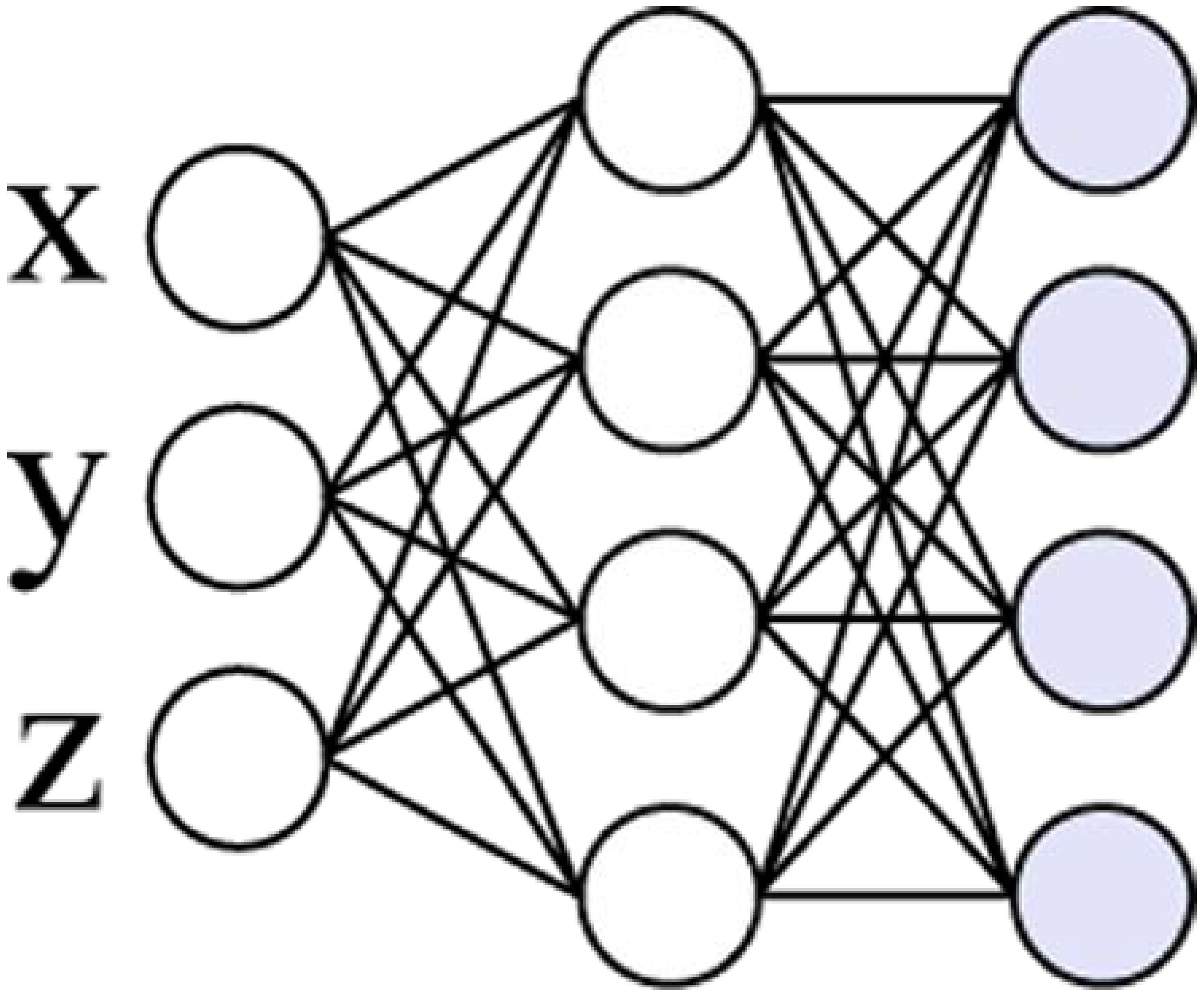
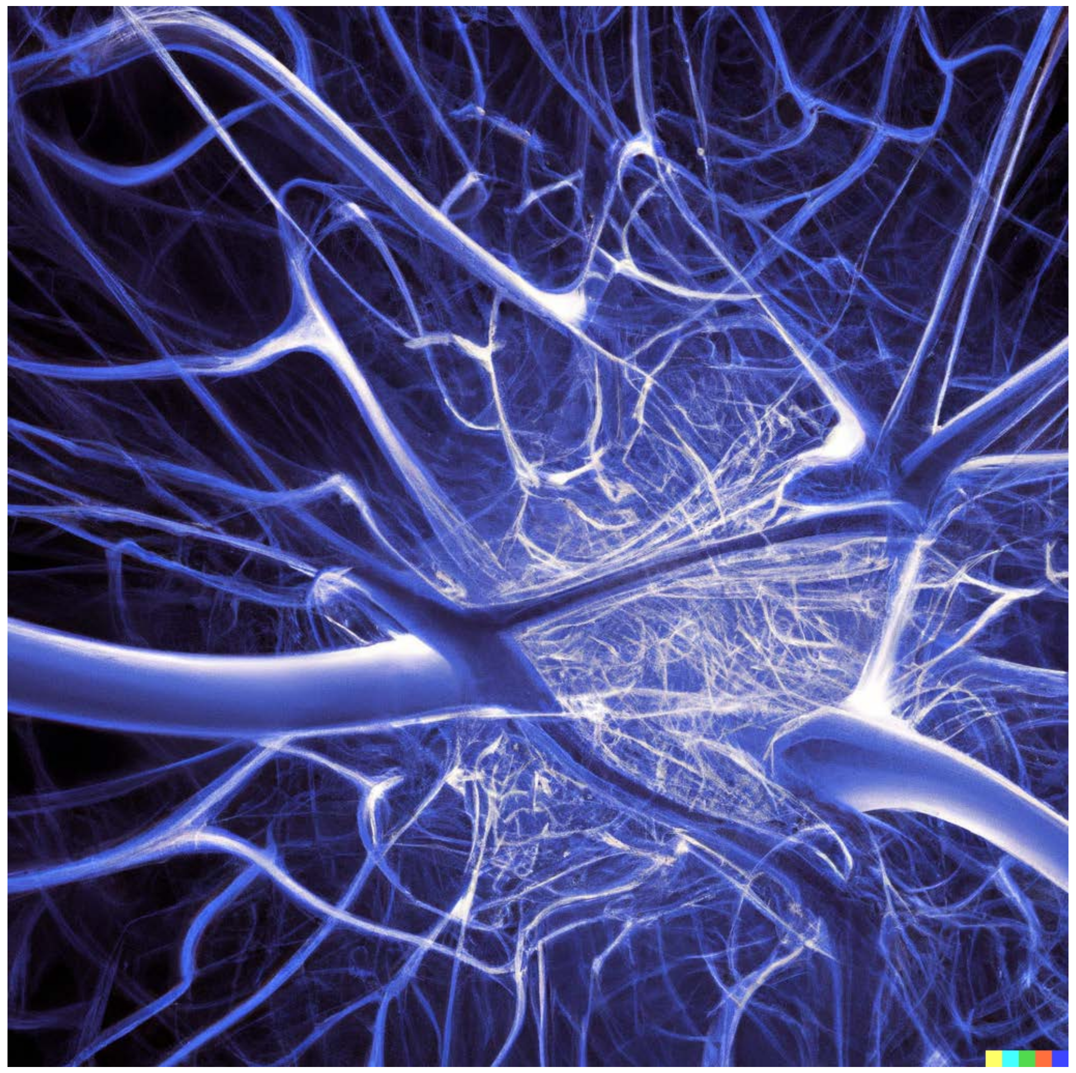
An Introduction to Neural Fields
in Computer Vision and Image Processing

Tomáš Kerepecký
kerepecky@utia.cas.cz





By DALL-E

An Introduction to Neural Fields in Computer Vision and Image Processing

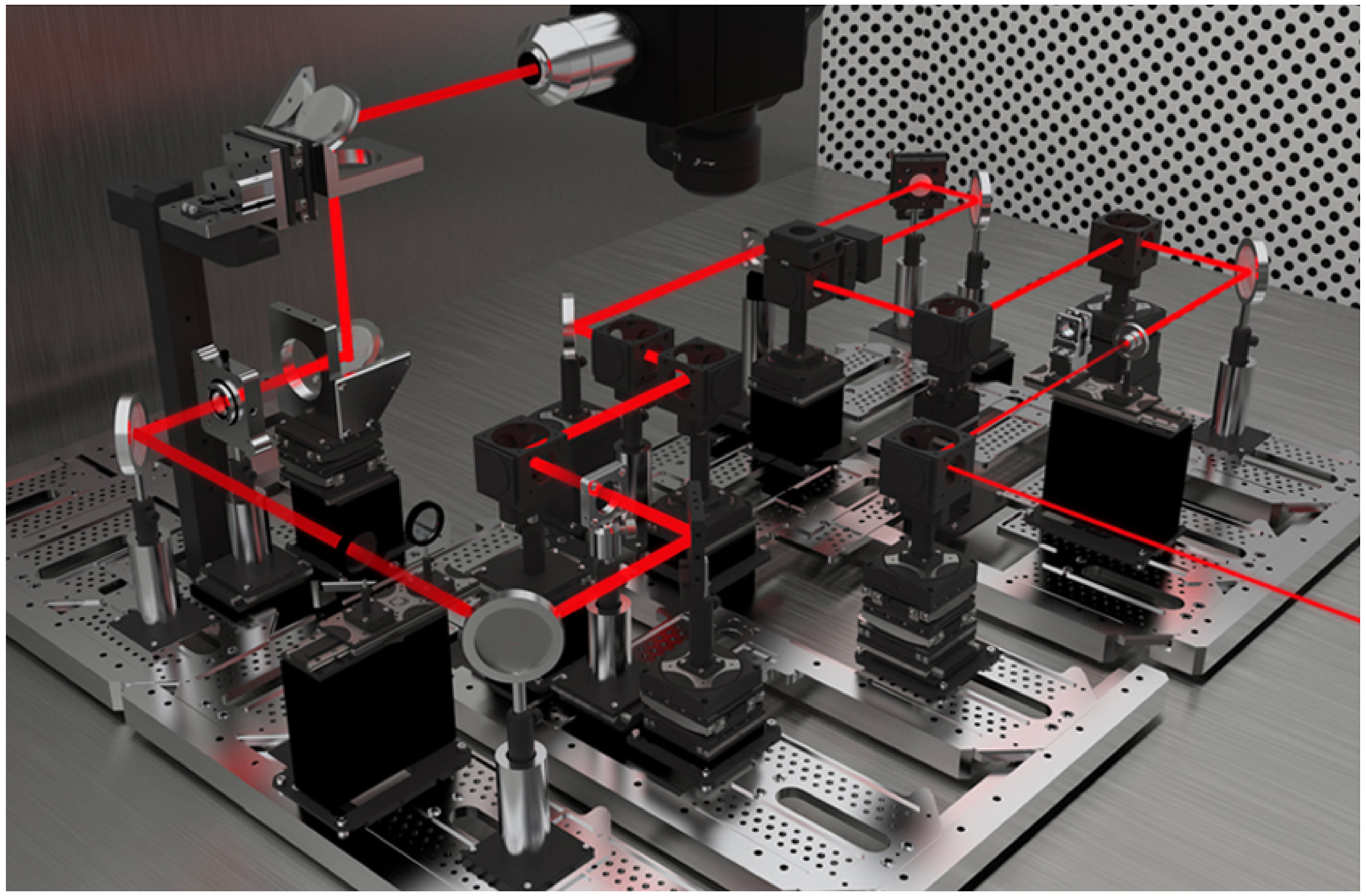
Tomáš Kerepecký
kerepecky@utia.cas.cz

CZECH TECHNICAL UNIVERSITY IN PRAGUE

TCM INTERNATIONAL INSTITUTE

An Introduction to Neural Fields
in Computer Vision and Image Processing

Started PhD course in Computational physics



attocube.com

An Introduction to Neural Fields in **Computer Vision and Image Processing**

Continue PhD course in Image processing

An Introduction to Neural Fields
in **Computer Vision and Image Processing**

Computer vision is the ability of computers to interpret and understand visual data.

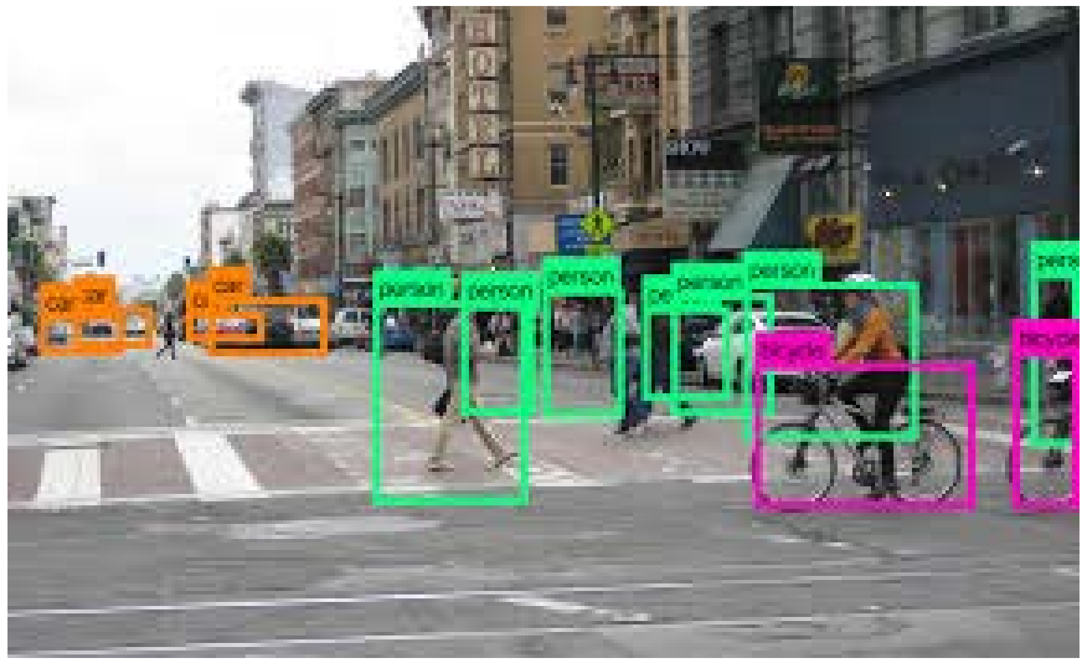
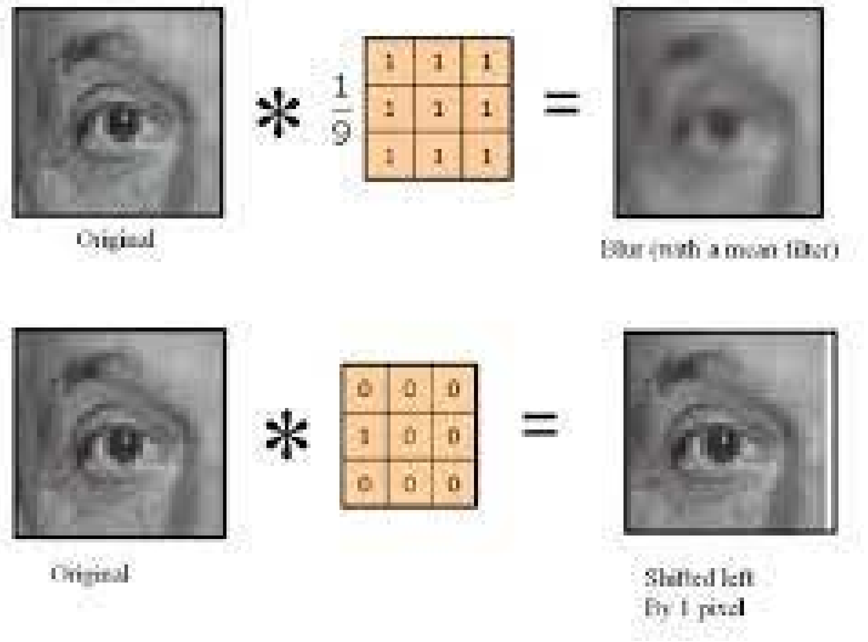


Image processing is the manipulation or analysis of images to extract information or enhance their appearance.

ChatGPT



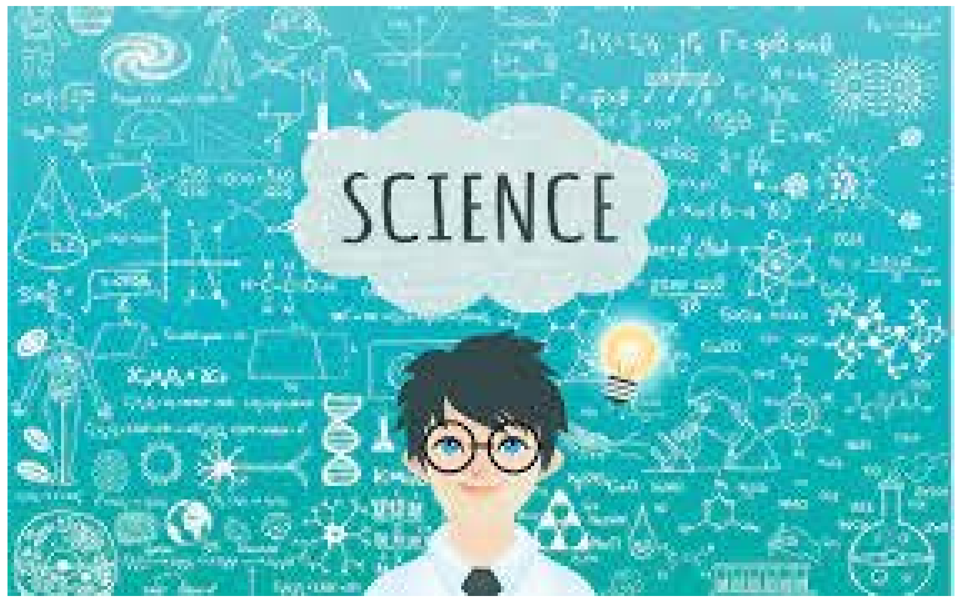
An Introduction to **Neural Fields**
in Computer Vision and Image Processing

Fulbright-Masaryk scholarship

FULBRIGHT.CZ

An Introduction to Neural Fields in Computer Vision and Image Processing

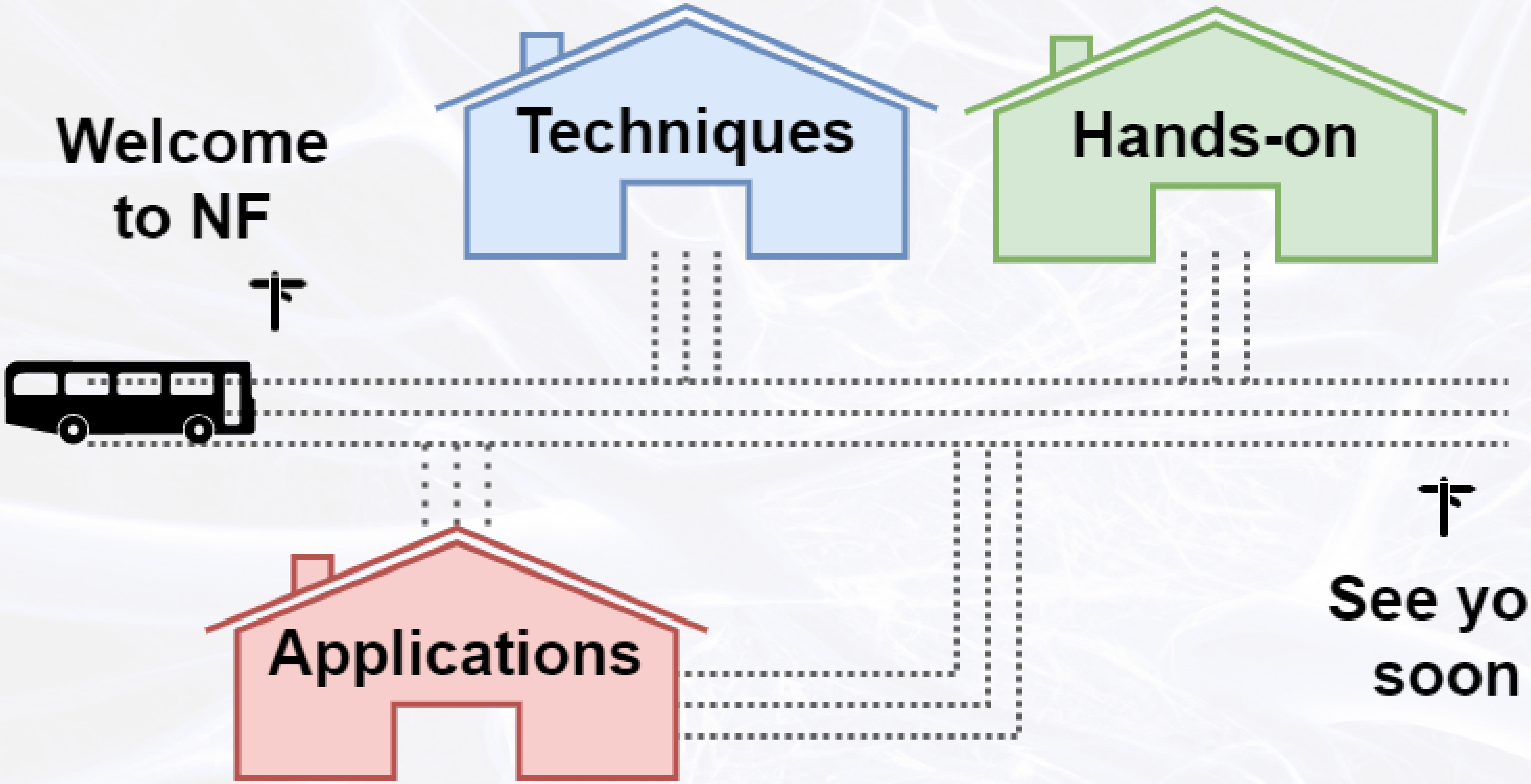
+



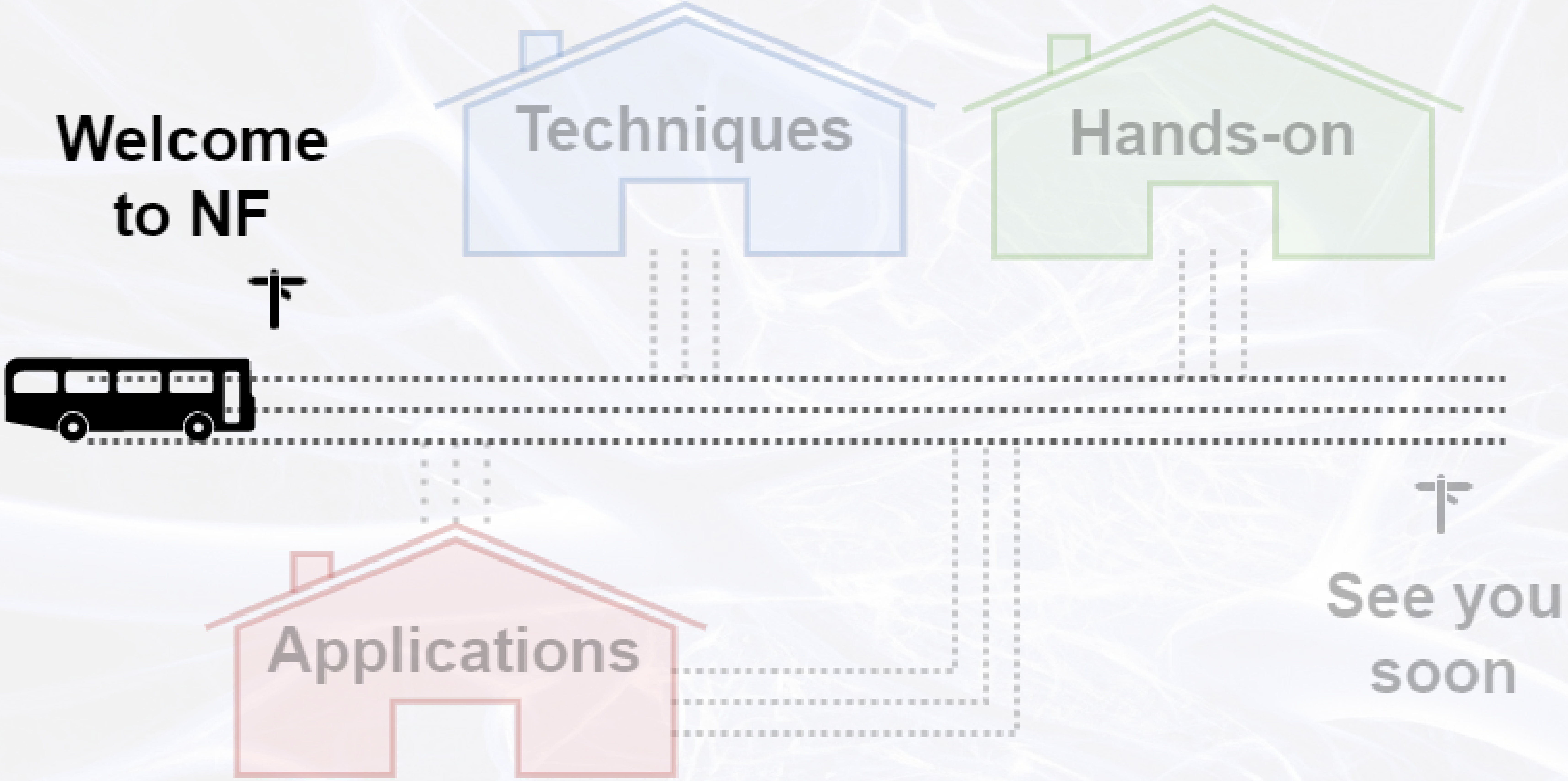
= **Fulbright-Masaryk scholarship**

FULBRIGHT.CZ

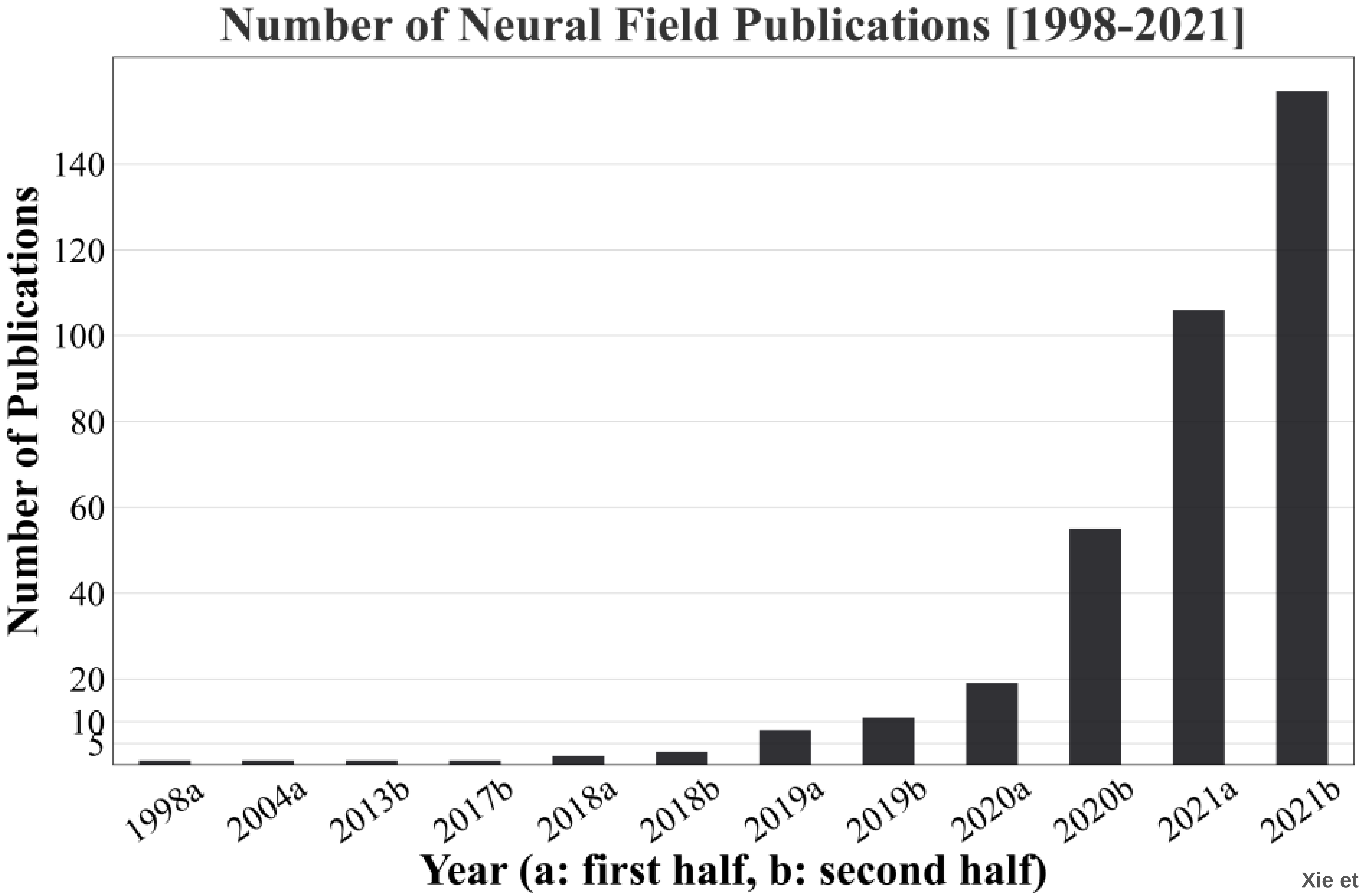
Let's go on a tour to NF



Let's go on a tour to NF



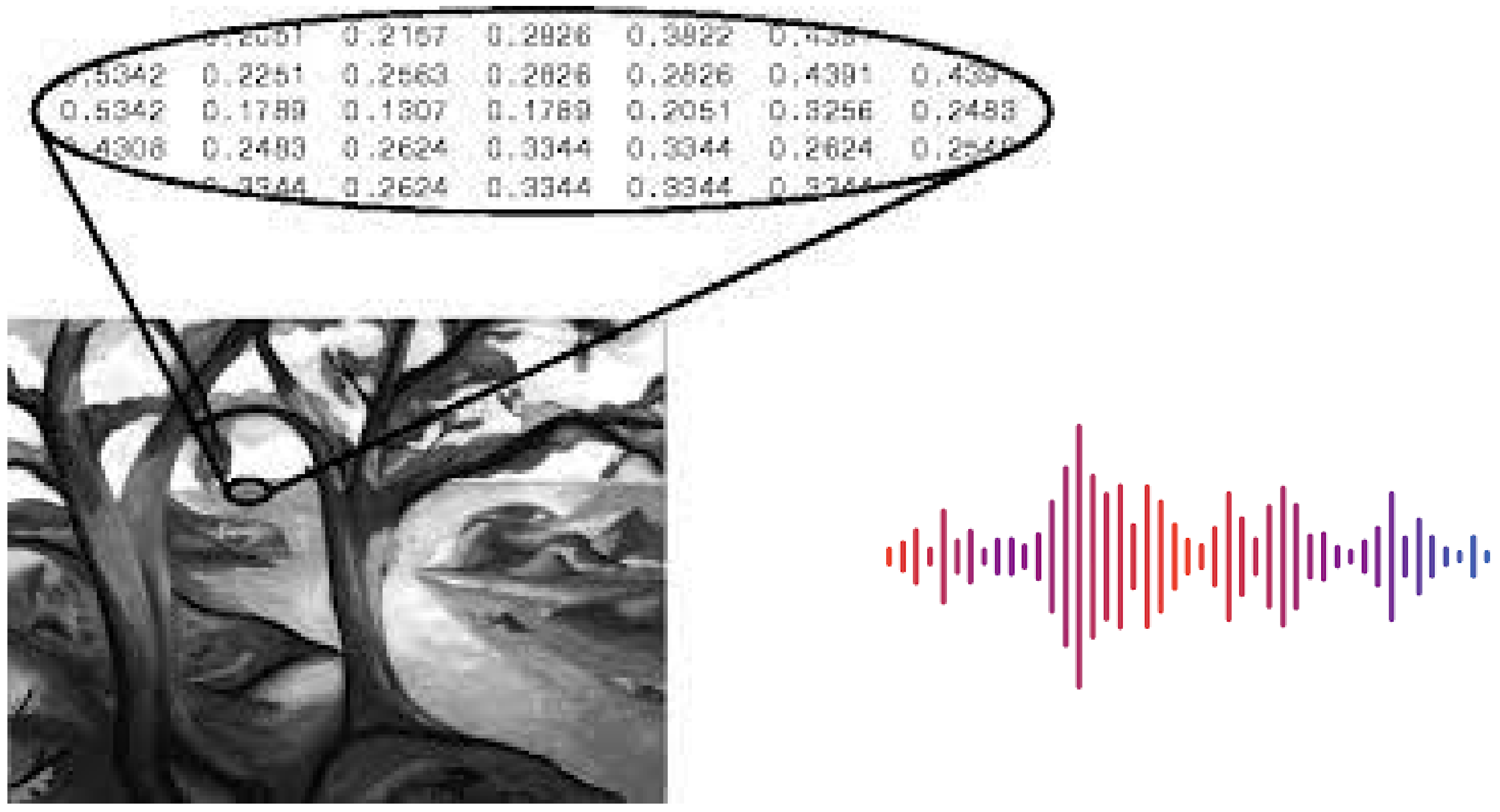
Welcome to NF



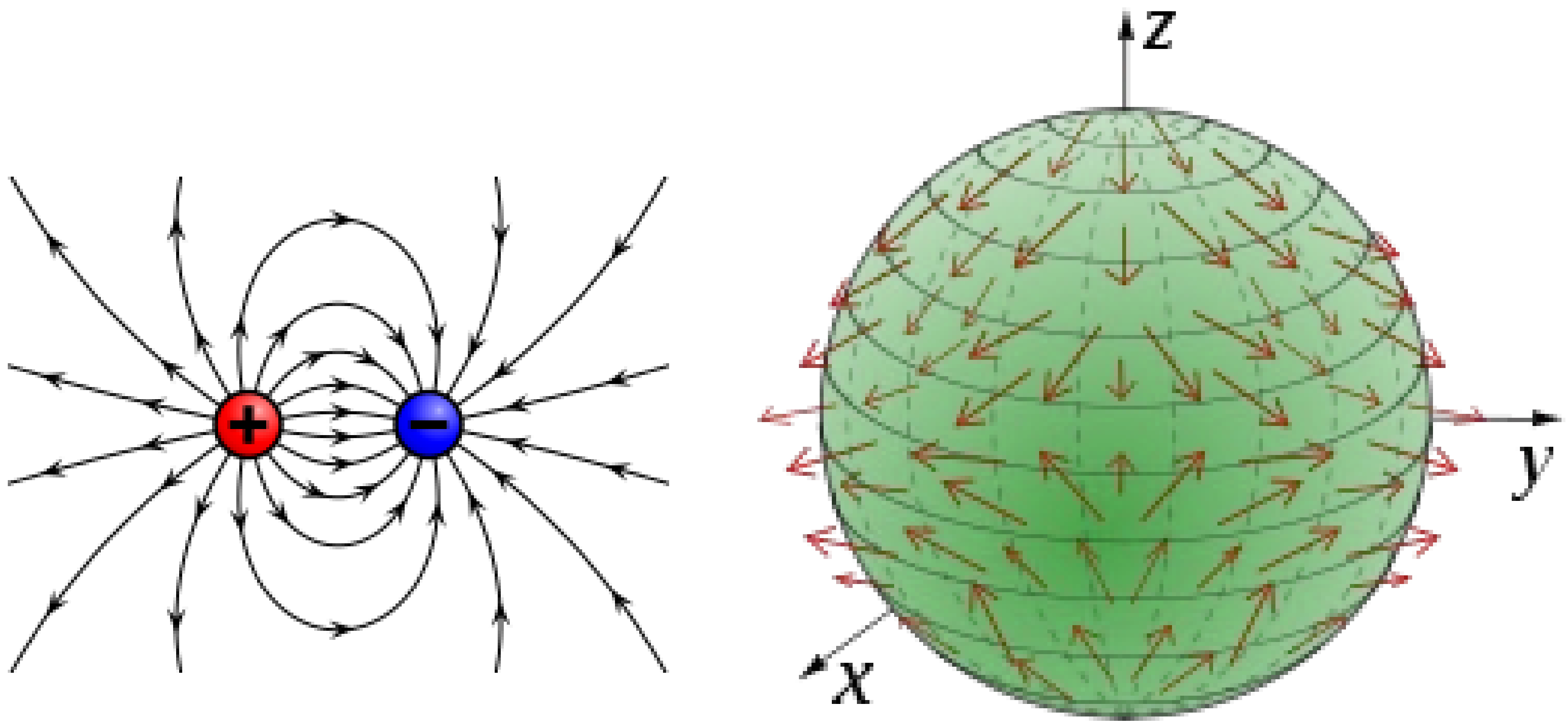
Xie et al., 2022

What is field?

A field is a quantity defined for all spatial and/or temporal coordinates.



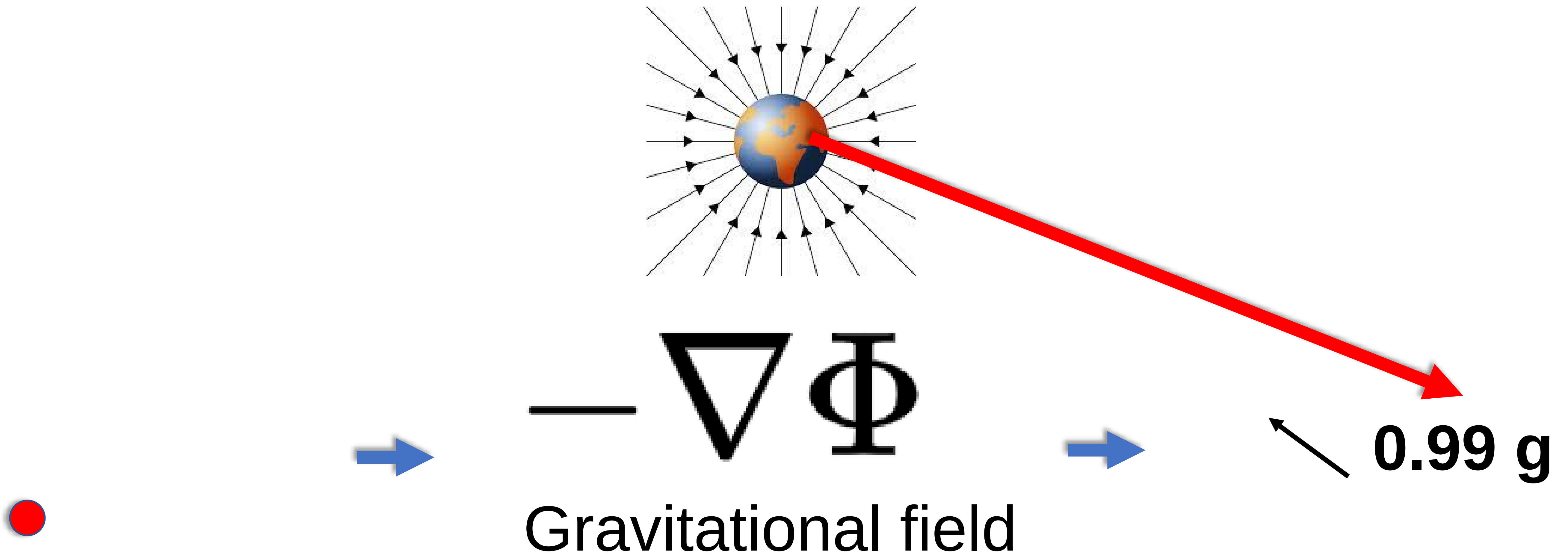
Scalar field



Vector field

What is field?

A field is a quantity defined for all spatial and/or temporal coordinates.



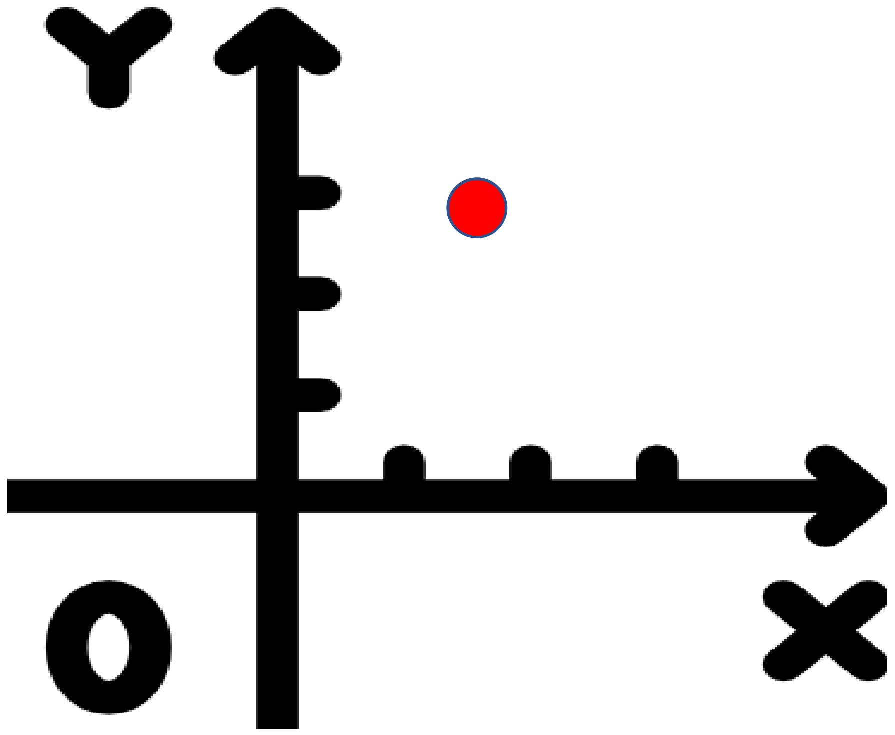
$-\nabla\Phi$

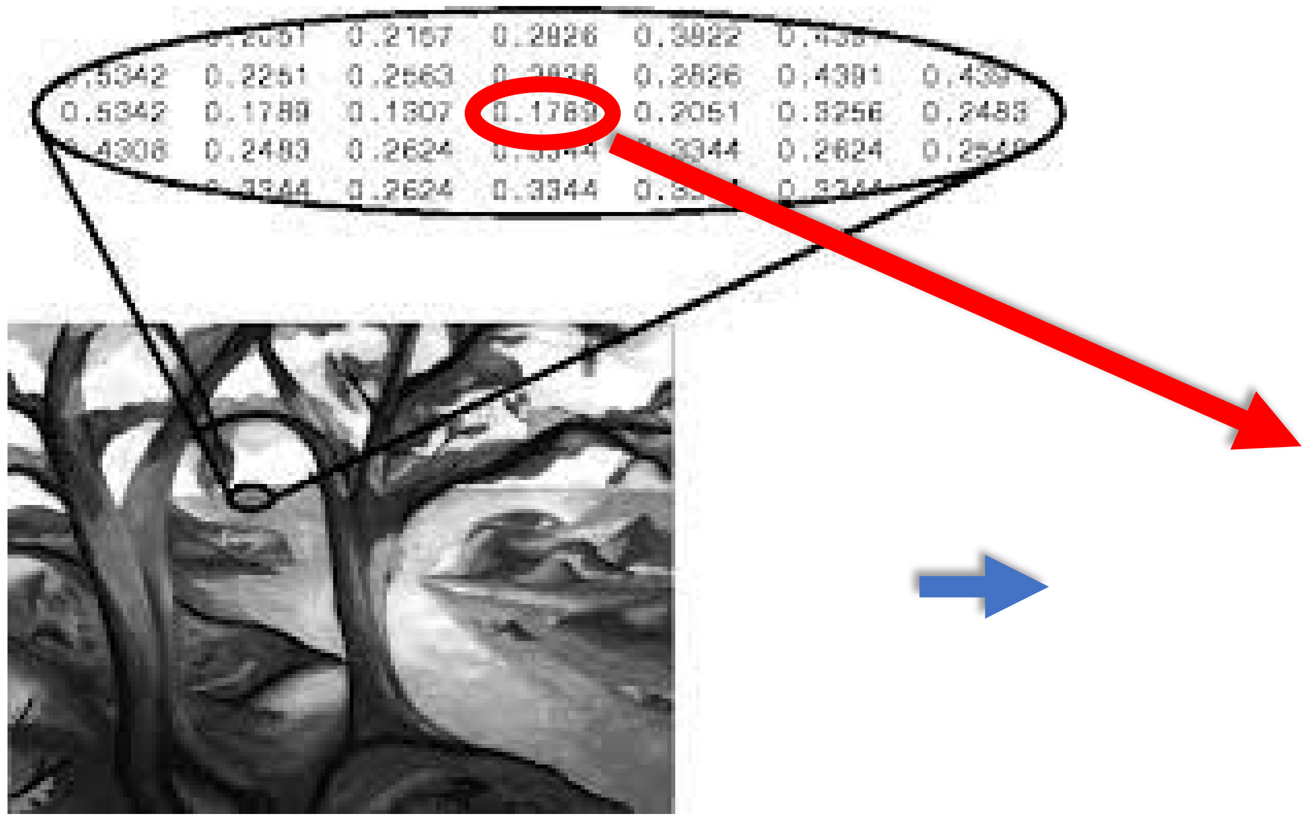
Gravitational field

0.99 g

What is field?

A field is a quantity defined for all spatial and/or temporal coordinates.





Image

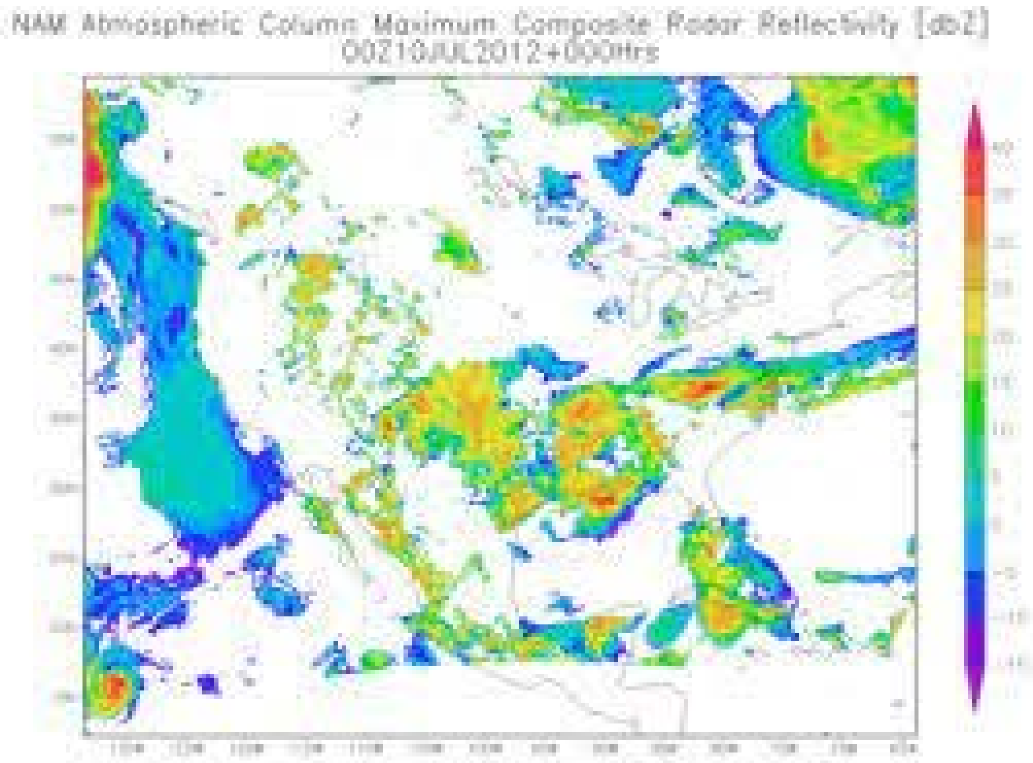
0.1769

What is field?

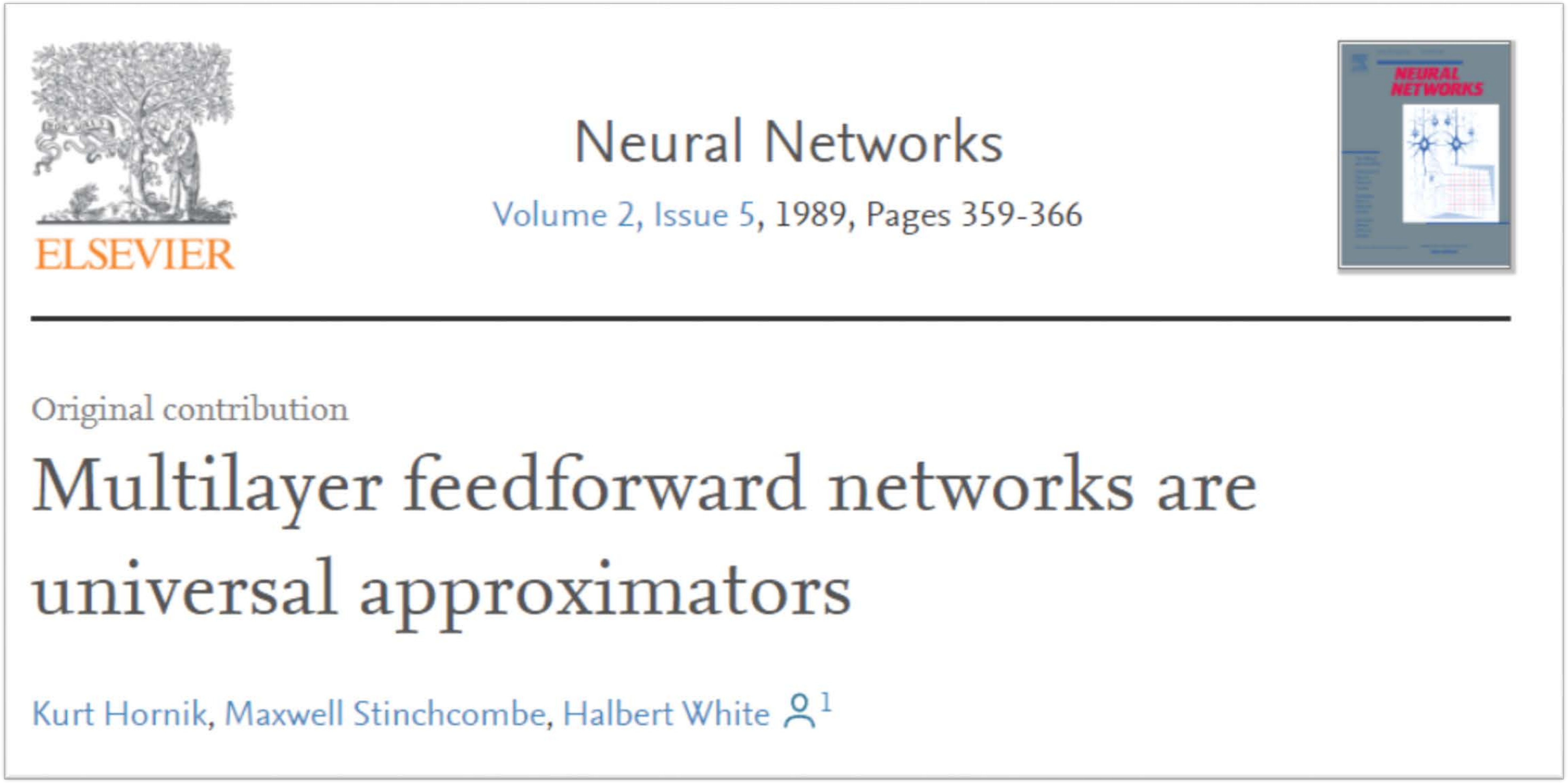
A field is a quantity defined for all spatial and/or temporal coordinates.

