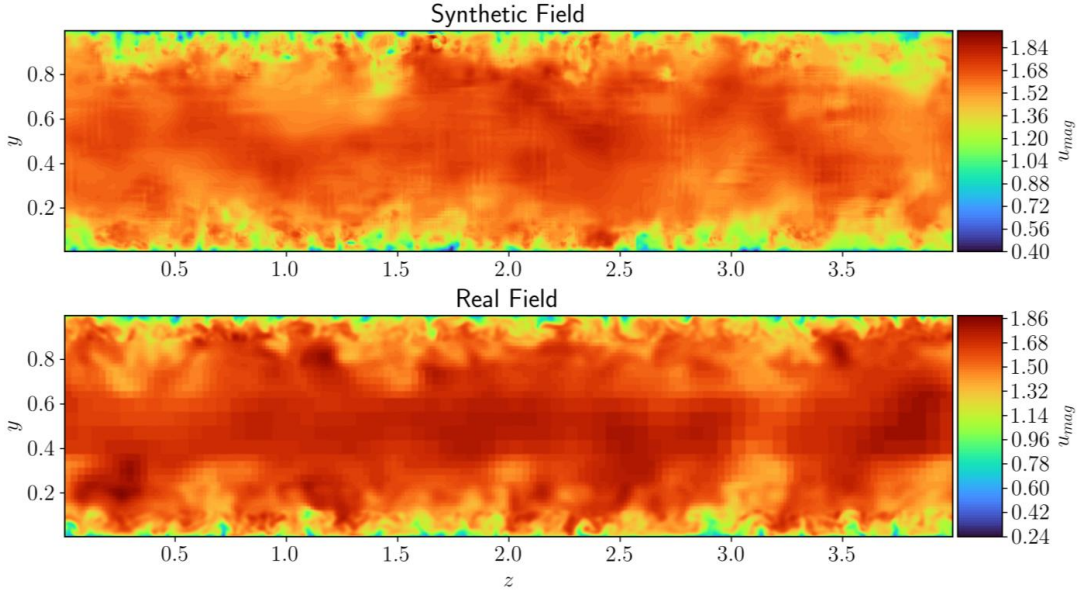
GENERATIVE MACHINE LEARNING FOR CREATION OF INITIAL CONDITIONS



PRESENTATION OUTLINE

1. Summary Points

2. Fluid Dataset

3. Machine Learning Model

4. Results

5. Recap and Assessment

6. A Deeper Diver into Details (Optional)

SUMMARY POINTS

50K FOOT VIEW

Some of the things we can do with ML:

- Super-resolution
- Surrogate models
- Simulation accelerators
- Data-driven numerical algorithms
- Turbulence modeling
- Design suggestions based on previous trends
- ⋮

Emerging trends in machine learning for computational fluid dynamics

Ricardo Vinuesa[1,2*] and Steven L. Brunton[3]

[1] FLOW, Engineering Mechanics, KTH Royal Institute of Technology, Stockholm, Sweden
[2] Swedish e-Science Research Centre (SeRC), Stockholm, Sweden
[3] Department of Mechanical Engineering, University of Washington, Seattle, WA 98195, United States

Unsupervised deep learning for super-resolution reconstruction of turbulence

Hyojin Kim[1], Junhyuk Kim[1,‡], Sungjin Won[2] and Changhoon Lee[1,2,†]

[1]Department of Mechanical Engineering, Yonsei University, Seoul 03722, Korea
[2]School of Mathematics and Computing, Yonsei University, Seoul 03722, Korea

PHYSICAL REVIEW FLUIDS **6**, 050504 (2021)

Invited Articles

Perspectives on machine learning-augmented Reynolds-averaged and large eddy simulation models of turbulence

Karthik Duraisamy

Department of Aerospace Engineering, University of Michigan, Ann Arbor, Michigan 48109, USA

Learning a reduced basis of dynamical systems using an autoencoder

David Sondak and Pavlos Protopapas
Phys. Rev. E **104**, 034202 – Published 3 September 2021

Machine learning–accelerated computational fluid dynamics

Dmitrii Kochkov[a,1,2], Jamie A. Smith[a,1,2], Ayya Alieva[a], Qing Wang[a], Michael P. Brenner[a,b,2], and Stephan Hoyer[a,2]

[a]Google Research, Mountain View, CA 94043; and [b]School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138

Today's topic

Using generative ML to seed CFD simulation

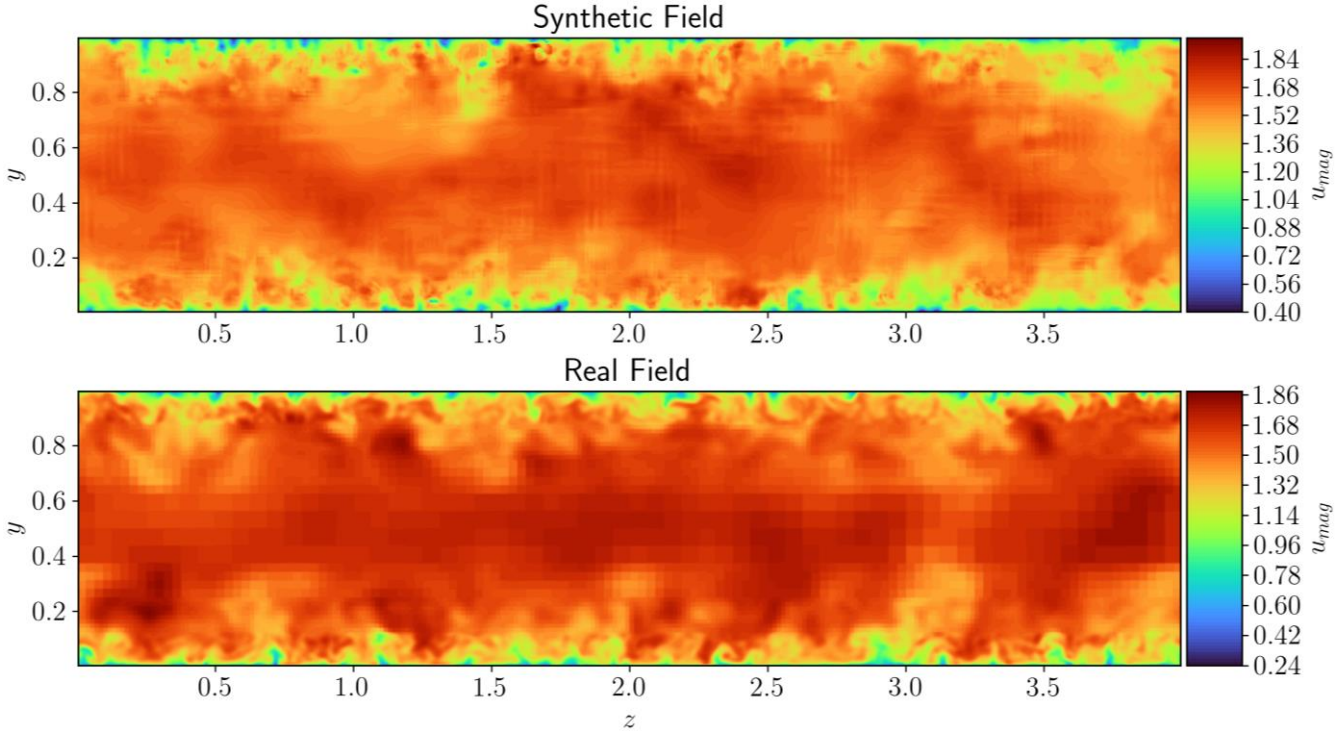
- **Seed 3D ICs to cut down on transients**
- Could also generate inflow conditions

INFINITY: Neural Field Modeling for Reynolds-Averaged Navier-Stokes Equations

Louis Serrano[1] Leon Migus[1,2] Yuan Yin[1] Jocelyn Ahmed Mazari[3] Patrick Gallinari[1,4]

SUMMARY POINTS

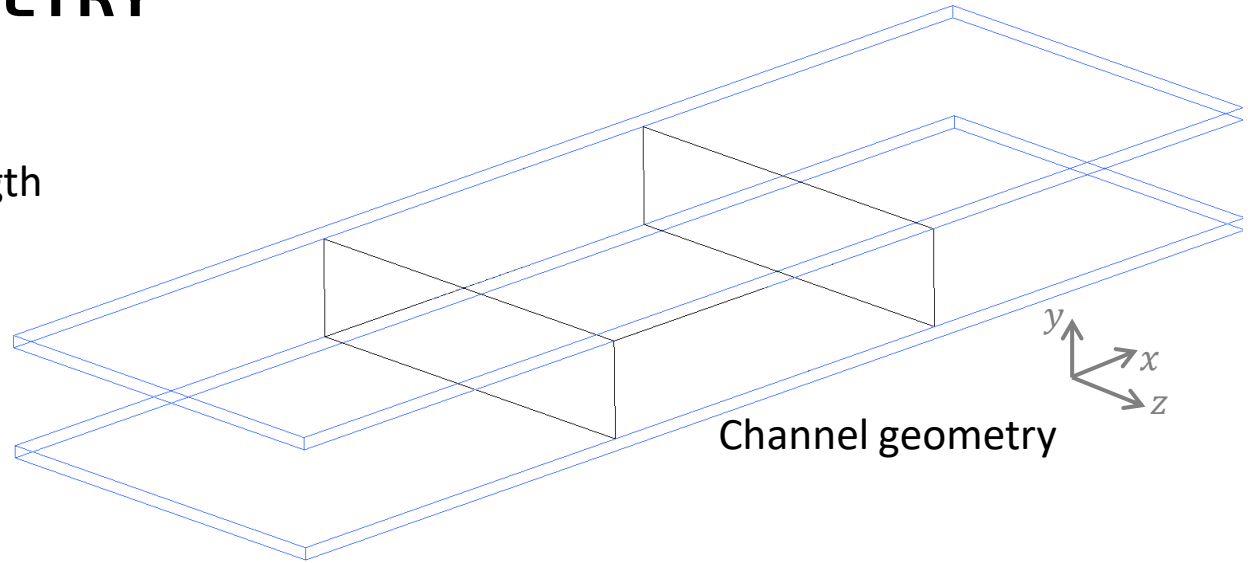
- Built generative machine learning model and accompanying machine learning pipeline
- Trained on CFD dataset
- ML-generated field used to seed CFD simulation



FLUID DATASET

CASE SETUP: GEOMETRY

- $L_{char} = 1\ [m]$
- Resolution: 128 per char. length
- $L_x \times L_y \times L_z = (12.5L_{char}, L_{char}, 4L_{char})$
- $\Delta y^+_{wall} = 35.3$

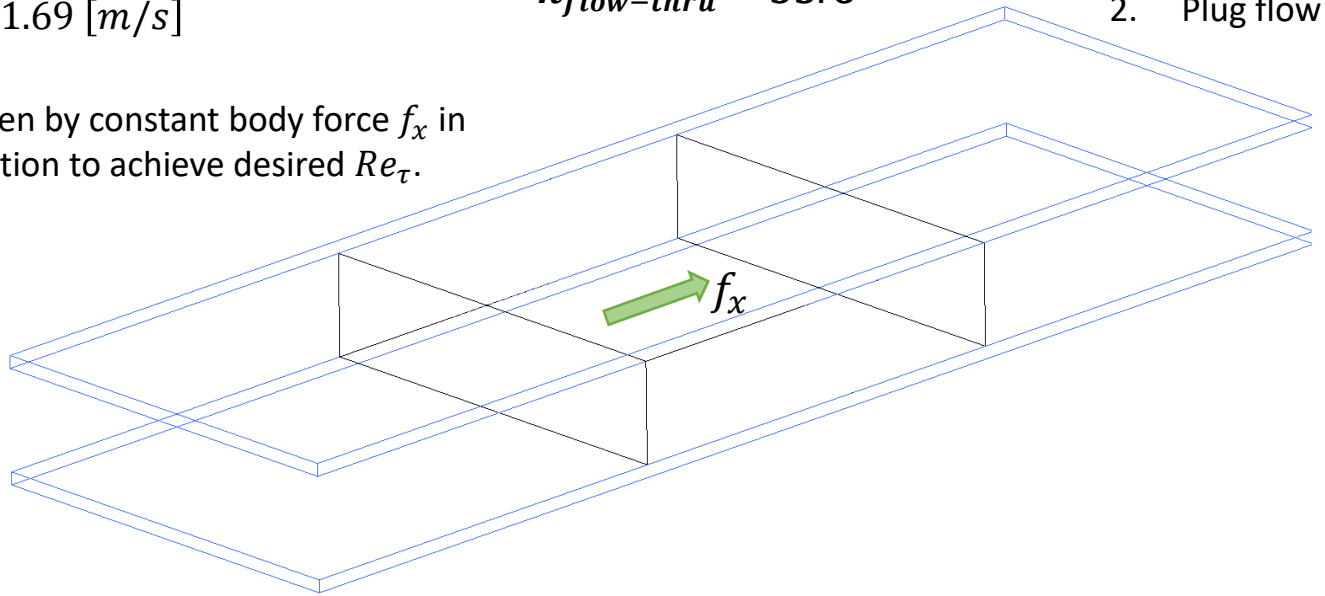


Channel geometry

CASE SETUP: PHYSICAL PARAMETERS

- Physical Parameters
  - Working fluid: air at STP
  - $Ma_{real} = 0.00494, Ma_{sim} = 0.172$
  - $\boldsymbol{Re_{bulk} = 1.13 \times 10^5}$
  - $\boldsymbol{Re_\tau = 2260}$
  - $U_{char} = 1.69\ [m/s]$
- Characteristic times
  - $\boldsymbol{\Delta t = 4.57 \times 10^{-4}\ [s]}$
  - $t_{final} = 239.6\ [s]$
  - $t_{flow-thru} = \frac{L_x}{U_{char}} \approx 7.37\ [s]$
  - $\boldsymbol{n_{flow-thru} \approx 35.6}$
- Boundary / Initial Conditions
  - Periodic in x and z
  - No-slip on walls
  - Initial conditions
    1. Sine / Cosine rolls
    2. Plug flow

Flow driven by constant body force f_x in $x-$direction to achieve desired Re_τ.

