ellis UNIT AMSTERDAM | new theory | Deep Thinking Hour

geometry-grounded representations in practice

hackathon 27 feb 2025

Overview

- Intro to Equivariant Neural Fields (60 min)
  - Neural Fields as signal representations
  - Theoretical background
- LUNCH! At NEO in the building next-door.
- Implementation of Equivariant Neural Fields in JAX (30 min)
- Problem introduction (10 min)

Equivariant Neural Fields

intro to learning with functa

ICLR 2025!

Grounding Continuous Representations in Geometry: Equivariant Neural Fields

David R. Wessels[*,1], **David M. Knigge**[*,1], **Riccardo Valperga**[1], **Samuele Papa**[1], **Sharvaree Vadgama**[1], **Efstratios Gavves** [†,1], **Erik J. Bekkers** [†,1]
[1]University of Amsterdam [2] Netherlands Cancer Institute
d.m.knigge@uva.nl, d.r.wessels@uva.nl

Abstract

Neural Fields (NeFs) are increasingly being leveraged as continuous signal representations. In a *conditional neural field*, a signal is represented by a latent variable that conditions the NeF, whose parametrisation is otherwise shared over an entire dataset. We propose Equivariant Neural Fields (ENFs) based on cross attention transformers, in which NeFs are conditioned on a geometric variable—a latent point cloud—that enables an *equivariant* decoding from latent to field. Our equivariant approach induces a *steerability* property by which both field and latent are grounded in geometry and amenable to transformation laws: if the field transforms, the latent representation transforms accordingly—and vice versa. Crucially, this equivariance relation ensures that the latent is capable of (1) *representing geometric patterns faitfhully*, allowing for geometric reasoning in latent space, (2) *weight-sharing over similar local patterns*, allowing for efficient learning of datasets of fields. These main properties are validated using classification experiments and a verification of the capability of fitting entire datasets, in comparison to other non-equivariant NeF approaches. We further validate the potential of ENFs by demonstrating unique (local) field editing properties.

Neural Fields

image (field)

I_j

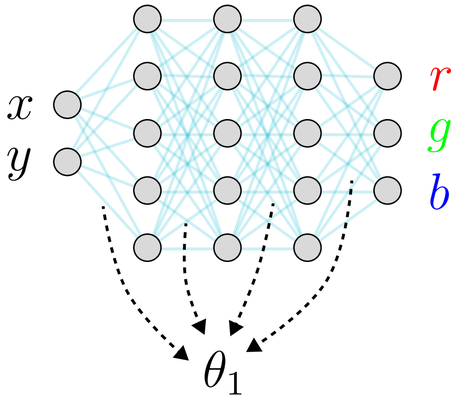
neural field

f_{θ_j}

$$I_j(\mathbf{x}) = f_{\theta_j}(\mathbf{x})$$

neural network weights

θ_j



Neural Fields

image (field)

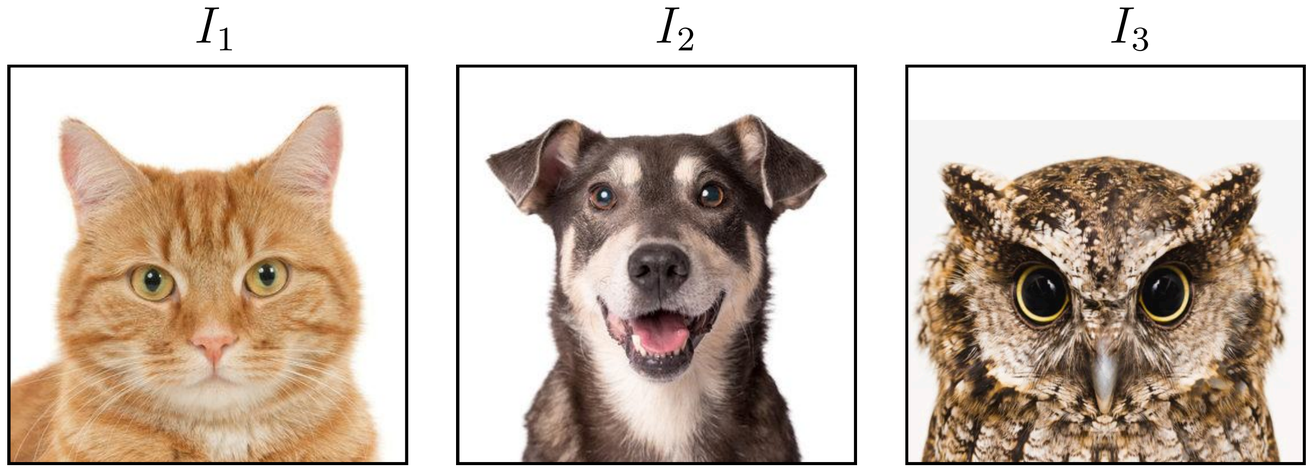
I_j

neural field

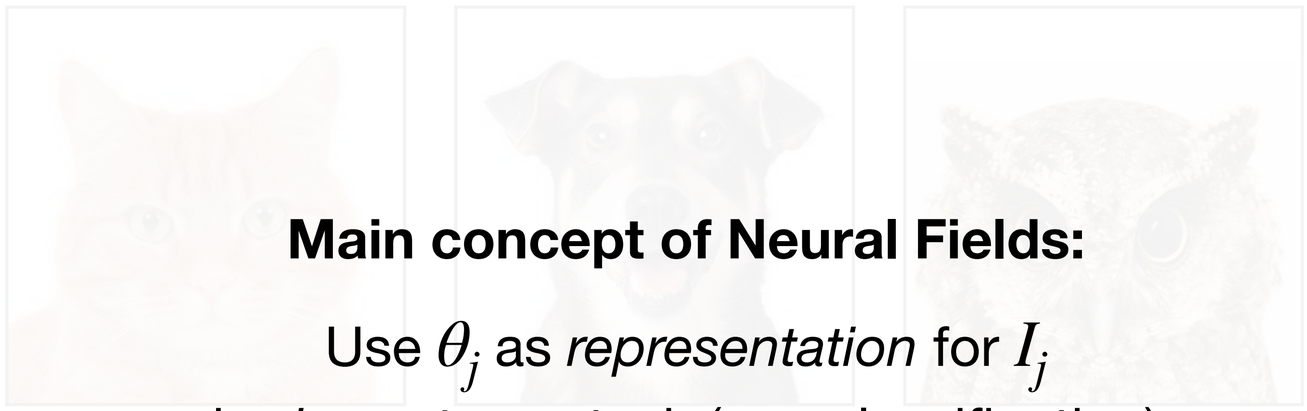
f_{θ_j}

neural network weights

θ_j



Neural Fields

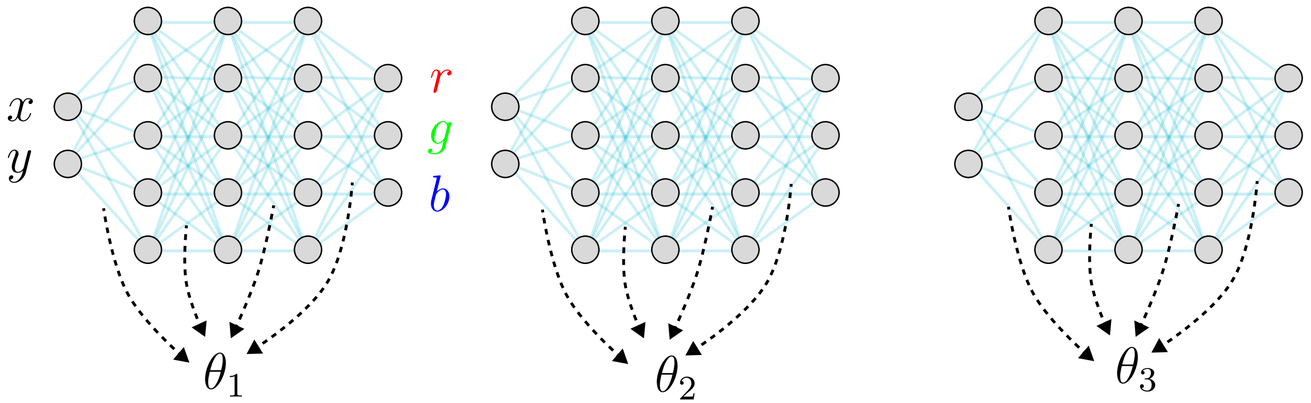


Main concept of Neural Fields:

Use θ_j as *representation* for I_j
in *downstream* task (e.g. classification)

neural network weights

θ_j

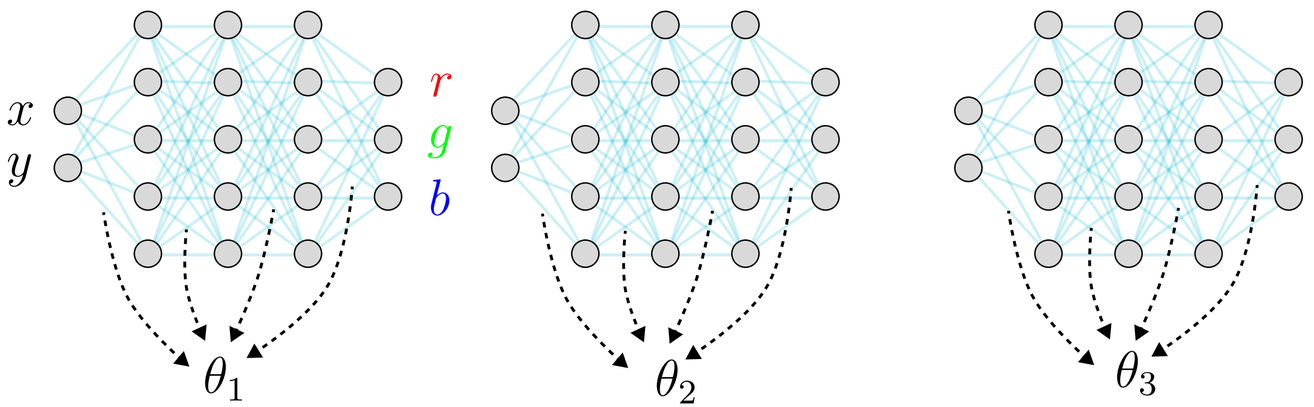


Neural Fields

WHY?

Main concept of Neural Fields:

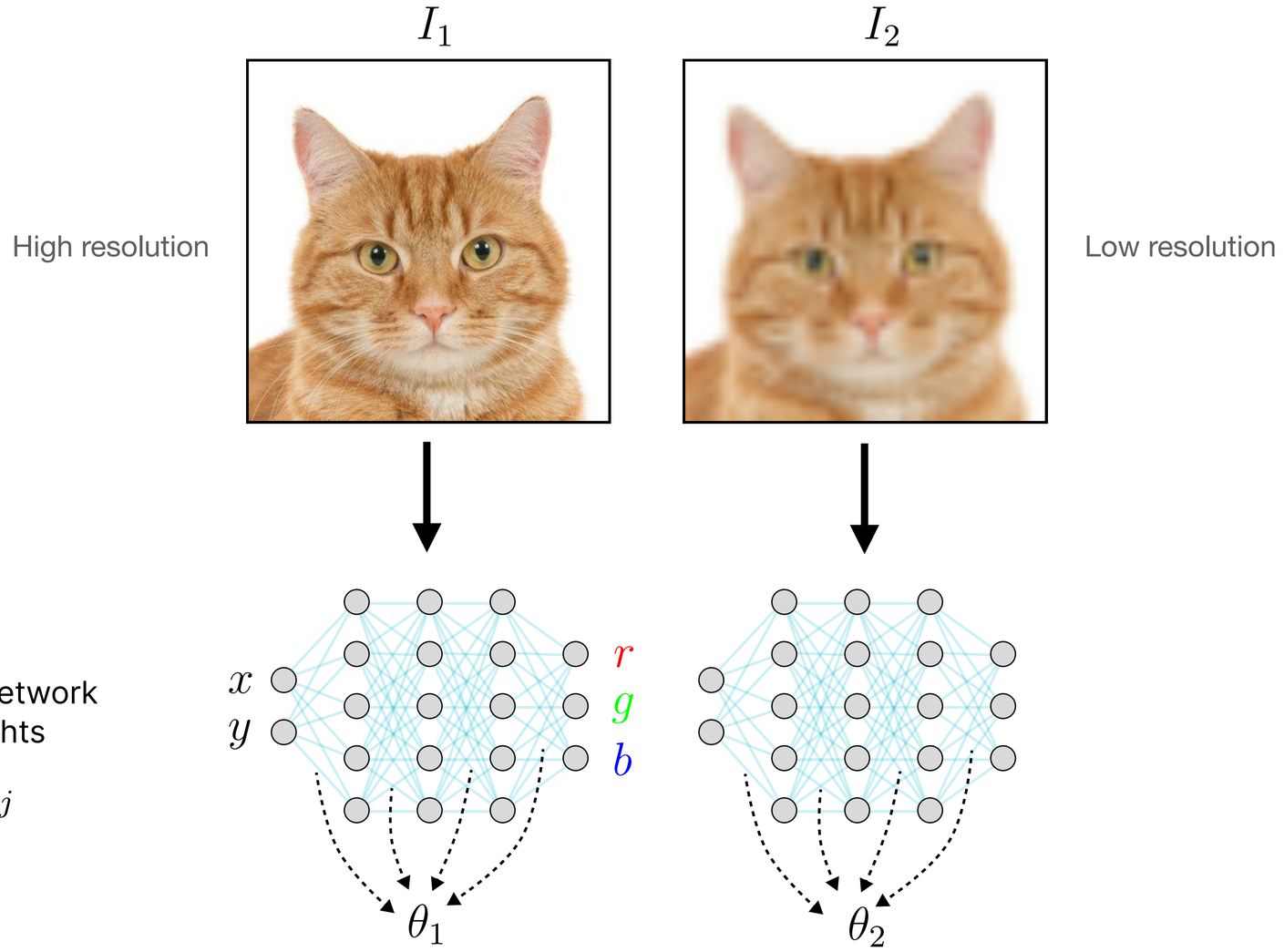
Use θ_j as *representation* for I_j
in *downstream* task (e.g. classification)



neural network weights

θ_j

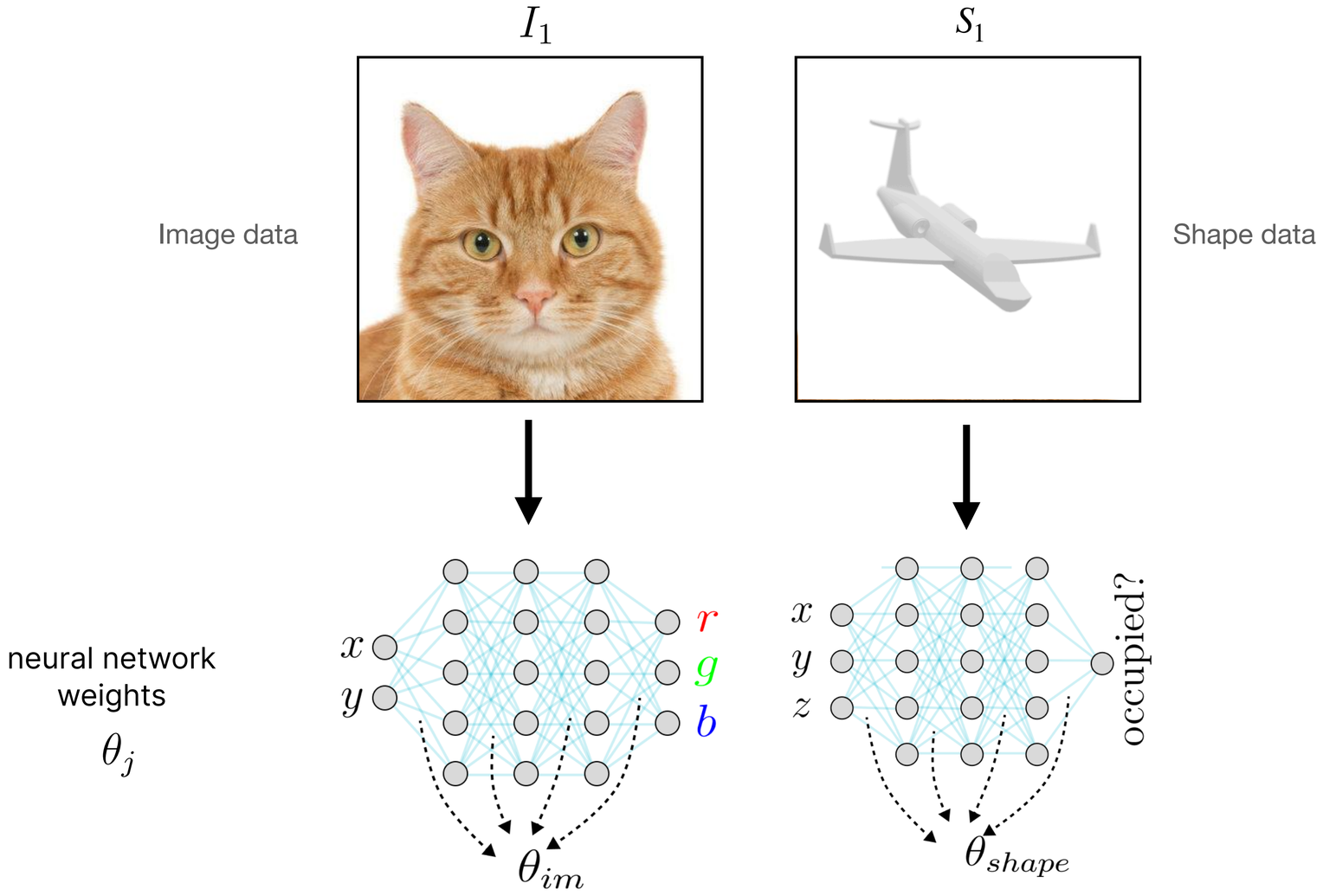
Neural Fields



Any downstream model inherits:

