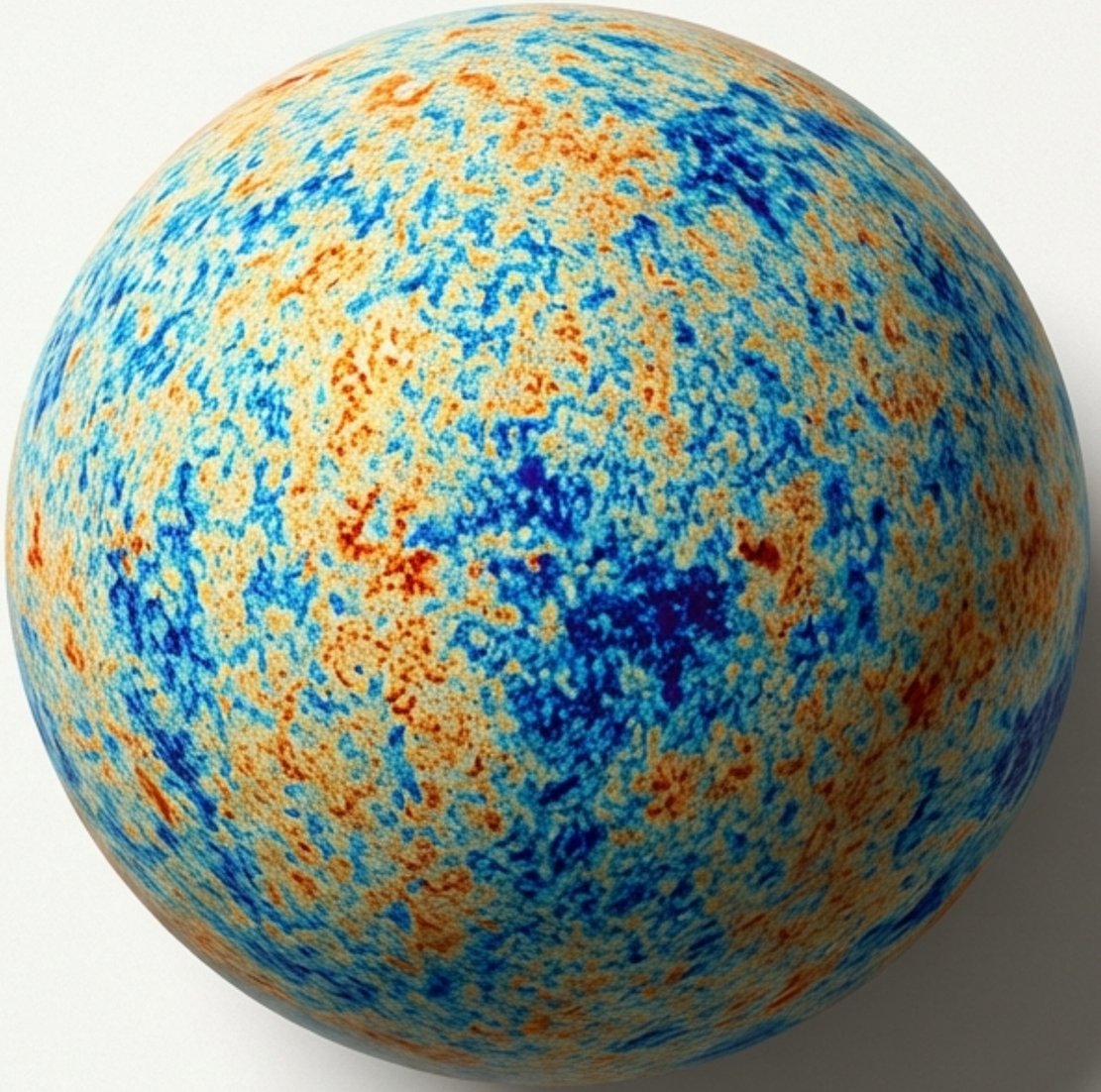
The Shape of Data: AI for a Spherical World

An Introduction to DeepSphere



Join the Geometric AI Revolution

This power is not theoretical. DeepSphere is open-source, well-documented, and ready for you to use on your own challenges.



https://github.com/deepsphere

Implementations



TensorFlow PyTorch

Integration



Works with standard libraries like PyTorch Geometric

Acknowledgments

DeepSphere was developed by researchers including Michaël Defferrard, Nathanaël Perraudin, and colleagues at the École Polytechnique Fédérale de Lausanne (EPFL), the Swiss Data Science Center (SDSC), and the Paul Scherrer Institut (PSI).

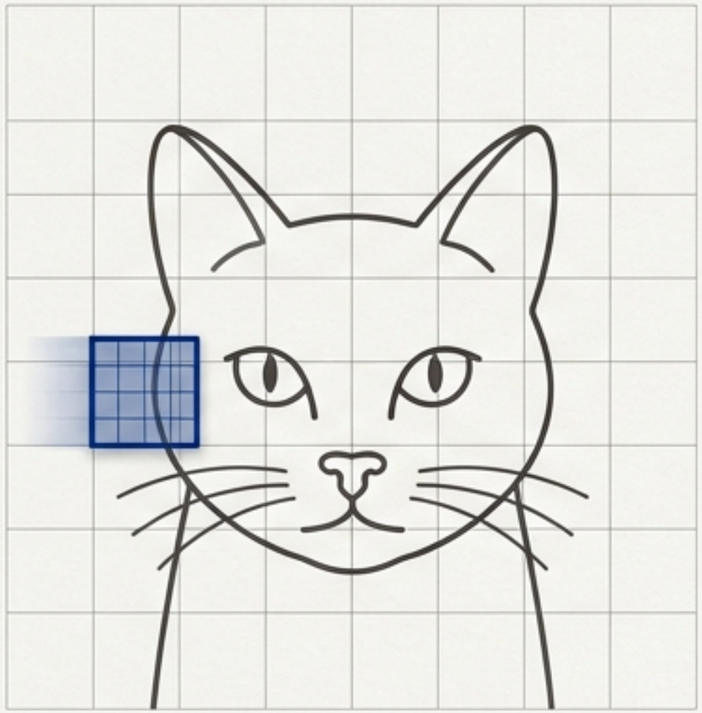


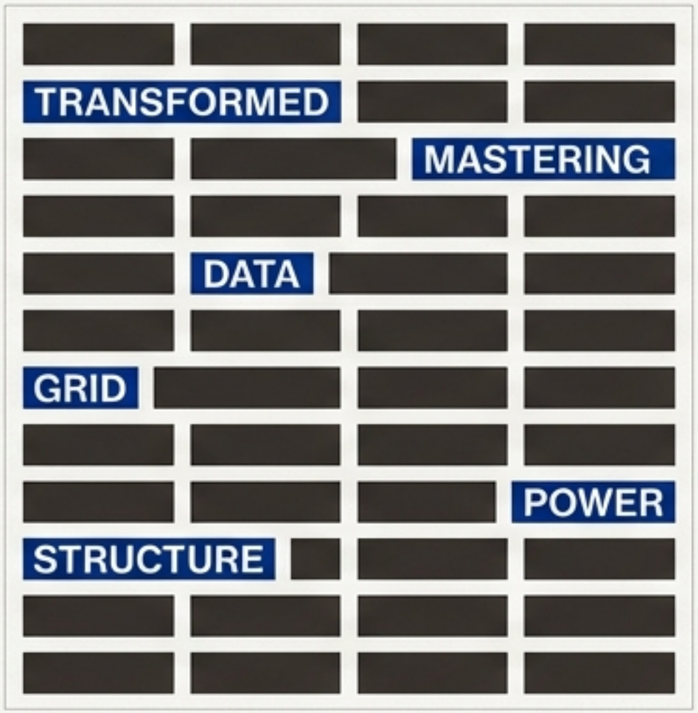




The Unstoppable Power of the Grid

Convolutional Neural Networks have transformed AI by mastering data on a grid. Their power comes from exploiting this regular, predictable structure.

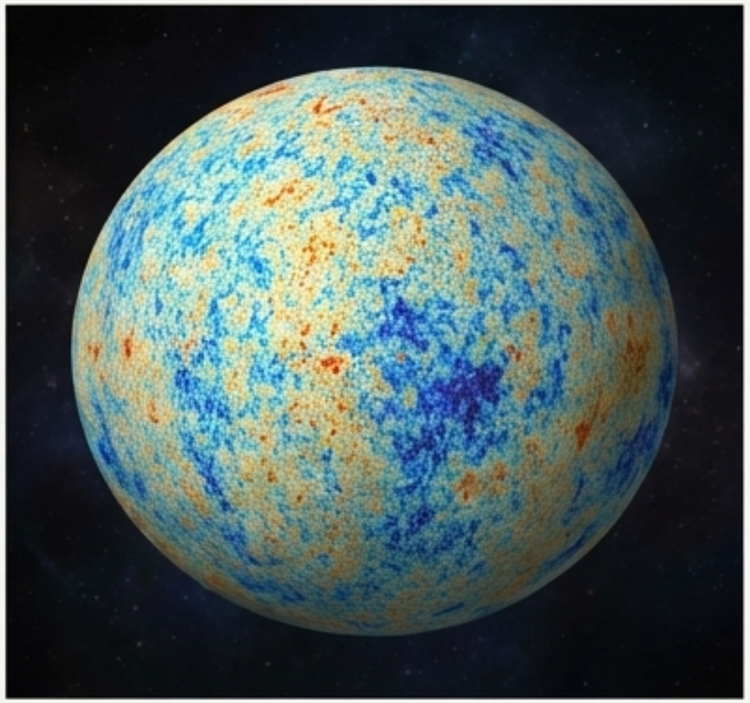




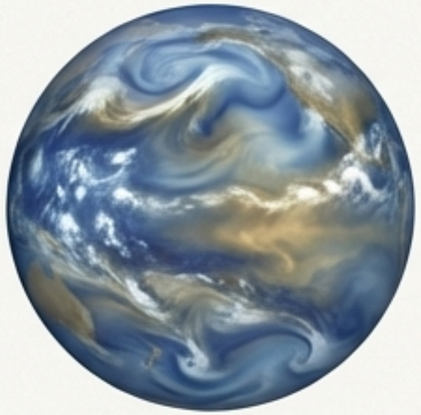


But much of the universe isn’t flat.

How do we apply the power of CNNs when data lives on a sphere?



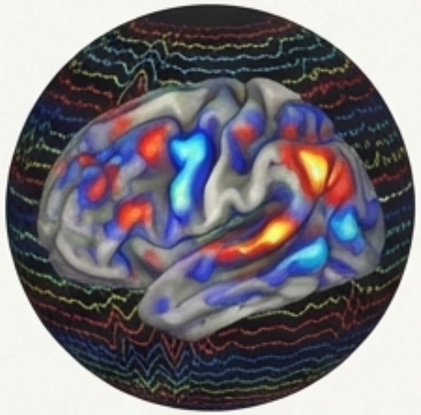
CMB temperature map (Planck 2015)



CAM5 Climate Simulation
(total precipitable water)



Simulated weak lensing
mass map



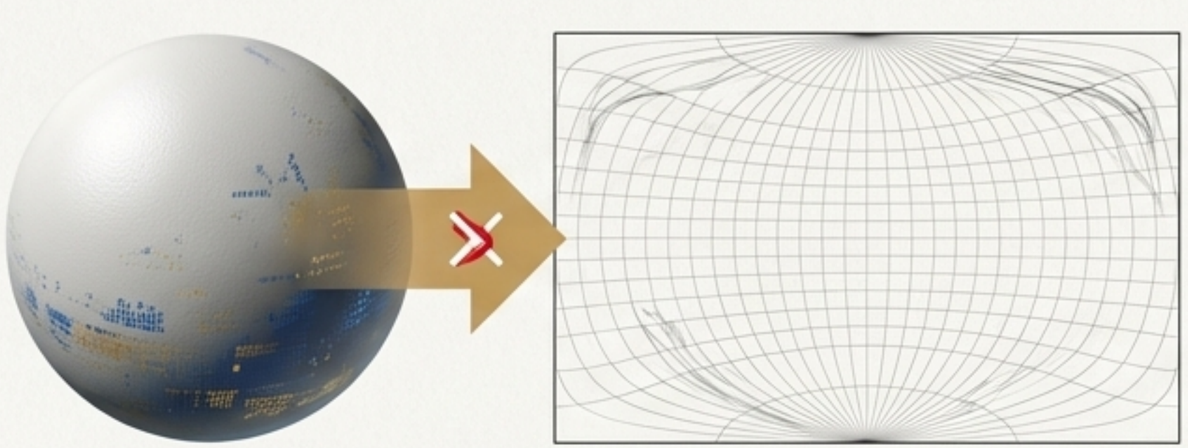
Brain activity (MEG evoked potential)

The critical trade-off: **Rotation Equivariance** vs. **Computational Efficiency.**

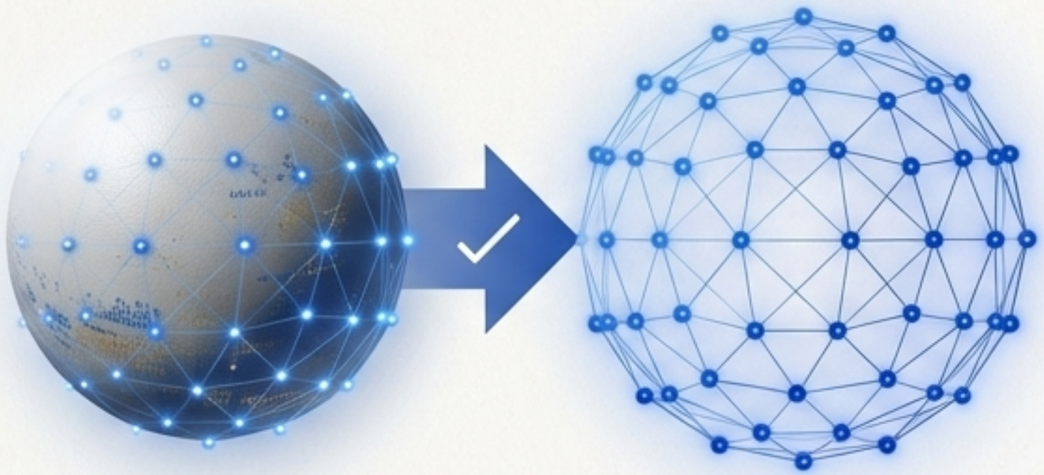
A New Paradigm: Geometric Deep Learning

Let the model learn from the data's true shape.

Instead of forcing spherical data onto a distorted flat grid, Geometric AI builds models that are native to the data's geometry. This isn't a workaround; it's a fundamentally new approach.



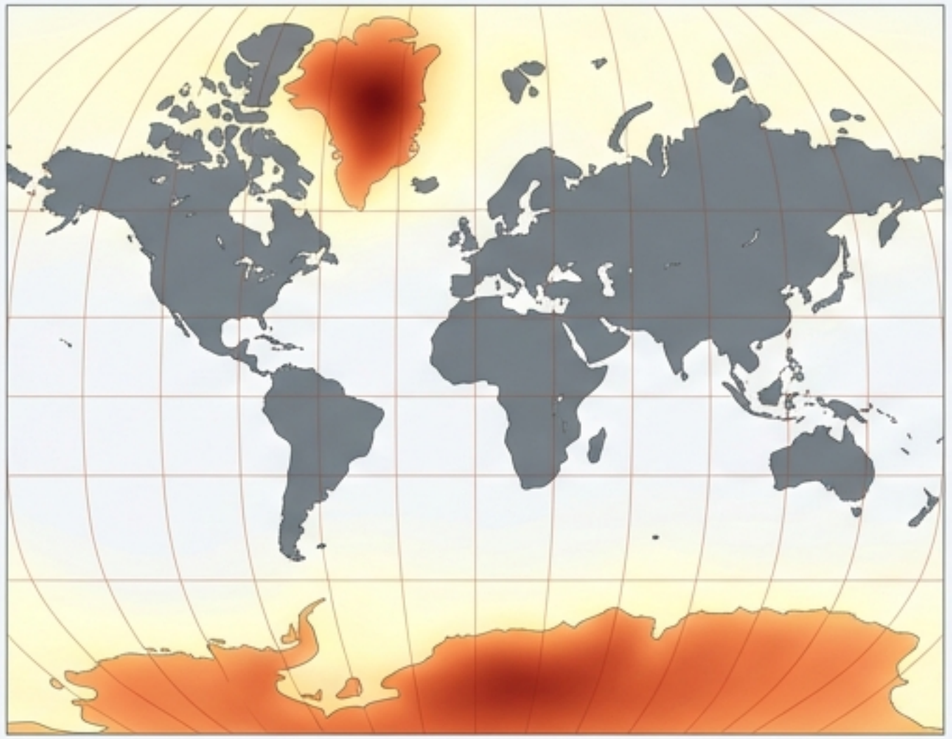
Problematic Flat Projection



Elegant Geometric Network

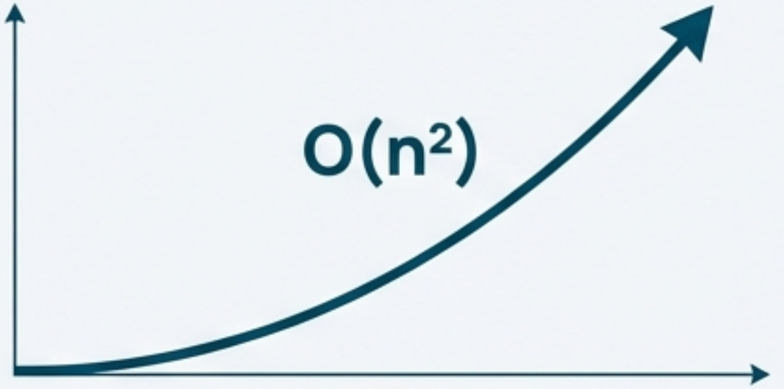
Traditional AI struggles with spherical data, leading to distortion or high computational cost.

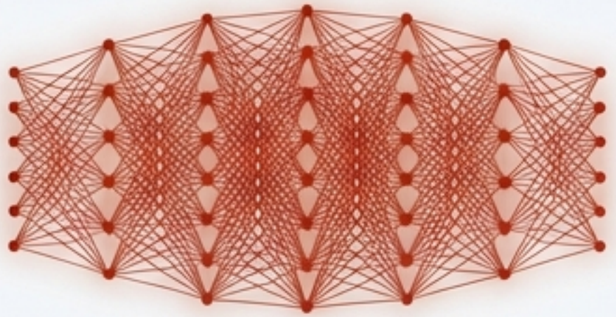
The Distortion Problem



Projecting to a 2D grid creates artifacts and breaks spatial relationships, hurting model accuracy.

The Cost Problem

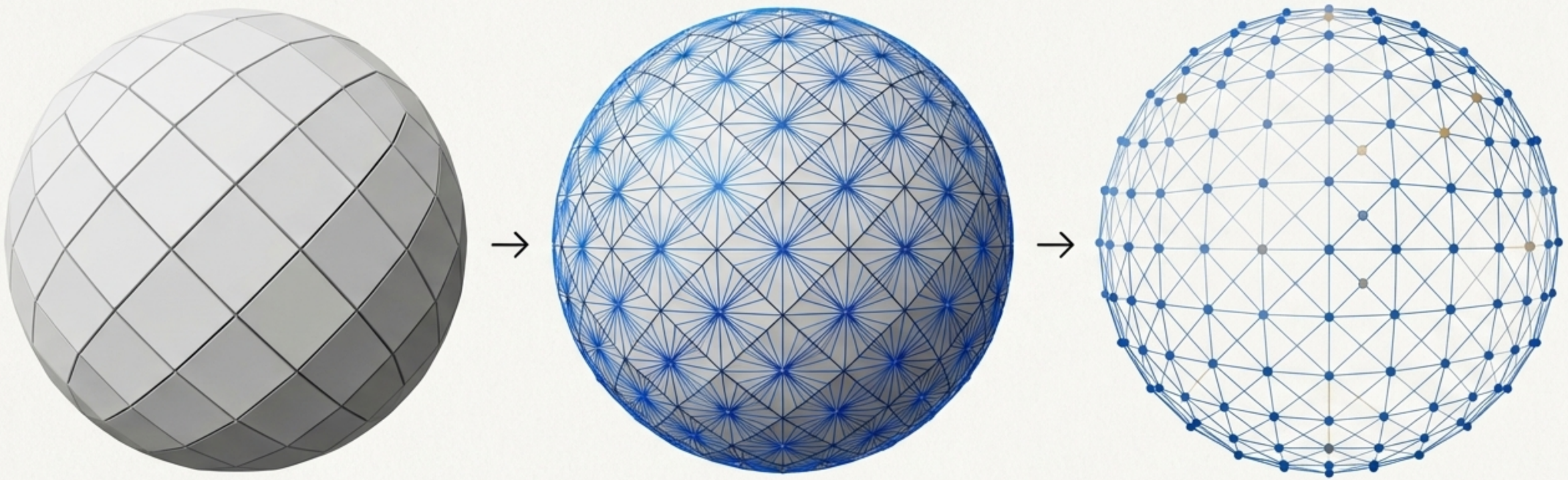




True spherical convolutions using Spherical Harmonics are rotation-equivariant but computationally expensive ($O(n^2)$ in general), limiting their use on high-resolution data.