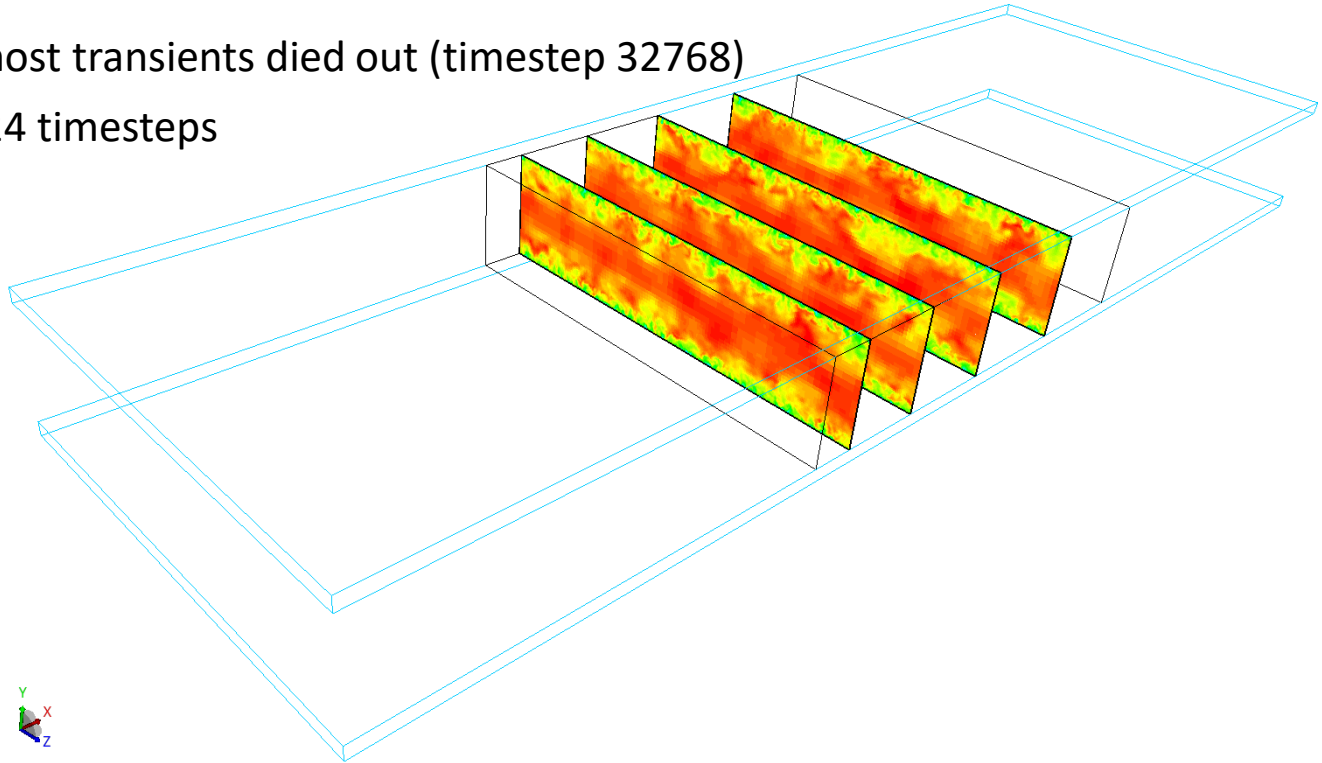


CASE SIZE AND RUN INFO

Cells	27961600
Timesteps	$\geq$524288
Hardware	Xeon E5-2680V4 2.4 GHz (Broadwell)
Core count	252
Run duration	85.8 hours, 3.57 days
CPU Hours	2.161×10^4
Data size (GB) (all fluid measurement windows)	160

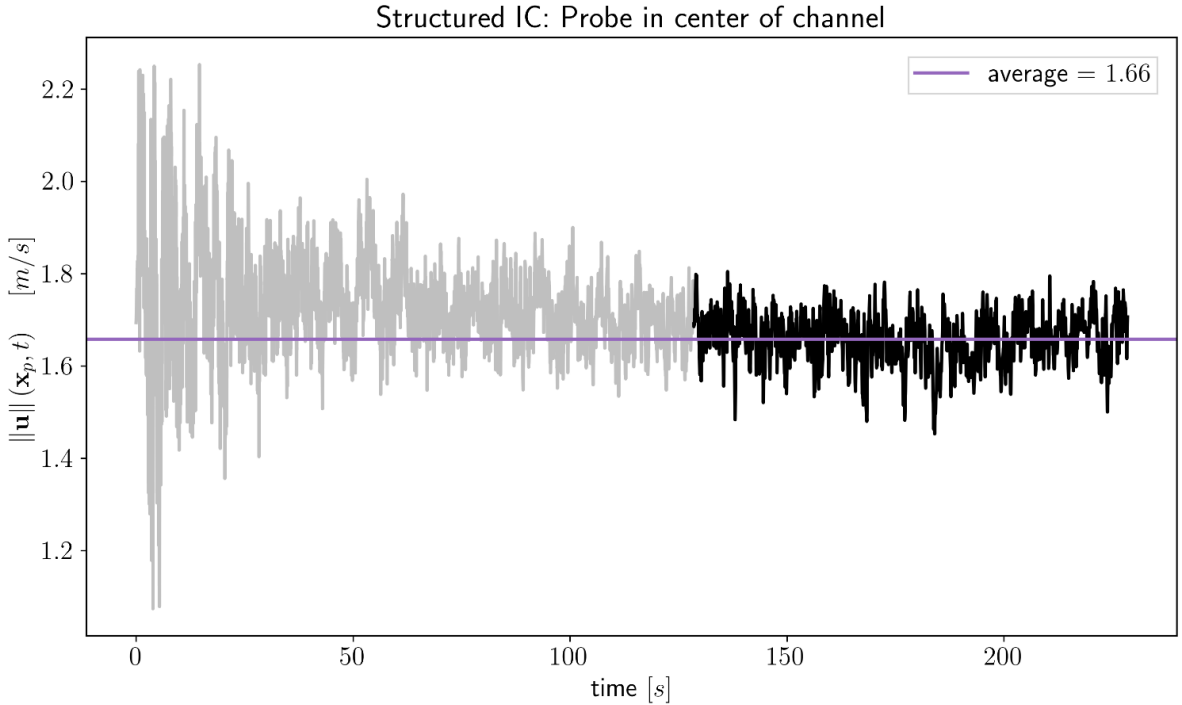
DATASET GENERATION AND MEASUREMENT INFO

- Dataset consists of $x-$ aligned slices at different downstream locations and times
- Started measuring after most transients died out (timestep 32768)
- Measurement period: 1024 timesteps
  - ≈ 0.06 flow-thru times



TRANSIENT DIAGNOSTICS

- Time to reach steady state: $t_{ss,0} \approx 128\ [s] \approx 281272$ timesteps
- Time in statistically steady state: $T_{ss} \approx 100\ [s] \approx 13$ flow-thru times



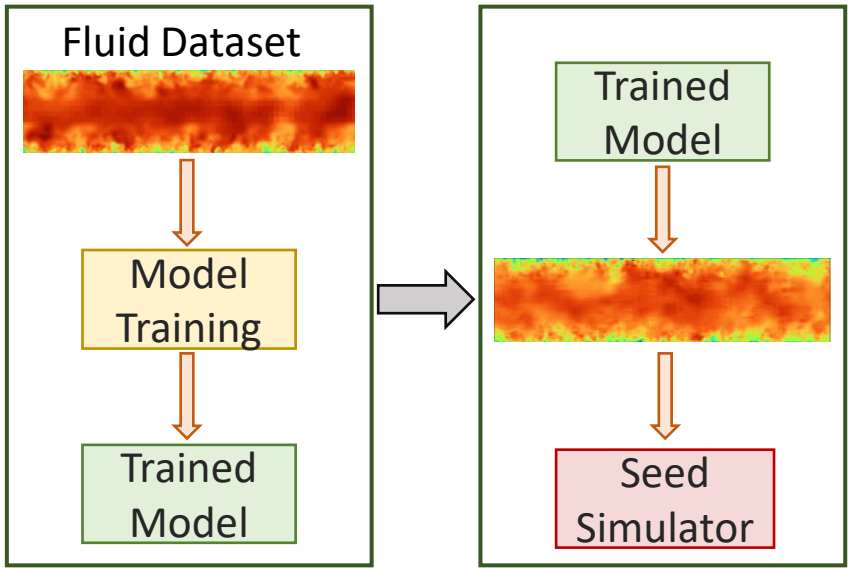
- Velocity magnitude at center of channel
  - Gray: Transients
  - Black: statistically steady state

MACHINE LEARNING MODEL

GENERATIVE MODELS FOR SEEDING INITIAL CONDITIONS

Basic Goal(s)

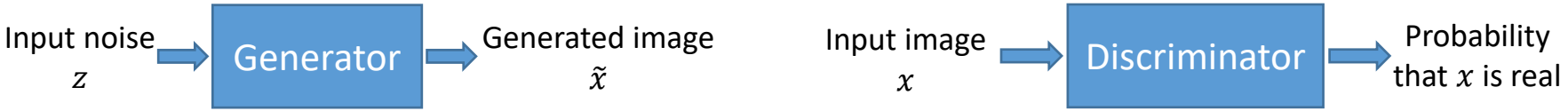
- Generate realistic, synthetic initial conditions from existing data
- Seed a simulation with synthetic initial conditions
- Does the ML-seeded run converge faster than a simulation started from scratch?



- Generative models are a natural fit for this aim
- Cutting edge generative models
  - **Generative Adversarial Networks (GANs)**
  - **Diffusion models***
  - Variational autoencoders
  - Bayesian networks
  - Boltzmann Machines
  - ⋮
- This work: Wasserstein GAN w/ Gradient penalty

A LITTLE BACKGROUND ON GANS FOR PERSPECTIVE

- GANs took the world by storm back in 2014.
  - The basic idea: generator generates a sample and a discriminator discriminates whether or not it's real.



  - Adversarial training: The generator is trying to fool the discriminator – it's a two-player game.
- GANs are able to generate incredibly realistic images.
- GANs are an *unsupervised* machine learning method
  - They don't require labeled data – just a dataset

Karras et al (2018)

- They have become one of the dominant generative models, but have some shortcomings:
  - The adversarial training can be unstable and hard to get just right without substantial tuning
  - There is a vanishing gradient problem, which leads to stagnating training
  - Mode collapse – generator predicts a single image that it knows can always full the discriminator.
- Some of these problems are addressed by Wasserstein GAN with Gradient Penalty (WGAN-GP)

ELEMENTS OF A MACHINE LEARNING ALGORITHM

Model

This is the overarching modeling paradigm used for getting the desired result. In our case, it is a GAN. Another option would be a diffusion model.

Optimization Algorithm

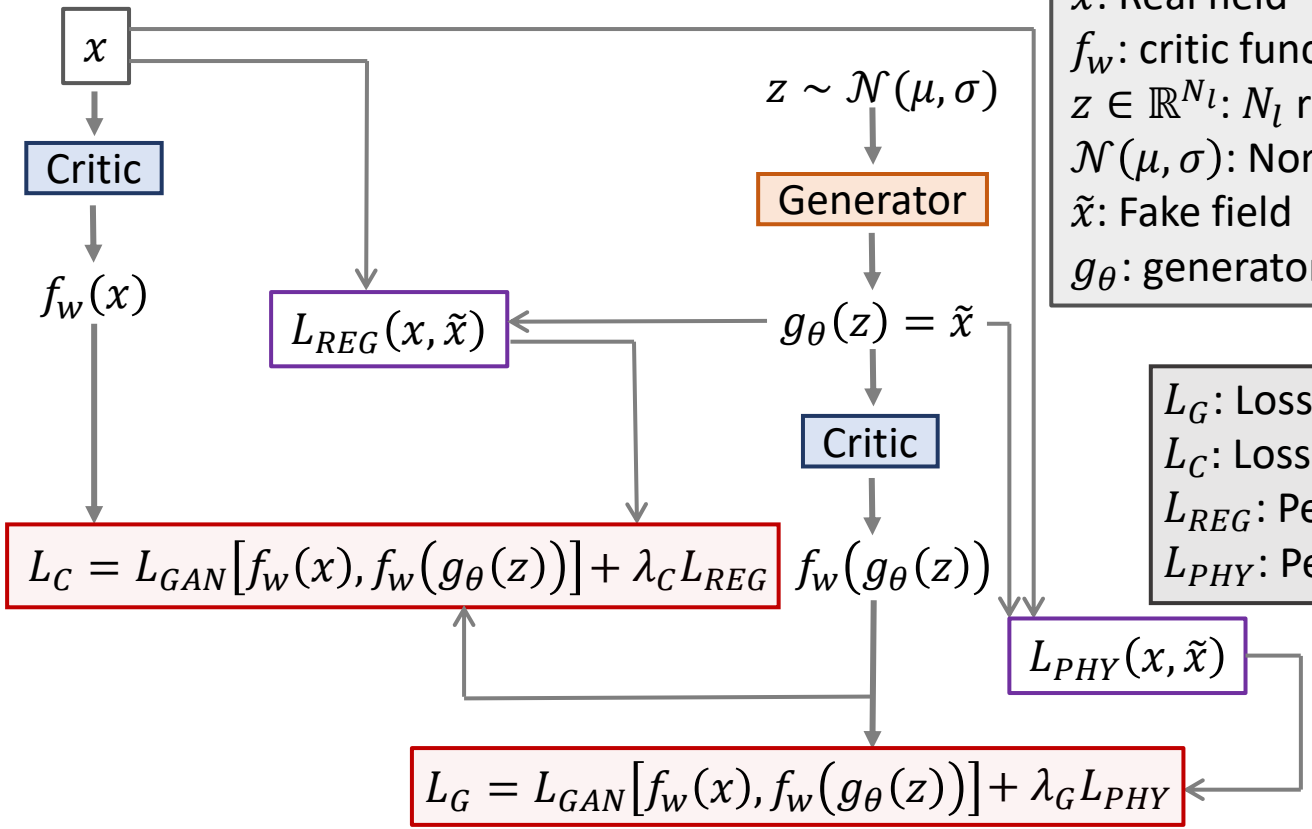
This is the way that the model parameters are updated. It plays a critical role in determining how well the model works.

Architectural Backbone

When working with neural networks, need to choose which architecture to use. These can include combinations of encoders and decoders, U-nets, transformers, and a variety of layers including fully-connected layers and convolutional layers.

