

DeepMind

Learning general purpose physical simulators

Presenter: Alvaro Sanchez-Gonzalez

April 19, 2022

Workshop on Representation Learning from
Heterogeneous/Graph-Structured Data

Learning to Discover – Institut Pascal Paris-Saclay



Our DeepMind team



Kelsey Allen



Peter Battaglia



Meire Fortunato



Jonathan Godwin



Tatiana Guevara



Jessica Hamrick



Tobias Pfaff



Yulia Rubanova



Alvaro Sanchez



Kimberly Stachenfeld

And external
collaborators:

Google Research
Dmitrii Kochkov

Flatiron Institute
Drummond Fielding
Can Cui
Shirley Ho

Princeton University
Miles Cranmer

Stanford University
Jure Leskovec
Rex Ying



Why *learn* simulation?

1. Speed

Learn subgrid dynamics, compensate for large time stepping
classical limits (e.g. CFL condition) don't directly apply

2. Differentiability

Improved design optimization, boundary inference, control

3. Learn unknown physics

compensate for unknown models/parameters

4. Distill reusable modules

which run efficiently on modern accelerator hardware






But!

AI, ML & DATA ENGINEERING

QCon Plus (May 17-28): Stay Ahead of Emerging Software Trends

Deep Learning Accelerates Scientific Simulations up to Two Billion Times

 LIKE  DISCUSS 

MAR 10, 2020 • 3 MIN READ

by

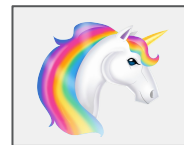
Anthony Alford

FOLLOW

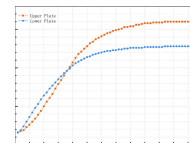
Input parameters

$t=0.3$
 $x=1.2$
 $v=9.9$

Magical NN



High-dim output



But!

AI, ML & DATA ENGINEERING

QCon Plus (May 17-28): Stay Ahead of Emerging Software Trends

Deep Learning Accelerates Scientific Simulations up to Two Billion Times

 LIKE  DISCUSS 

MAR 10, 2020 • 3 MIN READ

by

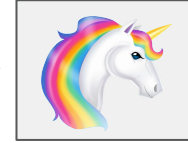
Anthony Alford

FOLLOW

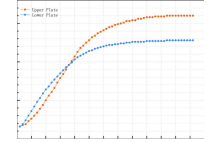
Input parameters

$t=0.3$
 $x=1.2$
 $v=9.9$

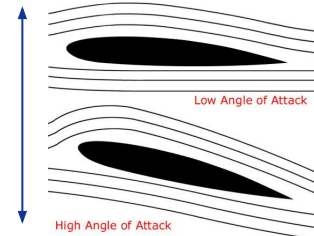
Magical NN



High-dim output



Training



Testing



But!

AI, ML & DATA ENGINEERING

QCon Plus (May 17-28): Stay Ahead of Emerging Software Trends

Deep Learning Accelerates Scientific Simulations up to Two Billion Times

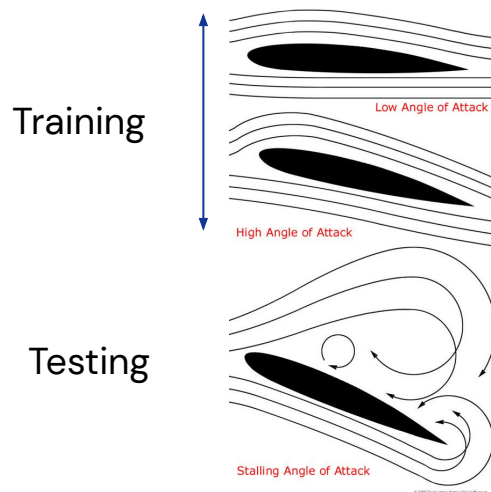
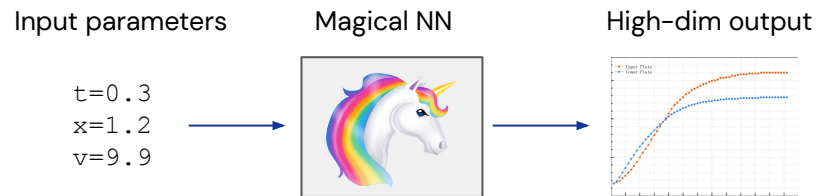
LIKE DISCUSS

MAR 10, 2020 • 3 MIN READ

by
Anthony Alford

FOLLOW

1. Neural networks are good at interpolation, bad at extrapolation
2. Learned physics models often don't learn anything close to the underlying physical equations
3. There's no way we can build a dataset that covers the input space of a general-purpose simulator



Strong generalization using Graph Networks

Structure the learning setup such that we learn *reusable* knowledge, similar to physical laws, and can apply our model far outside training set conditions.

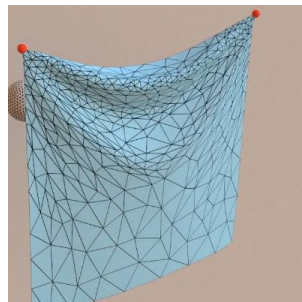


Learning to Simulate Complex Physics with Graph Networks

ICML 2020

arxiv.org/pdf/2002.09405

sites.google.com/view/learning-to-simulate



Learning Mesh-Based Simulation with Graph Networks

ICLR 2021, Outstanding paper award

arxiv.org/pdf/2010.03409.pdf

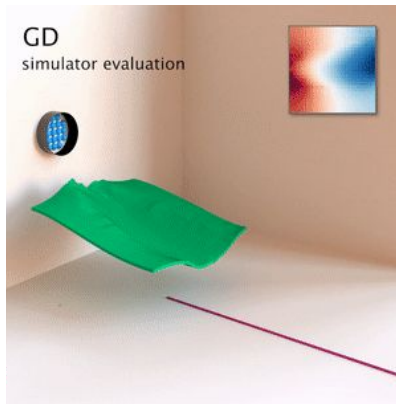
sites.google.com/view/meshgraphnets

Learned simulation model with very desirable properties:

- Strong generalization
- Graph models which scale (we demonstrate up to 100k nodes)
- Stable rollouts
- Speed: 10-100x faster than ground truth simulator
- Same model works on vastly different systems



Follow-up simulation work

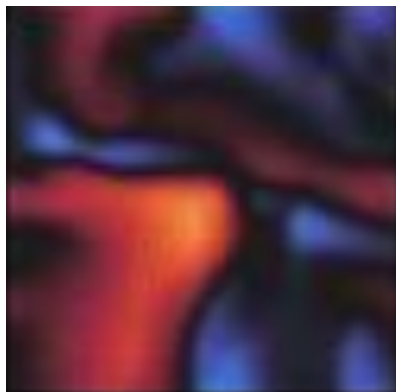


Physical Design using Differentiable Learned Simulators
arXiv (ICML 2022 submission)

arxiv.org/pdf/2202.00728.pdf
sites.google.com/view/learning-to-simulate

Constraint-based graph network simulator
arXiv (ICML 2022 submission)