- Resolution-free / grid-free-ness

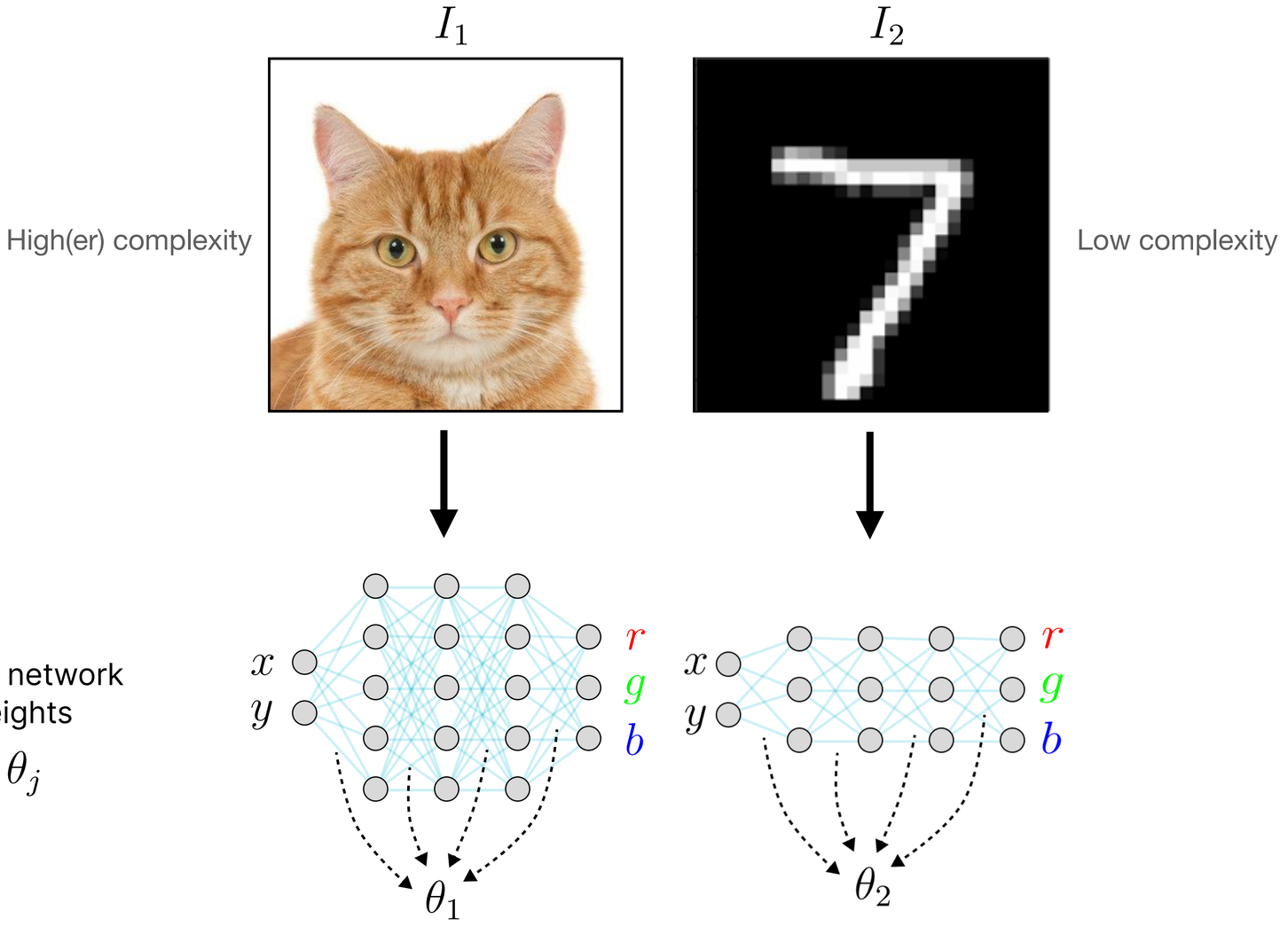
Neural Fields



Any downstream model inherits:

- Resolution-free / grid-free-ness
- Datatype agnosticity

Neural Fields



Any downstream model inherits:

- Resolution-free / grid-free-ness
- Datatype agnosticity
- Scaling w/ signal complexity

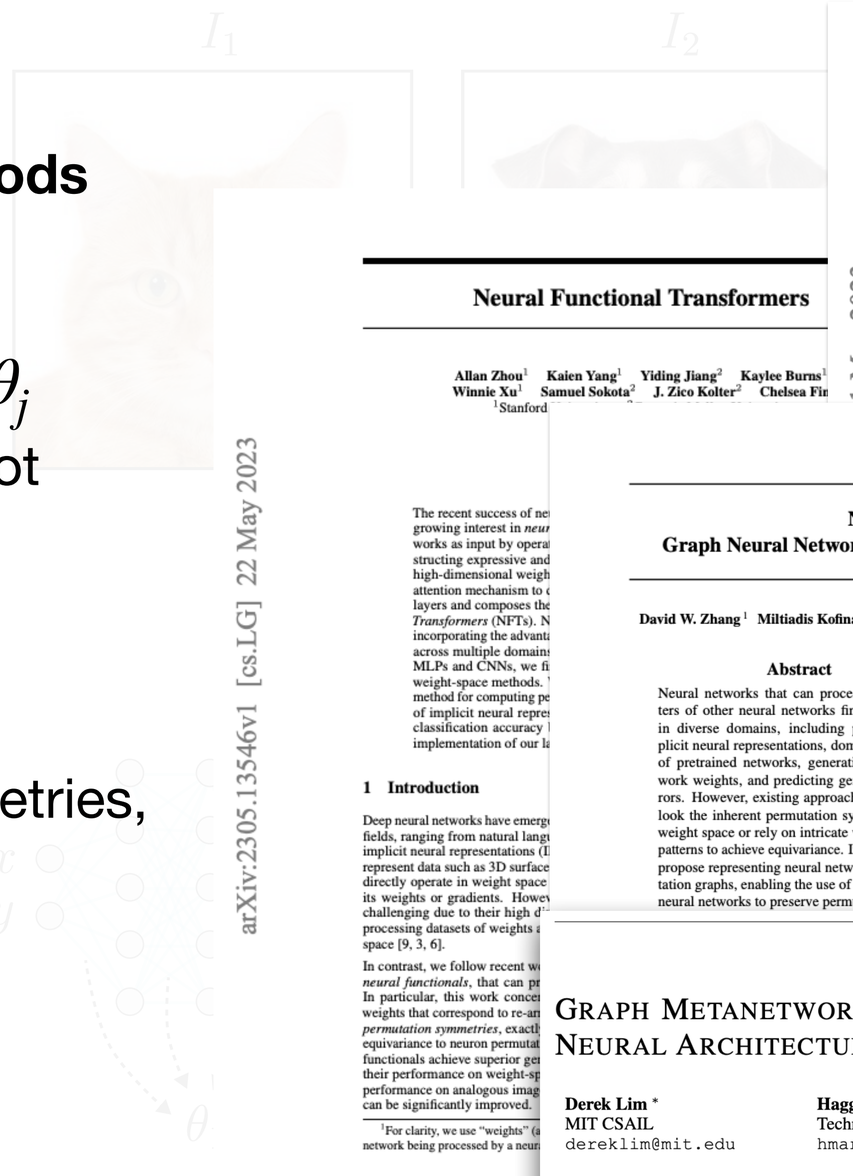
Neural Fields

Weight-space methods

- Operate directly on θ_j
- Works on any NN, not just Neural Fields

- Need to incorporate weight-space symmetries, ambiguities

- Very hard:



Equivariant Architectures for Learning in Deep Weight Spaces

Aviv Navon [*1] Aviv Shamsian [*1] Idan Achituve [1] Ethan Fetaya [1] Gal Chechik [1 2] Haggai Maron [2]

Abstract

Neural Functional Transformers

Allan Zhou[1] Kaien Yang[1] Yiding Jiang[2] Kaylee Burns[1]
Winnie Xu[1] Samuel Sokota[2] J. Zico Kolter[2] Chelsea Fin
[1]Stanford

The recent success of ne
growing interest in *neur*
works as input by opera
structing expressive and
high-dimensional weigh
attention mechanism to
layers and composes the
Transformers (NFTs). N
incorporating the advanta
across multiple domains
MLPs and CNNs, we fi
weight-space methods.
method for computing pe
of implicit neural repres
classification accuracy
implementation of our la

1 Introduction

Deep neural networks have emerg
fields, ranging from natural langu
implicit neural representations (I
represent data such as 3D surface
directly operate in weight space
its weights or gradients. Howev
challenging due to their high d
processing datasets of weights a
space [9, 3, 6].

In contrast, we follow recent w
neural functionals, that can pr
In particular, this work concer
weights that correspond to re-ar
permutation symmetries, exactl
equivariance to neuron permutat
functionals achieve superior ge
their performance on weight-sp
performance on analogous imag
can be significantly improved.

[1]For clarity, we use "weights" (a
network being processed by a neur

JOURNAL OF LATEX CLASS FILES, VOL. 14, NO. 8, AUGUST 2021 1

Deep Learning on Object-centric 3D Neural Fields

Pierluigi Zama Ramirez* , Luca De Luigi*, Daniele Sirocchi*
Adriano Cardace, Riccardo Spezialetti, Francesco Ballerini, Samuele Salti, Luigi Di Stefano
of Bologna
{adriano.cardace2, riccardo.spezialetti}@unibo.it

Representing a 3D scene by encoding it with a continuous function parameterized as an MLP separates the memory cost of the representation from the spatial resolution. In other words, starting from the same fixed number of parameters, it is possible to reconstruct a surface with arbitrarily fine resolution or to

Neural Networks Are Graphs!
Graph Neural Networks for Equivariant Processing of Neural Networks

David W. Zhang [1] Miltiadis Kofinas [1] Yan Zhang [2] Yunlu Chen

Abstract

Neural networks that can process the parameters of other neural networks find applications in diverse domains, including processing implicit neural representations, domain adaptation of pretrained networks, generating neural network weights, and predicting generalization errors. However, existing approaches either overlook the inherent permutation symmetry in the weight space or rely on intricate weight-sharing patterns to achieve equivariance. In this work, we propose representing neural networks as computation graphs, enabling the use of standard graph neural networks to preserve permutation symme-

How to Train Neural Field Representations: A Comprehensive Study and Benchmark

Samuele Papa[1,2], Riccardo Valperga[1], David Knigge[1,2], Miltiadis Kofinas[1], Phillip Lippe[1],
Jan-Jakob Sonke[2], Efstratios Gavves[1]
[1] University of Amsterdam, [2] Netherlands Cancer Institute
s.papa@{uva,nki}.nl, {r.valperga, dm.knigge, m.kofinas, p.lippe}@uva.nl,
j.sonke@nki.nl, e.gavves@uva.nl

Abstract

recently emerged as a versatile
s of various modalities, includ-
enes. Subsequently, a number
use of NeFs as representations
classifying an image based on
at has been fit to it. However,
parameters on their quality as
is scarcely understood and re-

36, 51], to medical imaging [35, 53], audio [18, 44], and physical systems [24, 41, 50]. The continuous nature of coordinate-based models $f_\theta : x \mapsto y, x \in \mathbb{R}^d, y \in \mathbb{R}^c$ (e.g. $d = 2$ for images and $d = 3$ for 3D shapes), allows for modeling signals without the limitations in resolution inherent in discrete representations; their capacity for modeling fine detail is only limited by the expressivity of the underlying NeF architecture. Furthermore, some modalities, such as scenes and shapes, do not have array representations that can be easily adapted to existing deep learning methods.

GRAPH METANETWORKS FOR PROCESSING DIVERSE NEURAL ARCHITECTURES

Derek Lim *
MIT CSAIL
dereklim@mit.edu

Haggai Maron
Technion / NVIDIA
hmaron@nvidia.com

Marc T. Law
NVIDIA
marcl@nvidia.com

Neural Fields

Weight-space methods

- Operate directly on θ_j
- Works on any NN, not just Neural Fields

- Need to incorporate weight-space symmetries, ambiguities

- Very hard:

Table 1: Test accuracy (%) for weight-space INR classification in the MNIST, FashionMNIST, and CIFAR-10 datasets. INR2ARRAY significantly improves over current state-of-the-art results in this setting. For the NFN baseline [46] we report the higher performance out of their NP and HNP variants. Uncertainties indicate standard error over three runs.

	MNIST	FashionMNIST	CIFAR-10
DWS [31]	85.7 ± 0.57	65.5 ± 0.48	–
NFN [46]	92.9 ± 0.38	75.6 ± 1.07	46.6 ± 0.13
Spatial NFN	92.9 ± 0.46	70.8 ± 0.53	45.6 ± 0.11
INR2ARRAY$^{\text{NFN}}$ (Ours)	94.6 ± 0.00	76.7 ± 0.00	45.4 ± 0.00
INR2ARRAY$^{\text{NFT}}$ (Ours)	$\mathbf{98.5 \pm 0.00}$	$\mathbf{79.3 \pm 0.00}$	$\mathbf{63.4 \pm 0.00}$

Conditional Neural Fields

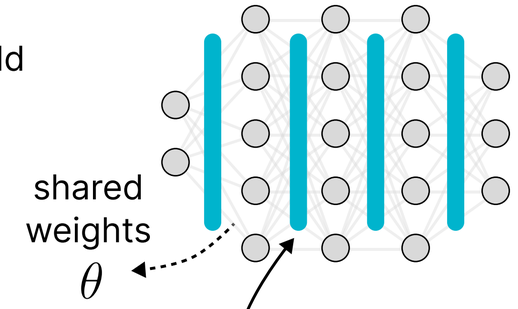
I_1

image (field)

I_j

conditional neural field

$f_\theta(\,\cdot\,, z_j)$



Latent vectors

z_j

that modulate/condition
network layers

$$\begin{pmatrix} 0.63 \\ -1.80 \\ -0.74 \\ 0.23 \\ 0.74 \\ \vdots \end{pmatrix}$$

z_1

$$I_j(\mathbf{x}) = f_\theta(\mathbf{x}, z_j)$$

Leverage consistencies between datapoints!
Reduce weight-space complexities!

Representations z_j and

network weights θ learned through **auto-decoding** and **meta-learning**

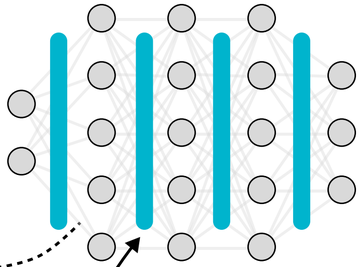
Conditional Neural Fields

image (field)
I_j

I_1 I_2 I_3



conditional neural field
$f_\theta(\,\cdot\,, z_j)$



shared weights
θ

$$I_j(\mathbf{x}) = f_\theta(\mathbf{x}, z_j)$$

Latent vectors
z_j
that modulate/condition network layers

$$\begin{pmatrix} 0.63 \\ -1.80 \\ -0.74 \\ 0.23 \\ 0.74 \\ \vdots \end{pmatrix} \quad \begin{pmatrix} 0.63 \\ -1.80 \\ -0.74 \\ 0.23 \\ 0.74 \\ \vdots \end{pmatrix} \quad \begin{pmatrix} 0.63 \\ -1.80 \\ -0.74 \\ 0.23 \\ 0.74 \\ \vdots \end{pmatrix}$$

z_1 z_2 z_3

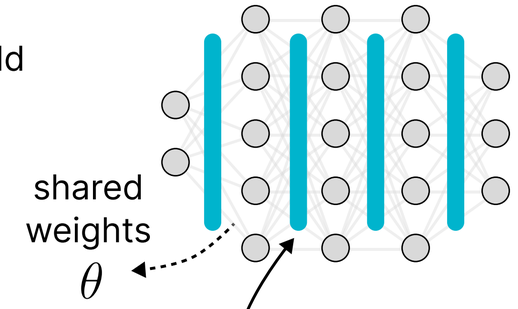
Conditional Neural Fields

Main concept of Conditional Neural Fields:

Share θ over dataset, modulate activations using latent z_j, use this as *representation* for I_j in *downstream* task (e.g. classification)

conditional neural field

$f_\theta(\,\cdot\,, z_j)$



$$I_j(\mathbf{x}) = f_\theta(\mathbf{x}, z_j)$$

Latent vectors

z_j

that modulate/condition network layers

$$z_1 = \begin{pmatrix} 0.63 \\ -1.80 \\ -0.74 \\ 0.23 \\ 0.74 \\ \vdots \end{pmatrix} \quad z_2 = \begin{pmatrix} 0.63 \\ -1.80 \\ -0.74 \\ 0.23 \\ 0.74 \\ \vdots \end{pmatrix} \quad z_3 = \begin{pmatrix} 0.63 \\ -1.80 \\ -0.74 \\ 0.23 \\ 0.74 \\ \vdots \end{pmatrix}$$

Conditio

image (field)

I_j

conditional neural field

$f_\theta(\cdot, z_j)$

shared weights

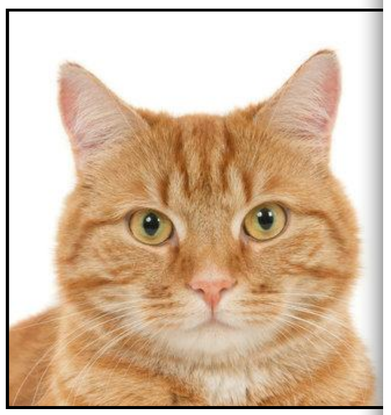
θ

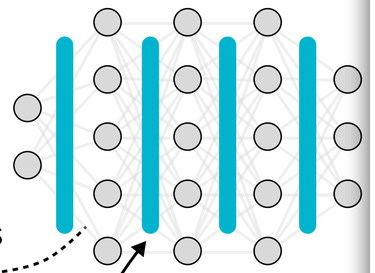
Latent vectors

z_j

that modulate/condition network layers

I_1





$$\begin{pmatrix} 0.63 \\ -1.80 \\ -0.74 \\ 0.23 \\ 0.74 \\ \vdots \end{pmatrix}$$

z_1

ICML '22 **From data to functa: Your data point is a function and you can treat it like one**

Emilien Dupont [*1] **Hyunjik Kim** [*2] **S. M. Ali Eslami** [2] **Danilo Rezende** [2] **Dan Rosenbaum** [3 2]

Abstract

It is common practice in deep learning to represent a measurement of the world on a discrete grid, e.g. a 2D grid of pixels. However, the underlying signal represented by these measurements is often continuous, e.g. the scene depicted in an image. A powerful continuous alternative is then to represent these measurements using an *implicit neural representation*, a neural function trained to output the appropriate measurement value for any input spatial location. In this paper, we take this idea to its next level: what would it take to perform deep learning on these functions instead, treating them as data? In this context we refer to the data as *functa*, and propose a framework for deep learning on functa. This view presents a number of challenges around efficient conversion from data to functa, compact representation of functa, and effectively solving downstream tasks on functa. We outline a recipe to overcome these challenges and apply it to a wide range of data modalities including images, 3D shapes, neural radiance fields (NeRF) and data on manifolds. We demonstrate that this approach has various compelling properties across data modalities, in particular on the canonical tasks of generative modeling, data imputation, novel view synthesis and classification. Code: `github.com/deepmind/functa`.