WASSERSTEIN GENERATIVE ADVERSARIAL NETWORKS

The Very Big Picture



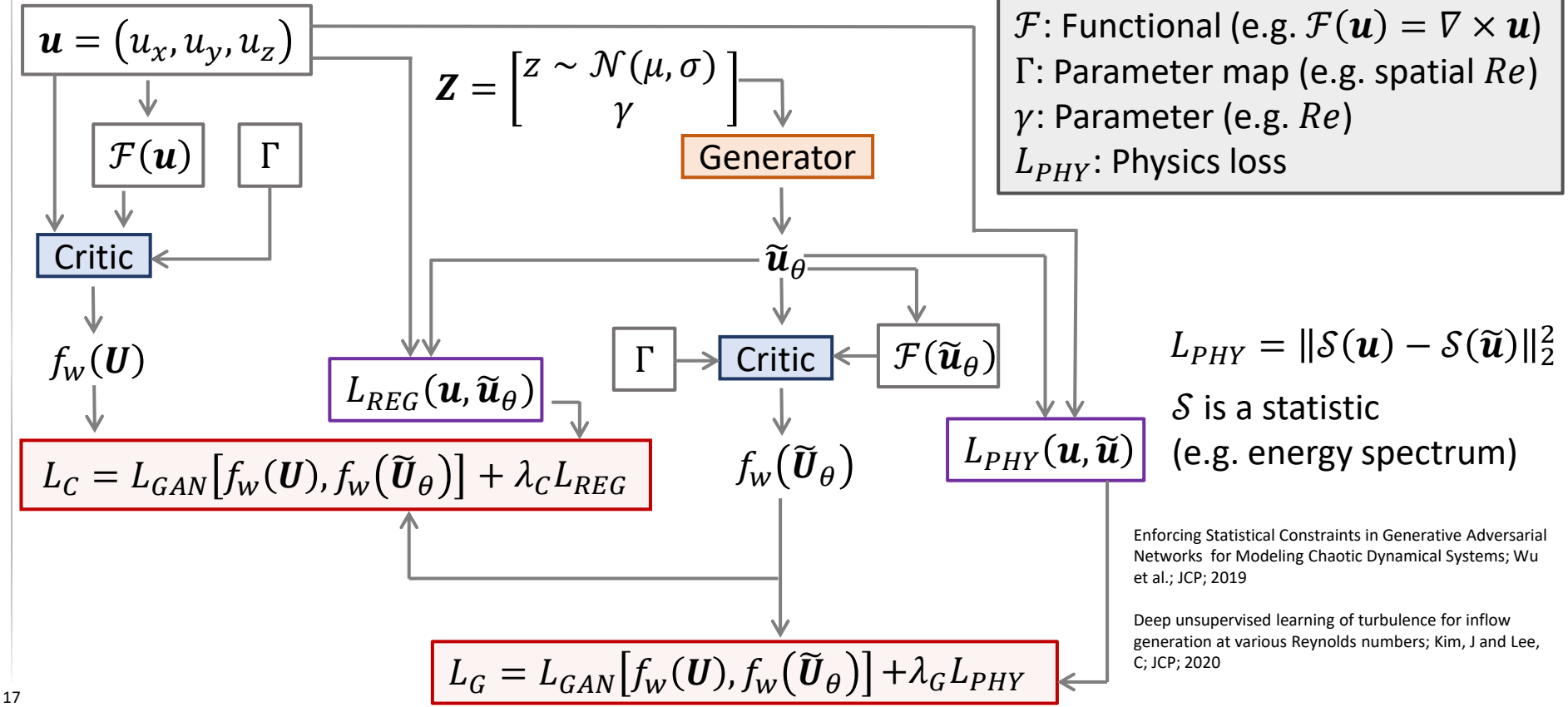
x: Real field
f_w: critic function with parameters w
$z \in \mathbb{R}^{N_l}$: N_l random numbers
$\mathcal{N}(\mu, \sigma)$: Normal w/ mean μ and std σ
$\tilde{x}$: Fake field
g_θ: generator function with parameters θ

L_G: Loss function for generator
L_C: Loss function for critic output
L_{REG}: Penalty for math constraints
L_{PHY}: Penalty for physics constraints

<u>Goal</u>
Find w to maximize L_C.
Find θ to minimize L_G.

WASSERSTEIN GENERATIVE ADVERSARIAL NETWORKS

Specialization to Fluid Physics



$\boldsymbol{U}$: Vector of inputs
$\mathcal{F}$: Functional (e.g. $\mathcal{F}(\boldsymbol{u}) = \nabla \times \boldsymbol{u}$)
Γ: Parameter map (e.g. spatial Re)
γ: Parameter (e.g. Re)
L_{PHY}: Physics loss

$$L_{PHY} = \|\mathcal{S}(\boldsymbol{u}) - \mathcal{S}(\widetilde{\boldsymbol{u}})\|_2^2$$

$\mathcal{S}$ is a statistic
(e.g. energy spectrum)

Enforcing Statistical Constraints in Generative Adversarial Networks for Modeling Chaotic Dynamical Systems; Wu et al.; JCP; 2019

Deep unsupervised learning of turbulence for inflow generation at various Reynolds numbers; Kim, J and Lee, C; JCP; 2020

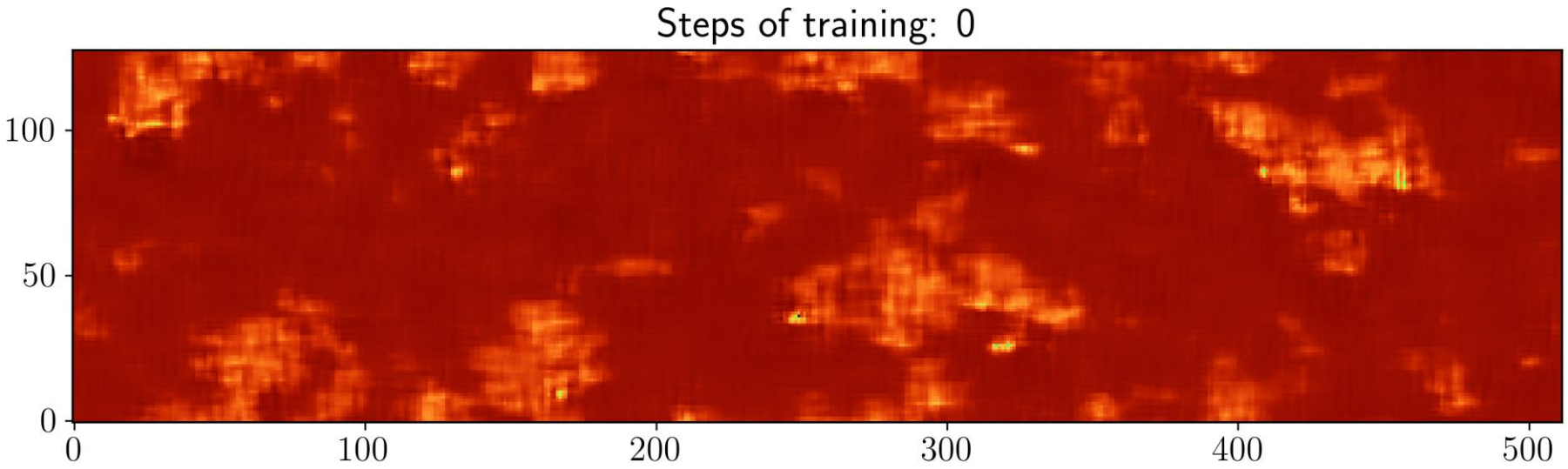
TRAINING DETAILS FOR DIFFERENT CASES

	Case 1	Case 2	Case 3
Dataset size available	$(32708 \times 3 \times 128 \times 512)$ 25 GB		
Dataset size	$(1637, 3, 128, 512)$	$(4089, 3, 128, 128)$	$(4080, 3, 128, 128)$
Dataset size (GB)	1.3	0.8	0.8
Physics constraint	No	No	Yes
# of generator parameters	1.1×10^6	6.8×10^5	6.8×10^5
Generator size (MB)	4.4	2.7	2.7
# of critic parameters	3.1×10^5	7.1×10^5	7.1×10^5
Critic size (MB)	1.2	2.8	2.8
Latent space dimension	128		
Training iterations	1000	5000	5000
Time per iteration	3.5 minutes	48 seconds	48 seconds
Total time	$\approx$ 2.4 days	$\approx$ 2.7 days	$\approx$ 2.7 days
Hardware	NVIDIA Tesla V100 32 GB RAM		NVIDIA A100 40 GB RAM

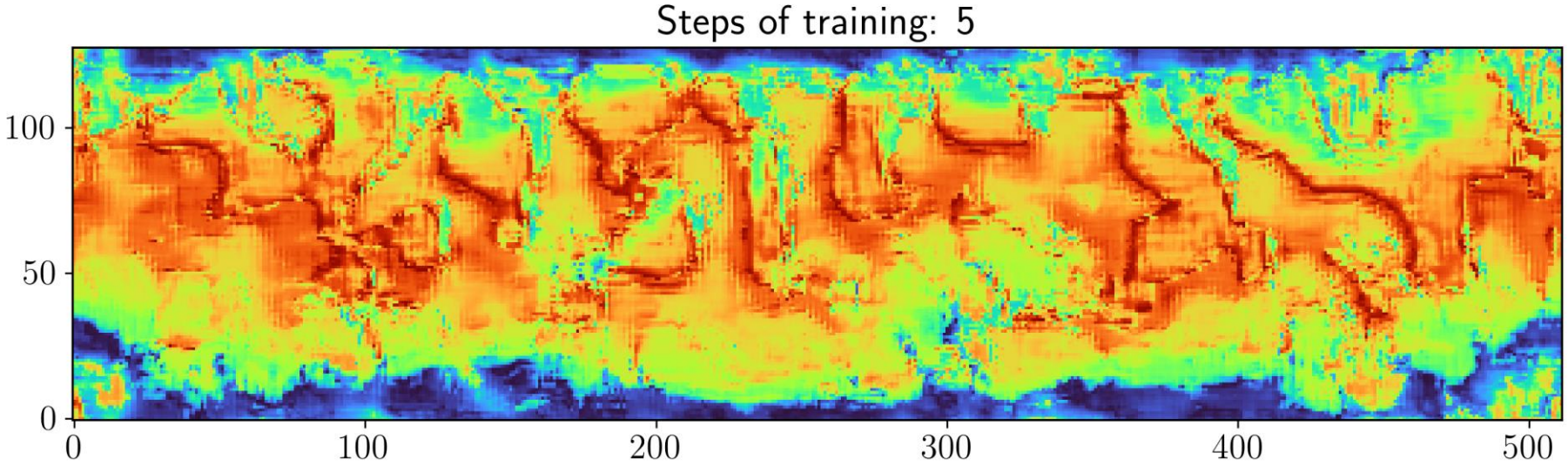
All cases use a type of convolutional network as the architectural backbone for the generator and critic.