arxiv.org/pdf/2112.09161.pdf
sites.google.com/view/constraint-based-simulator

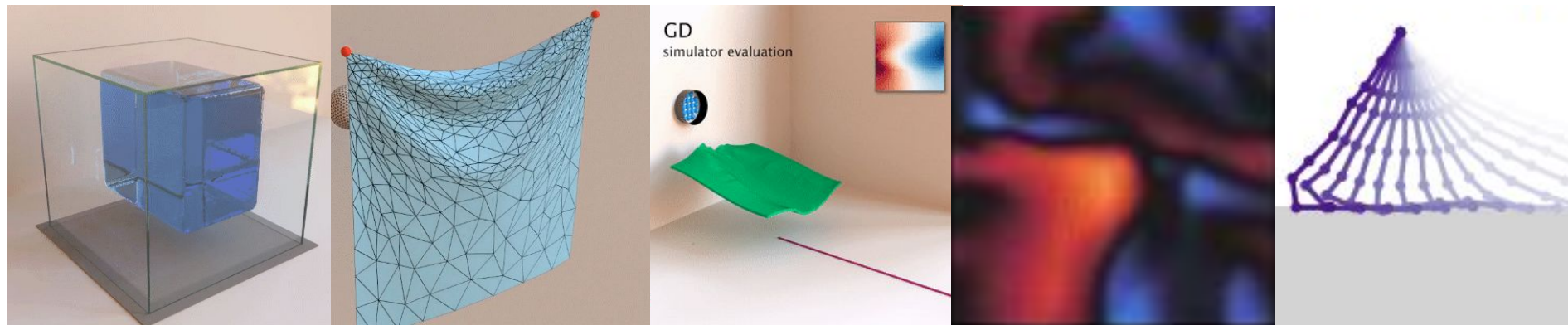


Learned Coarse Models for Efficient Turbulence Simulation
ICLR 2022

arxiv.org/pdf/2112.15275.pdf
sites.google.com/view/learning-to-simulate



Strong generalization using Graph Networks

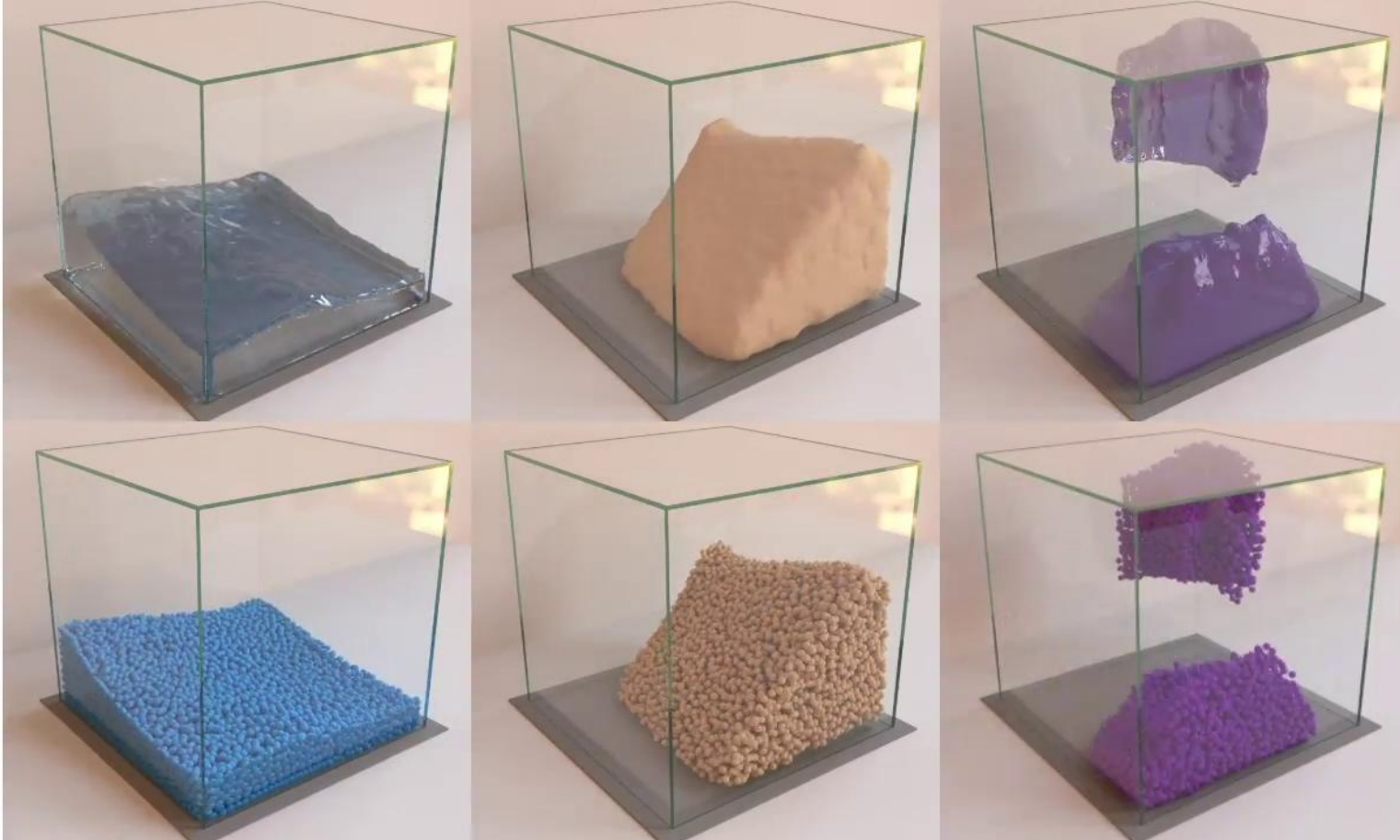


What this talk *isn't* about:

- An actual product (this is basic research :)
- Accuracy & convergence guarantees
- Mixing learned models with hard-coded solvers

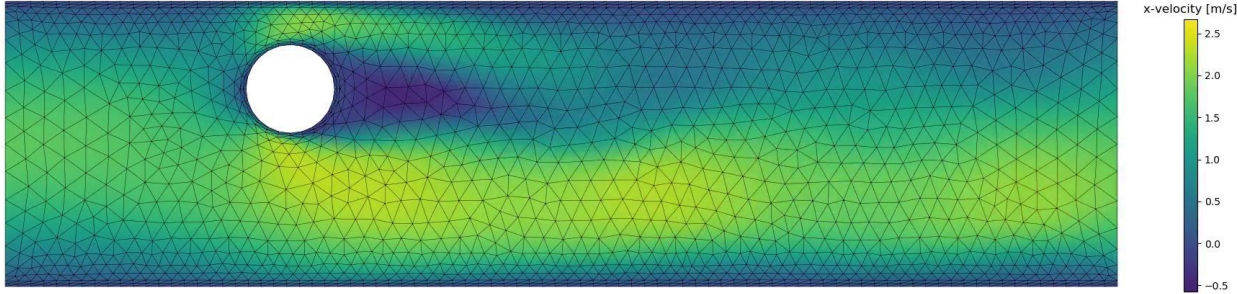


Particle-based simulation

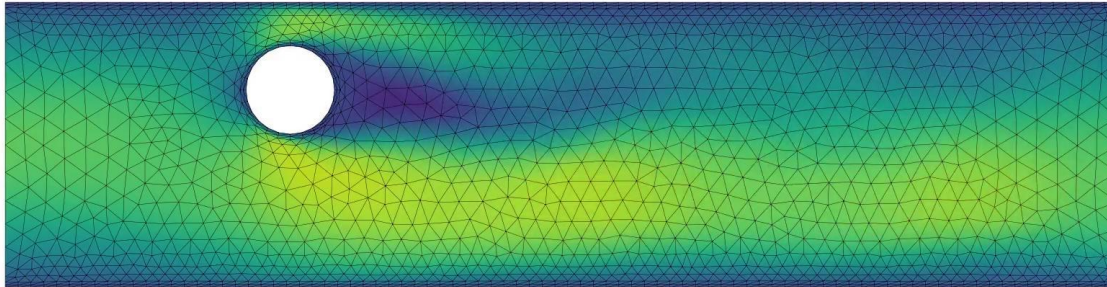


Incompressible fluids

Ground truth



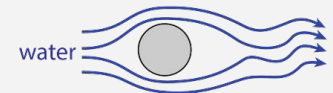
Prediction



Eulerian simulation
[COMSOL]

triangular mesh

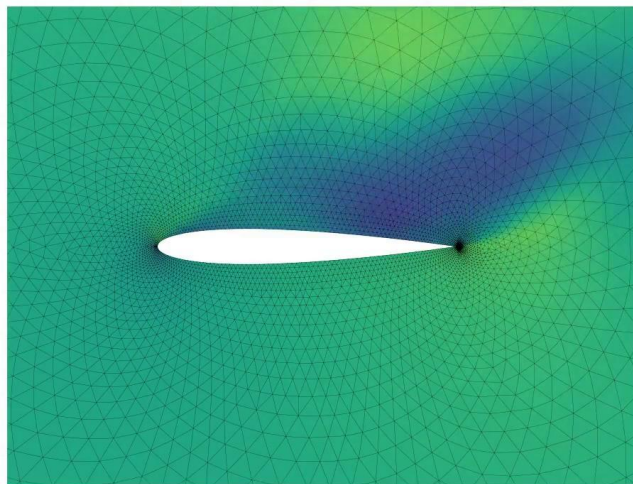
Network output:
velocity field
pressure field



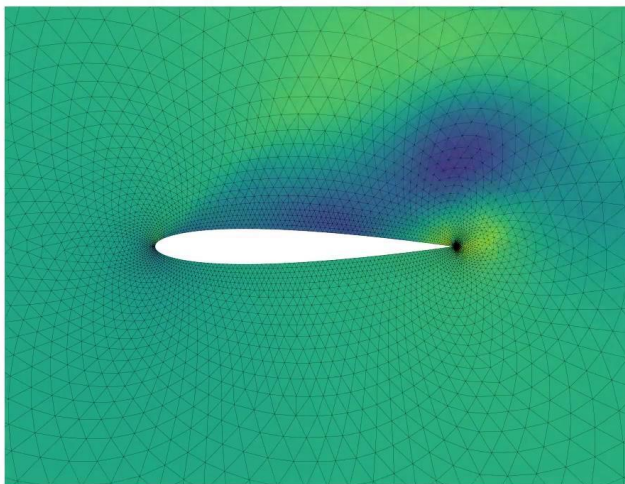
Aerodynamics

Ground truth

mach number 0.58
angle of attack 21.9



Prediction



Eulerian simulation
[SU2]