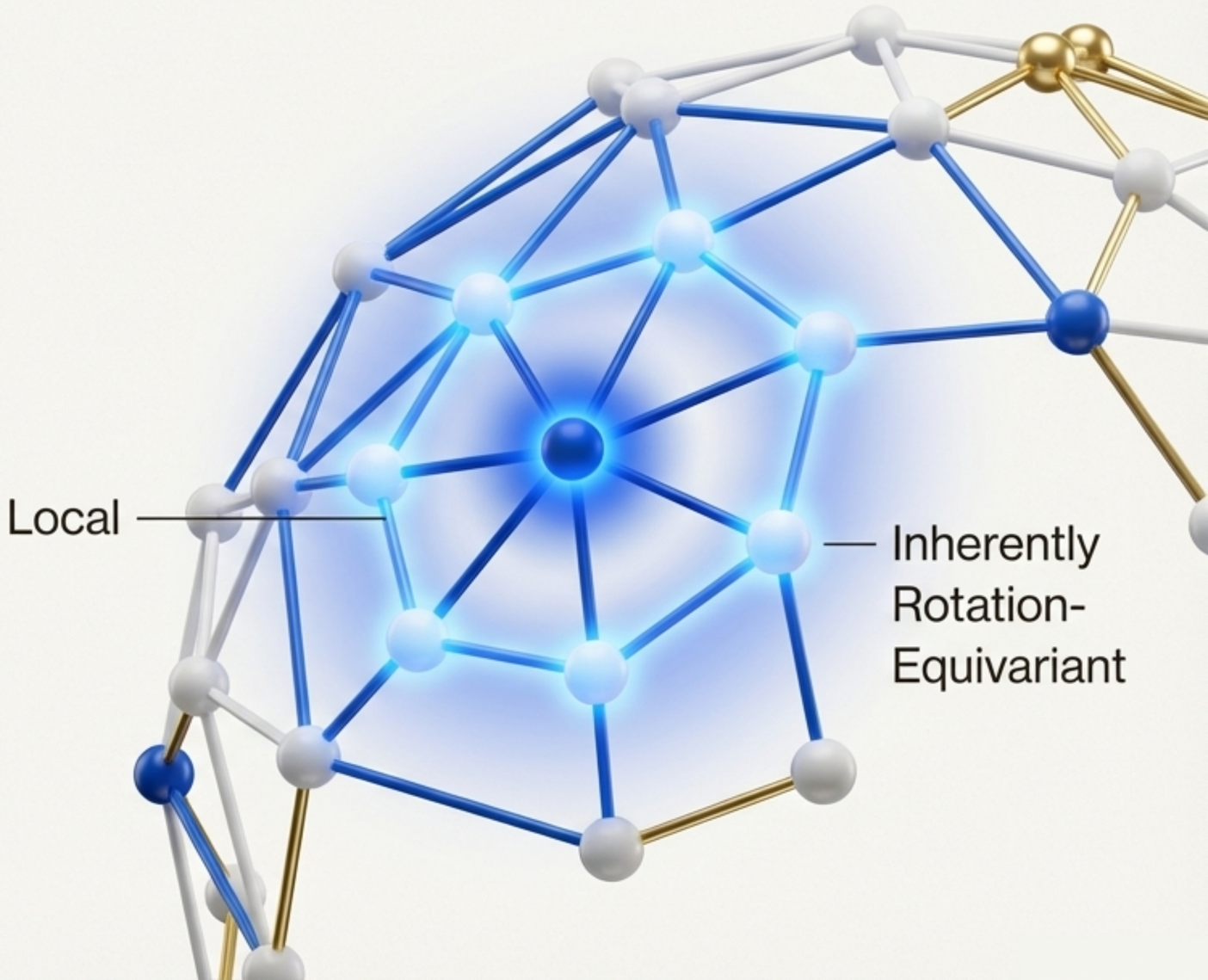
The Key Insight: A Sphere is a Graph

DeepSphere reimagines the discretized sphere not as a collection of pixels, but as a graph of connected neighbors.



Redefining “Convolution” on a Graph

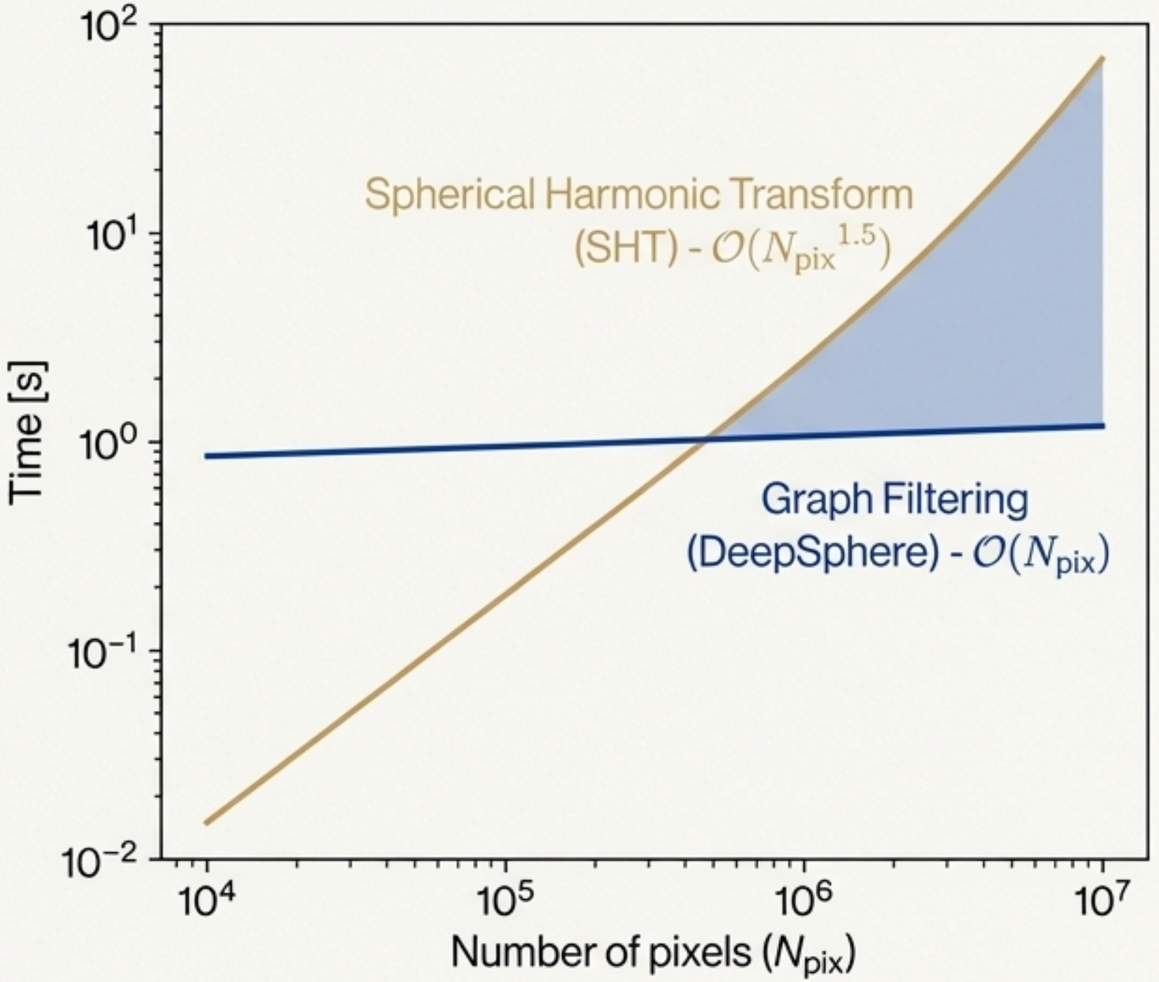
Instead of sliding a fixed filter patch across a grid, graph convolution works by passing information between connected neighbors. It’s like a ripple spreading locally from each node.



Elegance and Efficiency, Without Compromise.

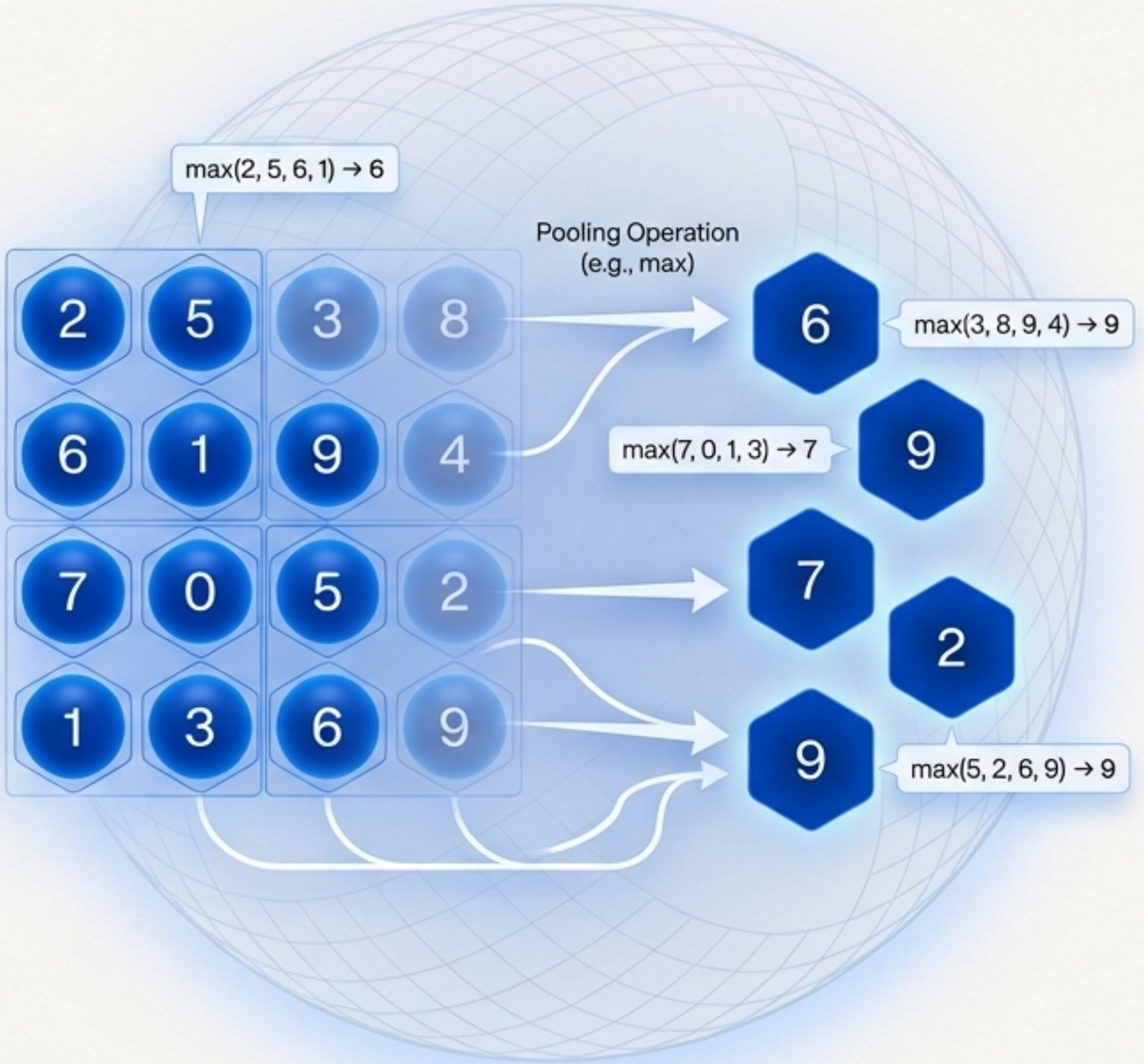
DeepSphere achieves the geometric precision of spherical methods with the computational speed of grid-based CNNs.

This is achieved by using an efficient polynomial approximation (Chebyshev polynomials) of the spectral filters, avoiding the need for a full, slow Fourier transform.
Equity Text A Regular.



Seeing the Big Picture at Multiple Scales

Just as traditional CNNs zoom out to see larger patterns, DeepSphere leverages the HEALPix hierarchical structure to analyze data at multiple resolutions.

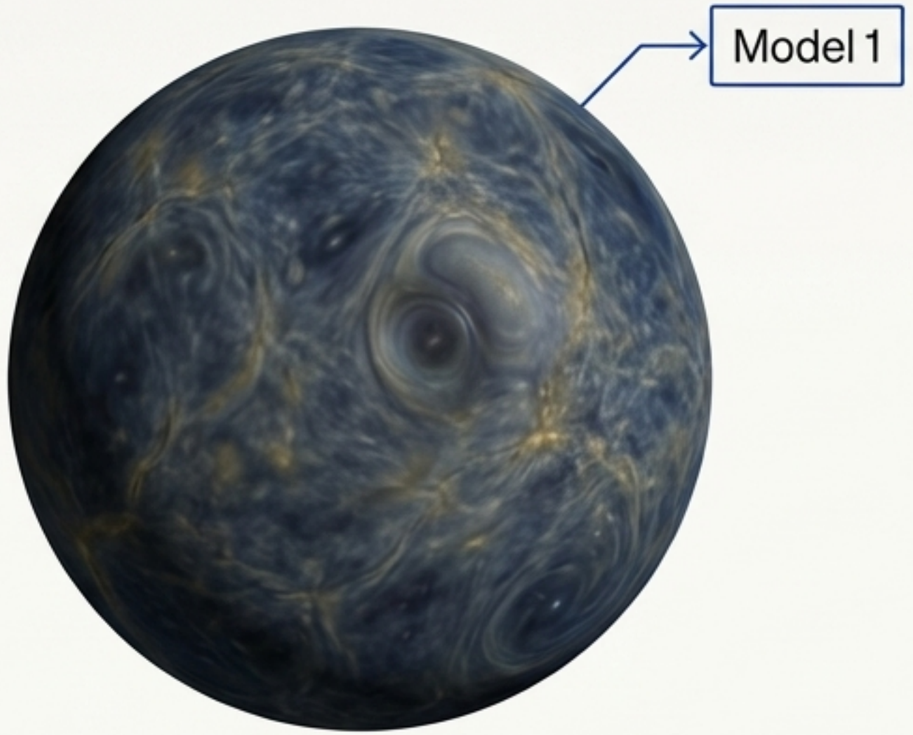


A Flexible Architecture for Diverse Scientific Tasks

DeepSphere is a general-purpose architecture that can be adapted for a wide range of prediction tasks.

Global Tasks: Classifying an Entire Sky

Is this map from Universe A or Universe B?



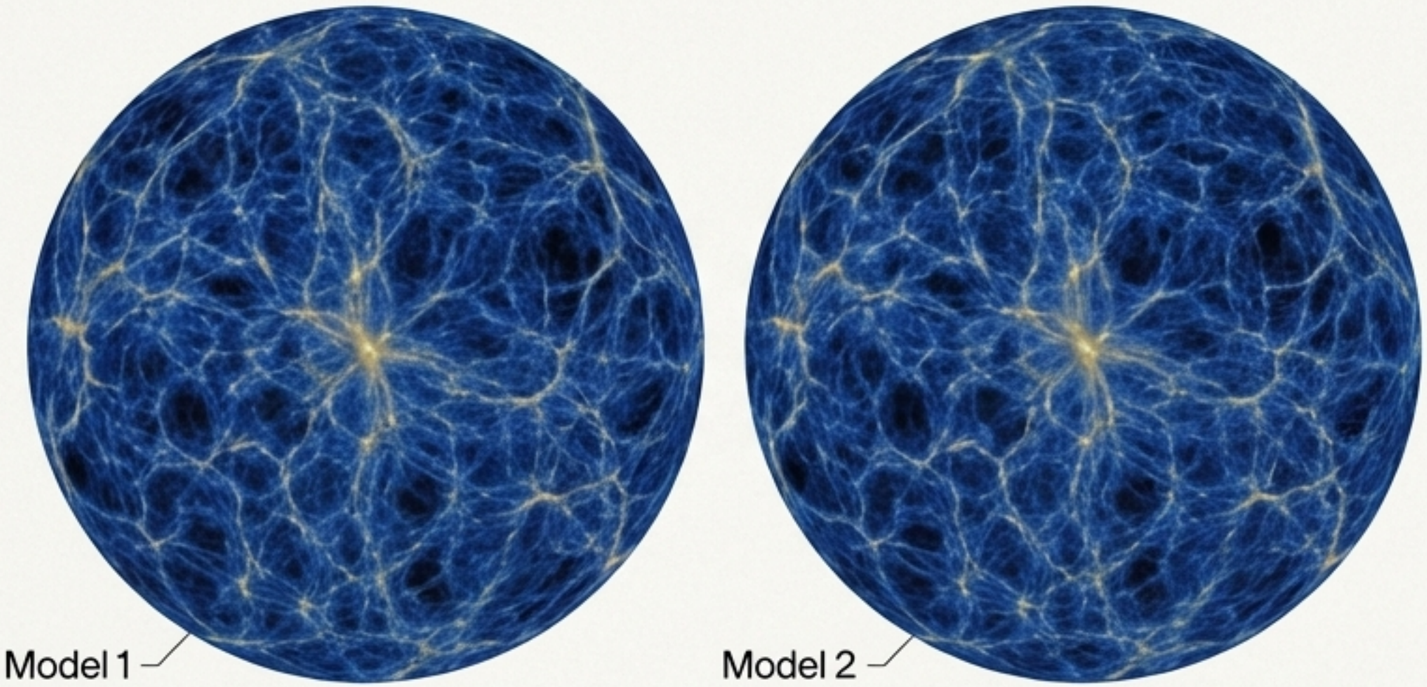
Dense Tasks: Segmenting Features on a Planet

Where are the cyclones in this climate map?

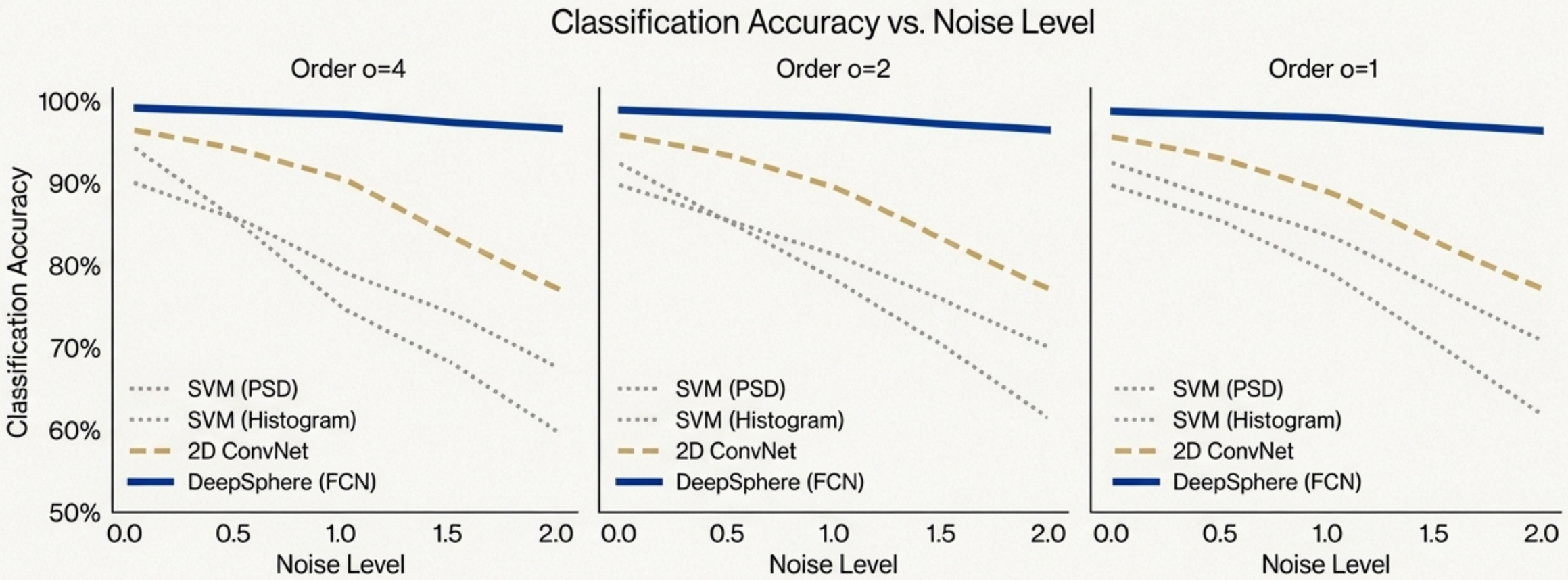


The Challenge: Can AI Distinguish Between Universes?