RESULTS

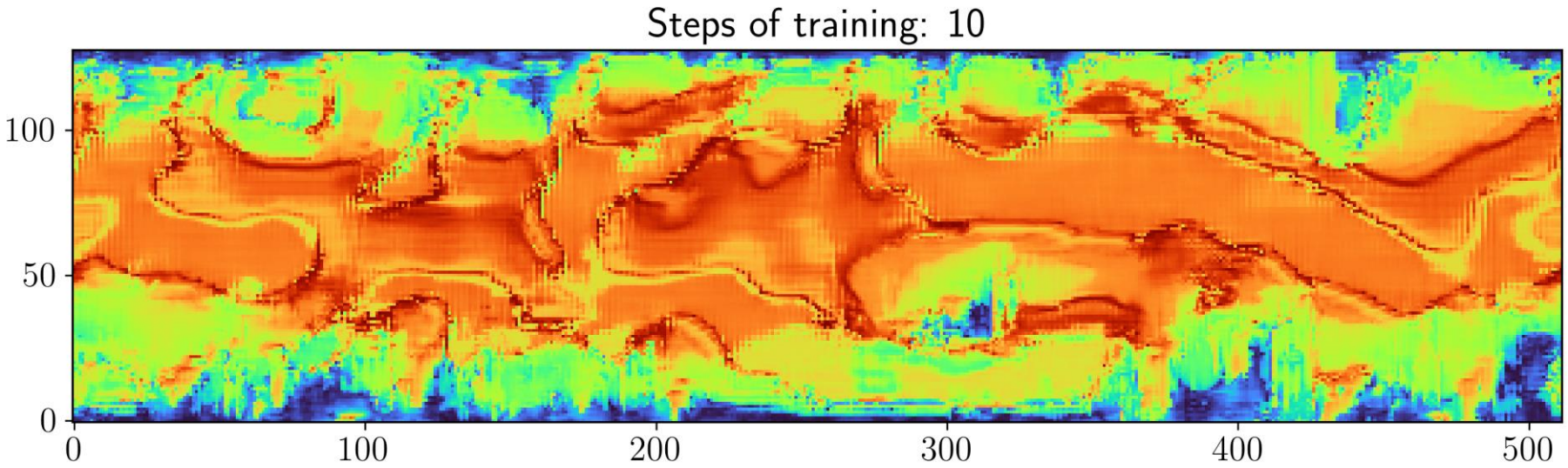
EVOLUTION OF THE GENERATOR



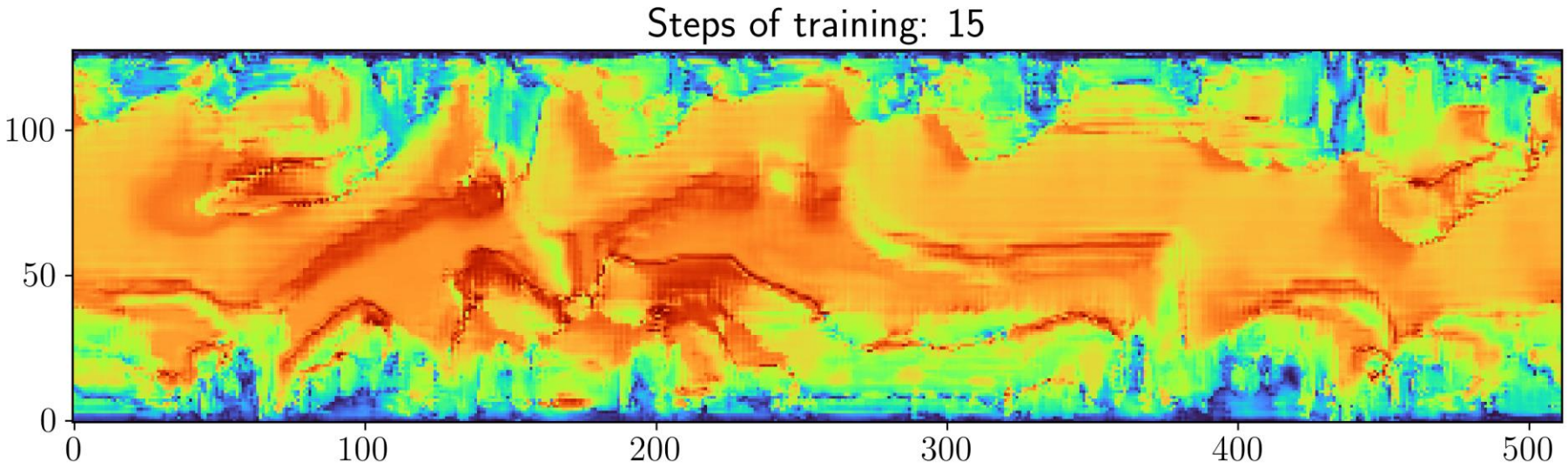
EVOLUTION OF THE GENERATOR



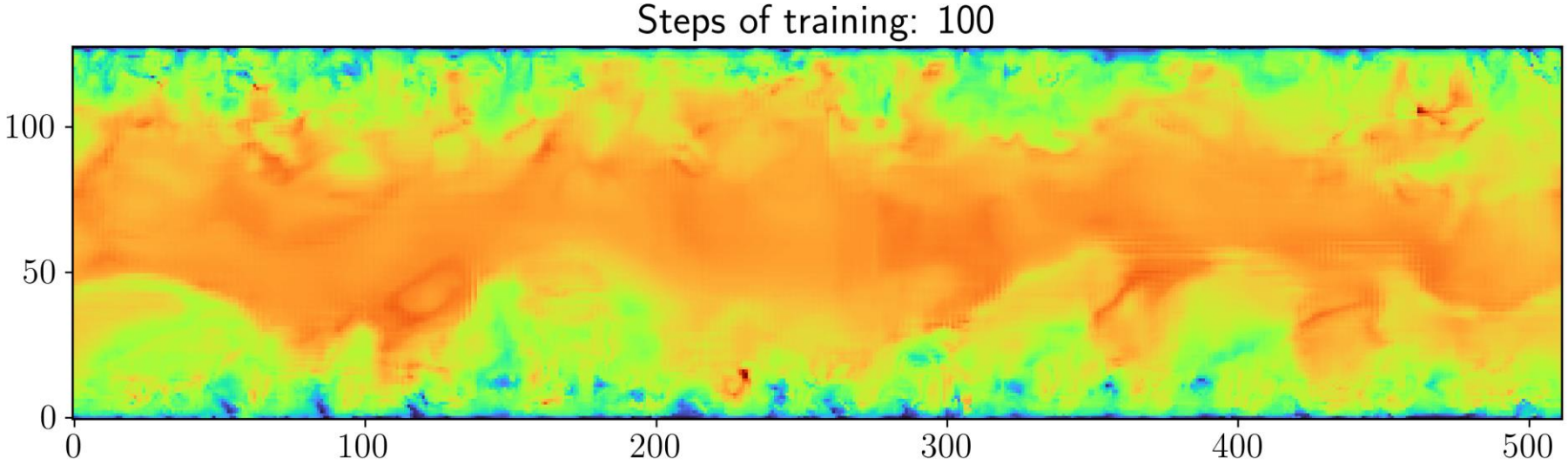
EVOLUTION OF THE GENERATOR



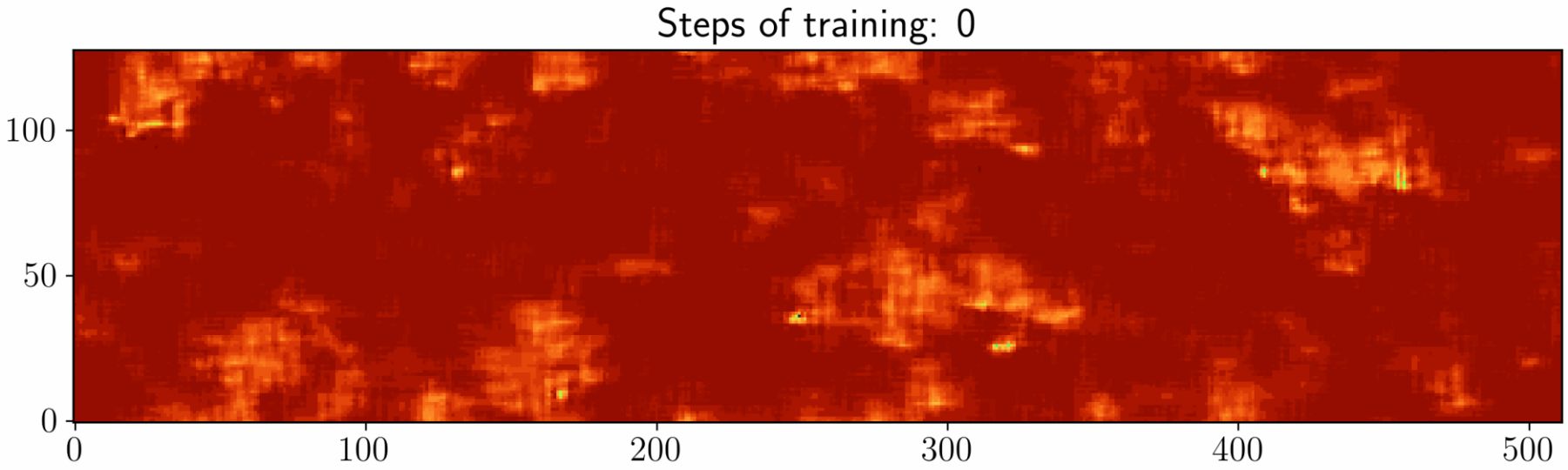
EVOLUTION OF THE GENERATOR



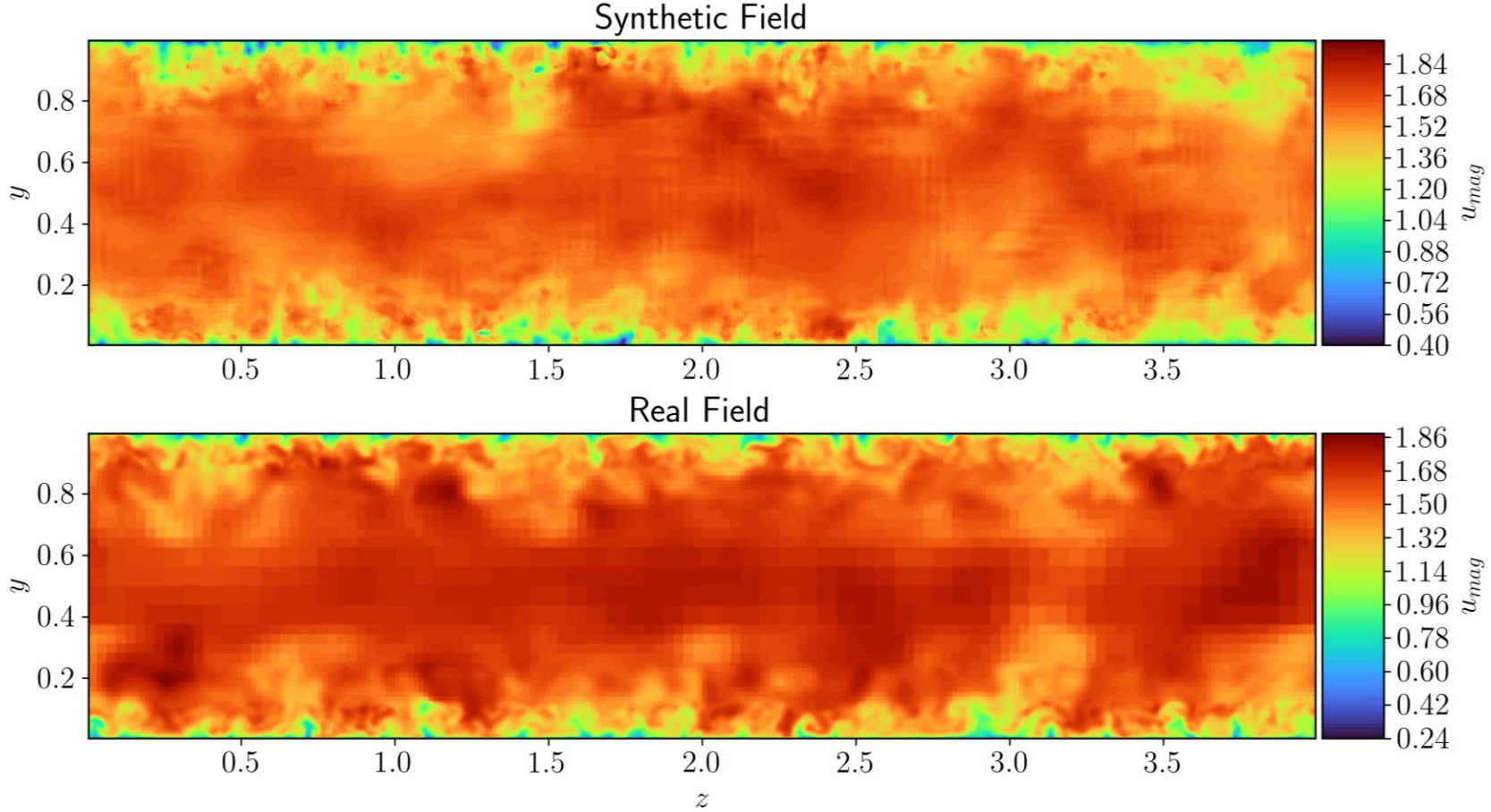
EVOLUTION OF THE GENERATOR



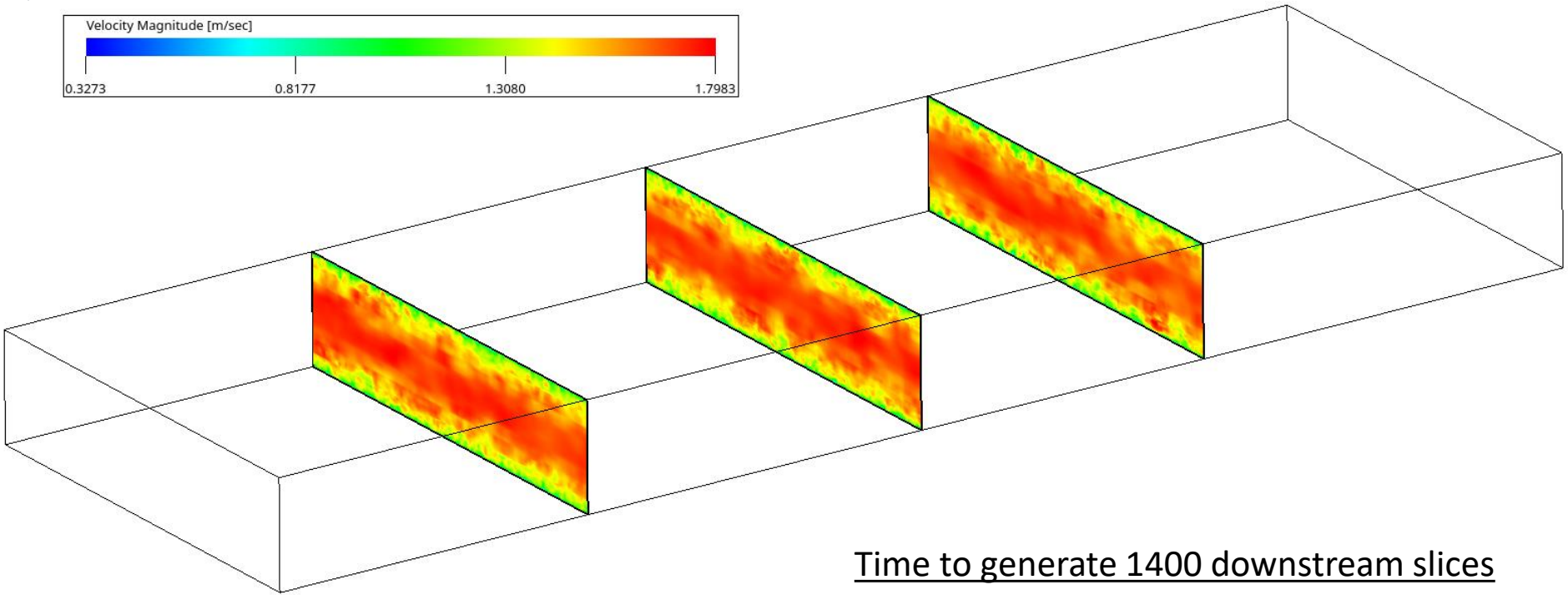
EVOLUTION OF THE GENERATOR



EXAMPLE OF A SYNTHETIC AND REAL FIELD



SEEDED FIELDS

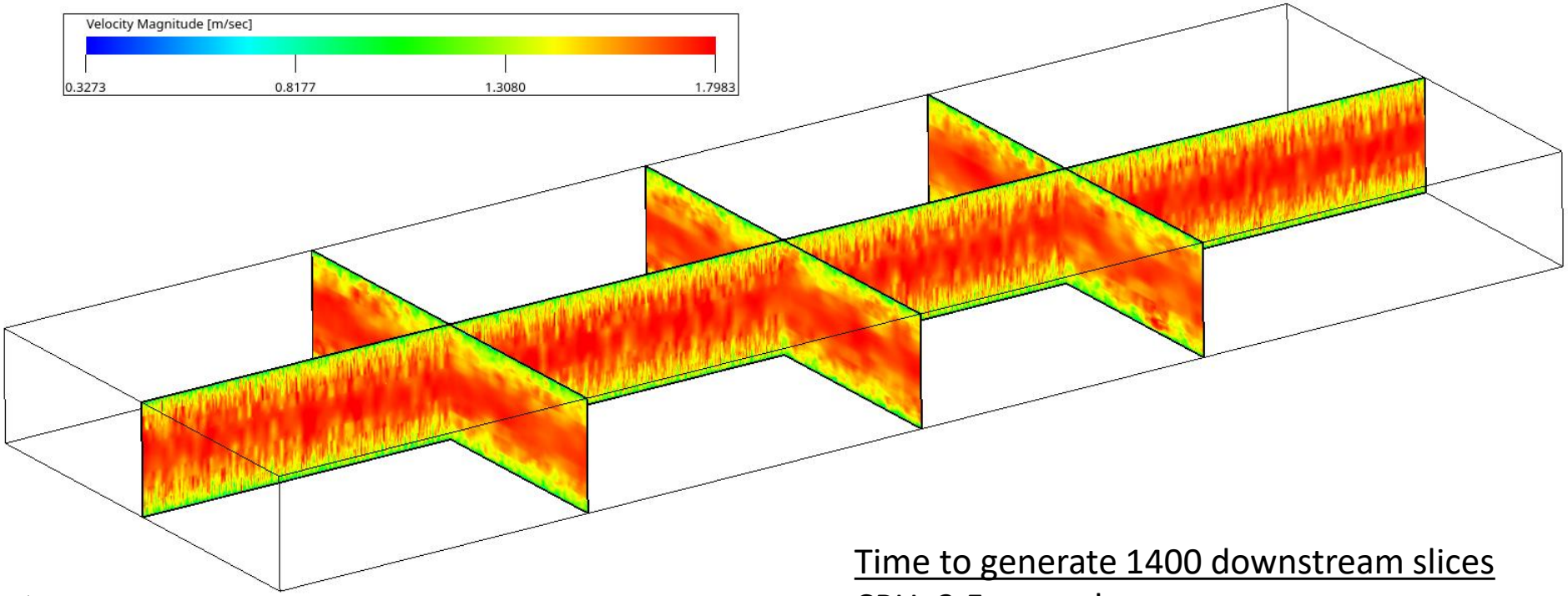


Time to generate 1400 downstream slices

CPU: 3.5 seconds

GPU (A100): 0.52 seconds

SEEDED FIELDS

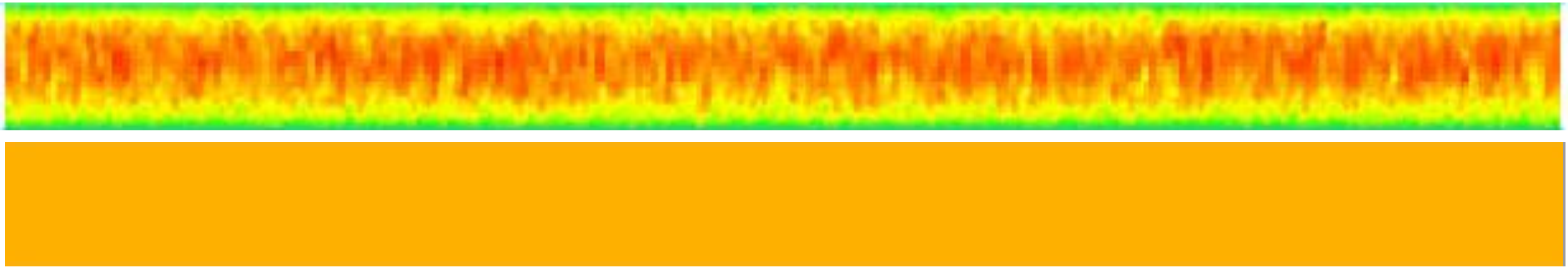


Time to generate 1400 downstream slices

CPU: 3.5 seconds

GPU (A100): 0.52 seconds

STARTUP OF SYNTHETIC SEEDING

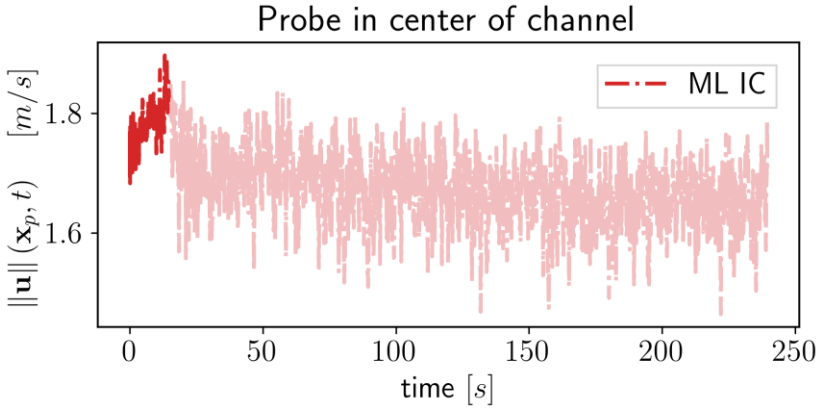


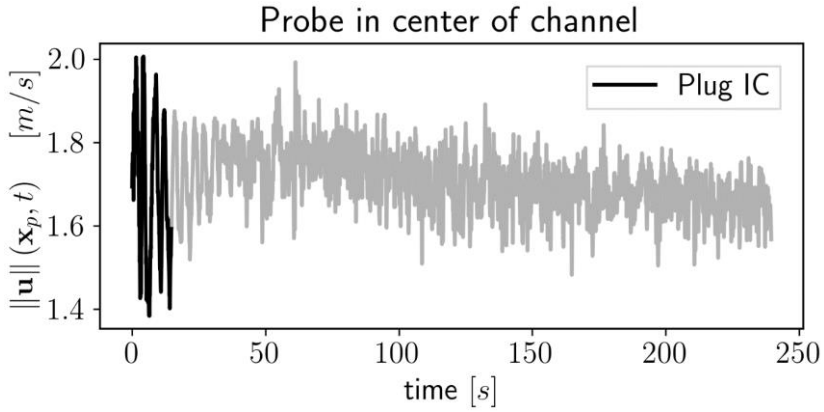
Startup of synthetic machine learning ICs