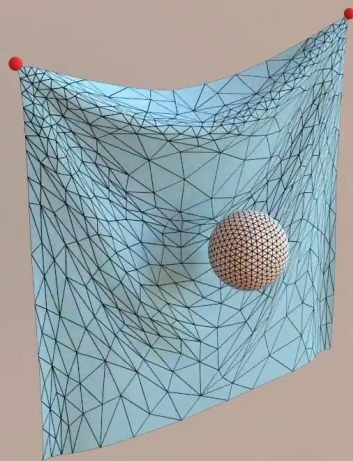
triangle mesh

Network output:
velocity field
density field
pressure field

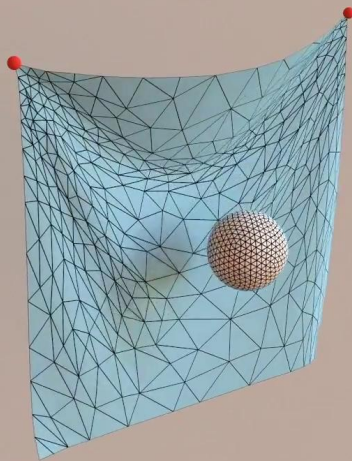


Cloth dynamics

Ground truth



Prediction
(with learned remeshing)



Lagrangian
simulation
[Arcsim]

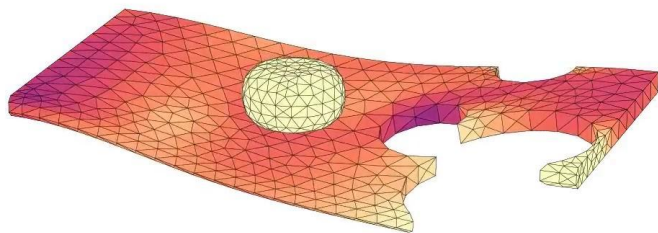
dynamic triangular
mesh

Network output:
per-node
acceleration

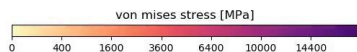
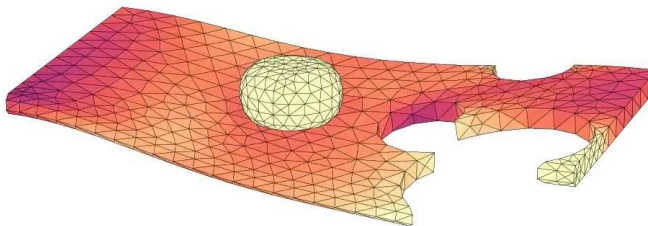


Structural mechanics

Ground truth



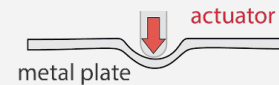
Prediction



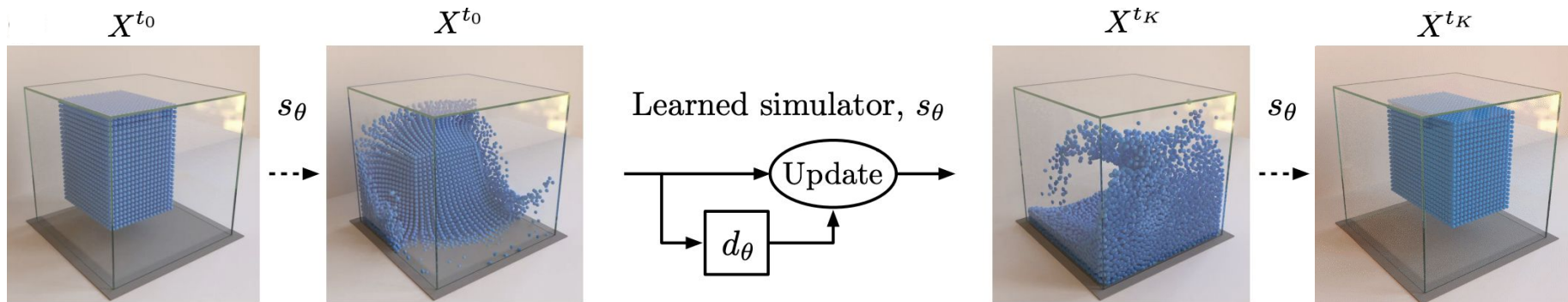
Lagrangian
quasi-static
simulation
[COMSOL]

tetrahedral mesh

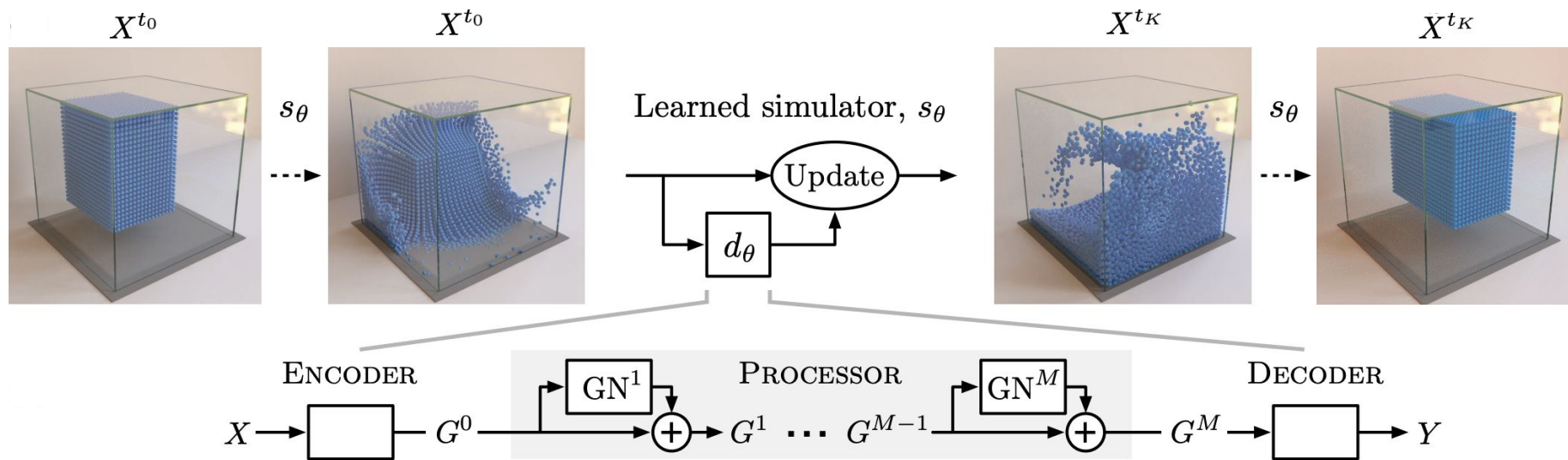
Network output:
per-node
position change



How does it work?



How does it work?



Main design principle: **neural networks are dumb, let's make their life easy**



***“If I have seen further it is by
standing on the shoulders of giants.”***

-Sir Isaac Newton-

**Our Neural Networks should also *have
the knowledge of giants!***



Inductive Biases

Inductive biases

“An **inductive bias** allows a learning algorithm to prioritize one solution (or interpretation) over another.”

Mitchell, T. M.. *The need for biases in learning generalizations*. (1980)

Inductive bias → Prior for generalization



A simple inductive bias: Inertial dynamics



Position: $x(t)$

Velocity: $v(t)$

$$\sum \mathbf{F} = m\mathbf{a} = m \frac{d^2 \mathbf{x}}{dt^2}$$

$$x^{t+1} = \mathbf{NN}(x^t, v^t)$$

Static prior

$$x^{t+1} = x^t + \mathbf{NN}(x^t, v^t)$$

Inertial prior

$$x^{t+1} = x^t + \Delta t \cdot v^t + \mathbf{NN}(x^t, v^t)$$

Has to learn to predict static motion

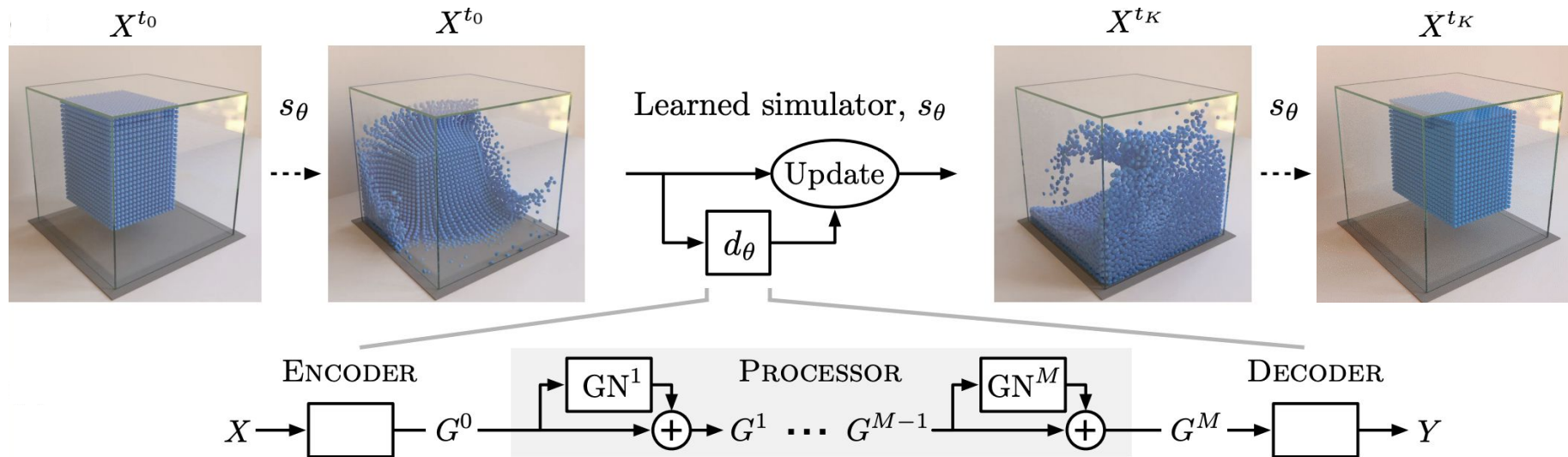
Trivial to predict static motion

Has to learn to predict inertial motion

Trivial to predict inertial motion!