→ $F(\boldsymbol{x};\Theta)$ →

Parametrized field


NAM Atmospheric Column Maximum Composite Radar Reflectivity [dbZ]
00Z10JUL2012+000Hrs

Universal approximation theorem



ELSEVIER

Neural Networks

Volume 2, Issue 5, 1989, Pages 359-366

Original contribution

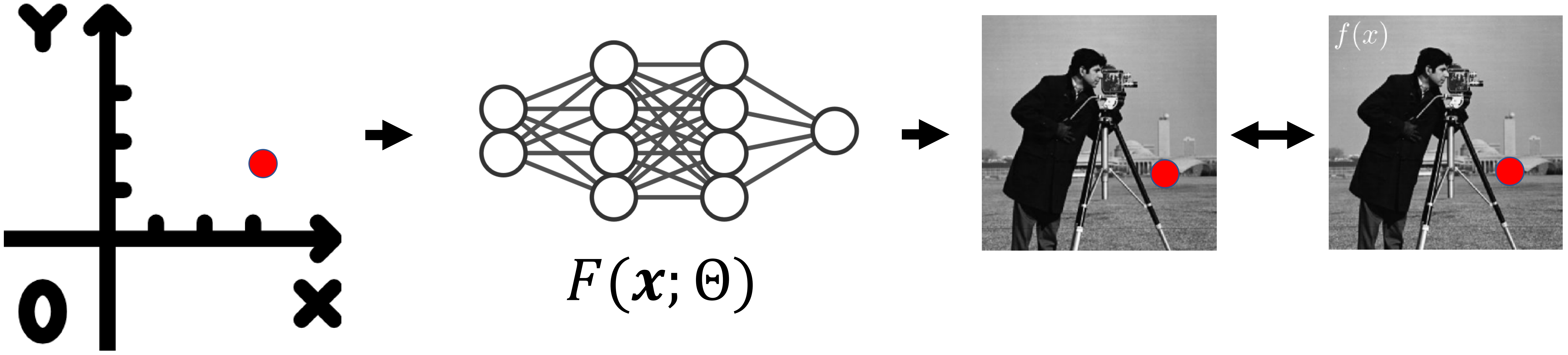
Multilayer feedforward networks are universal approximators

Kurt Hornik, Maxwell Stinchcombe, Halbert White [1]

Feedforward neural networks can approximate
any continuous function.

What is neural field?

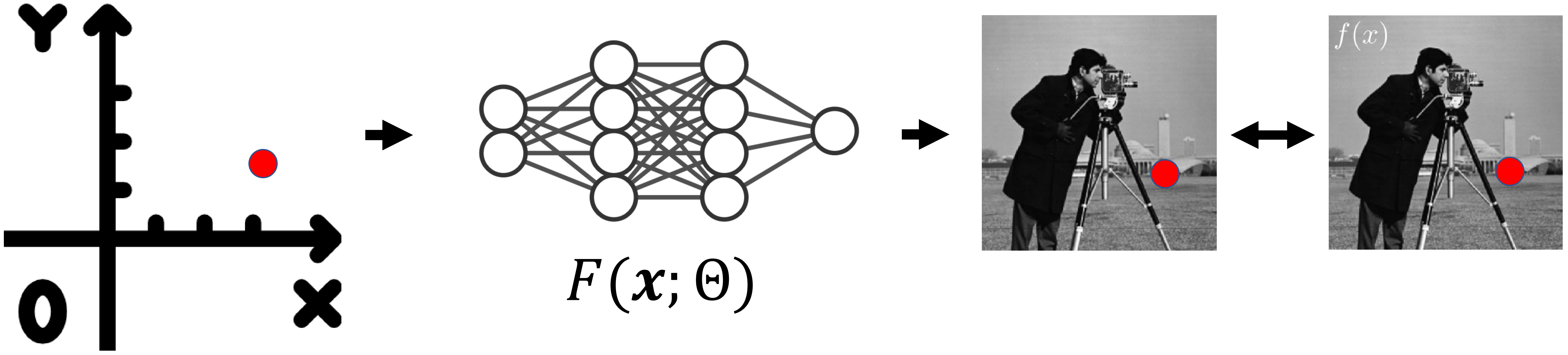
A neural field is a field that is parametrized fully or in part by a neural network.



$F(\boldsymbol{x};\Theta)$

What is neural field?

A neural field is a field that is parametrized fully or in part by a neural network.



➔ encoding objects and scenes in the weights of an MLP.

Terminology

Coordinate-based Neural Networks

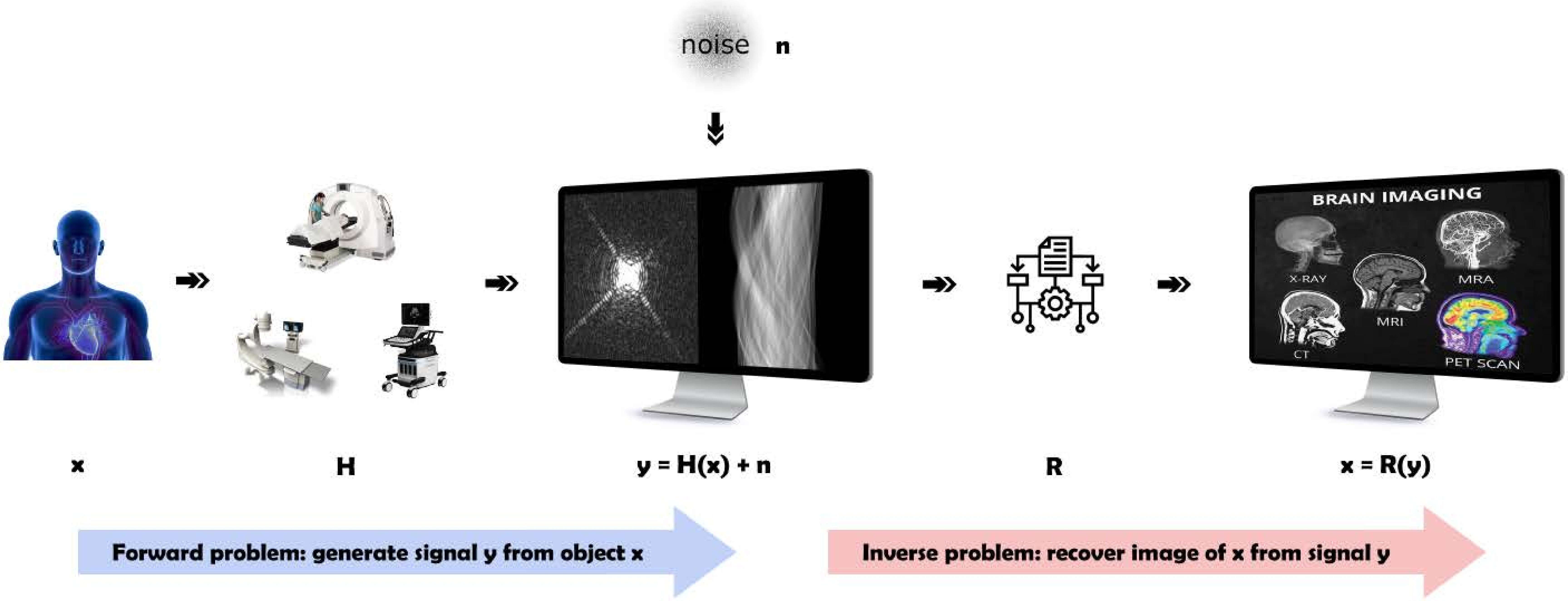
Neural Fields

NeRFs

Implicit Neural Representation

DALL-E

Forward and Inverse problem



x

H

y = H(x) + n

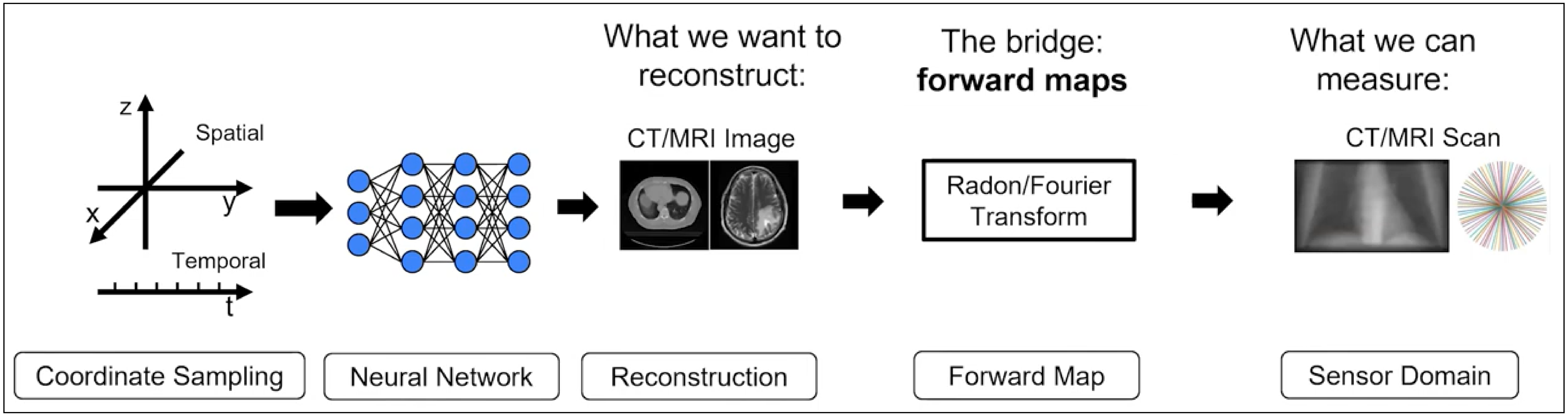
R

x = R(y)

Forward problem: generate signal y from object x

Inverse problem: recover image of x from signal y

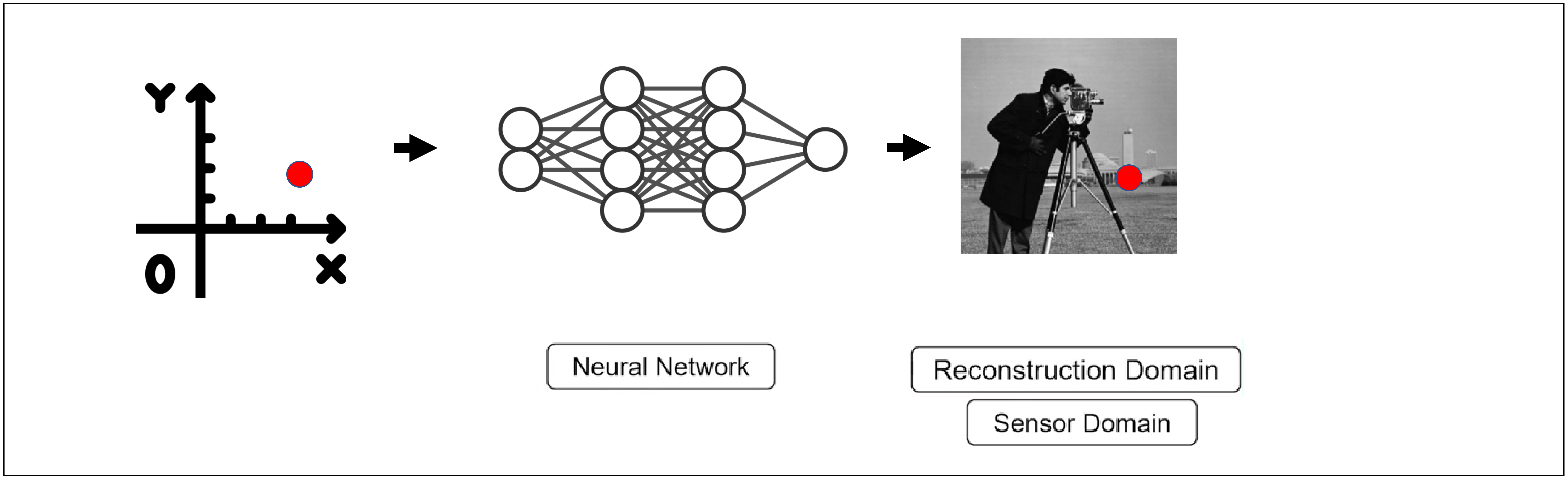
Neural Field framework



Xie et al., 2022

Welcome to NF

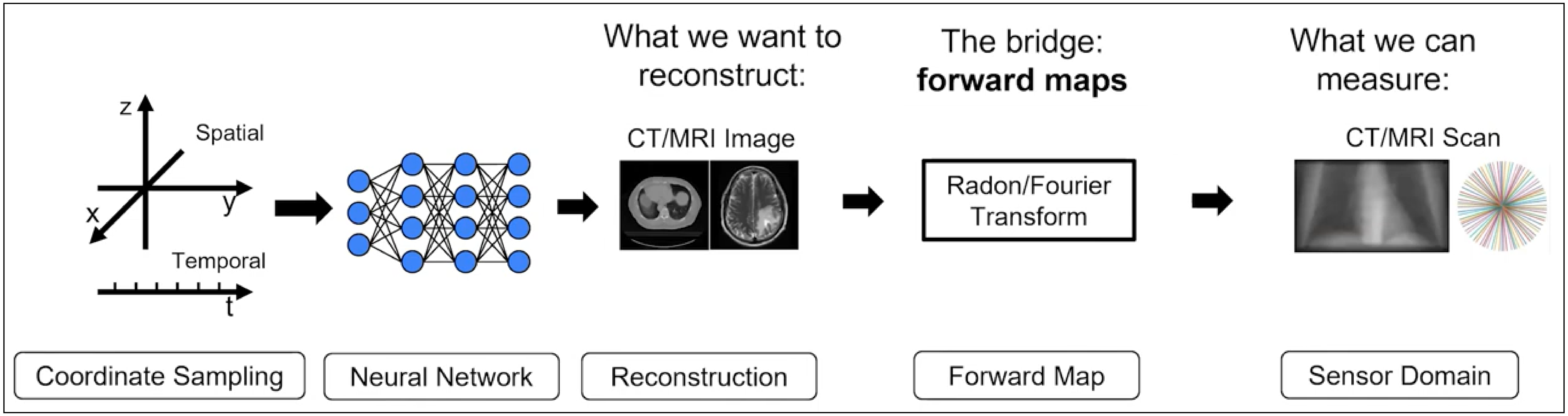
Neural Field framework



Neural Network

Reconstruction Domain

Sensor Domain



What we want to reconstruct:

CT/MRI Image

The bridge: **forward maps**

Radon/Fourier Transform

What we can measure:

CT/MRI Scan

Coordinate Sampling

Neural Network

Reconstruction

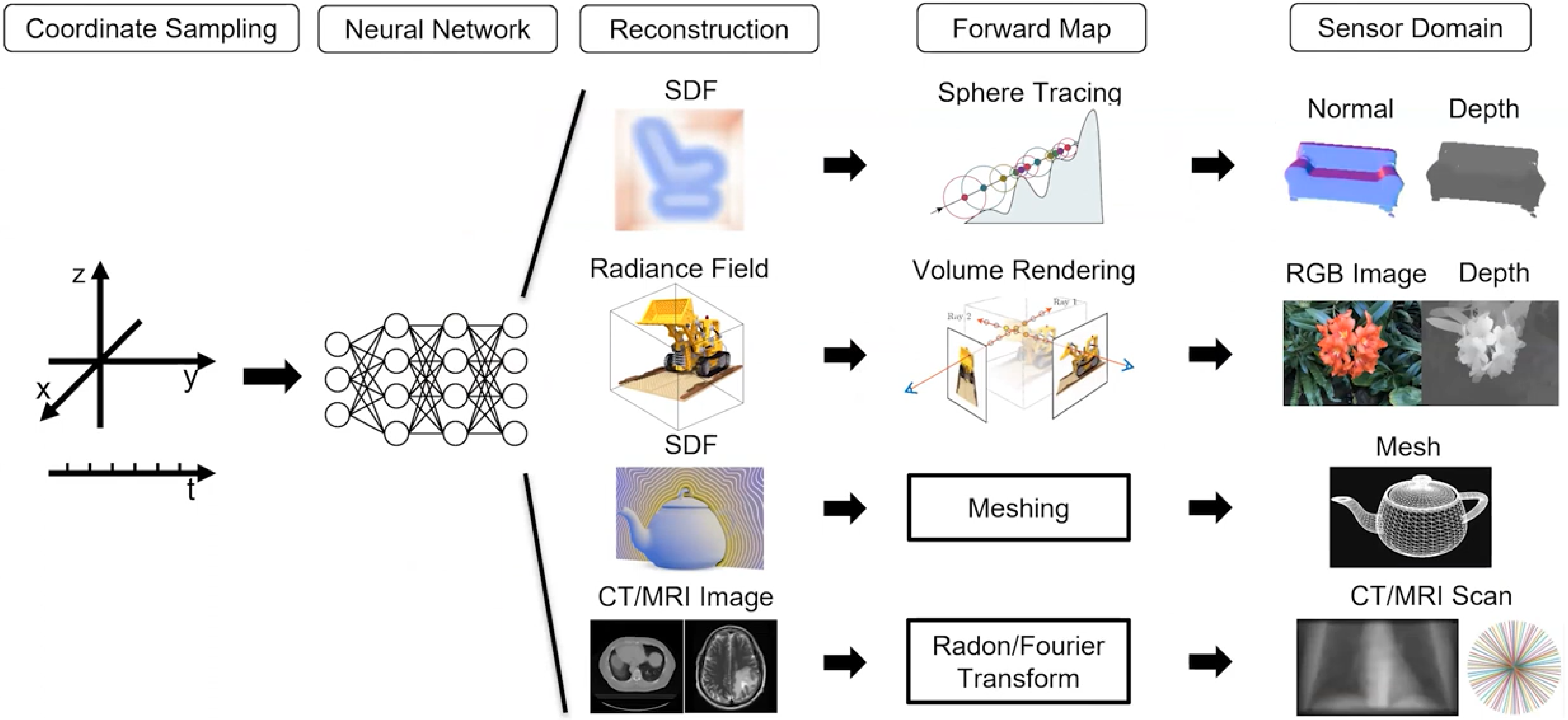
Forward Map

Sensor Domain

Xie et al., 2022

Welcome to NF

Neural Field framework



Coordinate Sampling

Neural Network

Reconstruction

Forward Map

Sensor Domain

SDF → Sphere Tracing → Normal Depth

Radiance Field → Volume Rendering → RGB Image Depth

SDF → Meshing → Mesh

CT/MRI Image → Radon/Fourier Transform → CT/MRI Scan

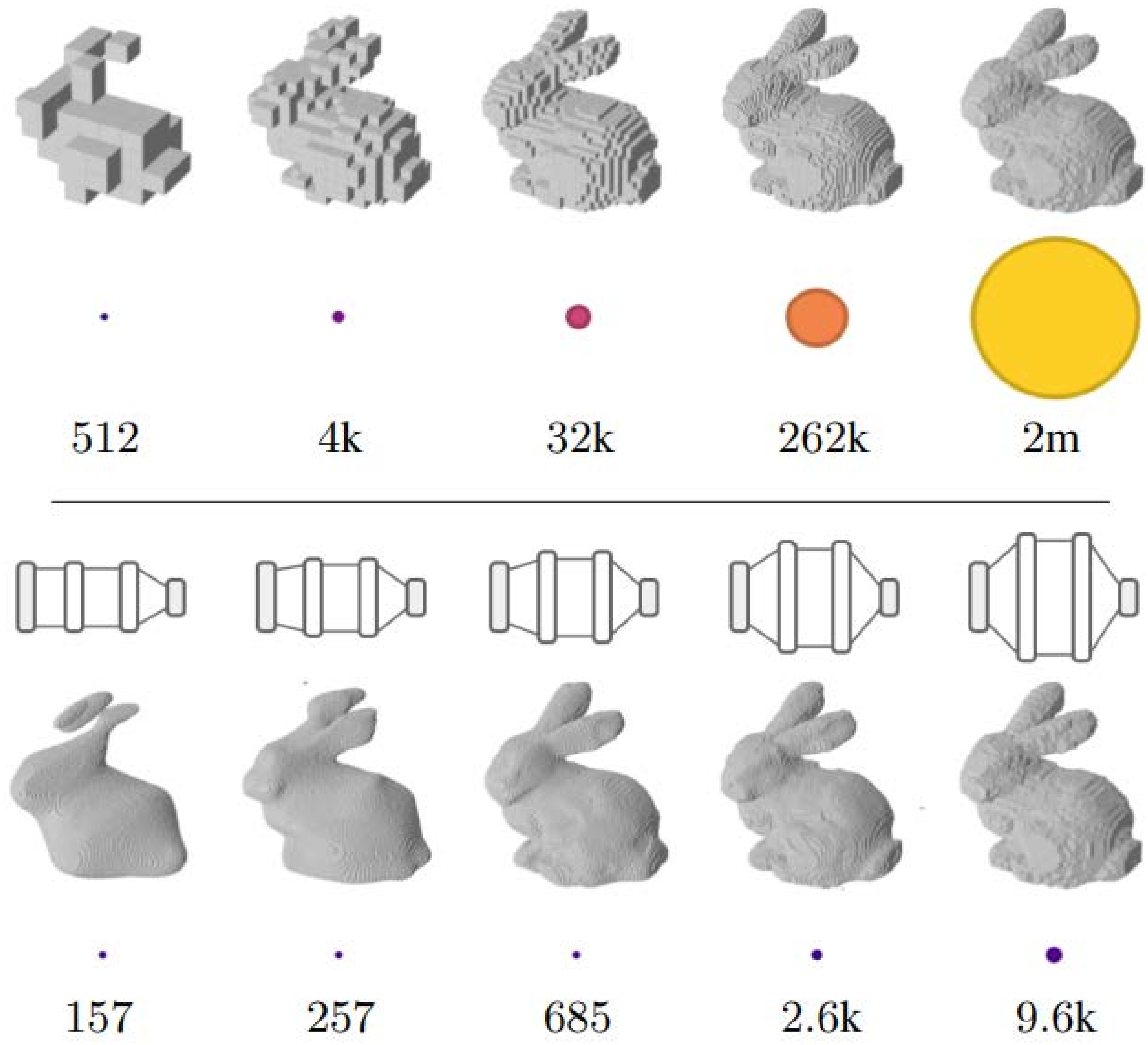
Xie et al., 2022

Why Neural Fields?

1. Continuous
2. Compactness
3. Regularization
4. Domain Agnostic

DALL-E

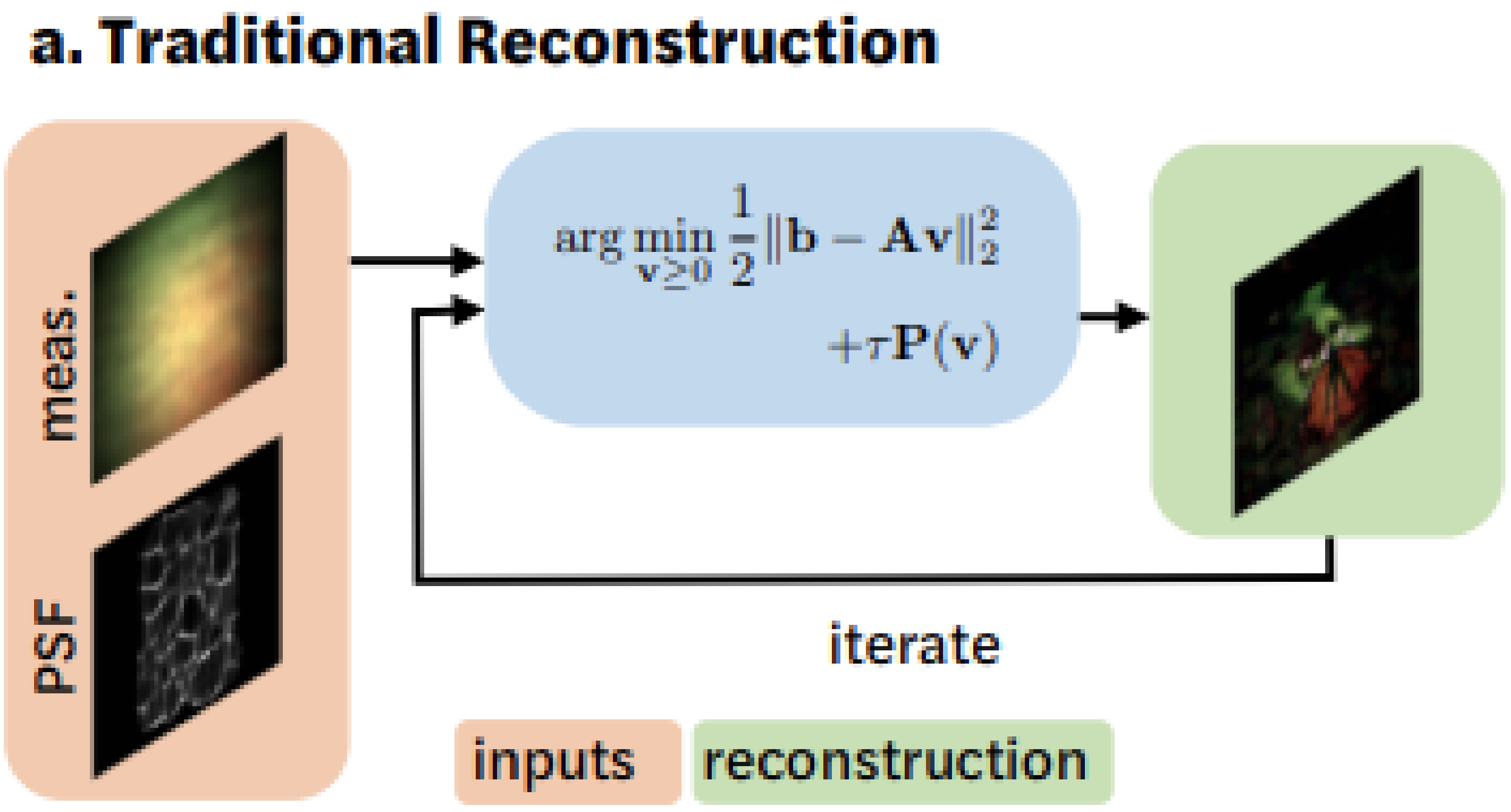
Why NF – Compactness



Neural representation scale much more gracefully with resolution than array representations.

Dupont et al. 2022

Why NF – Regularization


a. Traditional Reconstruction
meas.
PSF
$\arg\min_{\mathbf{v}\geq 0} \frac{1}{2}\|\mathbf{b}-\mathbf{A}\mathbf{v}\|_2^2 + \tau \mathbf{P}(\mathbf{v})$
iterate
inputs
reconstruction

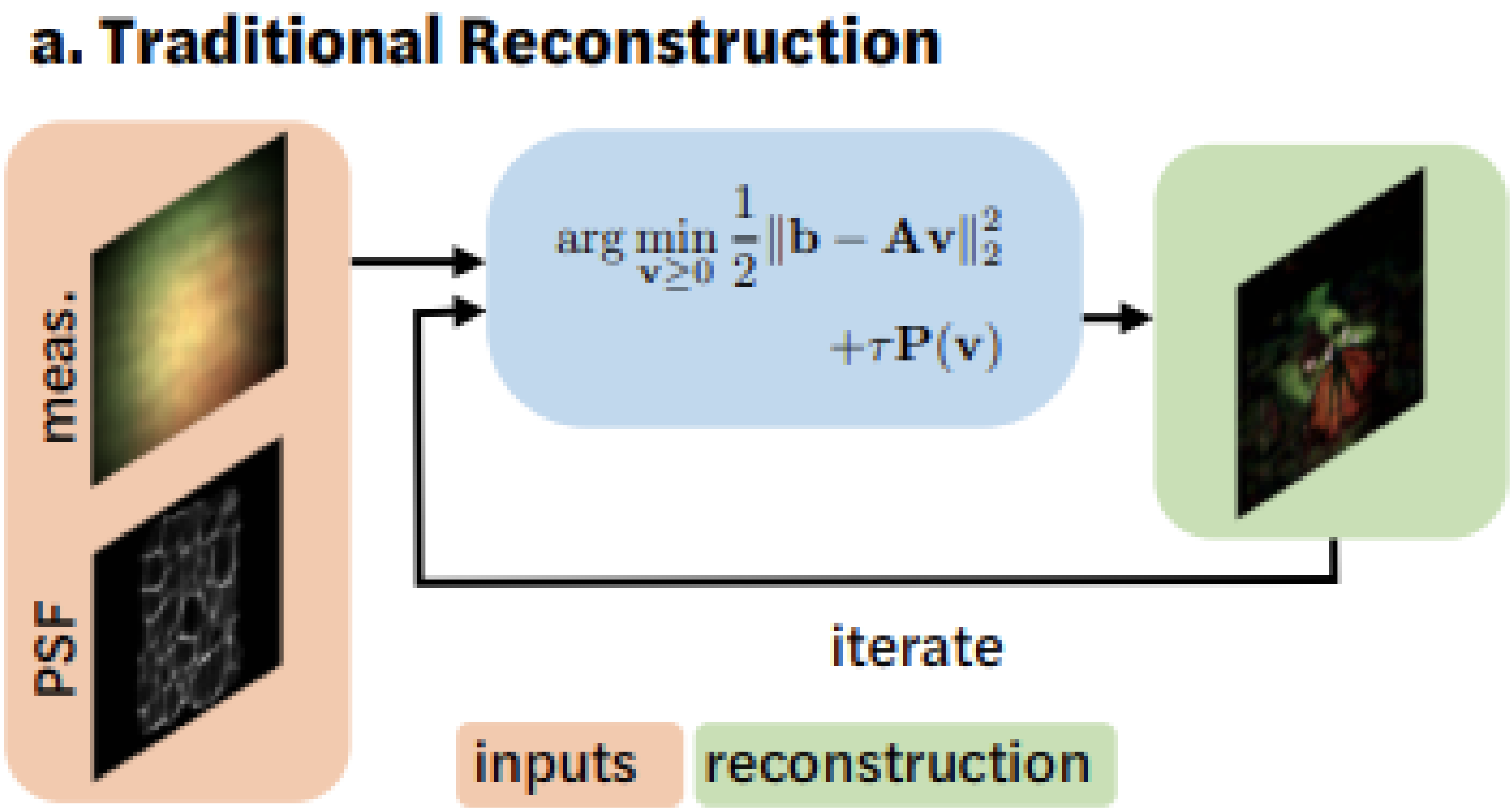
b… measurement
A… forward operator
v… unknown original

min ||b-Av||

P(v)… regularization

Monakhova et al. 2021

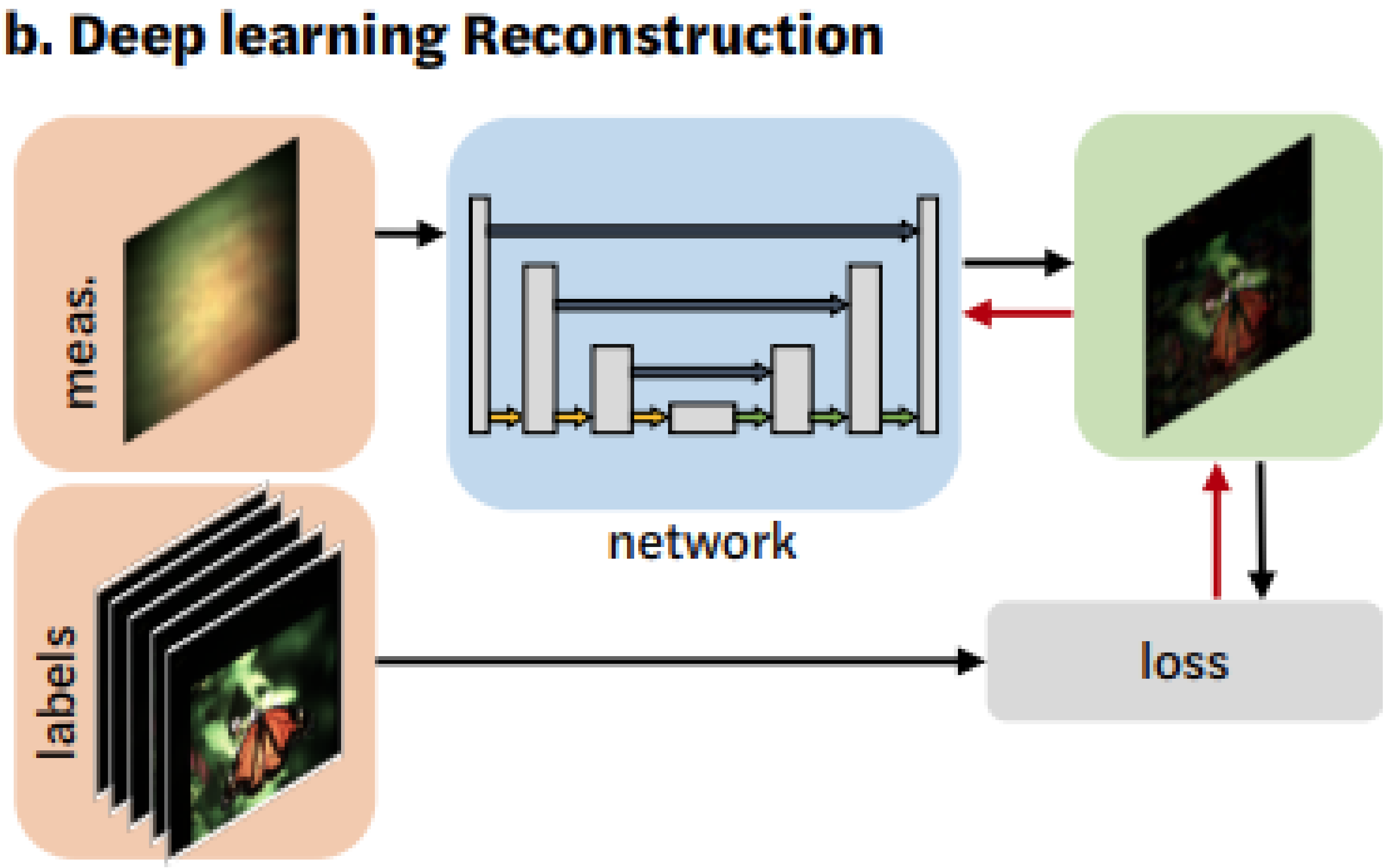
Why NF – Regularization


a. Traditional Reconstruction
meas.
PSF
$\arg\min_{\mathbf{v}\geq 0} \frac{1}{2}\|\mathbf{b}-\mathbf{A}\mathbf{v}\|_2^2$
$+\tau\mathbf{P}(\mathbf{v})$
iterate
inputs
reconstruction

b… measurement
A… forward operator
v… unknown original

min ||b-Av||

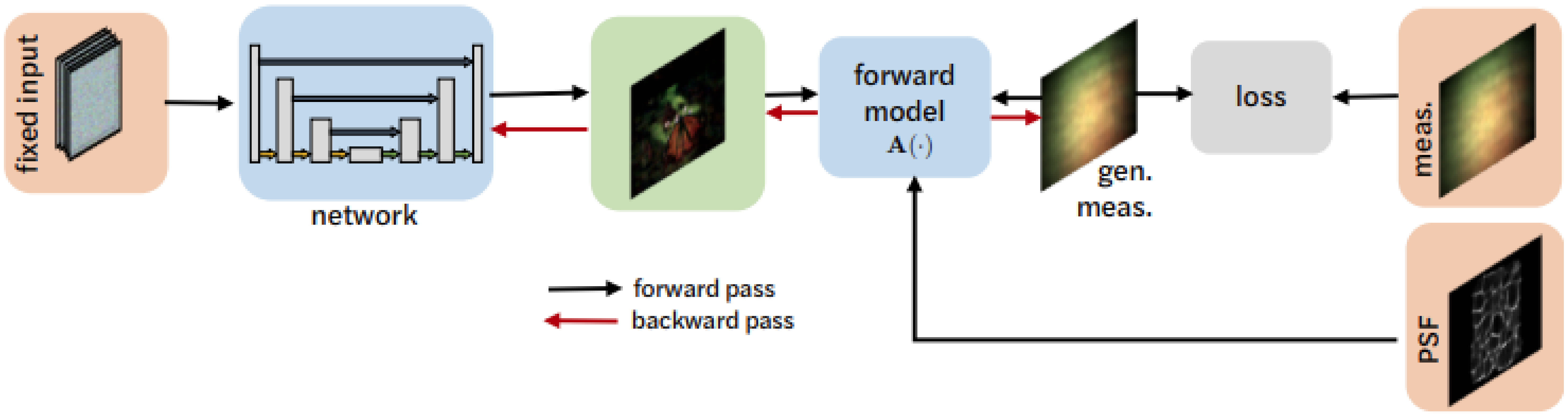
P(v)… regularization



Monakhova et al. 2021

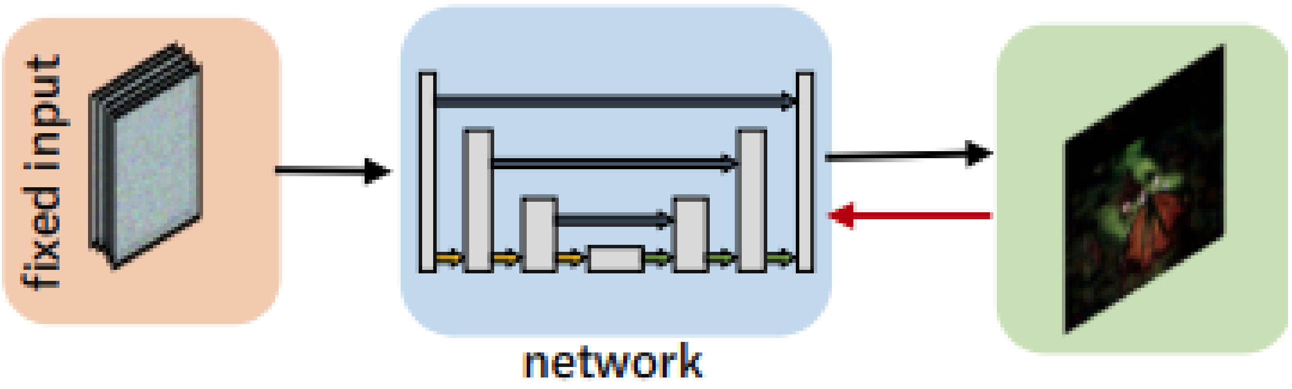
Why NF – Regularization

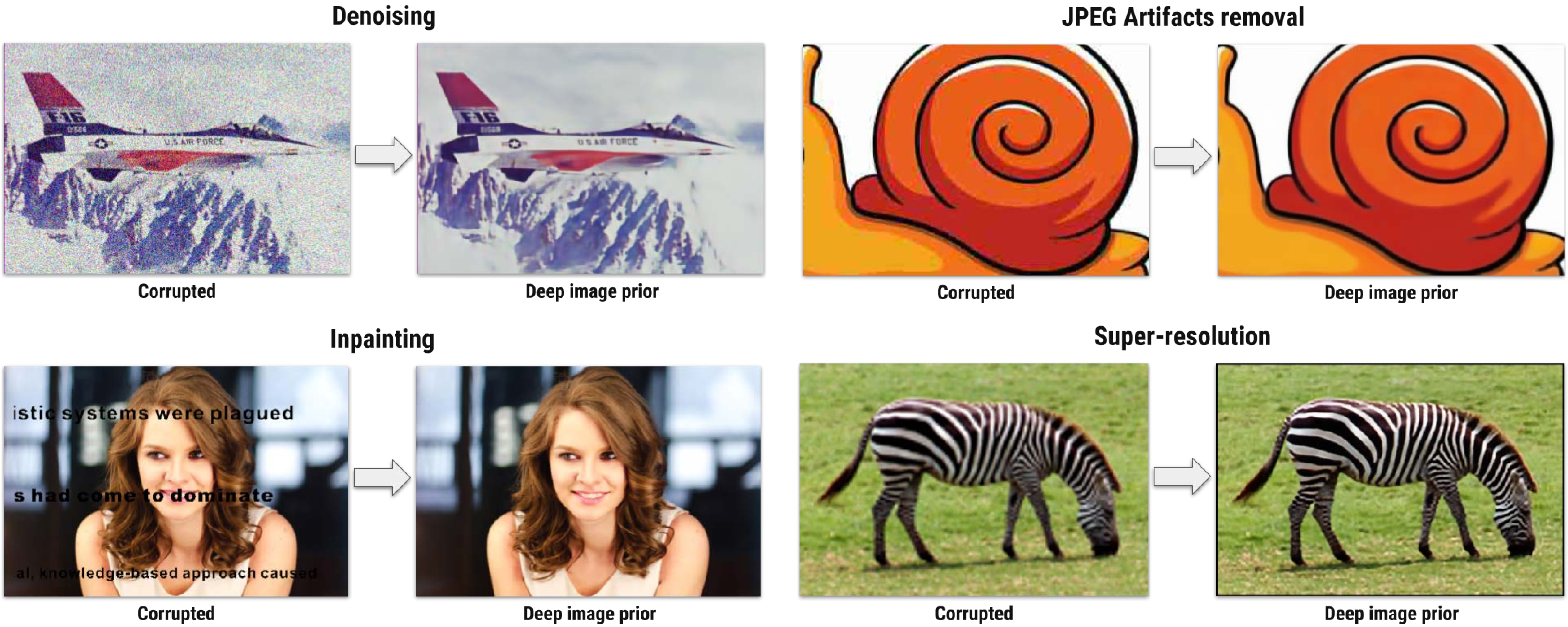
c. Untrained Deep Network (UDN) Reconstruction



Monakhova et al. 2021

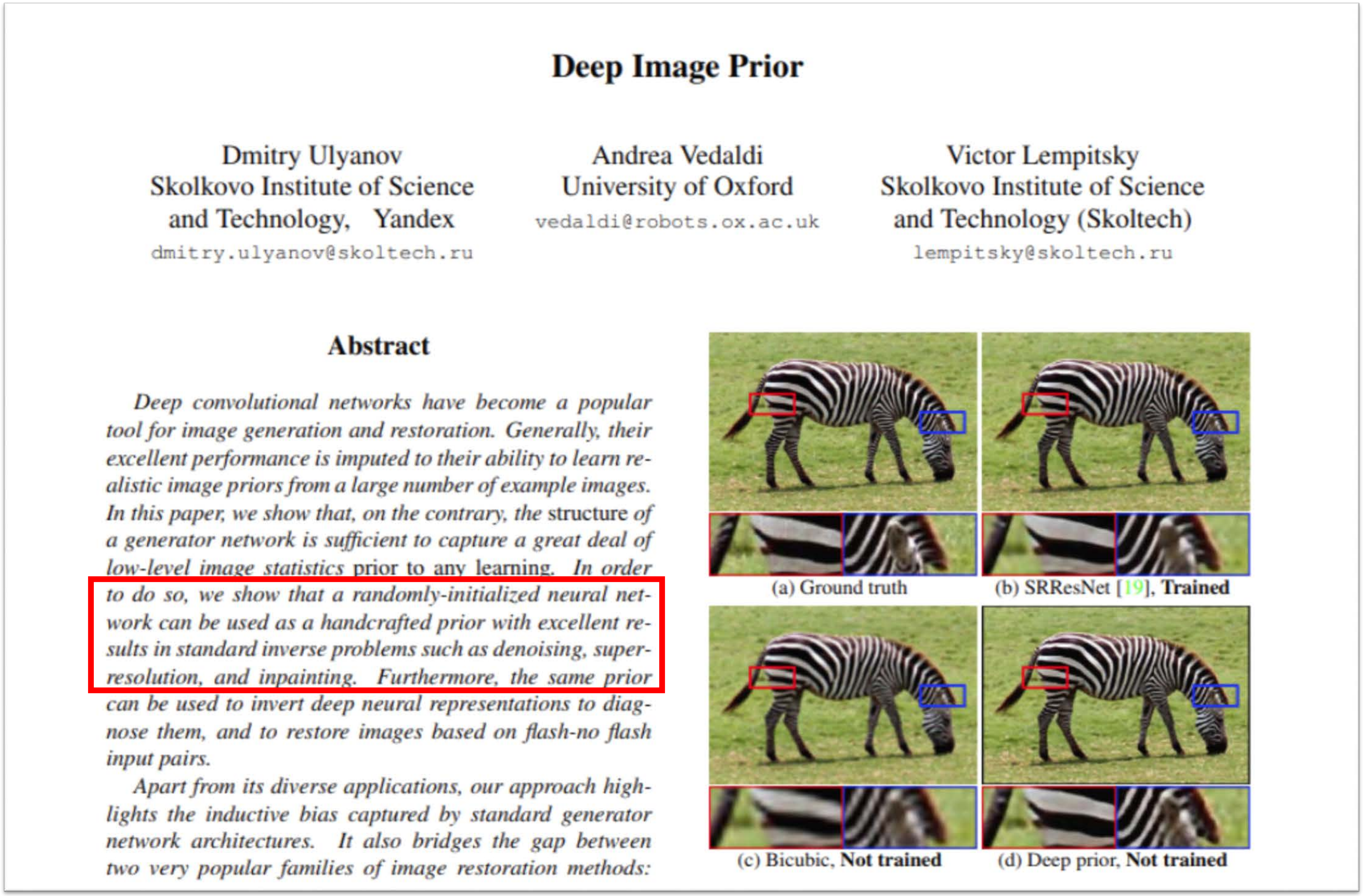
Why NF – Regularization





Ulyanov et al. 2018

Why NF – Regularization



Deep Image Prior

Dmitry Ulyanov
Skolkovo Institute of Science and Technology, Yandex
dmitry.ulyanov@skoltech.ru

Andrea Vedaldi
University of Oxford
vedaldi@robots.ox.ac.uk

Victor Lempitsky
Skolkovo Institute of Science and Technology (Skoltech)
lempitsky@skoltech.ru

Abstract

Deep convolutional networks have become a popular tool for image generation and restoration. Generally, their excellent performance is imputed to their ability to learn realistic image priors from a large number of example images. In this paper, we show that, on the contrary, the structure of a generator network is sufficient to capture a great deal of low-level image statistics prior to any learning. In order to do so, we show that a randomly-initialized neural network can be used as a handcrafted prior with excellent results in standard inverse problems such as denoising, super-resolution, and inpainting. Furthermore, the same prior can be used to invert deep neural representations to diagnose them, and to restore images based on flash-no flash input pairs.

Apart from its diverse applications, our approach highlights the inductive bias captured by standard generator network architectures. It also bridges the gap between two very popular families of image restoration methods:

(a) Ground truth

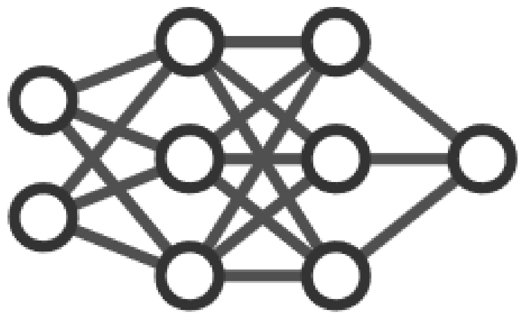
(b) SRResNet [19], **Trained**

(c) Bicubic, **Not trained**

(d) Deep prior, **Not trained**

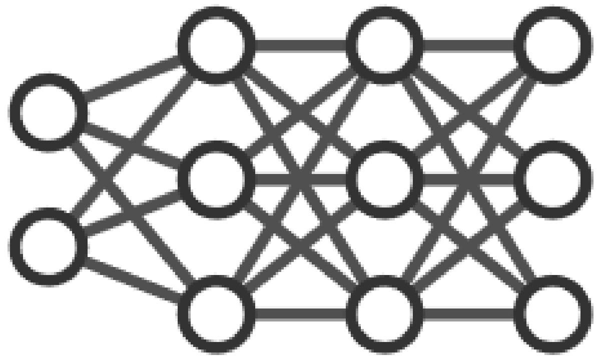
Why NF – Domain Agnostic

2D image



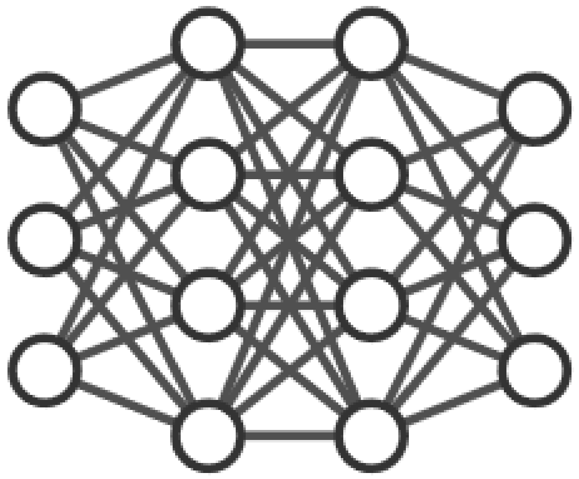
[x,y] → I

2D image



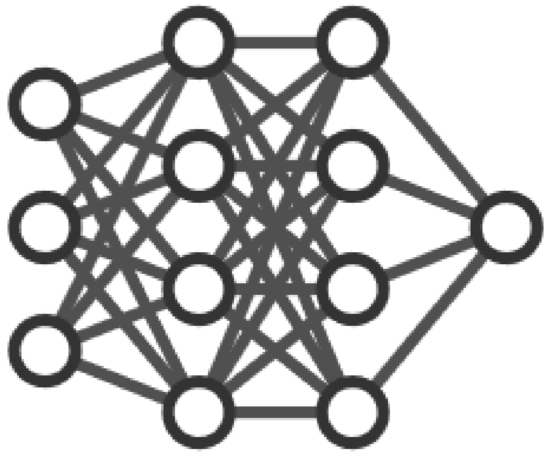
[x,y] → RGB

2D video



[x,y,t] → RGB

3D Image



[x,y,z] → I

NF are domain-agnostic and can model arbitrary quantities.