We trained DeepSphere on a difficult cosmological discrimination problem: Classify weak lensing mass maps from two different cosmological models. The models ($\Omega_m = 0.31, \sigma_8 = 0.82$ vs. $\Omega_m = 0.26, \sigma_8 = 0.91$) were specifically designed to have nearly identical power spectra, making them extremely difficult to distinguish with traditional methods.



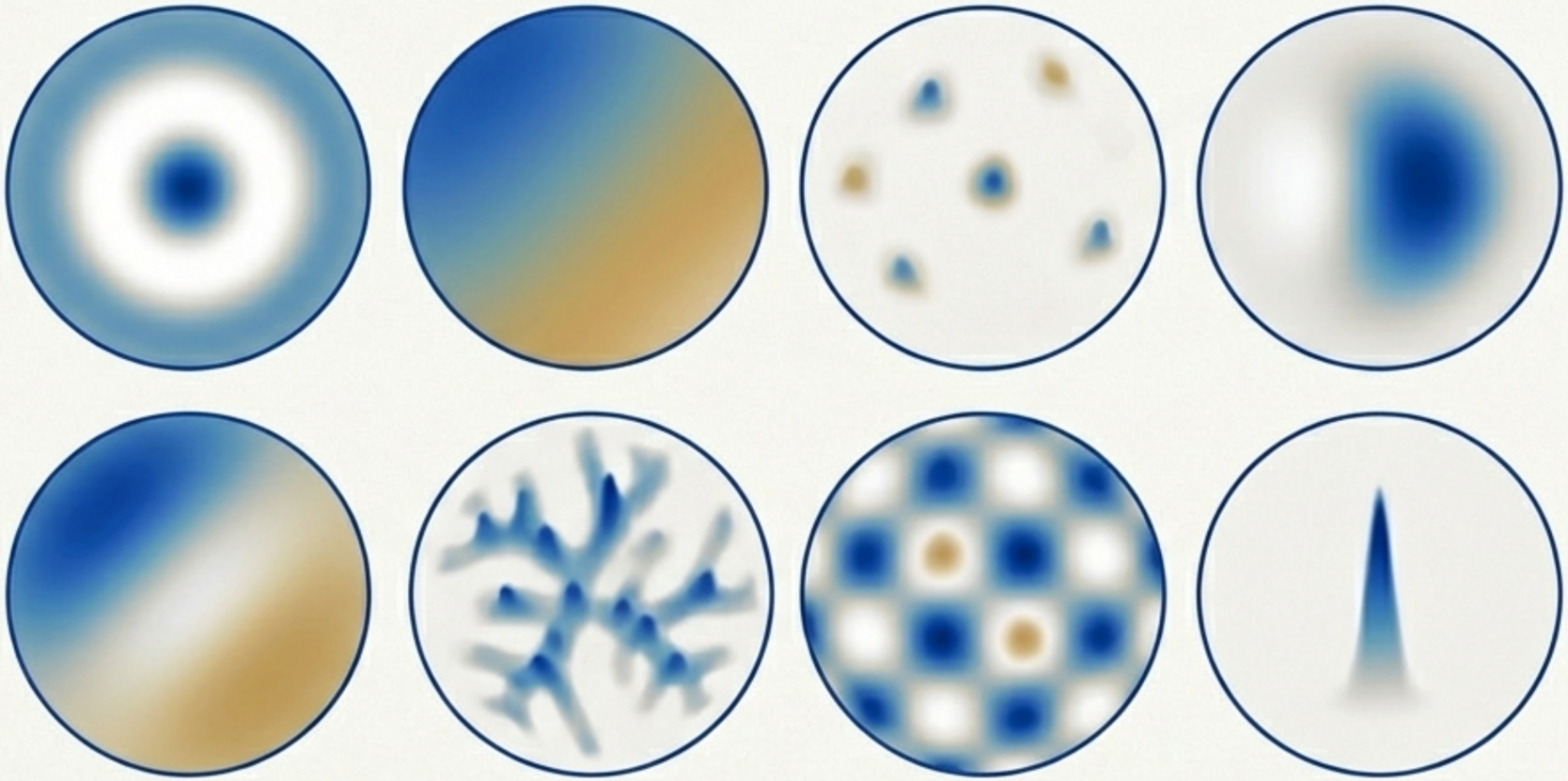
DeepSphere Achieves State-of-the-Art Performance



DeepSphere consistently outperforms traditional methods, especially when data is noisy or incomplete—just like in the real world.

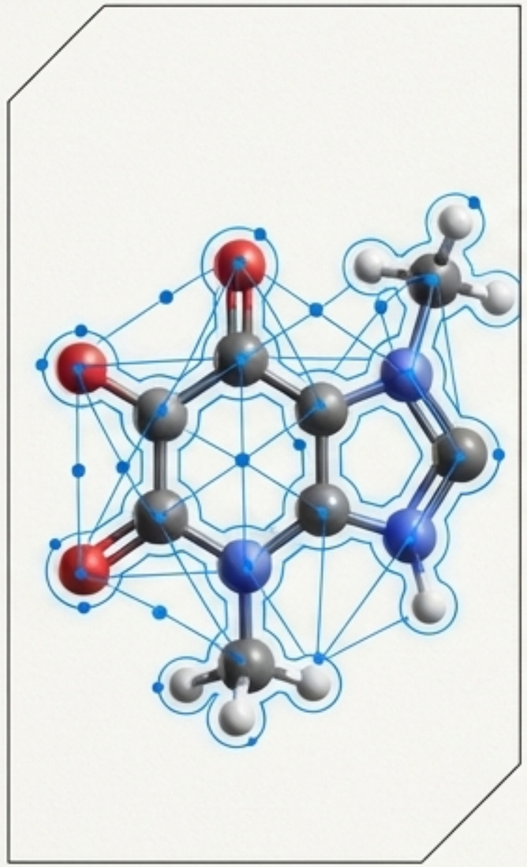
Peeking Inside the Mind of the Network

By visualizing the learned filters, we can see what the network has learned to look for. It autonomously discovers meaningful structures, like peaks and voids, directly on the sphere.

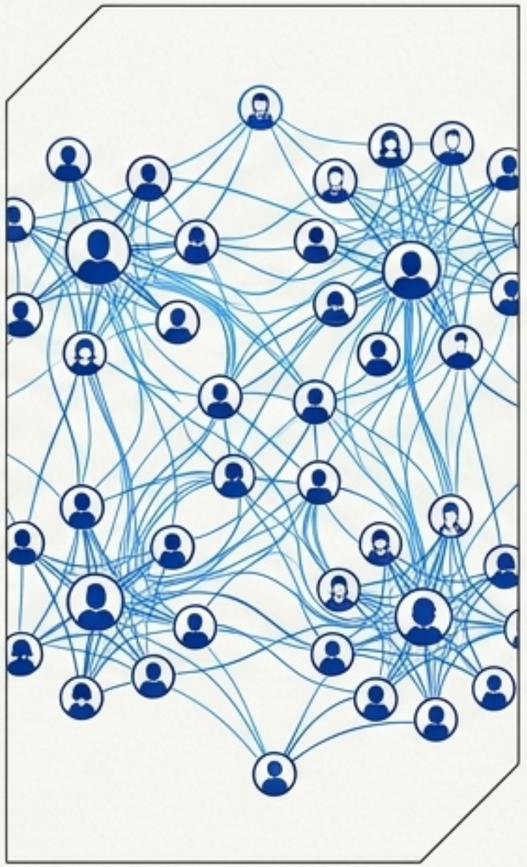


The Graph Revolution Extends Beyond the Sphere

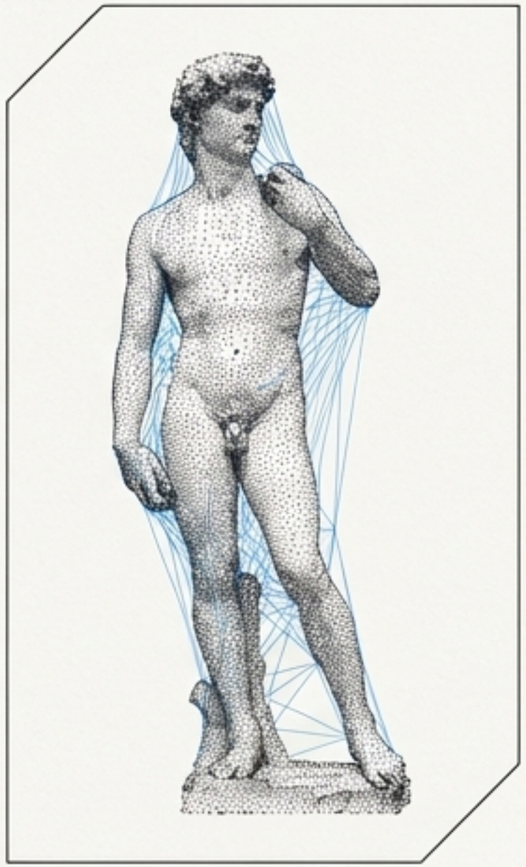
The core idea—modeling data as a graph and learning from its structure—is a cornerstone of Geometric AI. This approach is unlocking new insights in countless domains with complex, non-Euclidean data.



Molecules



Social Networks



3D Point Clouds

Get Started with DeepSphere

Repositories

PyTorch

github.com/deepsphere/deepsphere-pytorch

Modern reimplementation ideal for research and rapid prototyping.

TensorFlow

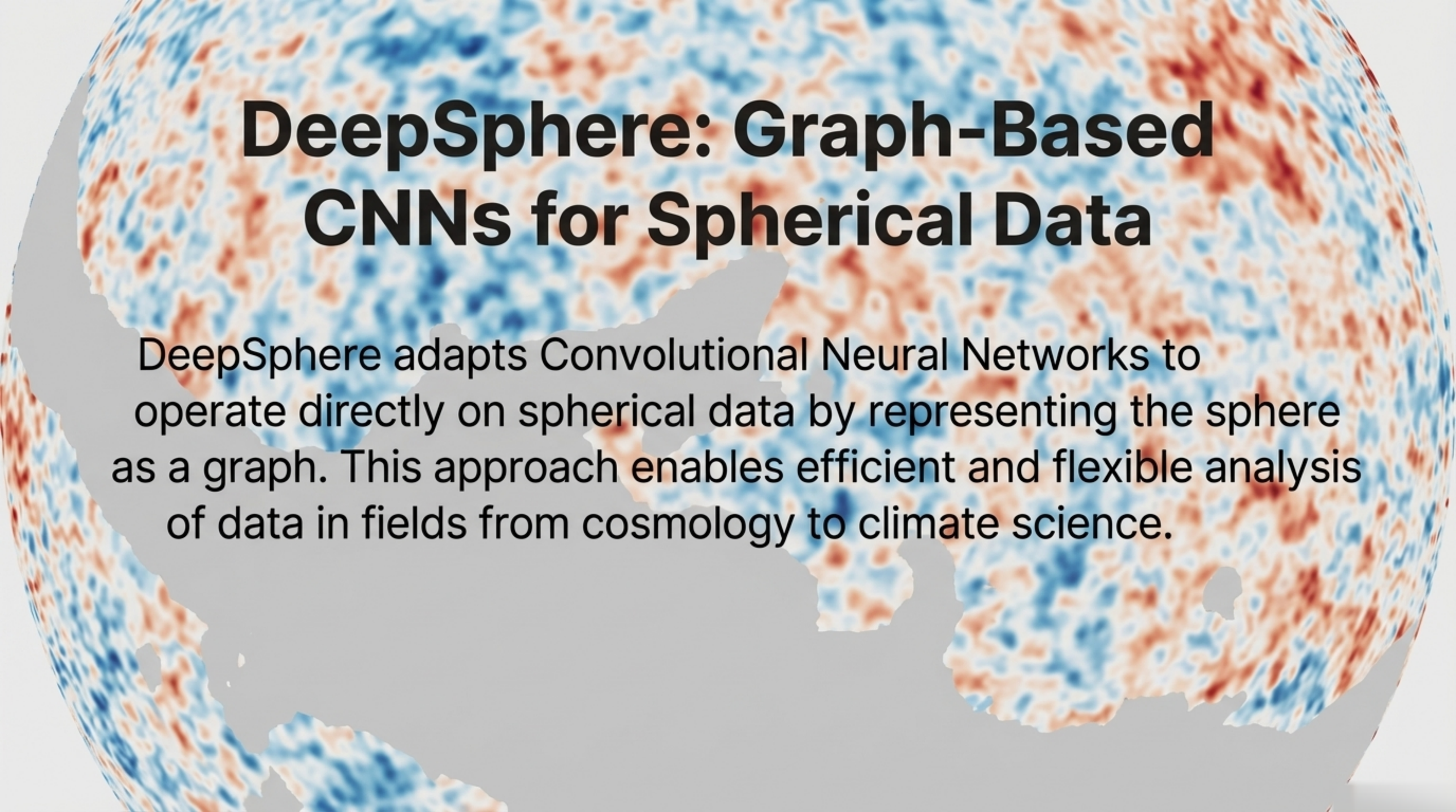
github.com/deepsphere/deepsphere-cosmo-tf2

Mature TFv2 implementation for production environments and generative models.

Key Papers

Defferrard, M., Milani, M., Gusset, F., & Perraudin, N. (2020). DeepSphere: a graph-based spherical CNN. *International Conference on Learning Representations (ICLR)*.

Perraudin, N., Defferrard, M., Kacprzak, T., & Sgier, R. (2019). DeepSphere: Efficient spherical CNN with HEALPix sampling for cosmological applications. *Astronomy and Computing*.

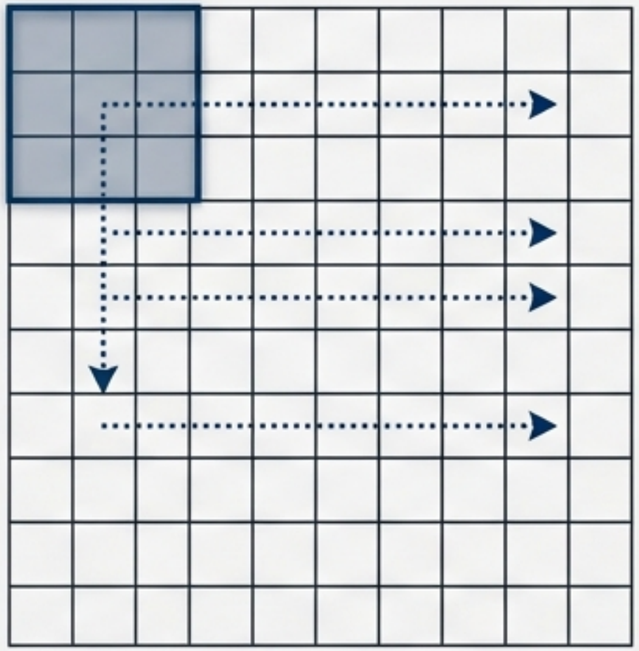


DeepSphere: Graph-Based CNNs for Spherical Data

DeepSphere adapts Convolutional Neural Networks to operate directly on spherical data by representing the sphere as a graph. This approach enables efficient and flexible analysis of data in fields from cosmology to climate science.

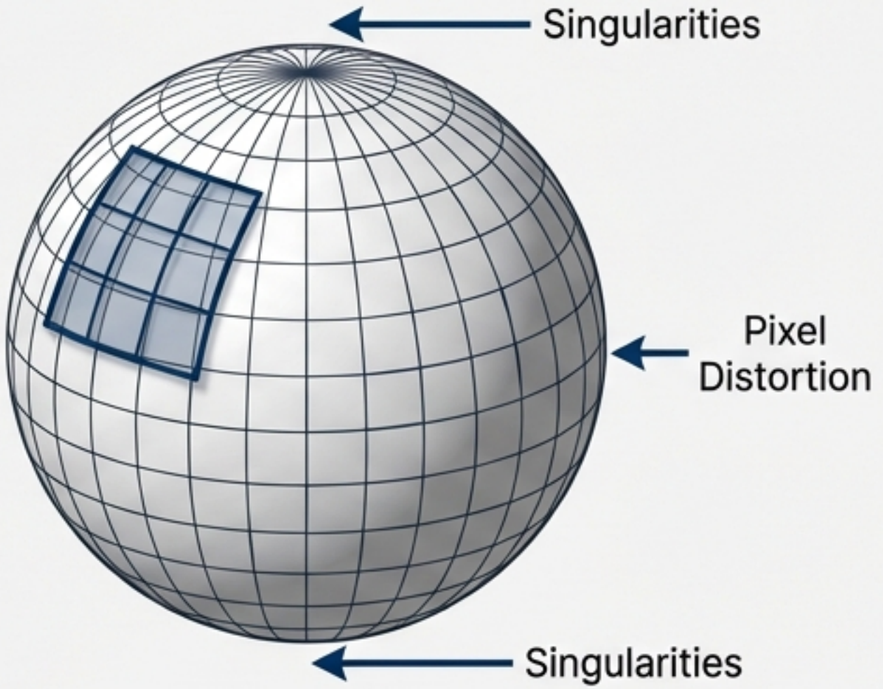
The Challenge: Applying Standard Convolutions to a Sphere

Standard Convolutions



Standard CNNs rely on a regular, Euclidean grid. The convolution operation is uniform and translation-equivariant.

Challenge on a Sphere

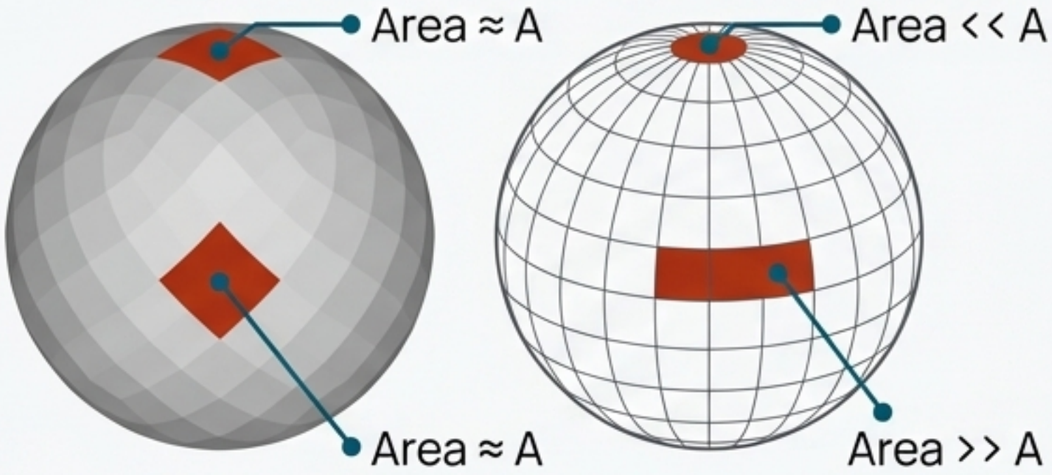


Spheres lack a uniform grid. Projections introduce distortions, and the concept of a sliding window is not well-defined. This breaks the property of rotation equivariance, which is critical for many scientific applications.

A new approach is needed to define a meaningful and efficient convolution operation directly on the sphere.

HEALPix provides an efficient, equal-area, and hierarchical grid for sampling the sphere.

Equal Area



Every pixel represents the same surface area, preventing sampling bias that can occur with other projection methods.