Video: Top Time trace: Bottom left

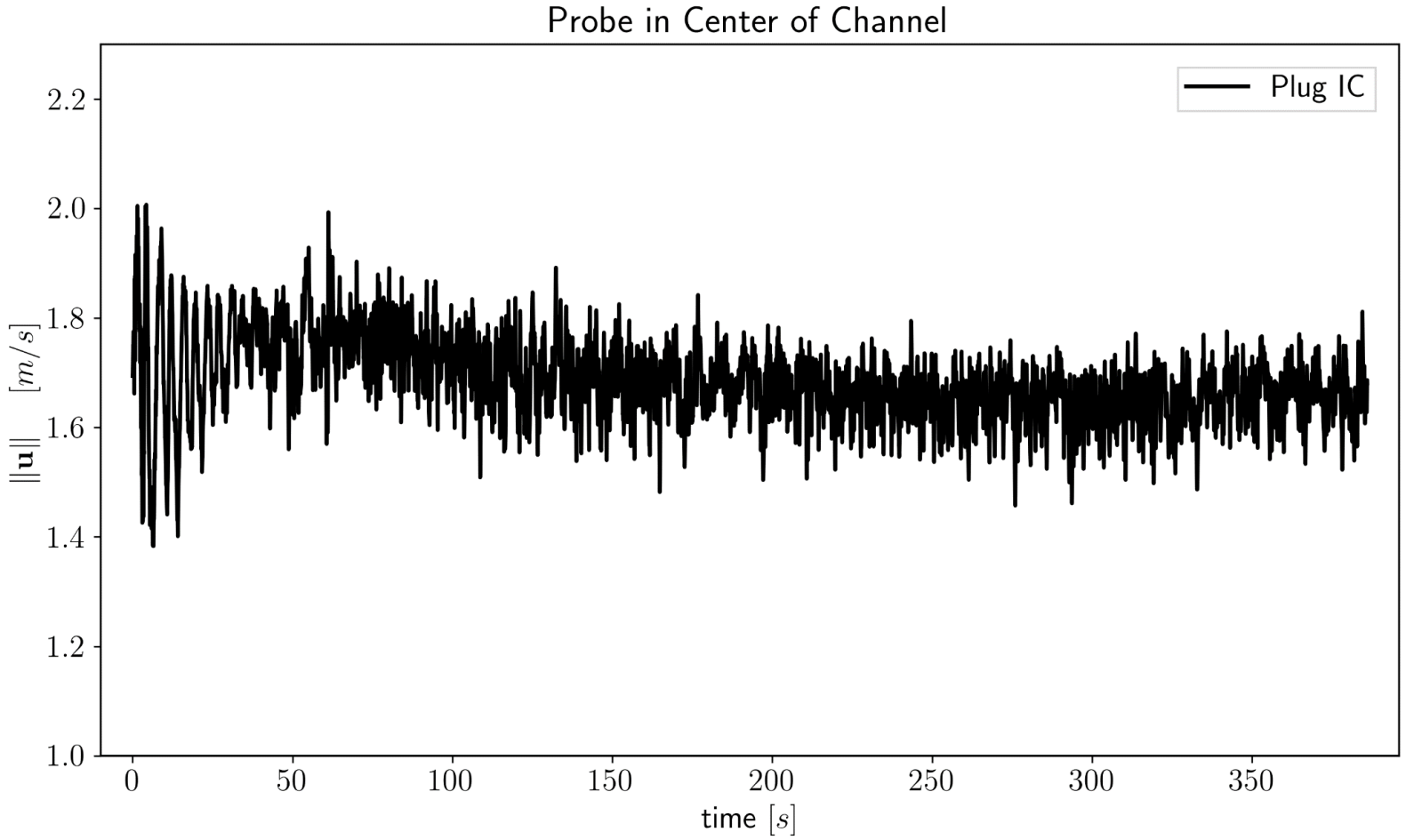
Startup of plug ICs

Video: Bottom Time trace: Bottom Right

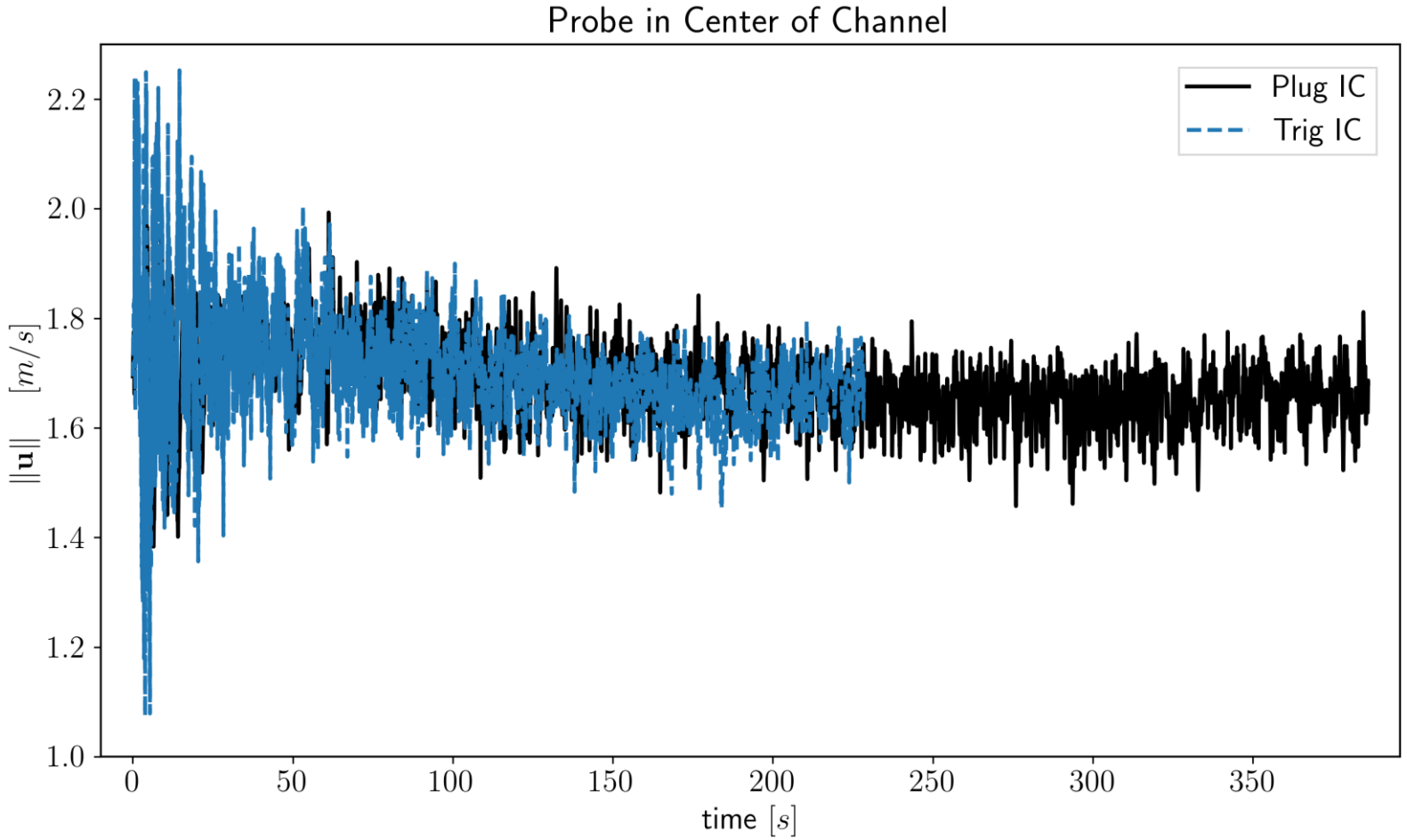




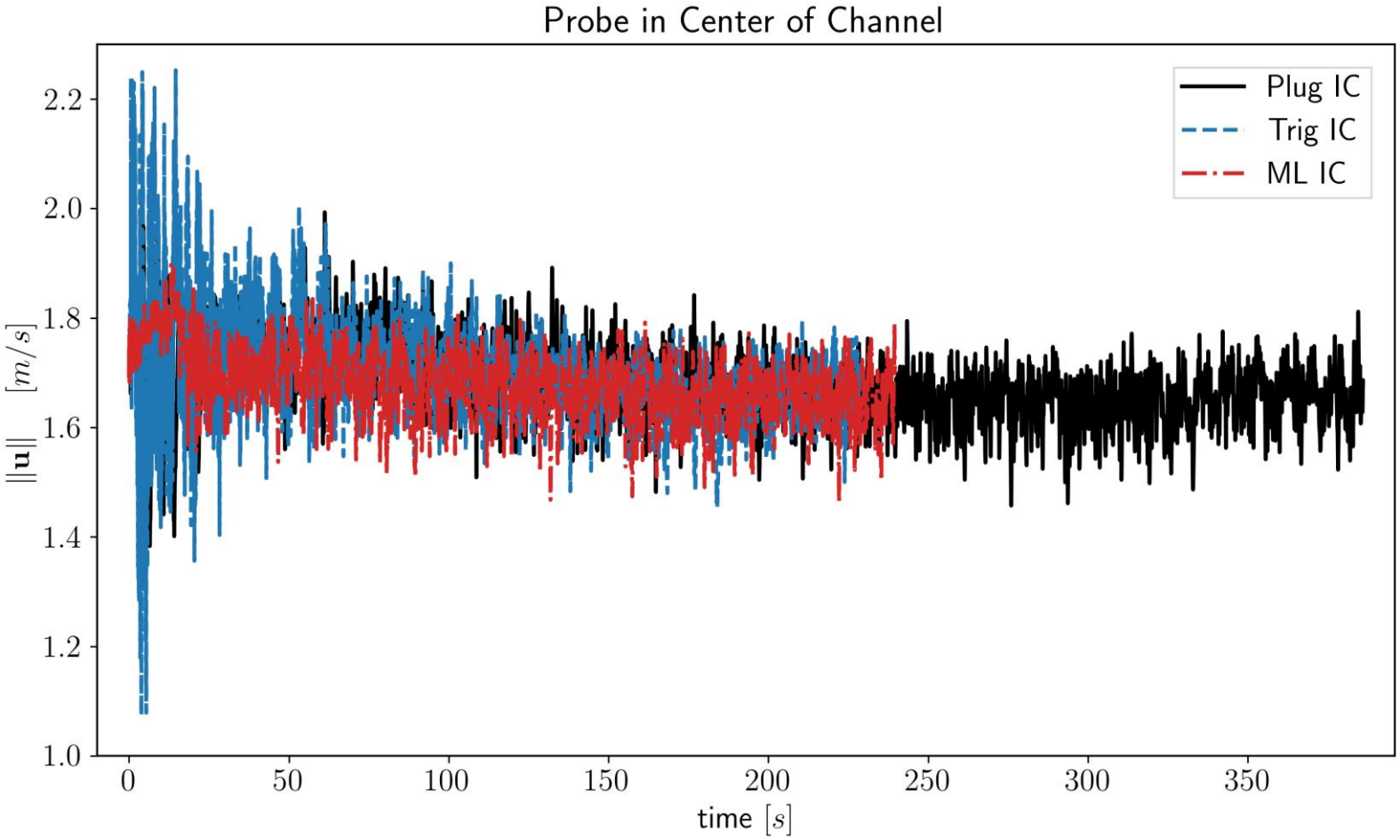
PROBE EVOLUTION TOWARDS STEADY STATE



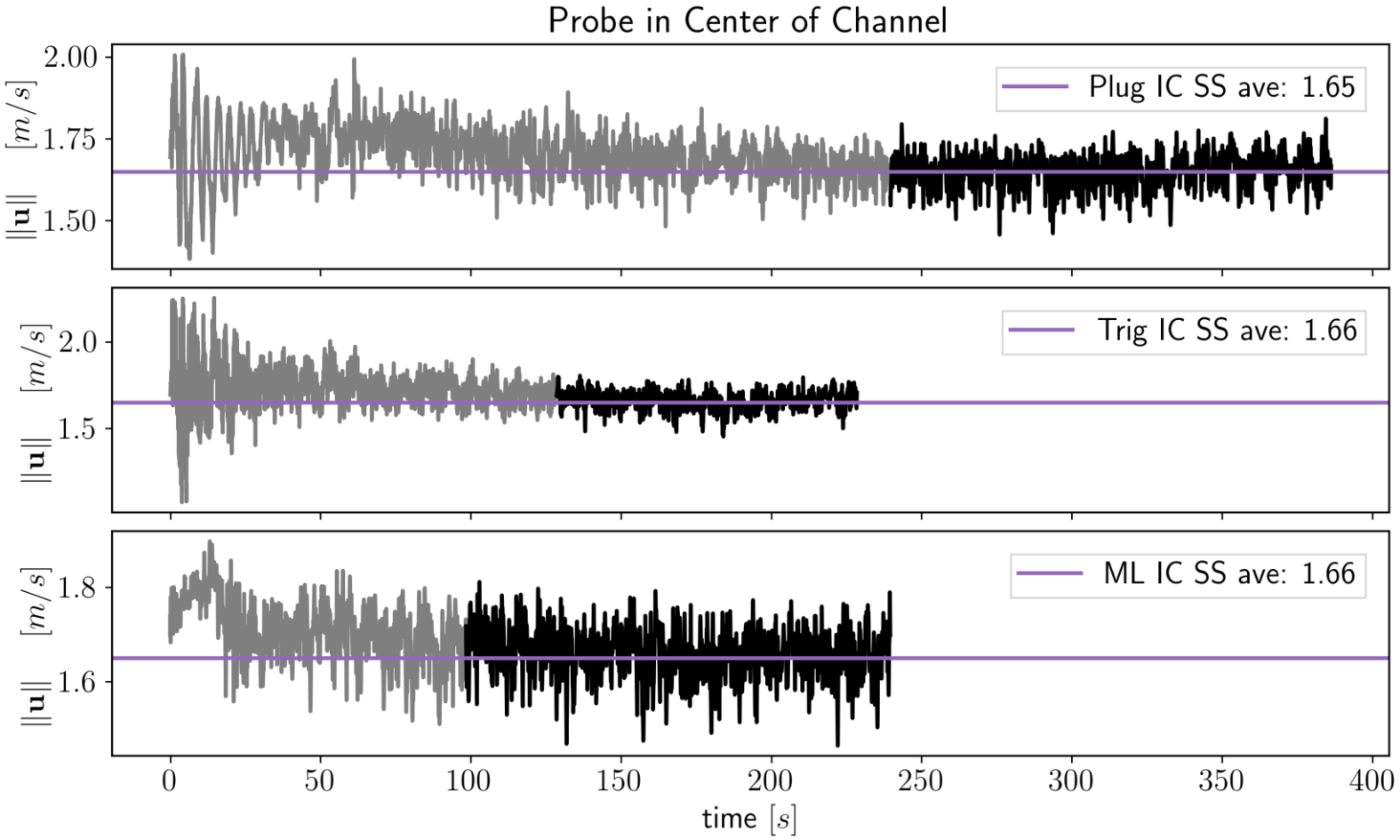
PROBE EVOLUTION TOWARDS STEADY STATE



PROBE EVOLUTION TOWARDS STEADY STATE



PROBE EVOLUTION TOWARDS STEADY STATE



SUMMARY OF RUNTIME REDUCTION

Comparison on steps to steady state and CPU-hours to steady state between different runs

Initial Condition	Steps to Steady State	CPU Hours to Steady State
Plug	524288*	2.03×10^4
Sine/Cosine	281272*	1.2×10^4
ML (Case 2)	215316	9.82×10^3

Reduction of compute-time between ML-seeded run and other ICs

	Steps to Steady State Reduction	CPU Hour Reduction
Sine / Cosine	23%	17%
Plug	59%	52%

- Substantial reduction in reaching statistically steady state when using synthetic ML ICs
  - This is true for the “easy” plug IC and the “human-engineered” trig IC
  - Could potentially enable more high-fidelity runs in design-relevant regimes
- Comments:
  - The steps to steady state for the trig ICs is an estimate

A DEEPER DIVE INTO DETAILS

ELEMENTS OF A MACHINE LEARNING ALGORITHM

Model

This is the overarching model used for getting the desired result. In our case, it is a GAN. Another option would be a diffusion model.

Optimization Algorithm

This is the way that the model parameters are updated. It plays a critical role in determining how well the model works.

Architectural Backbone

When working with neural networks, need to choose which architecture to use. These can include combinations of encoders and decoders, U-nets, transformers, and a variety of layers including fully-connected layers and convolutional layers. This design step includes significant effort.

OPTIMIZATION: GRADIENT DESCENT

A Primer on Gradient Descent

Given a dataset $\{(x, y)_i\}_{i=1}^N$, fit a model

$$\hat{y} = f(x; \theta)$$

to the data.

Choose metric to measure error between model predictions and observations $L = L(y, \hat{y})$: e.g.

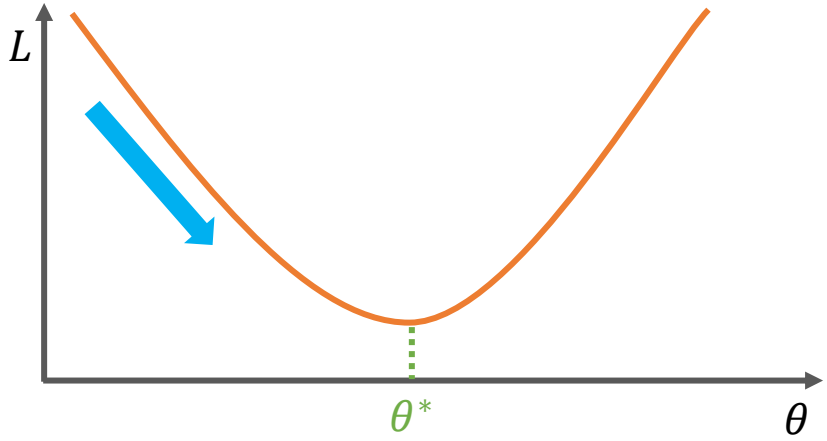
$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 .$$

Update the parameters based on gradient of L

$$\theta_n = \theta_{n-1} - \eta \nabla_\theta L.$$

Gradient descent w/ step size η will converge to local minimum.

Ok for convex functions but not for non-convex ones.




```
1  for n = 1, N_iters
2      for i = 1, N
3          Get {(x_i, y_i)}
4          ŷ_i = f(x_i)
5          ℓ += (y_i − ŷ_i)^2
6      end
7      θ_n = θ_{n−1} − η∇_θ (ℓ/N)
8  end
```

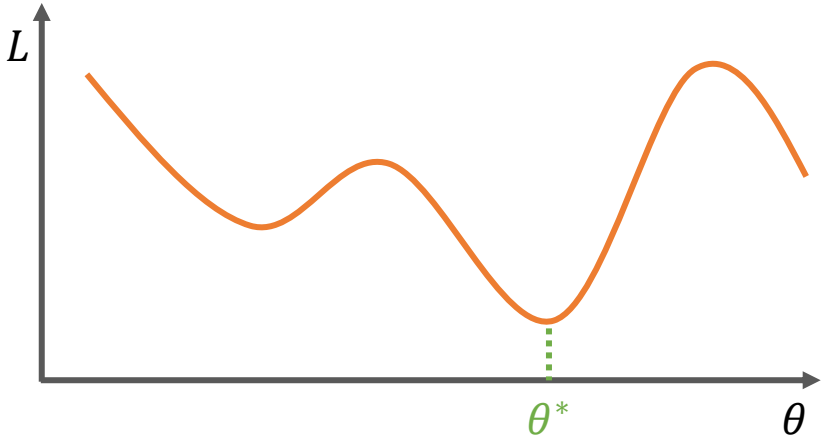
OPTIMIZATION STOCHASTIC GRADIENT DESCENT

A Primer on Stochastic Gradient Descent

Everything is the same as before ***except:***