Physics-inspired inductive biases!



**Spatial
equivariance**

**Local
interactions**

**Universal
rules**

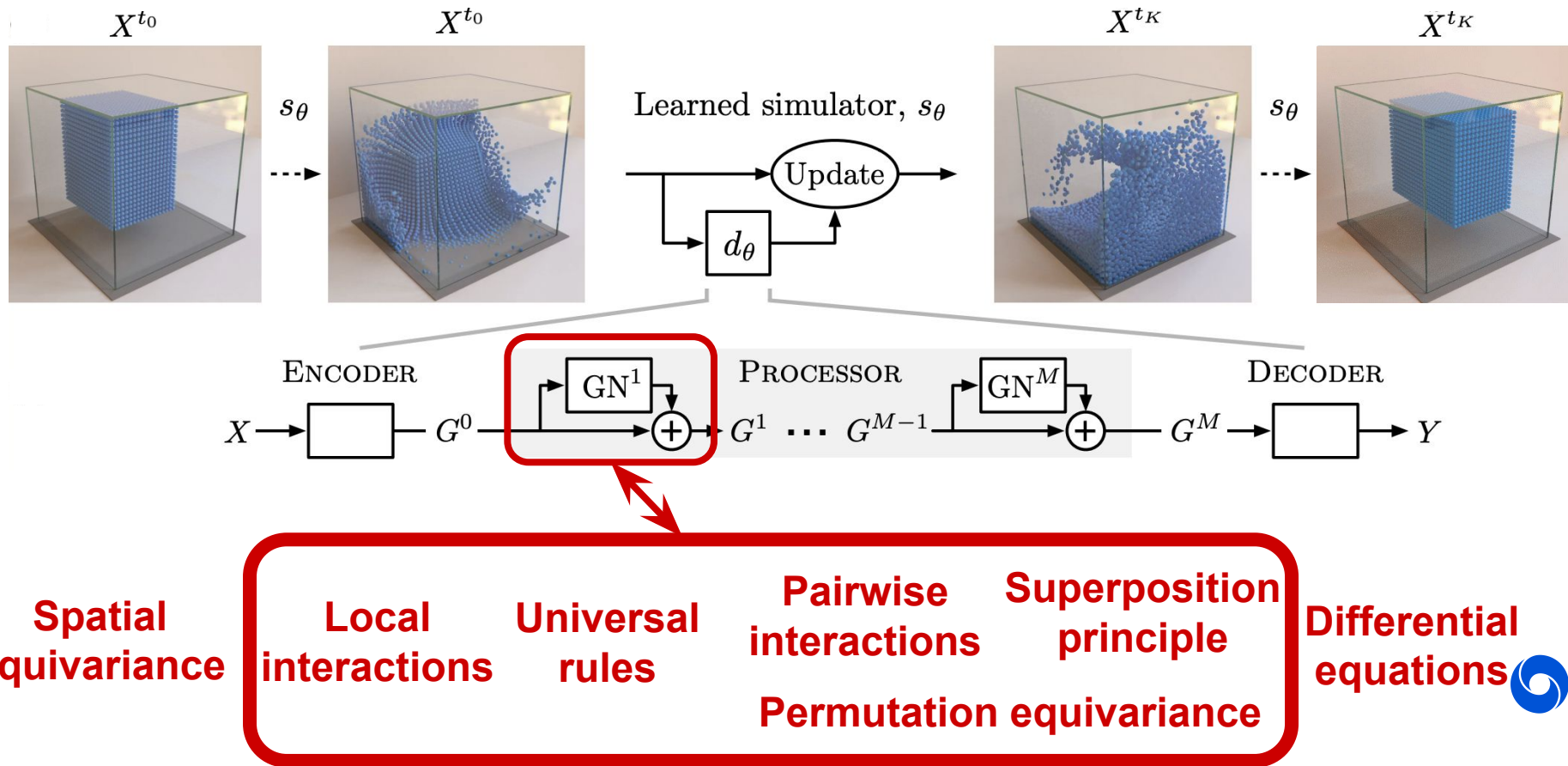
**Pairwise
interactions**
Permutation equivariance

**Superposition
principle**

**Differential
equations**

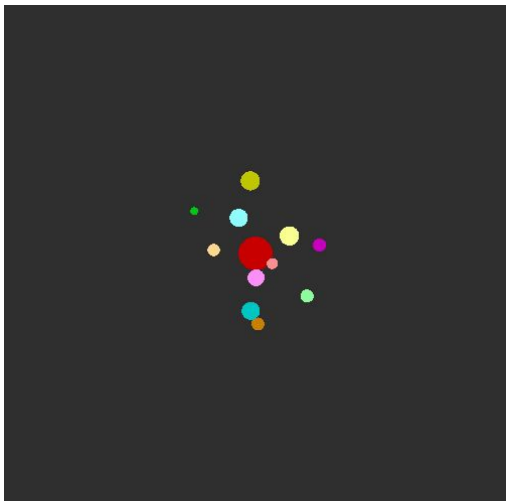


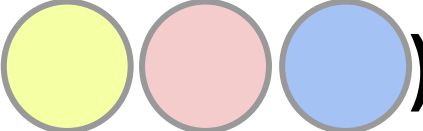
Physics-inspired inductive biases!



Graph Networks

- MLPs **operate over vectors**

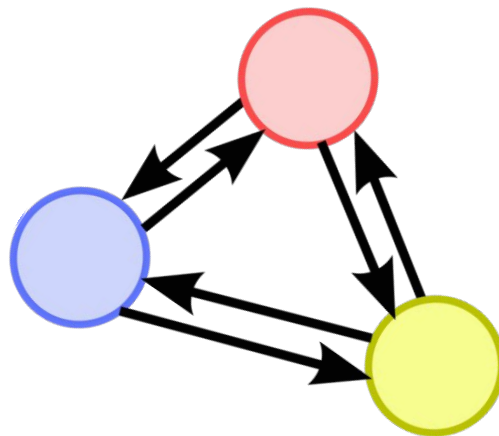
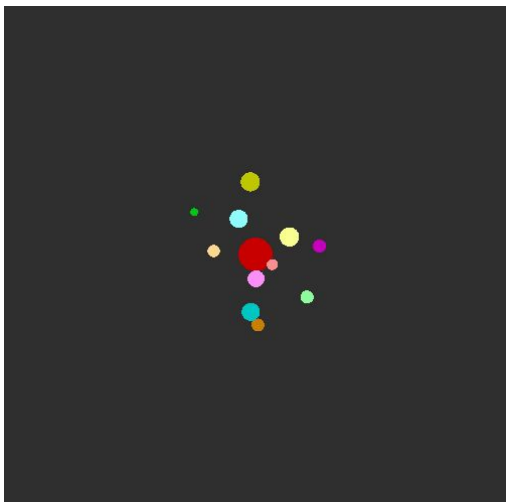


MLP()