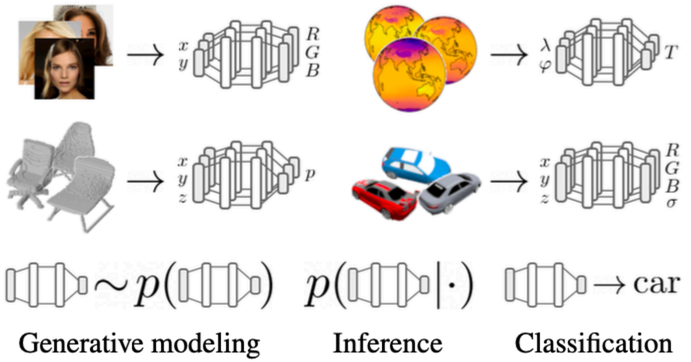


Figure 1. We convert array data into functional data parameterized by neural networks, termed *functa*, and treat these as data points for various downstream machine learning tasks.

tion. However, the underlying signal represented by these arrays is often continuous. It is therefore natural to consider representing such data with continuous quantities.

Recently, the idea of modelling data with continuous functions has gained popularity. An image, for example, can be represented by a continuous function mapping 2D pixel coordinates to RGB values. When such a function is parameterized by a neural network, it is typically referred to as an *implicit neural representation* (INR). INRs are generally applicable to a wide range of modalities – indeed, various works have demonstrated that INRs can be used to represent images (Stanley, 2007; Ha, 2016), 3D shapes (Mescheder

Conditional Neural Fields

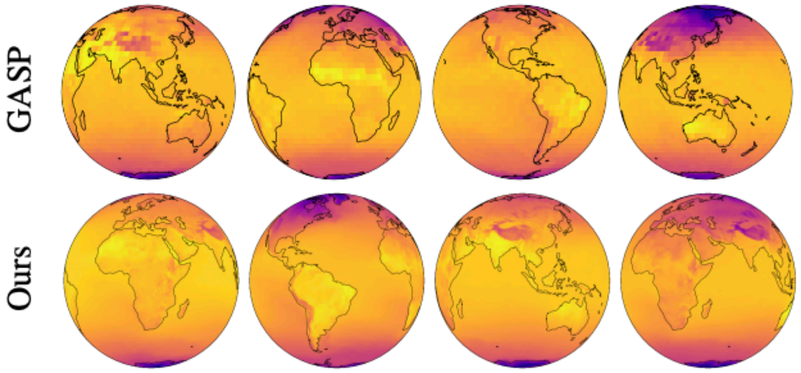
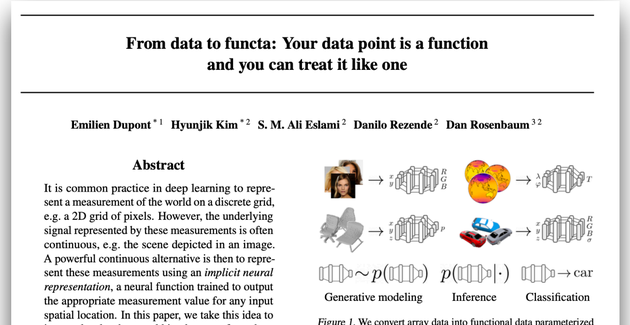


Figure 12. Uncurated samples from GASP and a flow trained on 256-dim modulations of ERA5. GIF: github.com/deepmind/functa#figure-12

Figure 5. Uncurated samples from GASP and DDPM (diffusion) trained on 256-dim CelebA-HQ 64×64 modulations.

- Better results than weight-space methods!
- But…

Figure 7. Imputation from partial test observations using the learned distribution over functa as a prior. GIF showing course of optimization: github.com/deepmind/functa#figure-7. Additional chairs: Figure 27



From data to functa: Your data point is a function and you can treat it like one

Emilien Dupont [*1] Hyunjik Kim [*2] S. M. Ali Eslami [2] Danilo Rezende [2] Dan Rosenbaum [3 2]

Abstract

It is common practice in deep learning to represent a measurement of the world on a discrete grid, e.g. a 2D grid of pixels. However, the underlying signal represented by these measurements is often continuous, e.g. the scene depicted in an image. A powerful continuous alternative is then to represent these measurements using an *implicit neural representation*, a neural function trained to output the appropriate measurement value for any input spatial location. In this paper, we take this idea to

Generative modeling Inference Classification

Figure 1. We convert array data into functional data parameterized

Conditional Neural Fields

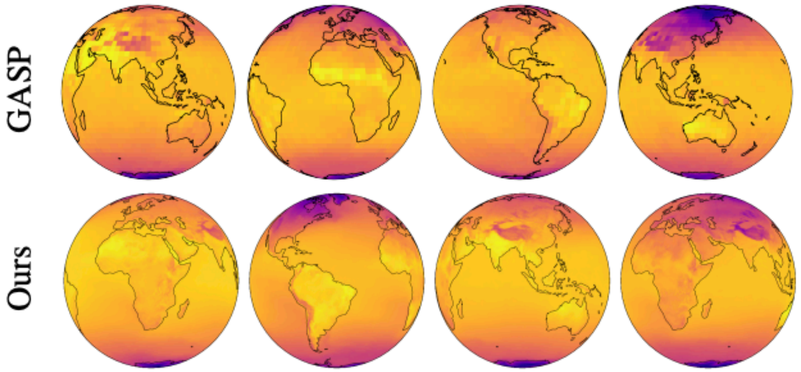


Figure 12. Uncurated samples from GASP and a flow trained on 256-dim modulations of ERA5. GIF: github.com/deepmind/functa#figure-12

Figure 5. Uncurated samples from GASP and DDPM (diffusion) trained on 256-dim CelebA-HQ 64×64 modulations.

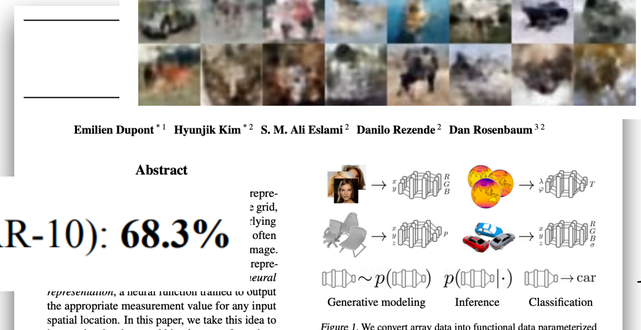
Figure 7. Imputation from partial test observations using the learned distribution over functa as a prior. GIF showing course of optimization: github.com/deepmind/functa#figure-7. Additional chairs: Figure 27

- Better results than weight-space methods!
- But…

Performance on non-globally-aligned data is still disappointing

Unconditional samples (CIFAR-10), FID = **78.2**

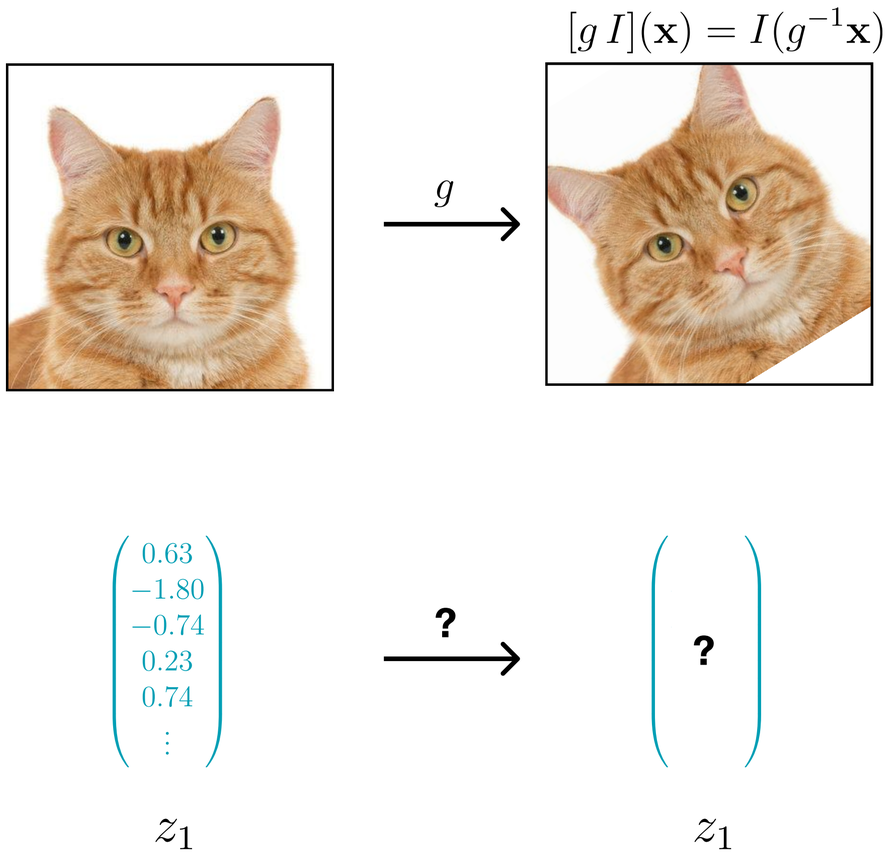
Classification top-1 accuracy (CIFAR-10): **68.3%**



Conditional Neural Fields

Our hypothesis on what is lacking:

- *Geometry*
  - Location of features
  - Spatial relation between features
- *Locality*
  - Small perturbations to latent have *global* effect on field, and vice-versa



Equivariant Neural Fields

Idea: introduce **geometry** and **locality** in Neural Field latent space.

Equivariant Neural Field:
Replace single latent $z_j \in \mathbb{R}^d$ by point cloud of smaller, localized latents $z_j := \{(p_i, \mathbf{c}_i\}_{i=1}^N$.

image (field)
I_j

I_1

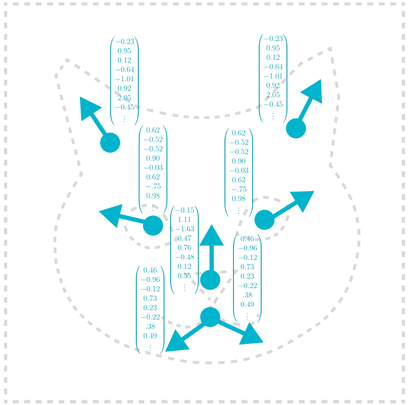
equivariant neural field (ENF)
$f_\theta(\,\cdot\,, z_j)$

$I_j(\mathbf{x}) = f_\theta(\mathbf{x}, z_j)$

latent point cloud
$z_j = \{(p_i^j, \mathbf{c}_i^j)\}_{i=1}^N$

poses
$p_i^j = \in G$

context vecs
$\mathbf{c}_i^j = \begin{pmatrix} 0.43 \\ -1.25 \\ 0.98 \\ 1.23 \\ 0.53 \\ \vdots \end{pmatrix} \in \mathbb{R}^{\boldsymbol{c}}$



z_1

Equivariant Neural Fields

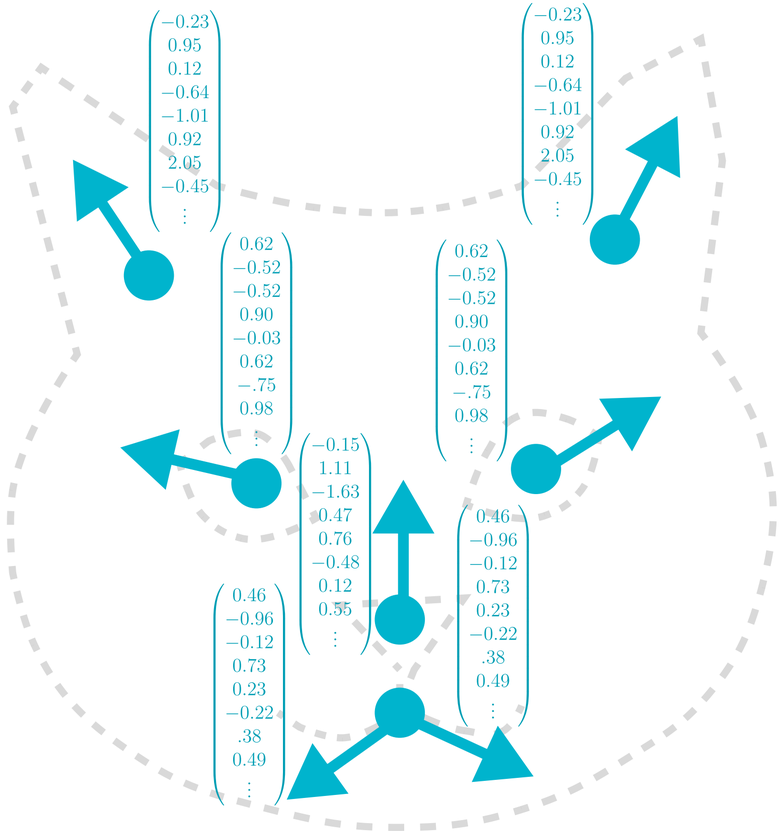
Image I_1

Equivariant
Neural Field

$f_\theta(\cdot\ , z_1)$

Equivariant
cross-attention
transformer

Latent (neural ideogram) z_1



Equivariant Neural Fields

image (field)

I_j

I_1 I_2 I_3



equivariant neural field (ENF)

$f_\theta(\cdot, z_j)$

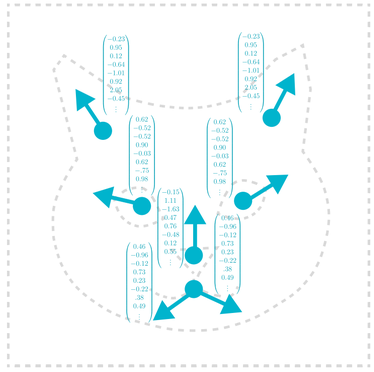
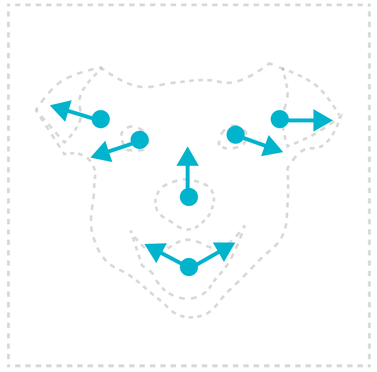
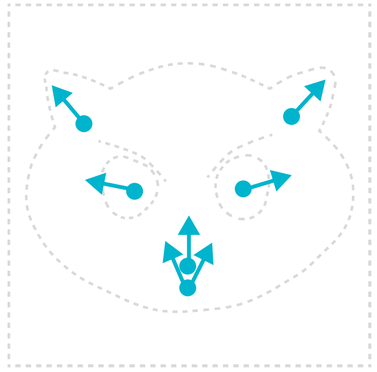
$$I_j(\mathbf{x}) = f_\theta(\mathbf{x}, z_j)$$

latent point cloud

$$z_j = \{(p_i^j, \mathbf{c}_i^j)\}_{i=1}^N$$

poses $p_i^j = \in G$

context vecs $\mathbf{c}_i^j = \begin{pmatrix} 0.43 \\ -1.25 \\ 0.98 \\ 1.23 \\ 0.53 \\ \vdots \end{pmatrix} \in \mathbb{R}^c$

z_1 z_2 z_3

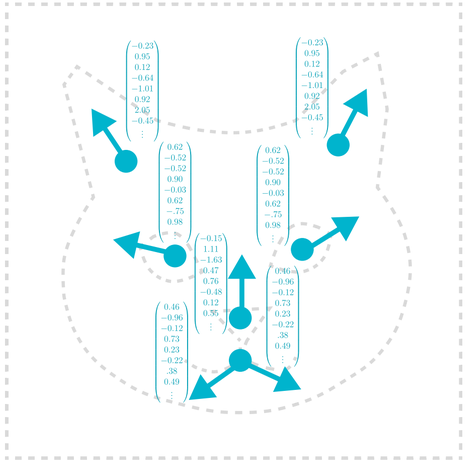
left action on field $I(\mathbf{x})$

g

$[g\,I](\mathbf{x}) = I(g^{-1}\mathbf{x})$

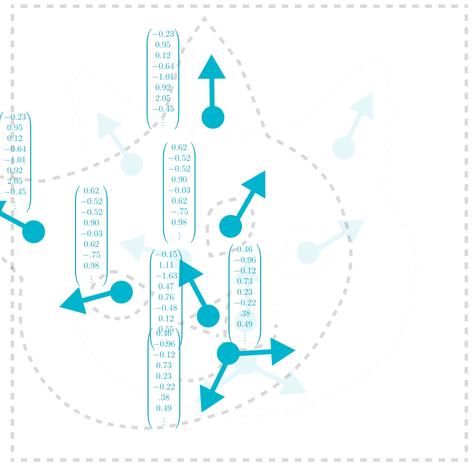
steerability property of ENFs

$f_\theta(g^{-1}\mathbf{x}, z) = f_\theta(\mathbf{x}, g\,z)$



z

g



$g\,z = \{(g\,p_i, \mathbf{c}_i)\}_{i=1}^N$

left action on latent poses

Equivariant Neural Fields

tecnical background

Equivariant Neural Fields

$$\textit{Steerability property:} \qquad \forall g \in G : f_\theta(g^{-1}x; z) = f_\theta(x; gz)\,.$$

Steerability intuition:

If I want to rotate the function modelled by the ENF, I can either:
- Rotate the function domain
Or
- Rotate the latents

Equivariant Neural Fields

Steerability property: $\forall g \in G : f_\theta(g^{-1}x; z) = f_\theta(x; gz)\,.$

- The **steerability** property imposes a **bi-invariance** constraint on the neural field:

$$f_\theta(g^{-1}x; z) = f_\theta(x; gz) \quad \textit{iff*} \quad f_\theta(gx; gz) = f_\theta(x; z)$$

Bi-invariance intuition:

If I rotate my latents but rotate my domain back, nothing happens to the function modelled by the ENF; the transformations cancel.

