mathcal{N}(\mathbf{p}_o)} \omega_k \, x_k \tag{15}$$

Where weights $\omega_k$ may come from different RBF kernels. Weights are normalized so that their sum equals 1 for an output value.

### C.1. Softmax

$$\omega_k = \frac{e^{-\frac{d(\mathbf{p}_k, \mathbf{p}_o)}{\mathcal{T}}}}{\sum_{i \in \mathcal{N}(\mathbf{p}_o)} e^{-\frac{d(\mathbf{p}_i, \mathbf{p}_o)}{\mathcal{T}}}} \tag{16}$$

$\mathcal{T}$ is the temperature hyperparameter that controls sharpness.

### C.2. Gaussian

$$\omega_k = e^{-\frac{d(\mathbf{p}_k, \mathbf{p}_o)^2}{2\sigma^2}} \tag{17}$$

### C.3. Hann

[9]:

$$\omega_k = \frac{1}{2}\left(1 + \cos\left(\pi \cdot d(\mathbf{p}_k, \mathbf{p}_o)\right)\right) \tag{18}$$

### C.4. Wendland-C2

[1]:

$$\omega_k = \left(1 - d(\mathbf{p}_k, \mathbf{p}_o)\right)^4 \cdot \left(1 + 4 \cdot d(\mathbf{p}_k, \mathbf{p}_o)\right) \tag{19}$$

## D. Continuous Spherical Spatial Correlation

Analogous to a corollary in Fourier analysis for planar CNNs, we prove that spherical correlation in the spatial domain corresponds to multiplication in the spherical harmonic (frequency) domain.

Let $f, \mathcal{K} : \mathbb{S}^2 \to \mathbb{C}$ denote signals on $\mathbb{S}^2$, we define their spherical correlation $f \star \mathcal{K}$ by:

$$(f \star \mathcal{K})(\omega) = \int_{\mathbb{S}^2} f(\omega') \, \overline{\mathcal{K}(R_\omega^{-1}\omega')} \, d\Omega(\omega'), \tag{20}$$

Where $\omega, \omega' \in \mathbb{S}^2$, $R_\omega$ is any fixed rotation in $\mathrm{SO}(3)$ such that $R_\omega(\mathbf{n}) = \omega$ with $\mathbf{n} = (0, 0, 1)$ denoting the north pole. $d\Omega(\omega')$ is the surface measure on $\mathbb{S}^2$, and $\bar{\cdot}$ denotes complex conjugation.

Any $f \in L^2(\mathbb{S}^2) : \mathbb{S}^2 \to \mathbb{C}$ admits a spherical harmonic expansion:

$$f(\omega) = \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} \hat{f}_{\ell,m} \, Y_\ell^m(\omega), \tag{21}$$

$$\hat{f}_{\ell,m} = \int_{\mathbb{S}^2} f(\omega) \, \overline{Y_\ell^m(\omega)} \, d\Omega(\omega), \tag{22}$$

Where $\{Y_\ell^m\}$ are harmonic functions that form an orthonormal basis of $L^2(\mathbb{S}^2)$, likewise for $\mathcal{K}(\omega)$.

We now derive the spherical harmonic coefficients of $f \star \mathcal{K}$

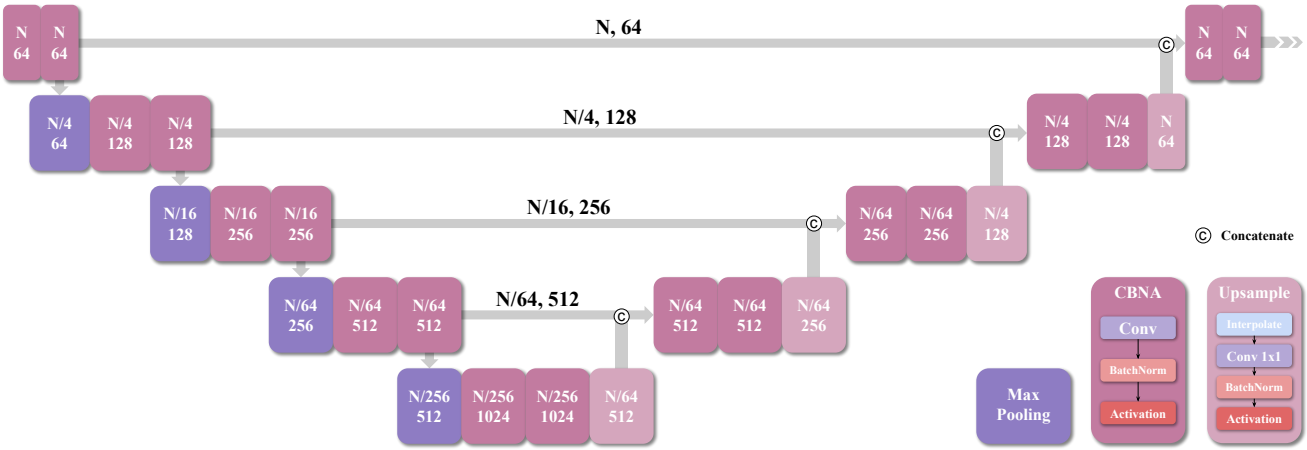



Figure 2. **UNet.** Core components include: **CBNA**: convolution, batch normalization, activation; **Upsample**: value interpolation, 1x1 channel-wise convolution, batch normalization, activation.