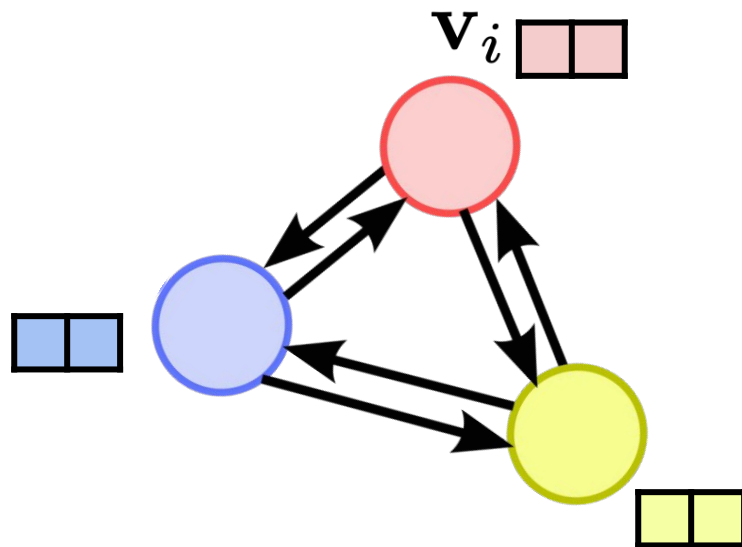
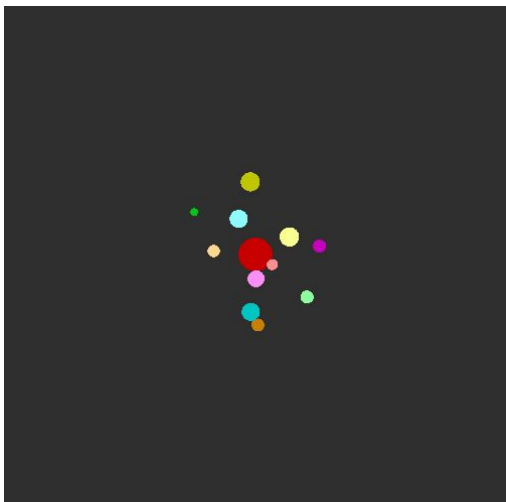
Graph Networks

- Neural networks that **operate over graphs**



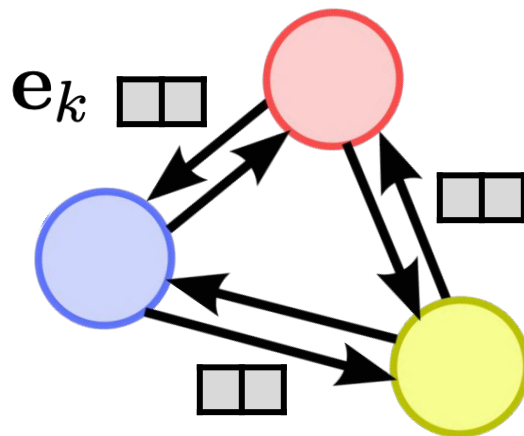
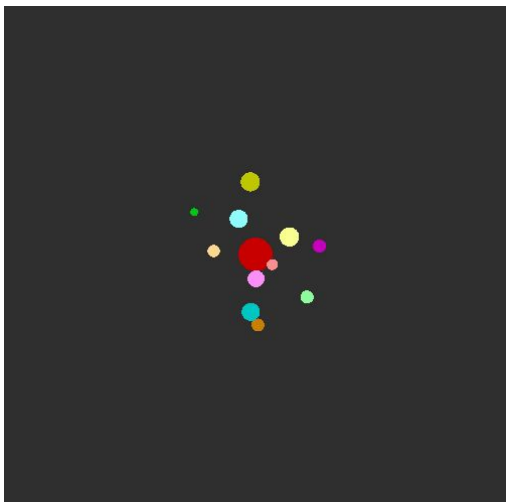
Graph Networks

- Neural networks that **operate over graphs**
 - Node features



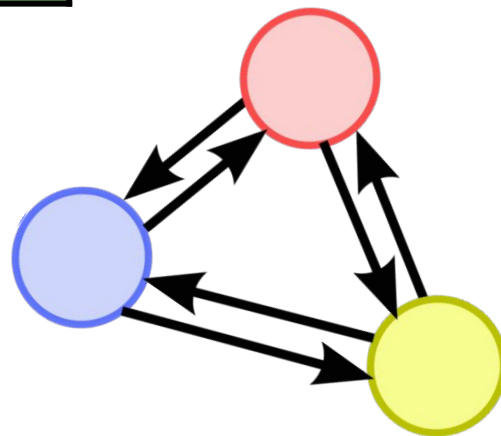
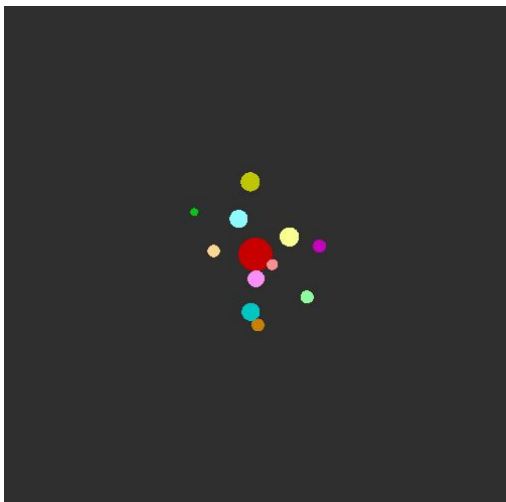
Graph Networks

- Neural networks that **operate over graphs**
 - Edge features



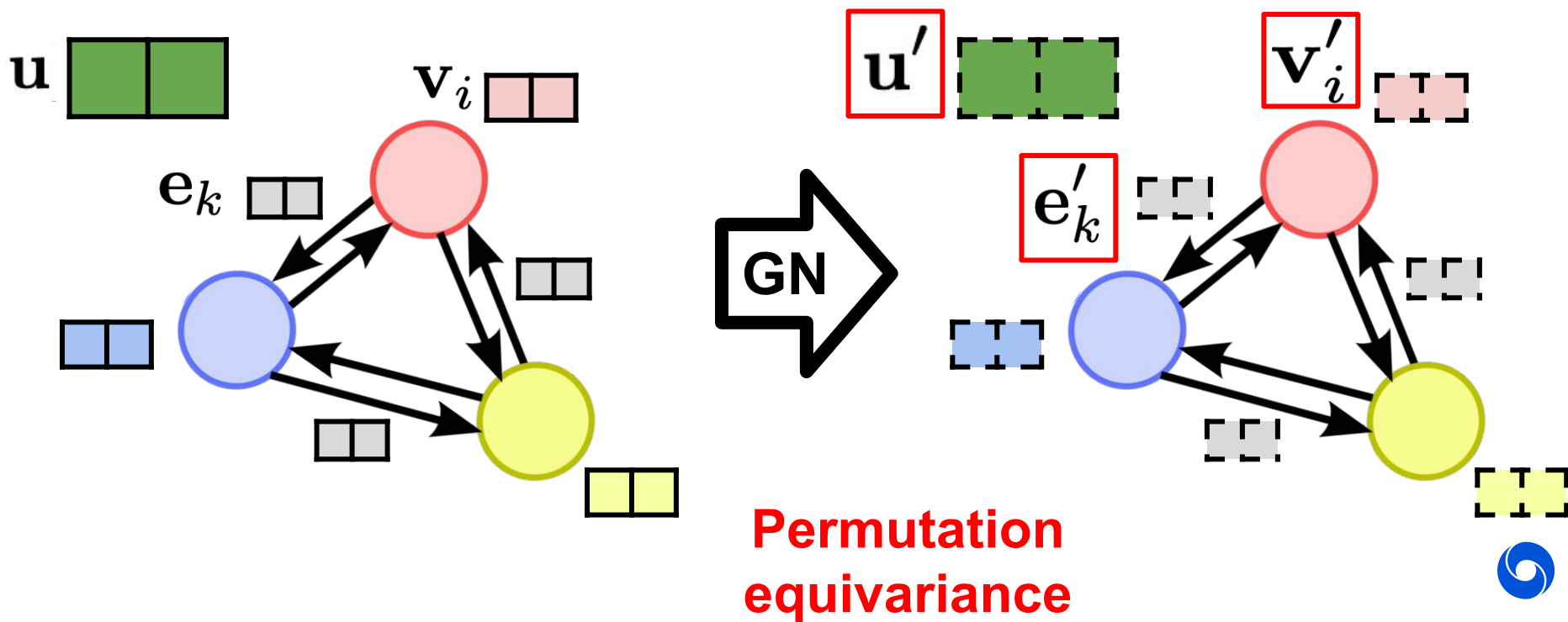
Graph Networks

- Neural networks that **operate over graphs**
 - Global features



Graph Networks

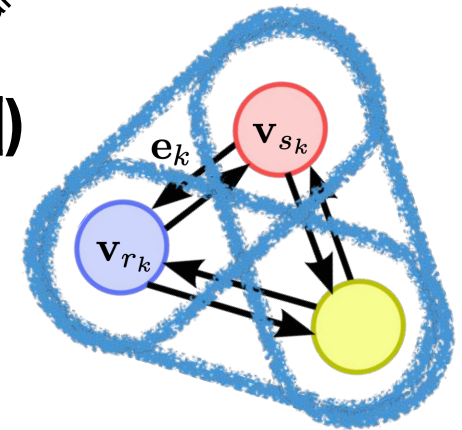
Update edge, node and global embeddings



Message passing: Edge update

Edge (message) function (for every edge)

$$\boxed{\mathbf{e}'_k} \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) := \text{NN}_e \left(\begin{array}{|c|c|c|c|c|c|c|c|} \hline \text{Edge features} & \text{Receiver node features} & \text{Sender node features} & \text{Global features} & & & & \\ \hline \end{array} \right)$$