Proof:

Assuming steerability

$\rightarrow f_\theta(gx; gz) = f_\theta(x; g^{-1}gz) = f_\theta(x; z)$

Assuming bi-invariance

$\leftarrow f_\theta(g^{-1}x; z) = f_\theta(gg^{-1}x; gz) = f_\theta(x; gz)$

Published as a conference paper at ICLR 2024

FAST, EXPRESSIVE SE(n) EQUIVARIANT NETWORKS THROUGH WEIGHT-SHARING IN POSITION-ORIENTATION SPACE

Erik J. Bekkers[1*], **Sharvaree Vadgama**[1*]
Rob D. Hesselink[1], **Putri A. van der Linden**[1], **David W. Romero**[2,3]
[1] University of Amsterdam [2] Vrije Universiteit Amsterdam [3] NVIDIA Research
[*] *Equal contribution.* *Code available at* https://github.com/ebekkers/ponita

ABSTRACT

Based on the theory of homogeneous spaces we derive *geometrically optimal edge attributes* to be used within the flexible message-passing framework. We formalize the notion of weight sharing in convolutional networks as the sharing of message functions over point-pairs that should be treated equally. We define equivalence classes of point-pairs that are identical up to a transformation in the group and derive attributes that uniquely identify these classes. Weight sharing is then obtained by conditioning message functions on these attributes. As an application of the theory, we develop an efficient equivariant group convolutional network for

Equivariant Neural Fields

- It is sufficient to make $f_\theta(x; z)$ bi-invariant; any bi-invariant function of x and z works.

$$f_\theta(g^{-1}x; z) = f_\theta(x; gz) \quad \textit{iff*} \quad f_\theta(gx; gz) = f_\theta(x; z)$$

- We look towards an equivariant formulation of *transformers,* obtaining outputs for x, z by:

$$f_\theta(x; z) := \text{cross\_attention}(x, z)$$

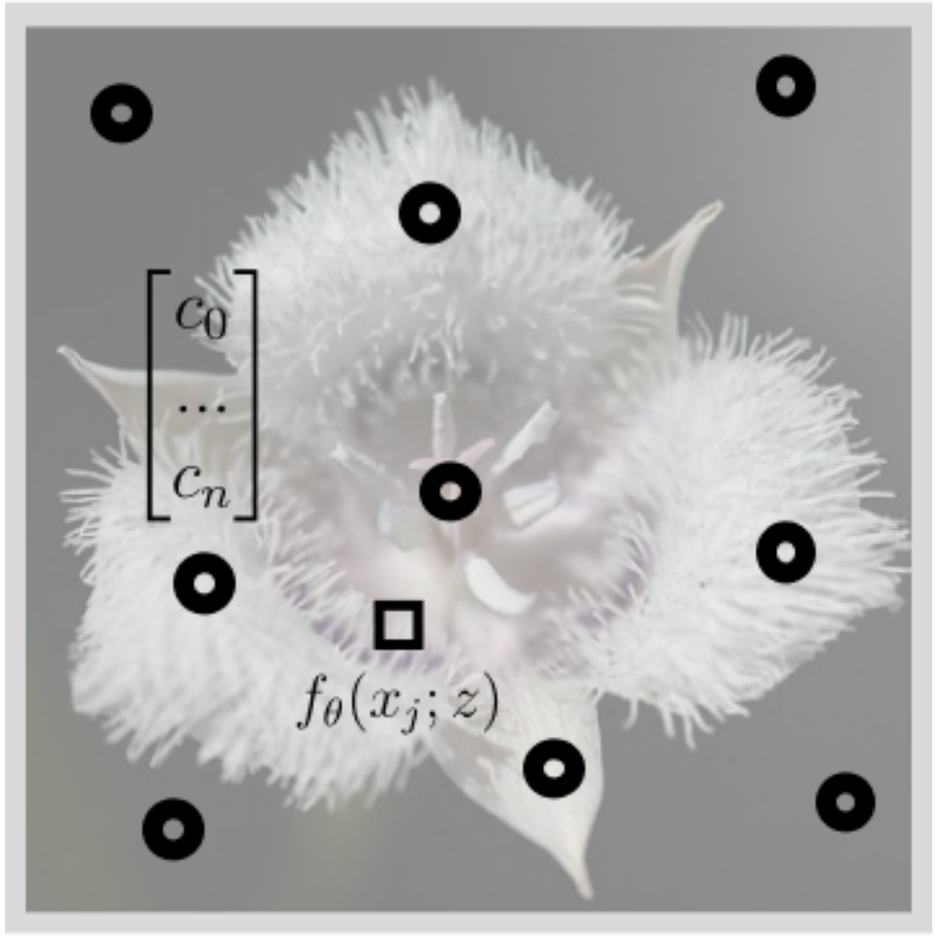
Now a set/
point cloud!

Equivariant Neural Fields

Attention operator in an ENF

Equivariant Neural Fields

Attention operator in an ENF | *inputs*



Latents z

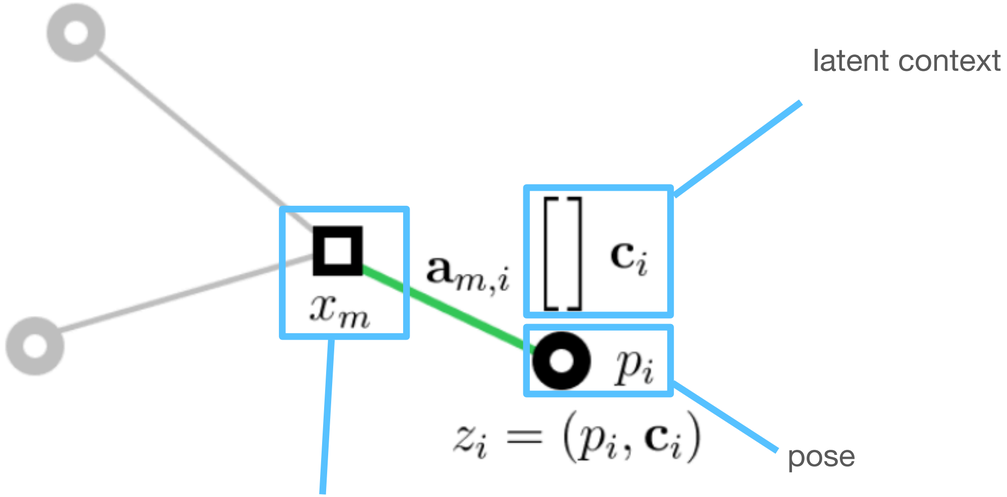
```python
def initialize_latents(
    batch_size: int,
    num_latents: int,
    latent_dim: int,
    data_dim: int,
    key: Any,
) -> Tuple[jnp.ndarray, jnp.ndarray]:
    """Initialize the latent variables."""
    key, subkey_0, subkey_1 = jax.random.split(key, 3)

    # Initialize positions of the latents [-1, 1]
    pose = jax.random.uniform(subkey_0, (batch_size, num_latents, data_dim)) * 2 - 1

    # Initialize context vectors of the latents
    context = jnp.zeros((batch_size, num_latents, latent_dim))

    return pose, context
```

Coordinates x

```python
# Coordinate grid, in [-1, 1] x [-1, 1]
# 10 points in each dimension, [num_points, 2]
x = jnp.stack(
    jnp.meshgrid(
        jnp.linspace(-1, 1, 10),
        jnp.linspace(-1, 1, 10),
    ),
    axis=-1,
).reshape(-1, 2)
```

Equivariant Neural Fields

Attention operator in an ENF | *calculation*



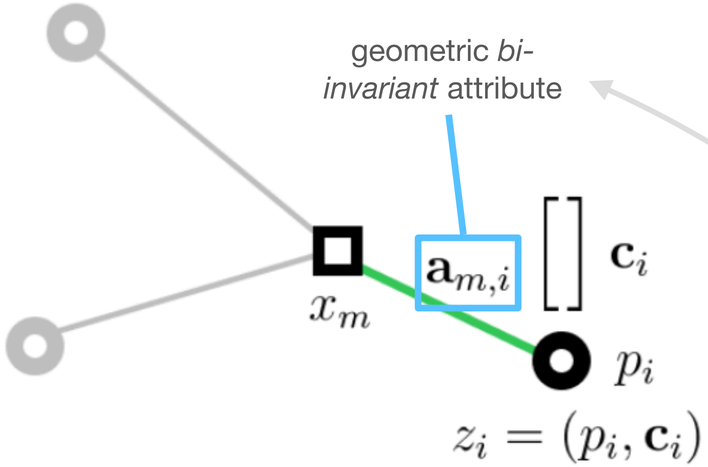
Translational bi-invariant $\mathbf{a}_{m,i}$

```
# Calculate the bi-invariants between the input
# coordinates and the latents.
# x: [num_points, 2]
# p: [batch_size, num_latents, 2]
bi_inv = x[None, :, None, :] - p[:, None, :, :]
```

Step 1: calculate bi-invariants for all *pose-coordinate* pairs p_i, x_m

Equivariant Neural Fields

Attention operator in an ENF | *calculation*



Translational bi-invariant $\mathbf{a}_{m,i}$

```
# Calculate the bi-invariants between the input
# coordinates and the latents.
# x: [num_points, 2]
# p: [batch_size, num_latents, 2]
bi_inv = x[None, :, None, :] - p[:, None, :, :]
```

Step 1: calculate bi-invariants for all *pose-coordinate* pairs p_i, x_m


Published as a conference paper at ICLR 2024

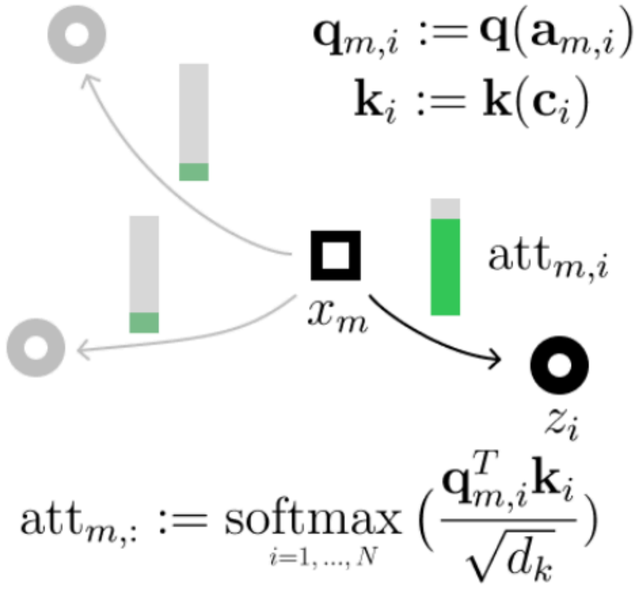
FAST, EXPRESSIVE SE(n) EQUIVARIANT NETWORKS THROUGH WEIGHT-SHARING IN POSITION-ORIENTATION SPACE

Erik J. Bekkers[1]*, **Sharvaree Vadgama**[1]*
Rob D. Hesselink[1], **Putri A. van der Linden**[1], **David W. Romero**[2,3]
[1] University of Amsterdam [2] Vrije Universiteit Amsterdam [3] NVIDIA Research

*Equal contribution. *Code available at https://github.com/ebekkers/ponita*

Equivariant Neural Fields

Attention operator in an ENF | *calculation*



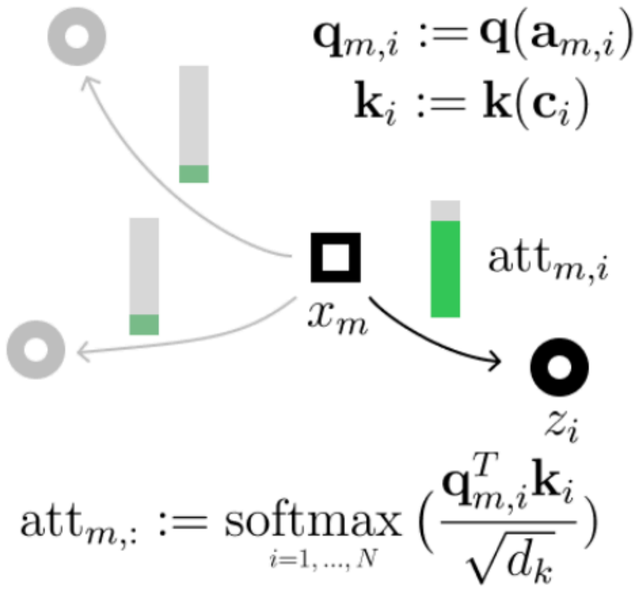
Step 2: calculate attention coefficients for all latent-coordinate pairs z_i, x_m

Calculating attention coefficients

```
# Calculate the query transform for the bi-invariants.
#   q: [batch_size, num_coords, num_latents, num_hidden]
q = pos_emb_q(bi_inv)

# Calculate the key and value transforms for the bi-invariants.
#   k: [batch_size, num_latents, num_hidden]
k = W_k @ c

# Calculate the value transform for the bi-invariants.
# Dot product between query and key transforms.
#   att_logits: [batch_size, num_coords, num_latents, 1]
att_logits = (q * k[:, None, :, :]).sum(axis=-1, keepdims=True) / k.shape[-1]

# Calculate the attention weights by softmaxing over the num_latents dimension.
#   att: [batch_size, num_coords, num_latents, 1]
att = jax.nn.softmax(att_logits, axis=2)
```

Equivariant Neural Fields

Attention operator in an ENF | *calculation*



Step 2: calculate attention coefficients for all latent-coordinate pairs z_i, x_m

Attention coefficient intuition:

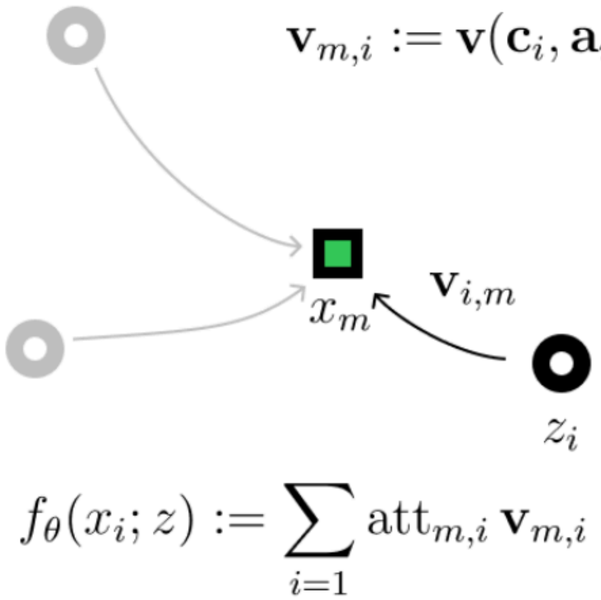
"How much does latent z_i influence the output at x_m?"