Challenges

1. Spectral Bias

2. Prior learning

3. Manipulation

4. Computation

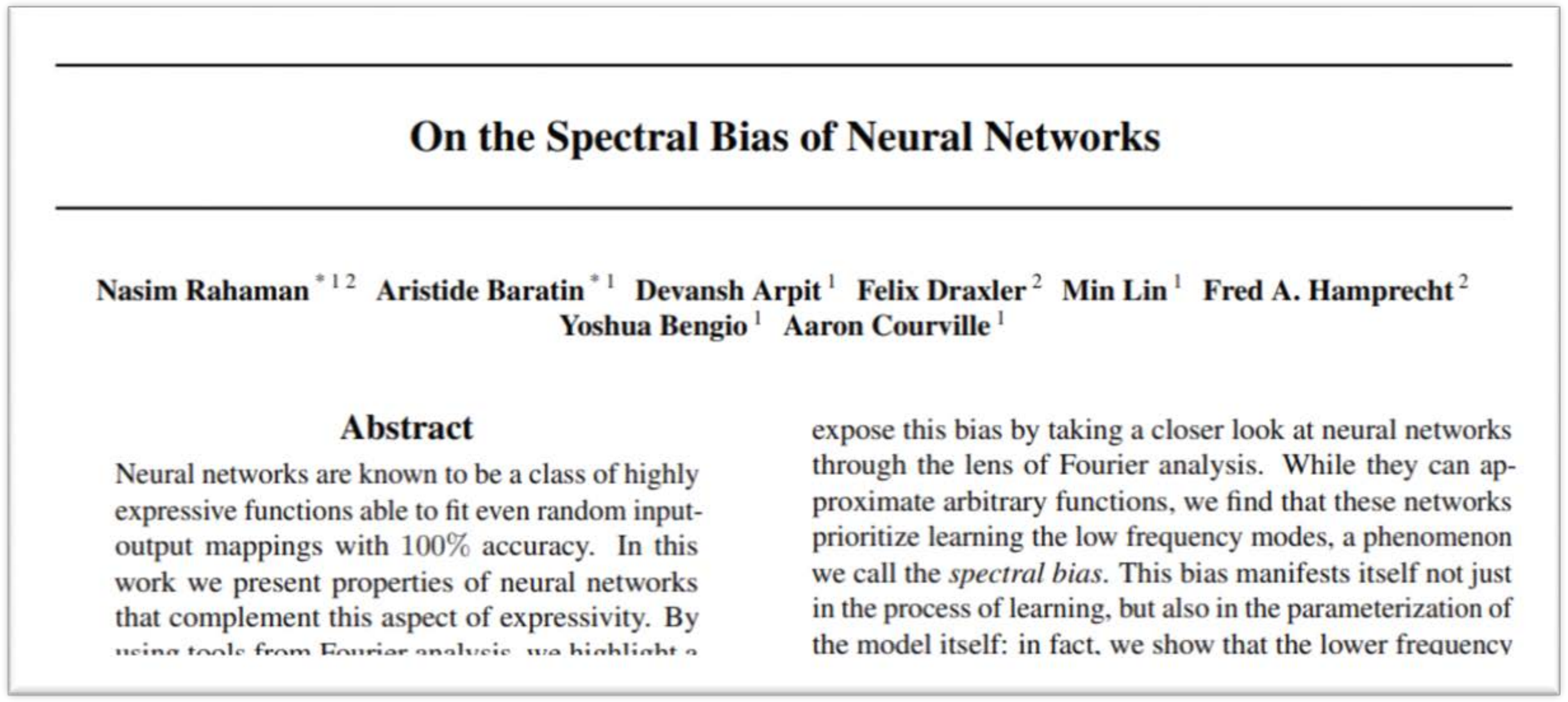
Challenges

1. Spectral Bias

2. Prior learning

3. Manipulation

4. Computation



On the Spectral Bias of Neural Networks

Nasim Rahaman [*] [1 2] Aristide Baratin [*] [1] Devansh Arpit [1] Felix Draxler [2] Min Lin [1] Fred A. Hamprecht [2]
Yoshua Bengio [1] Aaron Courville [1]

Abstract

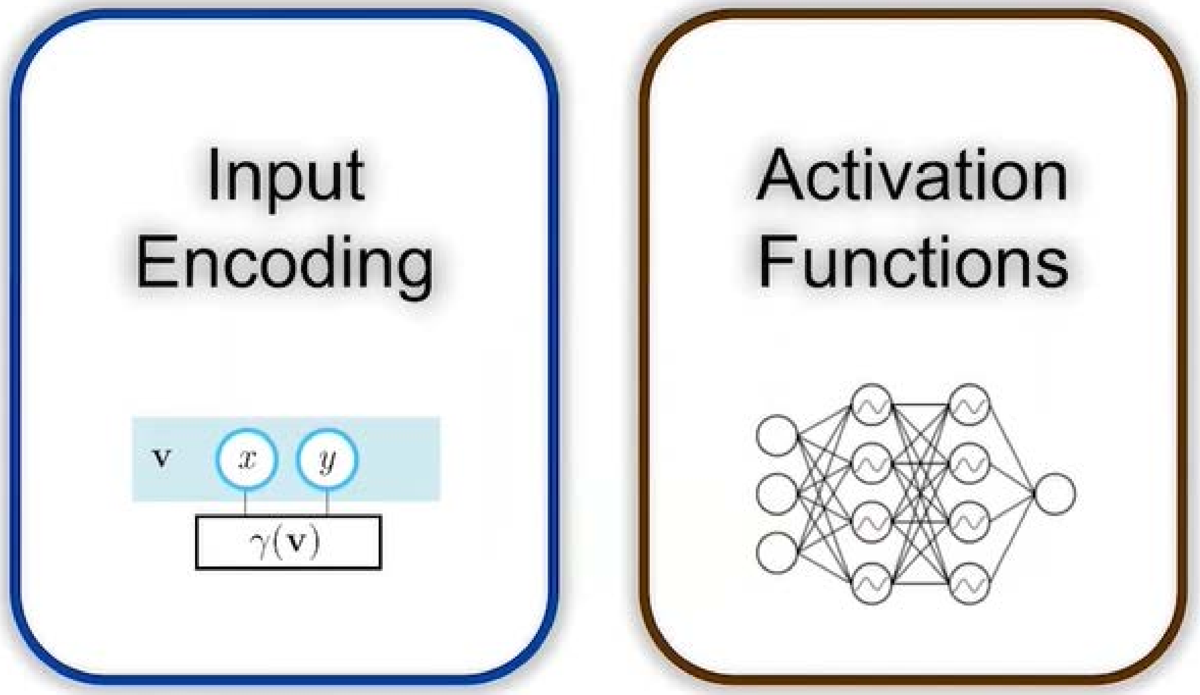
Neural networks are known to be a class of highly expressive functions able to fit even random input-output mappings with 100% accuracy. In this work we present properties of neural networks that complement this aspect of expressivity. By using tools from Fourier analysis, we highlight a

expose this bias by taking a closer look at neural networks through the lens of Fourier analysis. While they can approximate arbitrary functions, we find that these networks prioritize learning the low frequency modes, a phenomenon we call the *spectral bias*. This bias manifests itself not just in the process of learning, but also in the parameterization of the model itself: in fact, we show that the lower frequency

DALL-E

Challenges

1. Spectral Bias
2. Prior learning
3. Manipulation
4. Computation



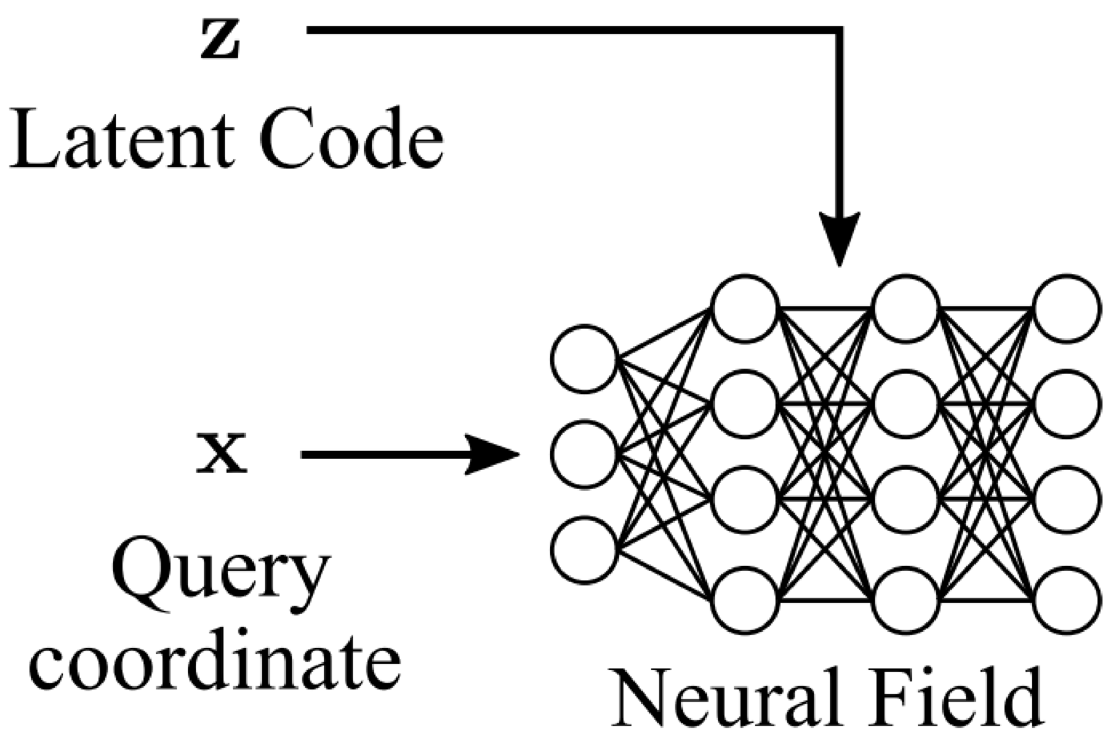
Input Encoding

Activation Functions

NF Tutorial, CVPR 2022

DALL-E

Challenges



Xie et al., 2022

1. Spectral Bias

2. Prior learning

3. Manipulation

4. Computation

DALL-E

Challenges

1. Spectral Bias
2. Prior learning
3. Manipulation
4. Computation

Neural fields
have limited tools for editing
and manipulation.

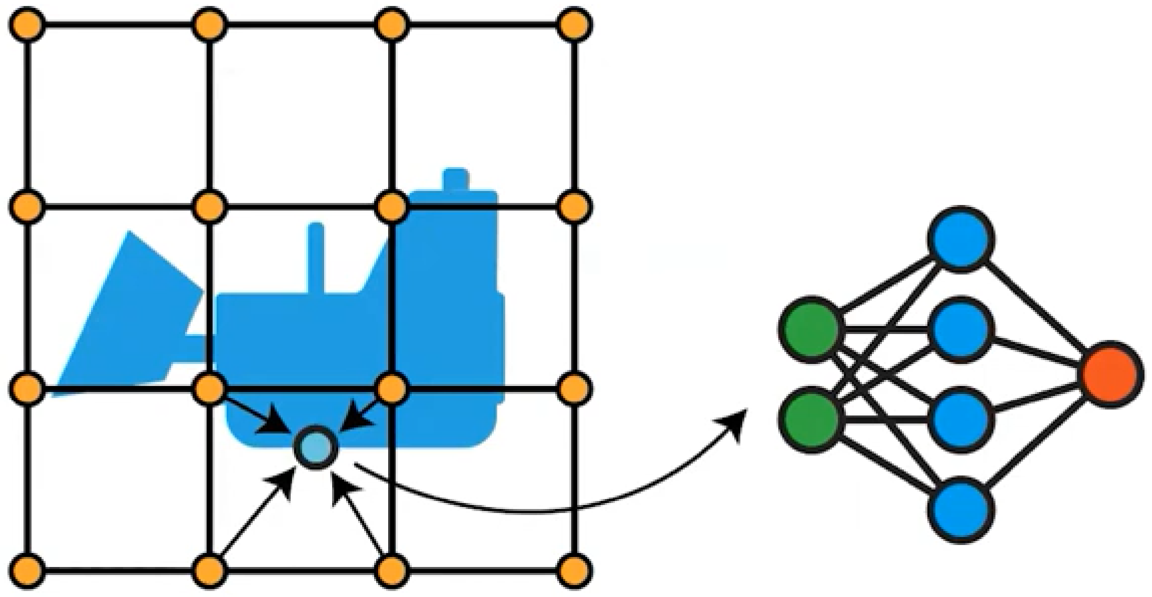
Challenges

1. Spectral Bias

2. Prior learning

3. Manipulation

4. Computation

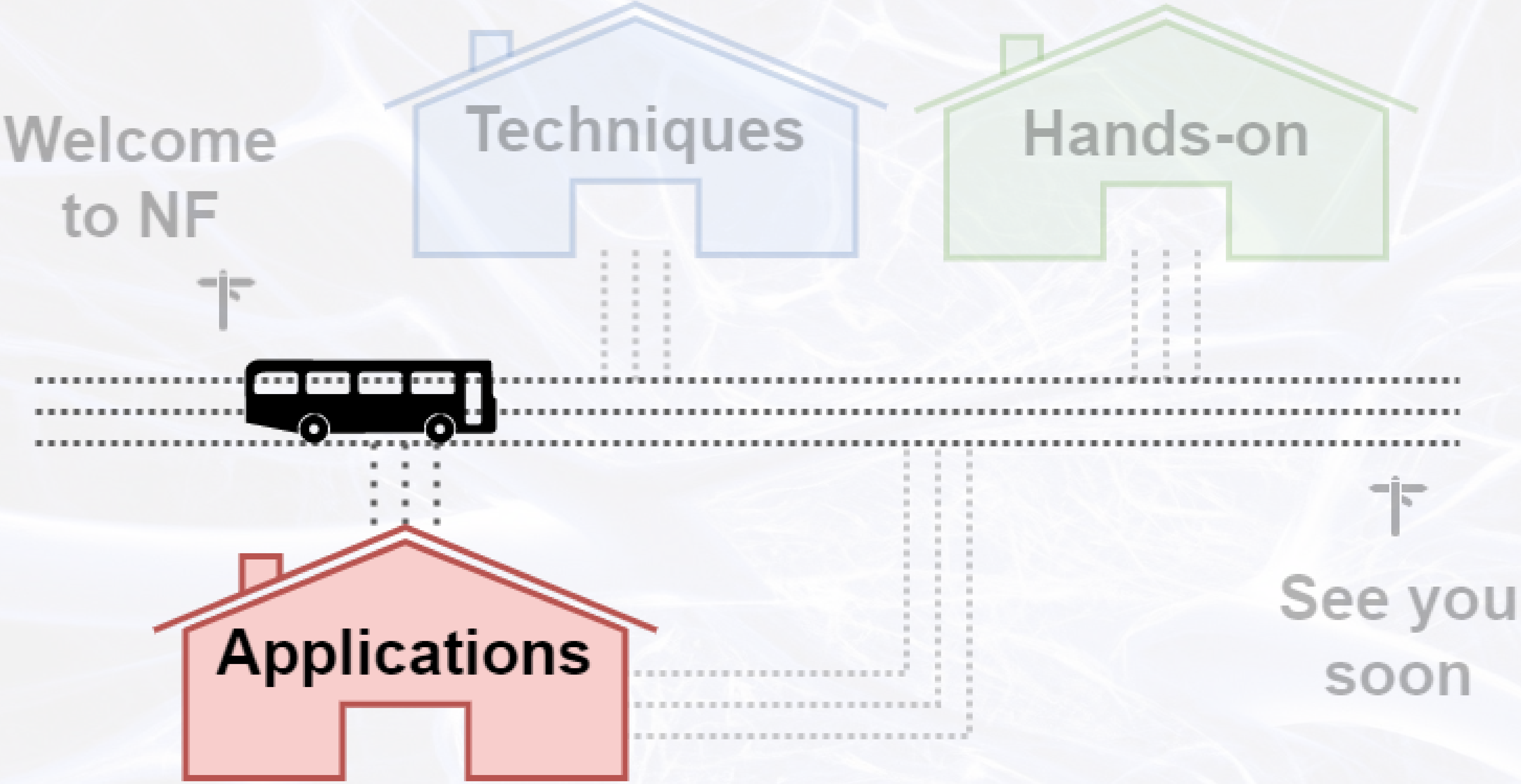


Hybrid representation

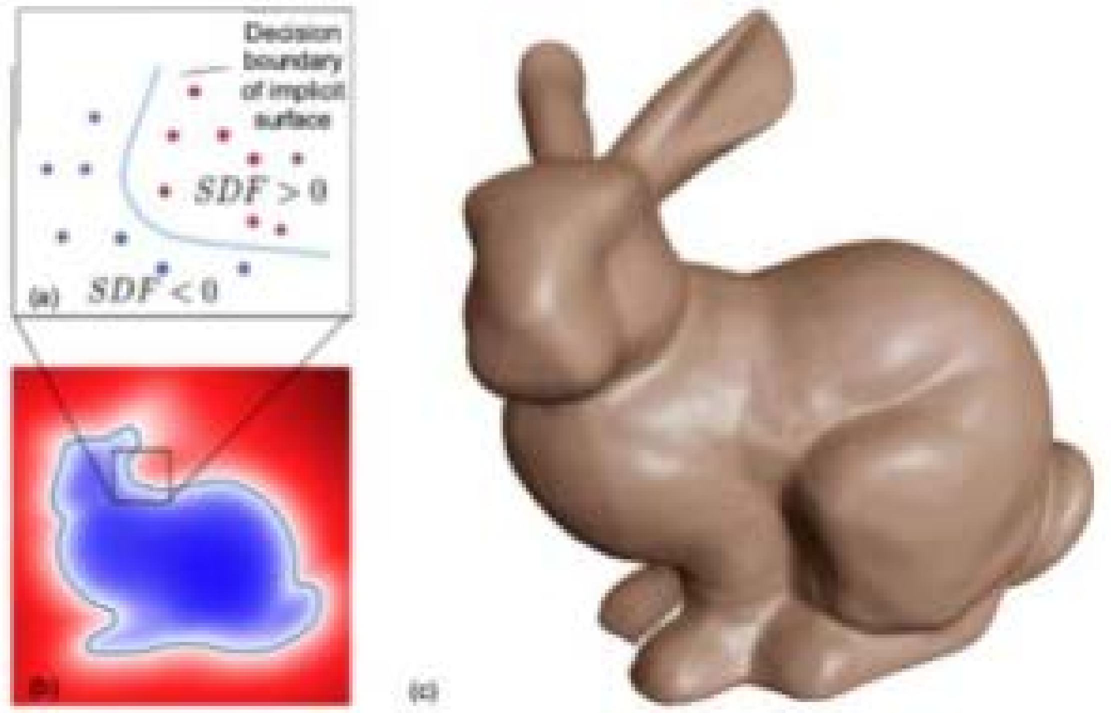
NF Tutorial, CVPR 2022

DALL-E

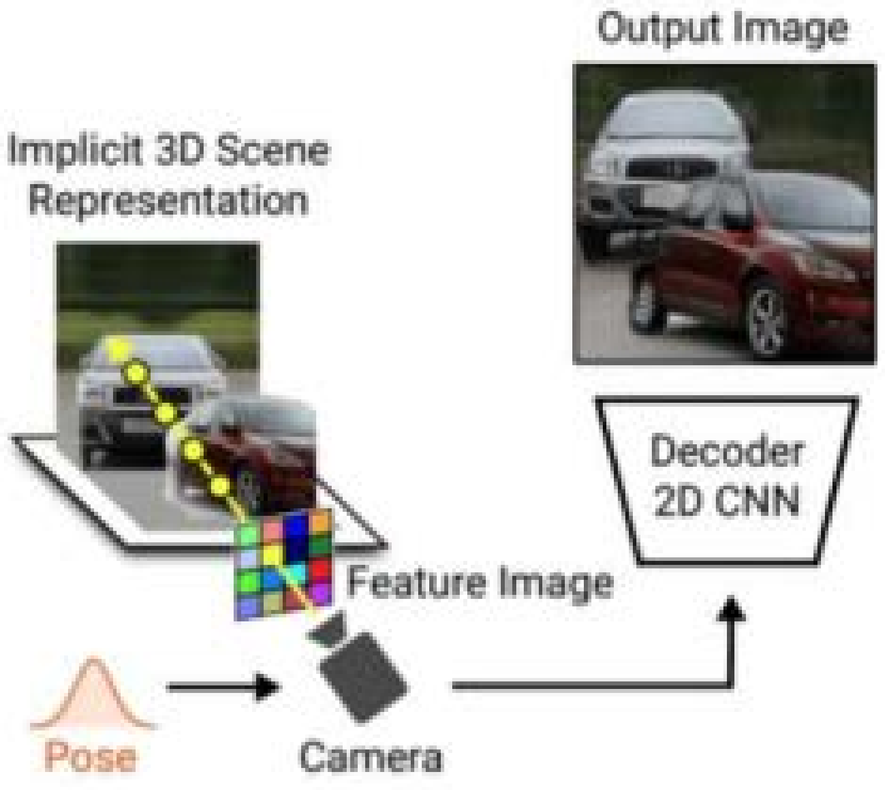
Let's go on a tour to NF



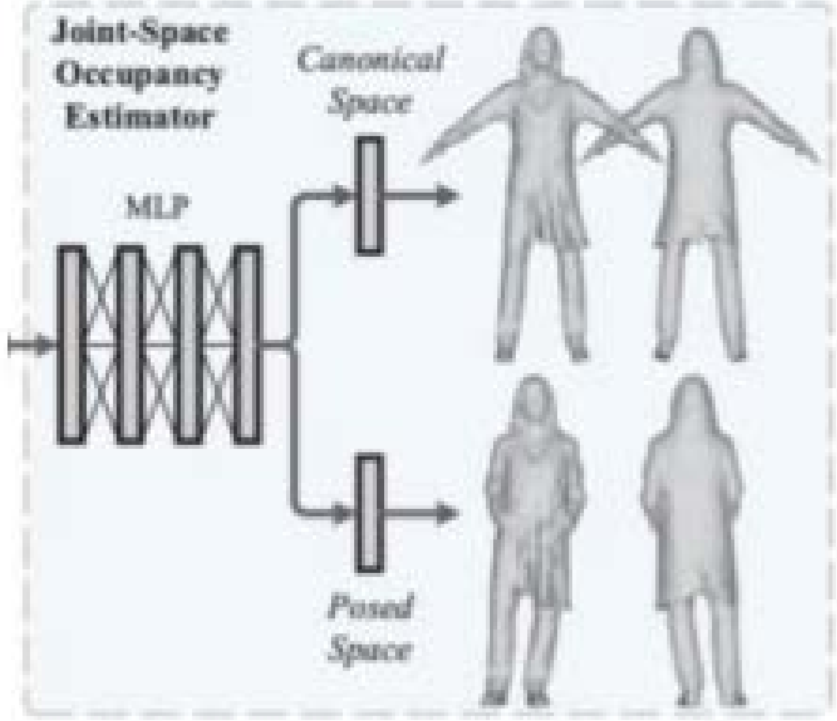
Applications



2D and 3D Reconstruction

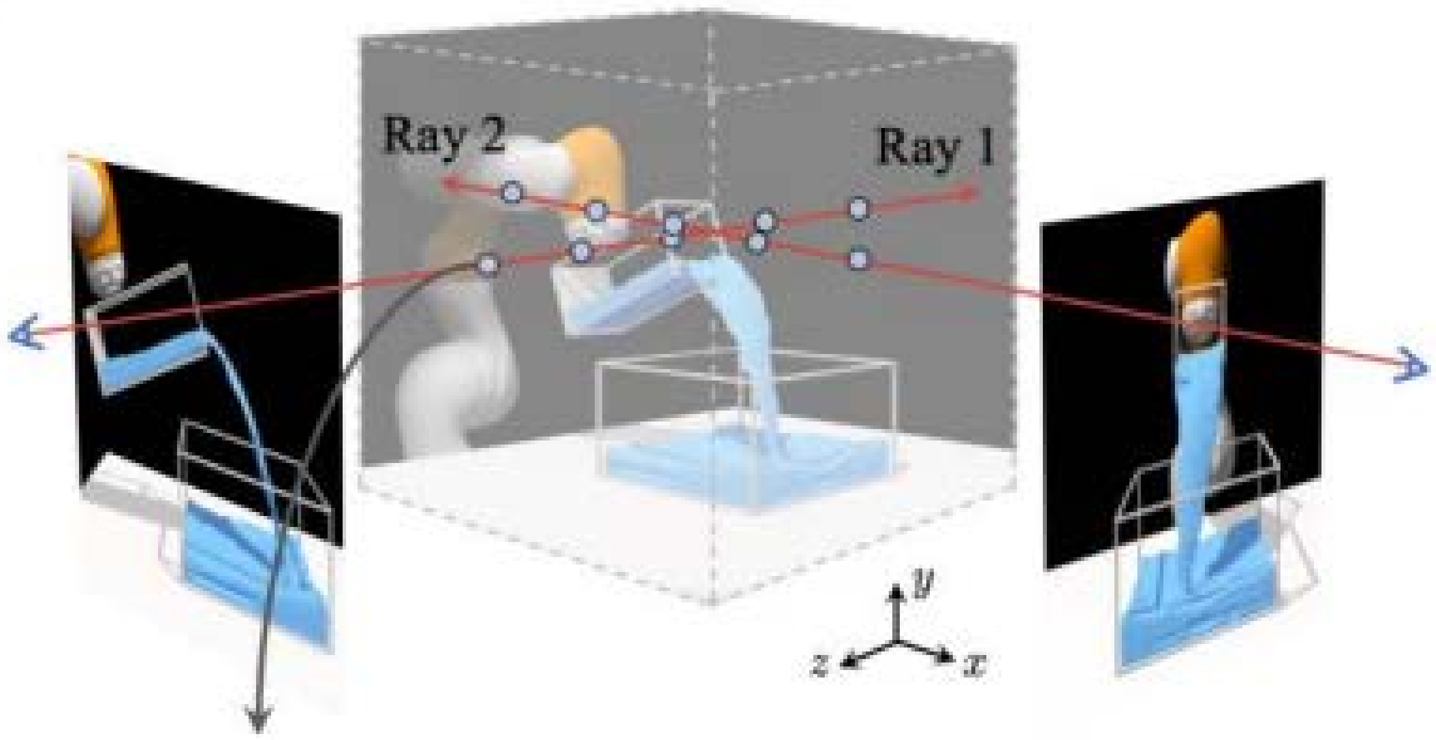


Generative Models

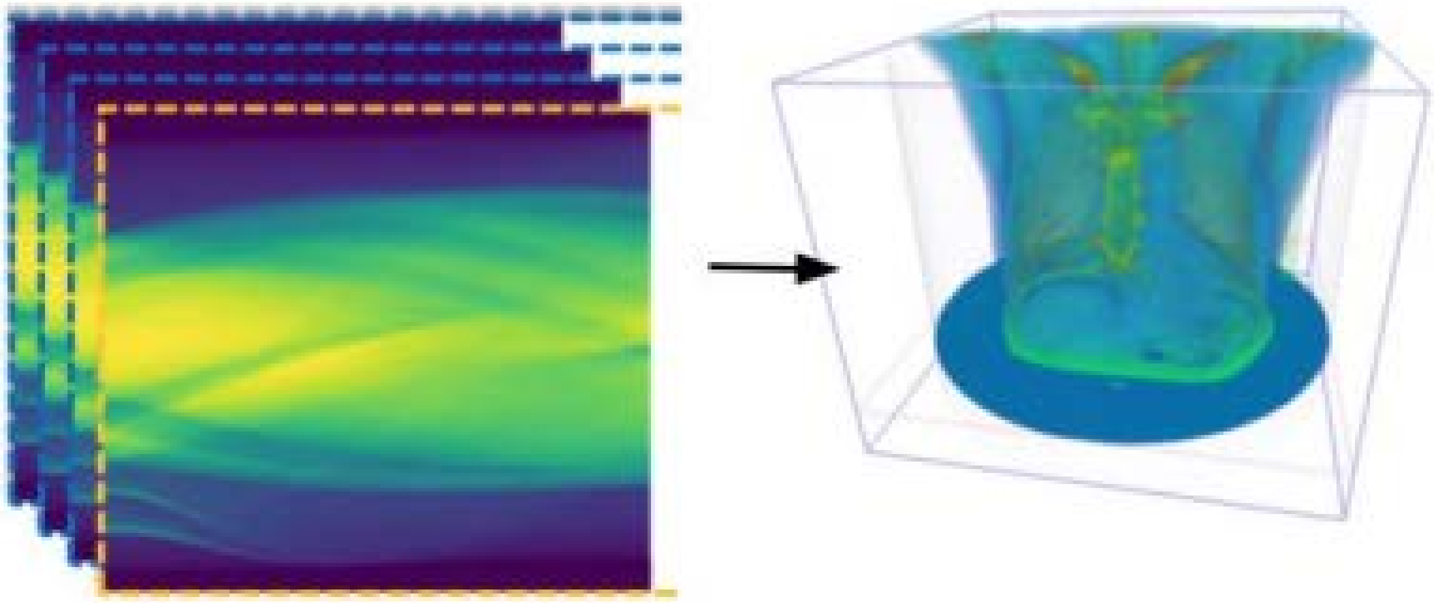


Digital Humans

Compression



Robotics



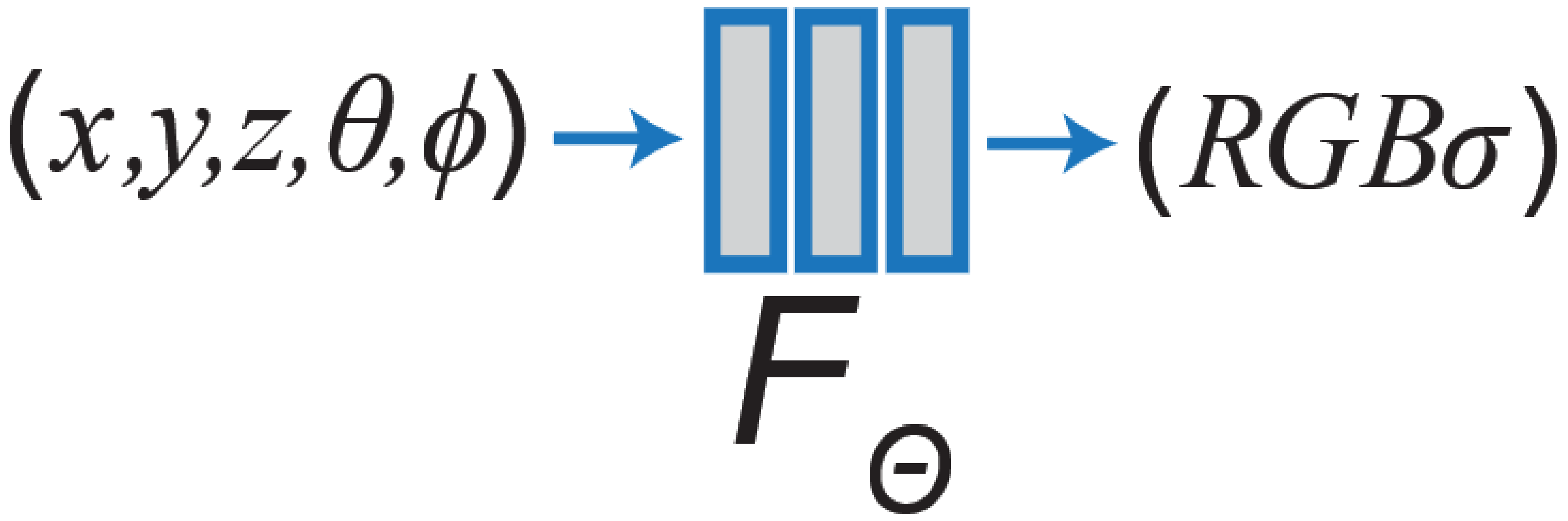
...and Beyond!

Xie et al., 2022

NeRF

Representing Scenes as Neural Radiance Fields
for View Synthesis
ECCV 2020, Tancik et al.

Want more…

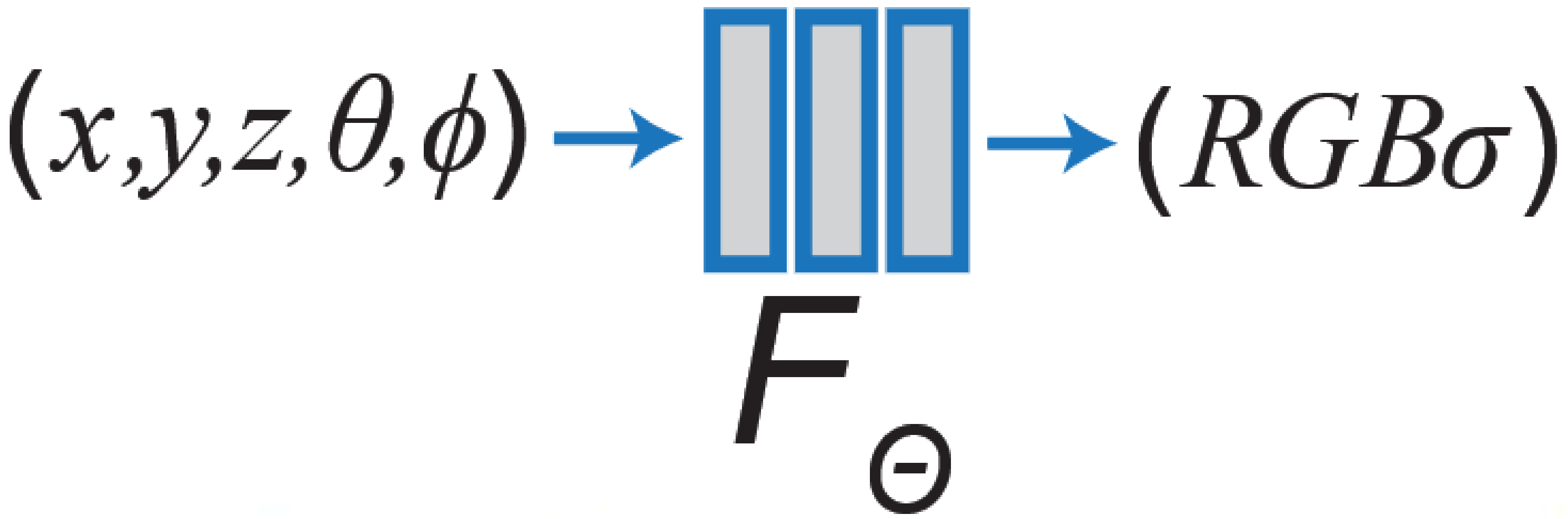


First continuous neural scene representation
that can render high-resolution photorealistic novel views of real objects
and scenes from RGB images captured in natural settings.

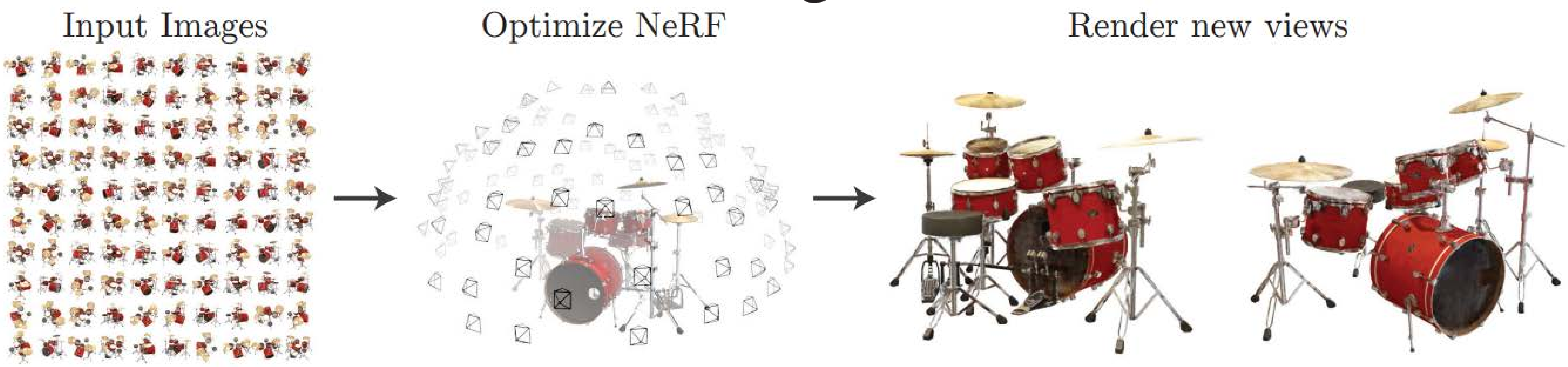
NeRF

Representing Scenes as Neural Radiance Fields
for View Synthesis
ECCV 2020, Tancik et al.

Want more…



$$(x,y,z,\theta,\phi) \rightarrow F_{\Theta} \rightarrow (RGB\sigma)$$



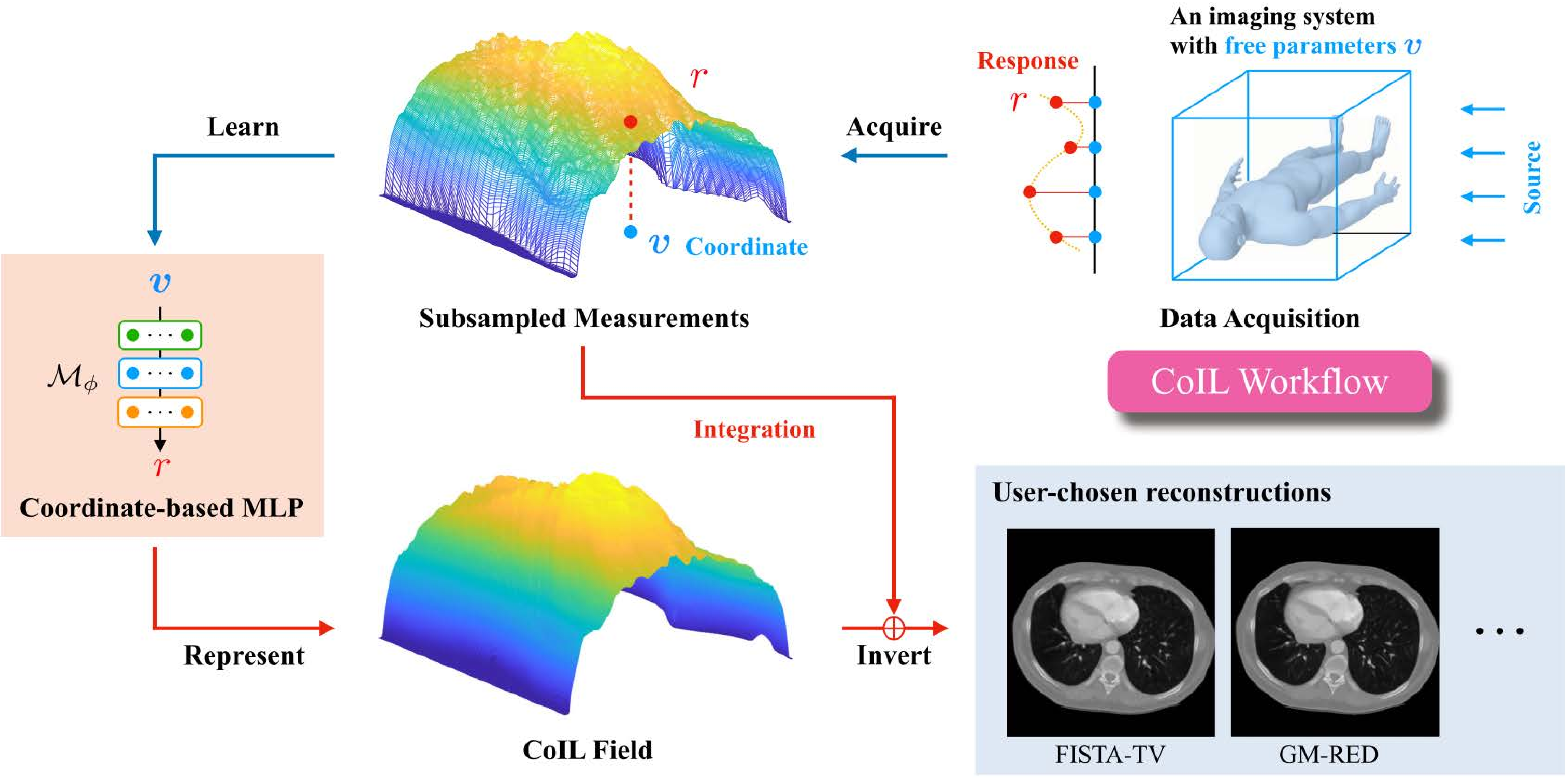
NeRF

Representing Scenes as Neural Radiance Fields for View Synthesis
ECCV 2020, Tancik et al.

Want more...

COIL

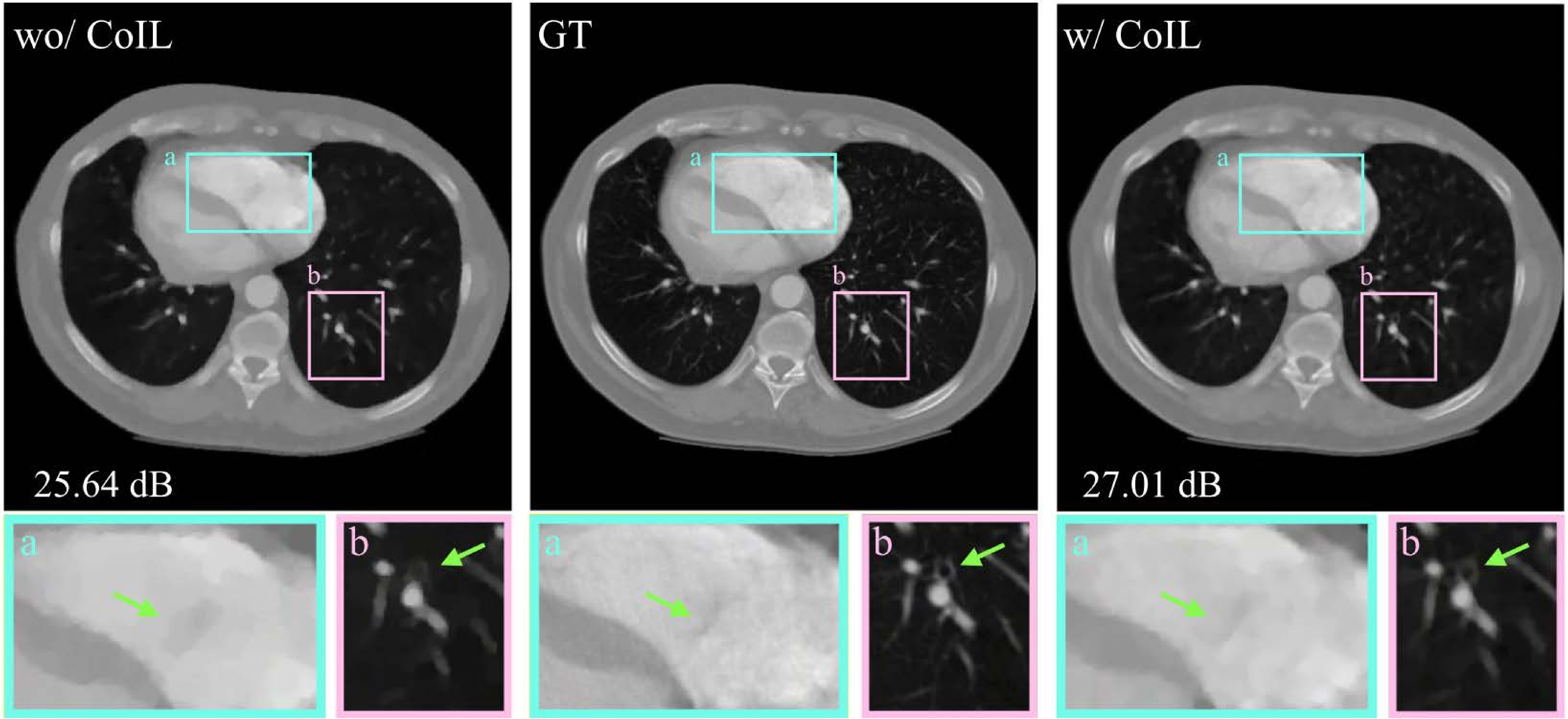
CoIL: Coordinate-Based Internal Learning for Tomographic Imaging
IEEE TCI 2021, Yu, et al



Applications

COIL

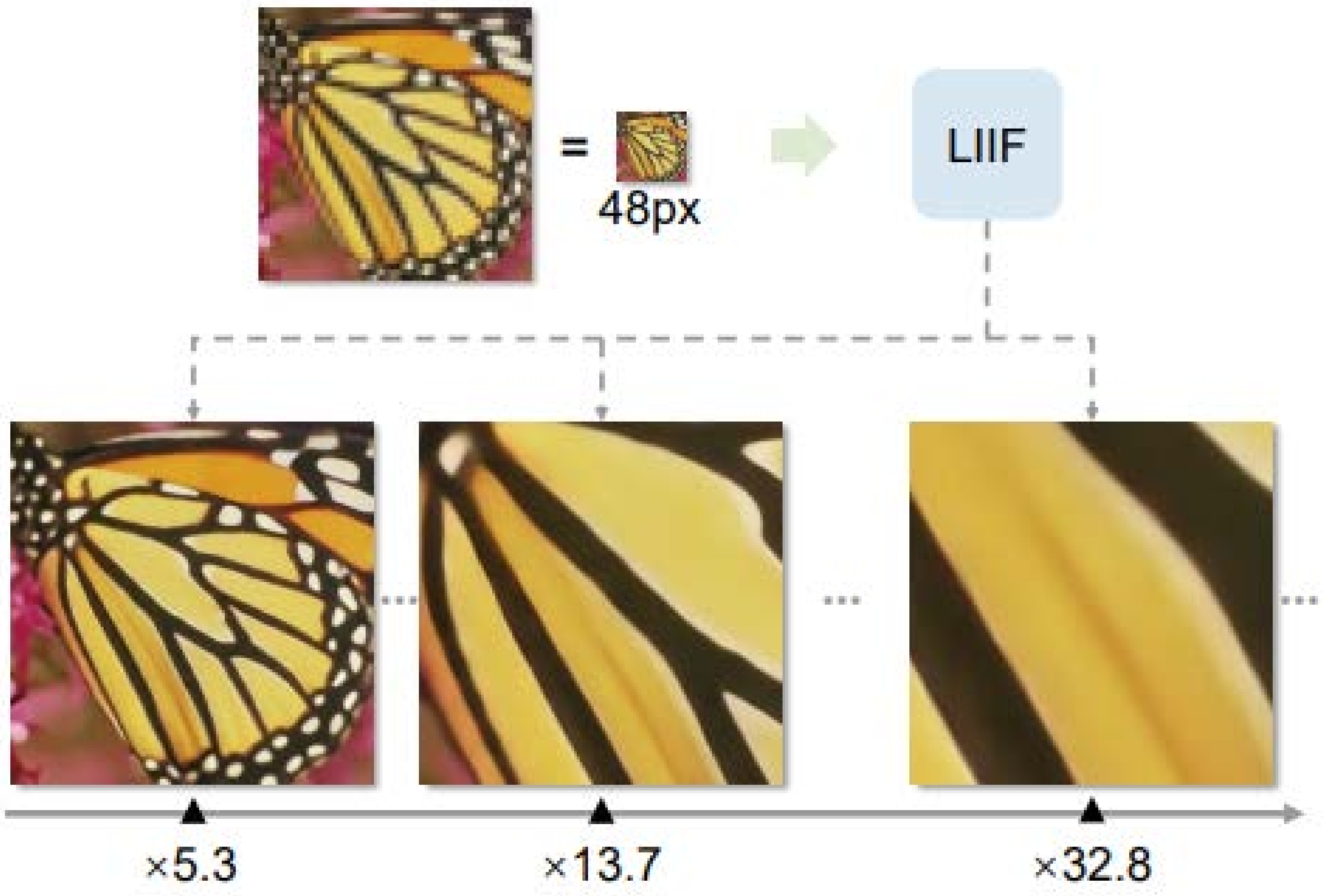
CoIL: Coordinate-Based Internal Learning for Tomographic Imaging
IEEE TCI 2021, Yu, et al



LIIF

Learning Continuous Image Representation
with Local Implicit Image Function
CVPR 2021, Chen et al.

Want more…



"While the visual world is presented in a continuous manner, machines store and see the images in a discrete way with 2D arrays of pixels."

Applications

LIIF

Learning Continuous Image Representation with Local Implicit Image Function CVPR 2021, Chen et al.