as:

$$\widehat{(f \star \mathcal{K})}_{\ell,m}$$

$$= \int_{\mathbb{S}^2} (f \star \mathcal{K})(\omega)\, \overline{Y_\ell^m(\omega)}\, d\Omega(\omega) \tag{23}$$

$$= \int_{\mathbb{S}^2} \left[ \int_{\mathbb{S}^2} f(\omega')\, \overline{\mathcal{K}(R_\omega^{-1}\omega')}\, d\Omega(\omega') \right] \overline{Y_\ell^m(\omega)}\, d\Omega(\omega) \tag{24}$$

$$= \int_{\mathbb{S}^2} f(\omega') \left[ \int_{\mathbb{S}^2} \overline{\mathcal{K}(R_\omega^{-1}\omega')}\, \overline{Y_\ell^m(\omega)}\, d\Omega(\omega) \right] d\Omega(\omega') \tag{25}$$

For general anisotropic kernel $\mathcal{K}$, $\widehat{(f \star \mathcal{K})}_{\ell,m}$ becomes a finite sum of products $\hat{f}_{\ell,m}\ \overline{\hat{\mathcal{K}}_{\ell,m}}$ multiplied by algebraic factors[3, 10]:

$$\widehat{(f \star \mathcal{K})}_{\ell,m} = \sum_{m'=-\ell}^{\ell} \hat{f}_{\ell,m'}\, \overline{\hat{\mathcal{K}}_{\ell,m'}}\, \Lambda_\ell(m, m'). \tag{26}$$

Which corresponds to multiplication in the spherical harmonic domain.

## E. Rotation Equivariant Correlation

As detailed in[4], when the kernel $\mathcal{K}$ is isotropic, meaning that it is zonal, radial, isotropic, and its value only depends on the relative geodesic distance, $\widehat{(f \star \mathcal{K})}_{\ell,m}$ simplifies to:

$$\widehat{(f \star \mathcal{K})}_{\ell,m} = \alpha_\ell\, \hat{f}_{\ell,m} \tag{27}$$

Which is purely diagonal pointwise multiplication in $(\ell, m)$, where $\alpha_\ell$ can be regarded as the kernel's "*frequency response*", instead of "*phase response*" ($m$-mode coefficients) that changes with rotation, hence preserving rotation-equivariance.

Intuitively, if we rotate all the coordinates $\omega$ of a spherical image with any rotation $R \in \mathrm{SO}(3)$, $\mathcal{K}$ activates $R\omega$ with the same set of weights for local signals around $R\omega$ regardless of orientation because it's isotropic.

## F. Computation Optimization

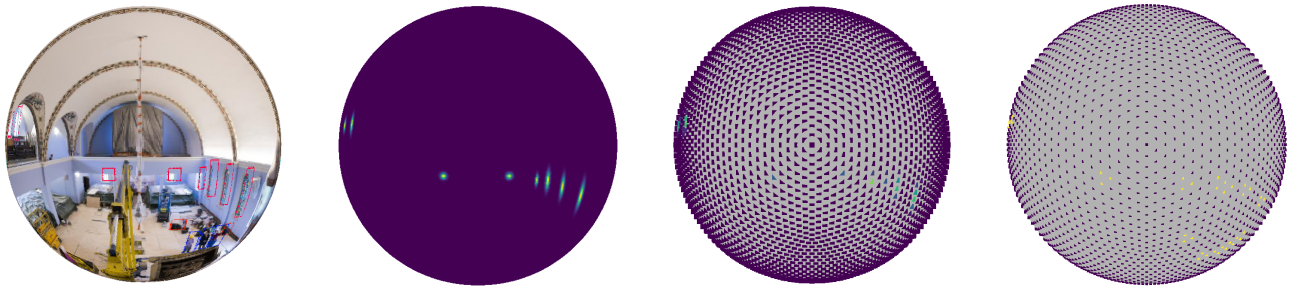
A key design consideration of USF is that operations contraction schemes changes only with respect to geometry and not feature values. This allows expensive geometric computations such as neighborhood construction, interpolation weights, and sparse aggregation structures to be precomputed once and reused across subsequent forward passes. This geometry caching mechanism is critical for making high-resolution spherical processing computationally feasible in practice.