**Pairwise
interactions**

**Universal
rules**



Message passing: Node update

Receiver edge aggregation (Message pooling) (for every node)

$$\bar{\mathbf{e}}'_i \leftarrow \sum_{r_k=i} \mathbf{e}'_k$$

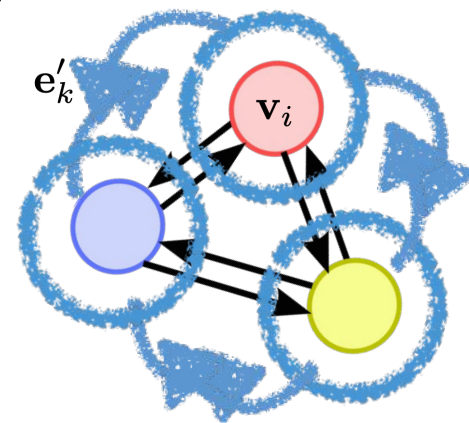
Superposition principle

Node function (for every node)

$$\boxed{\mathbf{v}'_i} \leftarrow \phi^v (\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) := \text{NN}_v (\text{Aggregated edge features} \quad \text{Node features} \quad \text{Global features})$$

Local interactions

Universal rules



Graph Networks: Global update

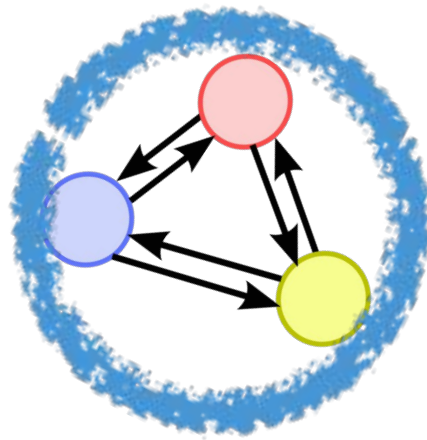
Global node and edge aggregation

$$\bar{\mathbf{v}}' \leftarrow \sum_i \mathbf{v}'_i \quad \bar{\mathbf{e}}' \leftarrow \sum_k \mathbf{e}'_k$$

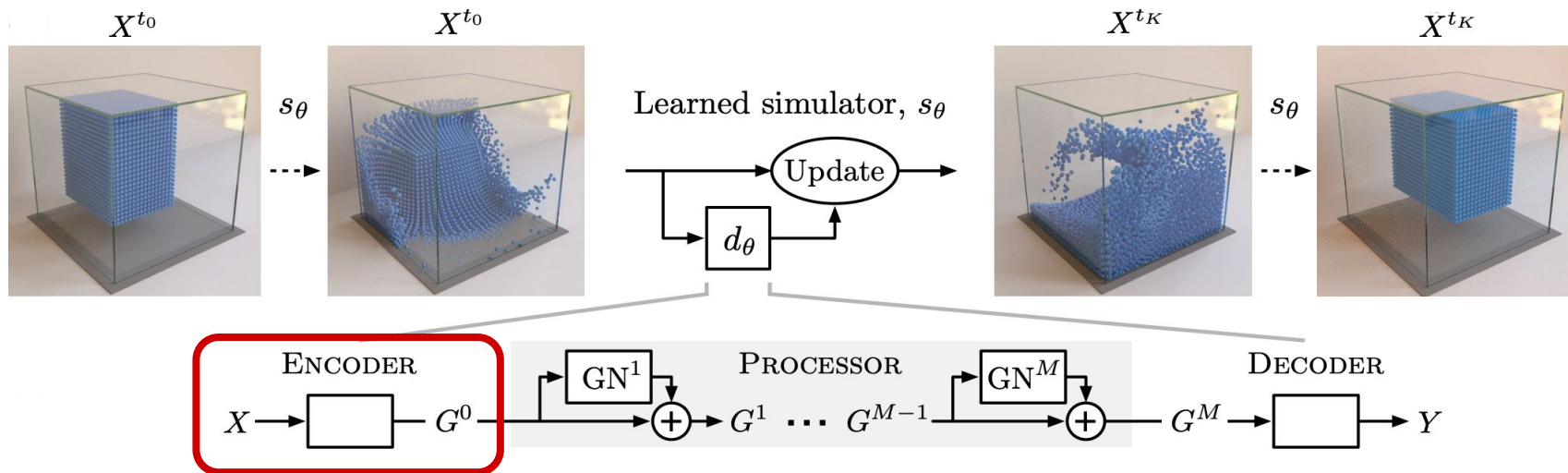
Global function

$$\boxed{\mathbf{u}'} \leftarrow \phi^u (\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) := \text{NN}_u (\text{Aggregated edge features} \quad \text{Aggregated node features} \quad \text{Global features})$$

~~Local interactions~~



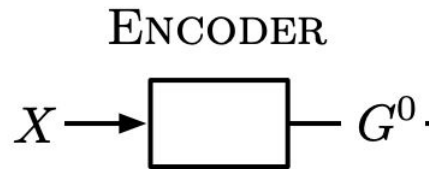
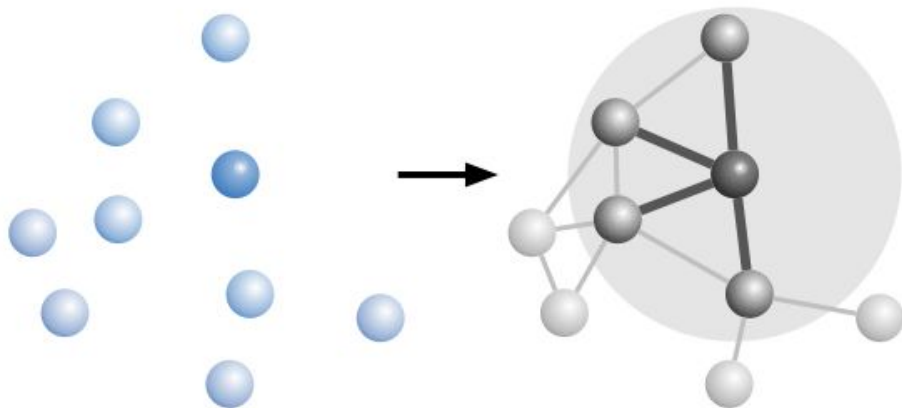
Encoding Graphs



Encoding Graphs

Transform the inputs into a graph

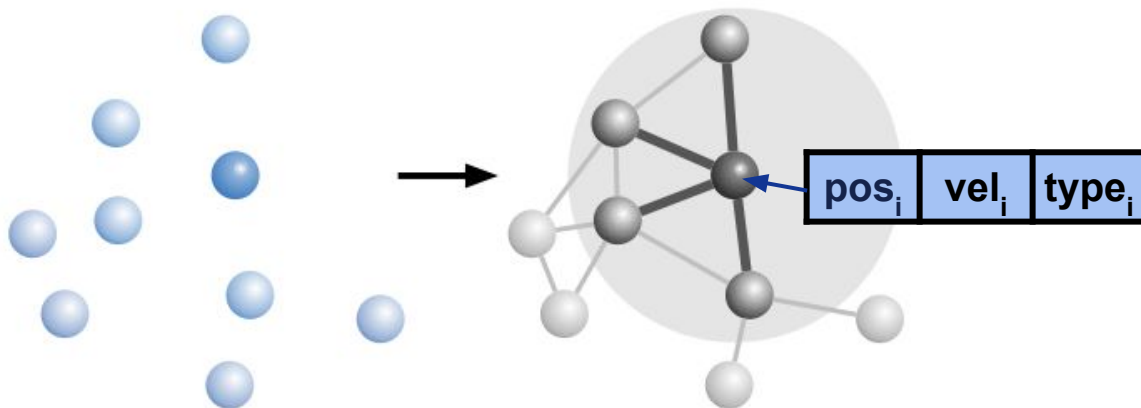
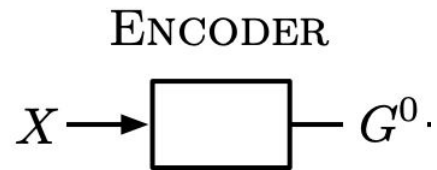
- Edges: Mesh edges (for meshes) or proximity-based (for particles)



Encoding Graphs

Transform the inputs into a graph

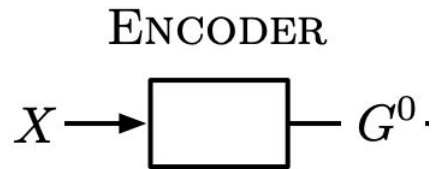
- Edges: Mesh edges (for meshes) or proximity-based (for particles)