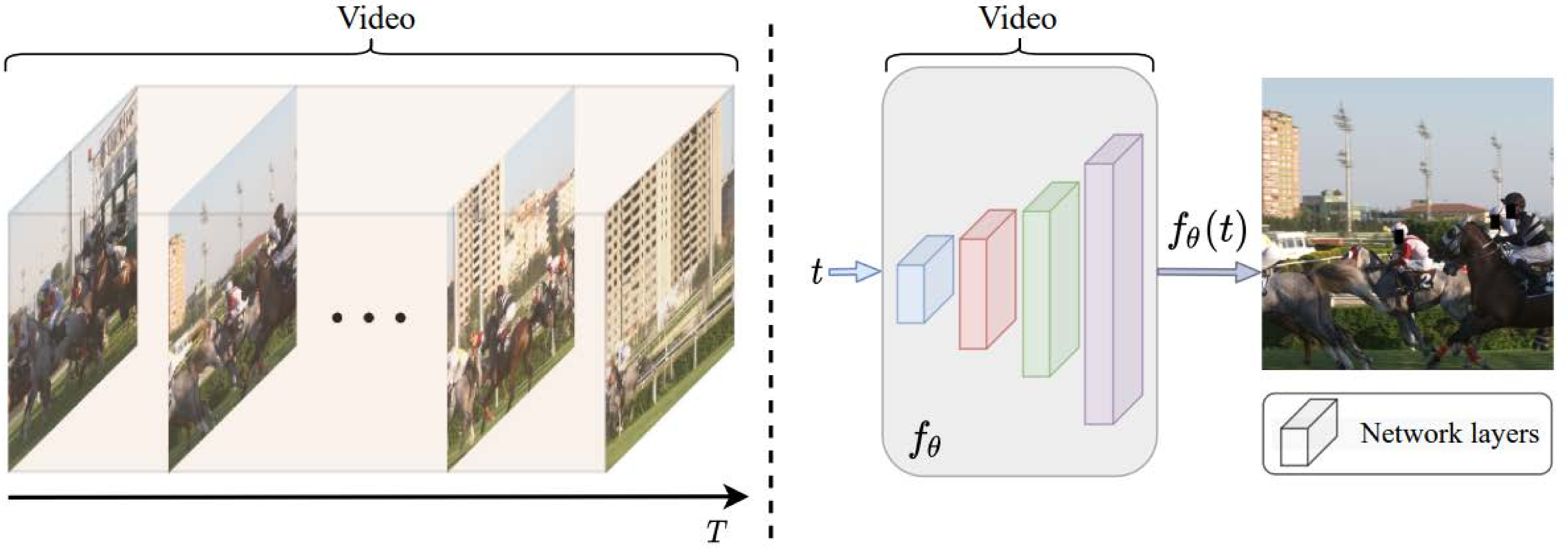
[Want more…]

Input (360px) Pixels Bilinear LIIF

NeRV

Neural Representations for Videos
NeurIPS 2021, Chen et al.

Want more…



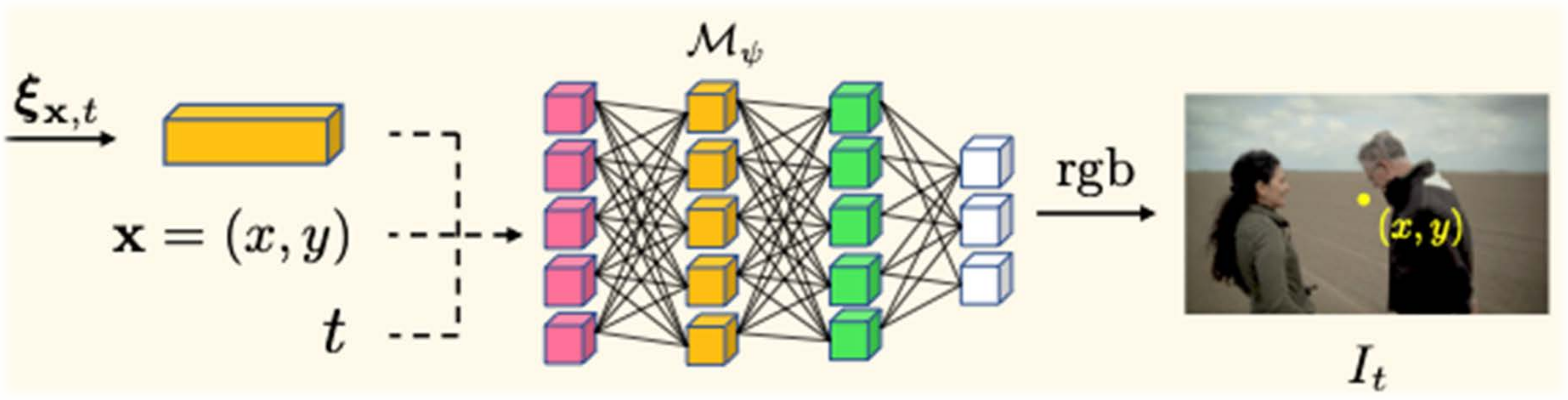
(a) Explicit representations for videos (e.g., HEVC) (b) Neural implicit representations for videos (e.g., NeRV)

Applications

CURE

CURE: Learning Cross-Video Neural Representations for High-Quality Frame Interpolation
ECCV 2022, Shangguan et al.

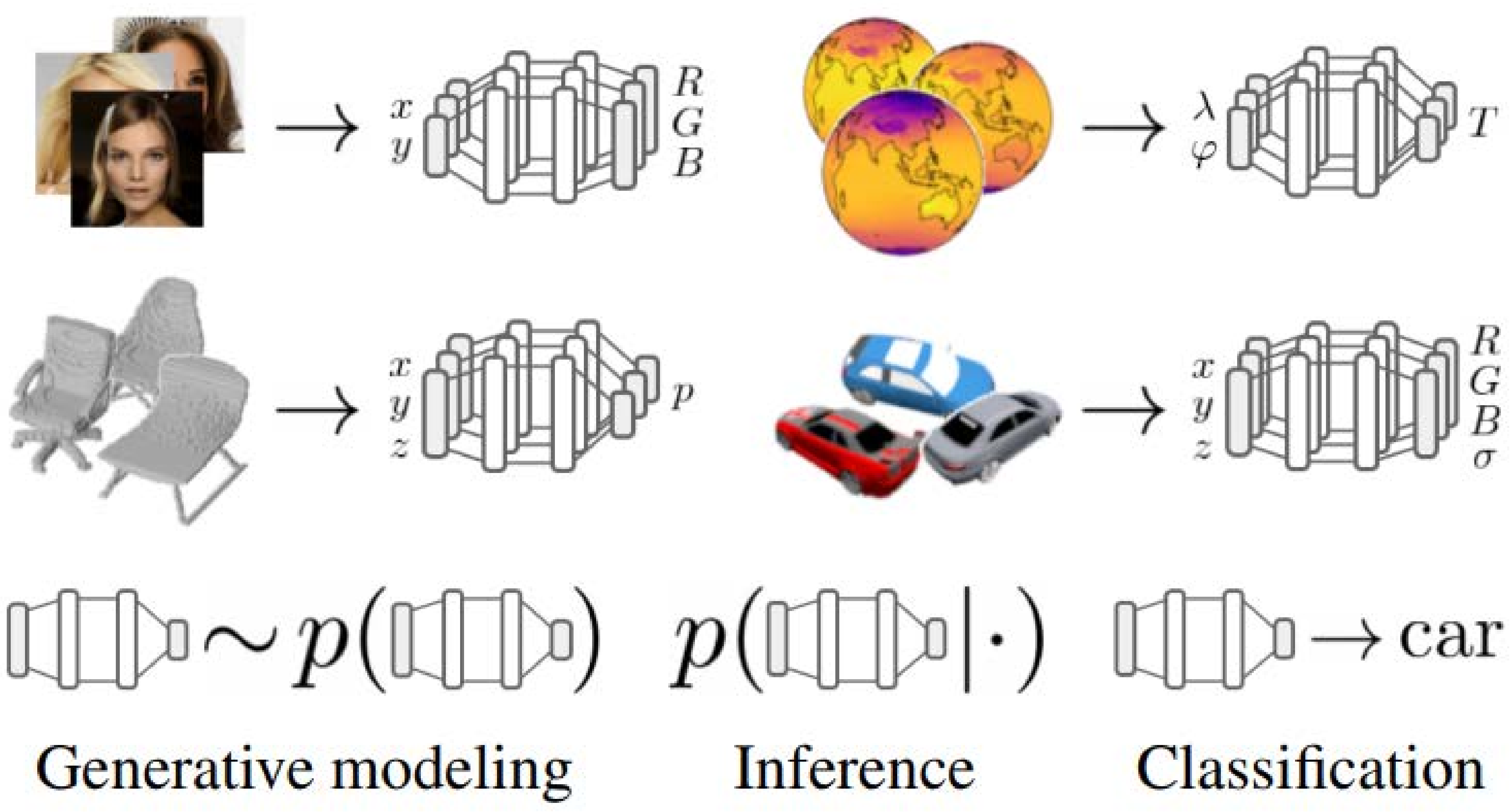
Want more…

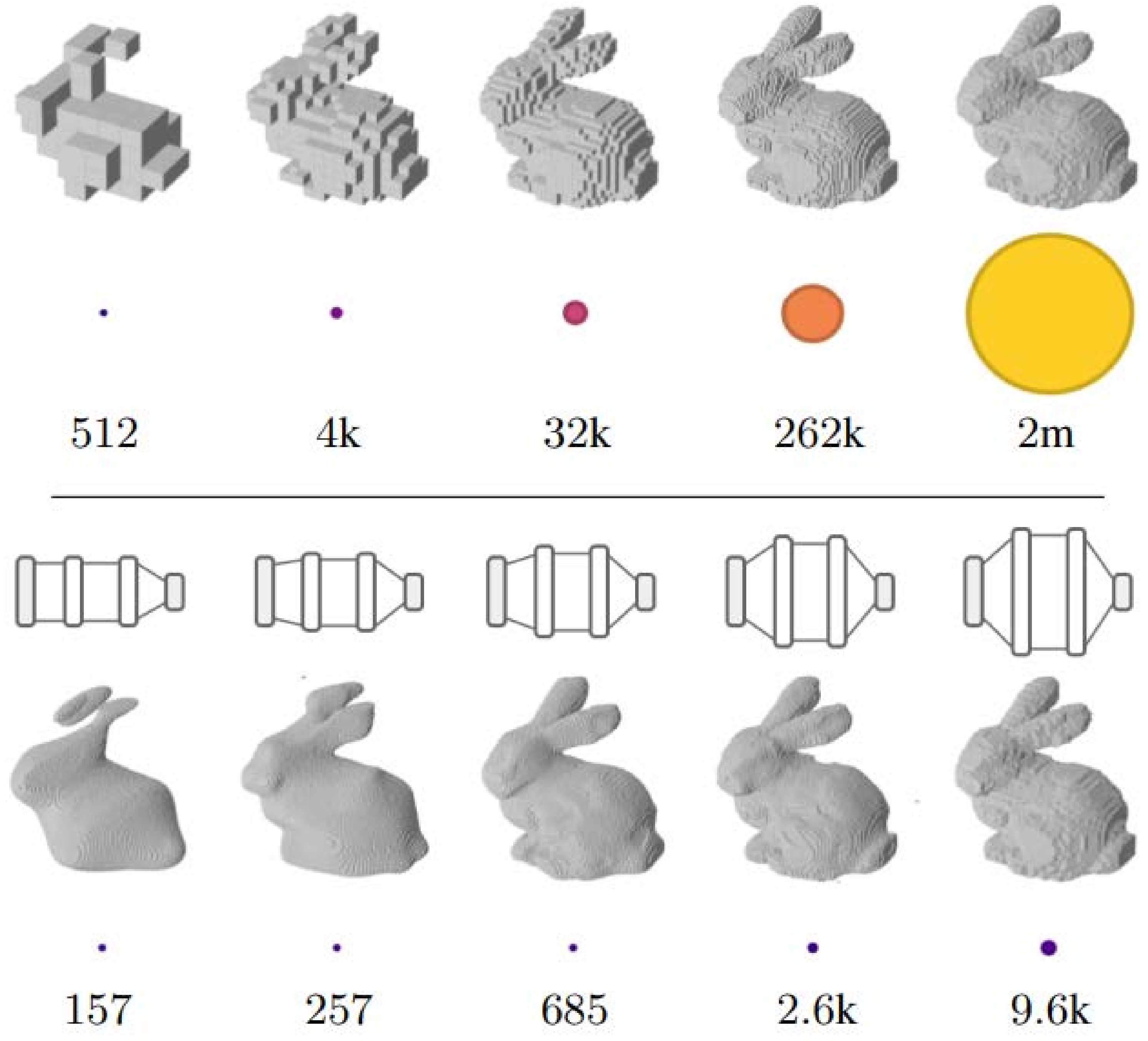


Functa

From data to functa: Your data point is a function and you can treat it like one
ICML 2022, Dupont et al.

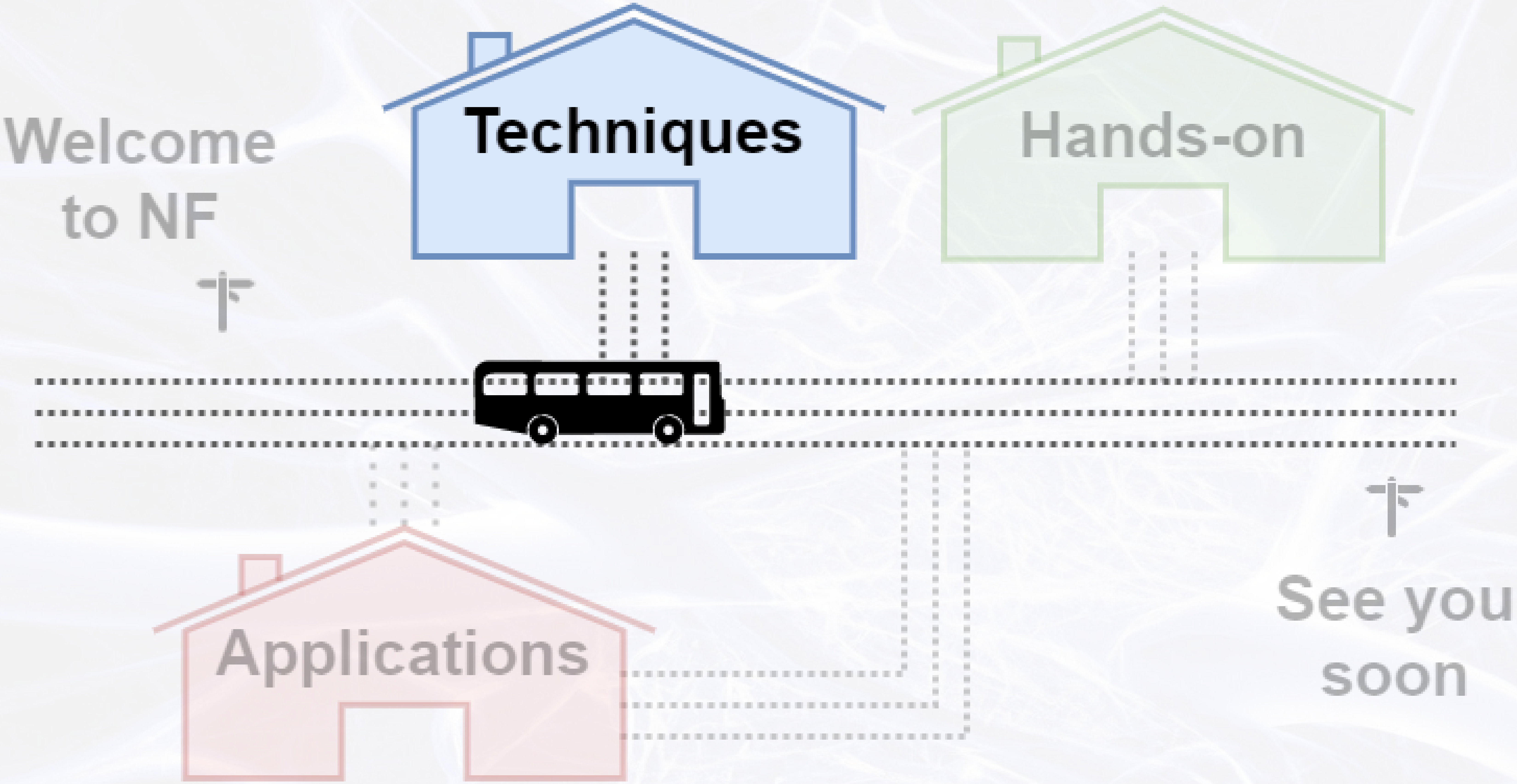
Want more...





by DeepMind

Let's go on a tour to NF



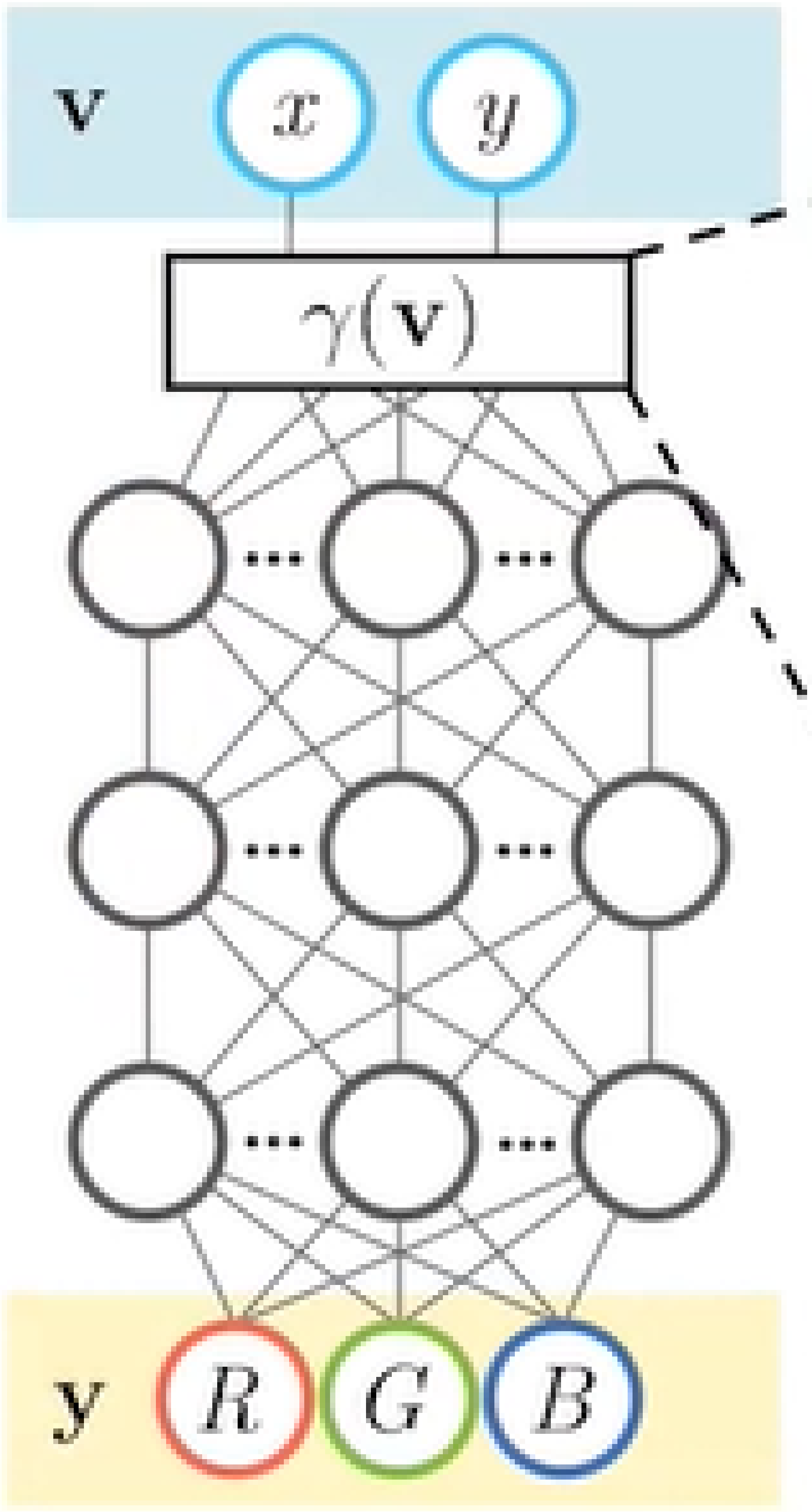
Techniques → Challenges

Positional encoding
Activation functions → Spectral Bias

Conditioning → Prior learning

Hybrid representations → Computation

Positional encoding



$$\gamma(\mathbf{x}) = [\gamma_1(\mathbf{x}), \gamma_2(\mathbf{x}), \ldots, \gamma_m(\mathbf{x})]$$

$$\gamma_{(2i)}(x) = sin(2^{i-1}\pi x),$$

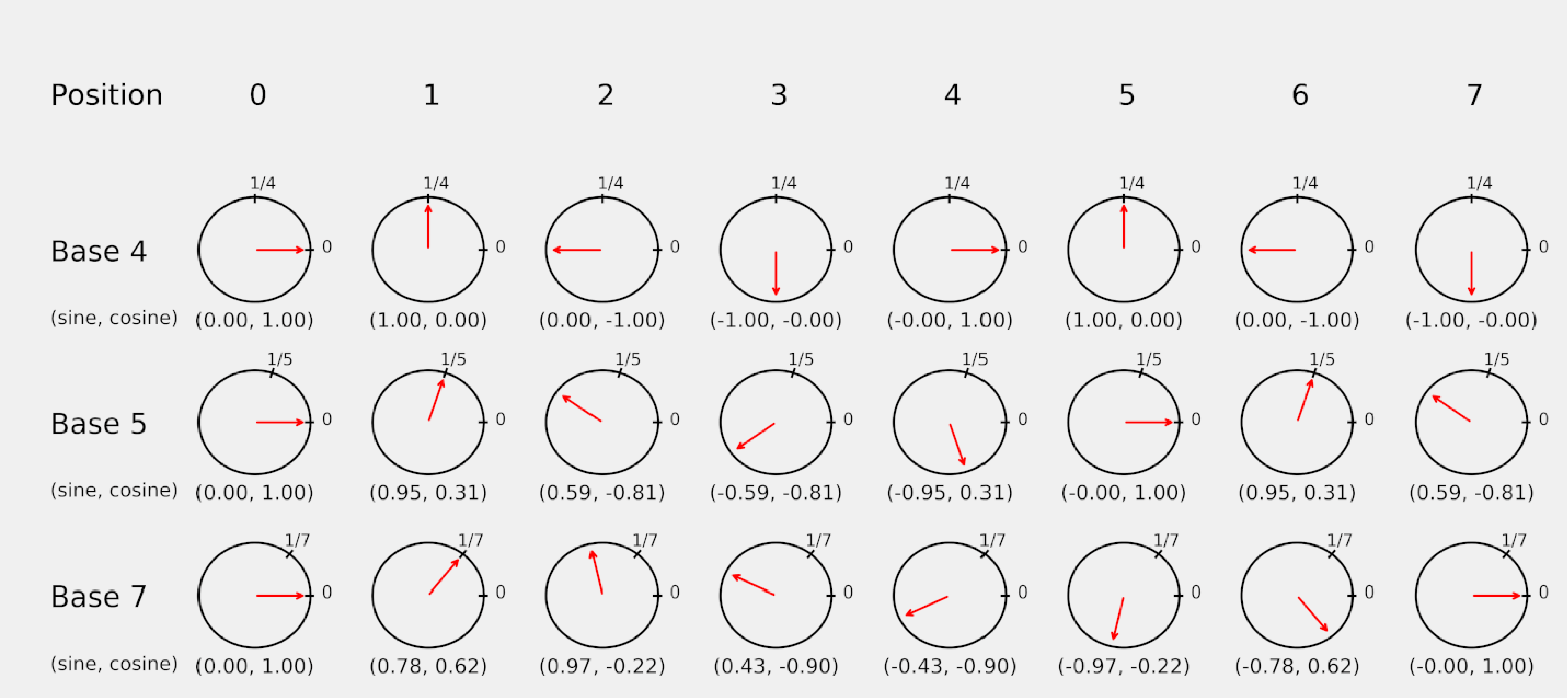
$$\gamma_{(2i+1)}(x) = cos(2^{i-1}\pi x)$$

[Vaswani et al. 2017]

Positional encoding

Godoy, Deep Learning with PyTorch Step-by-Step

Want more…



Position	0	1	2	3	4	5	6	7
Base 4 (sine, cosine)	(0.00, 1.00)	(1.00, 0.00)	(0.00, -1.00)	(-1.00, -0.00)	(-0.00, 1.00)	(1.00, 0.00)	(0.00, -1.00)	(-1.00, -0.00)
Base 5 (sine, cosine)	(0.00, 1.00)	(0.95, 0.31)	(0.59, -0.81)	(-0.59, -0.81)	(-0.95, 0.31)	(-0.00, 1.00)	(0.95, 0.31)	(0.59, -0.81)
Base 7 (sine, cosine)	(0.00, 1.00)	(0.78, 0.62)	(0.97, -0.22)	(0.43, -0.90)	(-0.43, -0.90)	(-0.97, -0.22)	(-0.78, 0.62)	(-0.00, 1.00)

Positional encoding

Godoy, Deep Learning with PyTorch Step-by-Step

Want more...

Position	0	1	2	3	4	5	6	7
sine (base 4)	0.00	1.00	0.00	-1.00	0.00	1.00	0.00	-1.00
cosine (base 4)	1.00	0.00	-1.00	0.00	1.00	0.00	-1.00	0.00
sine (base 5)	0.00	0.95	0.59	-0.59	-0.95	0.00	0.95	0.59
cosine (base 5)	1.00	0.31	-0.81	-0.81	0.31	1.00	0.31	-0.81
sine (base 7)	0.00	0.78	0.97	0.43	-0.43	-0.97	-0.78	0.00
cosine (base 7)	1.00	0.62	-0.22	-0.90	-0.90	-0.22	0.62	1.00

3		2		Diff
-1.00		0.00		-1.00
0.00		-1.00		1.00
-0.59	-	0.59	=	-1.18
-0.81		-0.81		0.00
0.43		0.97		-0.54
-0.90		-0.22		-0.68

Distance = ||Diff|| = 2.03

4		3		Diff
0.00		-1.00		1.00
1.00		0.00		1.00
-0.95	-	-0.59	=	-0.36
0.31		-0.81		1.12
-0.43		0.43		-0.87
-0.90		-0.90		0.00

Distance = ||Diff|| = 2.03

Positional encoding

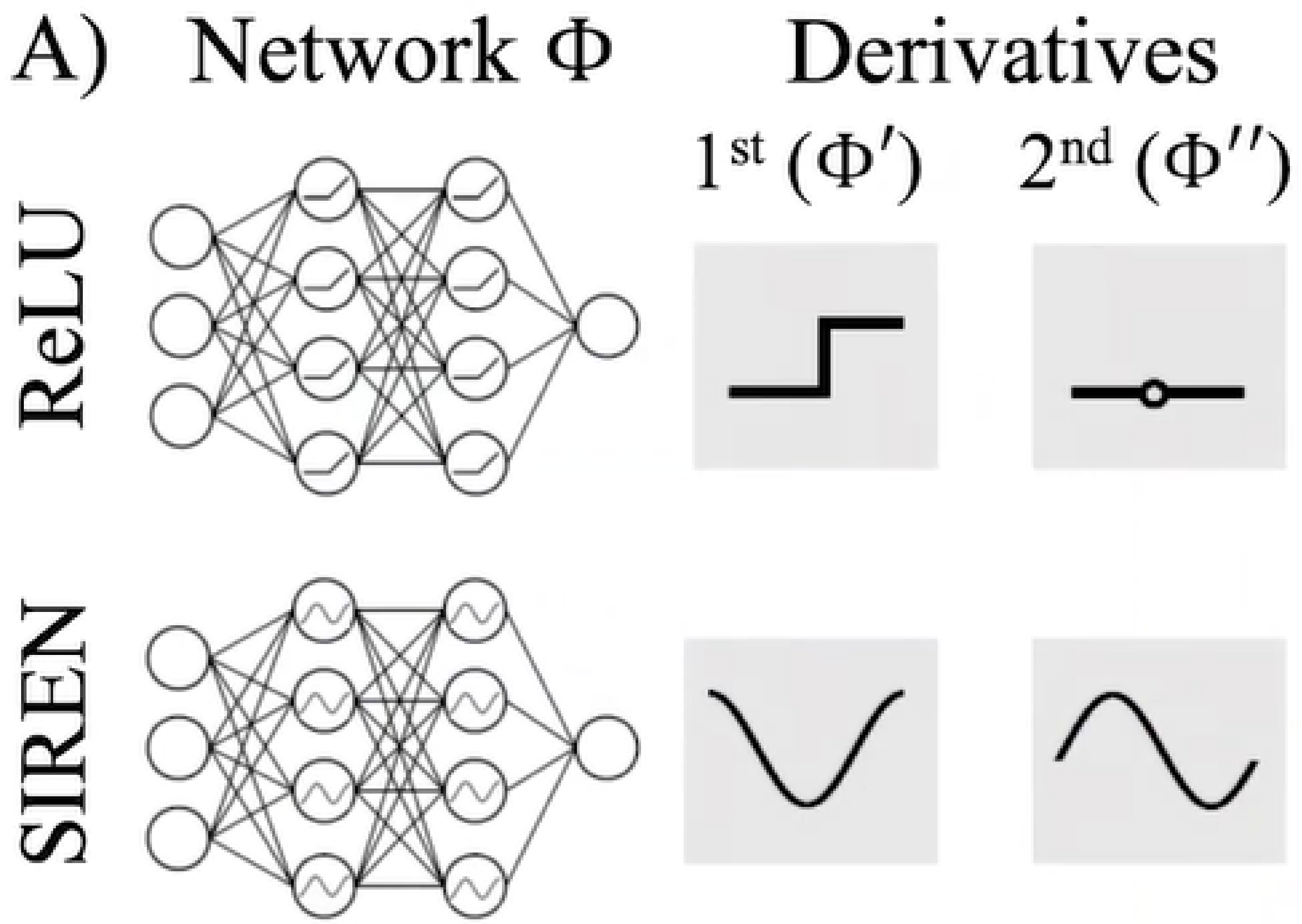
Want more…

Tancik et al. 2020

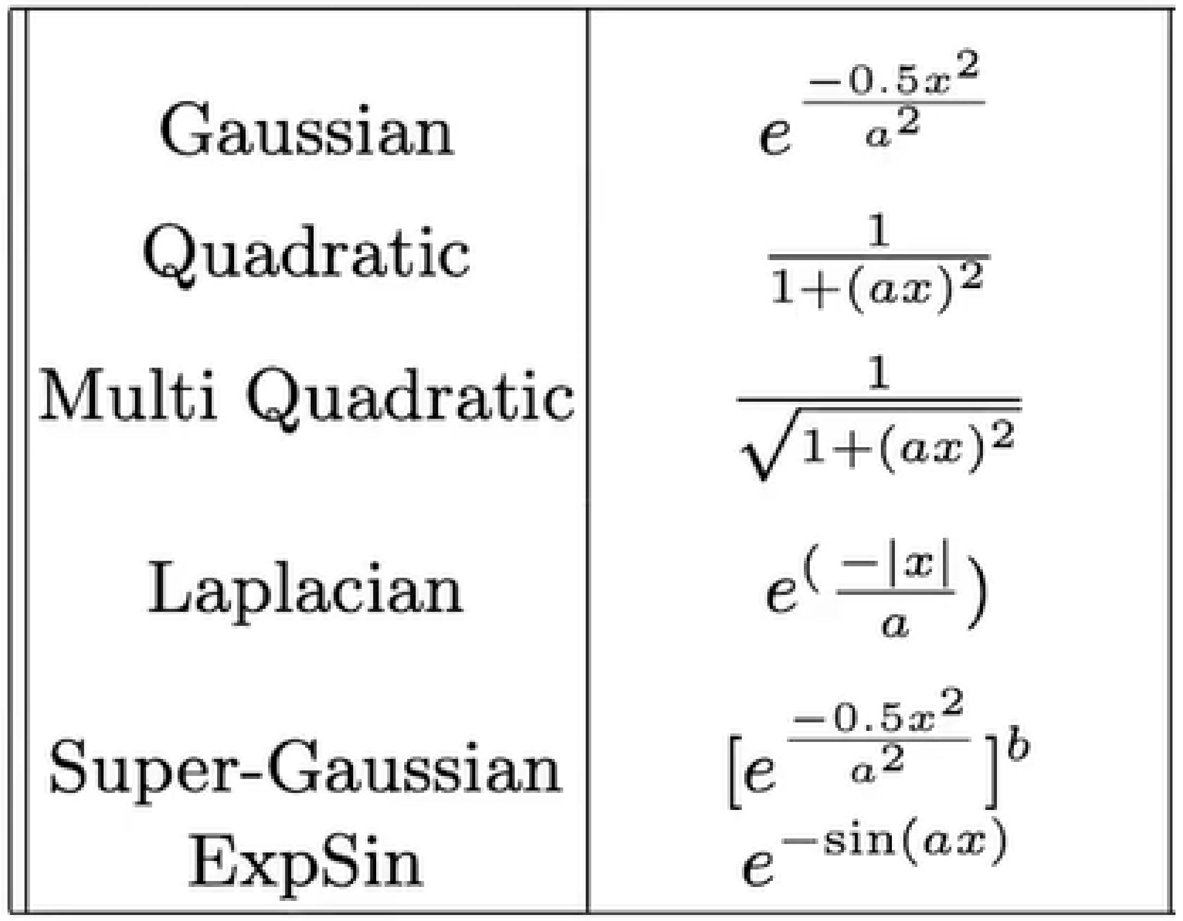
Passing input points through a simple Fourier feature mapping enables a multilayer perceptron (MLP) to learn high-frequency functions.

Activation functions

Want more...



[Sitzmann et al. 2021]



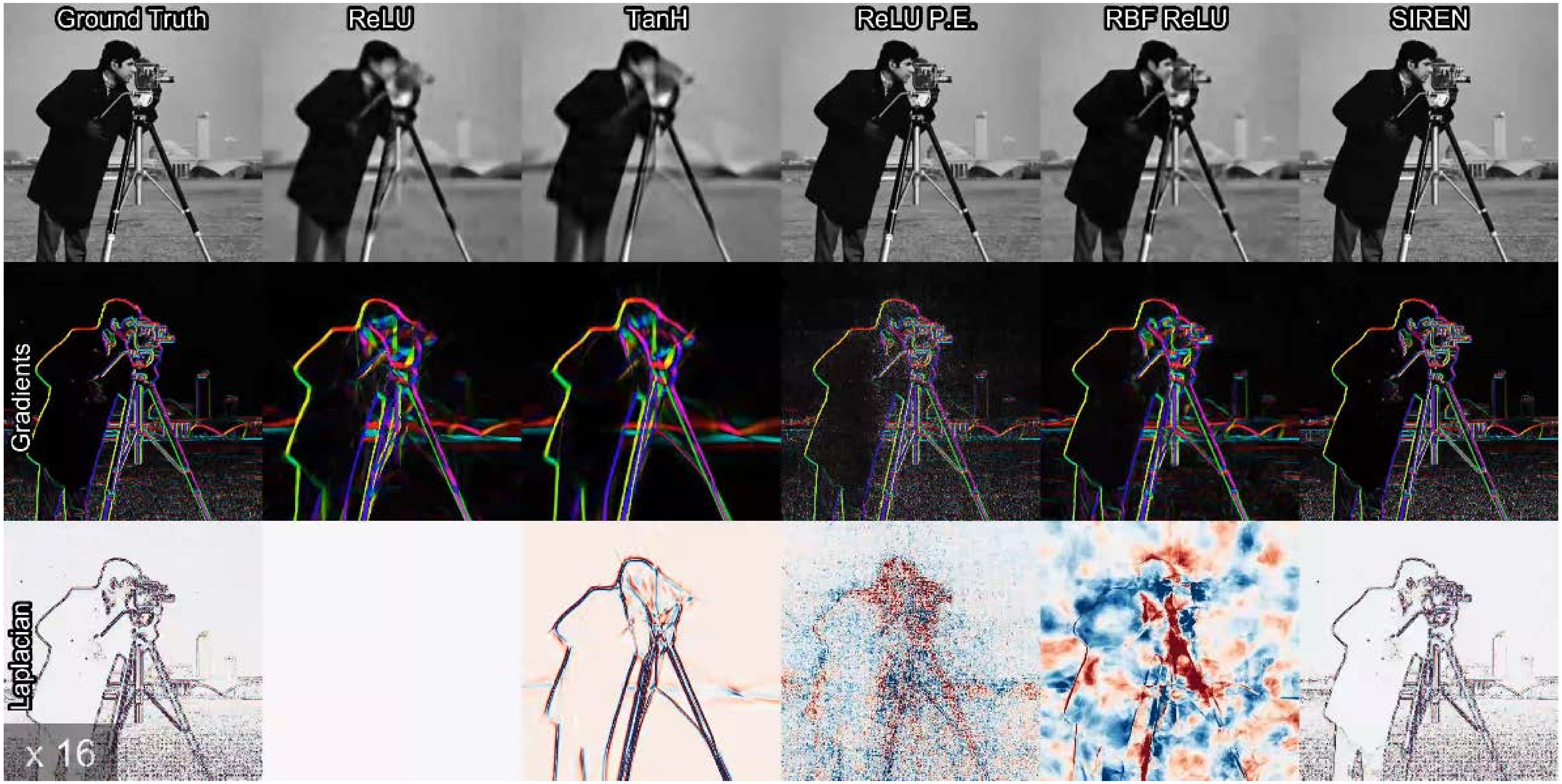
Gaussian	$e^{\frac{-0.5x^2}{a^2}}$
Quadratic	$\frac{1}{1+(ax)^2}$
Multi Quadratic	$\frac{1}{\sqrt{1+(ax)^2}}$
Laplacian	$e^{(\frac{-\lvert x\rvert}{a})}$
Super-Gaussian	$[e^{\frac{-0.5x^2}{a^2}}]^b$
ExpSin	$e^{-\sin(ax)}$

[Ramasinghe et al. 2021]

Activation functions

Sitzmann et al., 2021

Activation functions

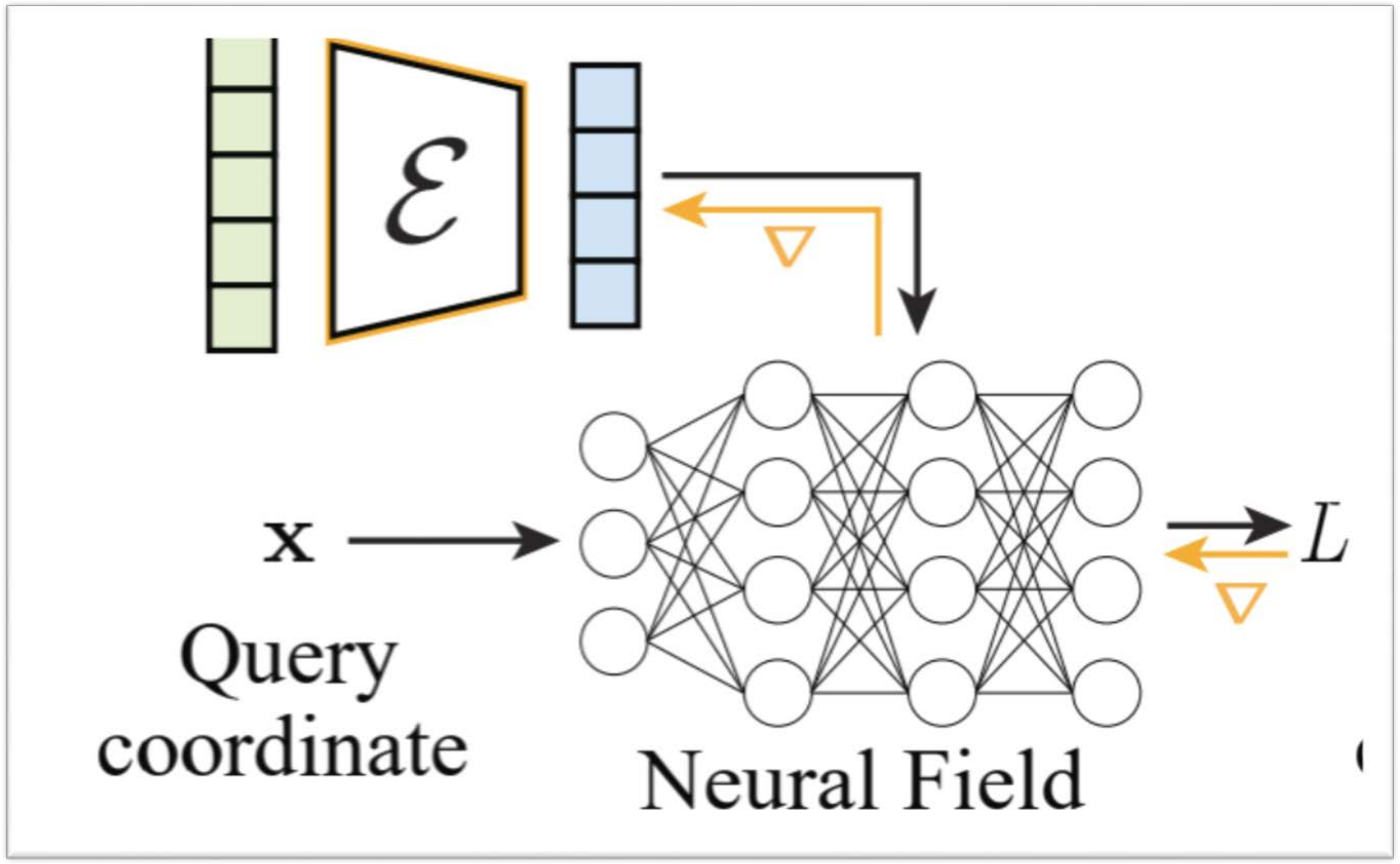


Sitzmann et al., 2021

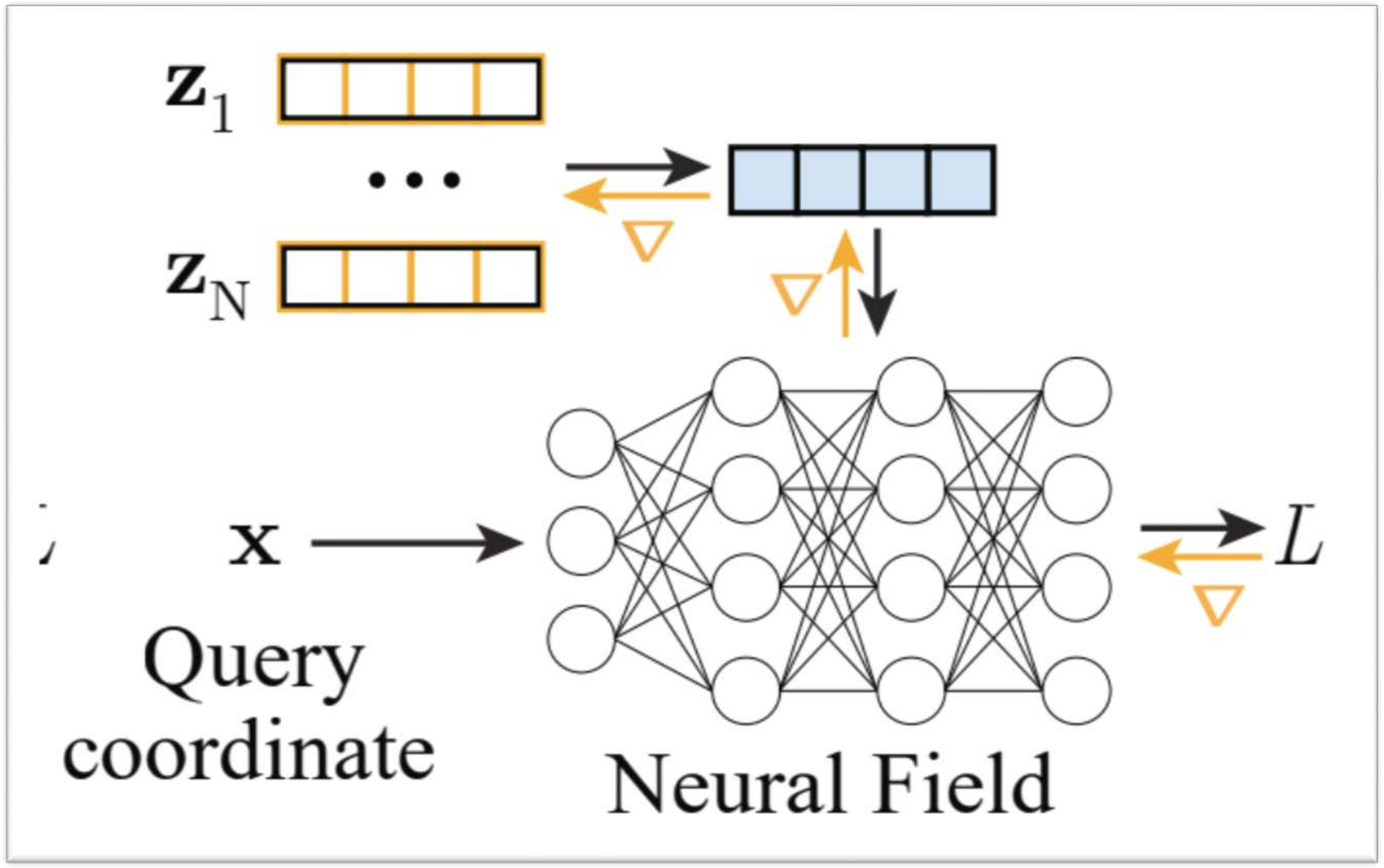
Conditioning

One NF for multiple instances.

Encoding



Auto-decoding

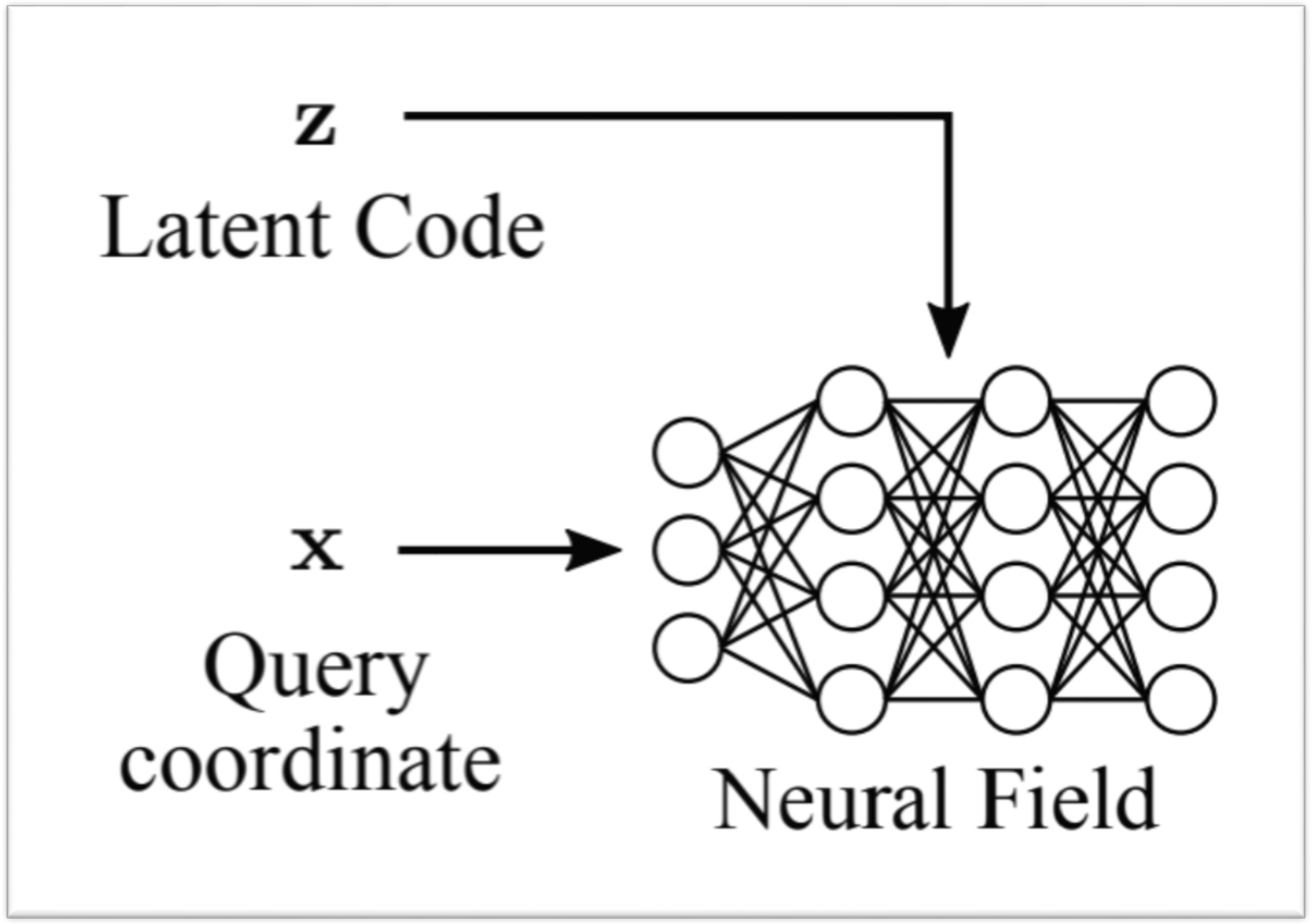


Xie et al., 2022

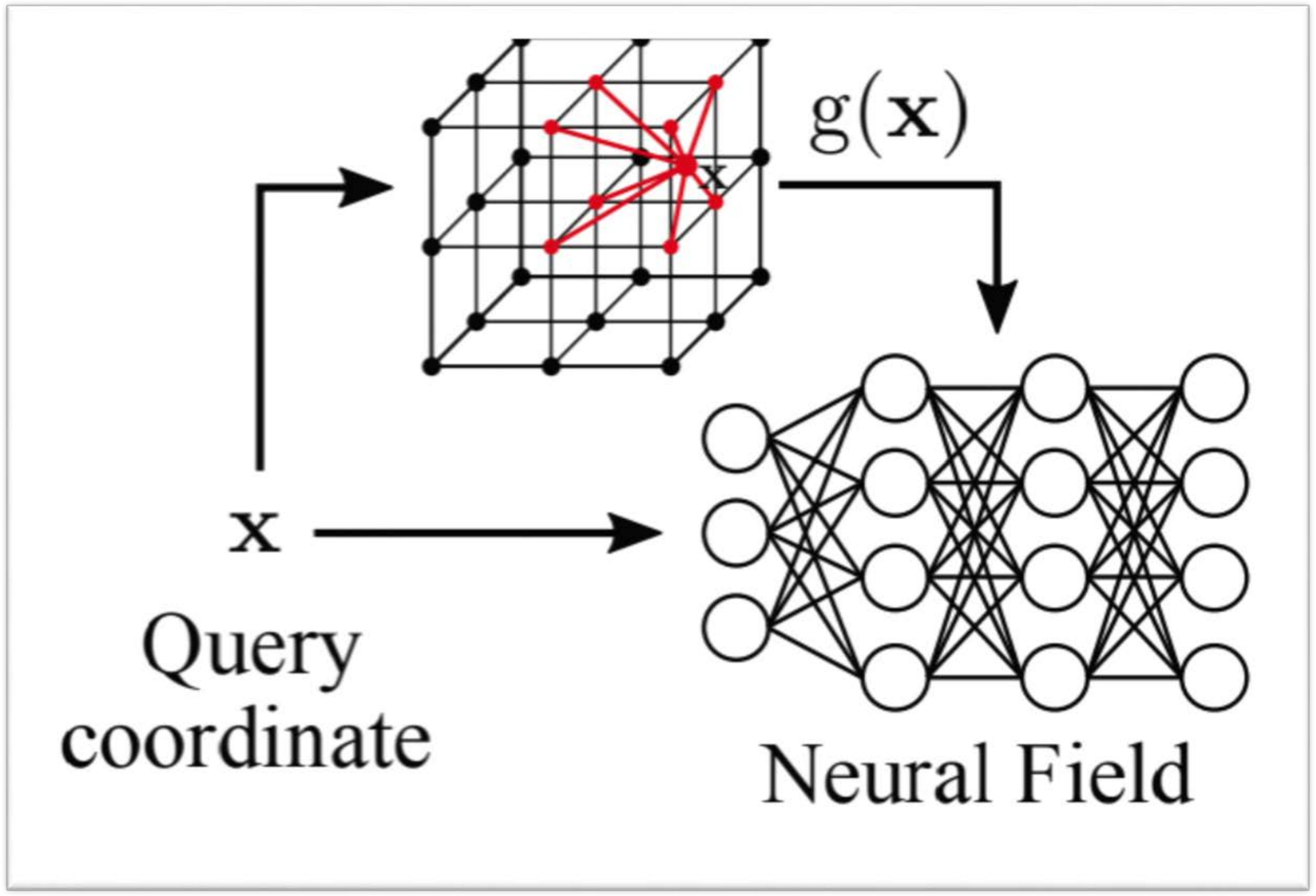
Conditioning

One NF for multiple instances.