Our implementation utilizes several optimized libraries for efficient spherical processing. We use *FAISS* [7] for fast nearest-neighbor search during neighborhood construction, *torch_scatter* [5] for efficient neighborhood aggregation and interpolation, and *opt_einsum* [12] for optimized tensor contraction ordering in dense operations. These components significantly reduce overhead compared to naive implementations and make large-scale spherical processing feasible within the PyTorch framework, even without custom CUDA kernels.

In all benchmarks, the input to both spherical and planar pipelines is a $960 \times 480$ panorama image with batch size 8 and RGB channels. All experiment results are averaged over 10 runs on an NVIDIA H200 GPU using PyTorch 2.8.0, CUDA 12.8, torch_scatter 2.1.2, and float32 precision.

### F.1. Location Sampling and Value Interpolation

The location sampling resolution factor is set to 1.0, meaning the number of output spherical samples is matched to the



(a) **Spherical Image with RBFoVs.** Red boxes denote windows.

(b) **Continuous Heatmap.** Per-category gaussian heatmaps are generated through gnomonic projection regardless of the point distribution on the sphere.

(c) **Discrete Heatmap.** Used in actual supervision.

(d) **Regression Mask.** L1 regression loss only supervises the yellow points of regression head outputs, where RBFoVs are centered.

Figure 3. **Object Detection Supervision.**

number of input pixels with less than 1% difference. The sampling benchmark measures the runtime of location sampling and value interpolation separately under both cold-start (first run) and sustained (cached) settings. In the cold-start scenario, geometric structures such as spherical neighborhood structure and interpolation weights must be constructed. In the sustained setting, these structures are reused and only value aggregation via matrix multiplication is performed.

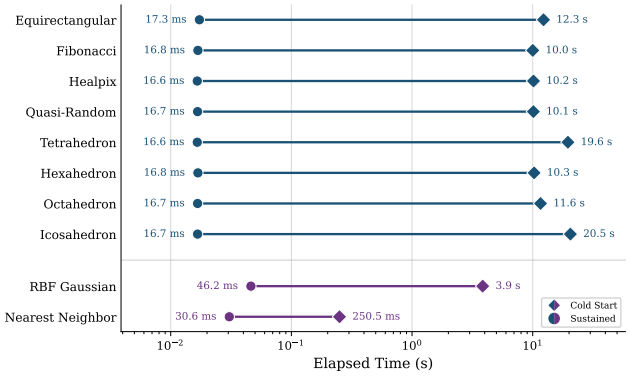



Figure 4. **Location Sampling and Value Interpolation Benchmarks.** Cold-start runtime includes geometric preprocessing such as neighborhood construction and interpolation weight computation. Sustained runtime reuses geometry and only performs value aggregation via matrix multiplication.

As shown in Fig. 4, the cold-start cost is dominated by geometric preprocessing and can take several seconds depending on the sampling method and interpolation kernel. However, once geometry is cached, the runtime drops to the millisecond level across all sampling methods, representing orders-of-magnitude speedup. This demonstrates that the computational bottleneck lies in geometry construction rather than interpolation itself, and confirms that geometry caching is essential for practical spherical processing.

This result highlights a fundamental property of spherical pipelines: while geometry-aware processing introduces an initial preprocessing cost, this cost is amortized over all subsequent forward passes, making sustained runtime comparable to standard image processing pipelines.

## F.2. Spherical Convolution and Pooling

We next benchmark individual spherical convolution and pooling operators and compare them with planar Conv2d and MaxPool2d layers. For fairness, the planar convolution uses a $5 \times 5$ kernel, since each spherical output location aggregates approximately $25 - 30$ neighboring samples on average, making the receptive field sizes comparable.

Three spherical convolution implementations are evaluated:

1. **Continuous Distance $\times$ Direction MLP:** Distance and direction weighting functions are parameterized by an MLP with hidden dims $[16, 16]$ and positional encoding $L = 8$. During training, the MLP must be evaluated for each neighbor pair, but during inference, weights can be fully cached once geometry is fixed.
2. **Discrete Distance Piecewise-Constant (PWC) (Dense):** The MLP is replaced by discrete distance bins implemented using torch embedding.
3. **Discrete Distance Piecewise-Constant (PWC) (Sparse):** Aggregation is performed by sparse matrix multiplication.

Figure 5 show that cold-start runtime is significantly higher due to kernel construction and sparse structure generation. However, sustained runtime drops substantially once geometry and weights are cached. Among spherical implementations, the discrete PWC (sparse) implementation achieves the fastest sustained runtime, while the continuous MLP variant benefits significantly from weight caching during inference.