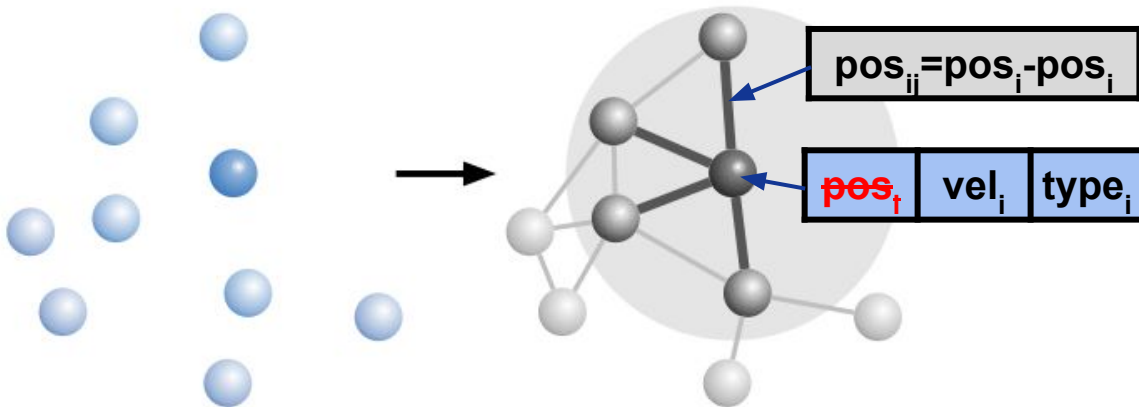
Encoding Graphs

Transform the inputs into a graph

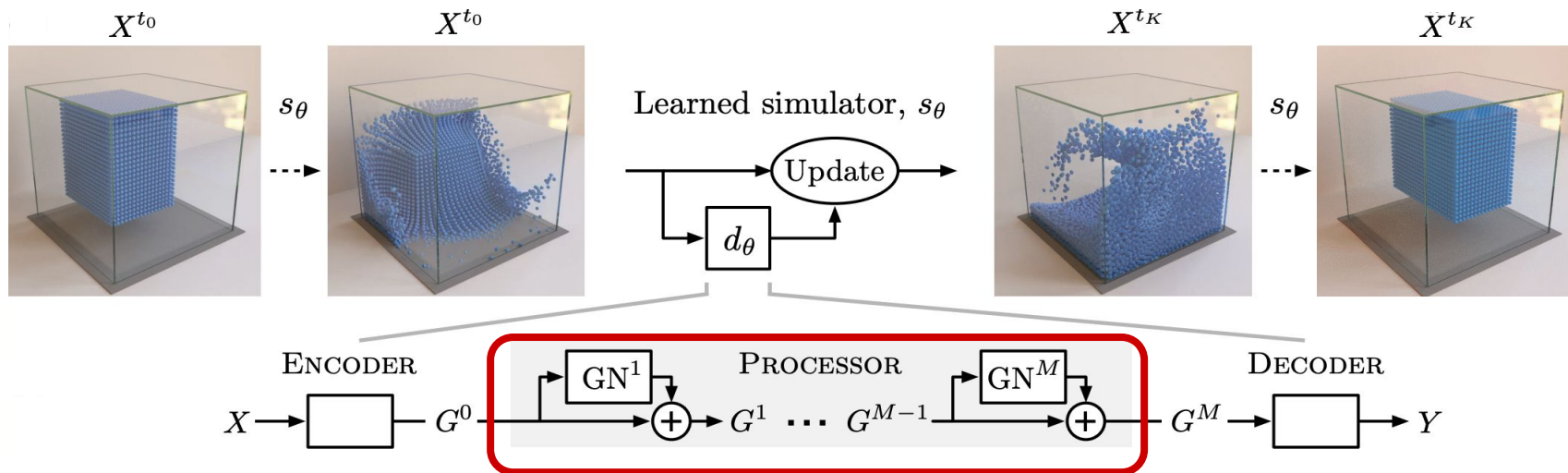
- Edges: Mesh edges (for meshes) or proximity-based (for particles)



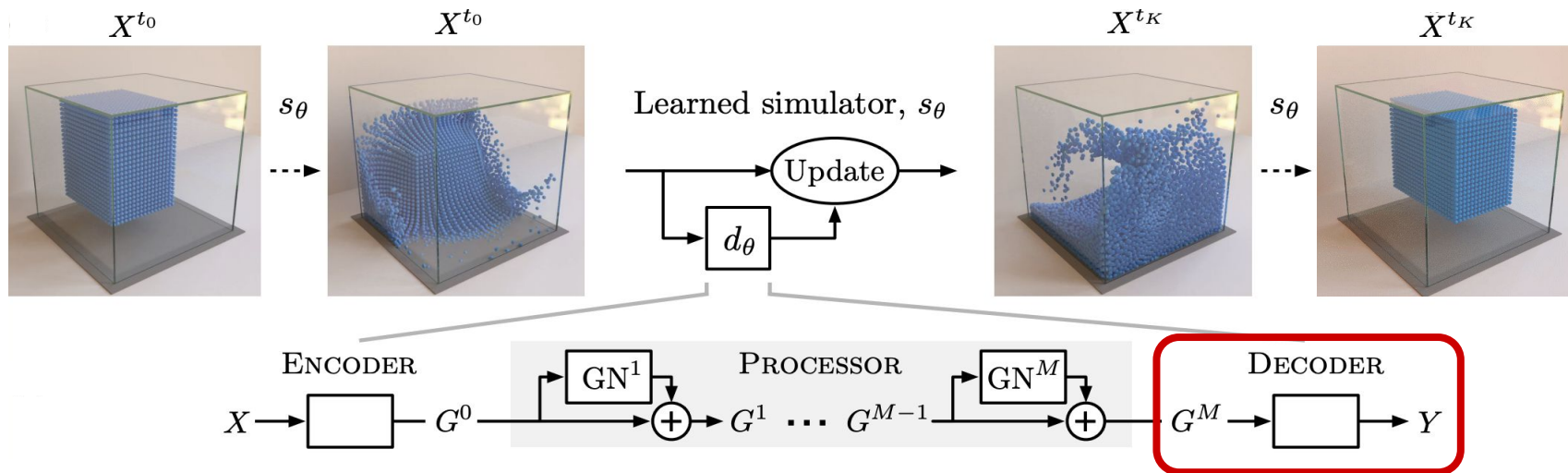
**Spatial
equivariance**



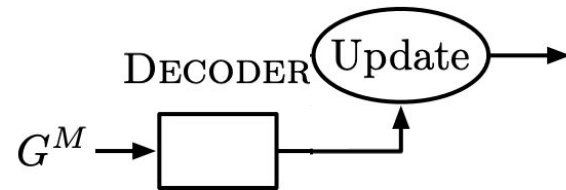
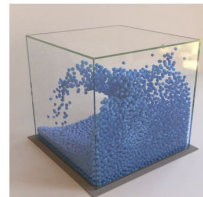
Encoding Graphs



Simulation model



Decoder and update

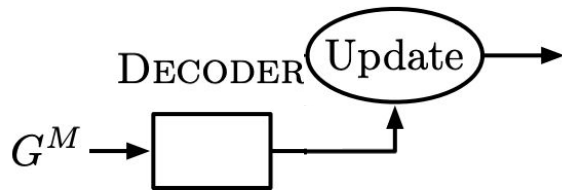
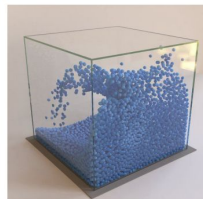


- predict position ?
 - $x^{t+1} = \text{decode}(G)$

x^{t+1}



Decoder and update

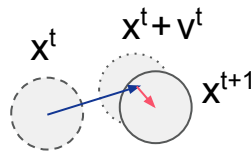


- predict acceleration !

- $v^{t+1} = v^t + \text{decode}(G)$

- $x^{t+1} = x^t + v^{t+1}$

“Acceleration” (Euler integrator with $dt=1$)