Hierarchical

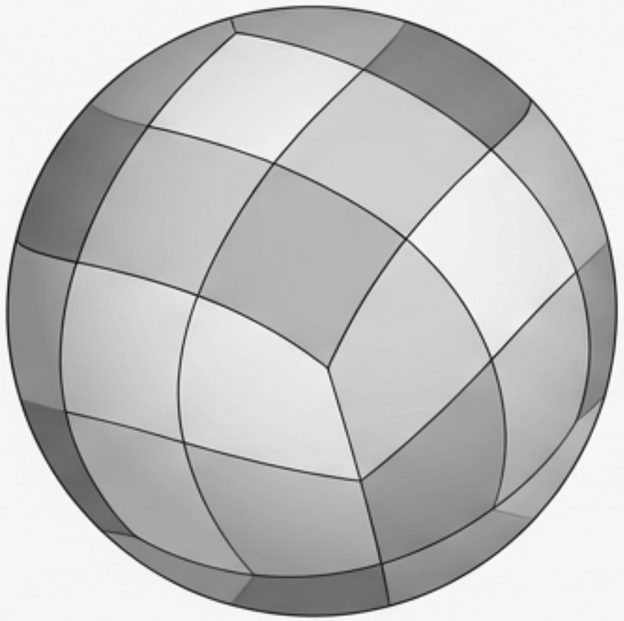


Pixels can be recursively subdivided, enabling natural multi-scale analysis and pooling operations within the neural network.

Key Takeaway: HEALPix is the ideal starting point for building a faithful and efficient graph representation of the sphere.

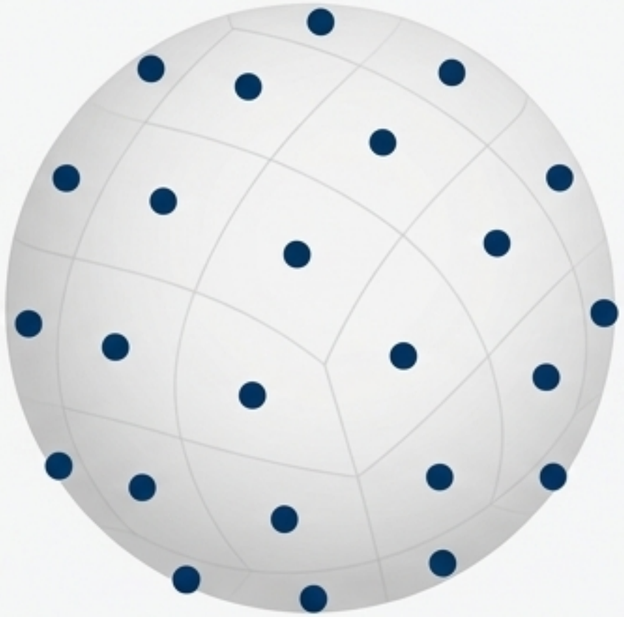
The Solution: Representing the Sampled Sphere as a Graph

Step 1



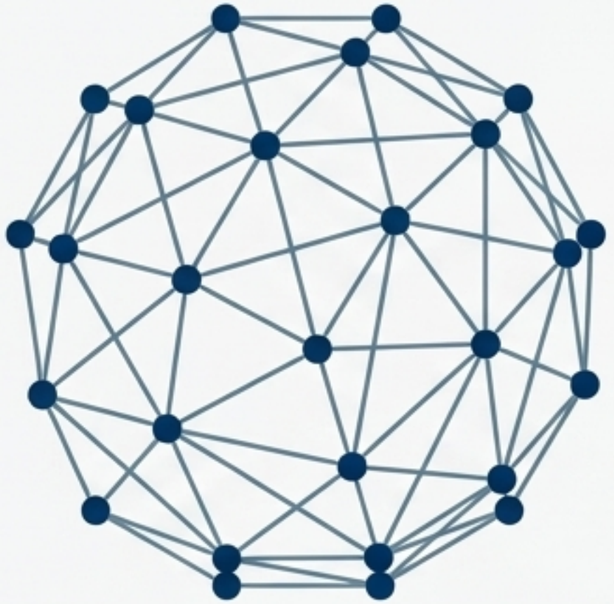
1. We begin with data sampled at discrete points on the sphere.

Step 2



2. Each sample point becomes a vertex in a graph.

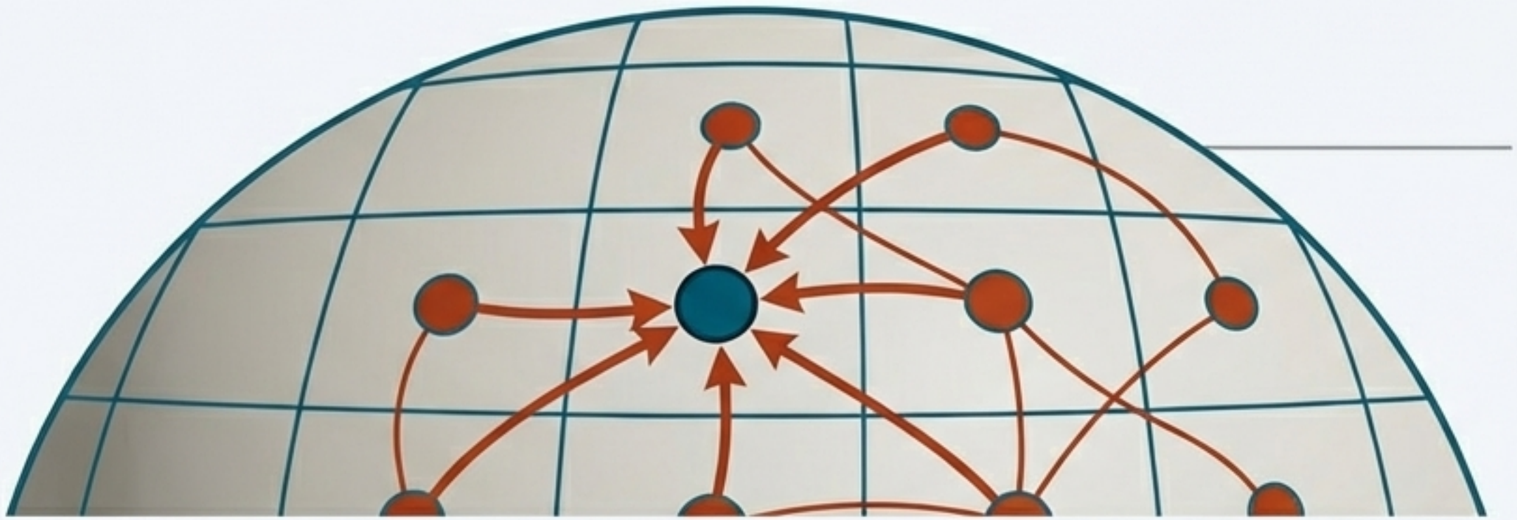
Step 3



3. Edges connect vertices based on proximity, approximating the geodesic distance. The data at each point becomes a signal on the graph.

Core Idea: By modeling the sphere as a graph, we can leverage the powerful framework of Graph Signal Processing to define operations like convolution.

DeepSphere performs convolutions efficiently in the vertex domain using the graph Laplacian.

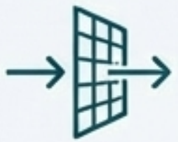


Instead of sliding a 2D filter, we aggregate information from a node's neighbors on the graph. This is computationally fast (O(n)) and localized, similar to standard CNNs.

Mechanism



The Graph Laplacian (L)
Briefly described as capturing the graph's structure.



Graph Convolution
Represented as a filtering operation, with the formula $g(\mathbf{L})f$

$$g(\mathbf{L})f$$



Efficient Approximation
Explained as using Chebyshev polynomials to avoid costly calculations.

$$g(\mathbf{L}) \approx \sum c_m T_m(\tilde{\mathbf{L}})$$

The Mathematical Engine: The Graph Laplacian

Weighted Adjacency Matrix: $A = (w_{ij})$, where w_{ij} is the similarity between vertices x_i and x_j.

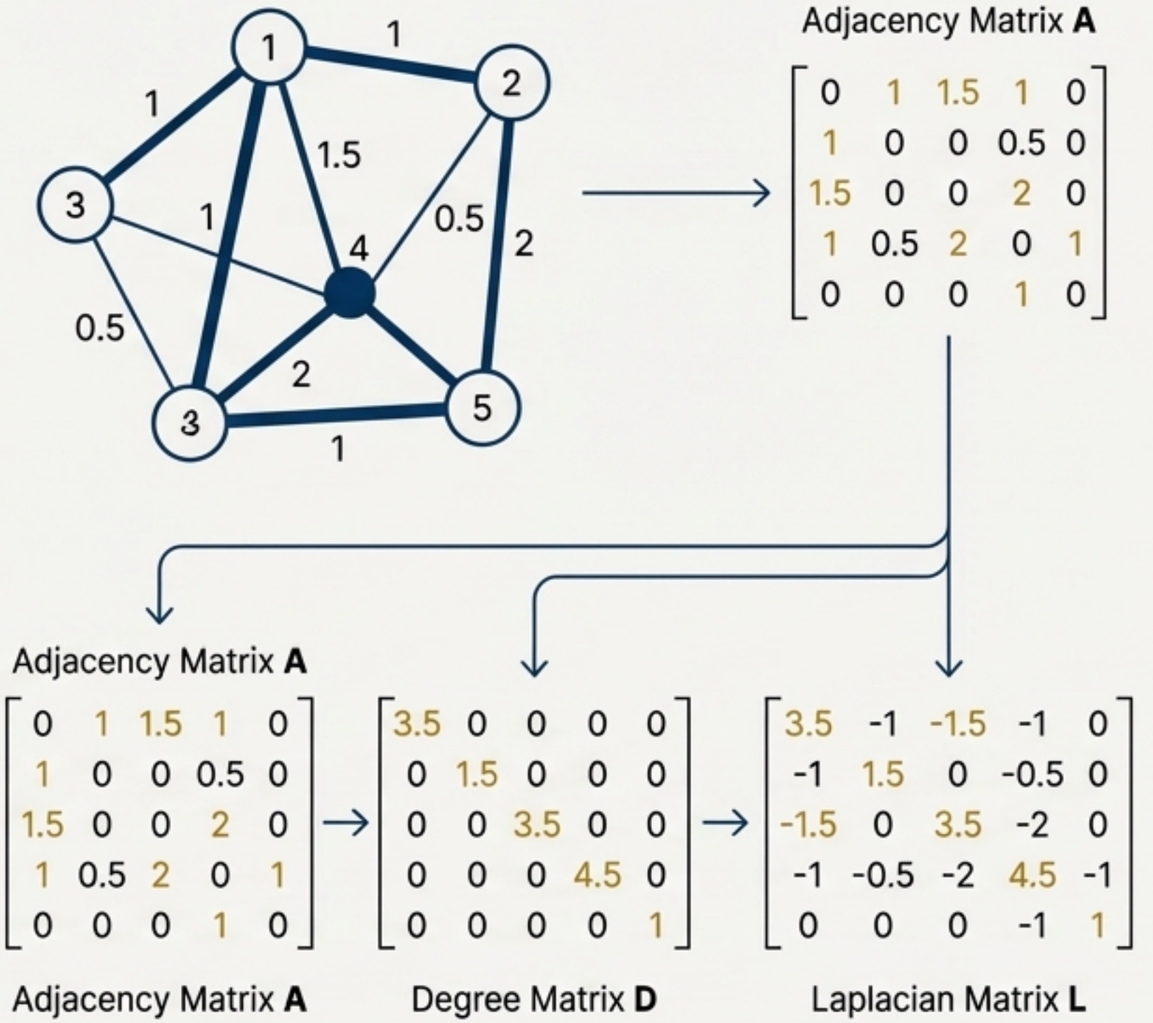
Degree Matrix: D, a diagonal matrix where $D_{ii} = \sum_j w_{ij}$.

Graph Laplacian: $L = D - A$

The Laplacian is a fundamental operator in graph theory that measures the 'smoothness' of a signal on the graph.

It's the discrete analogue of the Laplace-Beltrami operator on a continuous manifold. Its eigenvectors form a basis (the Graph Fourier basis) that can be used for spectral analysis.

The structure of **L** encodes the local connectivity of the sphere, which is essential for defining a local convolution.



Defining Convolution on a Graph

The Spectral Definition

In Graph Signal Processing, convolution with a filter g is defined as a multiplication in the graph Fourier domain.

Graph Fourier Transform of signal f: $\hat{f} = U^T f$

Filtering in Fourier domain: $(g * \hat{f}) = g(\Lambda)\hat{f}$

Inverse Transform: $g * f = U\, g(\Lambda)\, U^T f$

This can be expressed as applying a filter function g to the Laplacian L: $y = g(L)f$

The Computational Problem

This definition requires the eigendecomposition of the Laplacian ($\mathrm{L} = \mathrm{U}\Lambda\mathrm{U}^T$), which is computationally prohibitive for large graphs (e.g., millions of pixels).

For n pixels, this costs $O(n^3)$

Key Insight: To make graph convolutions practical, we need a way to compute $g(\mathrm{L})f$ without performing a full eigendecomposition.

The Key to Efficiency: Chebyshev Polynomial Approximation

Instead of diagonalizing L, the filter $g(L)$ is approximated by a truncated expansion of Chebyshev polynomials T_p of order P.

$$g(L)f \approx \left(\sum_{p=0}^{P} \alpha_p T_p(\tilde{L}) \right) f$$

where $\tilde{L}$ is the scaled Laplacian with eigenvalues in $[-1, 1]$.

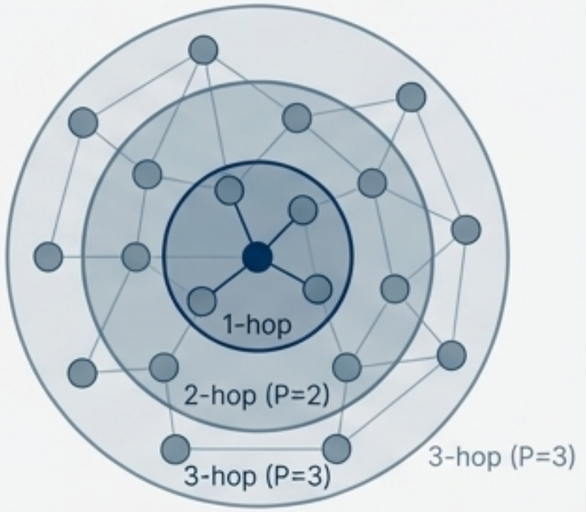
- The coefficients α_p are the learnable parameters of the filter.

Why this is efficient:

Chebyshev polynomials have a recursive property that allows for efficient computation, requiring only matrix-vector multiplications:

$$T_p(\tilde{L})f = 2\tilde{L}T_{p-1}(\tilde{L})f - T_{p-2}(\tilde{L})f$$

This avoids explicit computation of **U** and **Λ**. The cost is $O(P * |E|) \approx O(P * k * n)$, which is linear in the number of pixels **n** for a sparse graph.



Summary: This turns an intractable problem into a series of efficient, localized operations. The polynomial order **P** corresponds to the filter's receptive field size (how many 'hops' away on the graph it looks).

From Math to Code: Implementing the Chebyshev Convolution

The Math

$$T_p(x) = 2xT_{p-1}(x) - T_{p-2}(x)$$

$$y = \left(\sum_{p=0}^{P} \alpha_p T_p(\tilde{L}) \right) f$$

TensorFlow 2 / Keras Implementation

```python
# A simplified Keras Layer 'call' method
def call(self, inputs):
    # inputs is the signal x on the graph
    # self.laplacian is the sparse L̃ matrix

    # Chebyshev recursion
    chebyshev_polynomials = [inputs]  # T_0(L)x = x
    # T_1(L)x = L * x
    poly_1 = tf.sparse.sparse_dense_matmul(self.laplacian, inputs)
    chebyshev_polynomials.append(poly_1)

    for p in range(2, self.poly_order + 1):
        # T_p(L)x = 2*L*T_{p-1}(L)x - T_{p-2}(L)x
        poly_p = 2 * tf.sparse.sparse_dense_matmul(self.laplacian,
            chebyshev_polynomials[-1]) - chebyshev_polynomials[-2]
        chebyshev_polynomials.append(poly_p)

    # Stack polynomials and apply learnable weights
    stacked_polynomials = tf.stack(chebyshev_polynomials, axis=1)

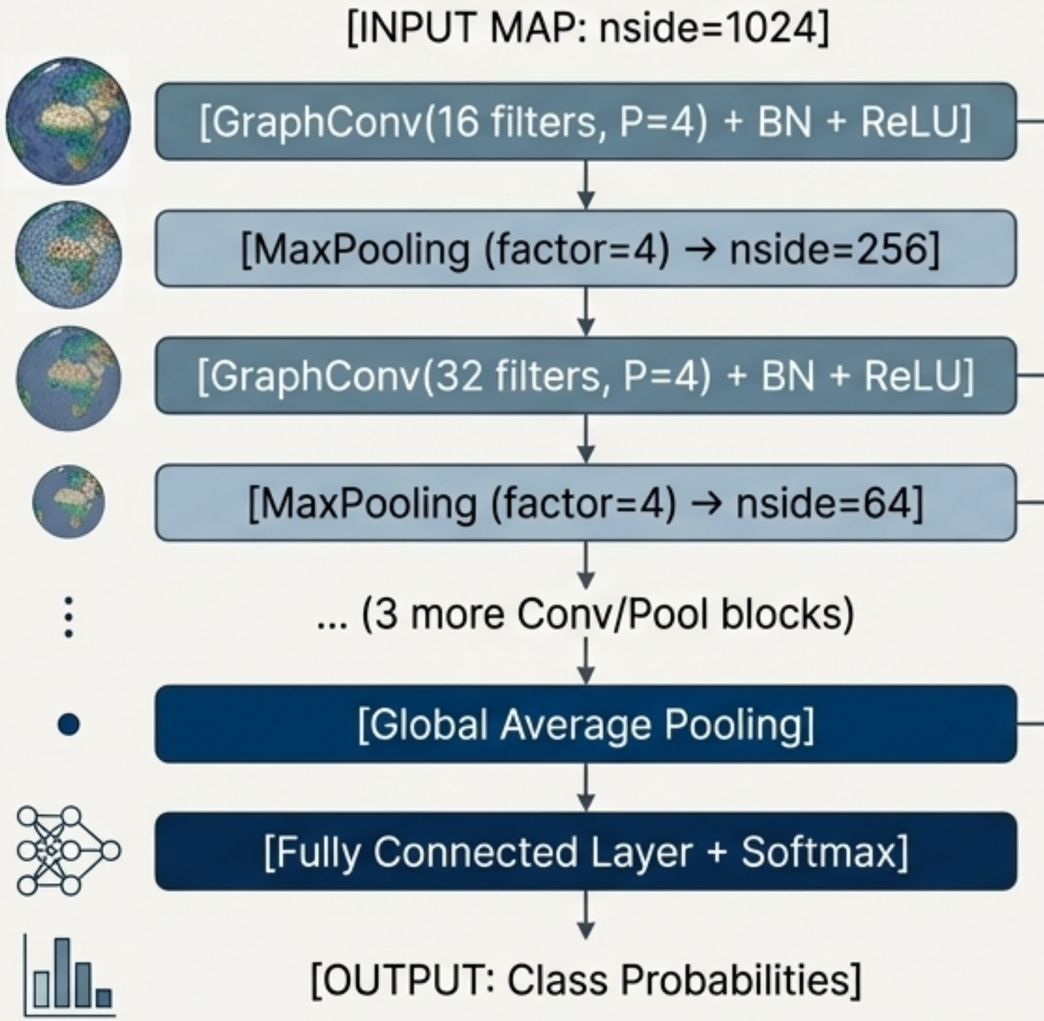
    # self.alpha are the learnable weights (α_p)
    # Output shape: (batch, N, F_out)
    output = tf.einsum('bpon, pof->bnf', stacked_polynomials, self.alpha)

    return output
```

The core of the DeepSphere convolution layer is a loop that implements the Chebyshev recursion using sparse matrix multiplication, followed by a weighted sum with learnable coefficients.

Assembling the Network: A DeepSphere CNN Architecture

Graph convolution and pooling layers are stacked to create a hierarchical model that learns features at multiple scales, analogous to a standard CNN.



Convolution: `deepsphere.layers.Chebyshev` layers perform feature extraction.

Pooling: Custom pooling layers reduce the graph size by leveraging the hierarchical nature of the HEALPix sampling.

Global Average Pooling (GAP): For rotation-invariant classification tasks, GAP summarizes features across the entire sphere before the final classification layer.