Calculating attention coefficients

```
# Calculate the query transform for the bi-invariants.
#   q: [batch_size, num_coords, num_latents, num_hidden]
q = pos_emb_q(bi_inv)

# Calculate the key transform for the latent context vectors.
#   k: [batch_size, num_latents, num_hidden]
k = W_k @ c

# Calculate the attention logits for combinations of latent and bi-invariant.
# Dot product between query and key transforms.
#   att_logits: [batch_size, num_coords, num_latents, 1]
att_logits = (q * k[:, None, :, :]).sum(axis=-1, keepdims=True) / k.shape[-1]

# Calculate the attention weights by softmaxing over the num_latents dimension.
#   att: [batch_size, num_coords, num_latents, 1]
att = jax.nn.softmax(att_logits, axis=2)
```

Equivariant Neural Fields

Attention operator in an ENF | *calculation*



Step 3: calculate contribution for coefficients for all latent-coordinate pairs z_i, x_m

Value function intuition:

"What does latent z_i *contribute to the output at* x_m*?"*

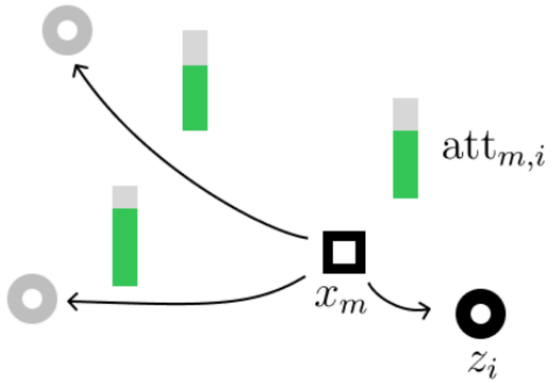
Calculating values and applying attention

```
# Calculate modulations of the value transform from the bi-invariants.
b_v, g_v = jnp.split(pos_emb_v(bi_inv), 2, axis=-1)

# Apply the modulations to the value transform.
# W_v @ c: [batch_size, num_latents, num_hidden]
# b_v: [batch_size, num_coords, num_latents, num_hidden]
# g_v: [batch_size, num_coords, num_latents, num_hidden]
# v: [batch_size, num_coords, num_latents, num_hidden]
v = (W_v @ c) * (1 + b_v) + g_v

# Attend the values to the queries and keys.
y = (att * v).sum(axis=2)
```

Equivariant Neural Fields

Attention operator in an ENF | *calculation*

$$\text{att}_{m,:} := \text{softmax}\left(\frac{\mathbf{q}_{m,i}^T \mathbf{k}_i}{\sqrt{d_k}}\right)$$



This formulation allows for
non-local activation of latents

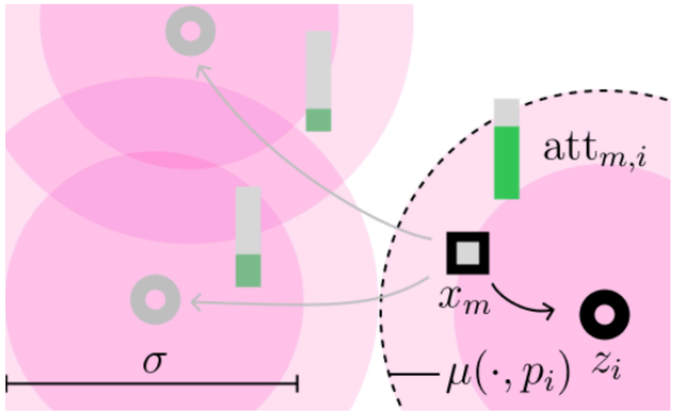
"Global" attention calculation

```
# Calculate the value transform for the bi-invariants.
# Dot product between query and key transforms.
#   att_logits: [batch_size, num_coords, num_latents, 1]
att_logits = (q * k[:, None, :, :]).sum(axis=-1, keepdims=True) / k.shape[-1]
```

Equivariant Neural Fields

Attention operator in an ENF | *calculation*

$$\text{att}_{m,:} := \text{softmax}\Big(\frac{\mathbf{q}_{m,i}^T \mathbf{k}_i}{\sqrt{d_k}} + \mu_\sigma(x_m, p_i)\Big)$$



Explicitly enforce locality with
Gaussian window on attention logits

Initialising windows parameters

```python
def initialize_latents(
    batch_size: int,
    num_latents: int,
    latent_dim: int,
    data_dim: int,
    key: Any,
) -> Tuple[jnp.ndarray, jnp.ndarray]:
    """Initialize the latent variables."""
    key, subkey_0 = jax.random.split(key, 3)  ⇥ tab

    # Initialize positions of the latents [-1, 1]
    pose = jax.random.uniform(subkey_0, (batch_size, num_latents, data_dim)) * 2 - 1

    # Initialize context vectors of the latents
    context = jnp.zeros((batch_size, num_latents, latent_dim))

    # Initialize window size for the gaussian window
    window_size = jnp.ones((batch_size, num_latents, 1))

    return pose, context, window_size
```

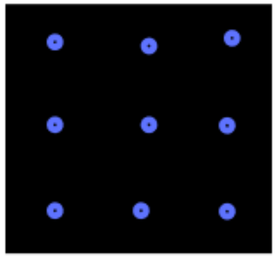
Weighting the attention logits by distance

```python
# Calculate distances based on distance to the latent
zx_mag = jnp.linalg.norm(x[None, :, None, :] - p[:, None, :, :], axis=-1)
att_logits = (q * k).sum(axis=-1, keepdims=True) - ((1 / g ** 2) * zx_mag)[..., None, :]
```

Equivariant Neural Fields

Obtaining z with meta learning | *initialisation*

field $f(\cdot)$



init z

Optimizing z

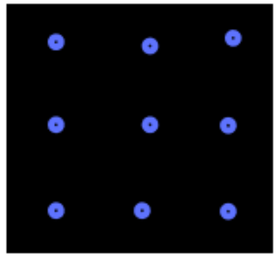
```python
@jax.jit
def recon_inner_loop(enf_params, coords, img, key):
    z = initialize_latents(
        batch_size=config.train.batch_size,
        num_latents=config.recon_enf.num_latents,
        latent_dim=config.recon_enf.latent_dim,
        data_dim=config.recon_enf.num_in,
        bi_invariant_cls=TranslationBI,
        key=key,
        noise_scale=config.train.noise_scale,
    )

    def mse_loss(z):
        out = recon_enf.apply(enf_params, coords, *z)
        return jnp.sum(jnp.mean((out - img) ** 2, axis=(1, 2)), axis=0)

    def inner_step(z):
        grads = jax.value_and_grad(mse_loss)(z)
        # Gradient descent update
        z = jax.tree.map(lambda z, grad, lr: z - lr * grad, z, grads, config.optim.inner_lr)
        return z

    # Perform inner loop optimization
    for _ in range(config.optim.inner_steps):
        z = inner_step(z)

    # Stop gradient if first order MAML
    if config.optim.first_order_maml:
        z = jax.lax.stop_gradient(z)
    return mse_loss(z), z
```

Equivariant Neural Fields

Obtaining z with meta learning | *optimization loop*

field $f(\cdot)$

$f_{\theta_{\text{recon}}}(\cdot; z)$



Optimizing z

```python
@jax.jit
def recon_inner_loop(enf_params, coords, img, key):
    z = initialize_latents(
        batch_size=config.train.batch_size,
        num_latents=config.recon_enf.num_latents,
        latent_dim=config.recon_enf.latent_dim,
        data_dim=config.recon_enf.num_in,
        bi_invariant_cls=TranslationBI,
        key=key,
        noise_scale=config.train.noise_scale,
    )

    def mse_loss(z):
        out = recon_enf.apply(enf_params, coords, *z)
        return jnp.sum(jnp.mean((out - img) ** 2, axis=(1, 2)), axis=0)

    def inner_step(z):
        grads = jax.value_and_grad(mse_loss)(z)
        # Gradient descent update
        z = jax.tree.map(lambda z, grad, lr: z - lr * grad, z, grads, config.optim.inner_lr)
        return z

    # Perform inner loop optimization
    for _ in range(config.optim.inner_steps):
        z = inner_step(z)

    # Stop gradient if first order MAML
    if config.optim.first_order_maml:
        z = jax.lax.stop_gradient(z)
    return mse_loss(z), z
```

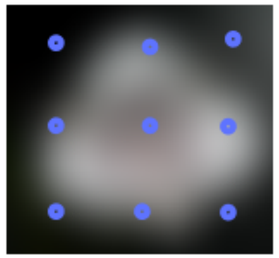
Equivariant Neural Fields

Obtaining z with meta learning | *optimization loop*

field $f(\cdot)$

$f_{\theta_{\text{recon}}}(\cdot; z)$

$\nabla_z \mathcal{L}$



step 0

Optimizing z

```python
@jax.jit
def recon_inner_loop(enf_params, coords, img, key):
    z = initialize_latents(
        batch_size=config.train.batch_size,
        num_latents=config.recon_enf.num_latents,
        latent_dim=config.recon_enf.latent_dim,
        data_dim=config.recon_enf.num_in,
        bi_invariant_cls=TranslationBI,
        key=key,
        noise_scale=config.train.noise_scale,
    )

    def mse_loss(z):
        out = recon_enf.apply(enf_params, coords, *z)
        return jnp.sum(jnp.mean((out - img) ** 2, axis=(1, 2)), axis=0)

    def inner_step(z):
        grads = jax.value_and_grad(mse_loss)(z)
        # Gradient descent update
        z = jax.tree.map(lambda z, grad, lr: z - lr * grad, z, grads, config.optim.inner_lr)
        return z

    # Perform inner loop optimization
    for _ in range(config.optim.inner_steps):
        z = inner_step(z)

    # Stop gradient if first order MAML
    if config.optim.first_order_maml:
        z = jax.lax.stop_gradient(z)
    return mse_loss(z), z
```

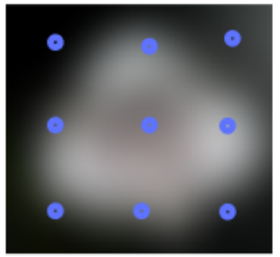
Equivariant Neural Fields

Obtaining z with meta learning | *optimization loop*

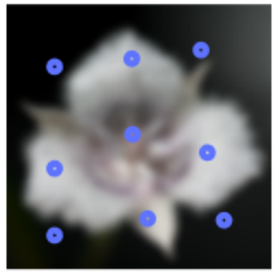
field $f(\cdot)$

$f_{\theta_{\text{recon}}}(\cdot; z)$

$\nabla_z \mathcal{L}$



step 0



step 1

Optimizing z

```python
@jax.jit
def recon_inner_loop(enf_params, coords, img, key):
    z = initialize_latents(
        batch_size=config.train.batch_size,
        num_latents=config.recon_enf.num_latents,
        latent_dim=config.recon_enf.latent_dim,
        data_dim=config.recon_enf.num_in,
        bi_invariant_cls=TranslationBI,
        key=key,
        noise_scale=config.train.noise_scale,
    )

    def mse_loss(z):
        out = recon_enf.apply(enf_params, coords, *z)
        return jnp.sum(jnp.mean((out - img) ** 2, axis=(1, 2)), axis=0)

    def inner_step(z):
        grads = jax.value_and_grad(mse_loss)(z)
        # Gradient descent update
        z = jax.tree.map(lambda z, grad, lr: z - lr * grad, z, grads, config.optim.inner_lr)
        return z

    # Perform inner loop optimization
    for _ in range(config.optim.inner_steps):
        z = inner_step(z)

    # Stop gradient if first order MAML
    if config.optim.first_order_maml:
        z = jax.lax.stop_gradient(z)
    return mse_loss(z), z
```

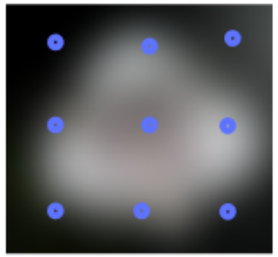
Equivariant Neural Fields

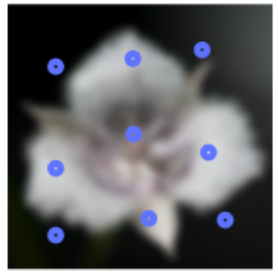
Obtaining z with meta learning | *optimization loop*

field $f(\cdot)$

$f_{\theta_{\text{recon}}}(\cdot; z)$

$\nabla_z \mathcal{L}$


step 0


step 1


step 2

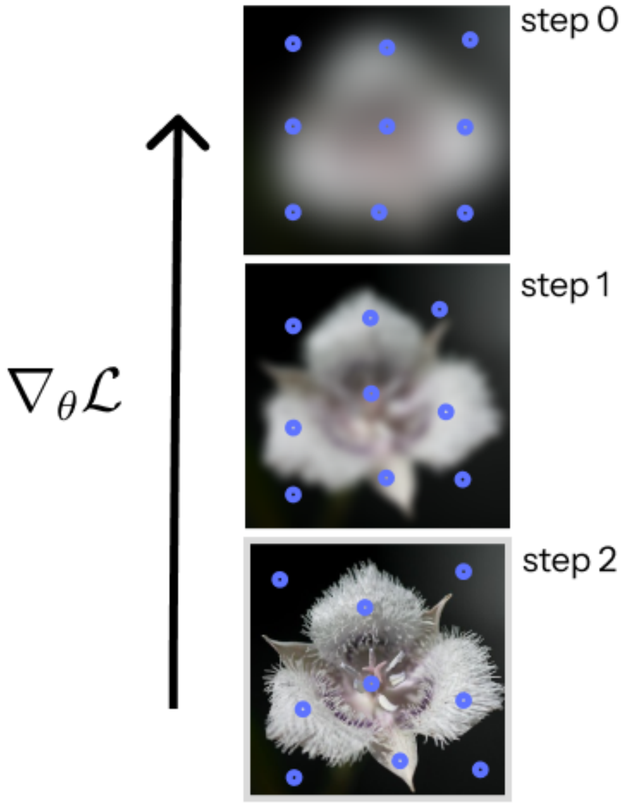
Optimizing z

```python
@jax.jit
def recon_inner_loop(enf_params, coords, img, key):
    z = initialize_latents(
        batch_size=config.train.batch_size,
        num_latents=config.recon_enf.num_latents,
        latent_dim=config.recon_enf.latent_dim,
        data_dim=config.recon_enf.num_in,
        bi_invariant_cls=TranslationBI,
        key=key,
        noise_scale=config.train.noise_scale,
    )

    def mse_loss(z):
        out = recon_enf.apply(enf_params, coords, *z)
        return jnp.sum(jnp.mean((out - img) ** 2, axis=(1, 2)), axis=0)

    def inner_step(z):
        grads = jax.value_and_grad(mse_loss)(z)
        # Gradient descent update
        z = jax.tree.map(lambda z, grad, lr: z - lr * grad, z, grads, config.optim.inner_lr)
        return z

    # Perform inner loop optimization
    for _ in range(config.optim.inner_steps):
        z = inner_step(z)

    # Stop gradient if first order MAML
    if config.optim.first_order_maml:
        z = jax.lax.stop_gradient(z)
    return mse_loss(z), z
```

Equivariant Neural Fields

Obtaining z with meta learning | *outer step*

field $f(\cdot)$

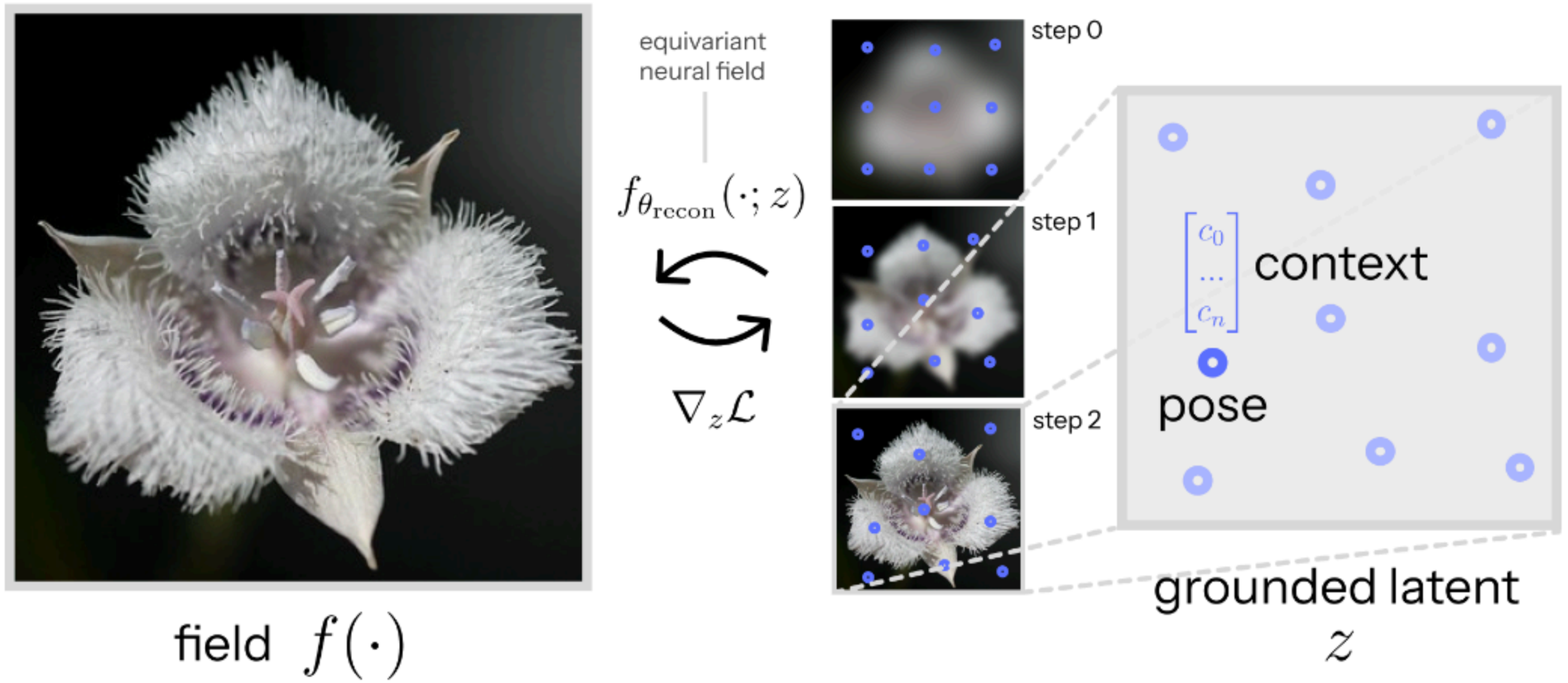


Optimizing θ

```
@jax.jit
def recon_outer_step(coords, img, enf_params, enf_opt_state, key):
    # Perform inner loop optimization
    key, subkey = jax.random.split(key)
    (loss, z), grads = jax.value_and_grad(
        recon_inner_loop, has_aux=True)(enf_params, coords, img, key)

    # Update the ENF backbone
    enf_grads, enf_opt_state = enf_opt.update(grads, enf_opt_state)
    enf_params = optax.apply_updates(enf_params, enf_grads)

    # Sample new key
    return (loss, z), enf_params, enf_opt_state, subkey
```