- Now randomly sample a *single* point and immediately update the model parameters.
- Repeat this until entire dataset is visited at each iteration.

```
1  for n = 1, N_iters
2      for i = 1, N
3          Randomly select {(x_i, y_i)}
4          ŷ_i = f(x_i)
5          ℓ = (y_i − ŷ_i)^2
6          θ_n = θ_{n−1} − η∇_θ (ℓ/N)
7      end
8  end
```



Main benefit today: one of the few effective ways to not get caught in a local minimum during non-convex optimization.

OPTIMIZATION: STOCHASTIC GRADIENT DESCENT PART 2

A Primer on Stochastic Gradient Descent with “mini-batching”

In between pure gradient descent and pure stochastic gradient descent.

Randomly select a “batch” of data and iterate over the batches until dataset exhausted.

Still stochastic, but faster than pure stochastic gradient descent.

```
for n = 1, N_iters
    for i = 1, N_batches
        Randomly select {(x_j, y_j)}_{j=1}^{M}
        for j=1, M
            ŷ_j = f(x_j)
            ℓ += (y_j − ŷ_j)^2
        end
        θ_n = θ_{n−1} − η∇_θ (ℓ / M)
end
```

Terminology:

Epoch: a training iteration

Batch size (M): The number of datapoints per batch

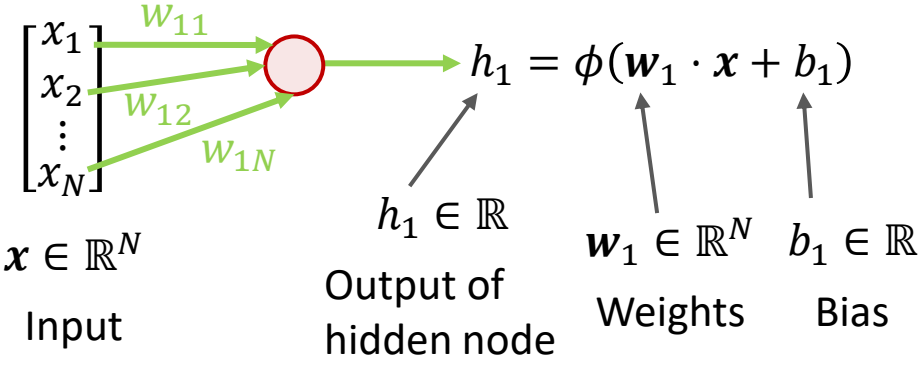
The number of batches is

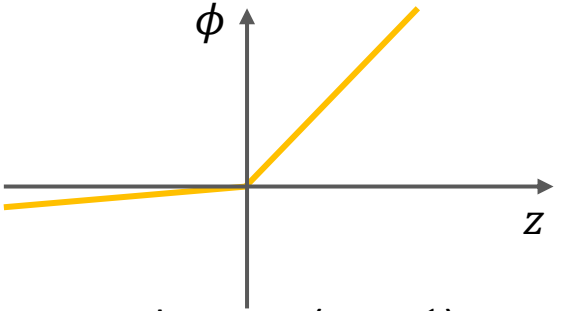
$$N_B = \left\lceil \frac{N}{M} \right\rceil .$$

WGAN-GP ALGORITHM

1	for i = 1, n_epochs	1 Loop over training iterations
2	for j = 1, n_batches	2 Loop over batches
3	Get $\{x\}_{l=1}^{n_b}$	3 Get a batch of data
4	for k = 1, n_critic_updates	4 Train the critic several times each batch
5	Get $\{z\}_{l=1}^{n_b}$	5 Create random input noise for generator
6	$\tilde{x} = g_\theta(z)$	6 Generate possible solution field
7	$\ell^{(j)} = f_w(x) - f_w(\tilde{x}) + L_{reg}(x, \tilde{x})$	7 Compute loss function
8	$w \leftarrow \text{Optimizer}\left(\frac{1}{n_b}\sum_{l=1}^{n_b} L^{(j)}; \boldsymbol{p}_{opt}\right)$	8 Average loss over batch; update critic w
9	end	9
10	$\tilde{x} = g_\theta(z)$	10 Create random noise for generator
11	$\ell^{(j)} = -f_w(\tilde{x}) + \|\mathcal{S}(x) - \mathcal{S}(\tilde{x})\|^2$	11 Compute generator loss function
12	$\theta \leftarrow \text{Optimizer}\left(-\frac{1}{n_b}\sum_{l=1}^{n_b} f_w(\tilde{x})\right)$	12 Average loss over batch; update generator θ
13	end	13
14	end	14

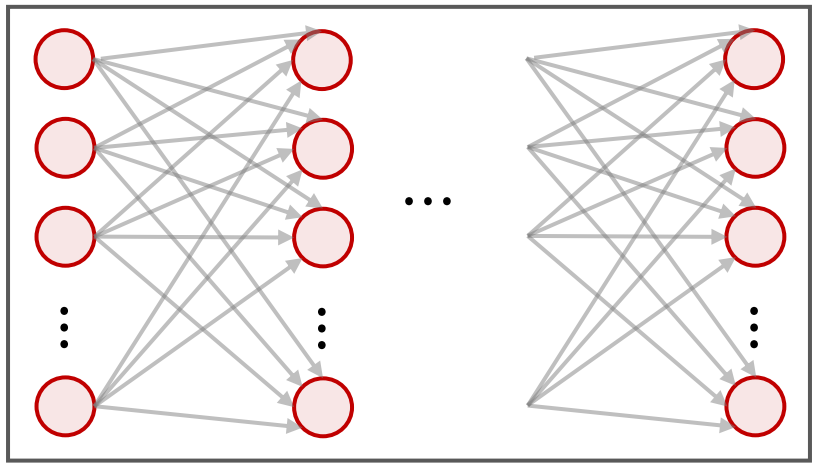
QUICK NEURAL NETWORK REVIEW





Leaky ReLU ($\alpha \ll 1$)

$$\phi(z) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$$



Deep, fully-connected feedforward network

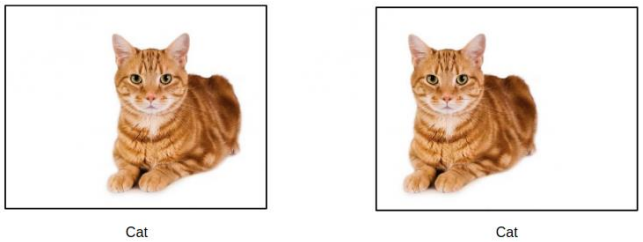
$$\boldsymbol{h}_K = \phi\big(\boldsymbol{W}^K \underbrace{\big(\phi(\boldsymbol{W}^{K-1}\boldsymbol{h}_{K-2} + \boldsymbol{b}^{K-1})\big)}_{\boldsymbol{h}_{K-1}} + \boldsymbol{b}^K\big)$$

Compositions of nonlinear function applied to matrix-vector multiplies.