Global



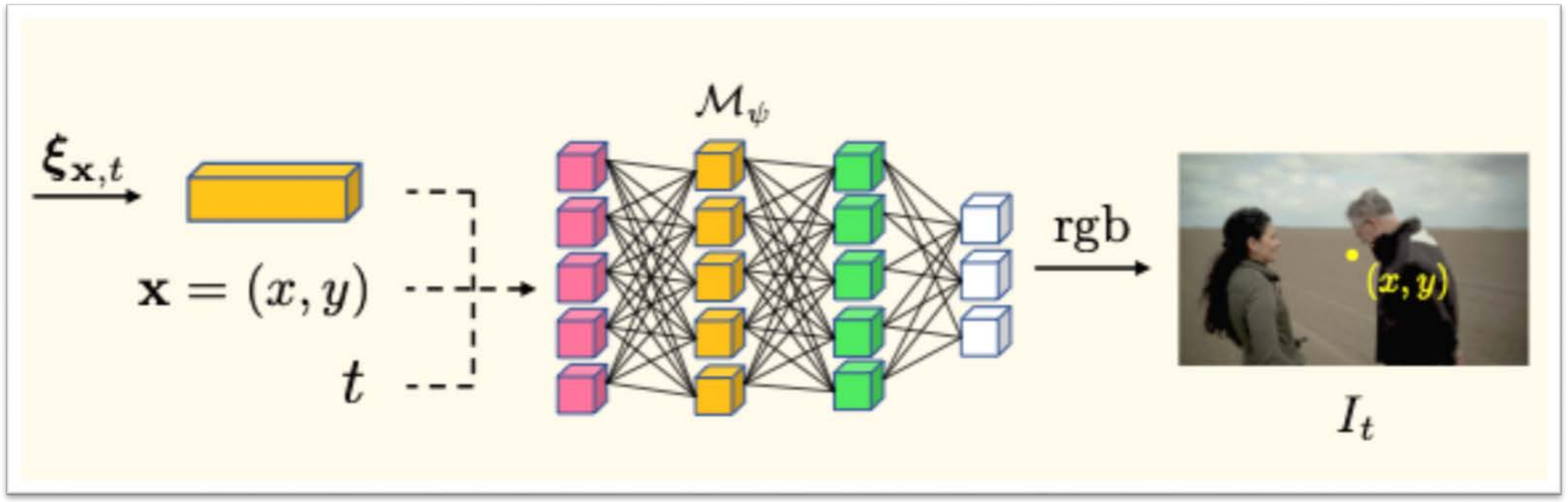
Local



Xie et al., 2022

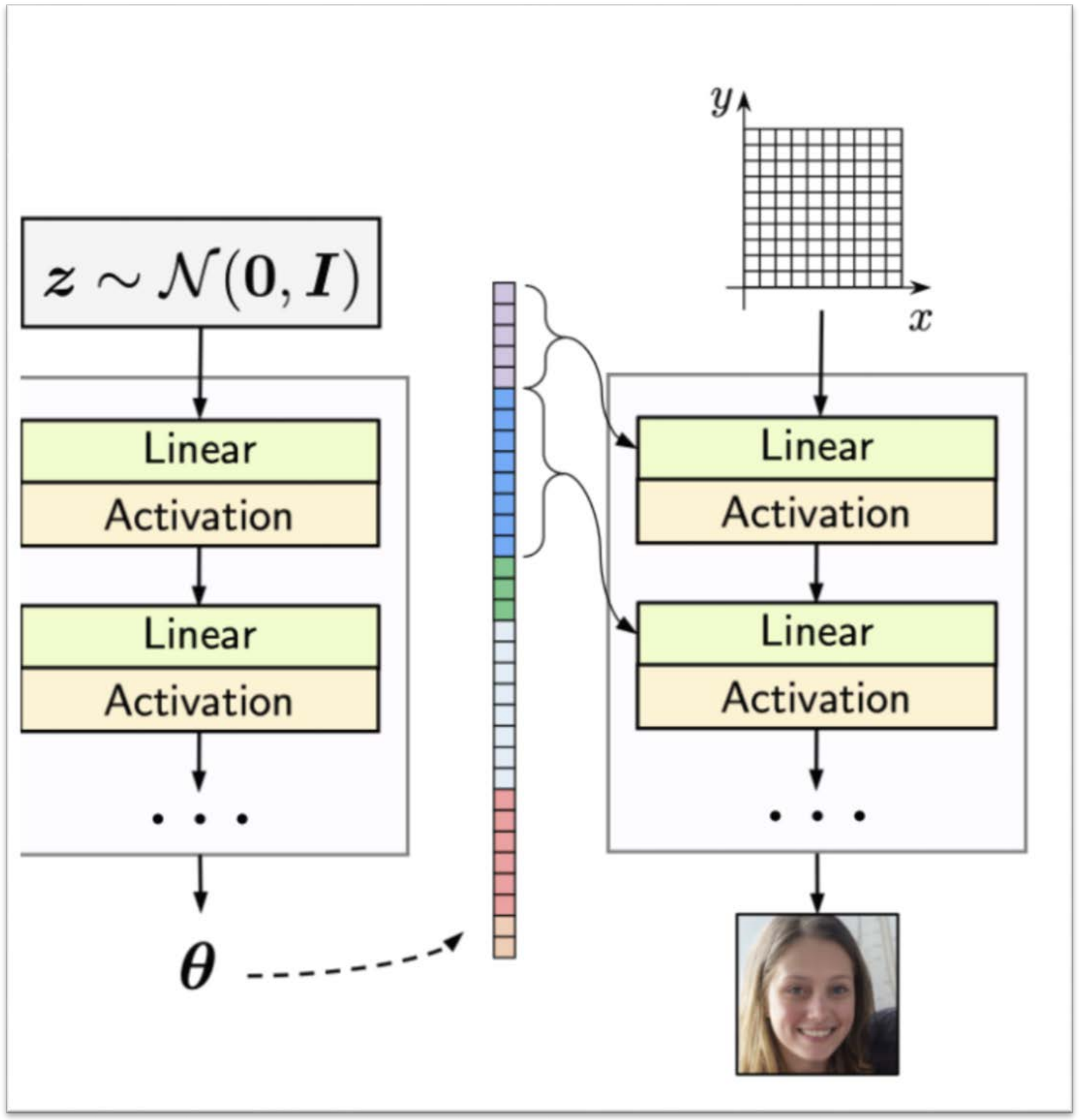
Conditioning

Concatenation

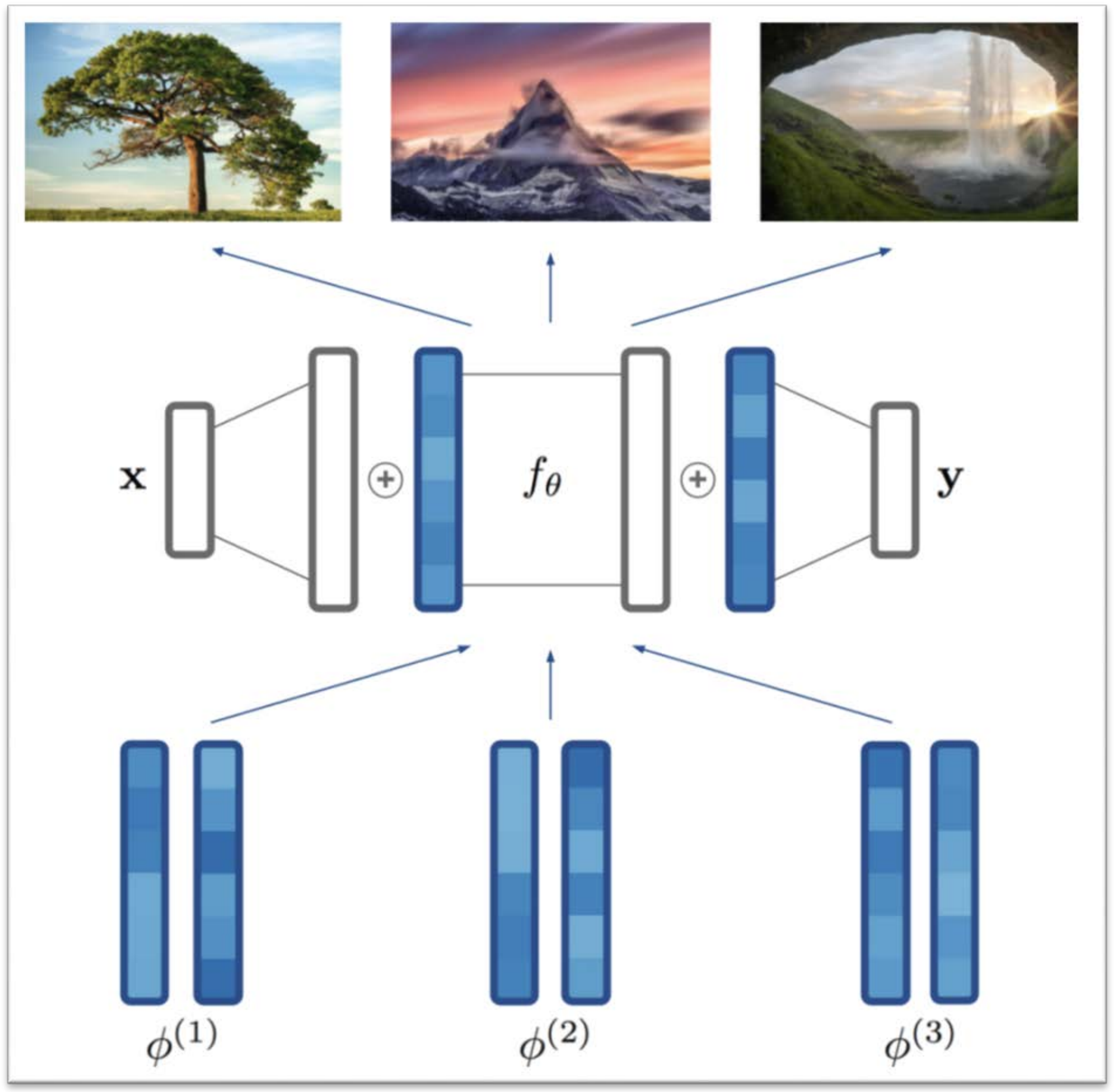


Shangguan et al., 2022

Hypernetwork



Skorokhodov et al., 2022



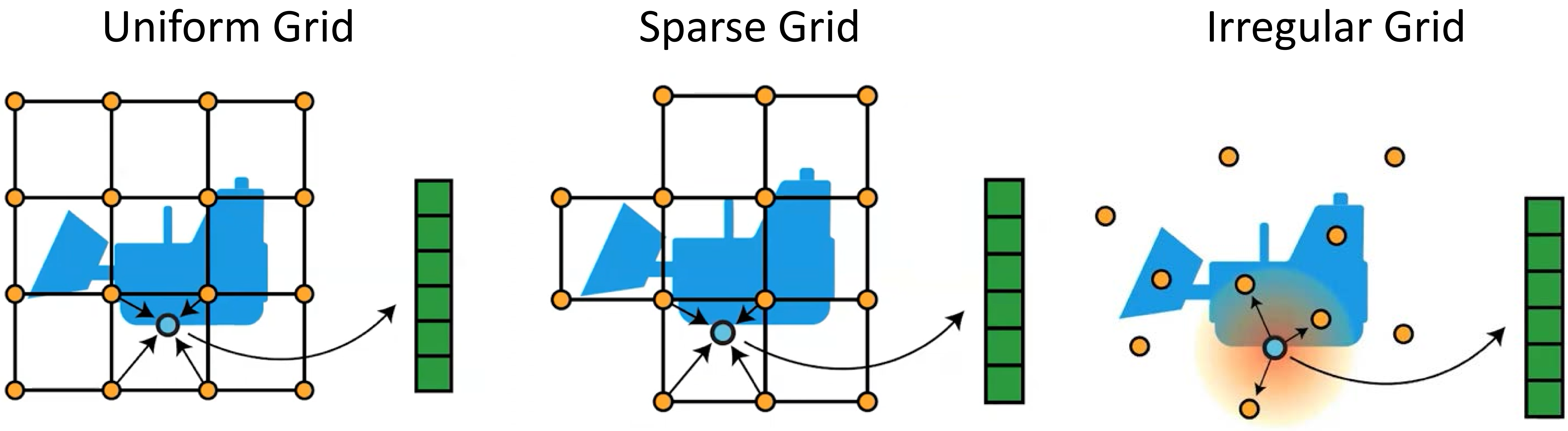
Modulations

Dupont et al., 2022

Hybrid representations

Combine neural fields with discrete data structures

- a collection of separate NFs are tiled across the input coordinate space
- given a coordinate x, retrieve specific parameters of the NF



Uniform Grid

Sparse Grid

Irregular Grid

Xie et al., 2022

Hybrid representations

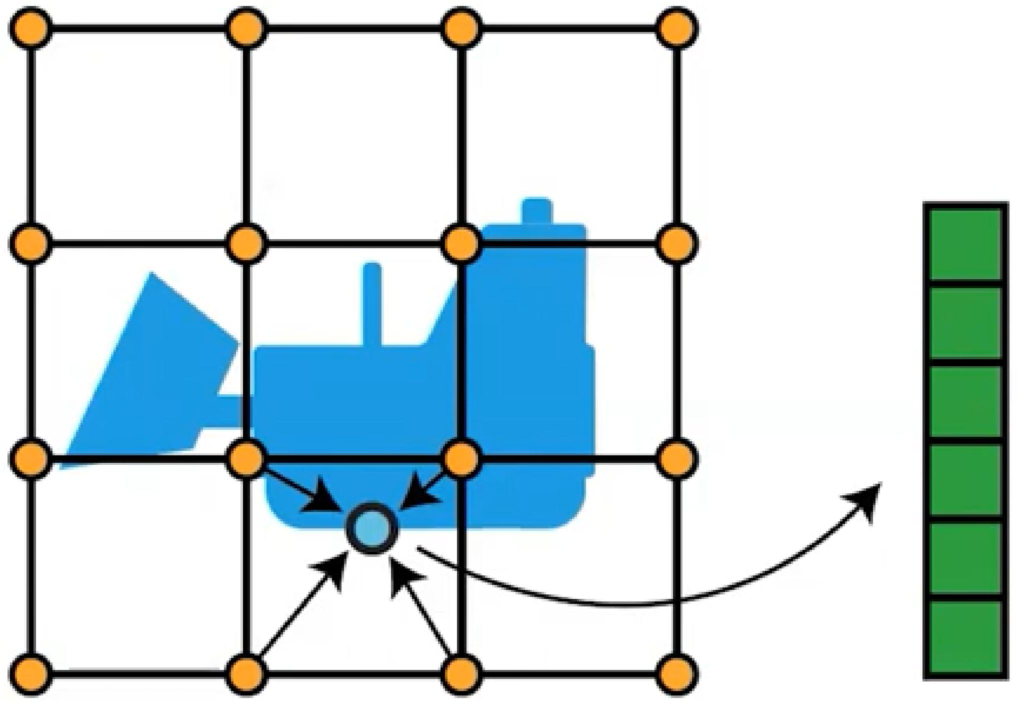
Want more…

Block-NeRF

CVPR 2022, Tancik et al.

Decomposing the city-scale scenes into individually trained NeRFs.

Uniform Grid



Hybrid representations

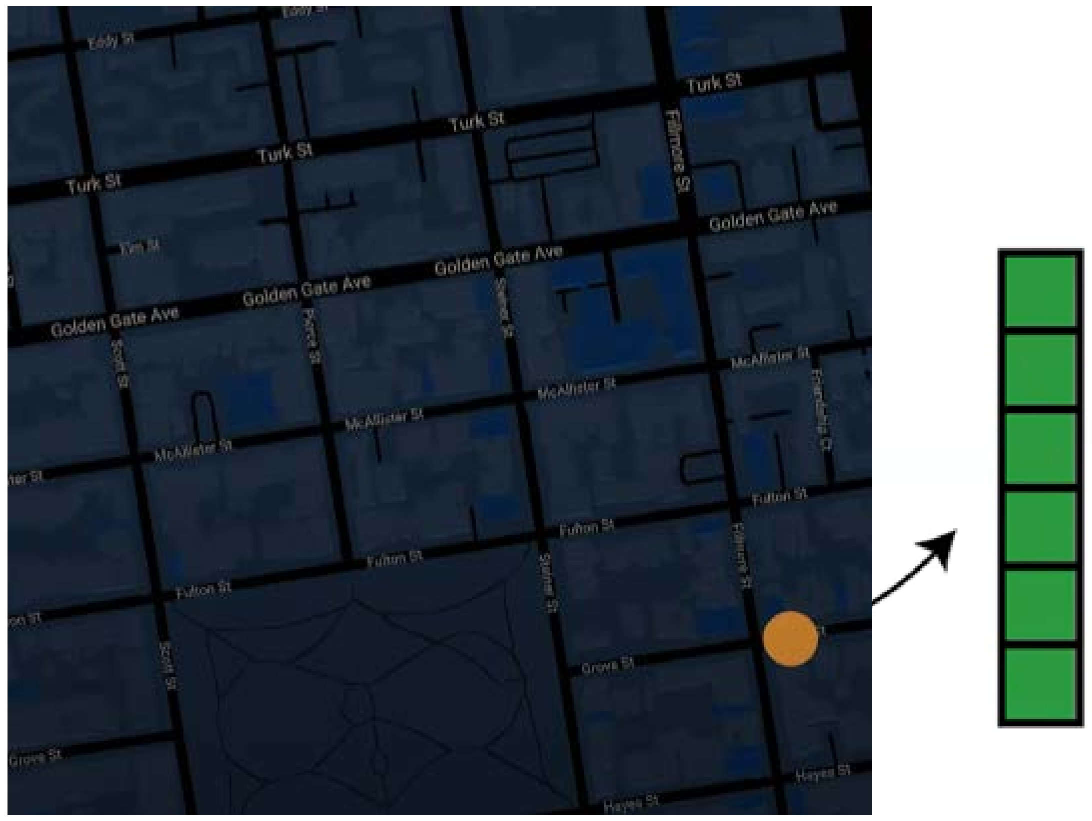
Want more...

Block-NeRF

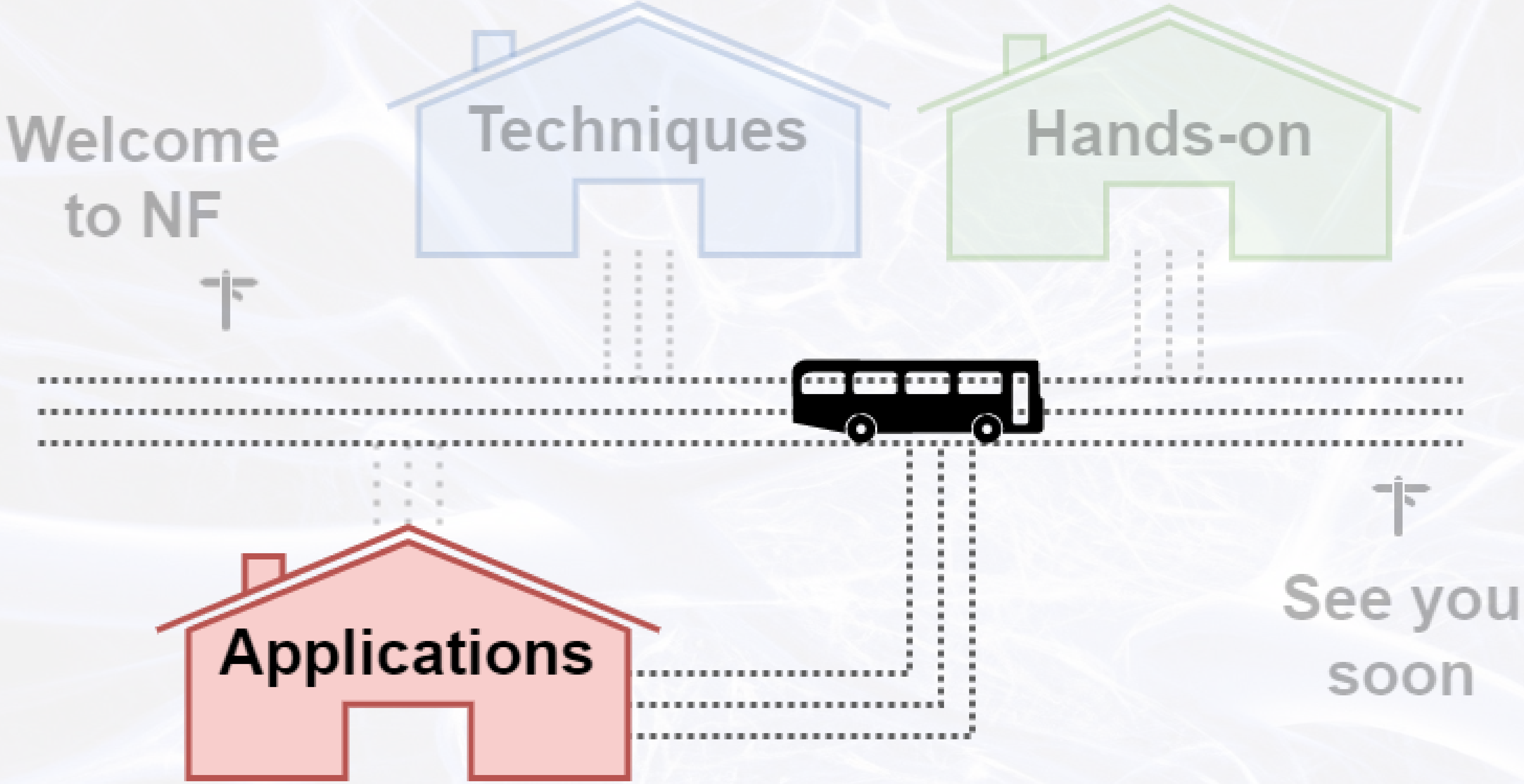
CVPR 2022, Tancik et al.

Decomposing the city-scale scenes into individually trained NeRFs.

Uniform Grid



Let's go on a tour to NF

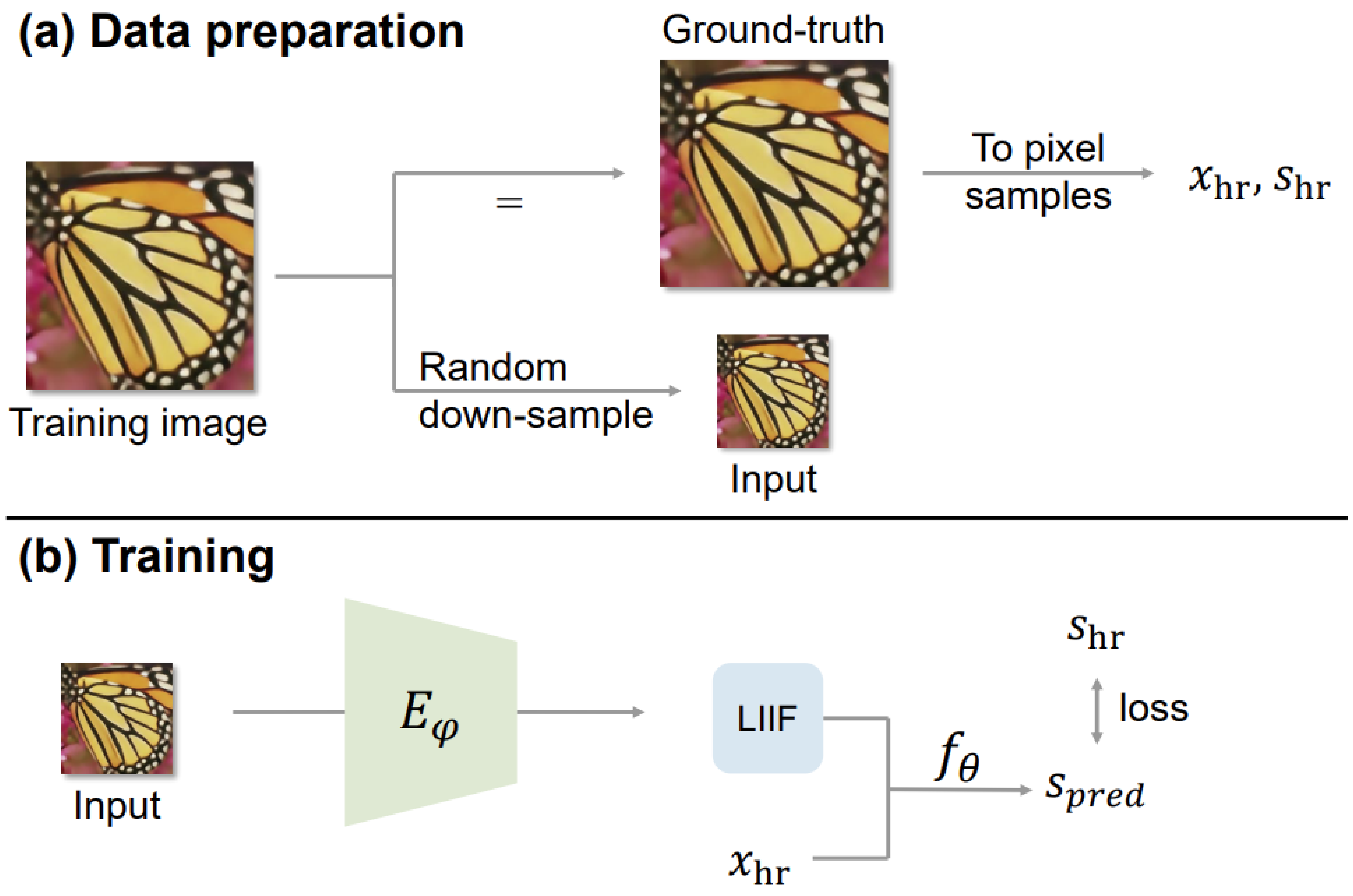


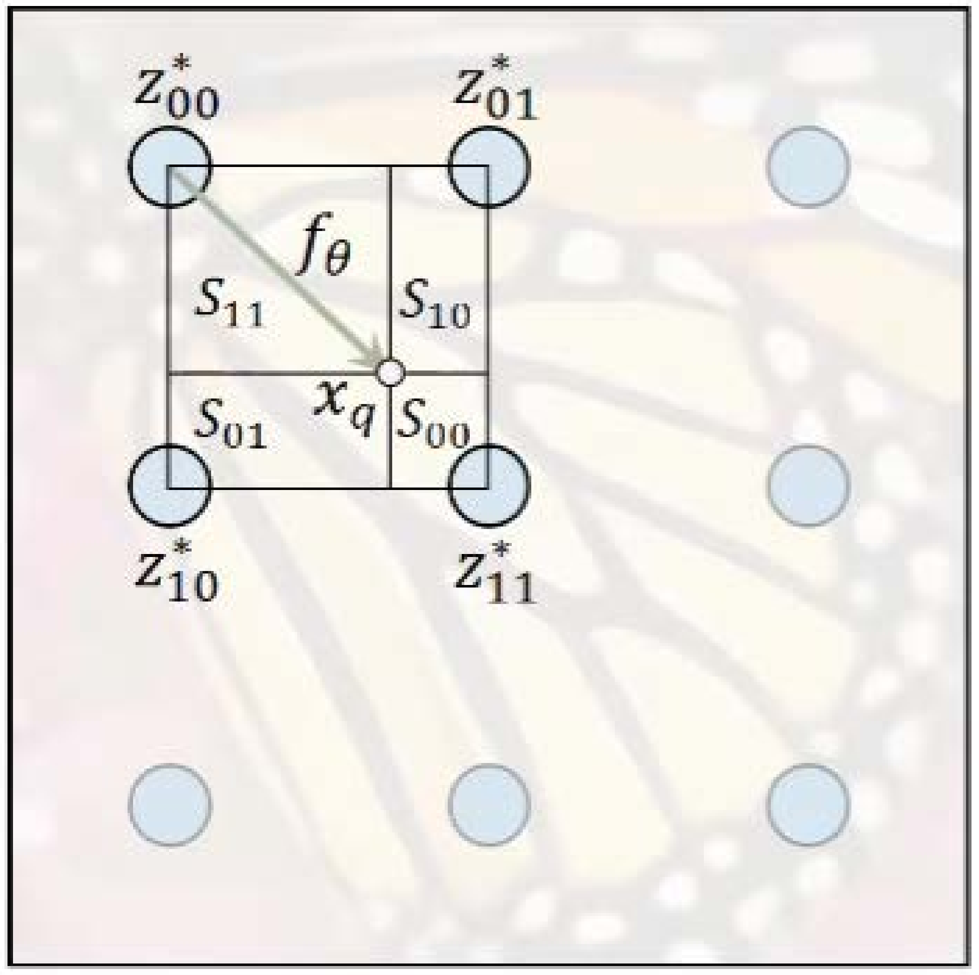
Applications

LIIF

Learning Continuous Image Representation with Local Implicit Image Function CVPR 2021, Chen et al.

Project Page





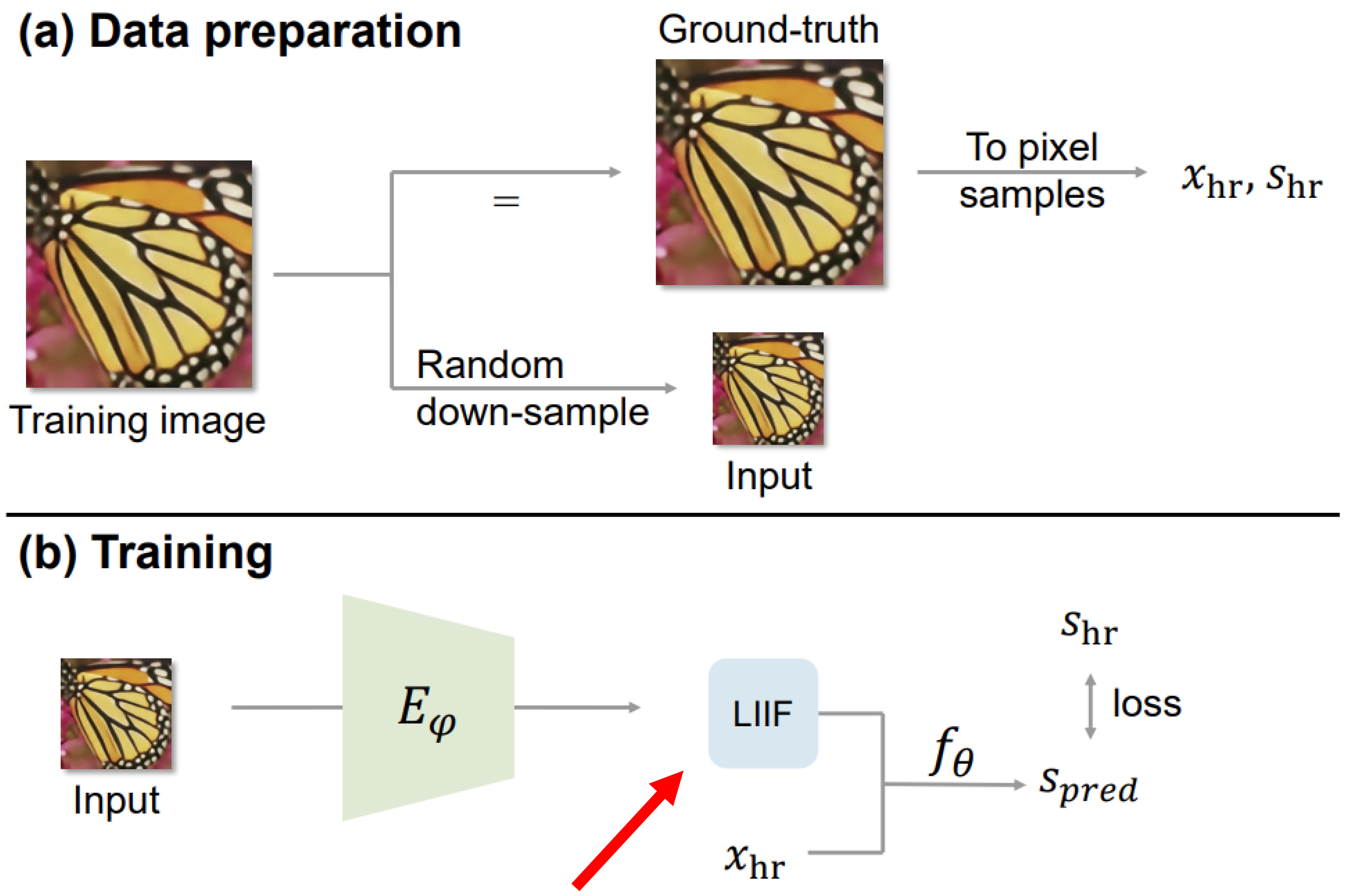
LIIF representation with local ensemble

LIIF

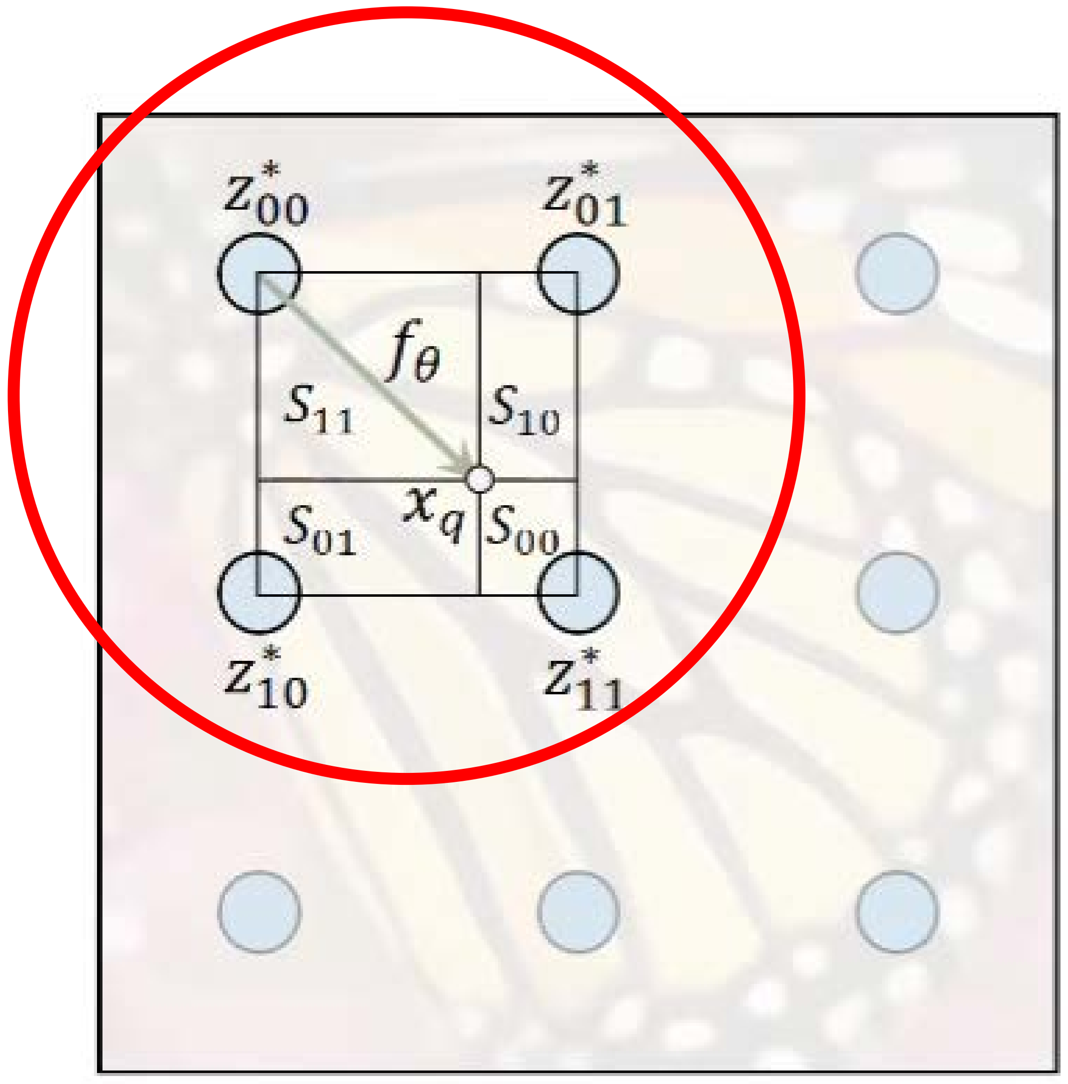
Learning Continuous Image Representation
with Local Implicit Image Function
CVPR 2021, Chen et al.

Project Page

Hybrid representation



Conditioning



LIIF representation with local ensemble

LIIF

Learning Continuous Image Representation with Local Implicit Image Function
CVPR 2021, Chen et al.

Project Page

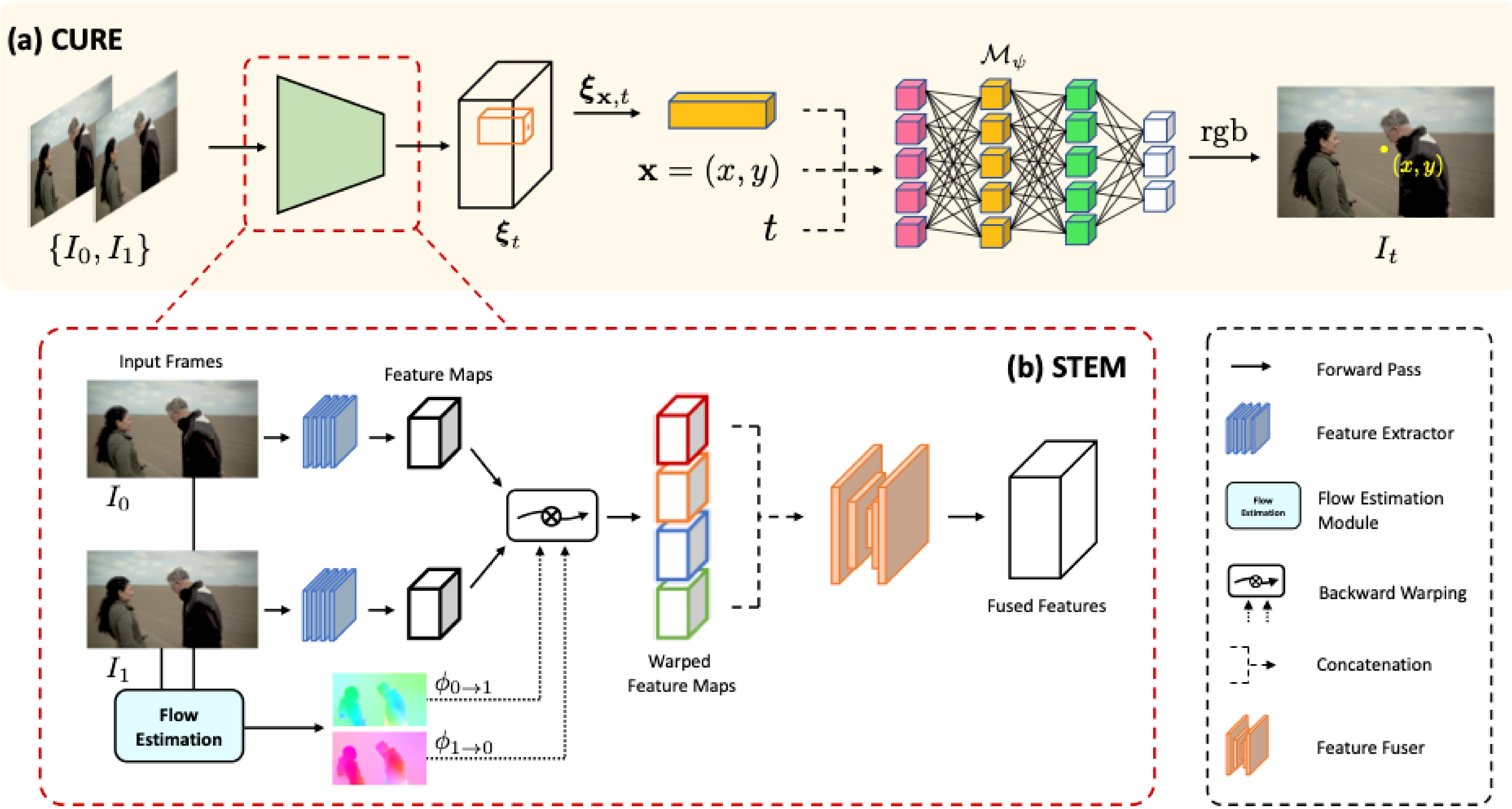
Input (32px)

LIIF in changing resolution (32px-640px)

CURE

CURE: Learning Cross-Video Neural Representations for High-Quality Frame Interpolation
ECCV 2022, Shangguan et al.

Project Page

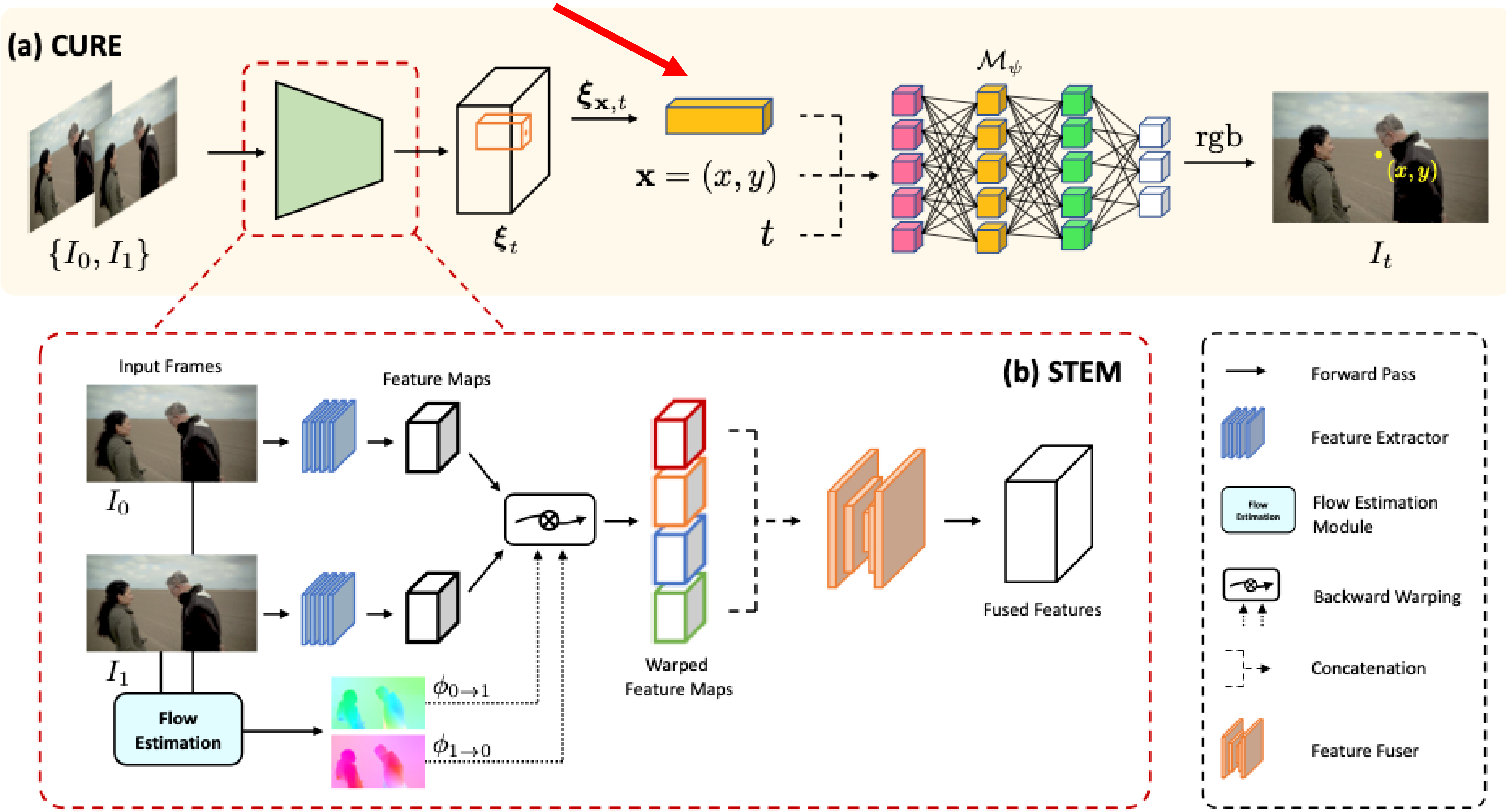


CURE

CURE: Learning Cross-Video Neural Representations for High-Quality Frame Interpolation
ECCV 2022, Shangguan et al.