Although spherical operators remain slower than highly optimized planar Conv2d kernels, the gap narrows significantly in sustained execution. This difference is largely due to the fact that planar convolutions are implemented as heav-

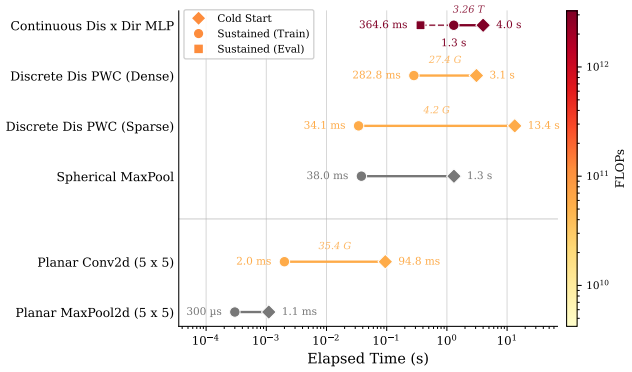



Figure 5. **Spherical Convolution and Pooling Benchmarks.** Sustained inference benefits from weight caching and reduces spherical convolution to matrix multiplication.

ily optimized CUDA kernels, whereas spherical operators are currently implemented using PyTorch scatter and sparse operations rather than custom fused CUDA kernels.

### F.3. Network-Level Comparison

Finally, we compare full network runtime between planar and spherical versions of YOLOv11, DeepLab v3, and UNet using identical macro-architectures, input resolution, batch size, and channel configurations. The only difference between models is the replacement of planar convolution and pooling layers with spherical counterparts, making the comparison strictly architecture-level and apples-to-apples.

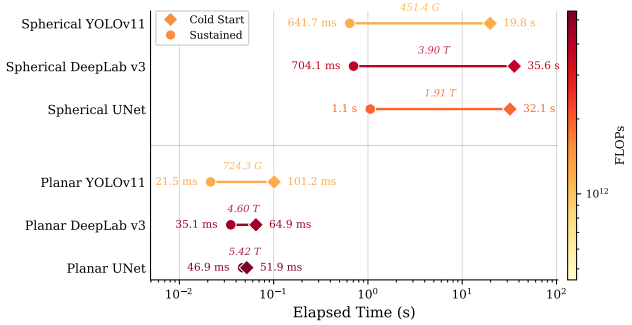



Figure 6. **End-to-End Network-Level Comparison Benchmarks.** Spherical networks achieve comparable runtime while requiring fewer FLOPs but currently lack fully optimized CUDA kernels.

Figure 6 shows that cold-start runtime for spherical networks is significantly higher due to geometry construction and kernel initialization. However, in sustained execution, spherical networks become much closer to planar networks in runtime. Across models, spherical networks take approximately $\mathbf{2}\times$ **the overall time** of their planar counterparts in full-scale training, which we consider acceptable given the additional geometric processing and built-in rotation-equivariance. Interestingly, the spherical networks often require **fewer FLOPs** than the planar counterparts while still taking longer in wall-clock time. This phenomenon is common in custom operator research: theoretical arithmetic complexity (FLOPs) does not directly translate to runtime when standard operators benefit from highly optimized low-level CUDA kernels, while custom operators rely on higher-level sparse and scatter operations. Further performance gains could be achieved by implementing dedicated CUDA kernels for spherical sampling and convolution.

Overall, these benchmarks demonstrate that **geometry caching is the key enabler** that makes high-resolution spherical processing practical. Without caching, the cost of geometric preprocessing would dominate runtime and make spherical pipelines infeasible for large-scale vision tasks. With caching, however, sustained runtime becomes comparable to planar networks while providing additional geometric consistency, rotation-equivariance, and lens-agnostic processing capabilities.

## G. Common Training Setup

If not otherwise mentioned in the main paper, all the full-scale experiments share the same configurations as follows:

- Icosahedron location sampler for both the resampling stage and all the intermediate output locations of spherical convolution and pooling layers.
- *AdamW* optimizer with learning rate $1 \times 10^{-3}$, weight decay $1 \times 10^{-2}$, learning rate scheduler warms up at the first $40\%$ steps and drops to $1 \times 10^{-2}$ of the learning rate at the end.
- Batch size of 8 with 2 distributed data parallel (DDP), trained for 200 epochs on NVIDIA A100 and H200 GPUs.
- Data augmentation consists of $50\%$ chance of random chroma, luma jitter, gaussian blur, $5\%$ chance of grayscaling, and $50\%$ chance of random horizontal reflection.

## H. Backbone Architecture

We applied our framework to three representative backbone architectures: YOLOv11 (Fig. 1), UNet (Fig. 2), and DeepLab v3 (Fig. 7).

We incorporate attention bias as positional encoding in the C2PSA self-attention layers with pairwise Euclidean distance in the planar domain and geodesic distance on the sphere, analogous to the mechanism in ALiBi[11].

Note that atrous (dilated) convolution is not defined for our generic spherical CNN, so we use larger kernels instead in ASPP block of spherical DeepLab v3.