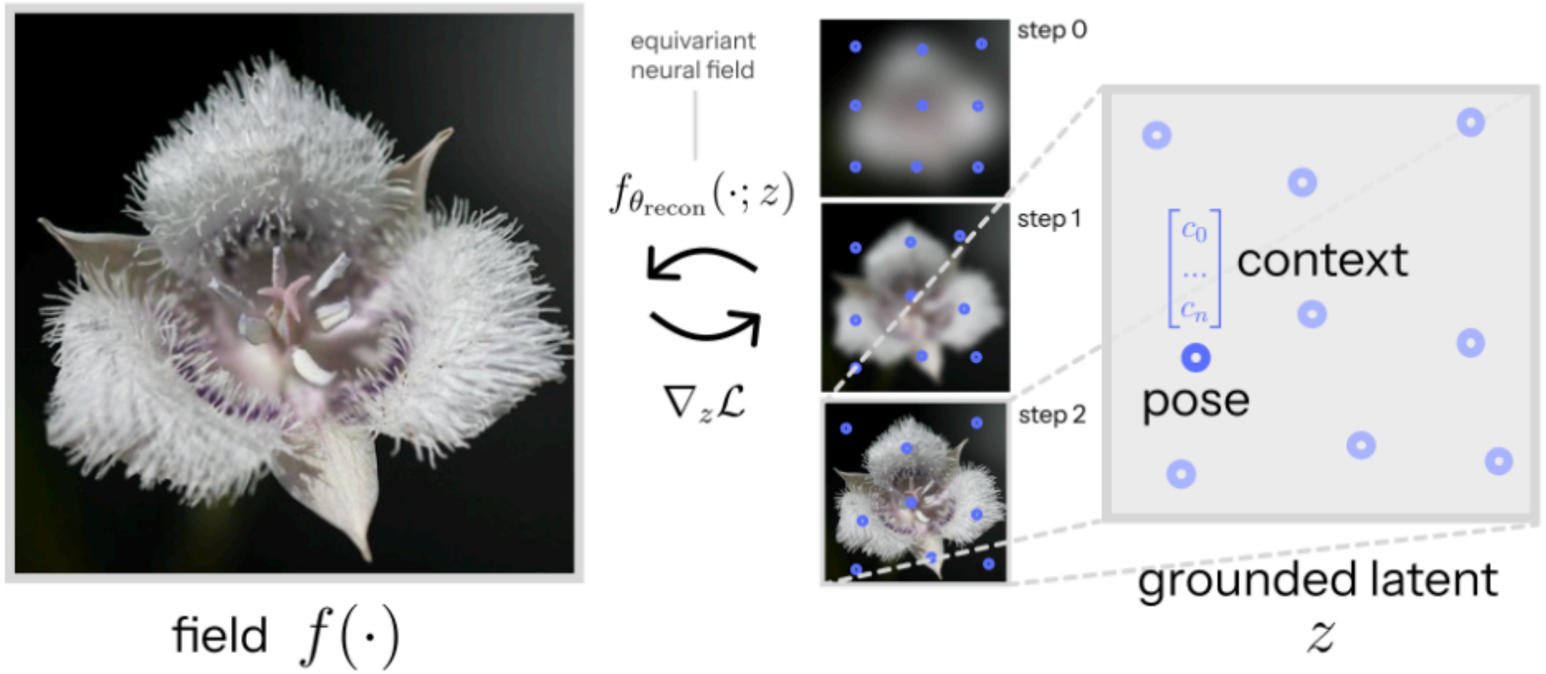
Equivariant Neural Fields

Obtaining z with meta learning

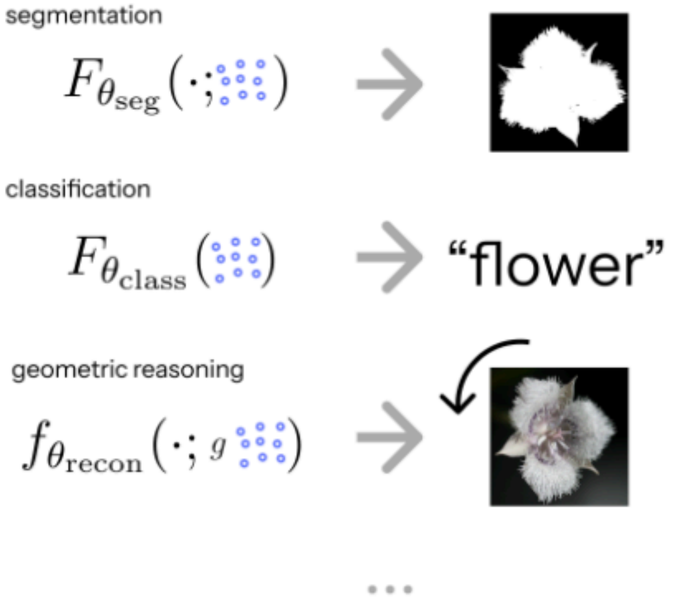


ENF Summary

1. optimize z to reconstruct $f(\cdot)$



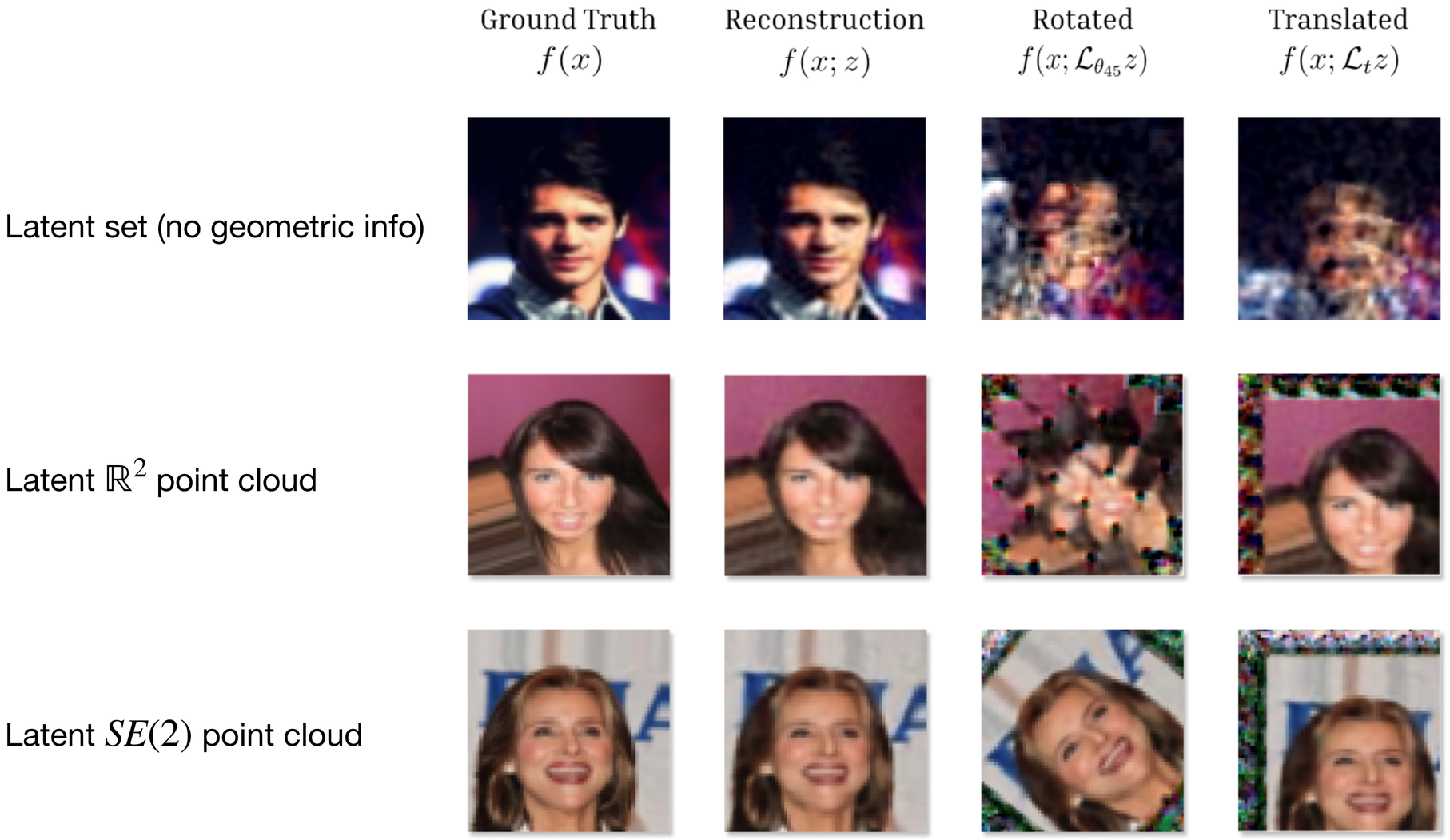
2. use z downstream



Equivariant Neural Fields

Experiments:
Capacity to represent signals

Comparison of representation capacity/ quality (1)



Comparison of representation capacity/ quality (2)

$$\begin{pmatrix} 0.63 \\ -1.80 \\ -0.74 \\ 0.23 \\ 0.74 \\ \vdots \end{pmatrix}$$

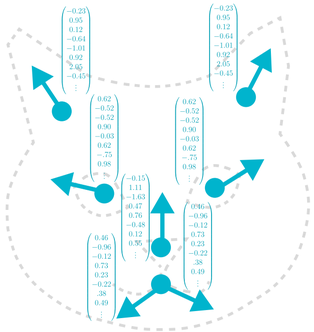


Table 1: Test-set reconstruction PSNR (db↑) on CIFAR10, CelebA64x64, ImageNet1k, test accuracy (%↑) on CIFAR10.

	CIFAR10	CELEBA	IMAGENET
TASK	RECON.	RECON.	RECON.
Functa	38.1	28.0	7.2
ENF $\mathbf{a}^{\emptyset}$	36.5	30.6	24.7
ENF $\mathbf{a}^{\mathbb{R}^2}$	**42.2**	**34.6**	**27.5**
ENF $\mathbf{a}^{\mathrm{SE}(2)}$	41.6	32.9	26.8

Table 2: Test reconstruction (IoU↑) on ShapeNet16 and ShapeNet55 and test classification accuracy (%↑) on ShapeNet16.

MODALITY	SHAPENET16 VOXEL (OCC)	SHAPENET55 P. CLOUD (SDF)
TASK	RECON.	RECON.
Functa	92.1	25.7
ENF $\mathbf{a}^{\emptyset}$	90.7	72.3
ENF $\mathbf{a}^{\mathbb{R}^3}$	**92.9**	**73.2**

Relevant ENF properties:

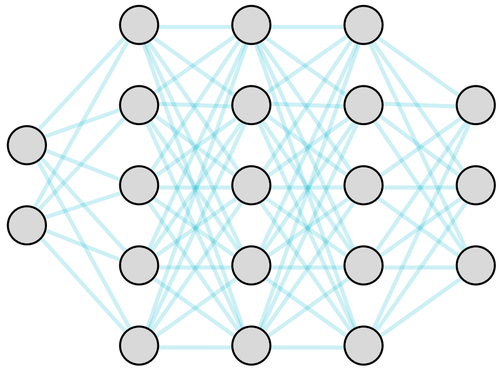
- Weight sharing
- Locality (localized cross-attention)

Equivariant Neural Fields

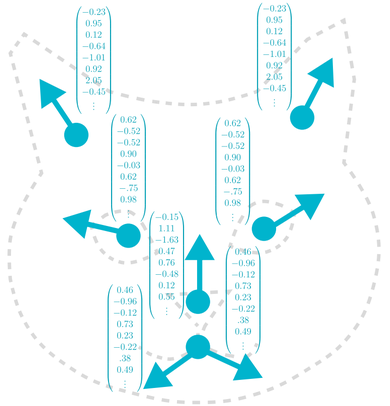
Experiments:
Geometry-grounded representation learning

Down-stream tasks (classification)

Representation:



$$\begin{pmatrix} 0.63 \\ -1.80 \\ -0.74 \\ 0.23 \\ 0.74 \\ \vdots \end{pmatrix}$$



Classifier:

Graph neural network

MLP

Equivariant graph NN / Transformer

“Owl”

“Dog”

“Cat”

Geometry-grounded: semantically meaningful latents

$$\begin{pmatrix} 0.63 \\ -1.80 \\ -0.74 \\ 0.23 \\ 0.74 \\ \vdots \end{pmatrix}$$

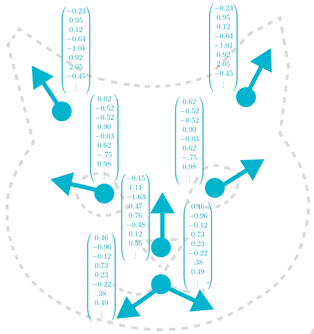


Table 1: Test-set reconstruction PSNR (db↑) on CIFAR10, CelebA64x64, ImageNet1k, test accuracy (%↑) on CIFAR10.

	CIFAR10
TASK	CLASS.
Functa	68.3
ENF $\mathbf{a}^{\emptyset}$	68.7
ENF $\mathbf{a}^{\mathbb{R}^2}$	**82.1**
ENF $\mathbf{a}^{\mathrm{SE}(2)}$	81.5

+13.8%

Table 2: Test reconstruction (IoU↑) on ShapeNet16 and ShapeNet55 and test classification accuracy (%↑) on ShapeNet16.

MODALITY	SHAPENET16 VOXEL (OCC)
TASK	CLASS.
Functa	90.3
ENF $\mathbf{a}^{\emptyset}$	96.4
ENF $\mathbf{a}^{\mathbb{R}^3}$	**96.6**

Unique editing features of ENFs

Geometry-grounded: Multi-modal segmentation

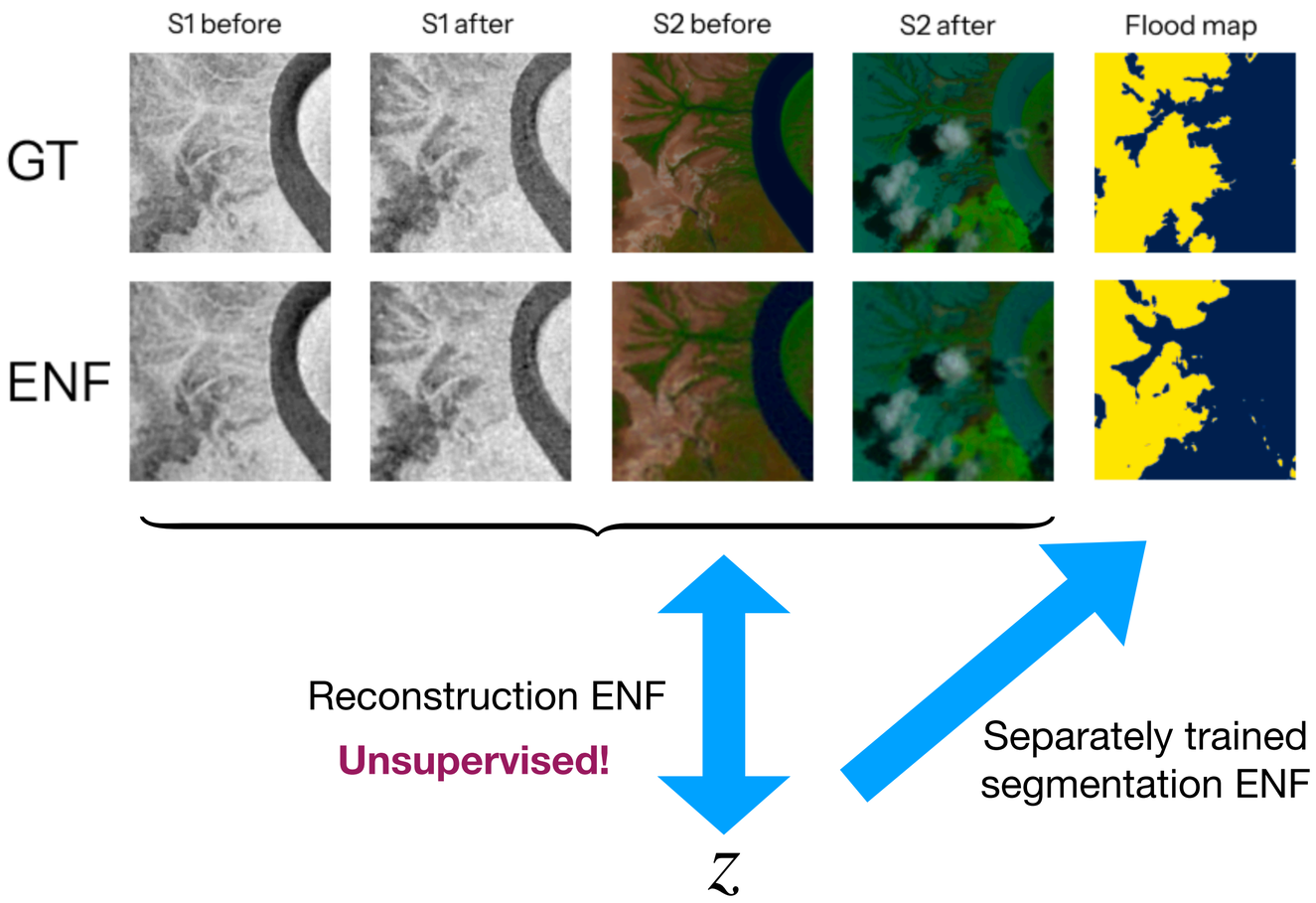


Table 4: Test IOU (↑) for flood map segmentation on OMBRIA, for different observation rates.