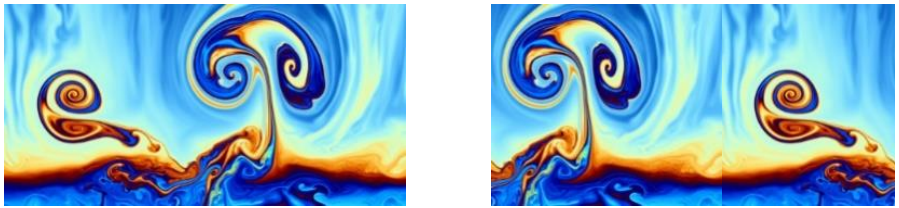
CONVOLUTIONAL NEURAL NETWORKS (CNNS)

Review and Basic Intuition

- CNNs have many nice properties
  - Sparsity
  - Weight-sharing
  - Translation equivariance



Cat translates to left, but it's still a cat.

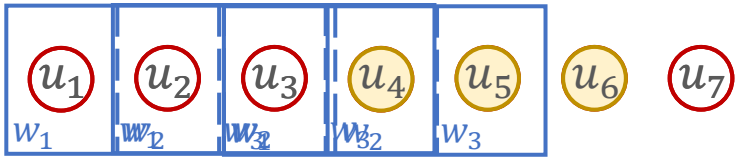


Temperature plumes move to left, but still plumes.

CONVOLUTIONAL NETWORKS

Basic Intuition

- For intuition, we will work with 1D arrays and move on to 2D later
- Start with solution array $\boldsymbol{u} \in \mathbb{R}^N$
- Introduce a filter k and sweep over the array to create the next layer



$$h_i = (u * k) = \sum_j u_j k_{i-j} = \sum_j u_{i-j} k_j$$

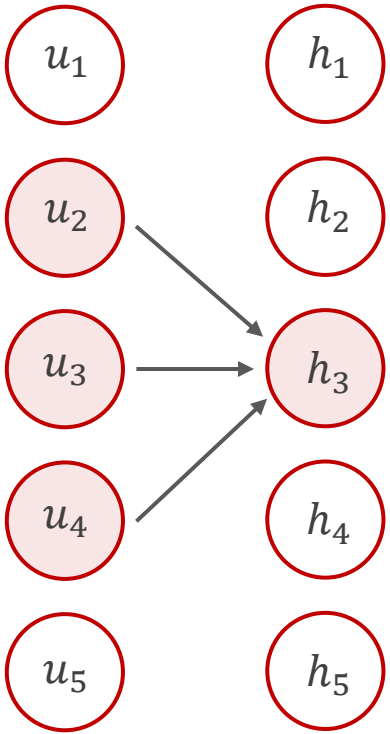
k has compact support

$$k = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

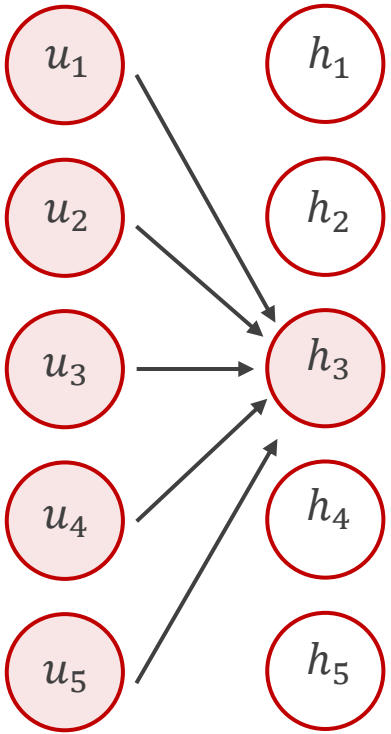
Need to determine parameters in the convolution kernel k.

After convolution, apply bias and activation function $h_i^{out} = \phi\big((u * k)_i + b_i\big)$.

SPARSITY OF CONVOLUTION VS. FULLY CONNECTED



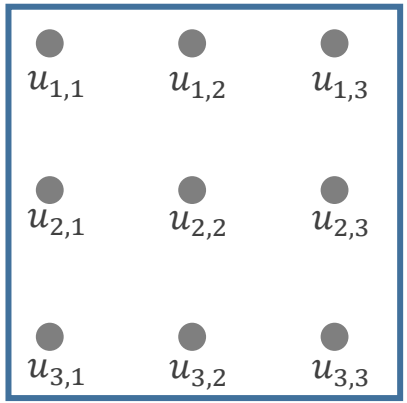
Convolutional Network



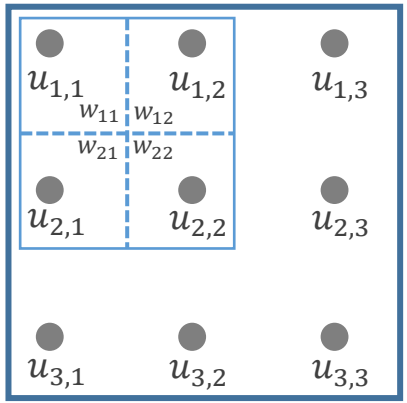
Fully Connected Network

- Convolutional networks can lead to substantial cost savings.
- First layer of fully-connected network has N weights per node.
- First layer of CNN has K weights per node where $K \ll N$ is the kernel size.

2D CNNS

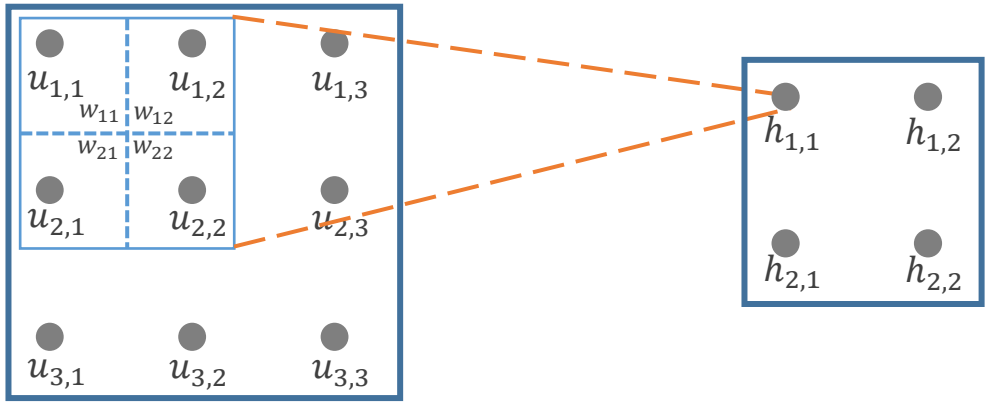


2D CNNS



$$k = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

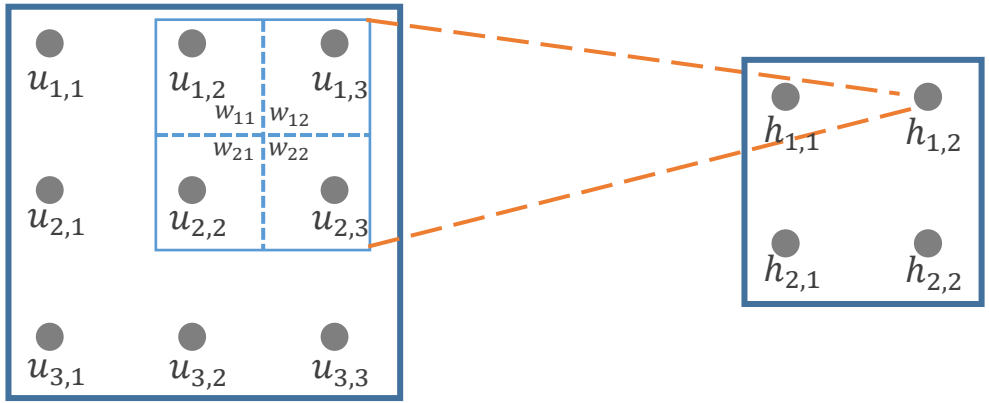
2D CNNS



$$k = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

$$h_{1,1} = u_{1,1}w_{11} + u_{1,2}w_{12} + u_{2,1}w_{21} + u_{2,2}w_{22}$$

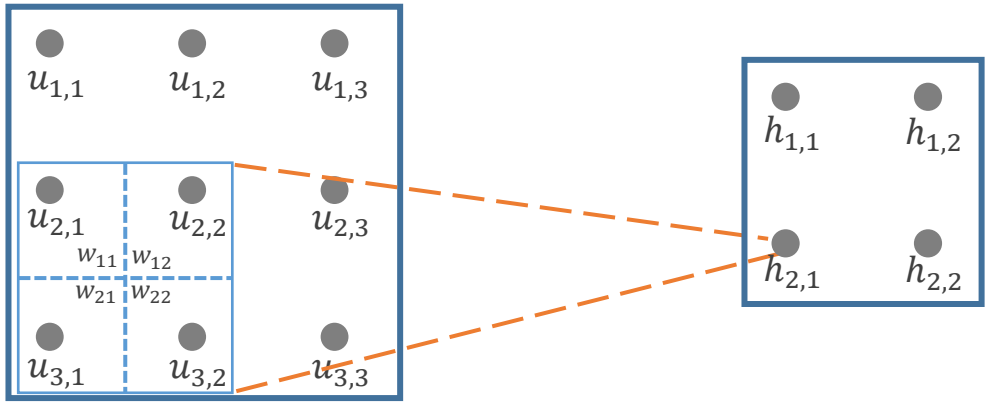
2D CNNS



$$k = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

$$h_{1,2} = u_{1,2}w_{11} + u_{1,2}w_{13} + u_{2,2}w_{21} + u_{2,3}w_{22}$$

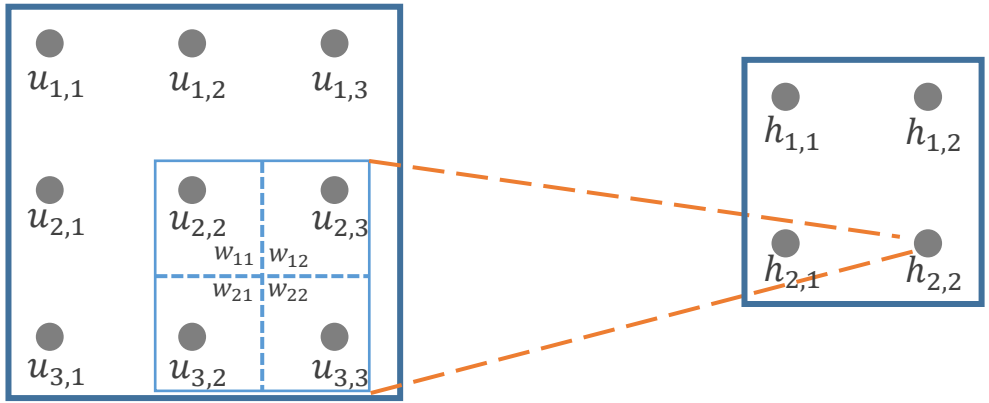
2D CNNS



$$k = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

$$h_{2,1} = u_{2,1}w_{11} + u_{2,2}w_{12} + u_{3,1}w_{21} + u_{3,2}w_{22}$$

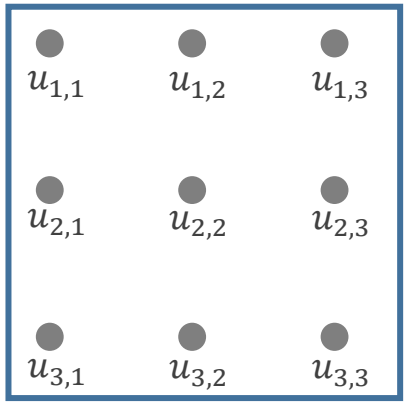
2D CNNS

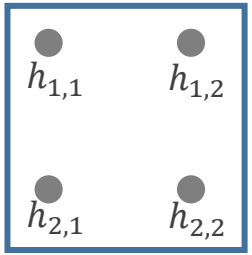


$$k = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

$$h_{2,2} = u_{2,2}w_{11} + u_{2,3}w_{12} + u_{3,2}w_{21} + u_{3,3}w_{22}$$

2D CNNS





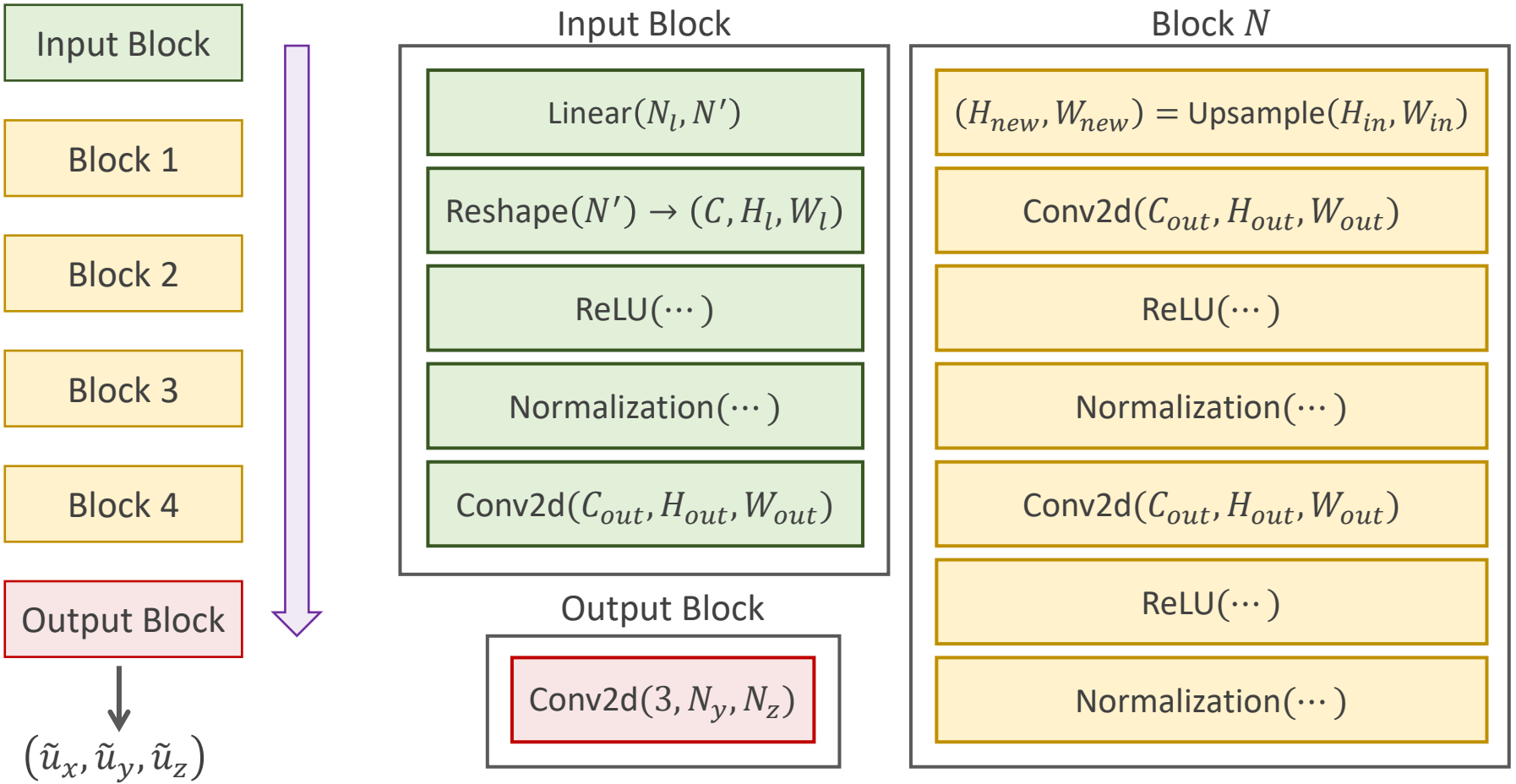
$$k = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

$$h_{ij} = (u * k)_{ij} = \sum_l \sum_m u_{l,m} k_{i-l,j-m} = \sum_l \sum_m u_{i-l,j-m} k_{lm}$$