Each downsampling layer in the spherical model reduces the number of output points to $\frac{1}{4}$ of the input, analogous to stride-2 downsampling in planar CNNs. Because location sampling is not invertible with respect to resolution factors, we align the output locations of spherical CNN upsampling

| Train \ Test | | Pinhole | | Fisheye | | Panoramic | |
|---|---|---|---|---|---|---|---|
| | | mIoU ↑ | mAcc ↑ | mIoU ↑ | mAcc ↑ | mIoU ↑ | mAcc ↑ |
| Planar DeepLab v3[2] | Pinhole | 42.53% | 55.75% | 33.53% | 47.36% | 36.07% | 53.61% |
| | Fisheye | 39.88% | 53.46% | 40.05% | 55.86% | 33.11% | 56.53% |
| | Panoramic | 29.66% | 43.85% | 24.91% | 37.54% | 35.01% | 58.30% |
| Spherical DeepLab v3 | Pinhole | 34.76% | 47.47% | 22.36% | 35.52% | 19.70% | 35.09% |
| | Fisheye | 19.44% | 31.52% | 30.44% | 44.21% | 28.16% | 43.99% |
| | Panoramic | 12.57% | 23.05% | 28.35% | 41.58% | 28.78% | 45.27% |

Table 1. **Semantic Segmentation Full-dataset Lens Adaptability Test.** Random Rotation is disabled.

layers with corresponding downsampling layers to ensure valid channel concatenations.

## I. Object Detection

The loss function includes focal loss for category classification and L1 loss for bounding box center $(\theta, \phi)$, size $(\alpha, \beta)$, and angle $\gamma$ regression. Formally, focal loss:

$$L_{\text{focal}} = \frac{1}{N}\sum_{i=1}^{N}[L_{\text{positive}} + L_{\text{negative}}], \tag{28}$$

$$L_{\text{positive}} = -\ln(p_i)(1-p_i)^{\lambda_{\text{positive}}} y_i, \tag{29}$$

$$L_{\text{negative}} = -\ln(1-p_i)p_i^{\lambda_{\text{positive}}}(1-y_i)^{\lambda_{\text{negative}}}. \tag{30}$$

Where $p_i$ is the predicted probability, $y_i$ is the ground truth heatmap, and $\lambda_{\text{positive}}, \lambda_{\text{negative}}$ are hyperparameter exponents controlling the relative focus on positive versus negative detections.

**Efficient Pairwise IoU.** We propose a generic and vectorized method to compute pairwise IoU between arbitrary-shaped bounding boxes. Each bounding box is converted into a binary mask over the same set of dense and uniformly sampled spherical points. The intersection and union are then approximated by applying logical AND and OR operations on these masks, with IoU estimated as the ratio between the number of points inside the intersection and those in the union.

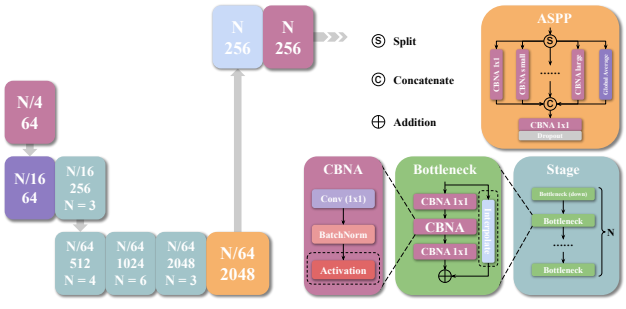



Figure 7. **DeepLab v3.** Core components include: **CBNA**: convolution, batch normalization, activation; **Stage**: N Bottleneck blocks; **ASPP**: atrous spatial pyramid pooling.

Finally, we apply Matrix Non-Maximum Suppression (NMS)[13] with a Gaussian decay factor $\sigma = 5.0$ and score threshold 0.3 to suppress overlapping but inconfident proposals.

As illustrated in Fig. 3, classification heatmap and bounding box regression are supervised with separate loss functions at different locations, and regression losses are only applied to points near the object's center.

Qualitative results are in Tab. 2 in addition to the quantitative results in the main paper. Observe the cyan boxes carefully for light patches on the ground in the NR and RR cases of Spherical YOLOv11, which exhibits the same orientation with or without random rotation. This behavior is expected because orientation is the raw output of one head, which should not change under rotation because of rotation-equivariance.

## J. Semantic Segmentation

Training uses a composite loss of 70% cross-entropy and 30% Dice loss, with proper class weights and 0.05 label smoothing. Formally, Dice loss:

$$L_{\text{Dice}} = 1 - \frac{1}{|S|}\sum_{c\in S}\text{Dice}_c, \tag{31}$$

$$\text{Dice}_c = \frac{2\cdot\sum_{i\in\Omega} p_{i,c}\, y_{i,c}}{\sum_{i\in\Omega} p_{i,c} + \sum_{i\in\Omega} y_{i,c}}. \tag{32}$$

Where $p_{i,c}$ and $y_{i,c}$ denote the predicted probability and ground-truth label for class $c$ at pixel $i$, $\Omega$ is the set of valid pixels, and $S$ is the set of non-ignored classes in $\Omega$.

We follow the official 3-fold cross-validation scheme as a benchmark on the Stanford-2D-3D-S dataset.