The Training Process in TensorFlow 2

Training a DeepSphere model follows a standard deep learning workflow. A custom training loop uses `tf.GradientTape` to compute gradients and update the model's weights (the Chebyshev coefficients).

```
# model is a tf.keras.Model instance
# optimizer is a tf.keras.optimizers instance
# loss_fn computes the loss (e.g., CrossEntropy)

for x_batch, y_batch in dataset:
    with tf.GradientTape() as tape:
        # ① Forward pass
①       y_pred = model(x_batch, training=True)

        # ② Compute loss
②       loss = loss_fn(y_batch, y_pred)

    # ③ Compute gradients of loss w.r.t. model variables
③   grads = tape.gradient(loss, model.trainable_variables)

    # ④ Apply gradients to update weights
④   optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

① **Forward Pass**: The model processes the input data to make a prediction.

② **Compute Loss**: The prediction is compared to the true label to calculate the error.

③ **Compute Gradients**: `GradientTape` calculates how much each model parameter contributed to the error.

④ **Update Weights**: The optimizer adjusts the parameters to reduce the error on the next pass.

The complexity of the spherical geometry is abstracted away by the custom layers. From the perspective of the training loop, it's a standard TensorFlow model.

DeepSphere is implemented in both TensorFlow and PyTorch, offering flexibility for research and development.



TensorFlow

The original implementation was in TFv1, later modernized to TFv2. It leverages a mature ecosystem ideal for production and diverse deployment needs.

`deepsphere/deepsphere-cosmo-tf2`

PyTorch

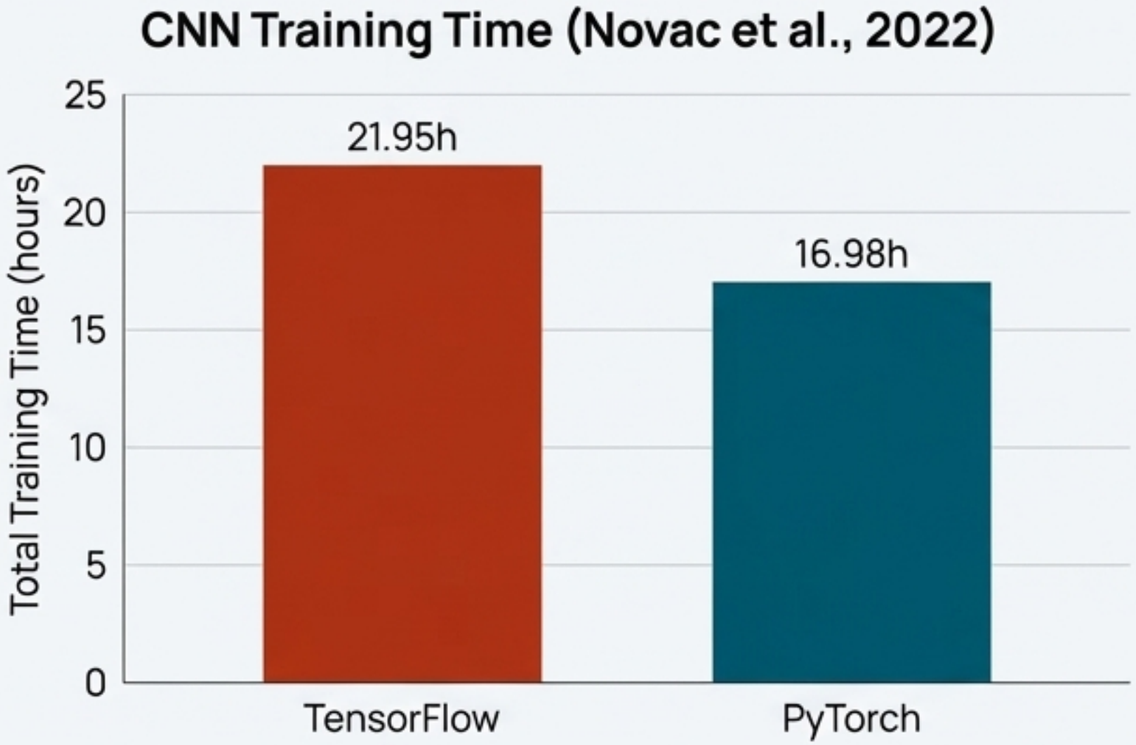
A modern reimplementation aligning with PyTorch's widespread adoption in the research community for its flexibility and intuitive, Pythonic style.

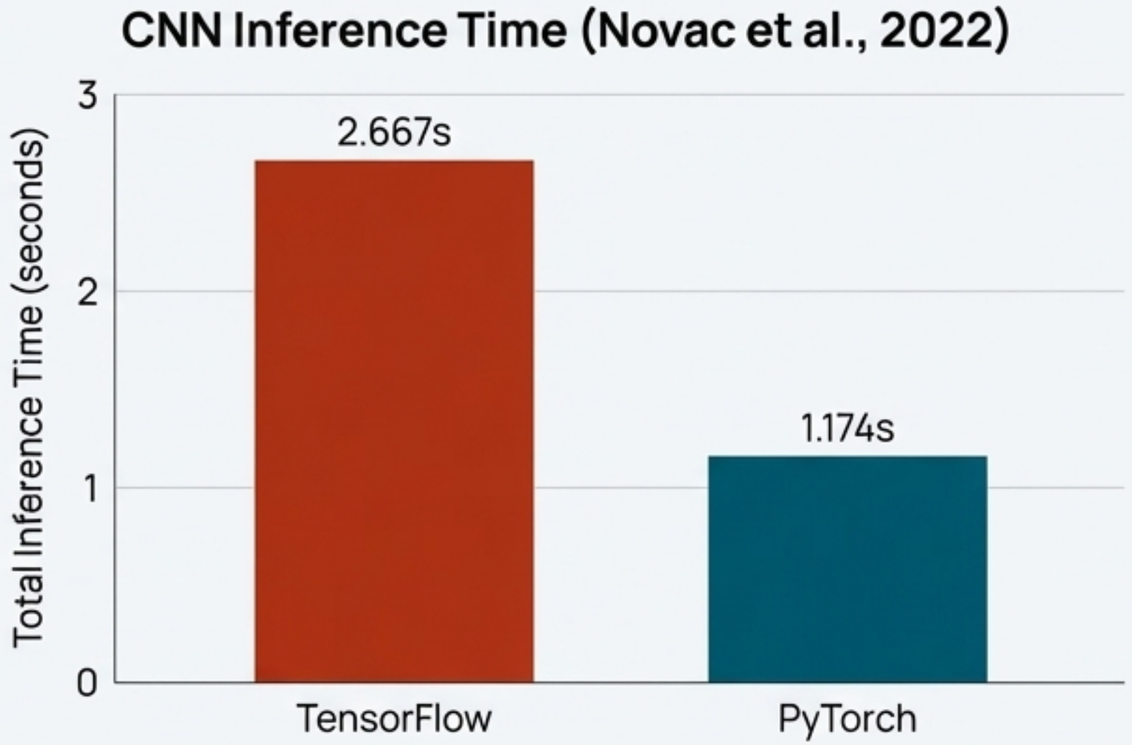
`deepsphere/deepsphere-pytorch`

The choice of framework depends on the project's context, from research prototyping to production deployment.

Feature	PyTorch	TensorFlow
Paradigm	Dynamic, "define-by-run". Feels like native Python, allowing for flexible control flow.	Eager execution by default, with powerful `@tf.function` for optional graph compilation and optimization.
Developer Exp.	Favored for rapid prototyping and research due to intuitive debugging and flexibility.	Integrated Keras API simplifies standard workflows. Strong tooling like TensorBoard and TFX for end-to-end pipelines.
Deployment	Mature options via TorchScript and ONNX export. TorchServe available for model serving.	Comprehensive ecosystem with TF Serving, TF Lite (mobile/edge), and TF.js (web). Generally considered stronger for diverse production targets.

Benchmarks show PyTorch often has lower overhead for training and inference, while TensorFlow's graph optimizations shine at scale.



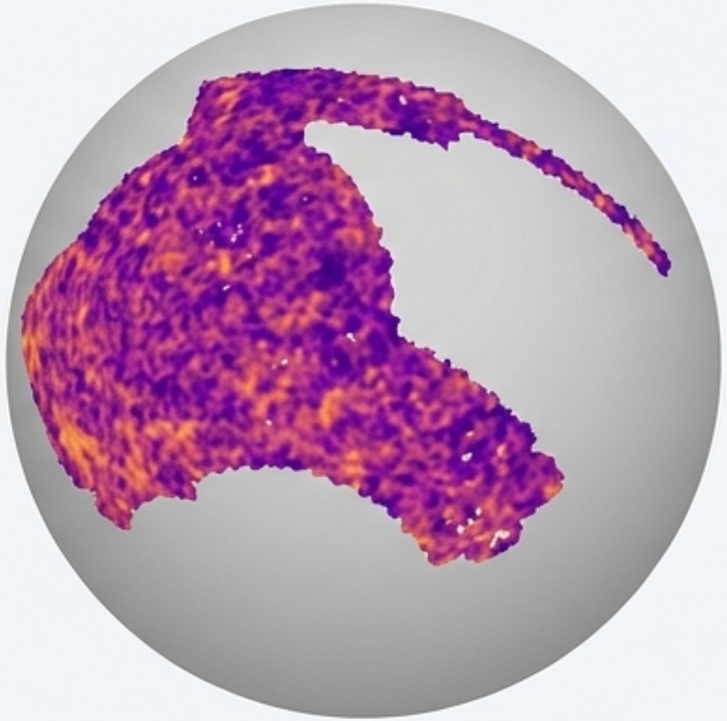


Important Caveat: Performance is highly task-dependent. DeepSphere's core graph operations are highly optimized in both frameworks, often leveraging the same underlying CUDA libraries. These general benchmarks indicate trends in framework overhead.

Application 1: Classifying cosmological maps with higher accuracy by improving equivariance.

The Task

Classifying partial weak lensing mass maps (which are noisy and high-frequency) into different cosmological models.



Performance Boost

Fewer Neighbors (k=8)	→	**87.1% Accuracy**
More Neighbors (k=40)	→	**92.5% Accuracy**

Insight

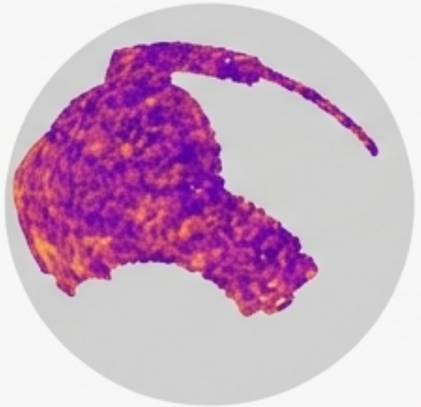
For tasks where orientation-agnostic features are key, better equivariance directly translates to better performance.

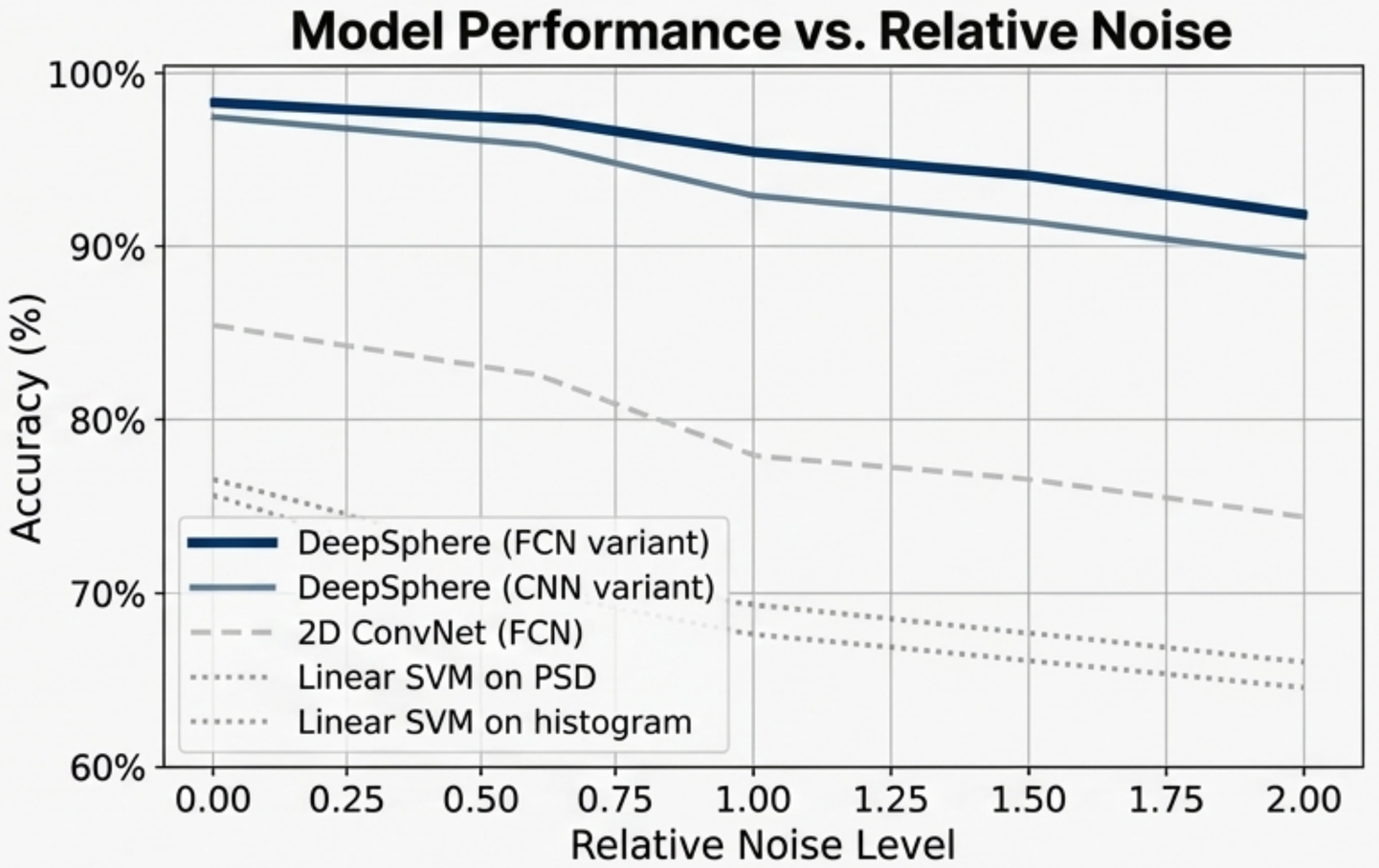
Application: Classifying Cosmological Models

Task & Data

Task
Classifying partial convergence maps from two different cosmological models ('ΛCDM'). The task is challenging due to high noise levels.

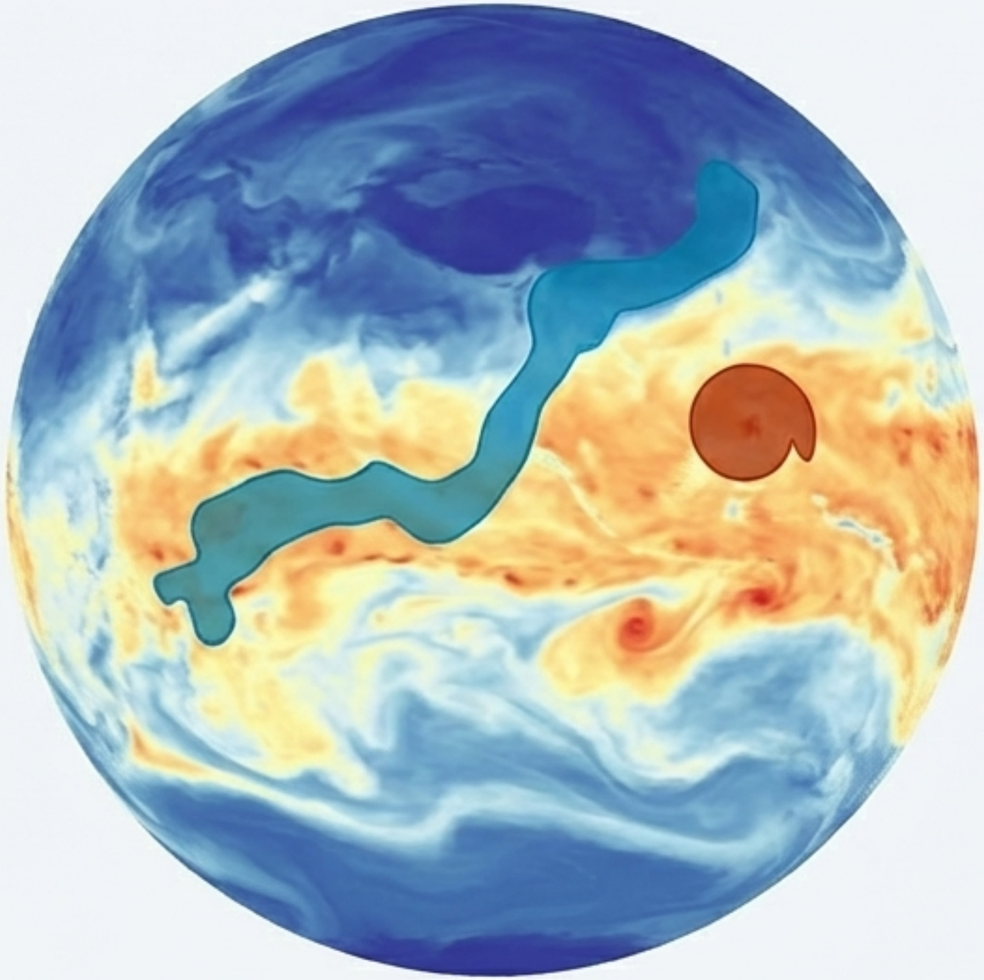
Data
The maps are partial spheres with ~10^6 pixels each, highlighting the model's scalability and flexibility in handling non-full-sphere data.





Key Finding: The FCN variant outperforms the CNN variant as it exploits the rotational symmetry of the task. Both variants largely beat the 2D ConvNet, whose performance suffers from the geometric distortions introduced by projection.

Application 2: Segmenting extreme climate events with state-of-the-art performance.



The Task

A complex segmentation task on 16-channel climate simulation data to identify Atmospheric Rivers and Tropical Cyclones.

Mean Average Precision (mAP)

89.16

State-of-the-Art Performance

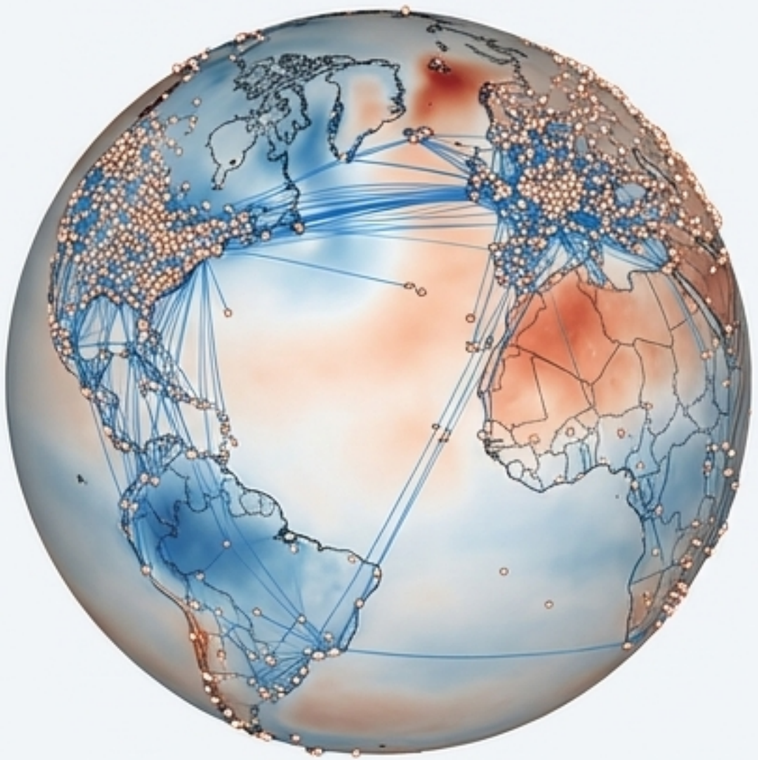
Insight

The model's isotropic filters proved highly effective, suggesting that for some complex tasks, a well-designed graph CNN can match or exceed more complex anisotropic models.

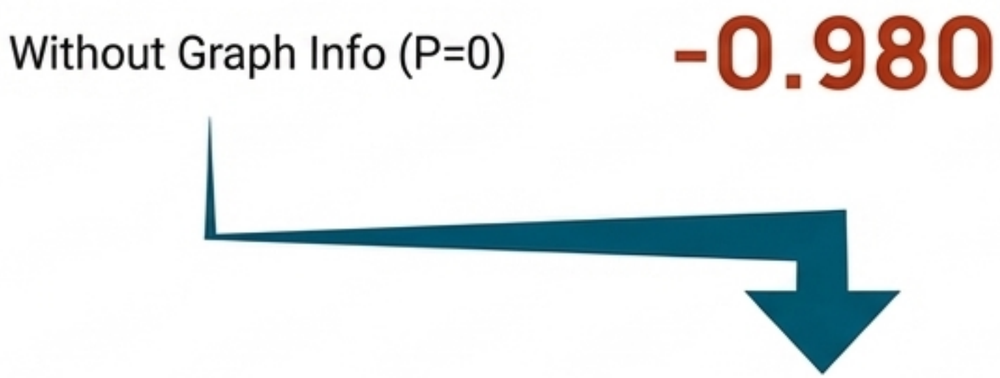
Application 3: The power of flexibility with unevenly sampled data.

The Task

Predicting temperature based on historical data from ~10,000 unevenly distributed weather stations across the globe.



Predicting Day from Precipitation (R^2 Score)



Insight

DeepSphere does not require a perfect, uniform grid. It works directly with the data's native structure, a major advantage for real-world scientific data.

The DeepSphere Advantage: Efficiency, Flexibility, and Equivaniance



Computational Efficiency

Graph convolutions scale linearly with the number of pixels (`O(N_pix)`), enabling analysis of high-resolution maps with millions of points.