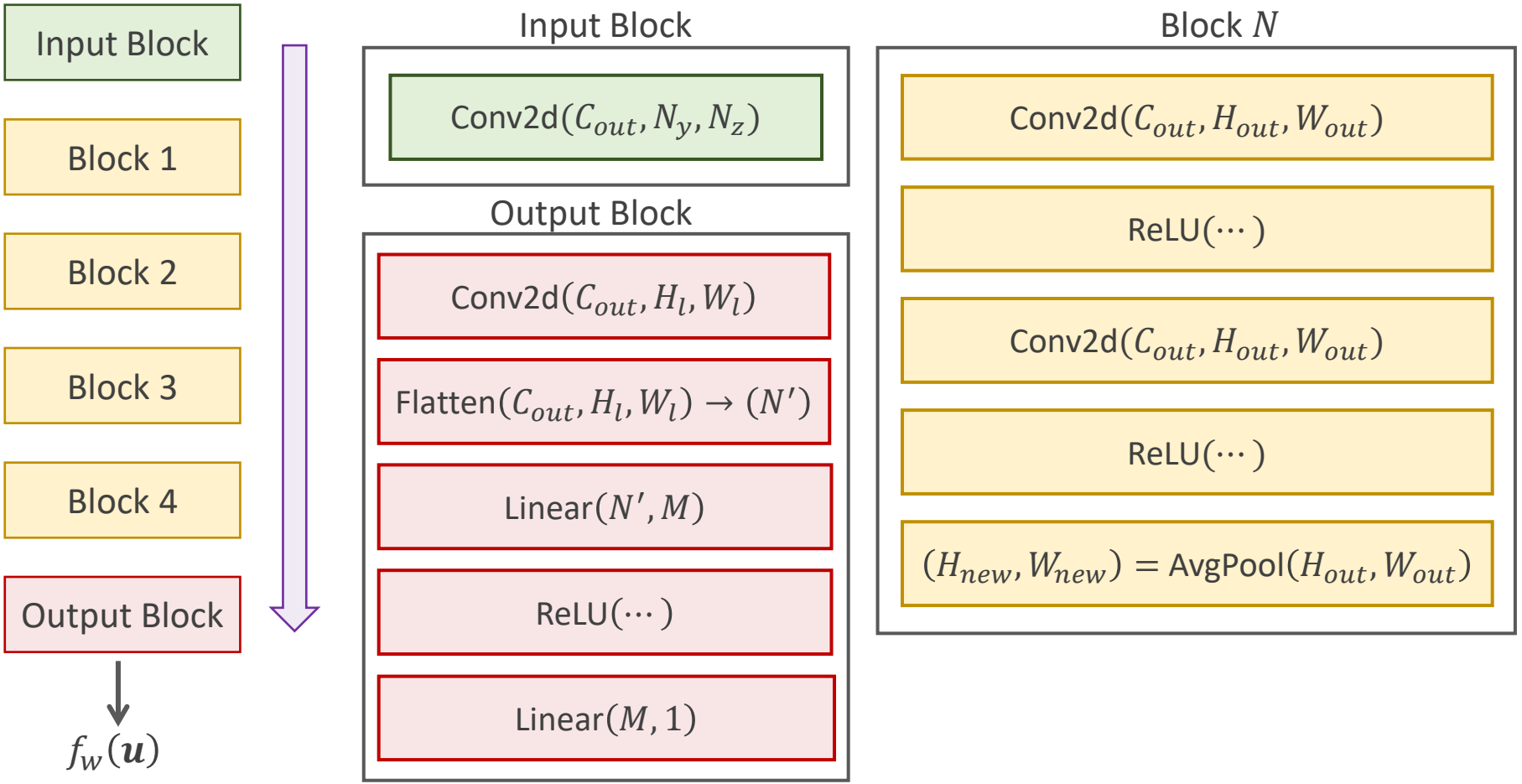
- Matrix inner-product with the kernel
- Train network to learn kernel parameters
- Can generalize to more than two dimensions

$$h_{ij}^{out} = \phi\left((u * k)_{ij} + b_{ij}\right)$$

WGAN-GP ARCHITECTURE: GENERATOR



WGAN-GP ARCHITECTURE: CRITIC



TRAINING DETAILS

Dataset size, model hyperparameters, and hardware

Total training samples available	32708 fields of size 128×512
Total dataset size available	25 GB
Batch size	64
Epochs	$1000 - 5000$
Critic updates per epoch	5
Optimizer	Adam
Learning rates (generator, critic)	$(5 \times 10^{-4}, 5 \times 10^{-4})$
Dimension of latent space to generator	128
Hardware	NVIDIA Tesla V100 (32 GB RAM)

SOME BACKGROUND ON GANS

The actual dataset represents samples from some underlying probability distribution P_r.

$$x \sim P_r$$

The generator is learning a mapping from a simple, known distribution (e.g. normal) $z \sim P_z$ to samples drawn from the real distribution $\tilde{x} \sim P_r$.

In this way, it learns an *implicit representation* of the real underlying distribution.

SOME BACKGROUND ON WGANS

We want to compare samples from the real distribution to samples from the generator. This can be formulated in the language of optimal transport theory.

Given two probability distributions P_r and P_g, how much does it cost to turn them into each other.

Another interpretation: treating P_r and P_g as "piles of dirt", how much does it cost to turn P_g into P_r?

This motivates the so-called "Earth-mover distance"

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma} [\|x - y\|] \quad \bigstar$$

where Π is the set of all joint distributions and γ is a joint distribution over x and y. $\inf$ is the infimum (like an infinite-dimensional minimum).

SOME BACKGROUND ON WGANS

★ is way too expensive to actually work with. It can be turned into a more tractable quantity via the Kantorovitch-Rubenstein duality from optimal transport theory. The Earth-move distance becomes

$$W(P_r, P_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} \left(\mathop{\mathbb{E}}_{x \sim P_r} [f(x)] - \mathop{\mathbb{E}}_{x \sim P_g} [f(x)] \right).$$

The function f must be a K −Lipschitz function. The basic motivation for Lipschitz continuity is that it provides a way to describe how fast a function changes. A function $f: \mathbb{R} \to \mathbb{R}$ is K − Lipschitz continuous for some positive, real constant K if

$$|f(p_1) - f(p_2)| \leq K|p_1 - p_2|$$

for two points p_1 and p_2.

The Wasserstein GAN uses this Earth-mover distance as a way of measuring how close the generated samples are from the true dataset.

SOME BACKGROUND ON WGANS

The Wasserstein GAN attempts to find critic parameters that maximize and generator parameters that minimize the $1-$ Lipschitz version of this re-cast Earth-mover distance

$$W(P_r, P_g) = \sup_{\|f\|_L \leq 1} \left(\mathop{\mathbb{E}}_{x \sim P_r} [f_w(x)] - \mathop{\mathbb{E}}_{\tilde{x} \sim P_g} [f_w(\tilde{x})] \right)$$

where $\tilde{x} = g_\theta(z)$ for $z \sim P_z$.

Remarks:

- The critic wants to keep the samples are apart. If it can do that, then it can tell which samples are real and fake.
- The generator wants to bring the samples close together. If it can do that, then it can reliably generate samples that look like those sample from the real distribution. Indeed, the generator may even begin to learn an implicit representation of the real distribution.
- There is still a question of how to enforce the Lipschitz constraint.

SOME BACKGROUND ON WGANS

One way to enforce the Lipschitz constraint is to simply cap the critic weights to a unit hypercube. This is *ad hoc* and can lead to undesirable behavior during training.

Instead, the Lipschitz constraint can be enforced softly using a penalty term by observing that a function f is $1-$ Lipschitz if and only if f has gradient norm equal to 1 everywhere. This indicates that a suitable penalty term should have the form $(\|\nabla f_w(\hat{x})\|_2 - 1)^2$ for some $\hat{x}$.

A "reasonable" choice for $\hat{x}$ is

$$\hat{x} = \alpha x + (1-\alpha)\tilde{x} \quad \alpha \sim \mathcal{U}[0,1]$$

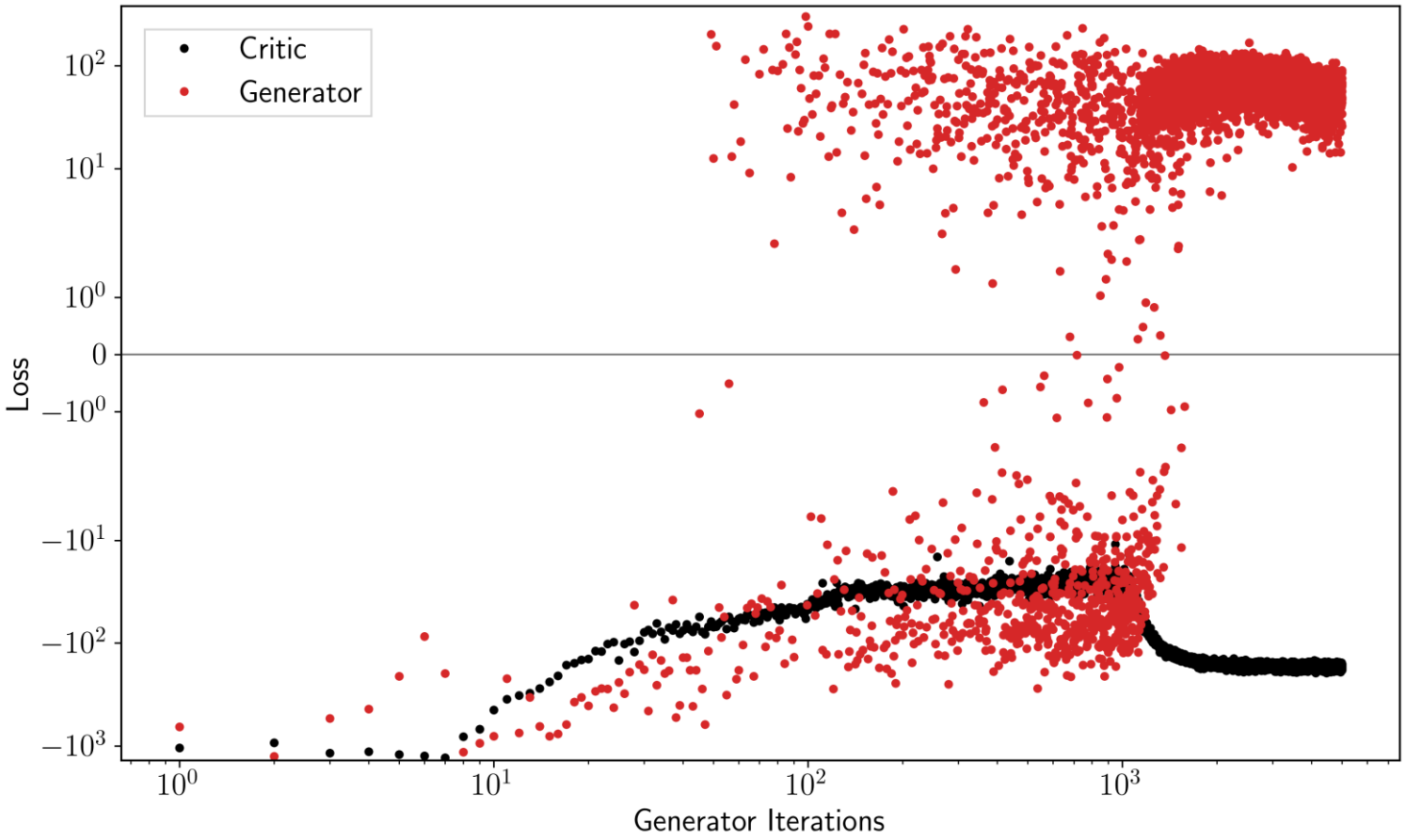
so that $\hat{x}$ is a sample along straight lines connecting the real and fake samples.

Finally, the WGAN with Gradient Penalty loss is as follows

$$\max_w \left(\mathop{\mathbb{E}}_{x\sim P_r}[f_w(x)] - \mathop{\mathbb{E}}_{z\sim P_z}\left[f_w\left(g_\theta(z)\right)\right] + \lambda_C(\|\nabla f_w(\hat{x})\|_2 - 1)^2 \right)$$

$$\min_\theta \left(\mathop{\mathbb{E}}_{x\sim P_r}[f_w(x)] - \mathop{\mathbb{E}}_{z\sim P_z}\left[f_w\left(g_\theta(z)\right)\right] \right)$$

RESULTS: LOSS FUNCTION DURING TRAINING



APPENDIX

WASSERSTEIN GAN

- The Wasserstein GAN (WGAN) offers a new interpretation on GANs and overcomes some of the traditional problems.
- The starting point for the WGAN is the Wasserstein distance.
  - Given two probability density functions (p_r and p_g), how much mass would you need to take from one distribution to turn it into the other distribution.
  - This is also called the Earth-mover distance
- After some technical details involving theorems from functional analysis, the loss function for the WGAN can be cast as

$$L(p_r, p_g) = \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_z}\left[f_w\left(g_\theta(z)\right)\right]$$

The goal is to find w that maximizes L:

$$\max_w \left(\mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_z}\left[f_w\left(g_\theta(z)\right)\right]\right)$$

Where f is a $K-$ Lipschitz continuous function.

- Need to enforce the $K-$ Lipshitz constraint somehow.
  - Ugly option: clip the weights during training

WASSERSTEIN GAN WITH GRADIENT PENALTY

- A function is $1-$ Lipschitz iff it has gradient norm equal to 1 almost everywhere.
- Enforce this softly as a penalty on the loss function.
- The new loss function, which will be *minimized,* is

$$L = -\mathbb{E}_{x\sim p_r}[f_w(x)] + \mathbb{E}_{z\sim p(z)}\left[f_w\left(g_\theta(z)\right)\right] + \lambda\mathbb{E}_{\hat{x}\sim p_{\hat{x}}}[(\|\nabla_{\hat{x}} f_w(\hat{x})\|_2 - 1)^2]$$

where λ is a penalty coefficient and $\hat{x}$ is a sample between the real sample x and fake sample $\tilde{x}$ given by

$$\hat{x} = \alpha x + (1-\alpha)\tilde{x}$$

with $\alpha \sim U[0,1]$.