[Project Page]

Conditioning



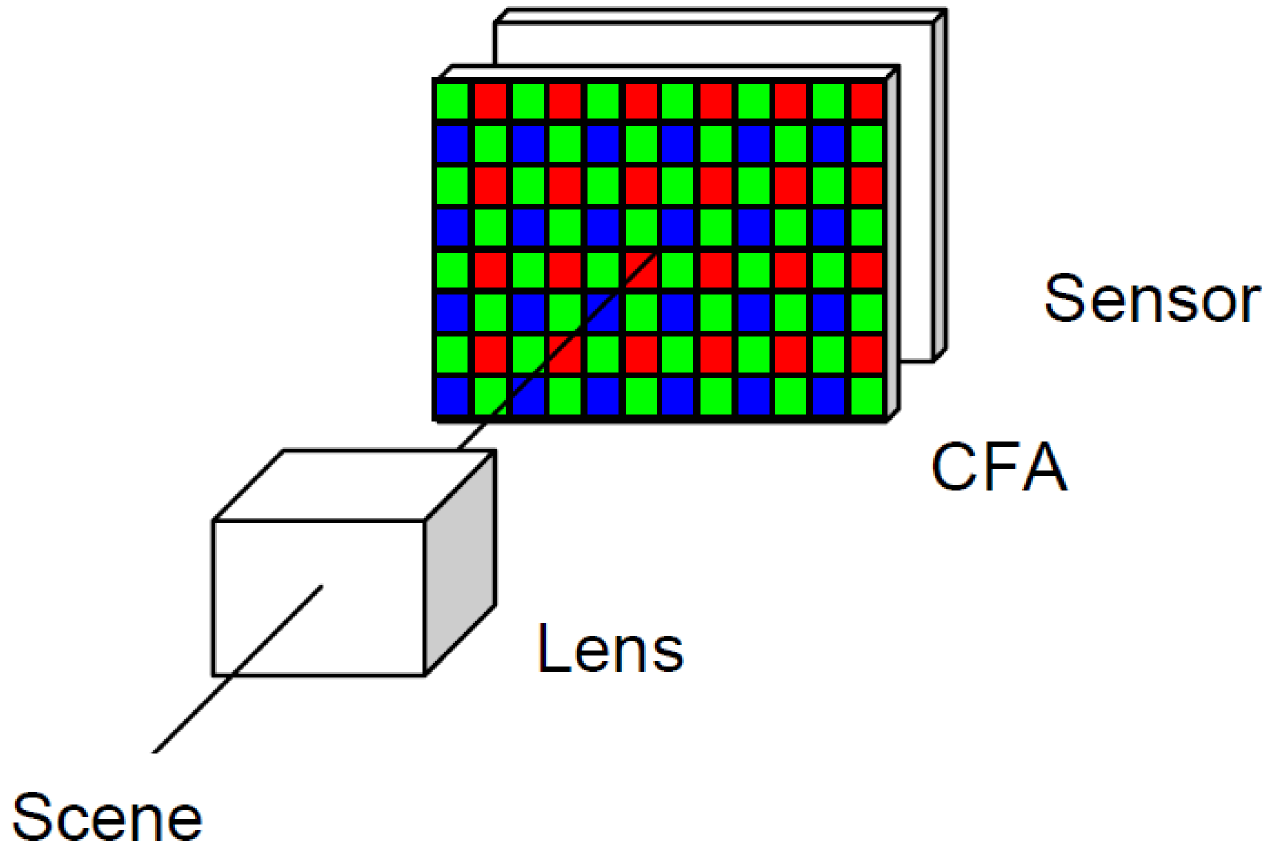
CURE

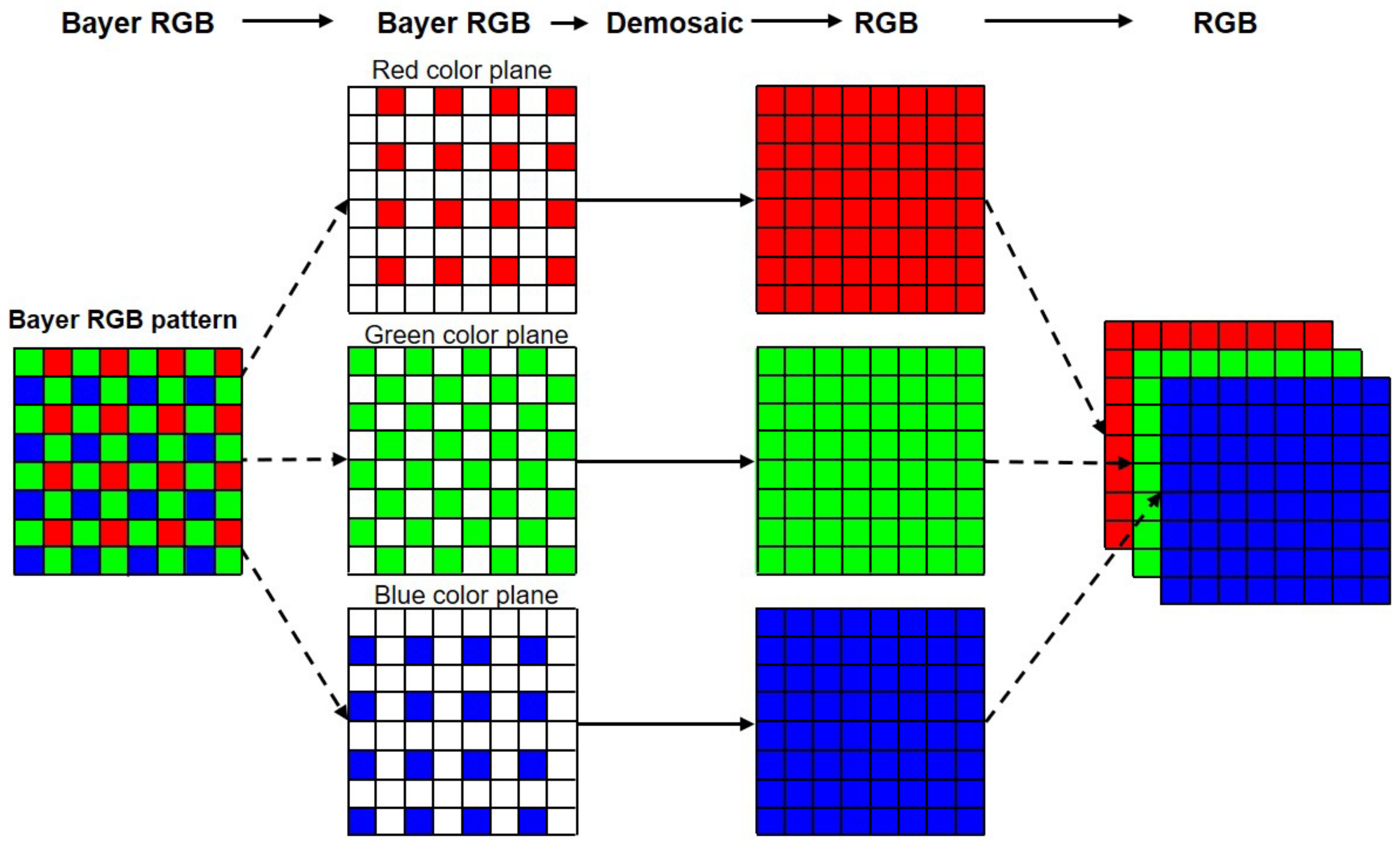
CURE: Learning Cross-Video Neural Representations for High-Quality Frame Interpolation
ECCV 2022, Shangguan et al.

Project Page

Demosaicing

by Neural Fields



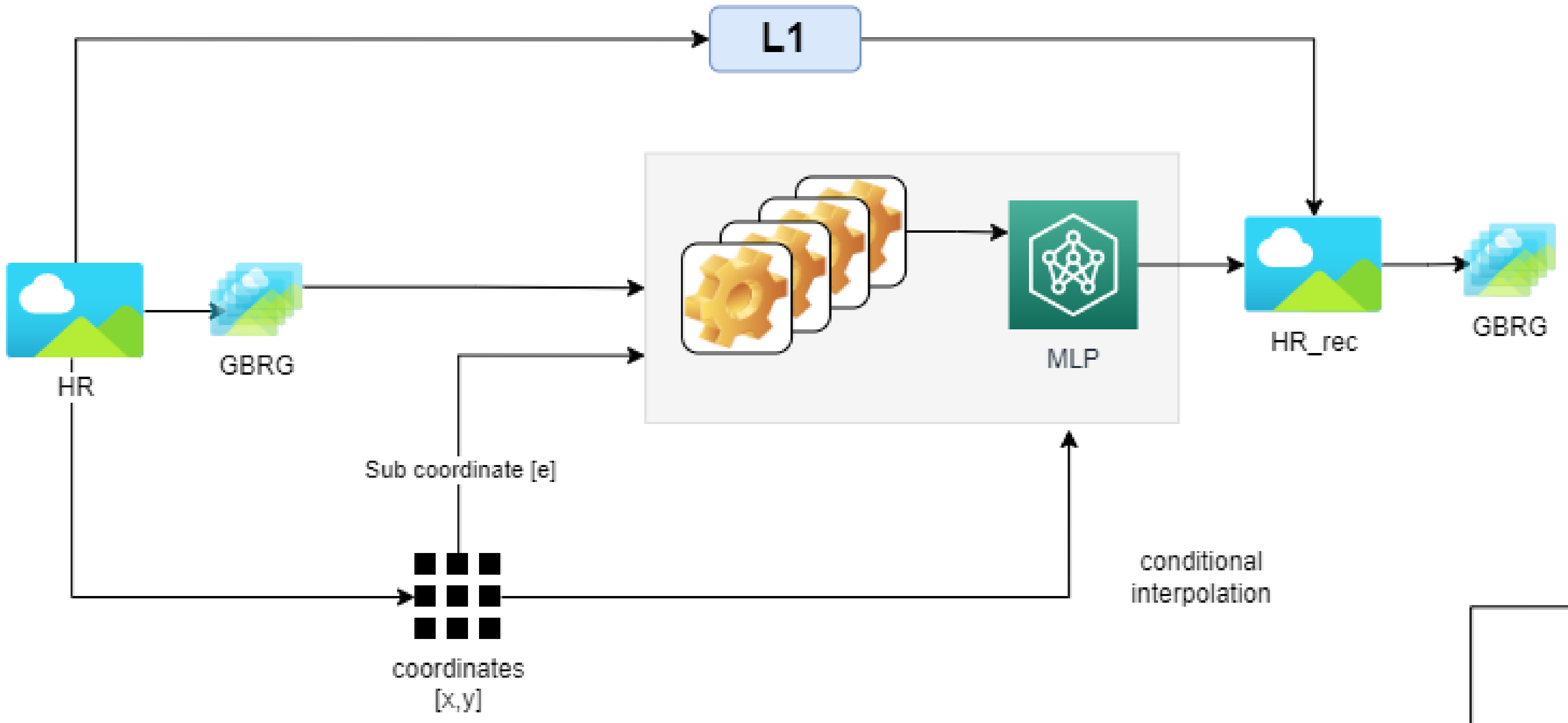


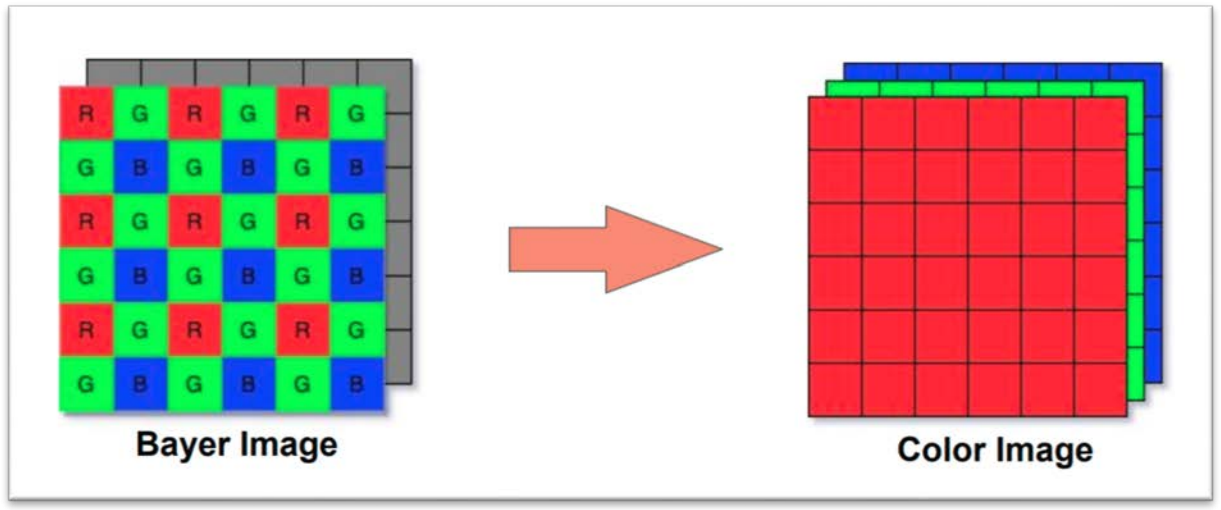
Applications

Demosaicing

by Neural Fields

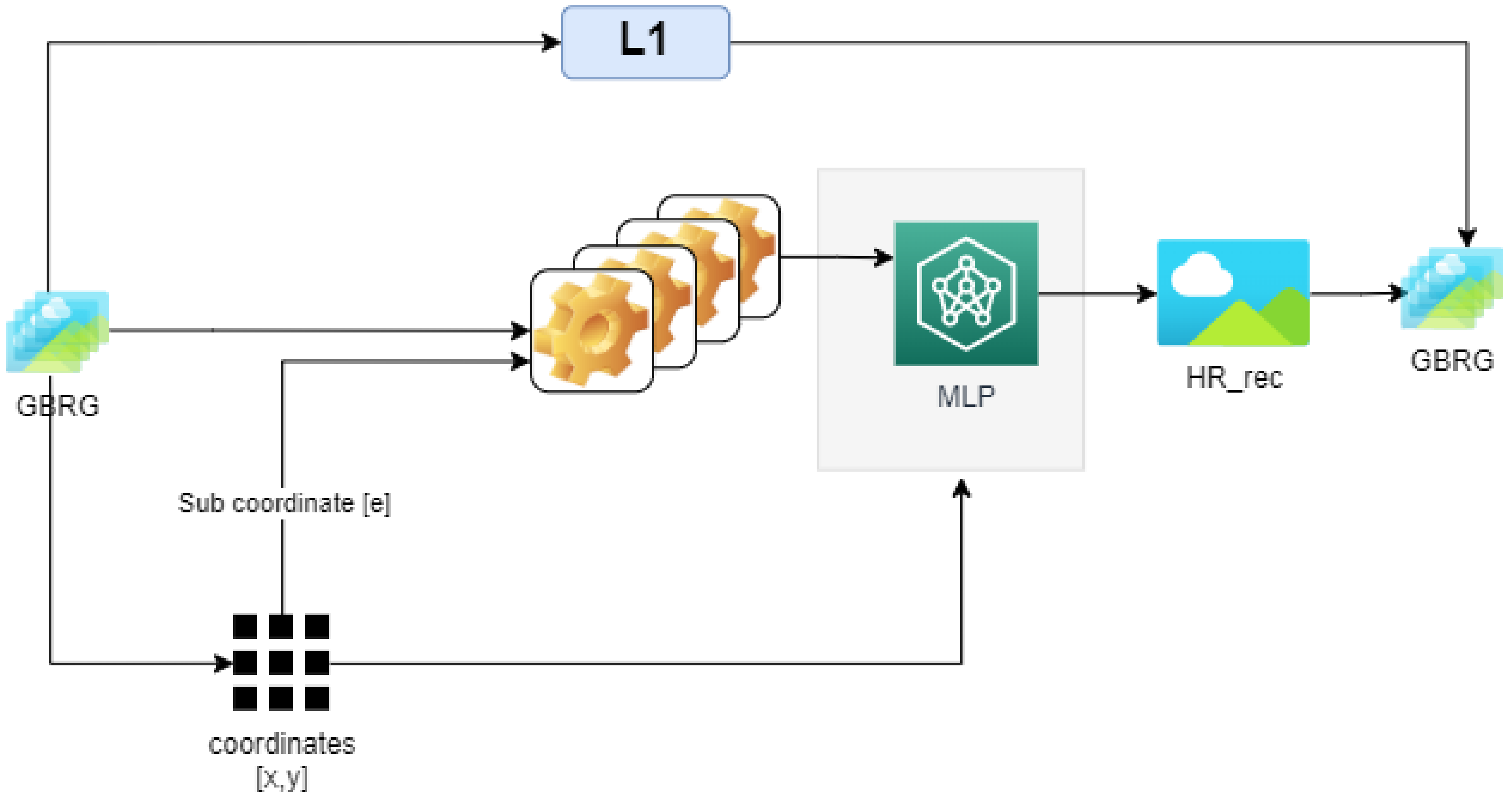
1. Supervised training phase (Enc + MLP)

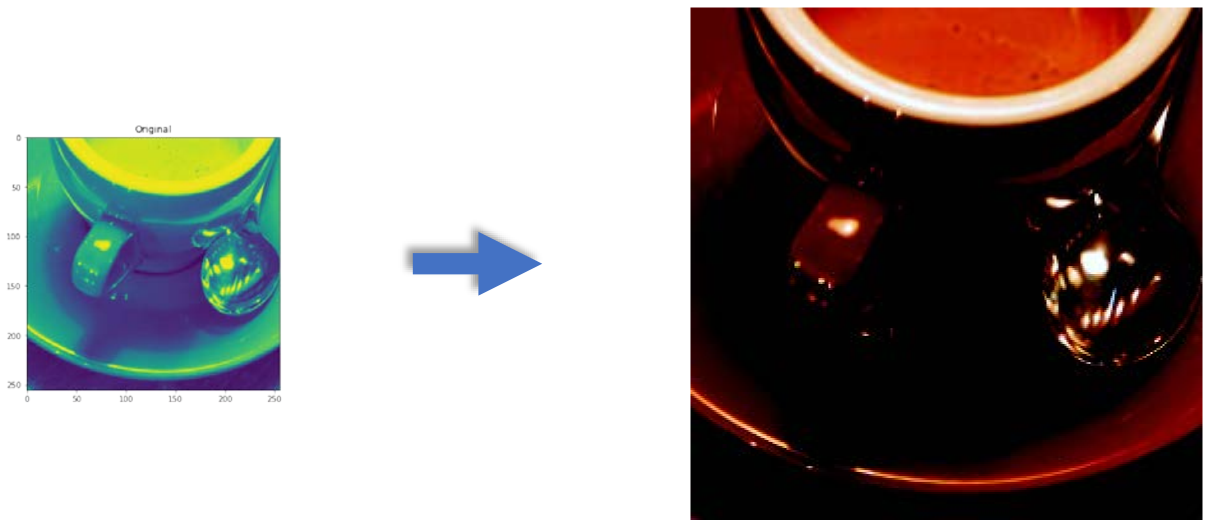




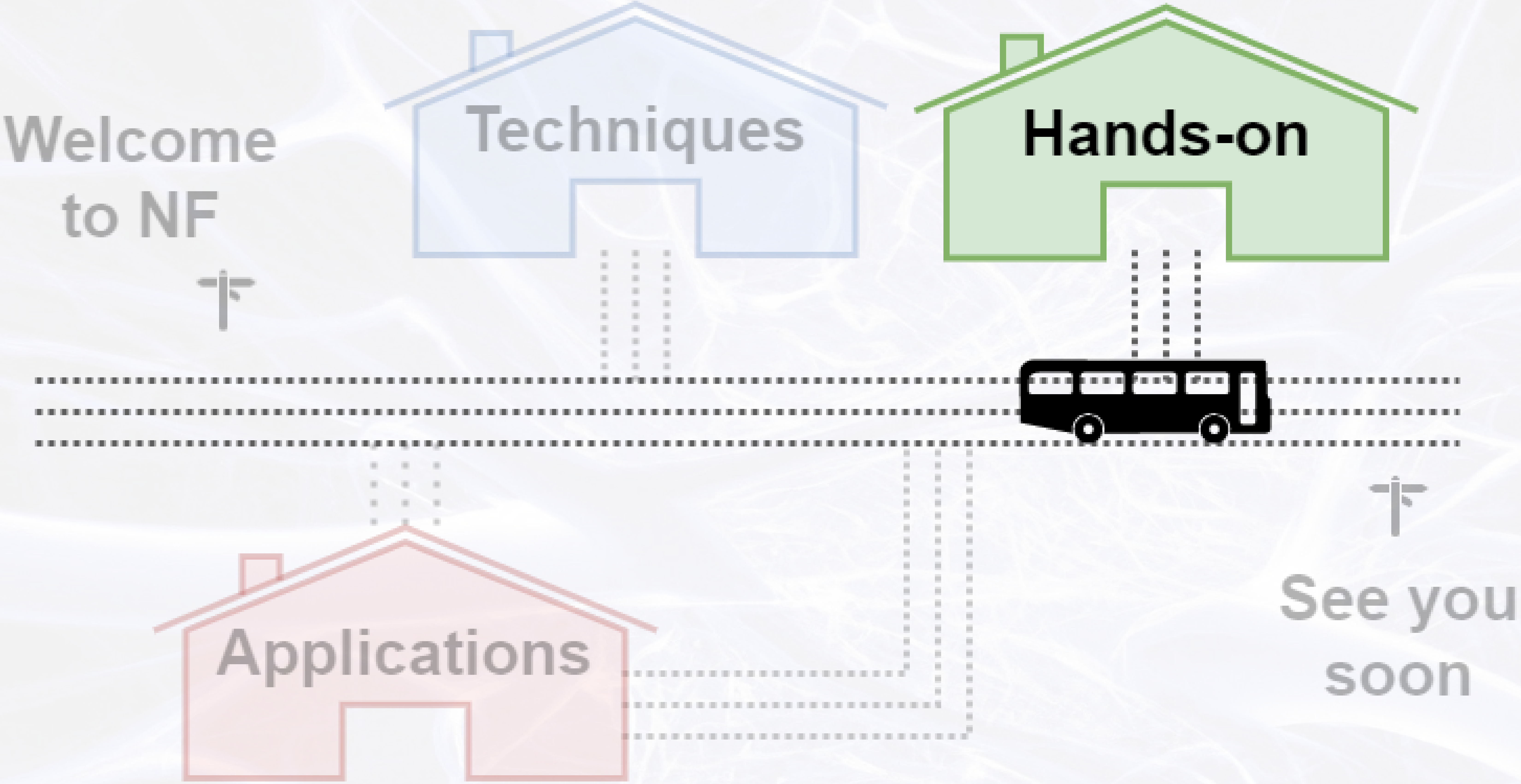
github.com/Howeng98/Demosaicing_SuperResolution

2. Self-supervised training (fine-tune MLP)





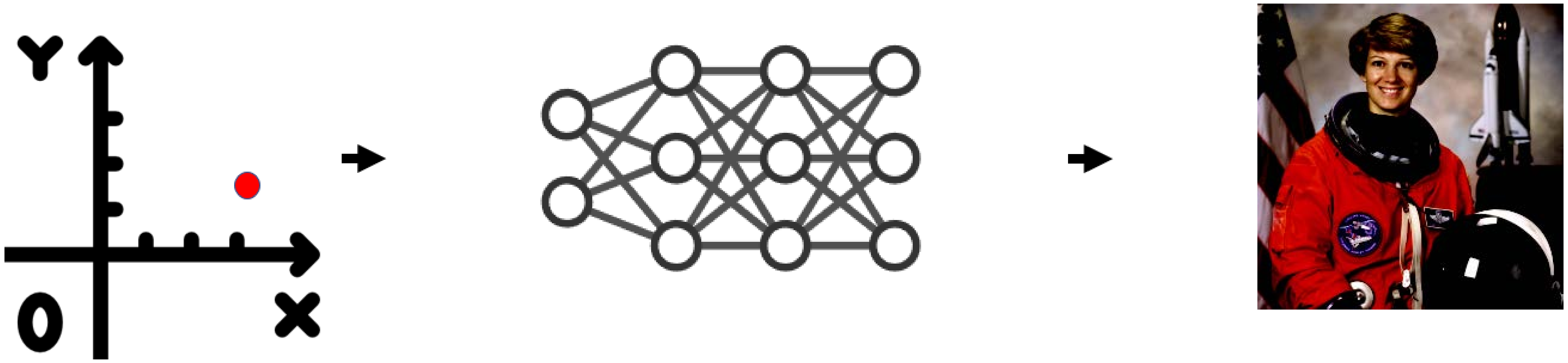
Let's go on a tour to NF

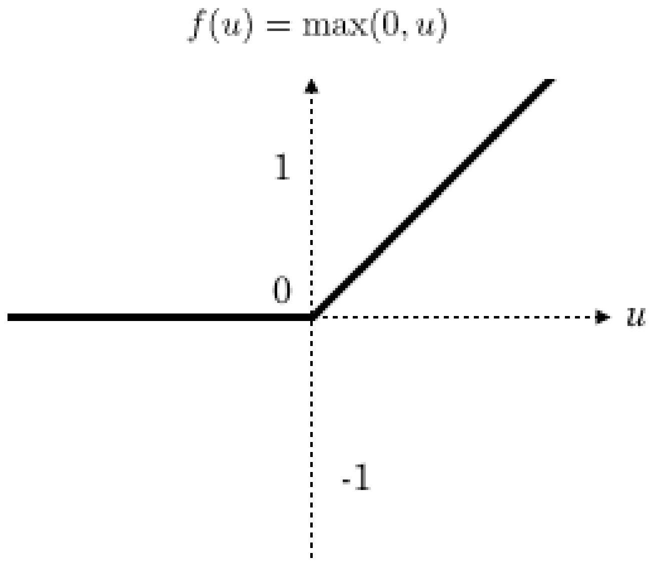


Hands-on

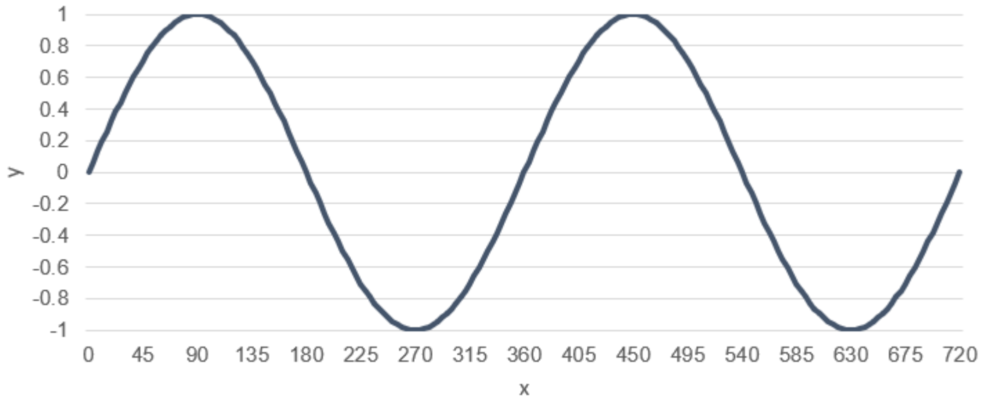
ReLU vs. ReLU+PE vs. SIREN

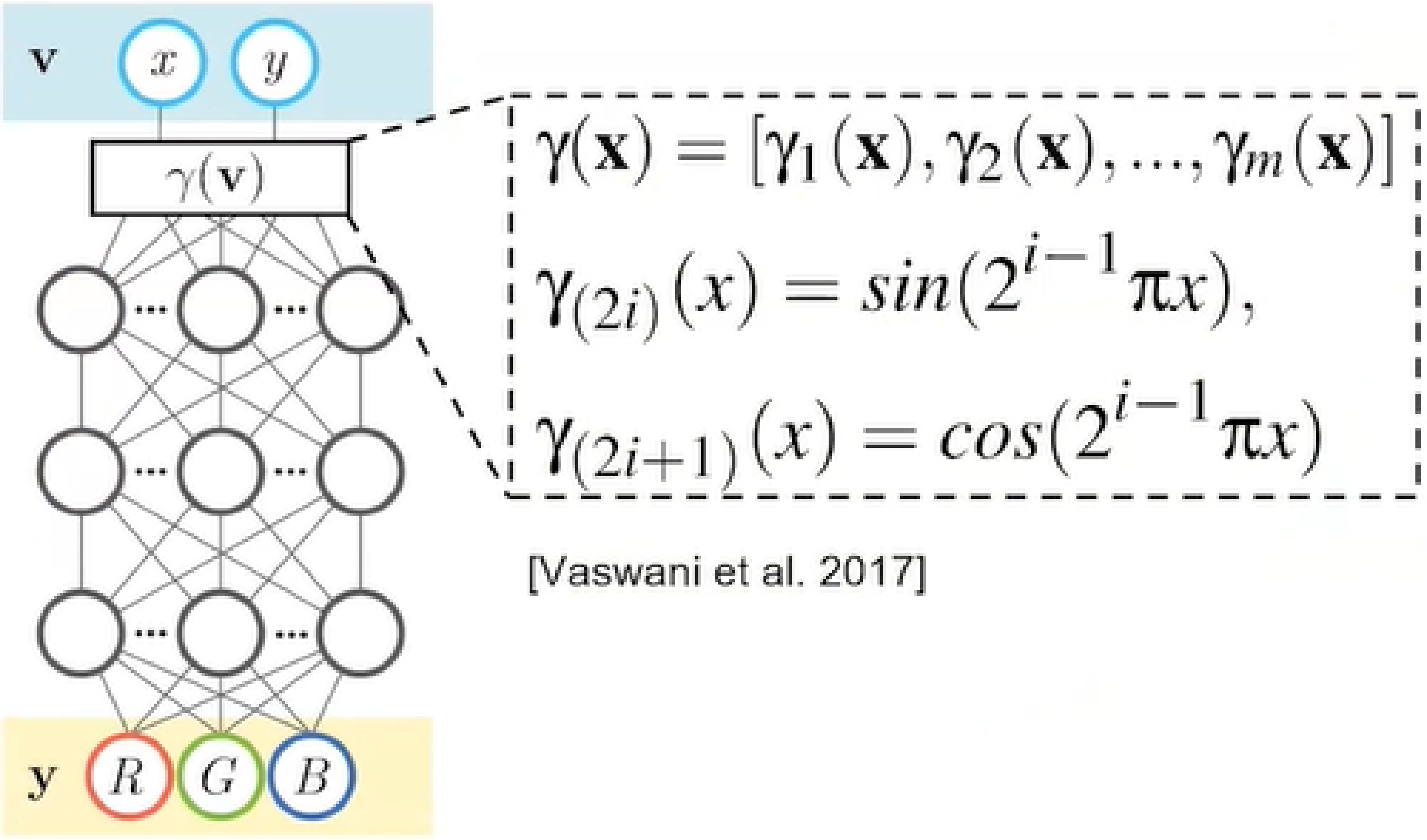
Full colab code





Graph of y = sin(x)





$$\gamma(\mathbf{x}) = [\gamma_1(\mathbf{x}), \gamma_2(\mathbf{x}), \ldots, \gamma_m(\mathbf{x})]$$

$$\gamma_{(2i)}(x) = sin(2^{i-1}\pi x),$$

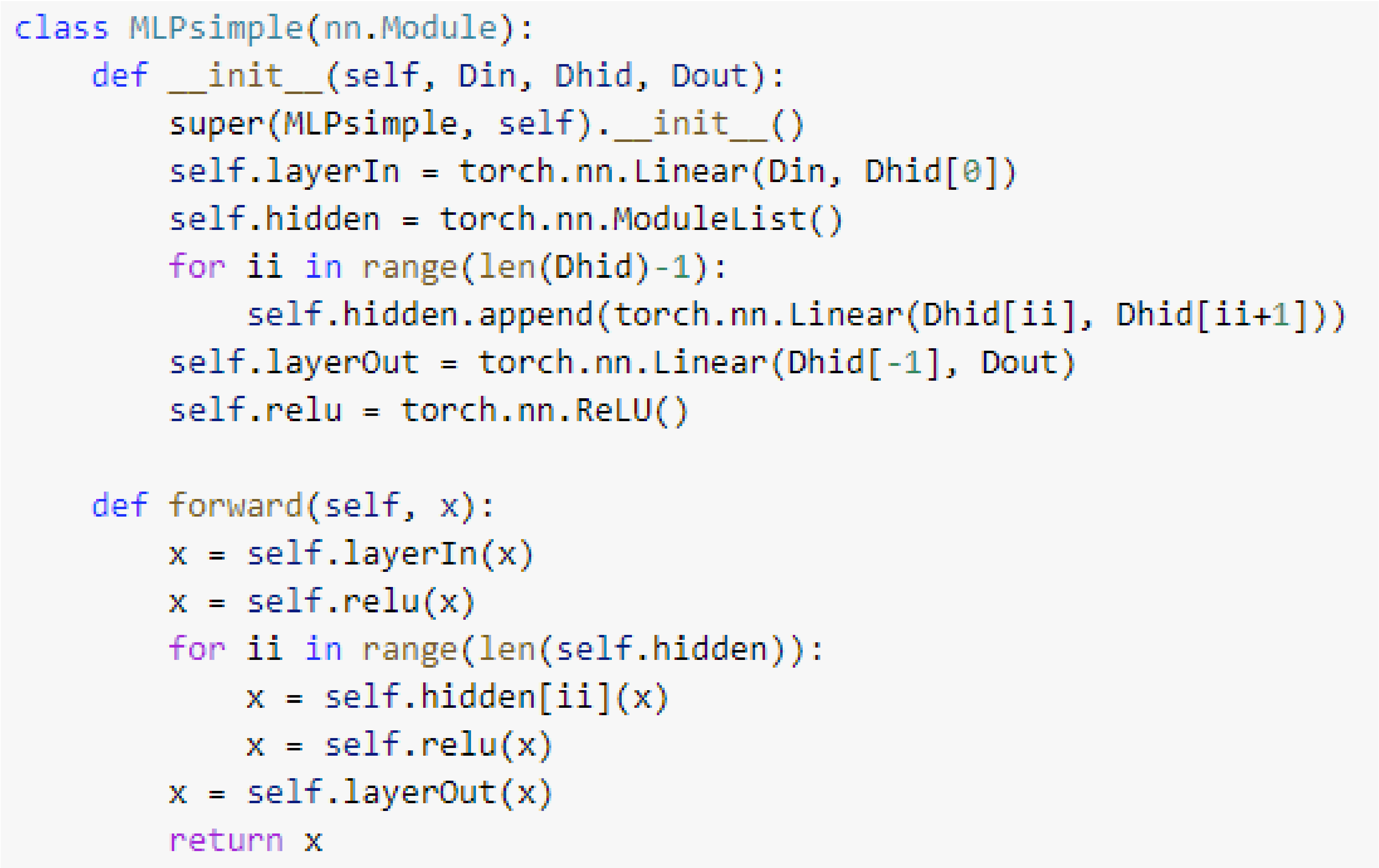
$$\gamma_{(2i+1)}(x) = cos(2^{i-1}\pi x)$$

[Vaswani et al. 2017]

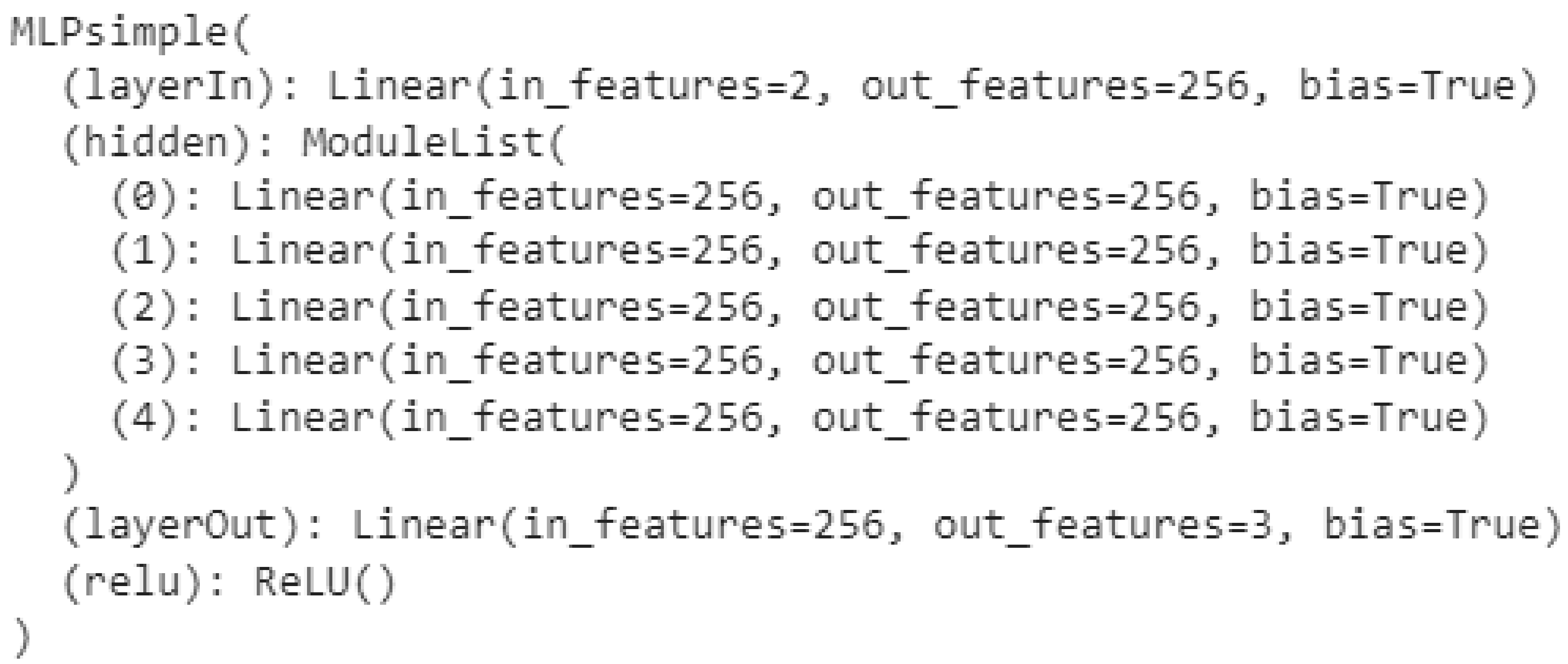
Hands-on

ReLU vs. ReLU+PE vs. SIREN

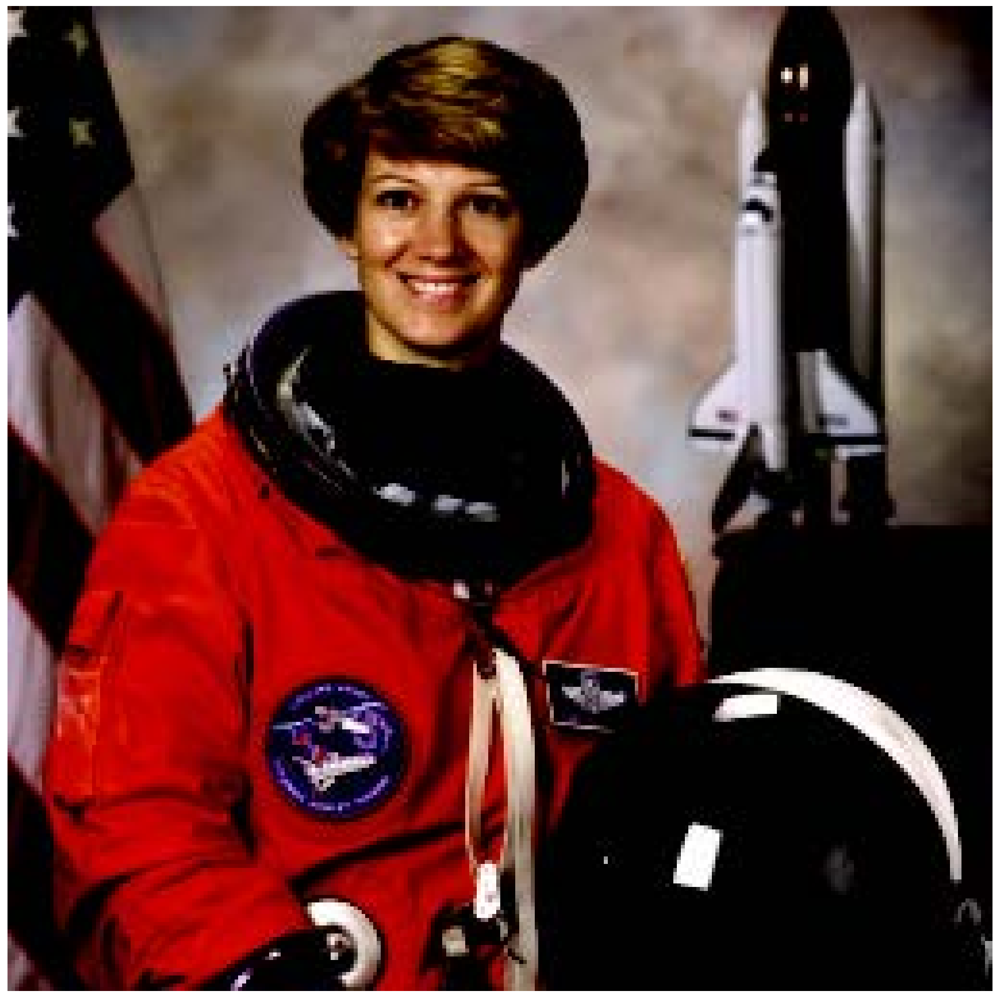
Full colab code



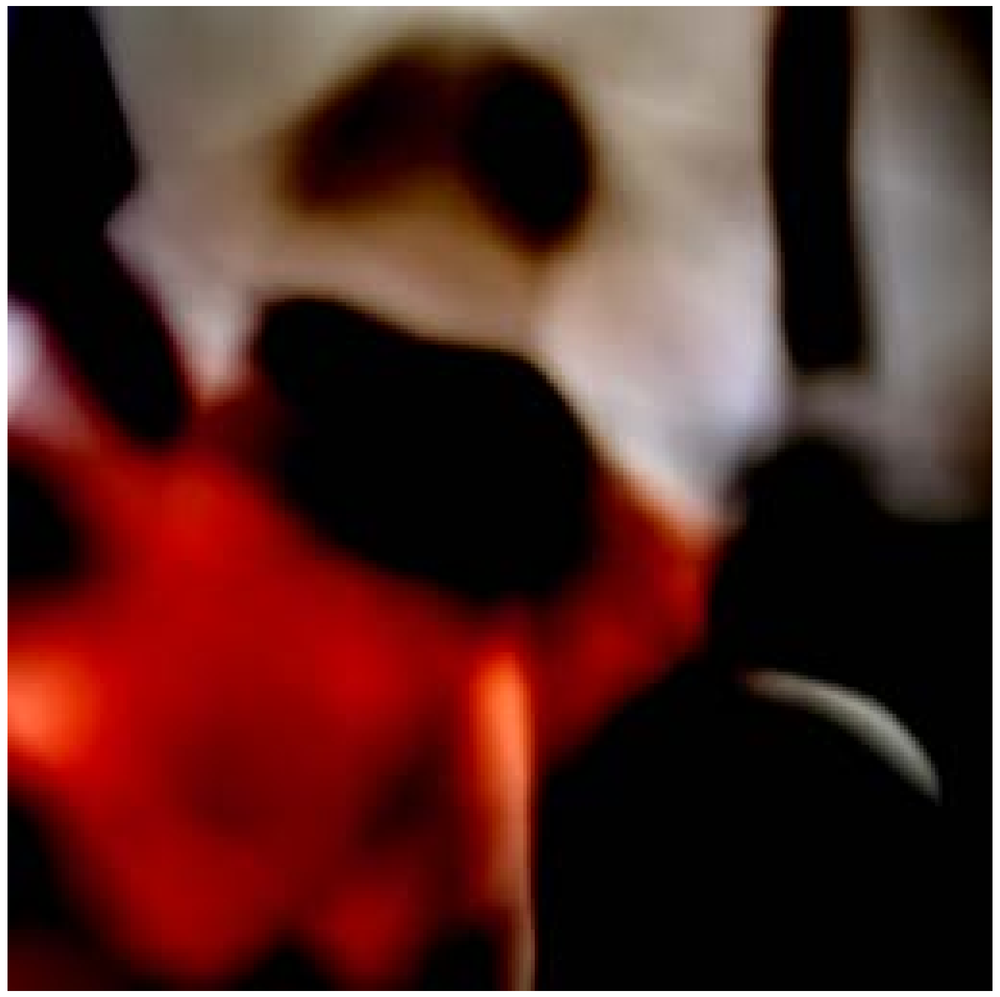
```
class MLPsimple(nn.Module):
    def __init__(self, Din, Dhid, Dout):
        super(MLPsimple, self).__init__()
        self.layerIn = torch.nn.Linear(Din, Dhid[0])
        self.hidden = torch.nn.ModuleList()
        for ii in range(len(Dhid)-1):
            self.hidden.append(torch.nn.Linear(Dhid[ii], Dhid[ii+1]))
        self.layerOut = torch.nn.Linear(Dhid[-1], Dout)
        self.relu = torch.nn.ReLU()

    def forward(self, x):
        x = self.layerIn(x)
        x = self.relu(x)
        for ii in range(len(self.hidden)):
            x = self.hidden[ii](x)
            x = self.relu(x)
        x = self.layerOut(x)
        return x
```



```
MLPsimple(
  (layerIn): Linear(in_features=2, out_features=256, bias=True)
  (hidden): ModuleList(
    (0): Linear(in_features=256, out_features=256, bias=True)
    (1): Linear(in_features=256, out_features=256, bias=True)
    (2): Linear(in_features=256, out_features=256, bias=True)
    (3): Linear(in_features=256, out_features=256, bias=True)
    (4): Linear(in_features=256, out_features=256, bias=True)
  )
  (layerOut): Linear(in_features=256, out_features=3, bias=True)
  (relu): ReLU()
)
```



GT

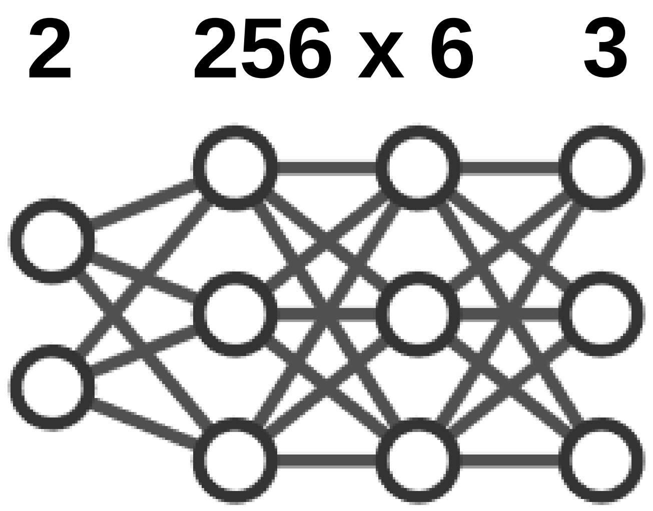


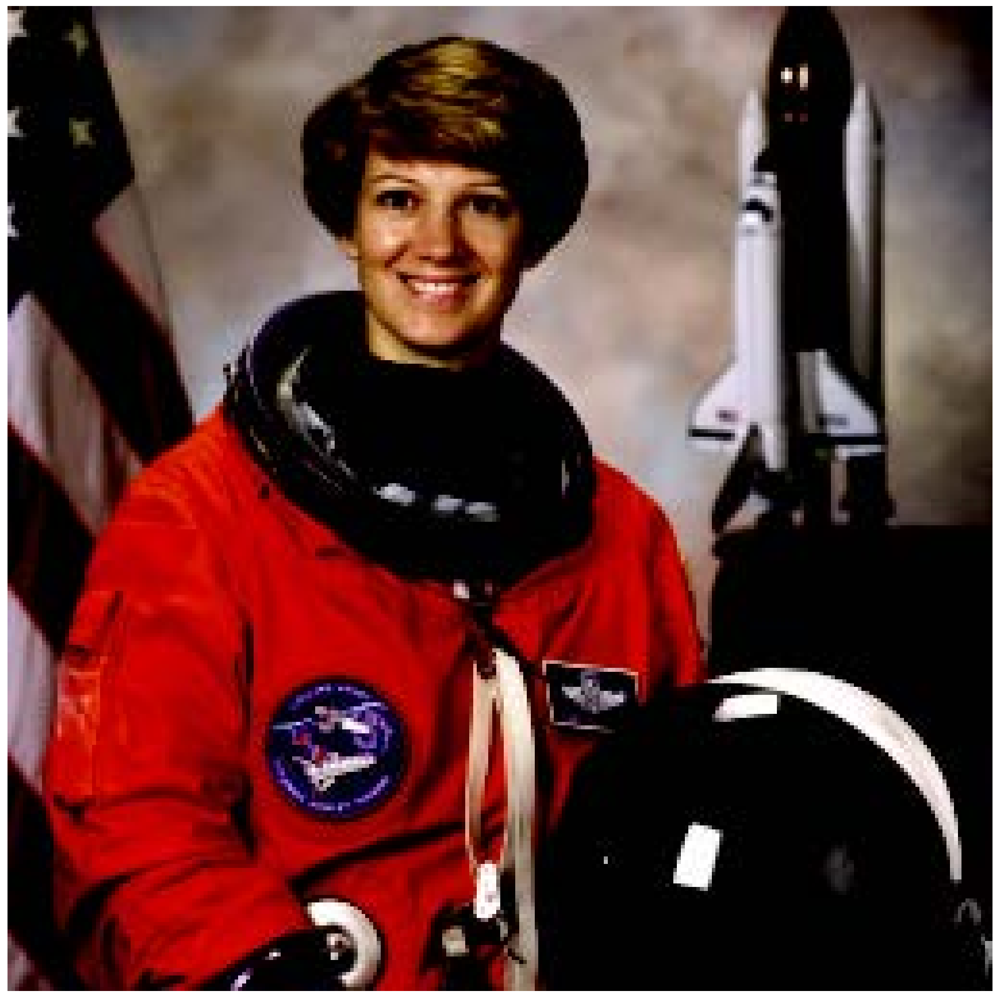
ReLU
result after
1000 epochs

Hands-on

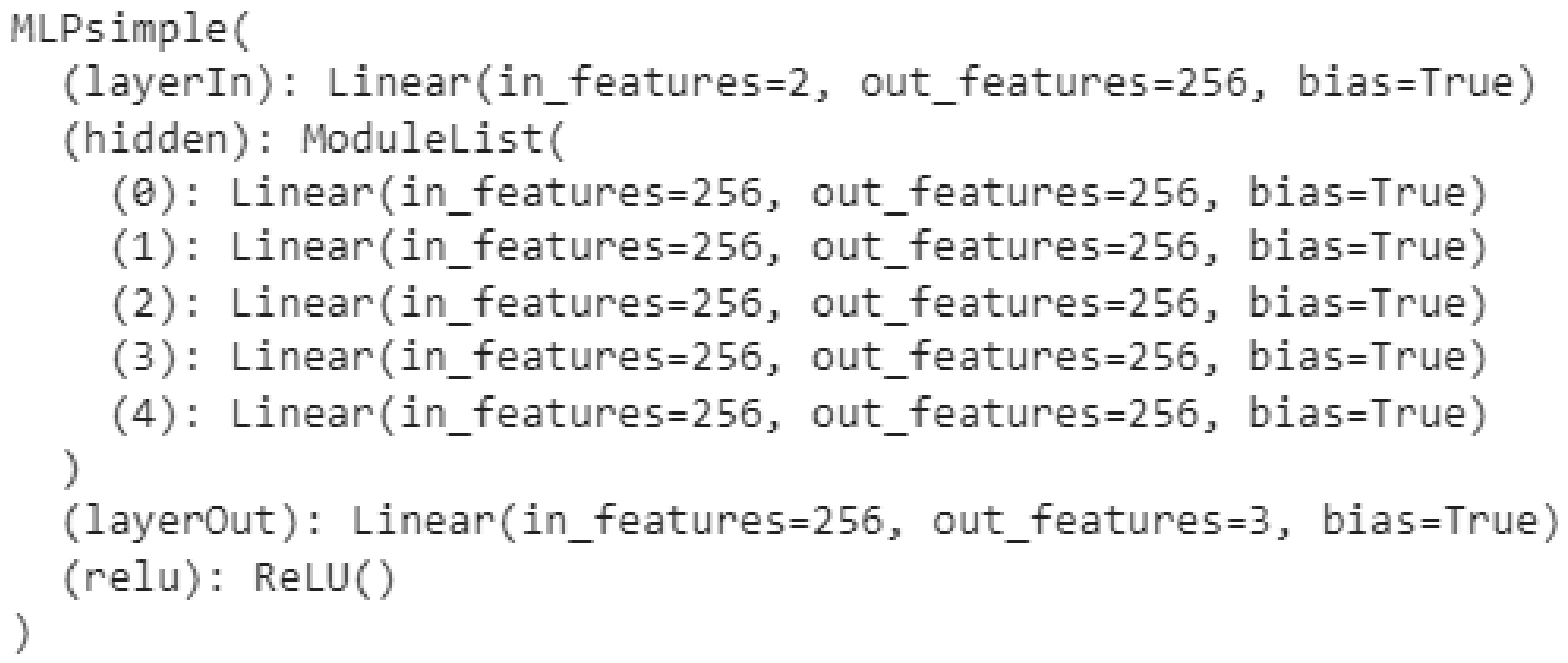
ReLU vs. ReLU+PE vs. SIREN

Full colab code

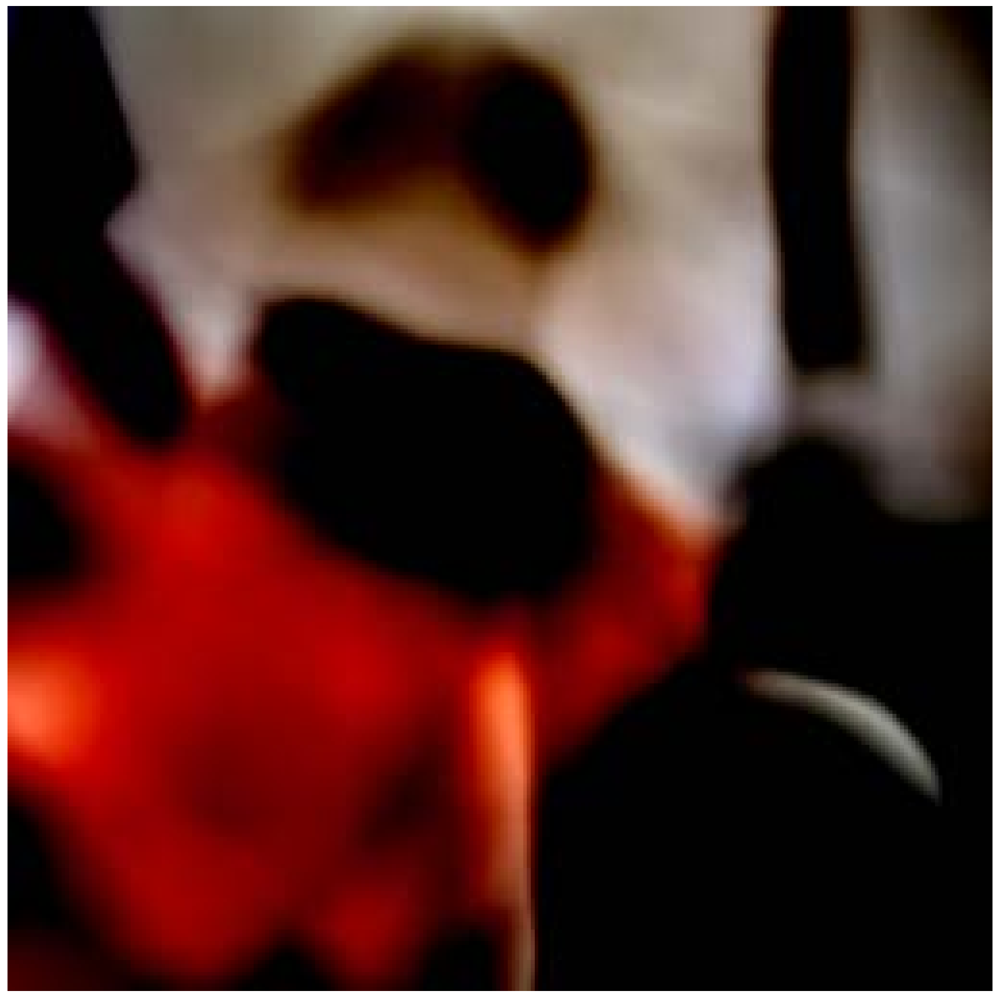




GT



```
MLPsimple(
  (layerIn): Linear(in_features=2, out_features=256, bias=True)
  (hidden): ModuleList(
    (0): Linear(in_features=256, out_features=256, bias=True)
    (1): Linear(in_features=256, out_features=256, bias=True)
    (2): Linear(in_features=256, out_features=256, bias=True)
    (3): Linear(in_features=256, out_features=256, bias=True)
    (4): Linear(in_features=256, out_features=256, bias=True)
  )
  (layerOut): Linear(in_features=256, out_features=3, bias=True)
  (relu): ReLU()
)
```