Random rotation qualitative results are visualized in Tab. 3 in addition to the quantitative results in the main paper. Zero-shot lens adaptability results trained on the full-scale dataset

are shown in Tab. 1 and Tab. 4. Ideally, a perfect model that adapts to all lenses would have the same performance in all the entries of the square performance matrix. USF reduces off-diagonal performance degradations compared to planar models, but not yet completely eliminates them.

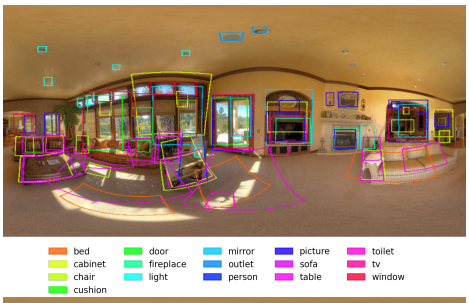
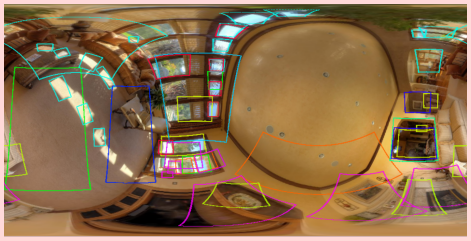
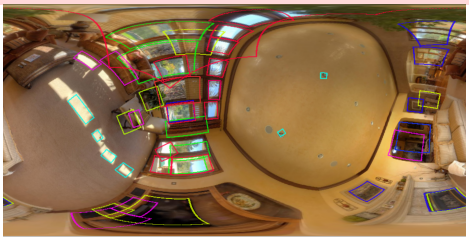
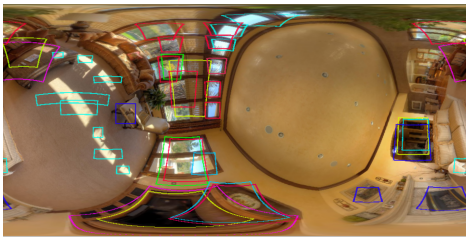
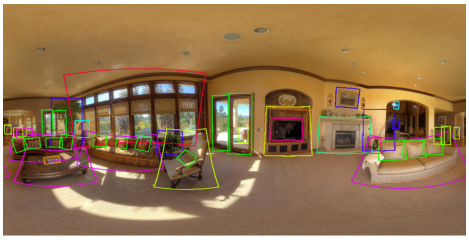
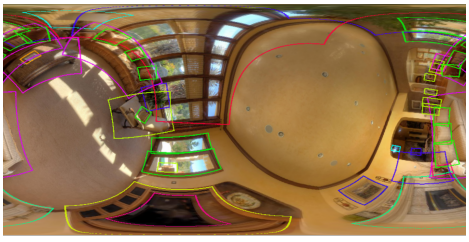
| Train \ Test | | NR | RR |
|---|---|---|---|
| Planar YOLOv11[8] | NR |  |  |
| | RR | |  |
| Spherical YOLOv11 | NR | |  |
| Ground Truth | |  |  |