Sampling Flexibility

The graph representation naturally handles non-uniform, partial, and irregular samplings—no interpolation to a fixed grid is required. This allows working directly on measurement data from weather stations, telescopes, or sensors.



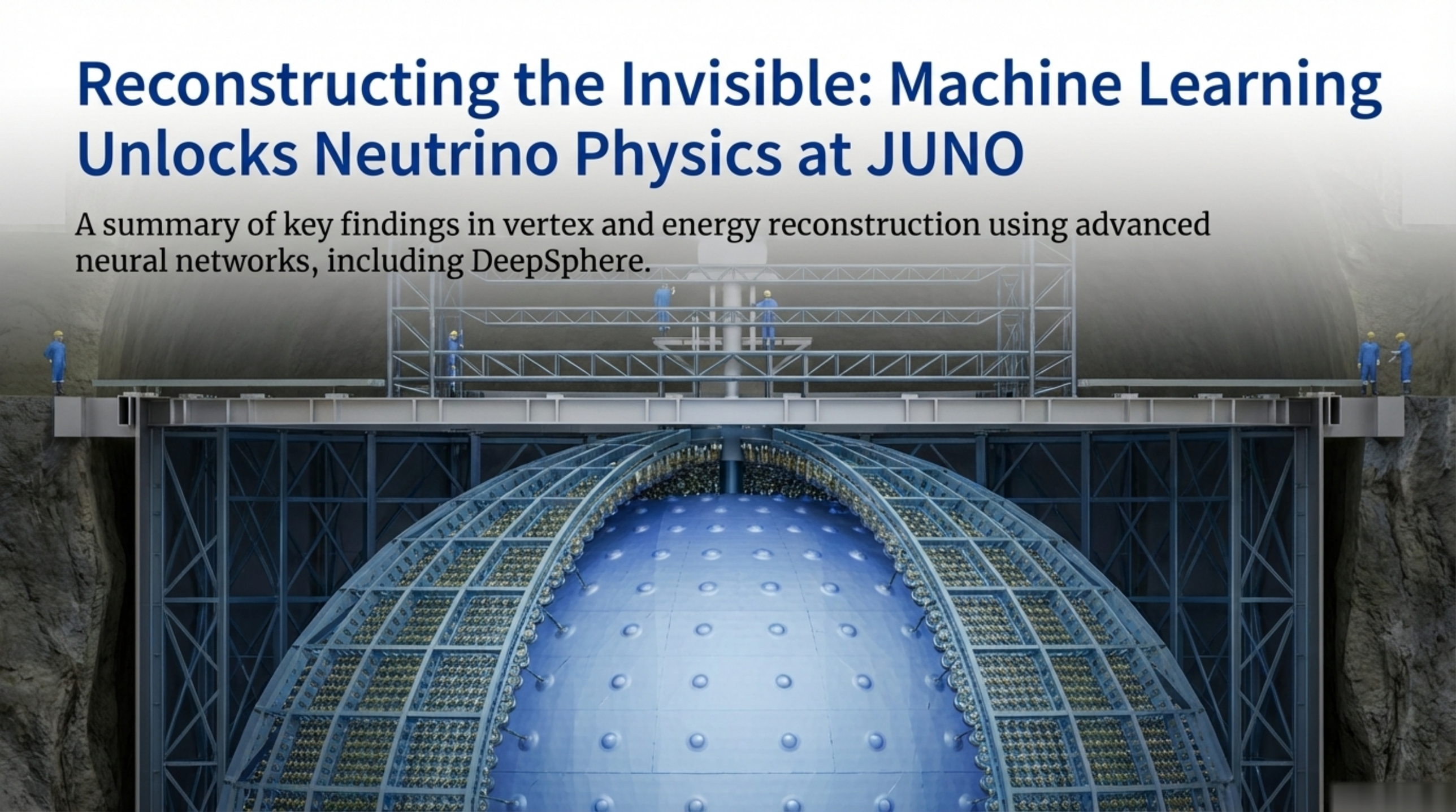
Controllable Equivariance

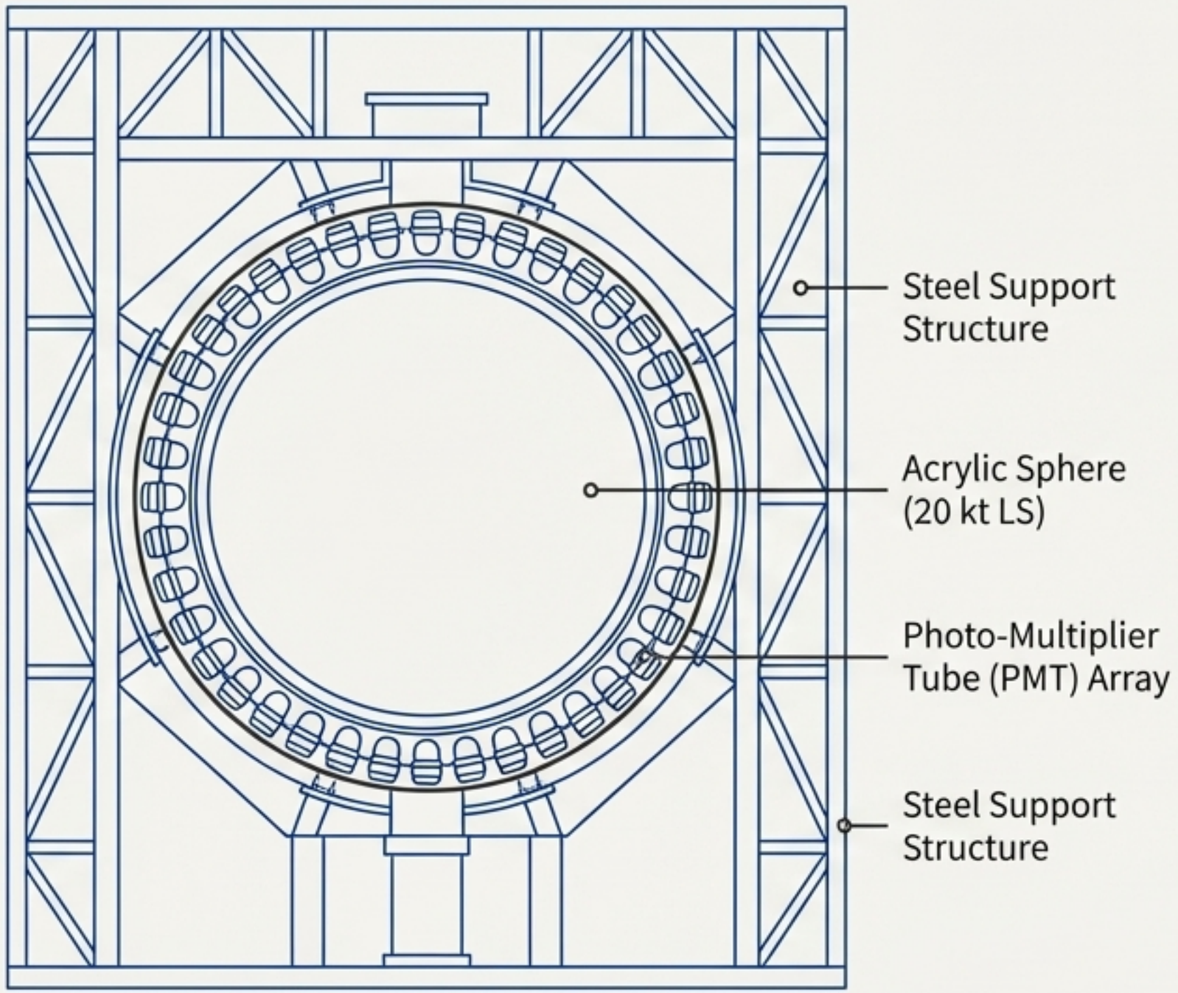
The graph Laplacian approximates the properties of the continuous sphere. By increasing By increasing the number of neighbors (`k`) in the graph construction, we can systematically improve rotation equivariance at a controlled computational cost.

DeepSphere provides a principled and performant framework for applying deep learning to spherical data, opening new possibilities for data-driven discovery in science.

Reconstructing the Invisible: Machine Learning Unlocks Neutrino Physics at JUNO

A summary of key findings in vertex and energy reconstruction using advanced neural networks, including DeepSphere.





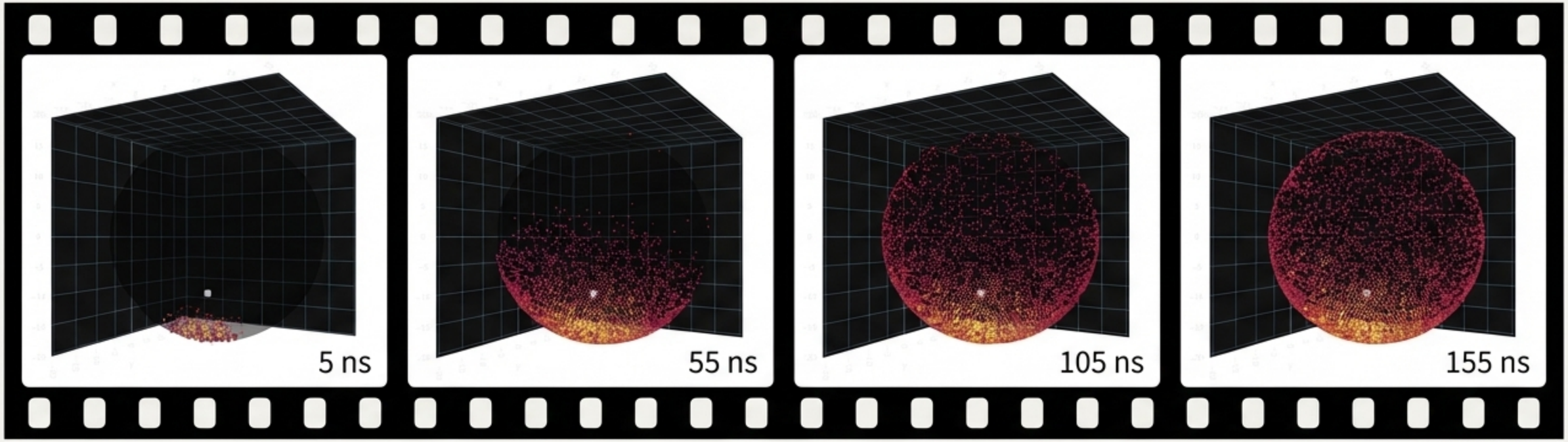
JUNO's mission is to solve one of physics' most significant open questions.

The Experiment: JUNO is a 20 kiloton liquid scintillator (LS) detector.

Primary Goal: To determine the neutrino mass ordering.

The Method: JUNO will achieve this by precisely measuring the energy spectrum of reactor antineutrinos from two nuclear power plants located 53 km away. These are detected via the inverse beta-decay (IBD) reaction.

Success demands unprecedented reconstruction accuracy



A single event unfolds as light propagates across the detector. Color represents the accumulated charge on each PMT.

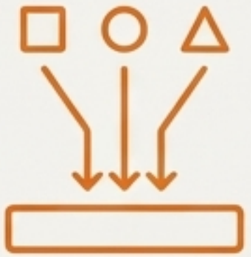
Neutrino interactions create faint light signals detected by over 17,600 large PMTs and 25,600 small PMTs. The raw data consists of the charge (number of photo-electron hits) and the first hit time on each PMT.

Vertex Resolution
< 10 cm at 1 MeV

Energy Resolution
≤ 3% at 1 MeV

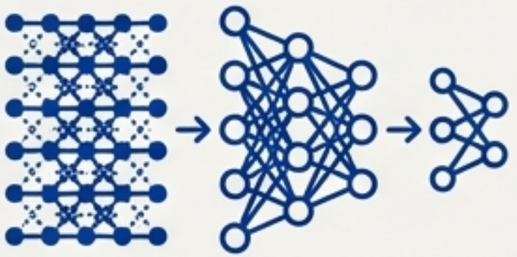
A spectrum of machine learning models was deployed to tackle the reconstruction task.

Two primary classes of models were studied, differing in the complexity of their input data.



Class 1: Simple & Fast (Aggregated Features)

- Models: Boosted Decision Trees (BDT), Deep Neural Networks (DNN).
- Input: A few high-level features like total hits, center of charge, and mean/dispersion of hit times.
- Analogy: *These models read a 'summary report' of the event.*



Class 2: Complex & Powerful (PMT-wise Data)

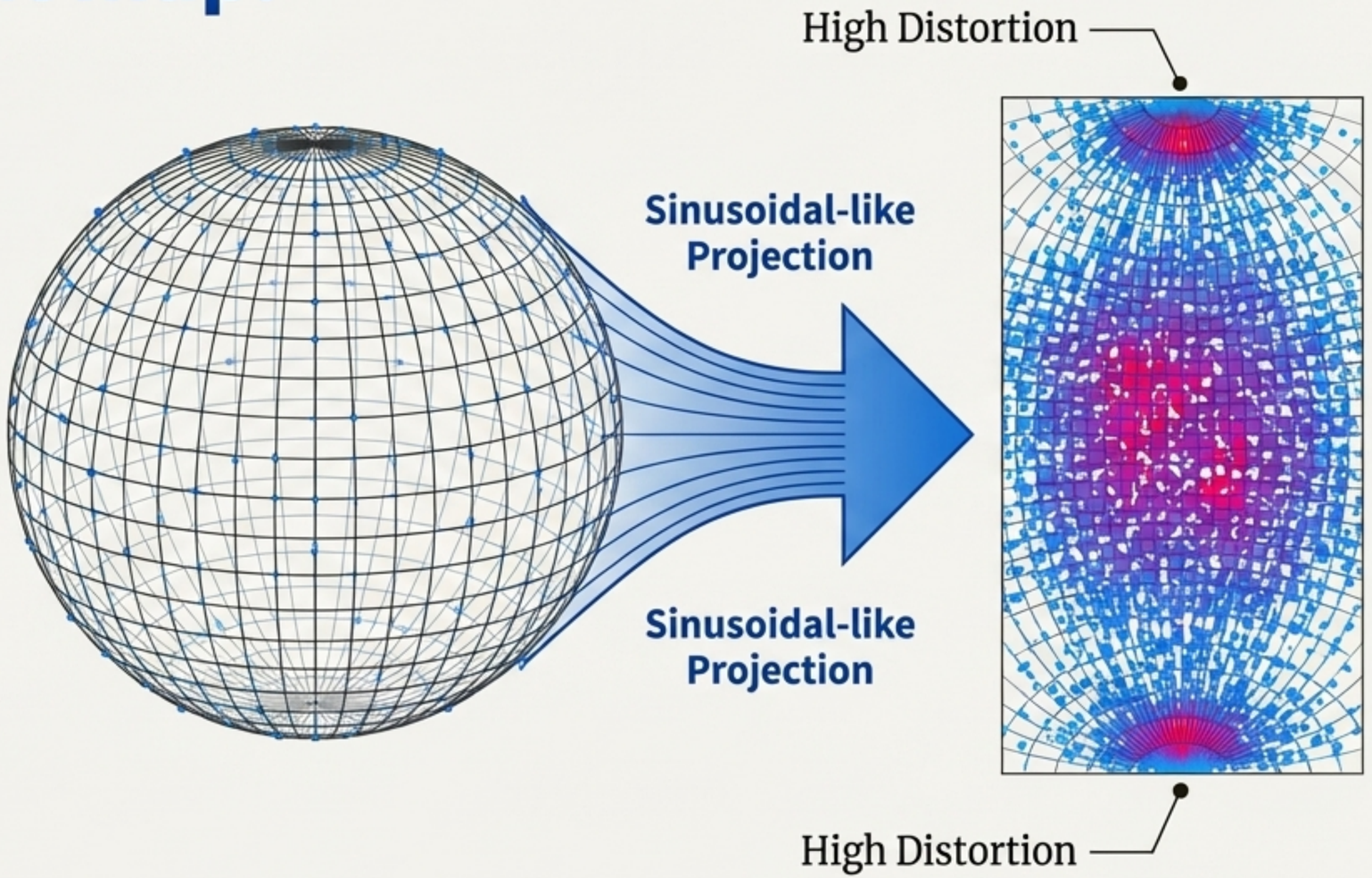
- Models: Convolutional Neural Networks (CNNs like VGG-J, ResNet-J) , Graph Neural Networks (GNN-J / DeepSphere).
- Input: The raw charge and time information from each individual PMT.
- Analogy: *These models see the 'full security footage' of the event.*

CNNs require projecting the spherical detector onto a flat map.

The Problem: PMTs are arranged on a sphere, a non-Euclidean domain. Standard CNNs operate on regular, grid-like data (images).

The Workaround: A custom sinusoidal-like projection method transforms the PMT signals into two 2D images: one for charge and one for first hit time. The final image size is 230 x 124 pixels.

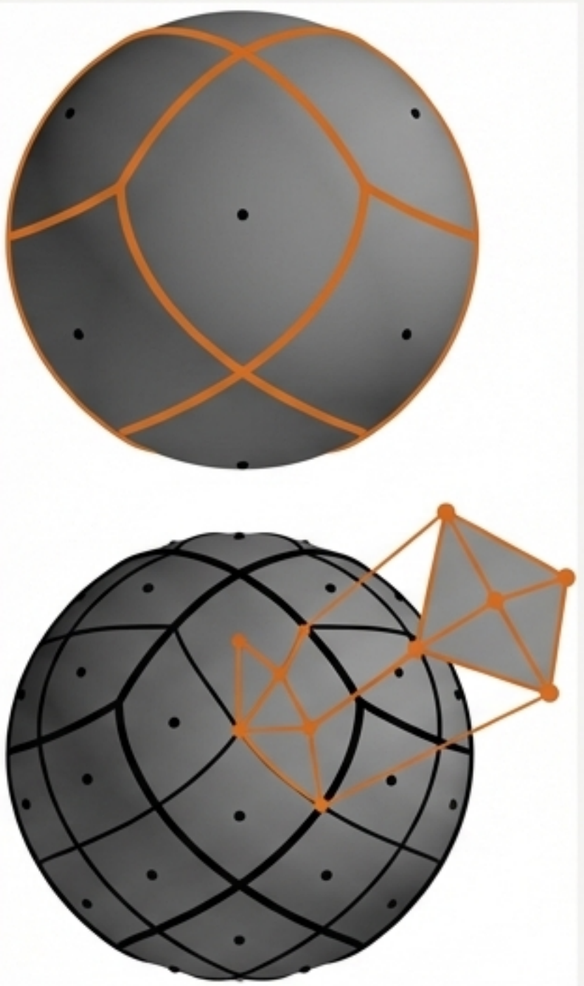
The Drawback: Any projection from a sphere to a plane inevitably introduces geometric distortion and breaks topological relationships, which can complicate the learning process for the network.



DeepSphere works directly on the detector's native spherical geometry.

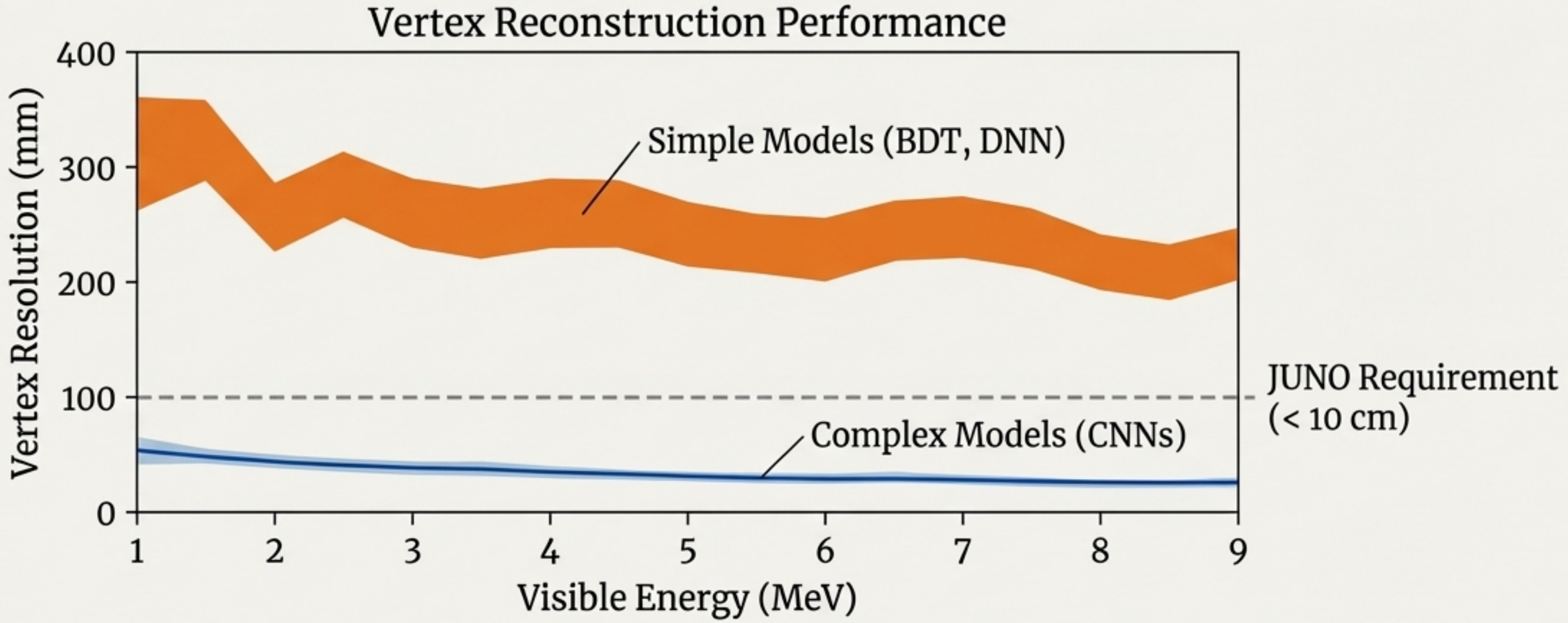
The GNN-J model is an adaptation of the DeepSphere architecture.

- **The Approach**: It treats PMTs as nodes on a graph, preserving their true 3D spatial relationships and avoiding projection-based distortion.
- **The Technology**: It uses the HEALPix algorithm to hierarchically partition the sphere's surface into 3072 equal-area pixels (at Nside=16). This structure enables efficient “spherical convolutions” and pooling operations directly on the graph.



HEALPix tessellation divides the sphere into a hierarchical graph of equal-area pixels, enabling convolutions without distortion.

For vertex position, seeing the full picture is critical.

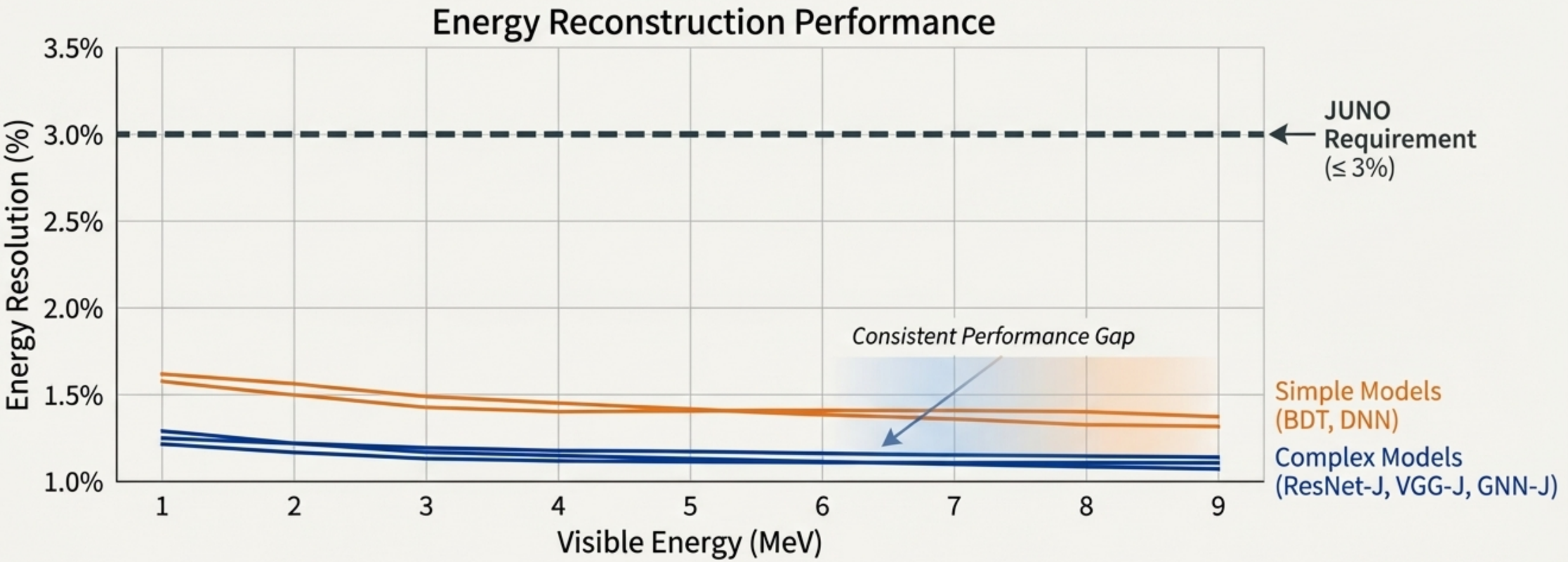


Key Finding: CNN models (VGG-J, ResNet-J) that use PMT-wise data dramatically outperform the BDT and DNN models that use aggregated features. The resolution is up to 4 times better.

The Result: Both VGG-J and ResNet-J achieve the required **< 10 cm vertex resolution** at 1 MeV visible energy.

Insight: This demonstrates that reconstructing the vertex relies heavily on the detailed spatial and timing patterns across the detector, which only the complex models can fully process.

In the race for energy precision, advanced models provide the winning edge.

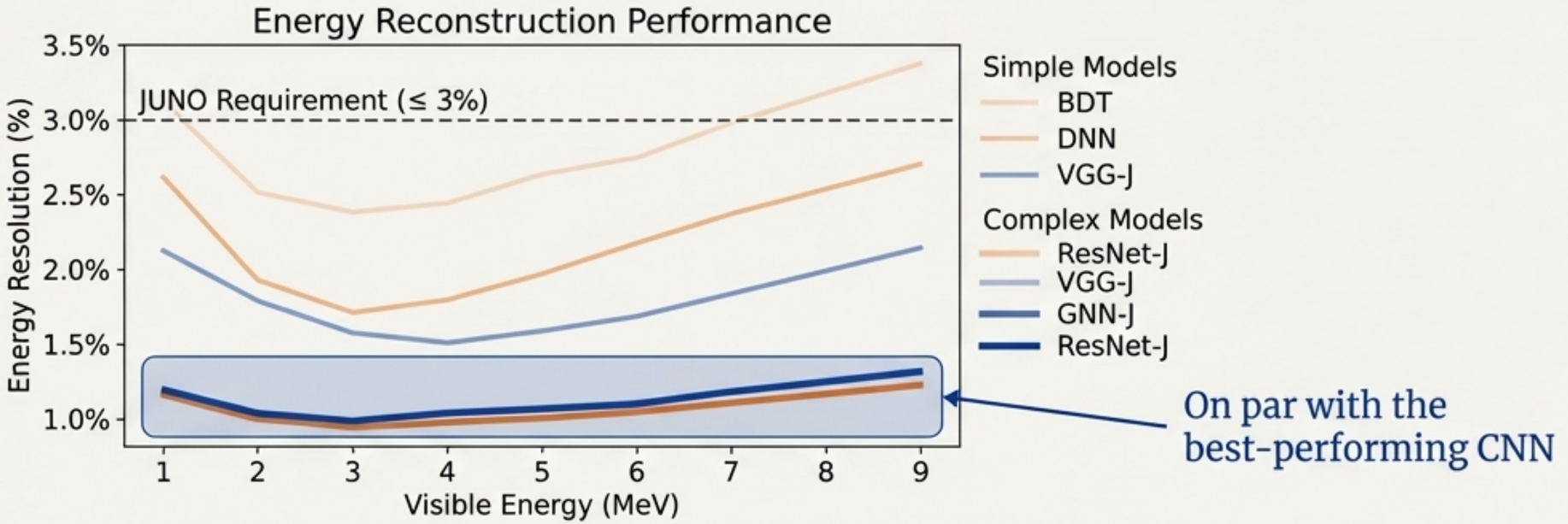


Key Finding: All studied models successfully meet the ≤ 3% energy resolution requirement at 1 MeV visible energy.

The Crucial Edge: The complex models (ResNet-J, VGG-J, and GNN-J) consistently achieve a finer resolution across all energies compared to BDT and DNN.

Insight: While the improvement is small, this enhanced precision is critical for achieving JUNO's primary physics goals.

DeepSphere delivers top-tier energy resolution without geometric compromise.



Direct Comparison: The GNN-J model's energy resolution is on par with the best-performing, and much larger, CNN model (ResNet-J).

The Advantage: It achieves this elite performance without the inherent distortions and topological issues of the 2D projection required by CNNs.