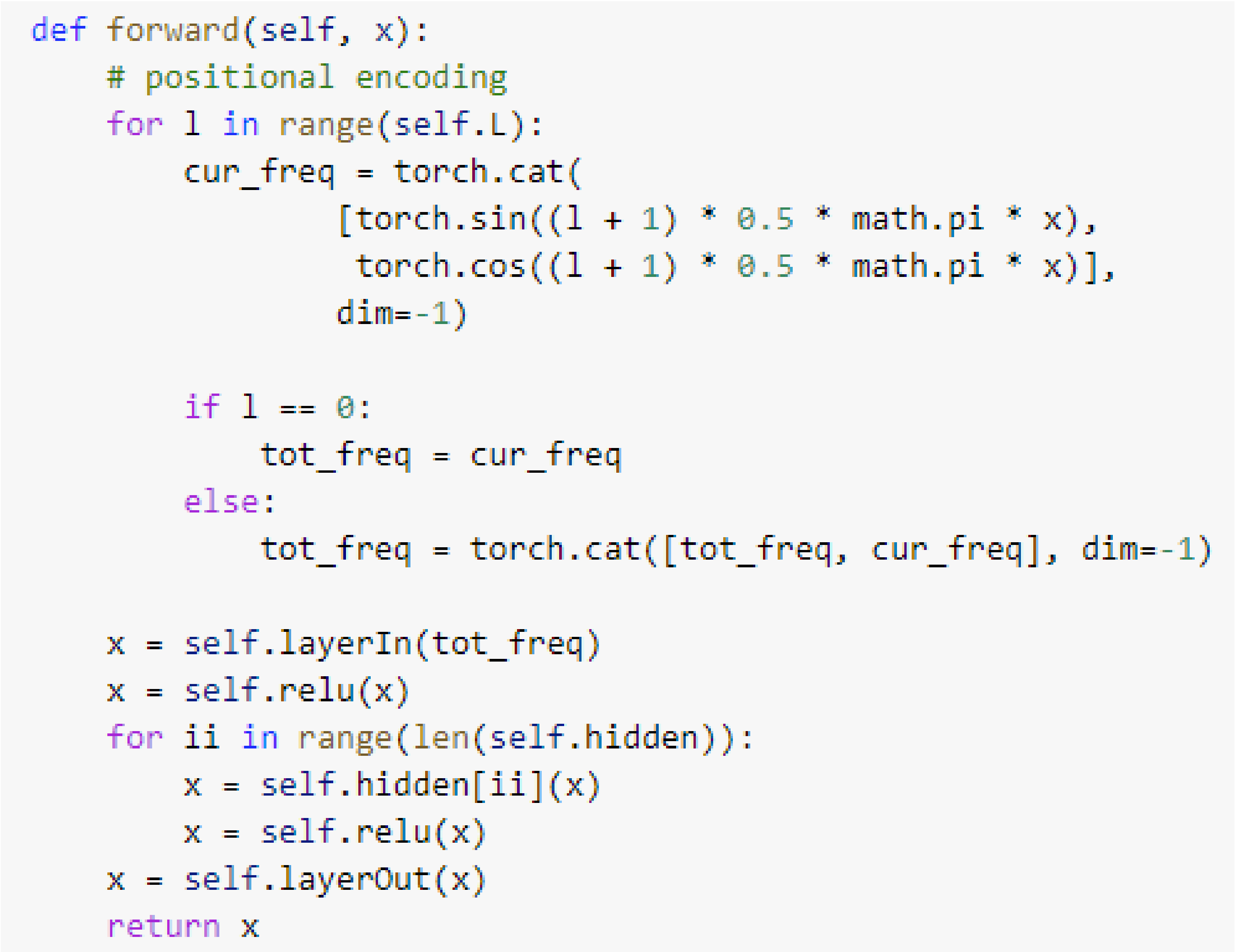


ReLU
result after
1000 epochs

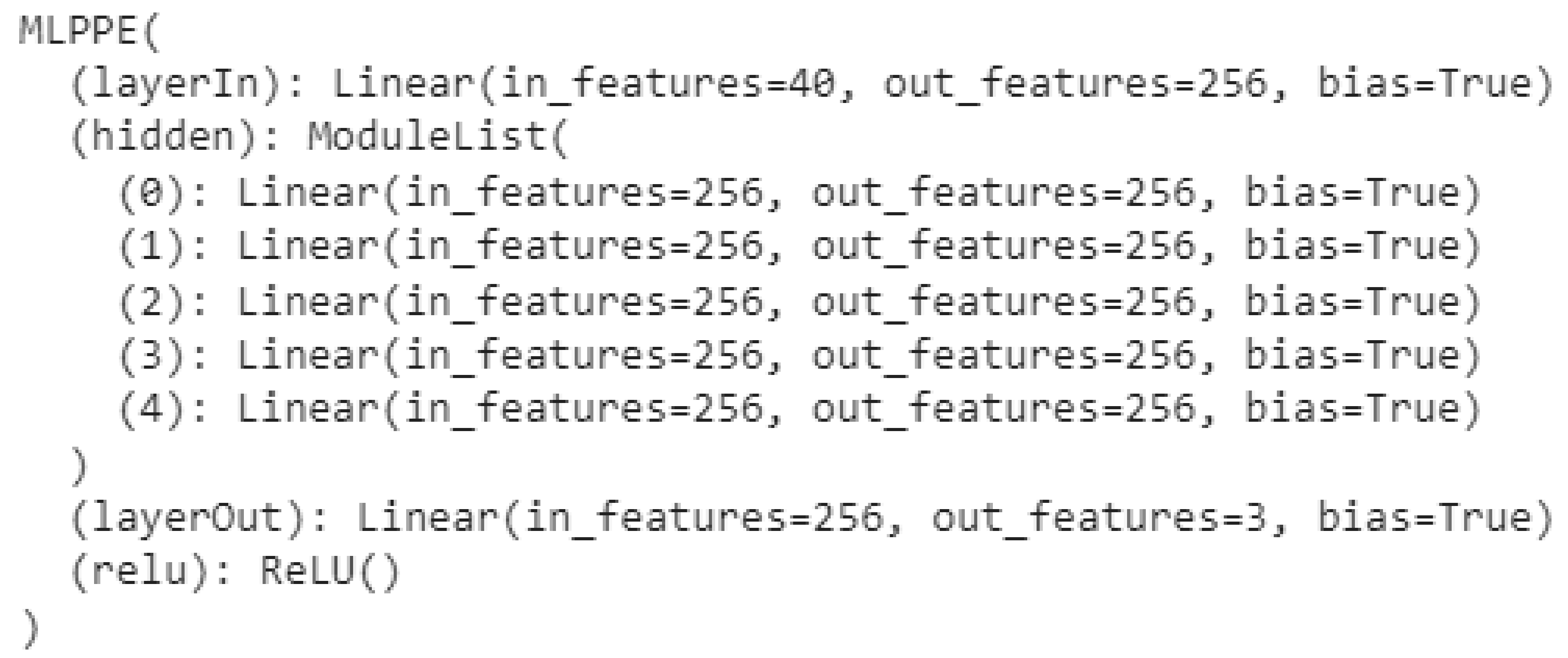
Hands-on

ReLU vs. ReLU+PE vs. SIREN

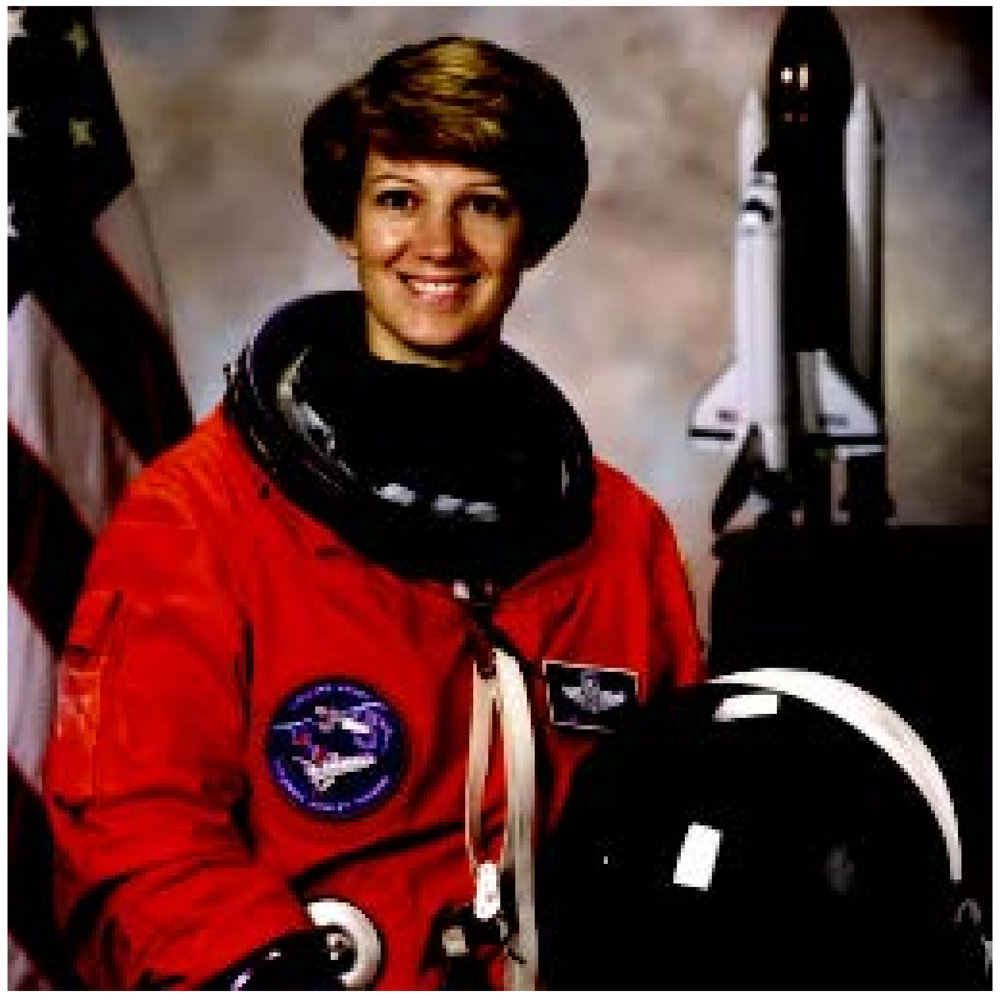
Full colab code



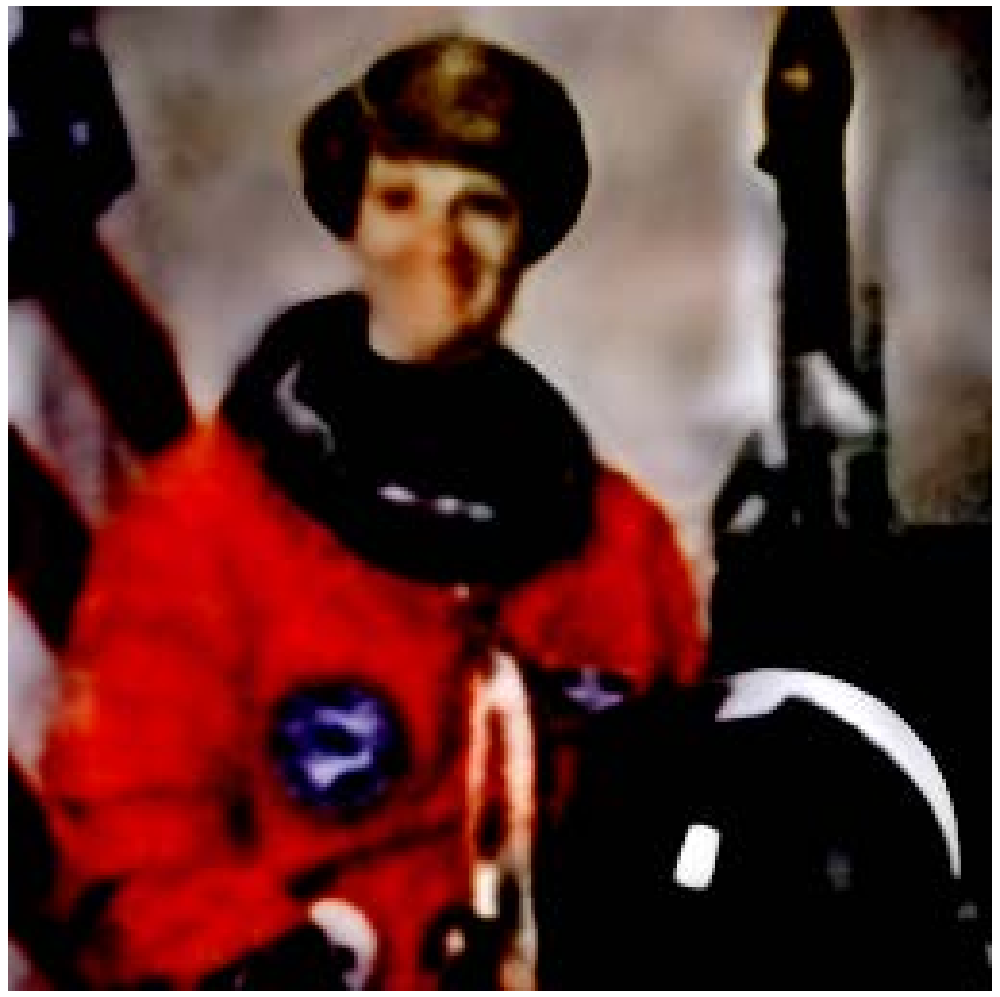
```
def forward(self, x):
    # positional encoding
    for l in range(self.L):
        cur_freq = torch.cat(
                [torch.sin((l + 1) * 0.5 * math.pi * x),
                 torch.cos((l + 1) * 0.5 * math.pi * x)],
                dim=-1)

        if l == 0:
            tot_freq = cur_freq
        else:
            tot_freq = torch.cat([tot_freq, cur_freq], dim=-1)

    x = self.layerIn(tot_freq)
    x = self.relu(x)
    for ii in range(len(self.hidden)):
        x = self.hidden[ii](x)
        x = self.relu(x)
    x = self.layerOut(x)
    return x
```



```
MLPPE(
  (layerIn): Linear(in_features=40, out_features=256, bias=True)
  (hidden): ModuleList(
    (0): Linear(in_features=256, out_features=256, bias=True)
    (1): Linear(in_features=256, out_features=256, bias=True)
    (2): Linear(in_features=256, out_features=256, bias=True)
    (3): Linear(in_features=256, out_features=256, bias=True)
    (4): Linear(in_features=256, out_features=256, bias=True)
  )
  (layerOut): Linear(in_features=256, out_features=3, bias=True)
  (relu): ReLU()
)
```



GT

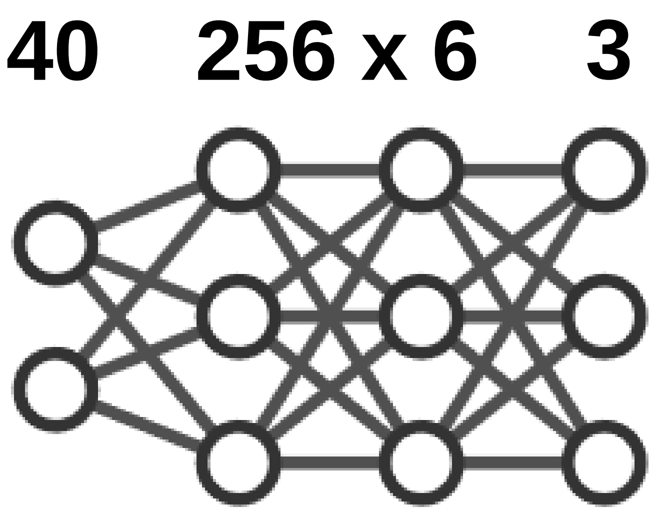


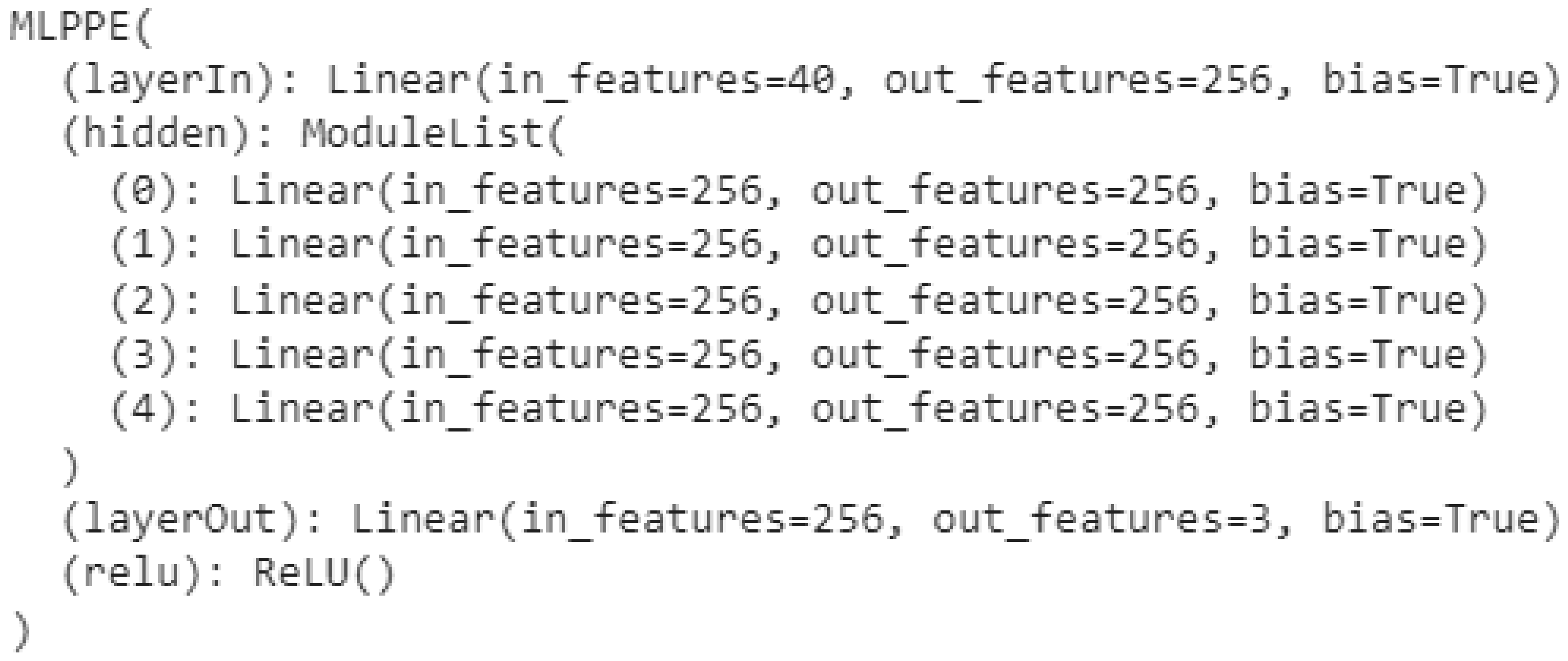
ReLU+PE
result after
1000 epochs

Hands-on

ReLU vs. ReLU+PE vs. SIREN

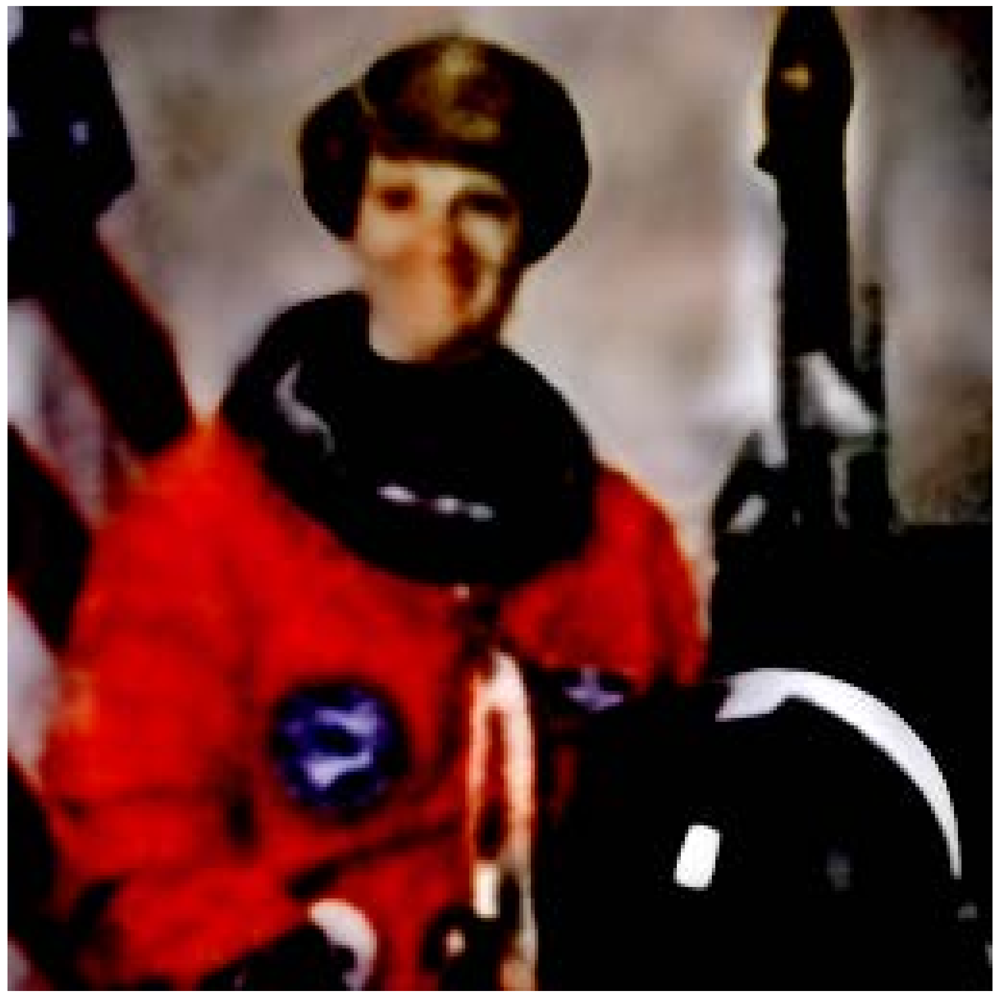
Full colab code





```
MLPPE(
  (layerIn): Linear(in_features=40, out_features=256, bias=True)
  (hidden): ModuleList(
    (0): Linear(in_features=256, out_features=256, bias=True)
    (1): Linear(in_features=256, out_features=256, bias=True)
    (2): Linear(in_features=256, out_features=256, bias=True)
    (3): Linear(in_features=256, out_features=256, bias=True)
    (4): Linear(in_features=256, out_features=256, bias=True)
  )
  (layerOut): Linear(in_features=256, out_features=3, bias=True)
  (relu): ReLU()
)
```

GT

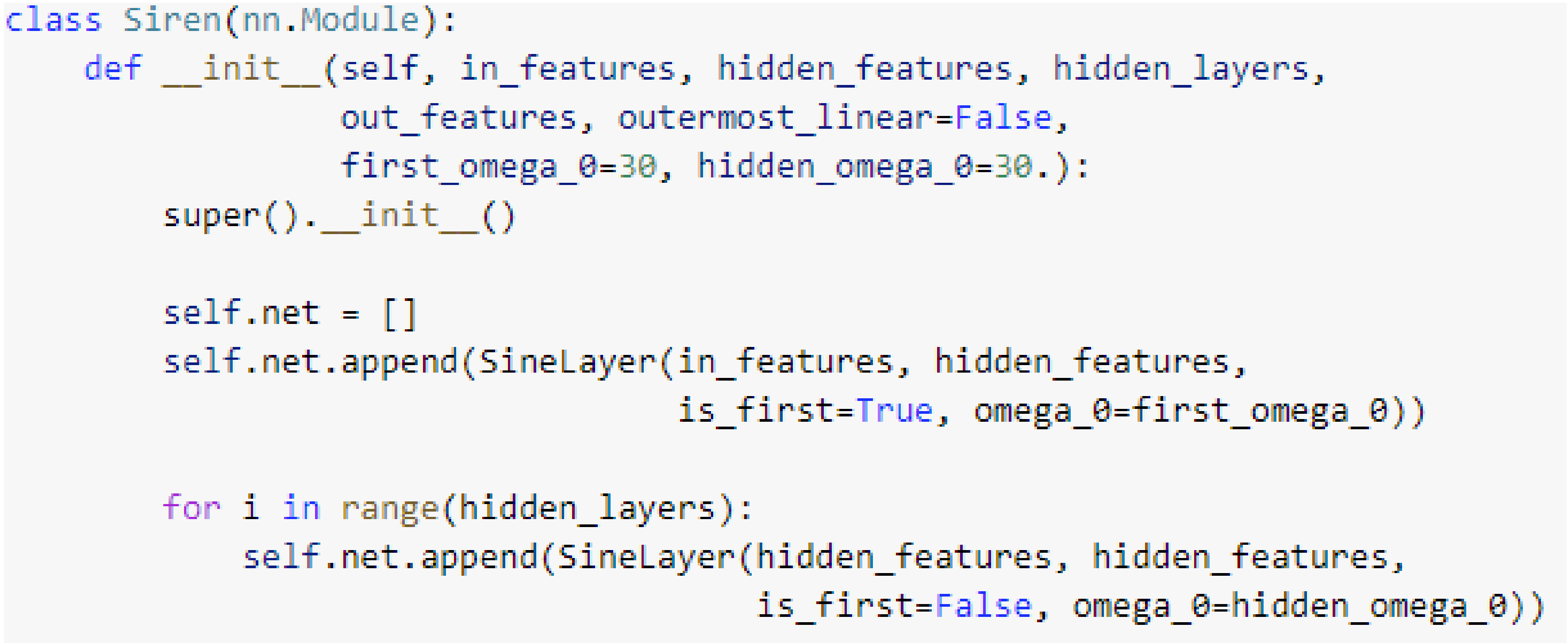


ReLU+PE
result after
1000 epochs

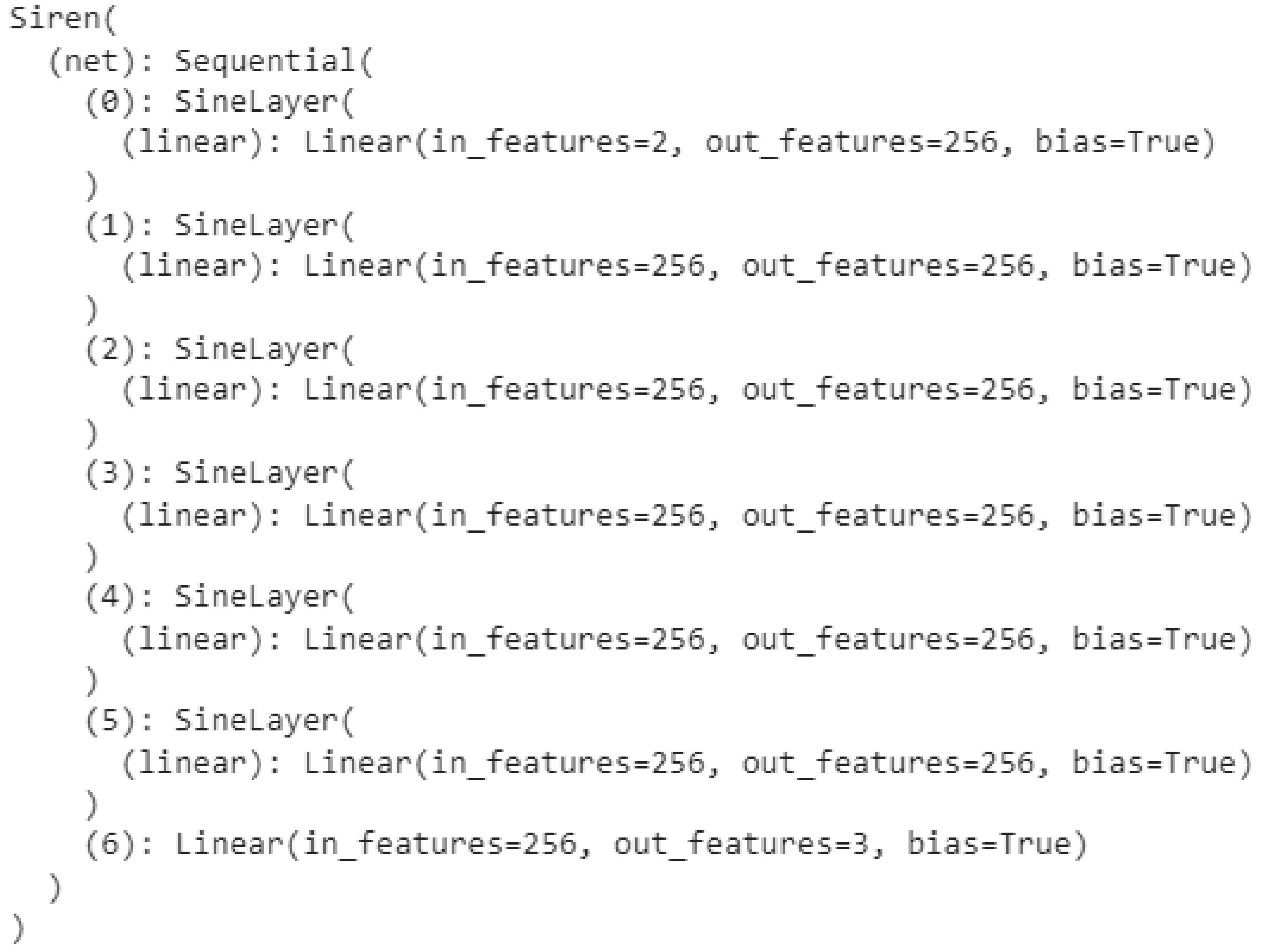
Hands-on

ReLU vs. ReLU+PE vs. SIREN

Full colab code

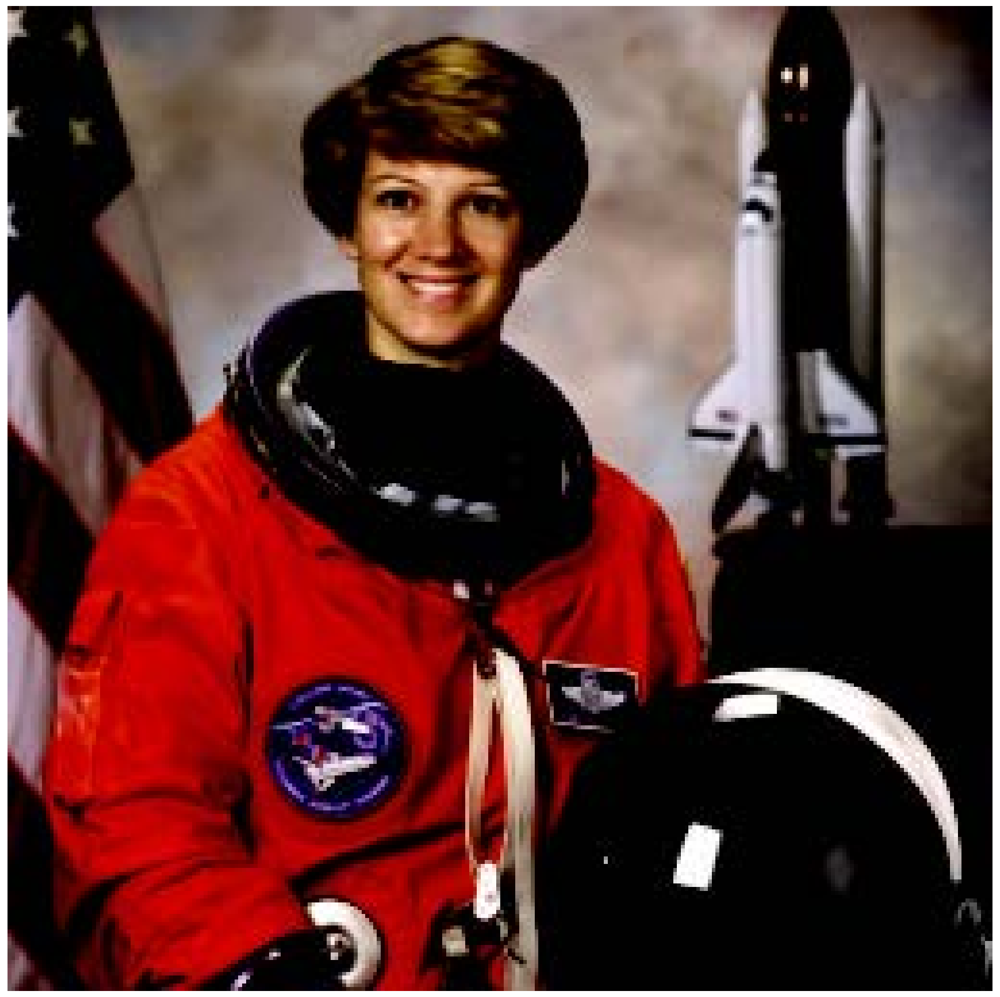


```
class Siren(nn.Module):
    def __init__(self, in_features, hidden_features, hidden_layers,
                 out_features, outermost_linear=False,
                 first_omega_0=30, hidden_omega_0=30.):
        super().__init__()

        self.net = []
        self.net.append(SineLayer(in_features, hidden_features,
                                  is_first=True, omega_0=first_omega_0))

        for i in range(hidden_layers):
            self.net.append(SineLayer(hidden_features, hidden_features,
                                      is_first=False, omega_0=hidden_omega_0))
```



```
Siren(
  (net): Sequential(
    (0): SineLayer(
      (linear): Linear(in_features=2, out_features=256, bias=True)
    )
    (1): SineLayer(
      (linear): Linear(in_features=256, out_features=256, bias=True)
    )
    (2): SineLayer(
      (linear): Linear(in_features=256, out_features=256, bias=True)
    )
    (3): SineLayer(
      (linear): Linear(in_features=256, out_features=256, bias=True)
    )
    (4): SineLayer(
      (linear): Linear(in_features=256, out_features=256, bias=True)
    )
    (5): SineLayer(
      (linear): Linear(in_features=256, out_features=256, bias=True)
    )
    (6): Linear(in_features=256, out_features=3, bias=True)
  )
)
```

GT

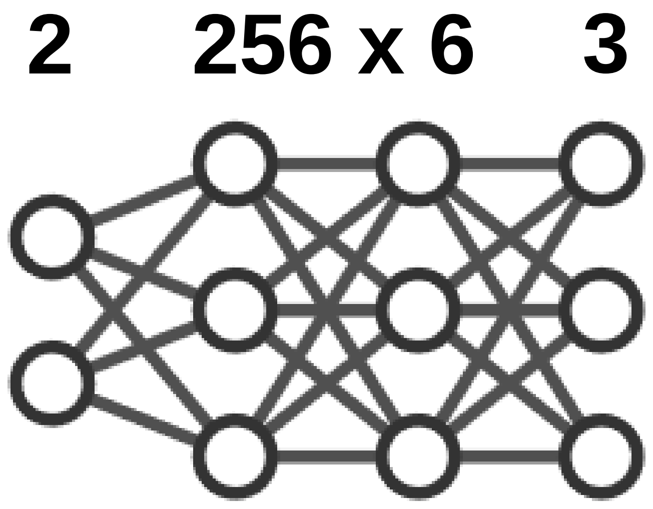


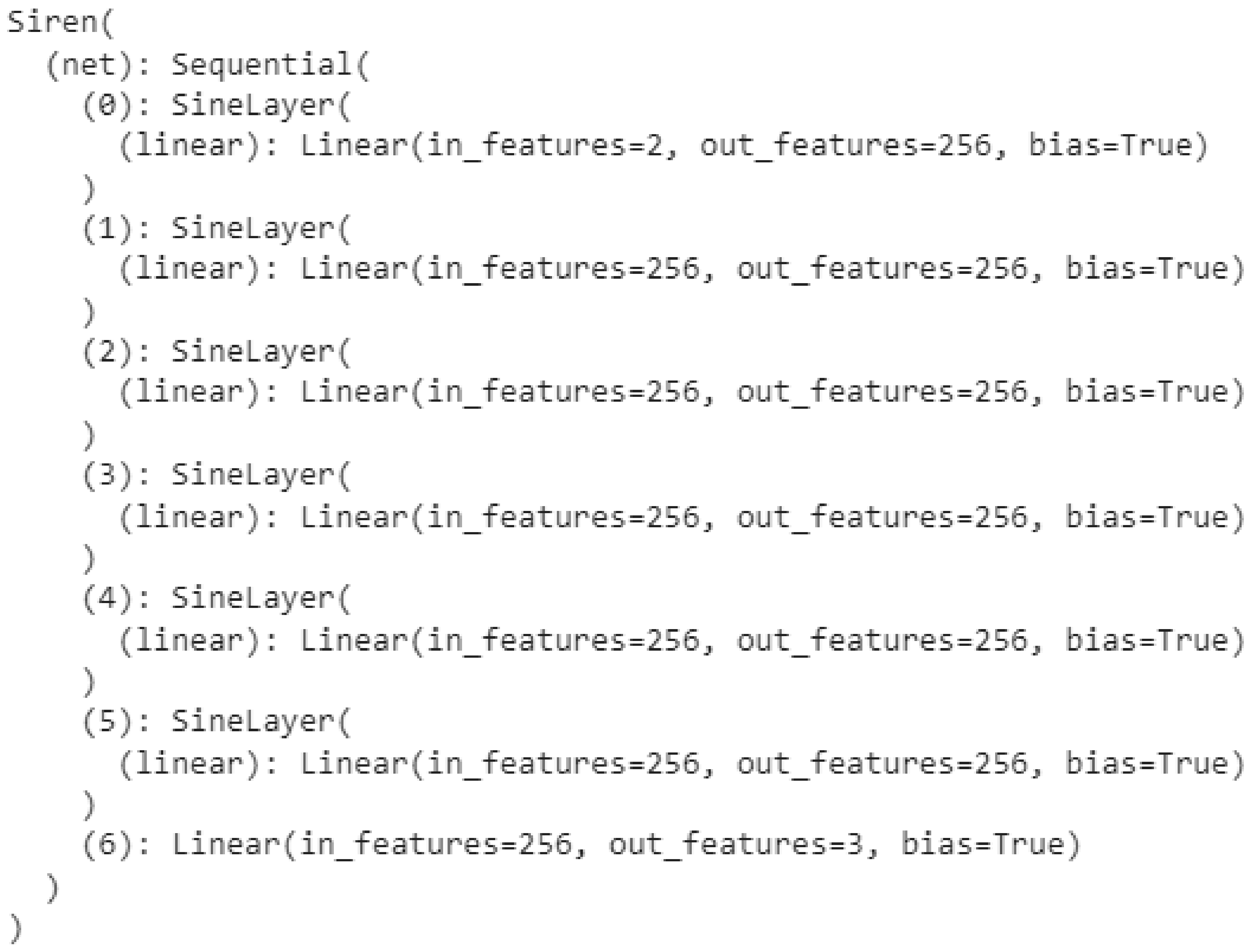
SIREN
result after
1000 epochs

Hands-on

ReLU vs. ReLU+PE vs. SIREN

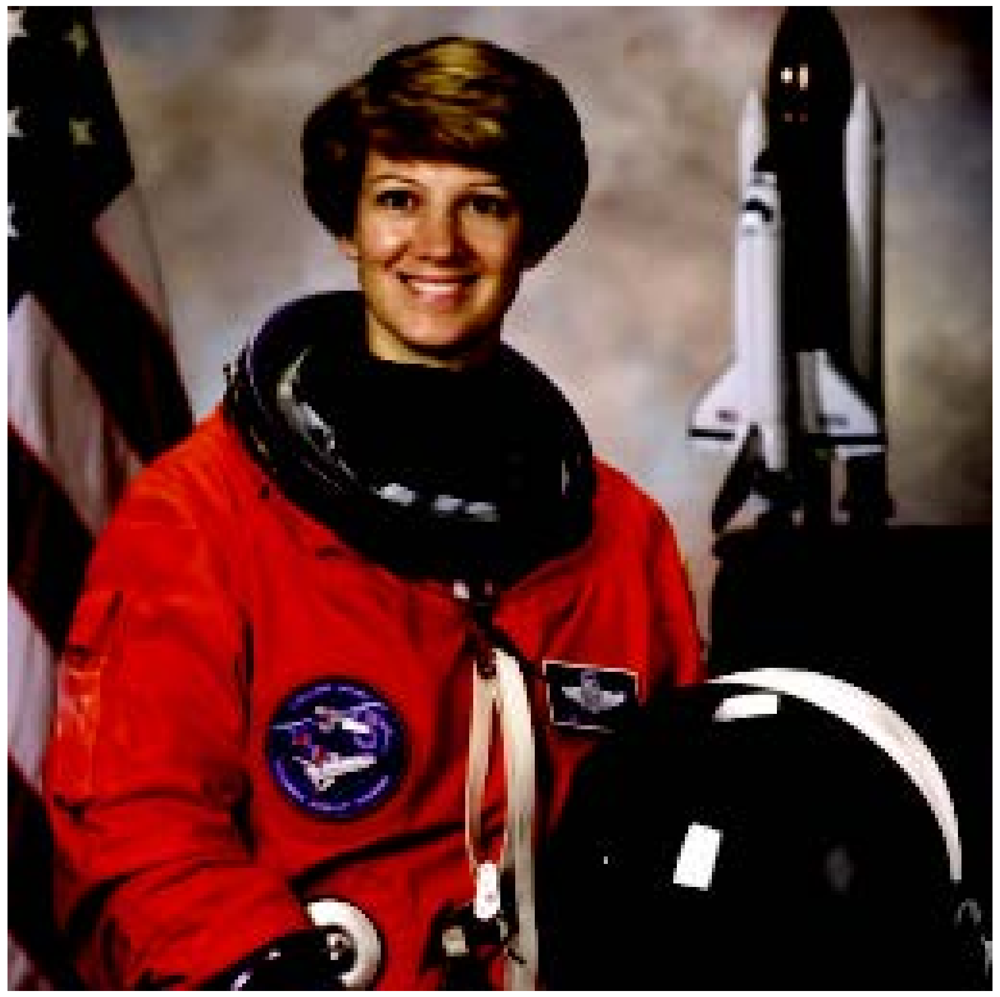
Full colab code





```
Siren(
  (net): Sequential(
    (0): SineLayer(
      (linear): Linear(in_features=2, out_features=256, bias=True)
    )
    (1): SineLayer(
      (linear): Linear(in_features=256, out_features=256, bias=True)
    )
    (2): SineLayer(
      (linear): Linear(in_features=256, out_features=256, bias=True)
    )
    (3): SineLayer(
      (linear): Linear(in_features=256, out_features=256, bias=True)
    )
    (4): SineLayer(
      (linear): Linear(in_features=256, out_features=256, bias=True)
    )
    (5): SineLayer(
      (linear): Linear(in_features=256, out_features=256, bias=True)
    )
    (6): Linear(in_features=256, out_features=3, bias=True)
  )
)
```