Model	PSNR (↑)	IoU (↑)
	100% of f_{IN} observed	
OmbriaNet	N.A.	72.36
Functa	16.77	42.75
ENF	**31.65**	**74.00**
	50% of f_{IN} observed	
OmbriaNet	N.A.	27.02
Functa	16.71	42.74
ENF	**31.37**	**73.65**
	10% of f_{IN} observed	
OmbriaNet	N.A.	0.0
Functa	16.77	42.92
ENF	**24.87**	**71.58**

Sparse/corrupted data

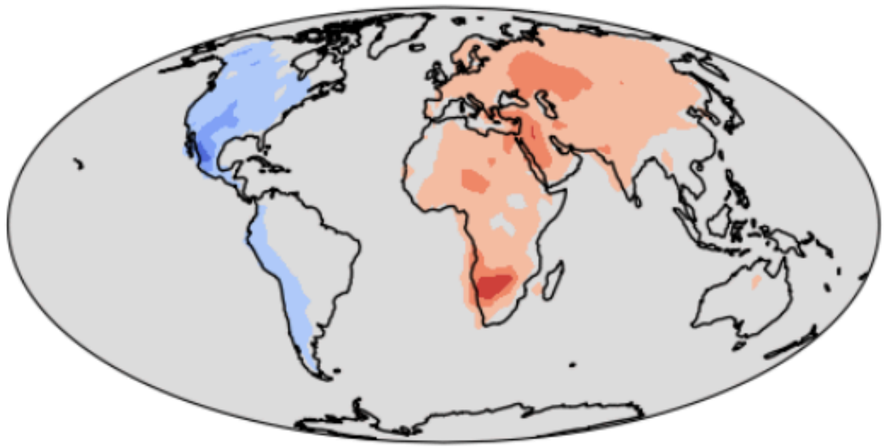
Table 5: Test IOU (↑) zero-shot resolution transfer on OMBRIA. $f_{\theta_{\text{recon}}}, f_{\theta_{\text{seg}}}$ were trained on 128×128 resolution.

Model	PSNR (↑)	IoU (↑)
	256×256 test resolution	
Functa	16.72	37.14
ENF	**28.61**	**72.92**
	128×128 test resolution	
Functa	16.71	35.48
ENF	**29.31**	**73.21**
	64×64 test resolution	
Functa	16.58	36.90
ENF	**33.31**	**72.50**

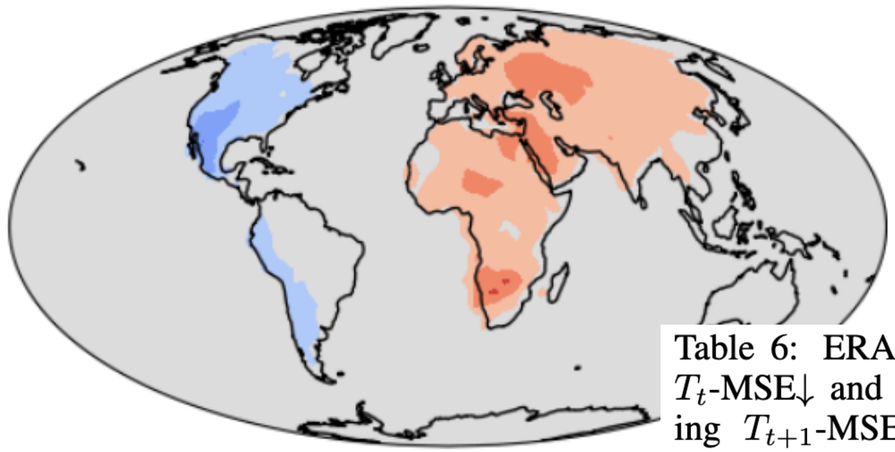
Zero-shot resolution transfer

Geometry-grounded: Forecasting

Ground truth $\Delta T = T_{t+1}(\cdot) - T_t(\cdot)$



Predicted $\Delta \hat{T} = f_\theta(\cdot, z_{t+1}) - f_\theta(\cdot, z_t)$



Spherical data!

Table 6: ERA5 reconstruction T_t-MSE↓ and 1-hour forecasting T_{t+1}-MSE↓. **MSE between ground truth observations at T_t and T_{t+1}.*

	T_t-MSE↓	T_{t+1}-MSE↓
Identity*	-	2.42E-05
Functa	5.75E-05	3.45E-03
ENF	**8.04E-06**	**9.44E-06**

Geometry-grounded: Does it scale?

Preliminary!

Original

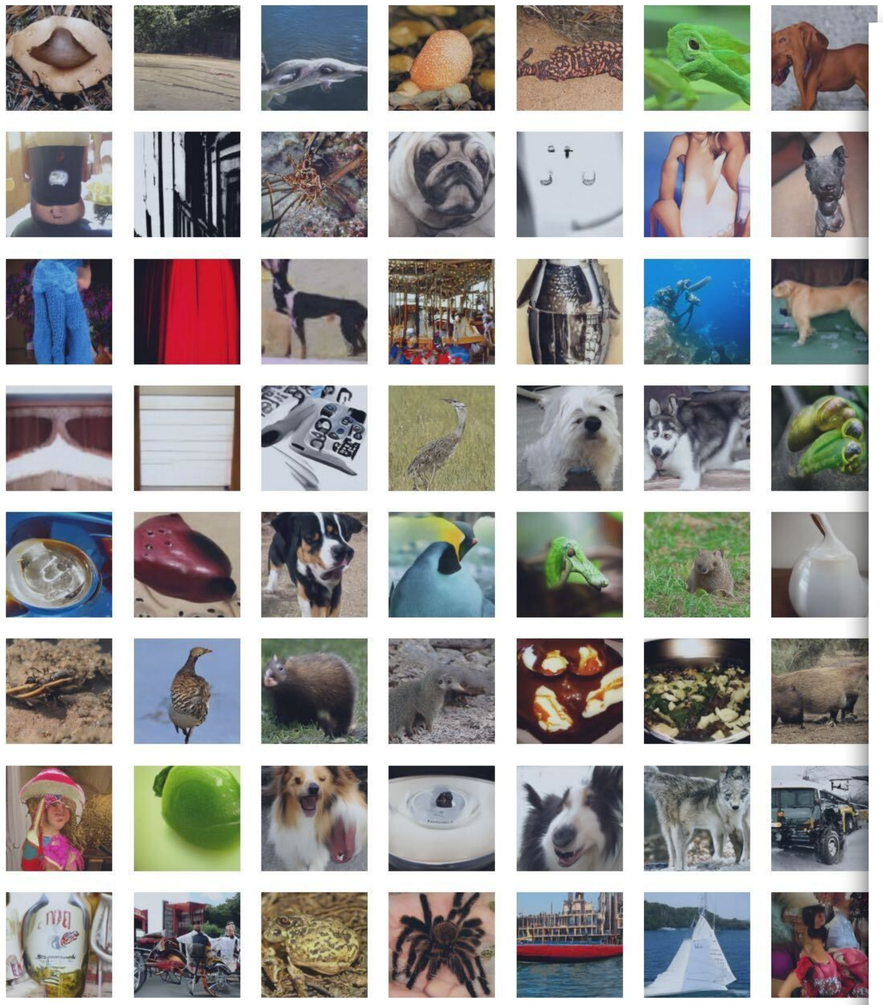
Reconstructed

~39 PSNR
ImageNet1K 256x256
1 SGD Step

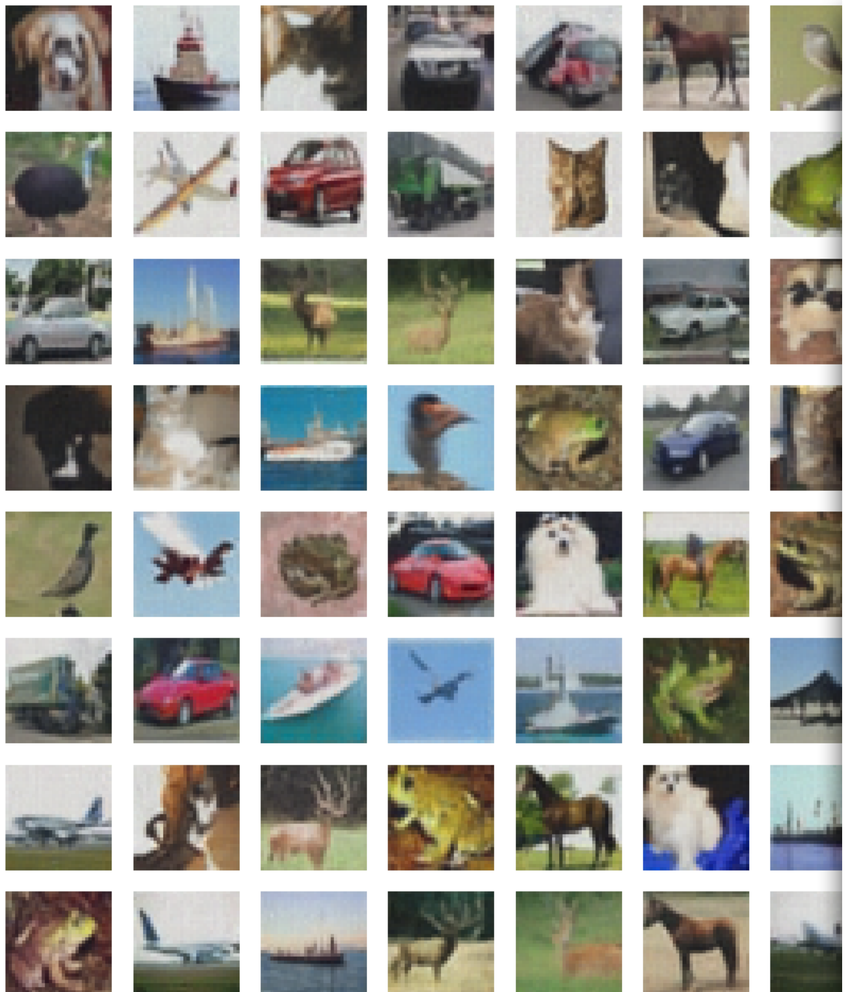
Geometry-grounded: Generative modeling

Preliminary!

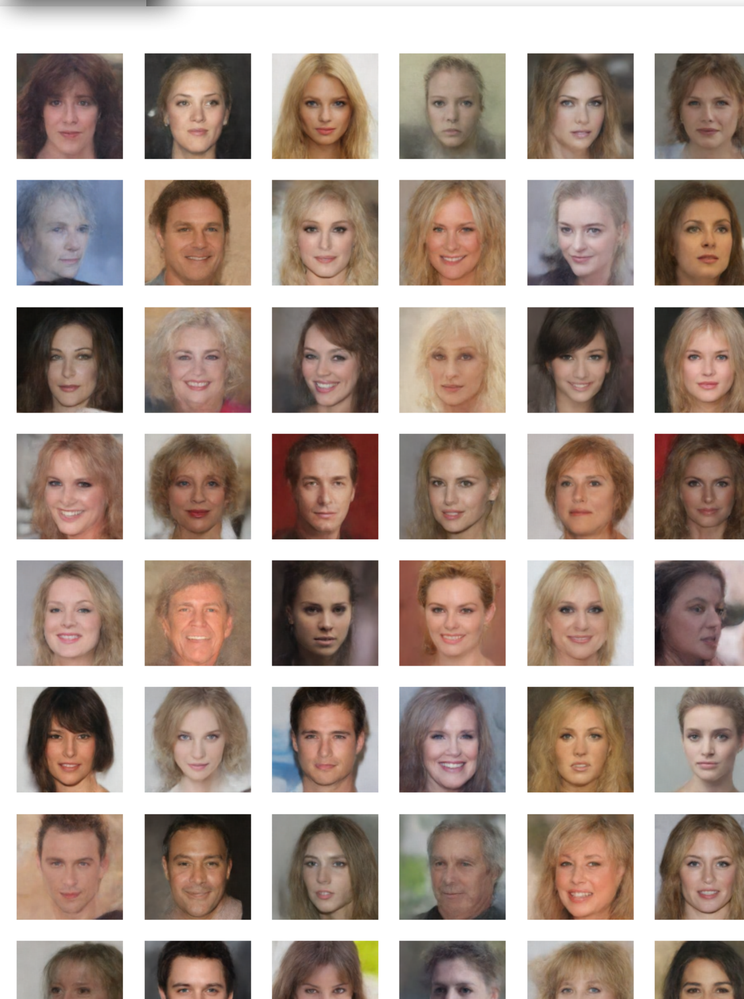
ImageNet1K



CIFAR



Celeb-A



NeurIPS ‘24

Space-Time Continuous PDE Forecasting using Equivariant Neural Fields

David M. Knigge[*,1], **David R. Wessels**[*,1], **Riccardo Valperga**[1], **Samuele Papa**[1], **Jan-Jakob Sonke**[2], **Efstratios Gavves** [†,1], **Erik J. Bekkers** [†,1]
[1]University of Amsterdam [2] Netherlands Cancer Institute
d.m.knigge@uva.nl, d.r.wessels@uva.nl

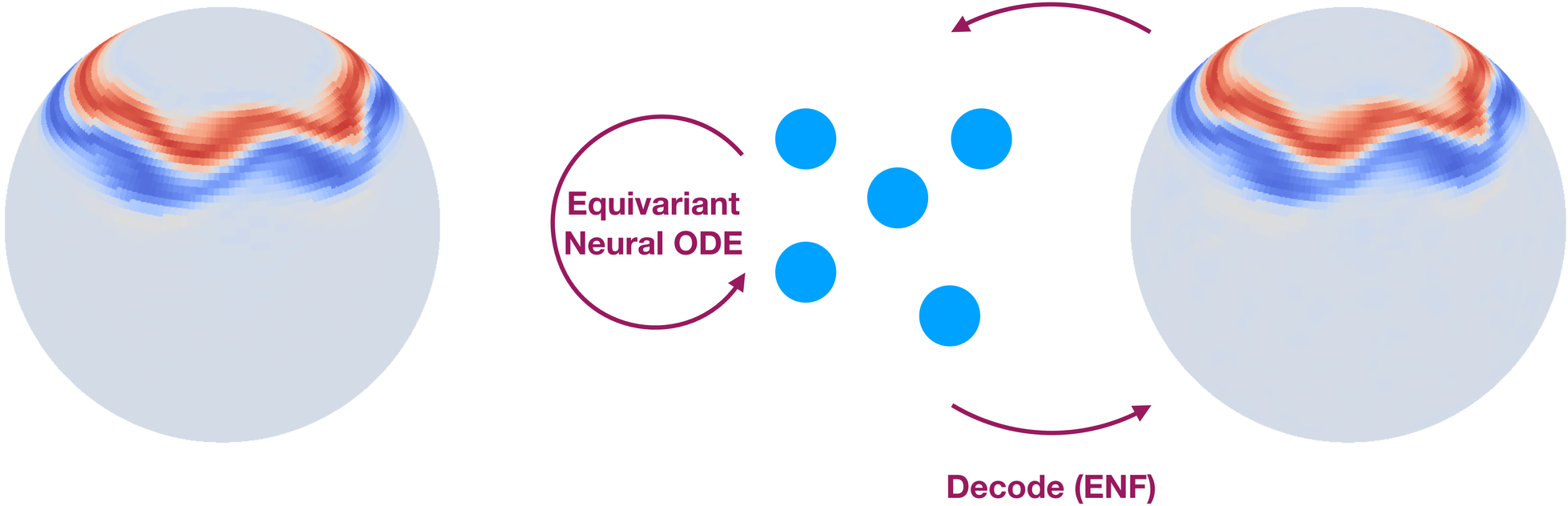
Abstract

Recently, Conditional Neural Fields (NeFs) have emerged as a powerful modelling paradigm for PDEs, by learning solutions as flows in the latent space of the Conditional NeF. Although benefiting from favourable properties of NeFs such as grid-agnosticity and space-time-continuous dynamics modelling, this approach limits the ability to impose known constraints of the PDE on the solutions – e.g. symmetries or boundary conditions – in favour of modelling flexibility. Instead, we propose a space-time continuous NeF-based solving framework that - by preserving geometric information in the latent space - respects known symmetries of the PDE. We show that modelling solutions as flows of pointclouds over the group of interest G improves generalization and data-efficiency. We validated that our framework readily generalizes to unseen spatial and temporal locations, as well as geometric transformations of the initial conditions - where other NeF-based PDE forecasting methods fail - and improve over baselines in a number of challenging geometries.