Conclusion: This validates Graph Neural Networks as a powerful, computationally efficient, and **architecturally elegant** approach for spherical detectors like JUNO.

Models were tested against realistic detector effects

The study incorporated two major sources of electronic noise into the simulation: Dark Noise (DN) and Transit Time Spread (TTS).



Transit Time Spread (TTS)
Source Sans 3 Semibold

Impacts **Timing** Information → Primarily degrades **Vertex** Resolution



Dark Noise (DN)
Source Sans 3 Semibold

Impacts **Charge** Information → Primarily degrades **Energy** Resolution

Note: All previously shown results are from the most realistic simulation, which includes both DN and TTS effects.

Machine learning successfully meets JUNO's reconstruction challenge



Mission Accomplished: All studied ML models achieve the vertex (<10 cm) and energy (≤3%) resolution required for JUNO's primary physics goals at 1 MeV.



Complexity Pays Off: Models using raw PMT data (CNNs, GNNs) consistently outperform simpler models using aggregated features, providing a critical precision edge for both vertex and energy.



A New Paradigm for Spherical Detectors: DeepSphere (GNN-J) proves to be a highly effective approach, matching state-of-the-art CNN performance with a more fundamentally sound geometric model that avoids projection compromises.

The path forward: From simulation to reality



Future Work:

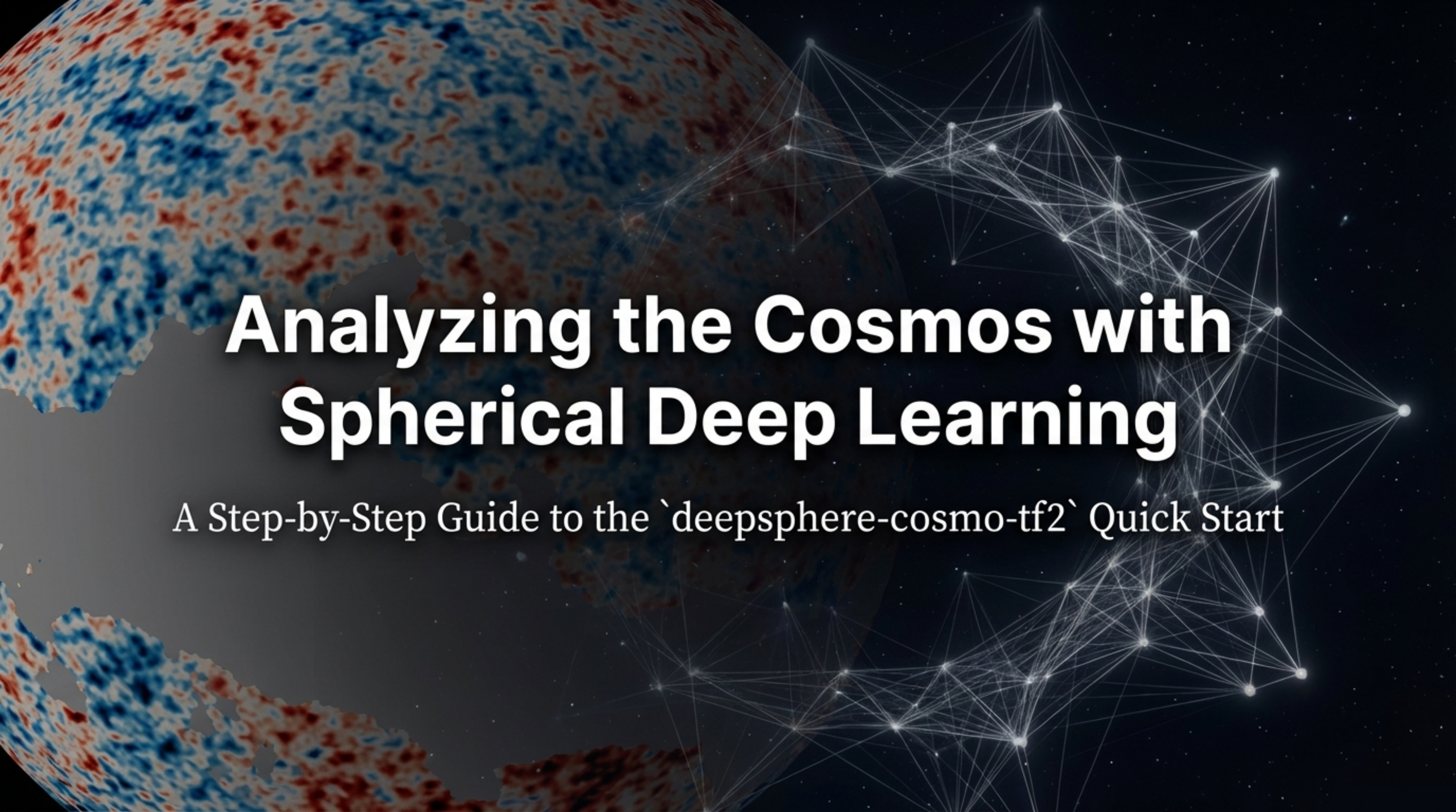
- **Fine-tuning with Real Data:** Using calibration data from the JUNO detector to refine and validate the models, potentially using domain adaptation techniques.
- **Model Optimization:** Reducing the number of parameters and improving the prediction speed of the large CNN models (ResNet-J, VGG-J).
- **Next-Generation GNN:** Developing a GNN model that uses PMT data directly without the initial HEALPix aggregation, possibly using a graph coarsening algorithm like Graclus.



Thank you.

Questions?

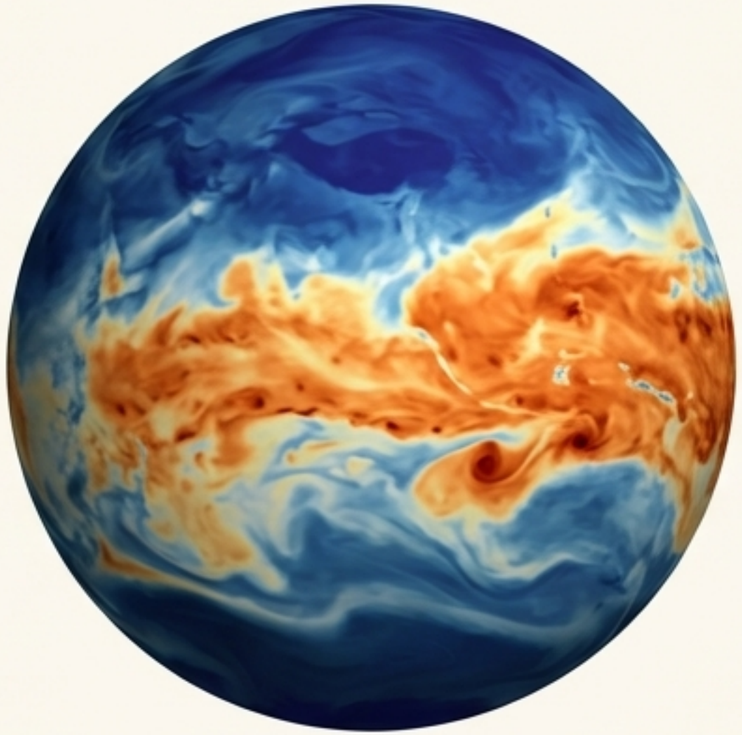
Full Paper: arXiv:2101.04839


Analyzing the Cosmos with Spherical Deep Learning
A Step-by-Step Guide to the `deepsphere-cosmo-tf2` Quick Start

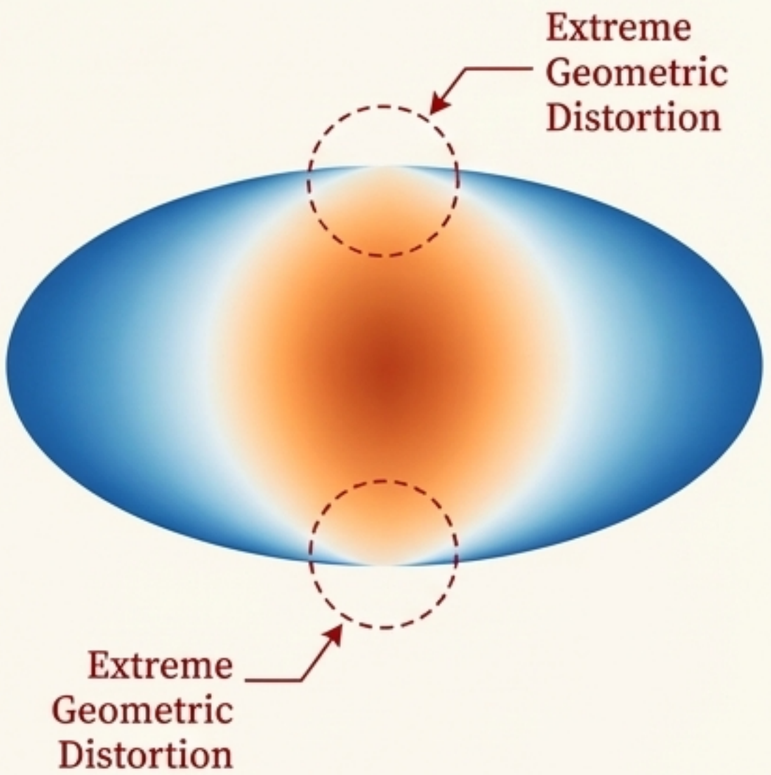
The Challenge of Analyzing Spherical Data

- Scientific fields from cosmology to climate science and neuroscience generate data that naturally exists on a **sphere**.
- Applying standard **2D Convolutional Neural Networks** (CNNs) to this data is problematic.
- Projecting a sphere onto a 2D plane introduces significant **geometric distortions**, especially at the poles, which the network must learn to compensate for.
- More importantly, standard 2D convolutions **are not equivariant to 3D rotations**. The network's ability to detect a pattern depends on its orientation on the sphere, breaking a fundamental symmetry of the data and leading to poor generalization.

Native Spherical Data



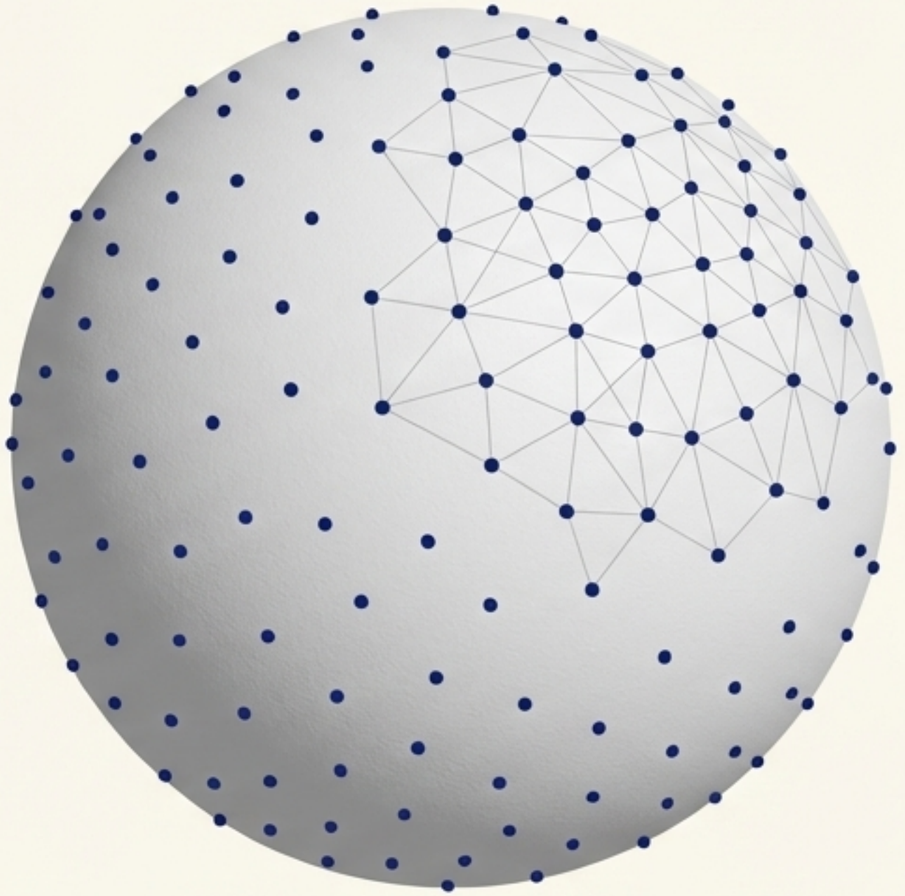
2D Planar Projection



A Graph-Based Solution: The DeepSphere Framework

DeepSphere is a novel approach to spherical CNNs that models the sampled sphere as a graph, connecting nearby pixels. This design provides a powerful combination of benefits:

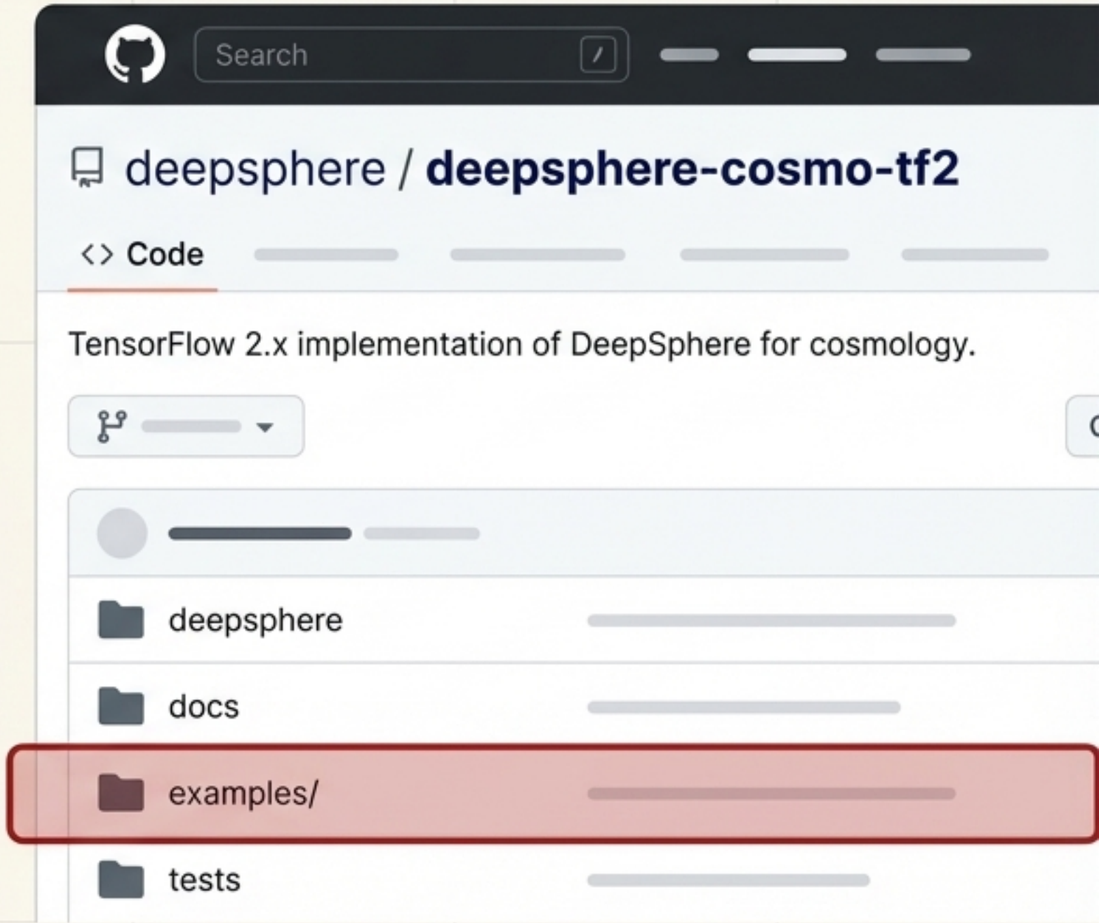
- **Computational Efficiency**: Graph convolutions scale linearly with the number of pixels ($O(N_{\text{pix}})$), making it feasible to analyze high-resolution maps with millions of pixels.
- **Sampling Flexibility**: The graph structure can accommodate non-uniform, partial, and changing samplings without requiring interpolation. This is critical for real-world data from weather stations or telescopes with specific survey footprints.
- **Approaching Equivariance**: The graph Laplacian's properties allow the convolutions to approximate rotation equivariance, enabling the network to learn features independent of their orientation.



Our Toolkit: deepsphere-cosmo-tf2

For this walkthrough, we will use the **deepsphere-cosmo-tf2** repository.

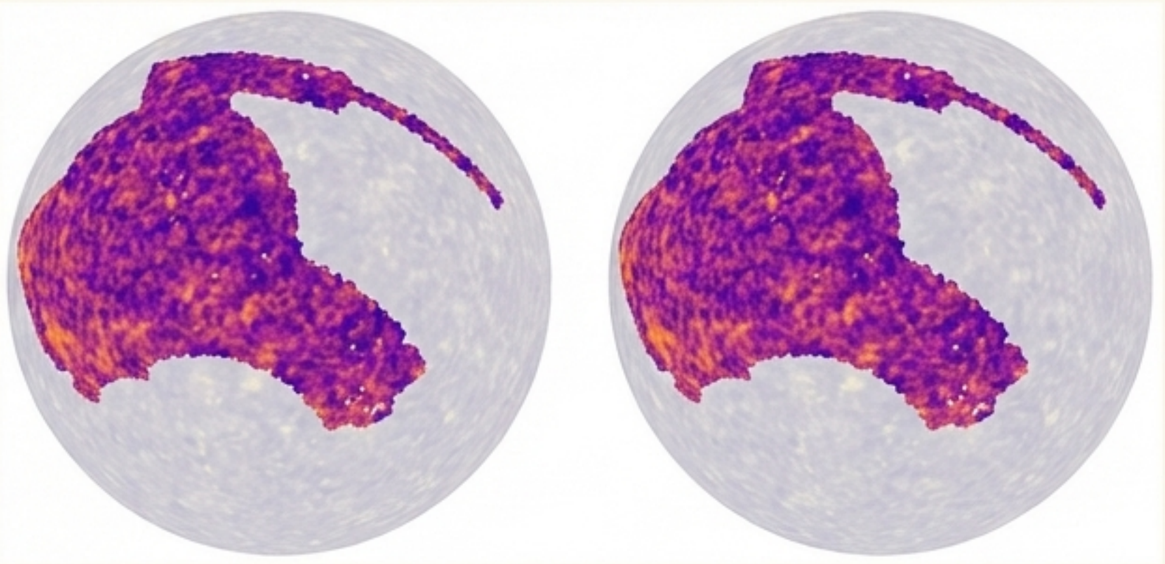
- **Purpose:** A modern, TensorFlow 2.x implementation of DeepSphere specifically tailored for cosmological applications.
- **Core Feature:** Optimized for the **HEALPix** (Hierarchical Equal Area isoLatitude Pixelization) sampling, a standard in cosmology and astrophysics.
- **Ecosystem:** Part of a family of DeepSphere repositories, including the original TensorFlow 1.x version and a PyTorch implementation, providing options for researchers.



The Quick Start Mission: Classifying Cosmological Models

The `Quick Start` notebook guides us through a fundamental task in physical cosmology: classifying spherical maps based on the underlying cosmological model that generated them.

- **The Task:** We will train a DeepSphere CNN to distinguish between weak lensing convergence maps simulated from two different ΛCDM cosmological models.
- **The Data:** The maps represent the distribution of mass in the universe. They are noisy, and the subtle statistical differences between the two models are what the network must learn to identify.
- **The Goal:** Achieve high classification accuracy, demonstrating the network's ability to learn rotation-invariant features from complex scientific data.



Cosmological Model A Cosmological Model B

Part 1: Setting Up Your Environment

First, we clone the repository and install the necessary dependencies. The `deepsphere` package is designed to be installed in editable mode for easy development.

Code Block 1: Installation