Table 2. **Object Detection Qualitative Result.**

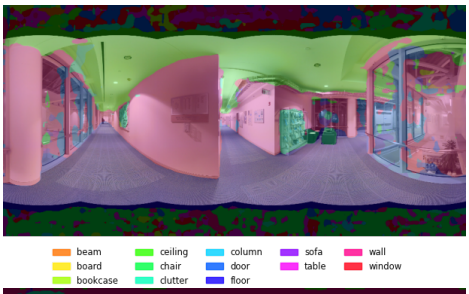
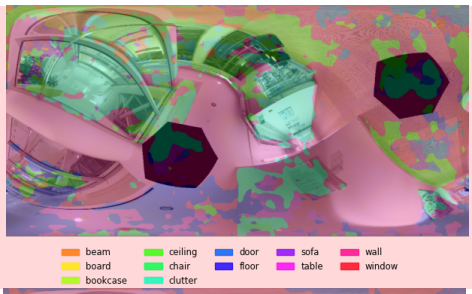
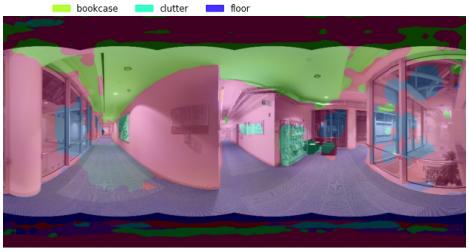
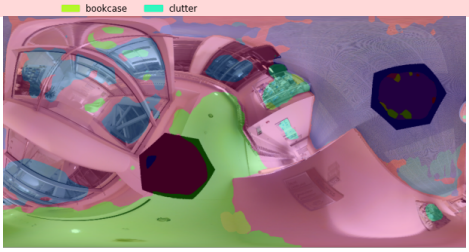
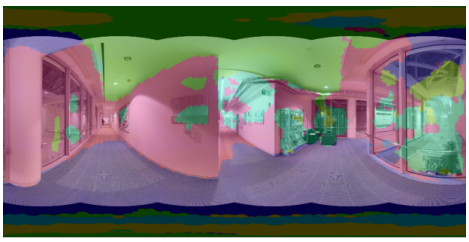
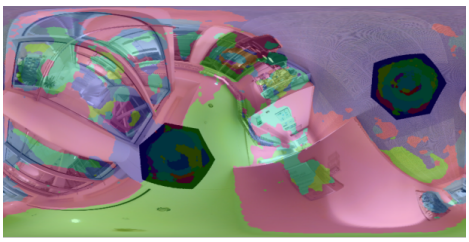
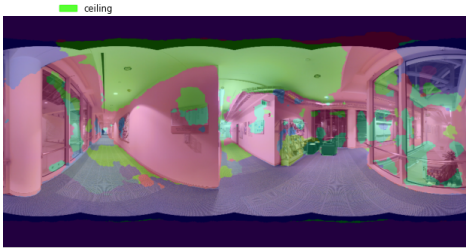
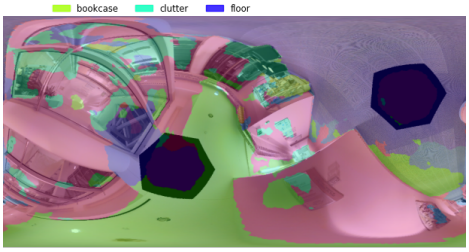
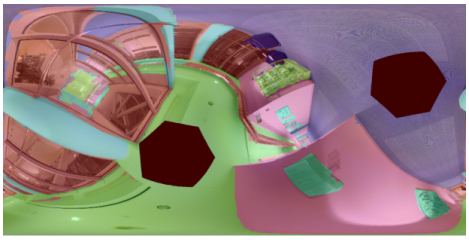
| Train \ Test | | NR | RR |
|---|---|---|---|
| Planar DeepLab v3[2] | NR |   |   |
| | RR |   |   |
| Spherical DeepLab v3 | NR |   |   |
| | RR |   |   |
| Ground Truth | | |   |

Table 3. **Semantic Segmentation Qualitative Result.**

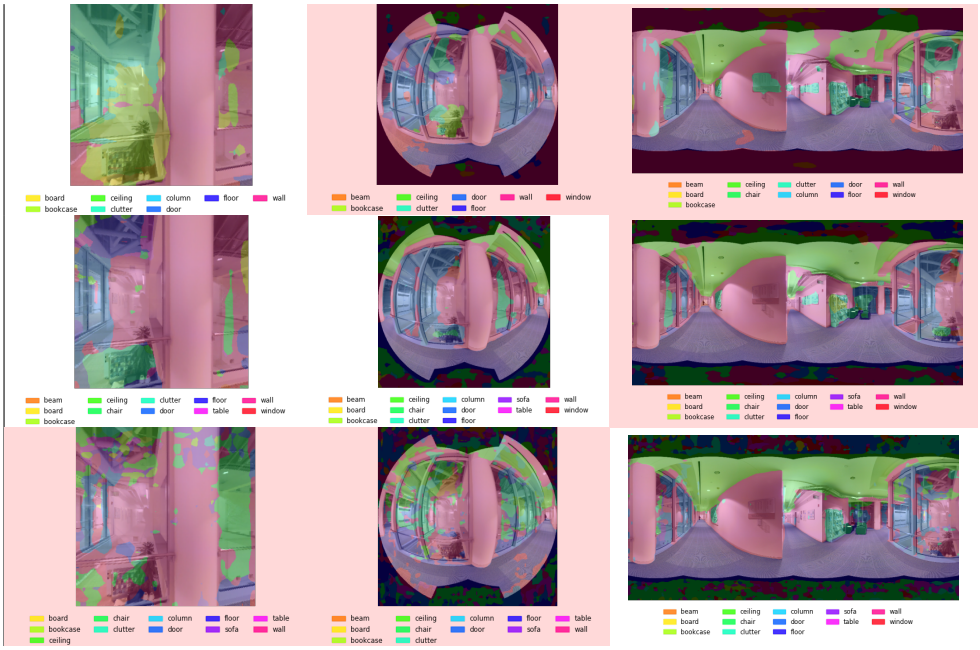
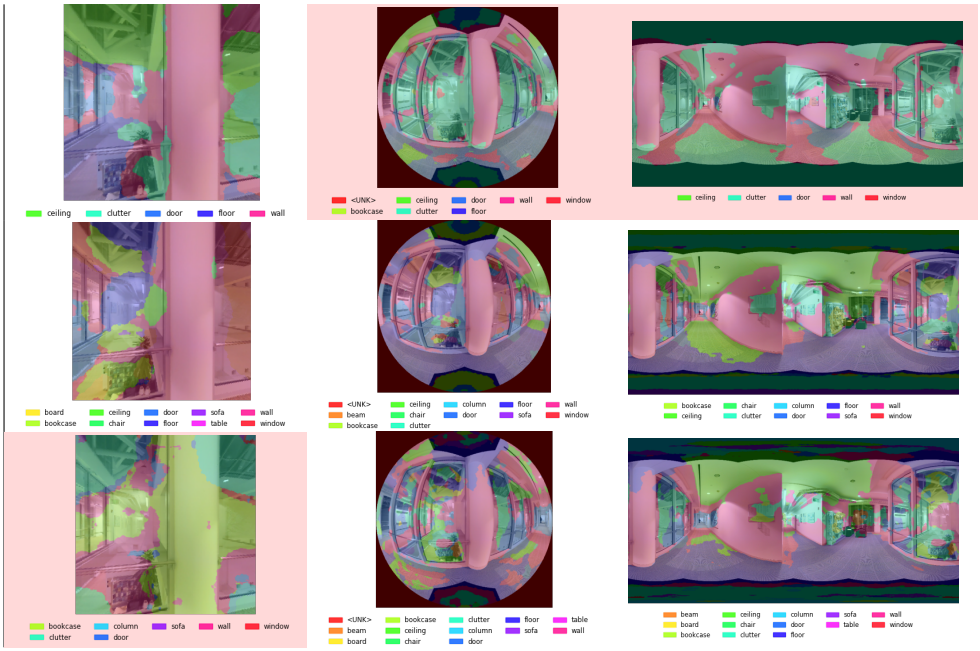
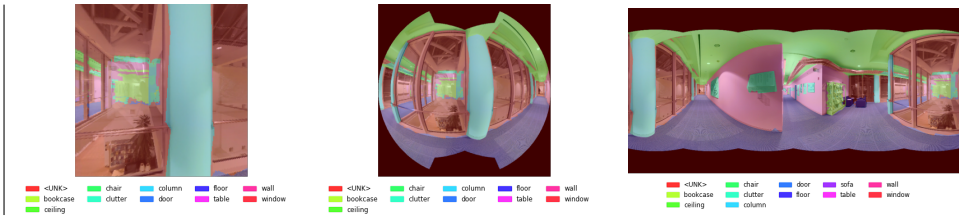
| Train \ Test | Pinhole | Fisheye | Panoramic |
|---|---|---|---|
| Planar DeepLab v3[2] | Pinhole |  | |
| | Fisheye | | |
| | Panoramic | | |
| Spherical DeepLab v3 | Pinhole |  | |
| | Fisheye | | |
| | Panoramic | | |
| Ground Truth | |  | |