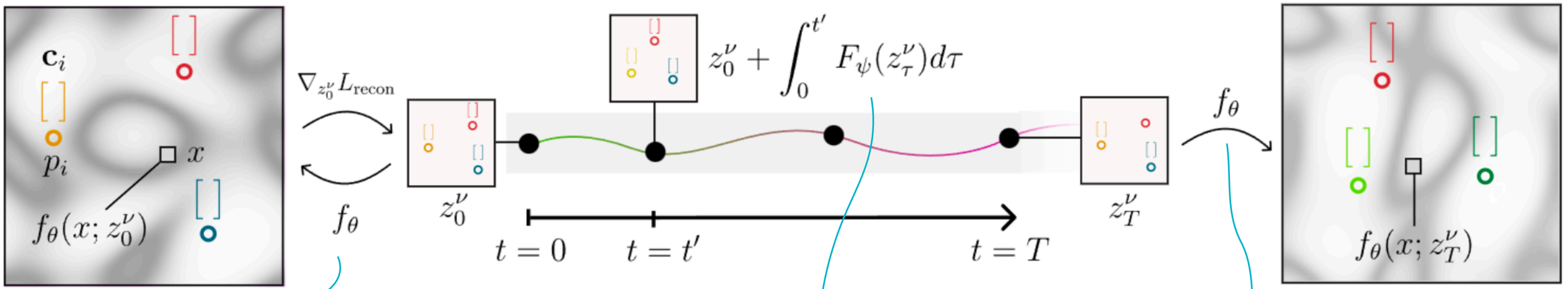
Equivariant PDE forecasting in latent space!

Ground truth

ENF-Based PDE Forecasting

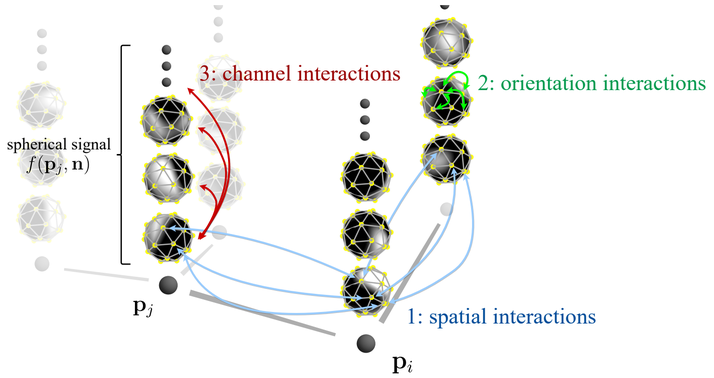


Equivariant PDE forecasting in latent space!



Equivariant encoding of state using the ENFs (using meta-learning/ auto-decoding)

Equivariant Neural ODE via Ponita (ICLR'24)



Equivariant decoding: sample ENF

Equivariant PDE forecasting in latent space!

$$\min_{\theta,\psi,z_\tau} \quad \mathbb{E}_{\nu\in D, x\in\mathcal{X}, t\in[\![T]\!]} \left\|\nu_t(x) - f_\theta(x; z_t^\nu)\right\|_2^2, \quad \text{where} \quad z_t^\nu = z_0^\nu + \int_0^t F_\psi(z_\tau^\nu)\, d\tau$$

GT solution at x for time t

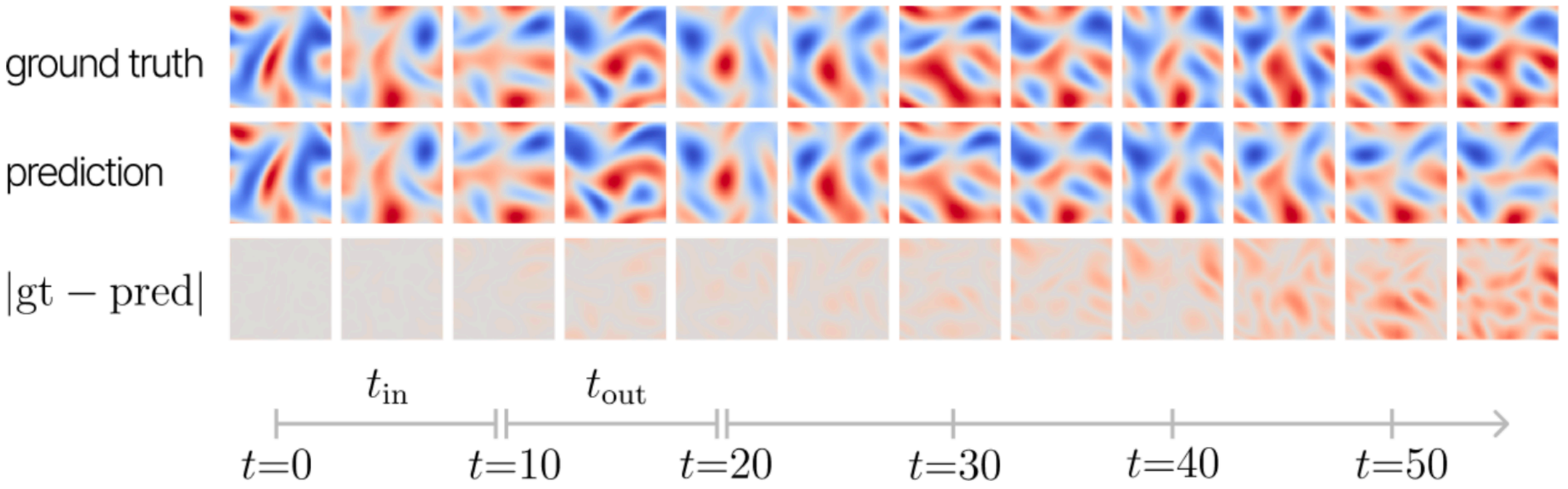
ENF recon with latent z_t

ENF latent that reconstructs initial state (from inner loop, 3 steps)

Trajectory through ENF latent space, modelled by F_ψ (neural ODE)

Equivariant PDE forecasting in latent space!

Navier stokes (no force term) + periodic boundaries



Equivariant PDE forecasting in latent space!

Navier stokes (no force ter

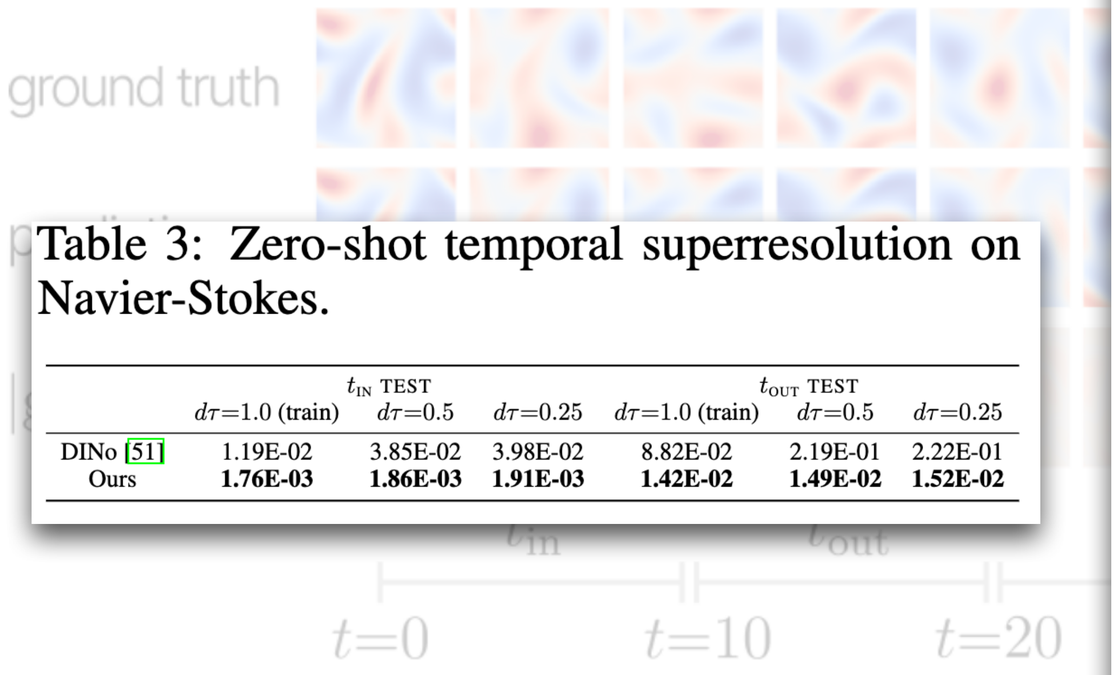


Table 3: Zero-shot temporal superresolution on Navier-Stokes.

	t_{IN} TEST			t_{OUT} TEST		
	$d\tau$=1.0 (train)	$d\tau$=0.5	$d\tau$=0.25	$d\tau$=1.0 (train)	$d\tau$=0.5	$d\tau$=0.25
DINo [51]	1.19E-02	3.85E-02	3.98E-02	8.82E-02	2.19E-01	2.22E-01
Ours	**1.76E-03**	**1.86E-03**	**1.91E-03**	**1.42E-02**	**1.49E-02**	**1.52E-02**

Table 2: MSE ↓ for Navier-Stokes on $\mathbb{T}^2$.

	t_{IN} TRAIN	t_{OUT} TRAIN	t_{IN} TEST	t_{OUT} TEST
		100% OF ν_0 OBSERVED		
CNODE [2]	6.02E-02	3.35E-01	5.48E-02	3.17E-01
FNO	9.43E-05	2.11E-03	8.44E-05	1.60E-03
G-FNO	**3.13E-05**	**3.49E-04**	**3.15E-05**	**3.52E-04**
DINo [49]	8.20E-03	6.85E-02	1.11E-02	9.08E-02
Ours AD,$\mathbf{a}^{\mathbb{T}_2/\pi}$	5.60$\pm$0.43E-02	0.37$\pm$0.34E-01	6.75$\pm$0.62E-02	4.00$\pm$0.38E-01
Ours $\mathbf{a}^{\emptyset}$	1.41$\pm$1.83E-02	1.67$\pm$1.27E-01	2.60$\pm$3.16E-02	2.14$\pm$1.46E-01
Ours $\mathbf{a}^{\mathbb{T}_2/\pi}$	1.45$\pm$0.08E-03	9.14$\pm$0.36E-03	1.57$\pm$0.09E-03	1.16$\pm$0.14E-02
		50% OF ν_0 OBSERVED		
CNODE [2]	1.38E-01	6.33E-01	1.52E-01	6.76E-01
FNO	3.31E-02	1.39E-01	3.20E-02	1.47E-01
G-FNO	2.75E-02	1.17E-01	2.32E-02	1.01E-01
DINo [49]	3.67E-02	2.81E-01	3.74E-02	2.83E-01
Ours AD,$\mathbf{a}^{\mathbb{T}_2/\pi}$	6.89$\pm$2.68E-02	3.95$\pm$2.18E-01	7.01$\pm$3.56E-02	4.01$\pm$2.29E-01
Ours $\mathbf{a}^{\emptyset}$	1.05$\pm$0.04E-02	1.45$\pm$0.01E-01	2.60$\pm$3.16E-02	2.14$\pm$1.46E-01
Ours $\mathbf{a}^{\mathbb{T}_2/\pi}$	**1.50$\pm$0.17E-03**	**8.97$\pm$1.57E-03**	**5.75$\pm$2.58E-03**	**5.03$\pm$2.63E-02**
		5% OF ν_0 OBSERVED		
CNODE [2]	1.23E+01	2.14E+01	1.20E+01	4.35E+01
FNO	4.13E-01	7.70E-01	3.84E-01	7.07E-01
G-FNO	3.56E-01	7.09E-01	3.40E-01	6.47E-01
DINo [49]	3.67E-02	2.81E-01	3.94E-02	2.91E-01
Ours AD,$\mathbf{a}^{\mathbb{T}_2/\pi}$	6.89$\pm$2.68E-02	3.95$\pm$2.18E-01	7.01$\pm$3.56E-02	4.01$\pm$2.29E-01
Ours $\mathbf{a}^{\emptyset}$	7.31$\pm$1.37E-02	2.97$\pm$2.42E-01	7.96$\pm$1.65E-02	3.35$\pm$3.41E-01
Ours $\mathbf{a}^{\mathbb{T}_2/\pi}$	**3.19$\pm$1.07E-02**	**1.33$\pm$0.35E-01**	**3.44$\pm$1.43E-02**	**1.61$\pm$4.93E-01**

Equivariant PDE forecasting in latent space!

Shallow-Water equation on the sphere

Table 3: MSE ↓ on Shallow-Water equations on the sphere.

	t_{IN} TRAIN	t_{OUT} TRAIN	t_{IN} TEST	t_{OUT} TEST
		TRAIN RESOLUTION		
DINo [49]	1.75E-04	**1.36E-03**	2.01E-04	**1.37E-03**
Ours $\mathbf{a}^{\text{SW}}$	**9.94**$\pm$0.41**E-05**	1.89$\pm$0.03E-03	**1.09**$\pm$1.14**E-04**	1.87$\pm$0.04E-03
		ZERO-SHOT 2X SUPER-RESOLUTION		
DINo [49]	3.03E-04	2.03E-03	3.37E-04	2.03E-03
Ours $\mathbf{a}^{\text{SW}}$	**1.58** $\pm$ 0.02**E-04**	**1.96** $\pm$0.02**E-03**	**1.61** $\pm$0.01**E-04**	**1.93** $\pm$0.02**E-03**

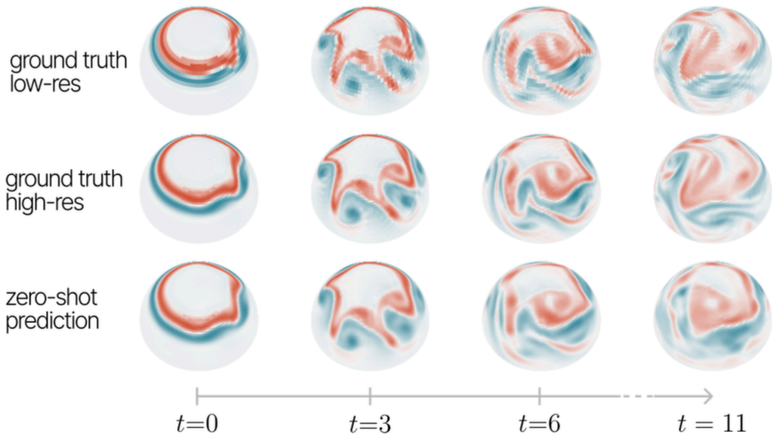


Figure 7: Test samples at train resolution (top), 2× train resolution (middle) and corresponding predictions from our equivariant model ($\mathbf{a}^{\text{SW}}$ Eq. 8 (bottom). The model does not produce significant upsampling artefacts, but fails to capture dynamics outside the training horizon.

Internally-heated convection on the ball

Table 4: MSE ↓ on Internally-Heated Convection in the ball.

	t_{IN} TRAIN	t_{OUT} TRAIN	t_{IN} TEST	t_{OUT} TEST
DINo [49]	2.94E-03	7.56E-02	3.06E-03	7.78E-02
Ours $\mathbf{a}^{\mathbb{B}^3}$	**5.79** $\pm$0.17**E-04**	**7.72** $\pm$0.55**E-03**	**5.99**$\pm$0.15**E-04**	**7.97**$\pm$0.46**E-03**

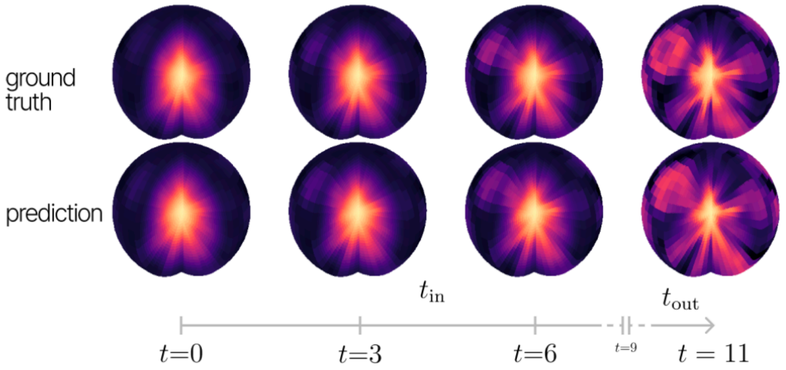


Figure 8: Test samples (top) and corresponding predictions from our model equivariant to S^2-rotations in the ball. (Eq. 14)

Known limitations of ENF

Problem: Underpowered decoder → many/large latents
Maybe: Multi-layer ENF? Adding self-attention between latents?

Problem: MAML is problematic for inferring latent pose (maybe due to local minima?).
Maybe: Replace SGD with an encoder?

Problem: General limited performance of downstream models
Maybe: Adding additional structure to the ENF latent space? Maybe a VAE variant?

Problem: Current latents are forcibly local, some tasks require global information / features are better modelled globally
Maybe: Hierarchy? Global / local latent separation?

Question: How does the choice of RFF embedding (periodic) impact reconstruction / downstream performance?

Problem: Unclear how to use ENF to represent spatiotemporal data
Maybe: Predictor/corrector scheme like SAVI(++), MooG?