```
# Clone the repository
!git clone https://github.com/deepsphere/deepsphere-cosmo-tf2.git
%cd deepsphere-cosmo-tf2

# Install the package and its dependencies
!pip install -e .
```

- The code requires Python 3.10+.
- Key dependencies installed via `pip` include **tensorflow**, **healpy** (for HEALPix operations), and **pygsp** (for graph signal processing).
- You can verify the installation by running `pytest tests`.

Part 2: Imports and Experiment Parameters

Every experiment starts with importing the required libraries and defining key parameters. In this notebook, the most important parameter is `Nside`, which controls the resolution of our HEALPix spherical maps.

****Code Block 2: Imports & Setup****

```
import os
import shutil
import numpy as np
import tensorflow as tf
from matplotlib import pyplot as plt

from deepsphere import models
from deepsphere.data import HEALPixDataset

# Experiment parameters
Nside = 1024
learning_rate = 1e-4
n_epochs = 10
```

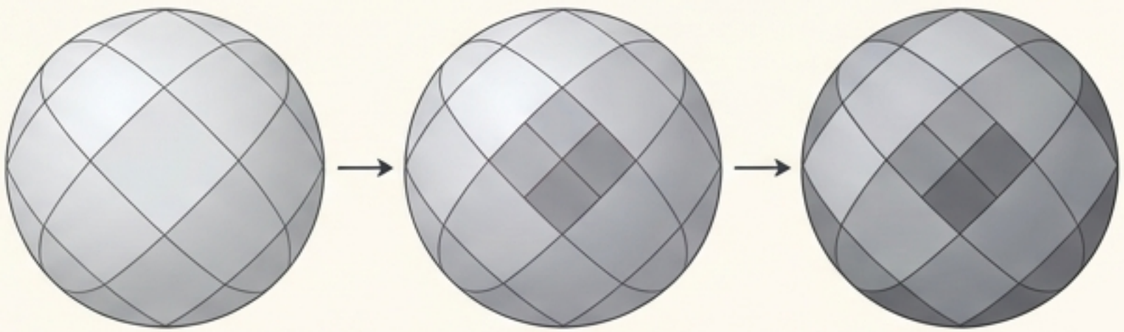
- We import standard libraries like `numpy` and `tensorflow`.
- Crucially, we import the `models` module from our `deepsphere` library and the `HEALPixDataset` helper class.
- `Nside = 1024` defines a high-resolution sphere with over 12 million pixels ($12 \times N_{side}^2$), demonstrating the scalability of DeepSphere.

A Primer on HEALPix Sampling

The HEALPix format is the standard for spherical data in cosmology. It divides the sphere into pixels of equal area, arranged in a hierarchical structure.

- Equal Area: Ensures that each pixel represents the same solid angle on the sky, preventing statistical bias.
- Hierarchical: The resolution is controlled by the `Nside` parameter. Pooling and upsampling are natural operations, simply moving between levels in the hierarchy.
- isoLatitude: Pixels are arranged in rings of constant latitude.

This structure is ideal for spherical harmonic transforms and for building the underlying graph in DeepSphere.



Part 3: Loading the Cosmological Maps

We use the `HEALPixDataset` class to manage our data. This class handles downloading the pre-simulated maps, creating TensorFlow datasets, and preparing batches for the model.

Code Block 3: Data Loading

```
# Path to data directory
datadir = "data/scnn_cosmo_data"

# Instantiate the dataset for training
train_dataset = HEALPixDataset(datadir, "train",
n_pixels=12*Nside**2)
train_tf_dataset = train_dataset.get_tf_dataset(batch_size=4)

# Instantiate the dataset for validation
val_dataset = HEALPixDataset(datadir, "val",
n_pixels=12*Nside**2)
val_tf_dataset = val_dataset.get_tf_dataset(batch_size=4)
```

- The dataset consists of partial-sky convergence maps, covering approximately 8% of the full sphere.
- The `HEALPixDataset` class abstracts away the file handling, returning a `tf.data.Dataset` object, which is highly efficient for model training in TensorFlow.

Part 4: Building the Spherical CNN

We construct the model using the **Keras Functional API**, stacking **DeepSphere's** custom layers. The architecture is a fully convolutional network (FCN) designed for rotation-invariant classification.

```
# Define model parameters
params = {
    'nsides': [1024, 512, 256, 128, 64],
    'n_neighbors': 20,
    'K': [4, 4, 4, 4, 4], # Chebyshev polynomial order
    'n_filters': [16, 32, 64, 64, 64],
    'pool': 'healpy_pool_max'
}

# Build the model
model = models.deepsphere_FCN(
    n_pixels=12*Nside**2,
    n_classes=2,
    **params
)

model.summary()
```



- **Input Layer:** Accepts flattened HEALPix maps of Nside=1024.



- **SphericalChebConv Layers:** These are the core convolutional layers. They operate on the graph of pixels, learning localized features using Chebyshev polynomial filters. **K** defines the size of the filter's neighborhood.



- **HealpyPool Layers:** Max-pooling layers that reduce the Nside of the map by a factor of 2 at each step, decreasing spatial resolution while increasing the feature depth.



- **Global Average Pooling (GAP):** This final layer averages the features across all pixels, producing a single feature vector for the entire map. This step is what makes the FCN architecture invariant to rotations.



- **Output Layer:** A dense layer with a softmax activation to produce classification probabilities for the two cosmological models.

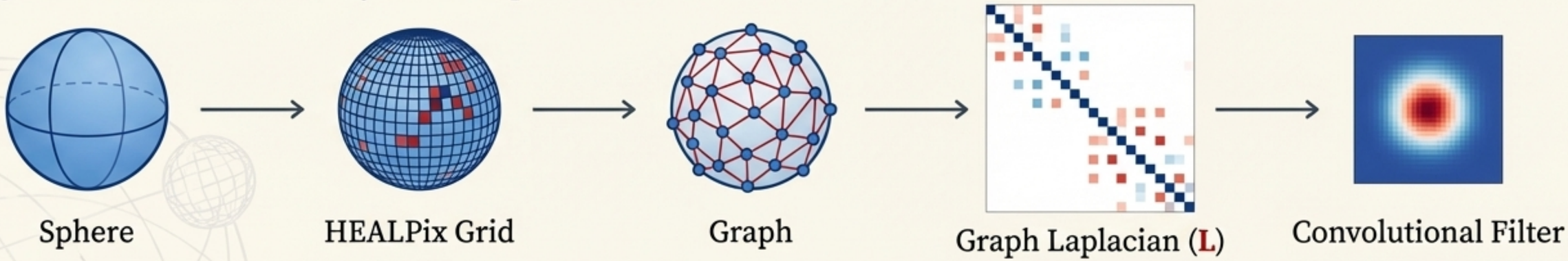
Deep Dive: How Spherical Graph Convolution Works

The `SphericalChebConv` layer is not a standard 2D convolution. It's a spectral filter defined on the graph Laplacian, L.

1. **The Graph:** First, a graph is constructed where each HEALPix pixel is a node, connected to its nearest neighbors.
2. **The Laplacian:** The graph Laplacian $L = D - A$ captures the graph's structure. Its eigenvectors are the graph equivalent of Fourier modes.
3. **The Filter:** Instead of costly eigendecomposition, the convolution is approximated by a polynomial of the Laplacian. DeepSphere uses Chebyshev polynomials for stability and efficiency:

$$g(L)x \approx \sum_{k=0}^{K-1} c_k T_k(\tilde{L})x.$$

The model learns the coefficients c_k for each filter during training. This allows it to learn features like edges and textures directly on the sphere.



Part 5: Compiling and Training the Model

Once the model is built, the training process uses the standard Keras API. We compile the model with an optimizer and loss function, then call `model.fit()` on our TensorFlow dataset.

Code Block 4: Training

```
# Compile the model
optimizer  = tf.keras.optimizers.Adam(learning_rate=learning_rate)
loss = tf.keras.losses.SparseCategoricalCrossentropy()
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

# Train the model
history = model.fit(
    train_tf_dataset,
    epochs=n_epochs,
    validation_data=val_tf_dataset
)
```

- We use the Adam optimizer, a robust choice for many deep learning tasks.
- `SparseCategoricalCrossentropy` is used for integer-based labels (0 or 1).
- The `model.fit` call iterates through the dataset for a specified number of epochs, updating the model's weights and printing progress.

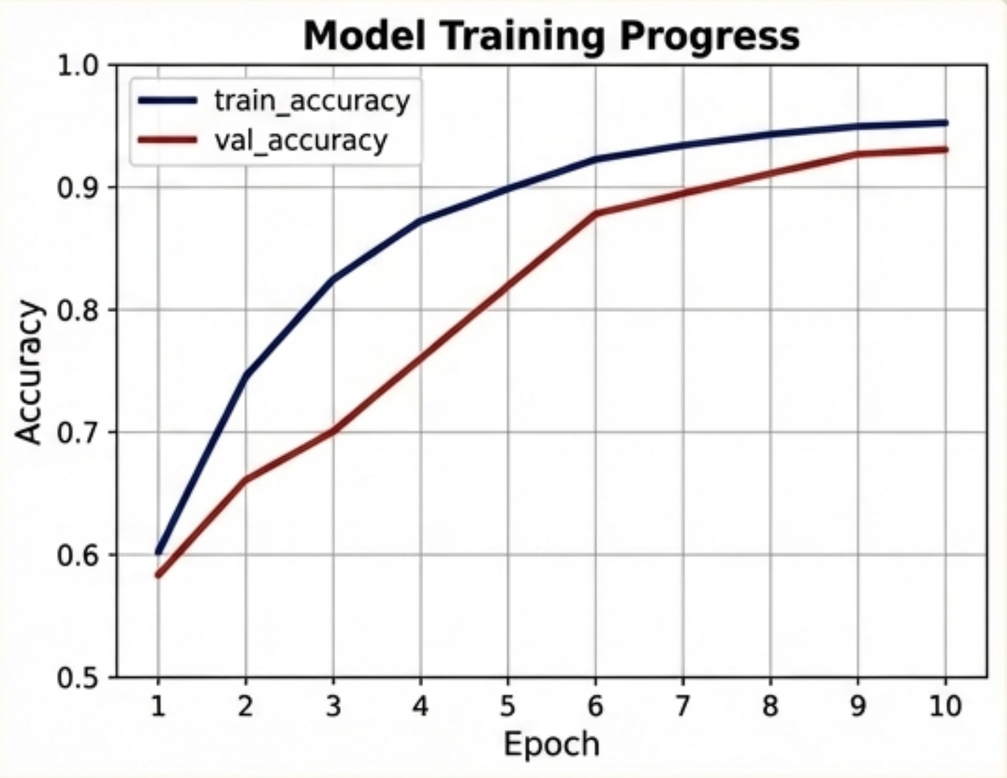
Part 6: Evaluating Performance

After training, we evaluate the model on the validation set to assess its generalization performance. The FCN architecture's rotation invariance allows it to achieve high accuracy, significantly outperforming baselines.

Code Block 5: Evaluation & Visualization

```
# Evaluate the final model
loss, accuracy = model.evaluate(val_tf_dataset)
print(f"Validation Accuracy: {accuracy:.4f}")

# Plot training history
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



The Path Forward: Beyond the Quick Start

The Quick Start notebook is just the beginning. The deepsphere-cosmo-tf2 repository contains further examples to deepen your understanding and tackle more complex problems:

Advanced Tutorial

Learn to build custom training loops with `tf.GradientTape`, work with custom survey masks to handle real-world observational data, and gain finer control over the model.

Generative Models

Explore how to build a spherical autoencoder using transpose convolutions to generate new cosmological maps.

Explore other Implementations

Check out the `deepsphere-pytorch` repository for a PyTorch-native version of the framework.