Table 4. **Semantic Segmentation Full-dataset Lens Adaptability Test.** Random Rotation is disabled.

## References


[1] Marco Evangelos Biancolini, Andrea Chiappa, Ubaldo Cella, Emiliano Costa, Corrado Groth, and Stefano Porziani. Radial Basis Functions Mesh Morphing. In *Computational Science – ICCS 2020*, pages 294–308, Cham, 2020. Springer International Publishing. 2

[2] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation, 2017. 6, 9, 10

[3] J. R. Driscoll and D. M. Healy. Computing Fourier Transforms and Convolutions on the 2-Sphere. *Advances in Applied Mathematics*, 15(2):202–250, 1994. 3

[4] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning SO(3) Equivariant Representations with Spherical CNNs. *International Journal of Computer Vision*, 128(3):588–600, 2020. 3

[5] Matthias Fey. torch_scatter: PyTorch extension library of optimized scatter operations. GitHub Repository: https://github.com/rusty1s/pytorch_scatter, 2023. Version 2.1.2. 3

[6] Krzysztof M. Gorski, Benjamin D. Wandelt, Frode K. Hansen, Eric Hivon, and Anthony J. Banday. The HEALPix Primer, 1999. arXiv:astro-ph/9905275. 1

[7] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021. 3

[8] Rahima Khanam and Muhammad Hussain. YOLOv11: An Overview of the Key Architectural Enhancements, 2024. 8

[9] Andi Kivinukk and Gert Tamberg. On Sampling Operators Defined By The Hann Window And Some Of Their Extensions. *Sampling Theory in Signal and Image Processing*, 2 (3):235–257, 2003. 2

[10] Peter J. Kostelec and Daniel N. Rockmore. FFTs on the Rotation Group. *Journal of Fourier Analysis and Applications*, 14(2):145–179, 2008. 3

[11] Ofir Press, Noah Smith, and Mike Lewis. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. In *International Conference on Learning Representations*, 2022. 5

[12] Daniel G. A. Smith and Johnnie Gray. Opt_einsum - A Python package for optimizing contraction order for einsum-like expressions. *Journal of Open Source Software*, 3(26):753, 2018. 3

[13] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. SOLOv2: Dynamic and Fast Instance Segmentation. In *Advances in Neural Information Processing Systems*, pages 17721–17732. Curran Associates, Inc., 2020. 6