GT

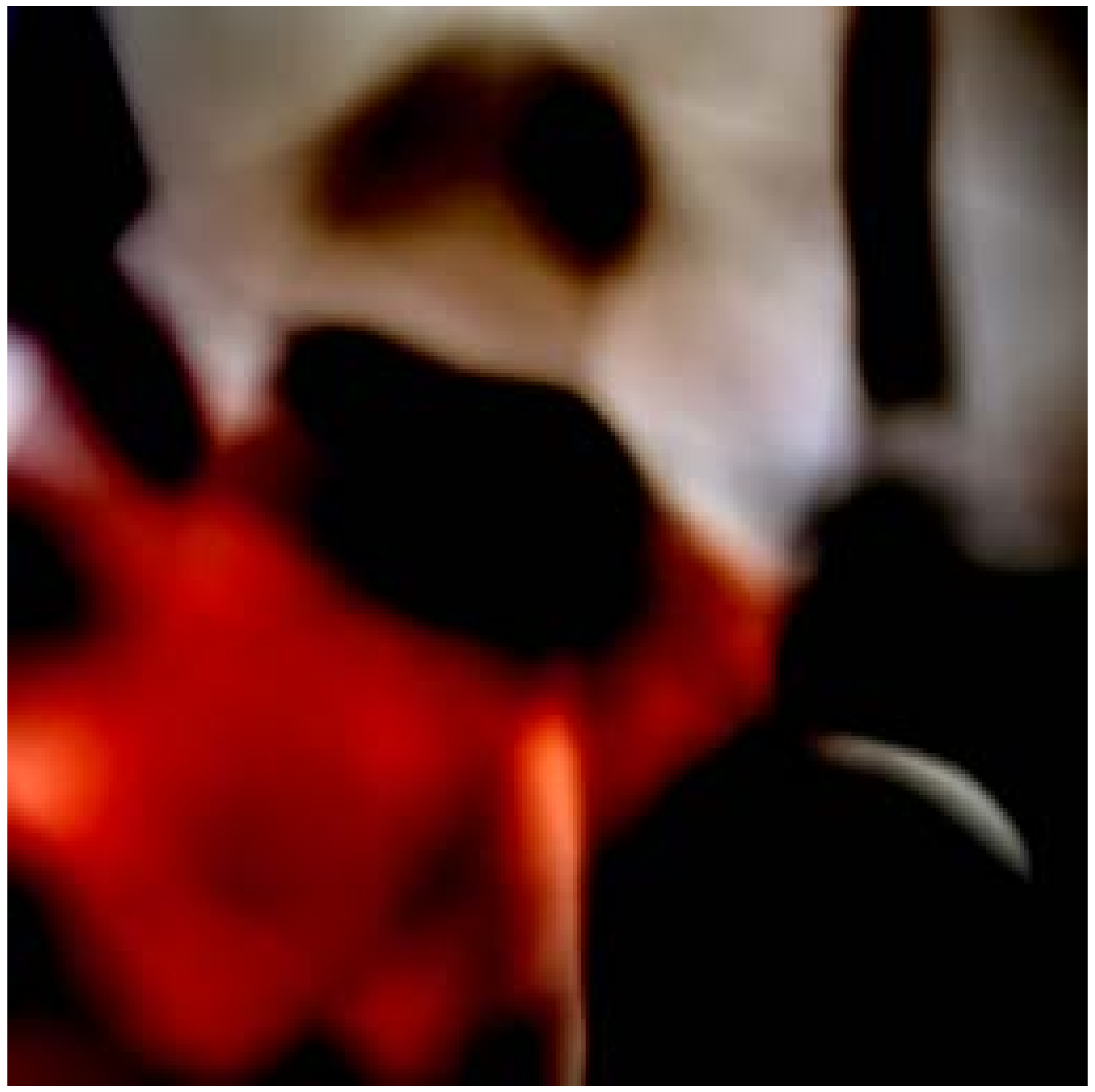


SIREN
result after
1000 epochs

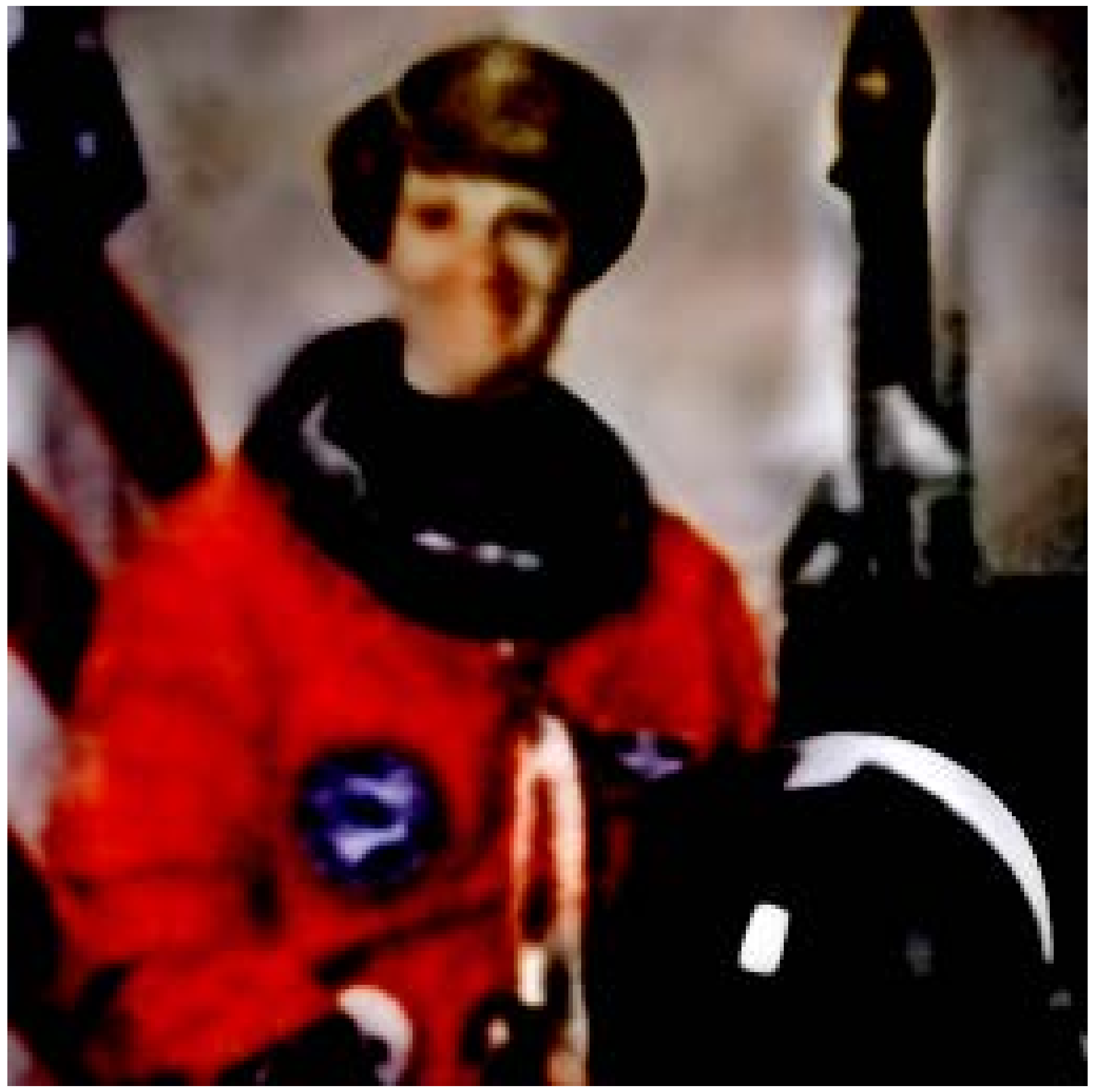
Hands-on

Comparison

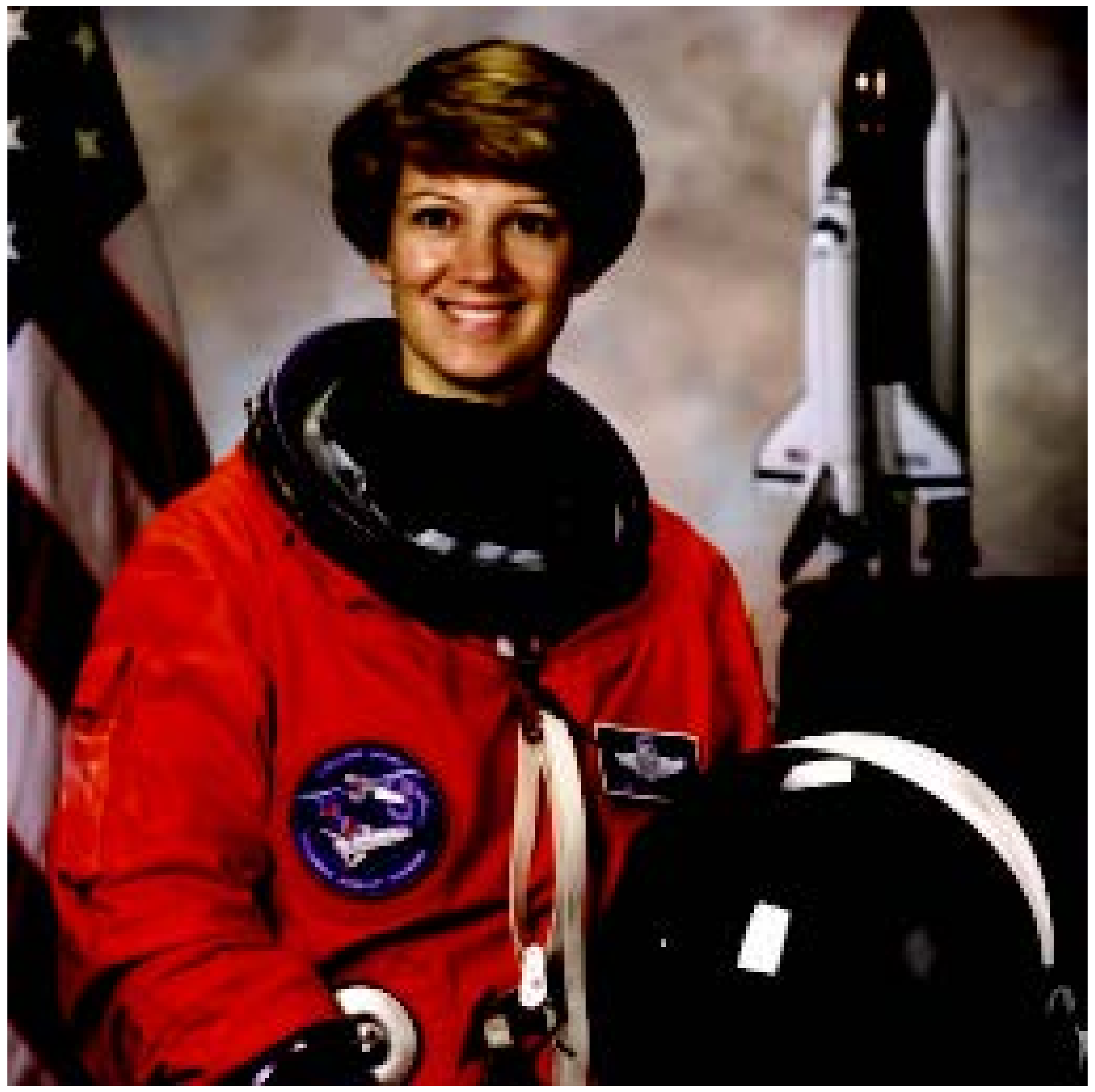
Full colab code



ReLU



ReLU+PE

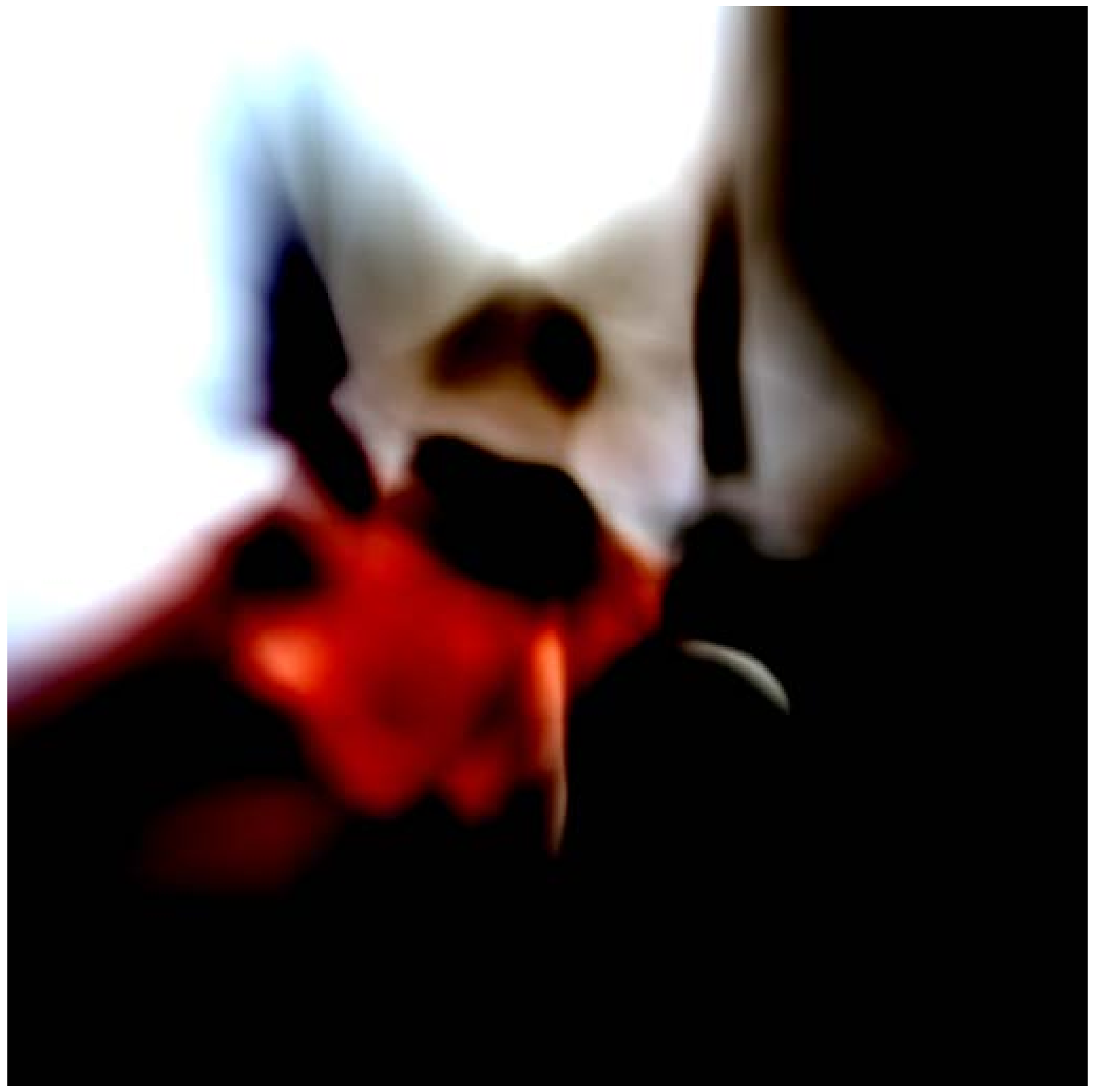


SIREN

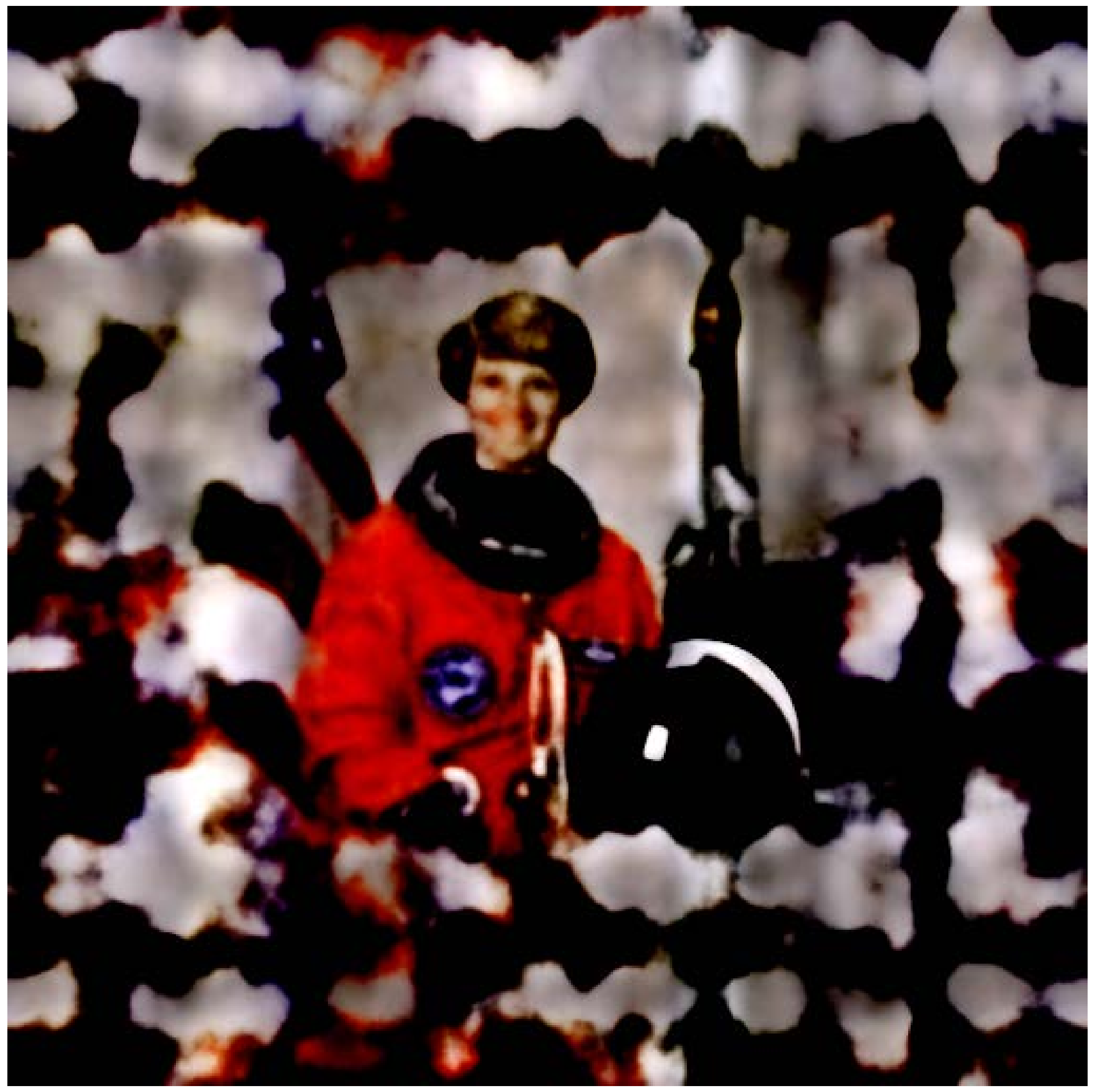
Hands-on

Extrapolation

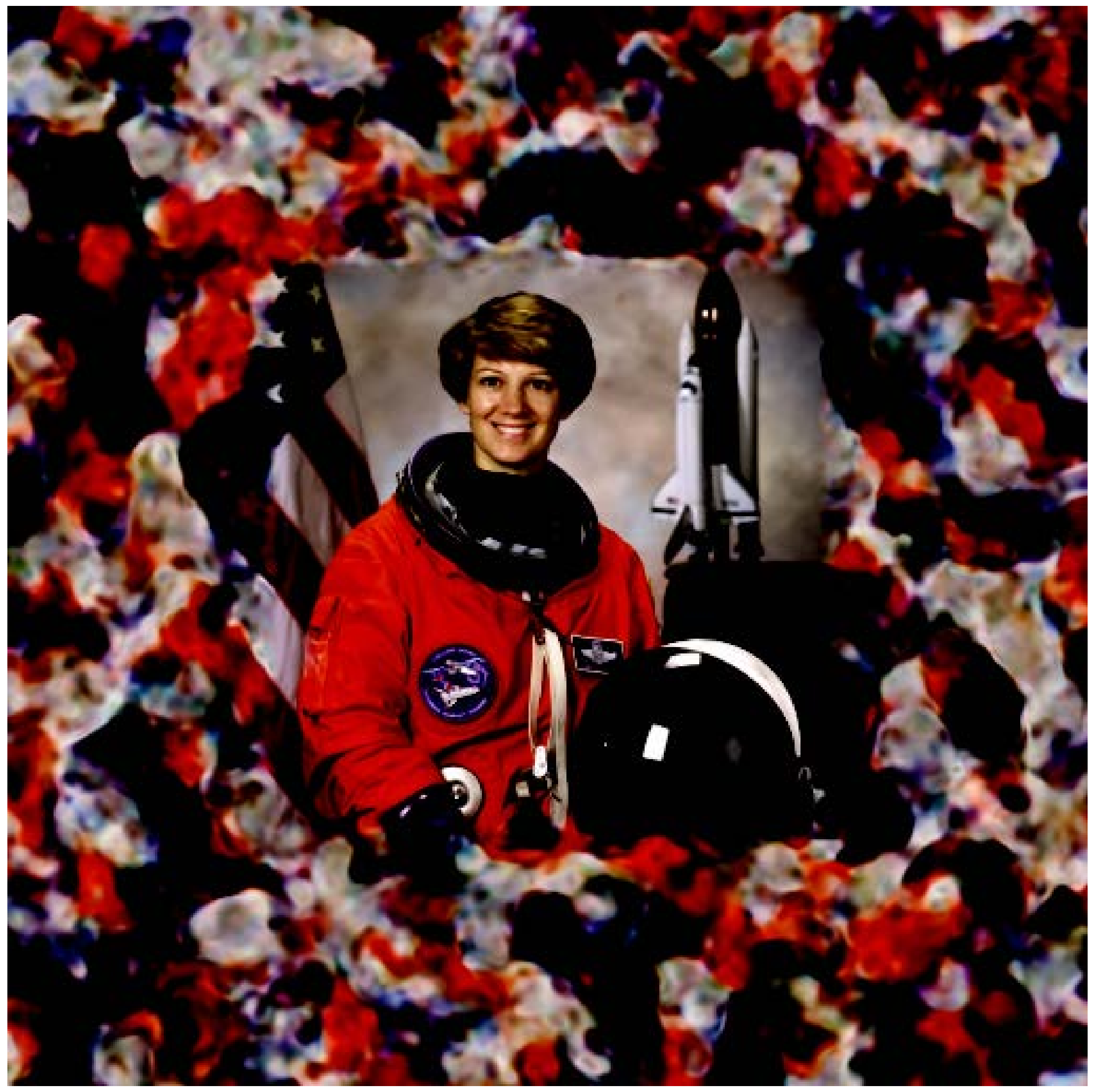
Full colab code



ReLU

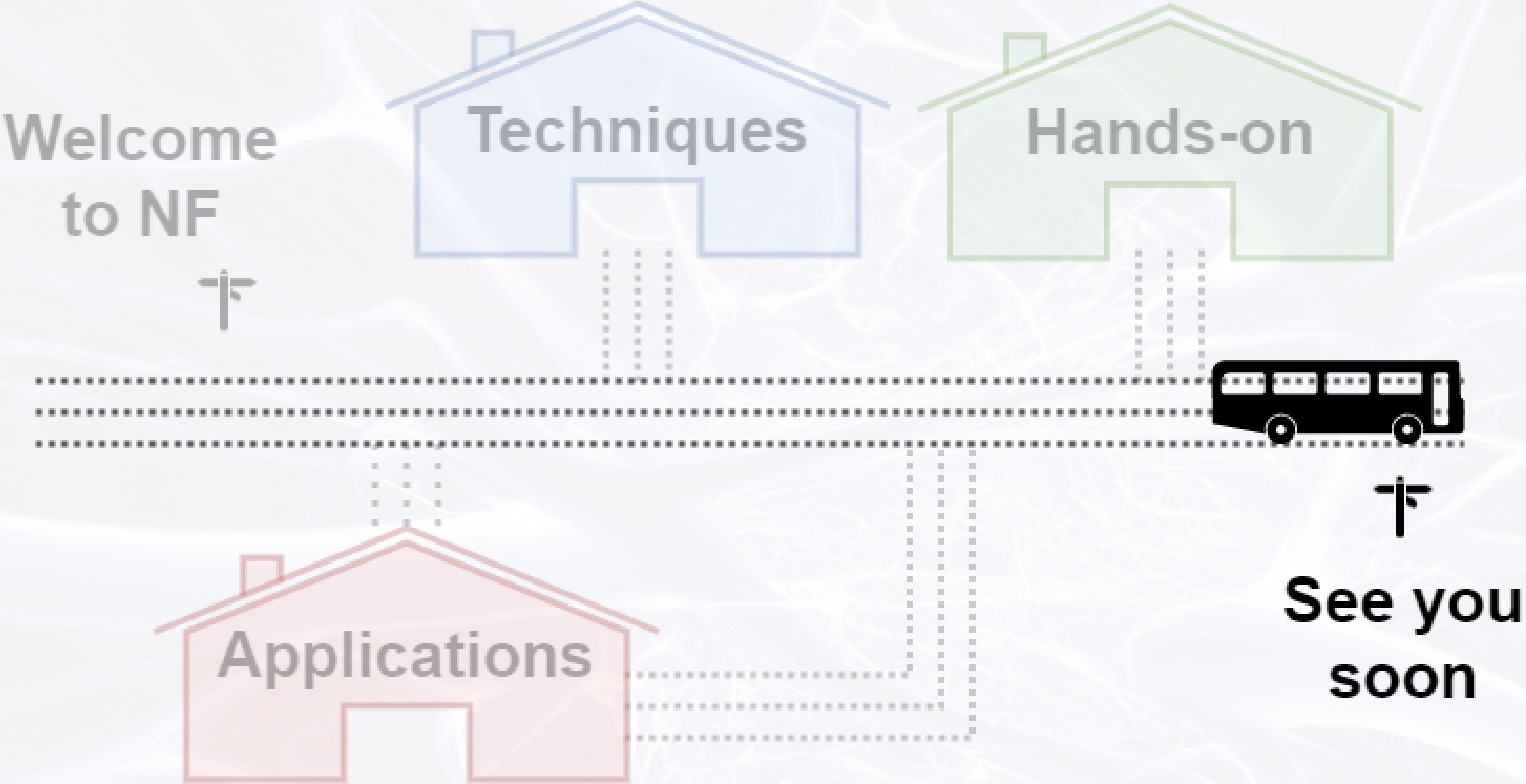


ReLU+PE



SIREN

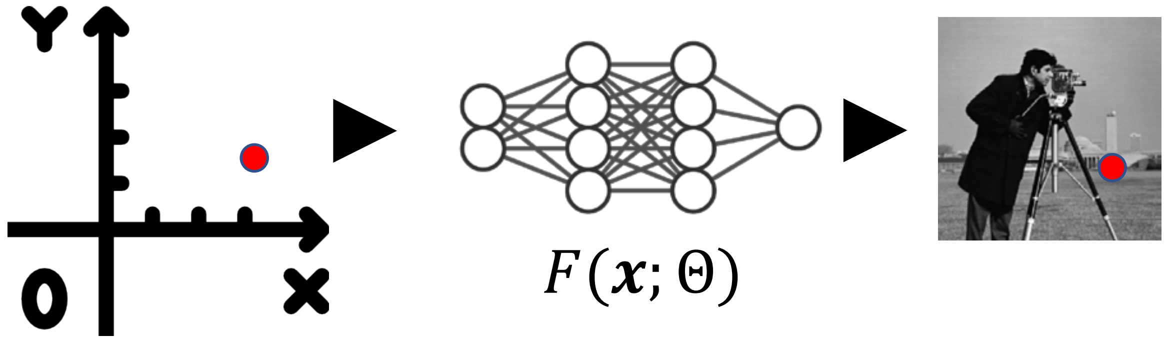
Let's go on a tour to NF



See you soon

What?

Neural fields

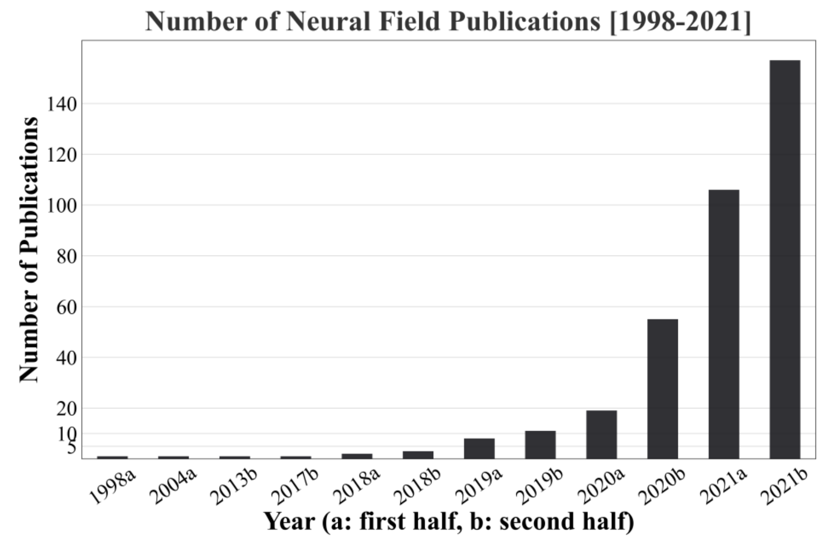


What for?

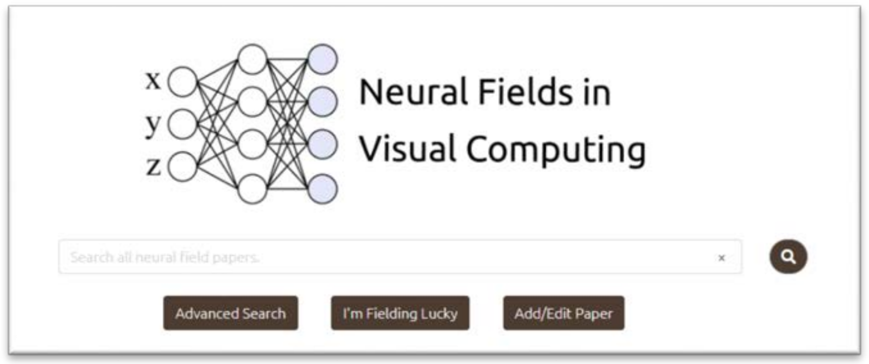
Continuous, compact, regularized, domain-agnostic

So what?

New paradigm with great potential



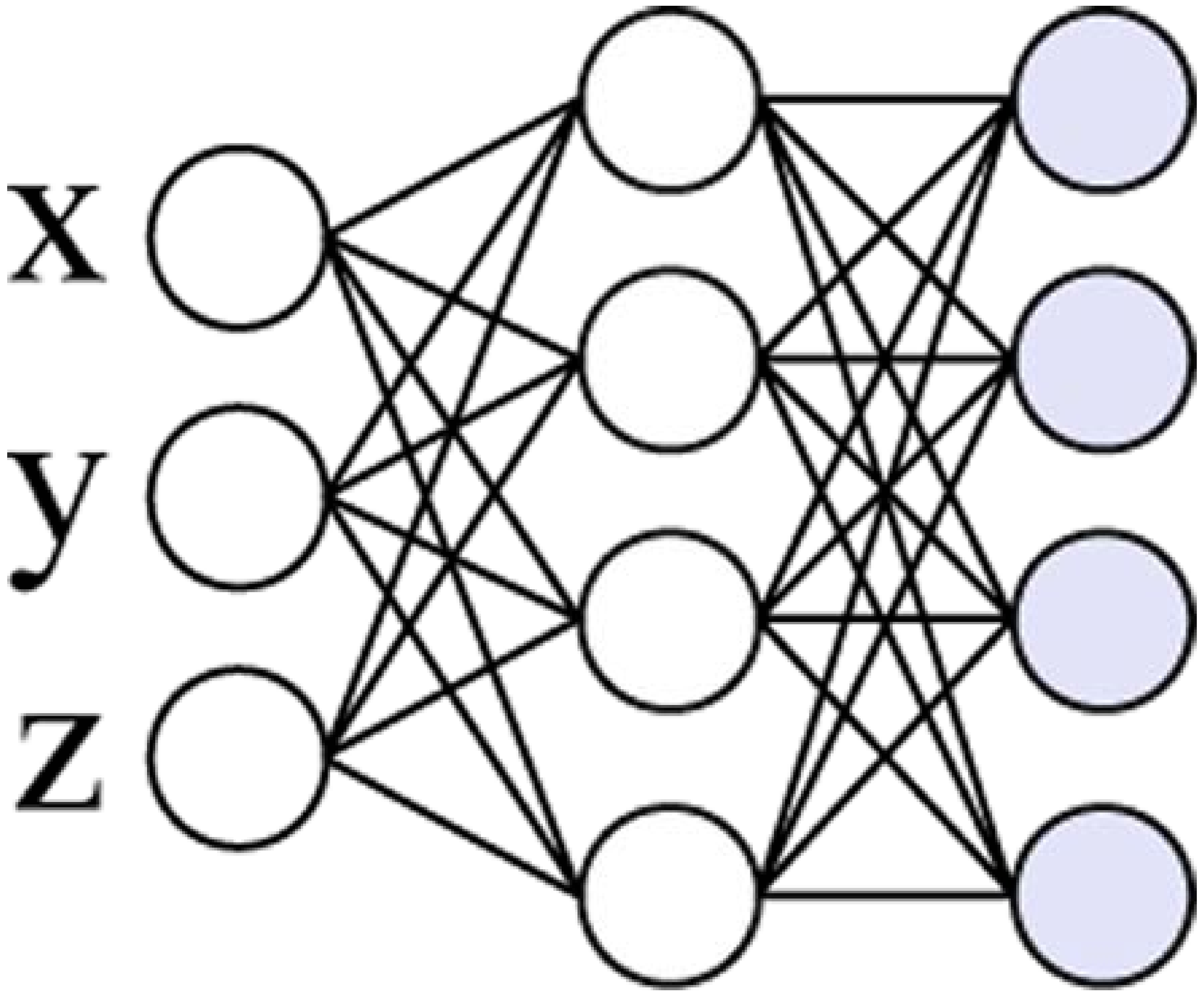
Now what?

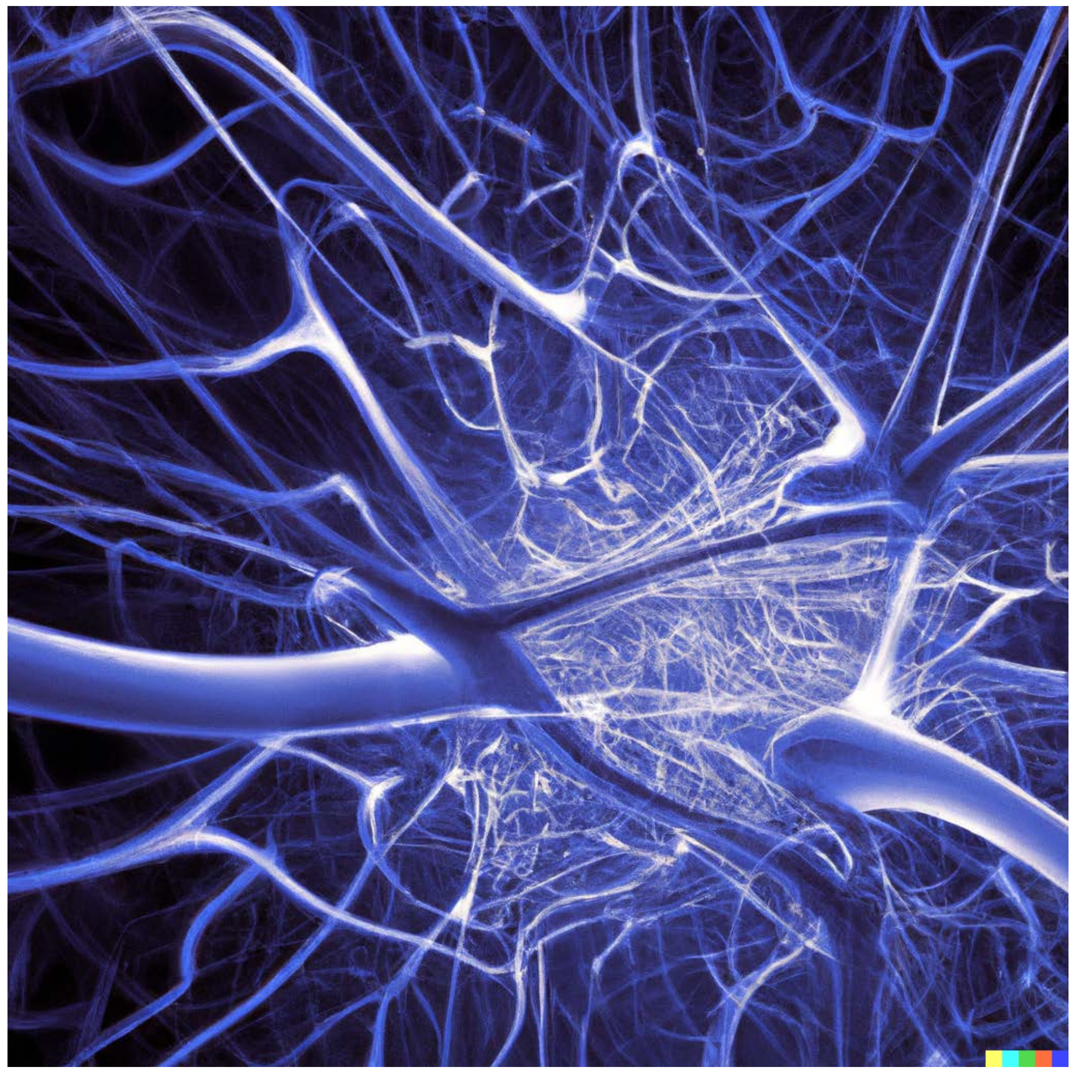

Neural Fields in Visual Computing

neuralfields.cs.brown.edu

Thanks

Tomáš Kerepecký
kerepecky@utia.cas.cz





By DALL-E

Applications

NeRF

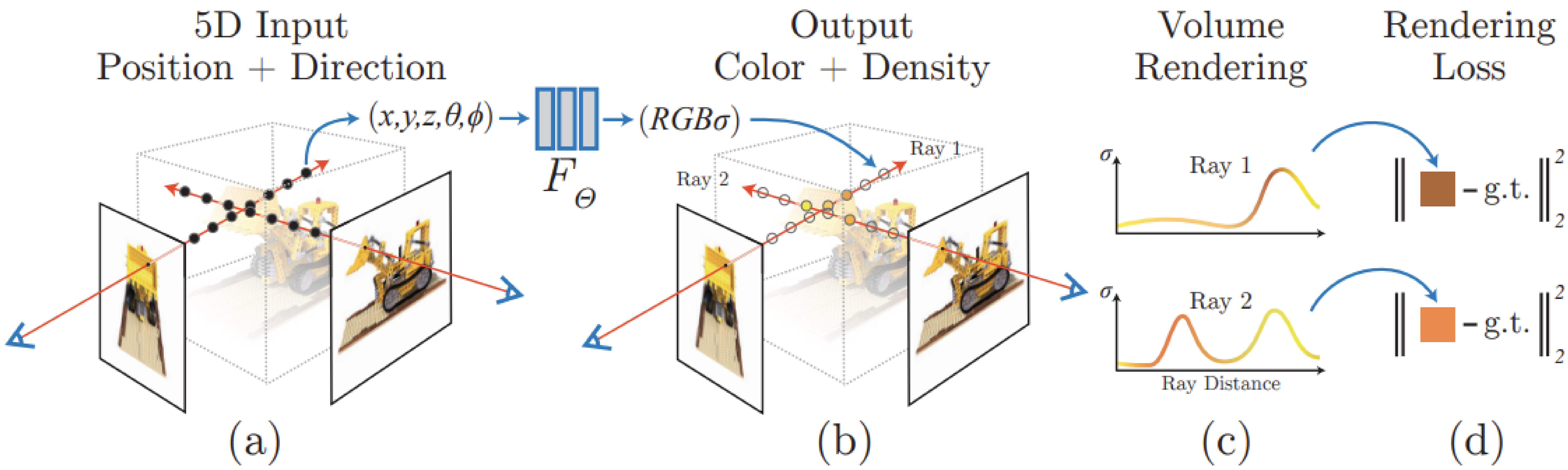
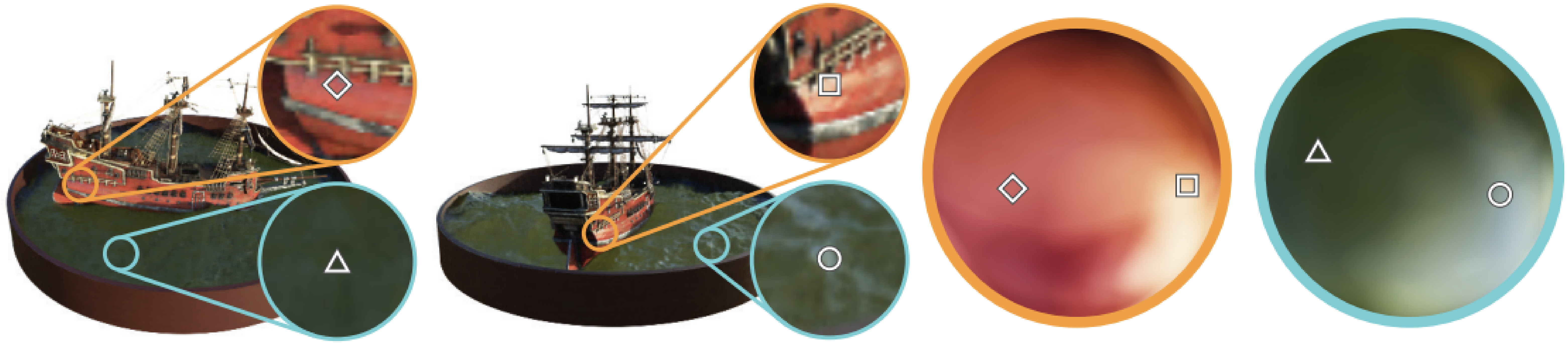


Fig. 2: An overview of our neural radiance field scene representation and differentiable rendering procedure. We synthesize images by sampling 5D coordinates (location and viewing direction) along camera rays (a), feeding those locations into an MLP to produce a color and volume density (b), and using volume rendering techniques to composite these values into an image (c). This rendering function is differentiable, so we can optimize our scene representation by minimizing the residual between synthesized and ground truth observed images (d).

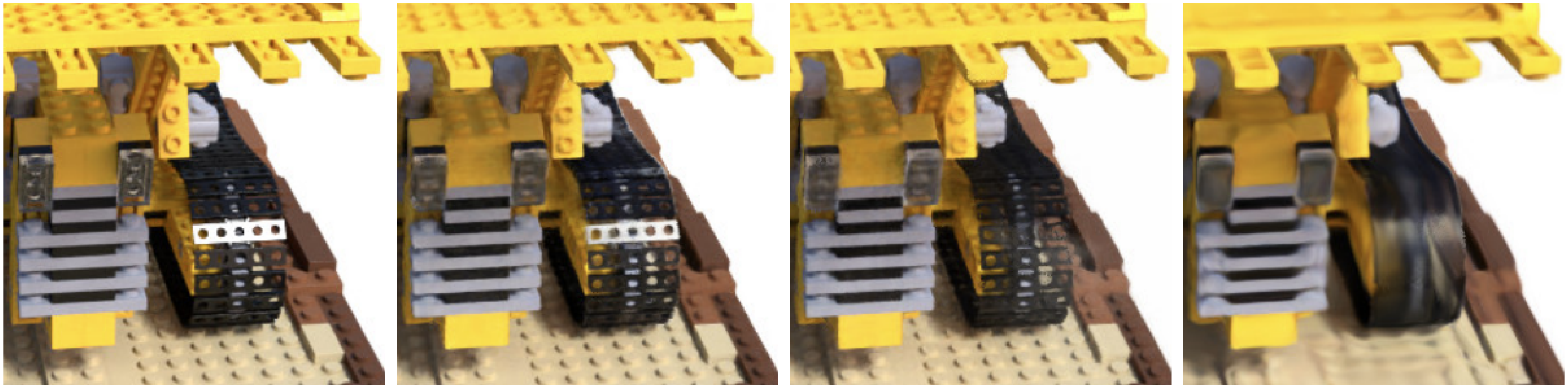
NeRF



(a) View 1 (b) View 2 (c) Radiance Distributions

Fig. 3: A visualization of view-dependent emitted radiance. Our neural radiance field representation outputs RGB color as a 5D function of both spatial position **x** and viewing direction **d**. Here, we visualize example directional color distributions for two spatial locations in our neural representation of the *Ship* scene. In (a) and (b), we show the appearance of two fixed 3D points from two different camera positions: one on the side of the ship (orange insets) and one on the surface of the water (blue insets). Our method predicts the changing specular appearance of these two 3D points, and in (c) we show how this behavior generalizes continuously across the whole hemisphere of viewing directions.

NeRF



Ground Truth Complete Model No View Dependence No Positional Encoding

Fig. 4: Here we visualize how our full model benefits from representing view-dependent emitted radiance and from passing our input coordinates through a high-frequency positional encoding. Removing view dependence prevents the model from recreating the specular reflection on the bulldozer tread. Removing the positional encoding drastically decreases the model's ability to represent high frequency geometry and texture, resulting in an oversmoothed appearance.

LIIF

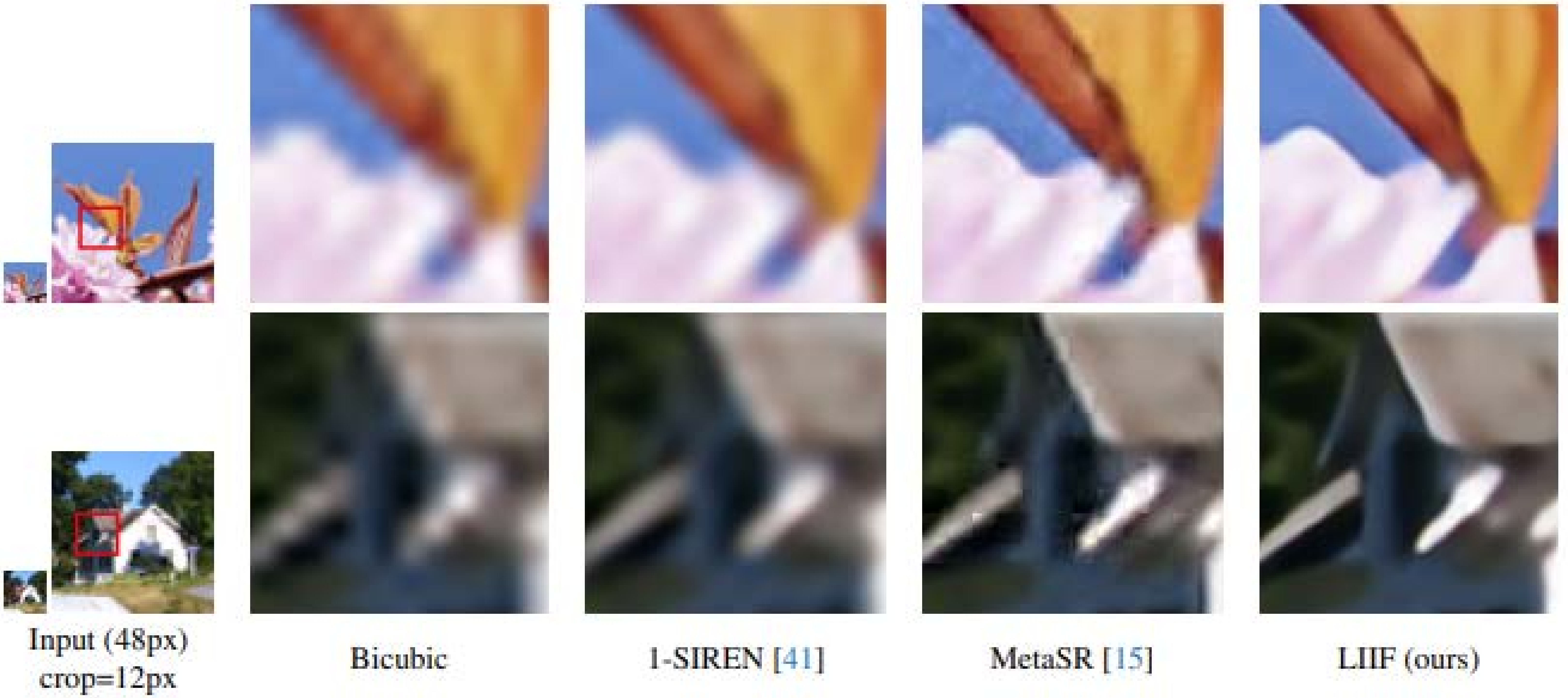


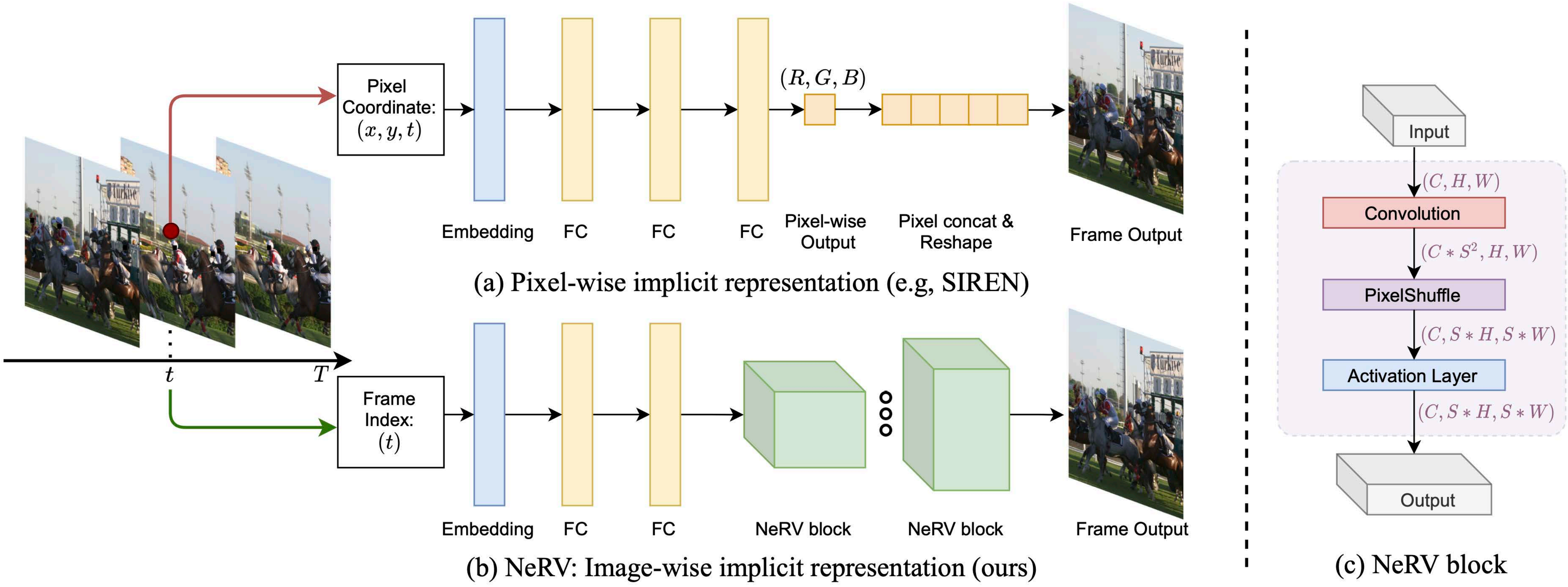
Figure 5: **Qualitative comparison of learning continuous representation.** The input is a 48 × 48 patch from images in DIV2K validation set, a red box indicates the crop area for demonstration (×30). 1-SIREN refers to fitting an independent implicit function for the input image. MetaSR and LIIF are trained for continuous random scales in ×1–×4 and tested for ×30 for evaluating the generalization to arbitrary high precision of the continuous representation.

Applications

NeRV

Neural Representations for Videos
NeurIPS 2021, Chen et al.

Want more…



(a) Pixel-wise implicit representation (e.g, SIREN)

(b) NeRV: Image-wise implicit representation (ours)

(c) NeRV block