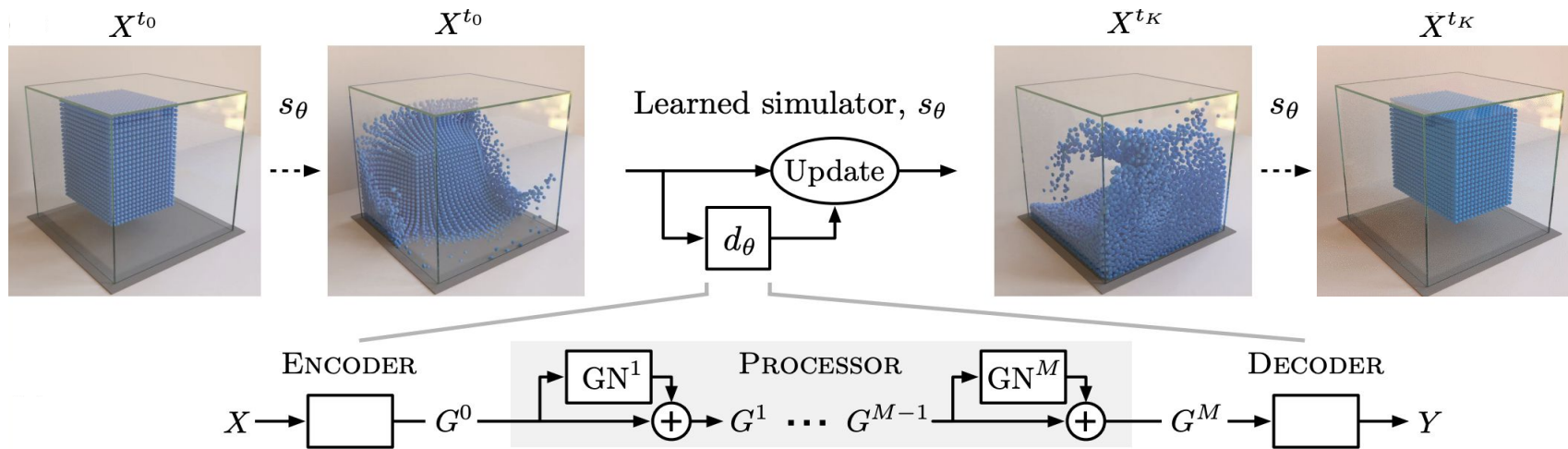


Easy to predict static dynamics

Easy to predict inertial dynamics → Good prior



Simulation model



Simulation model

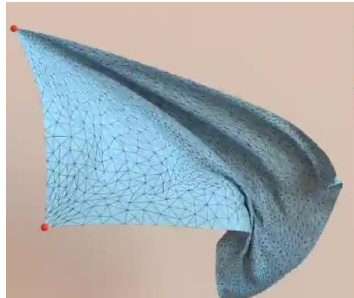
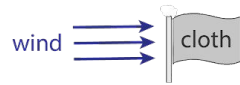
same model, same hyperparameters can simulate many systems

Liquids/granular materials



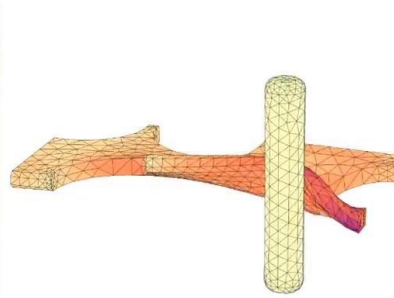
Lagrangian,
particle based

Cloth simulation



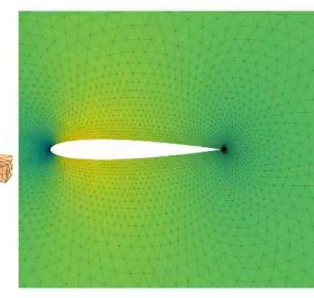
3d
triangular
adaptive

Structural mechanics



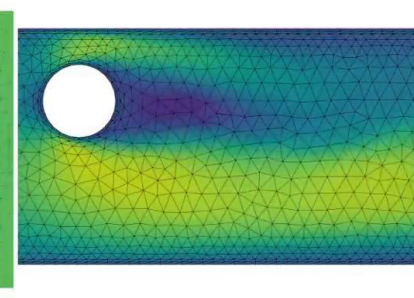
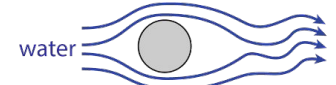
quasi-static
simulation

Compressible
aerodynamics



dynamic
simulation

Incompressible
fluid simulation



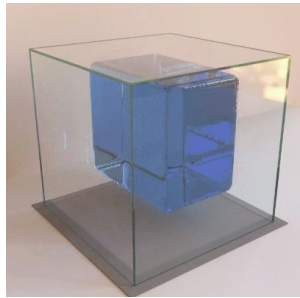
Eulerian,
2d triangular



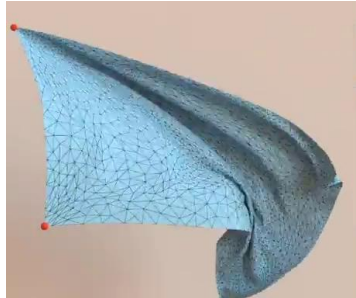
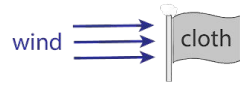
Simulation model

same model, same hyperparameters can simulate many systems

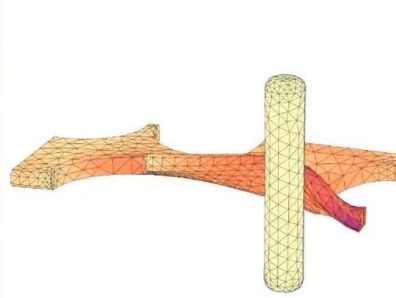
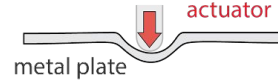
Liquids/granular materials



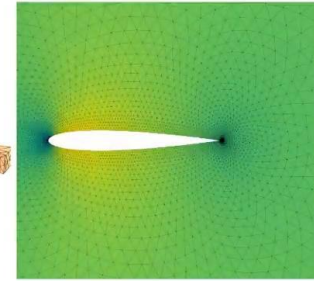
Cloth simulation



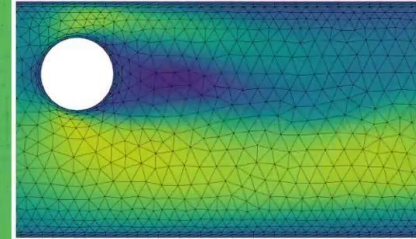
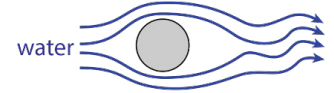
Structural mechanics



Compressible aerodynamics



Incompressible fluid simulation



$$\frac{d \dot{\mathbf{x}}}{dt} = f(\mathbf{x}, \dot{\mathbf{x}})$$

$$\frac{d \mathbf{x}}{dt} = f(\mathbf{x})$$

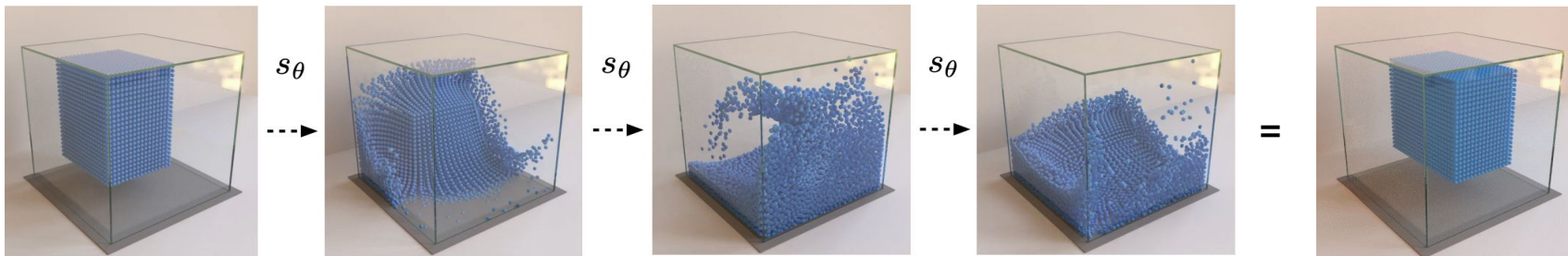
$$\left| \begin{array}{l} \frac{d \dot{\mathbf{v}}}{dt} = f(\mathbf{v}, \rho) \\ \frac{d \dot{\rho}}{dt} = f(\mathbf{v}, \rho) \end{array} \right|$$

$$\frac{d \dot{\mathbf{v}}}{dt} = f(\mathbf{v})$$

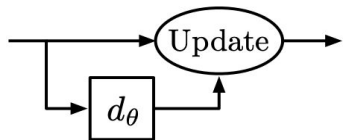


Model rollouts

Train on next-step prediction, unroll for 1000s of steps

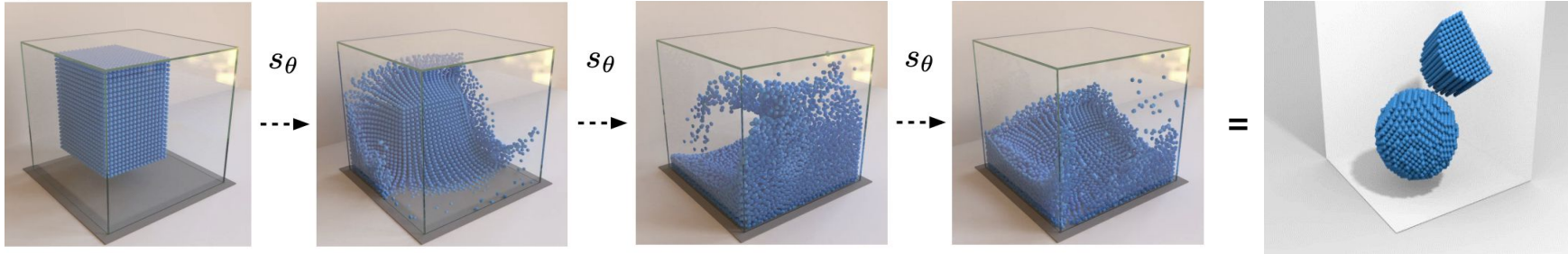


Learned simulator, s_θ



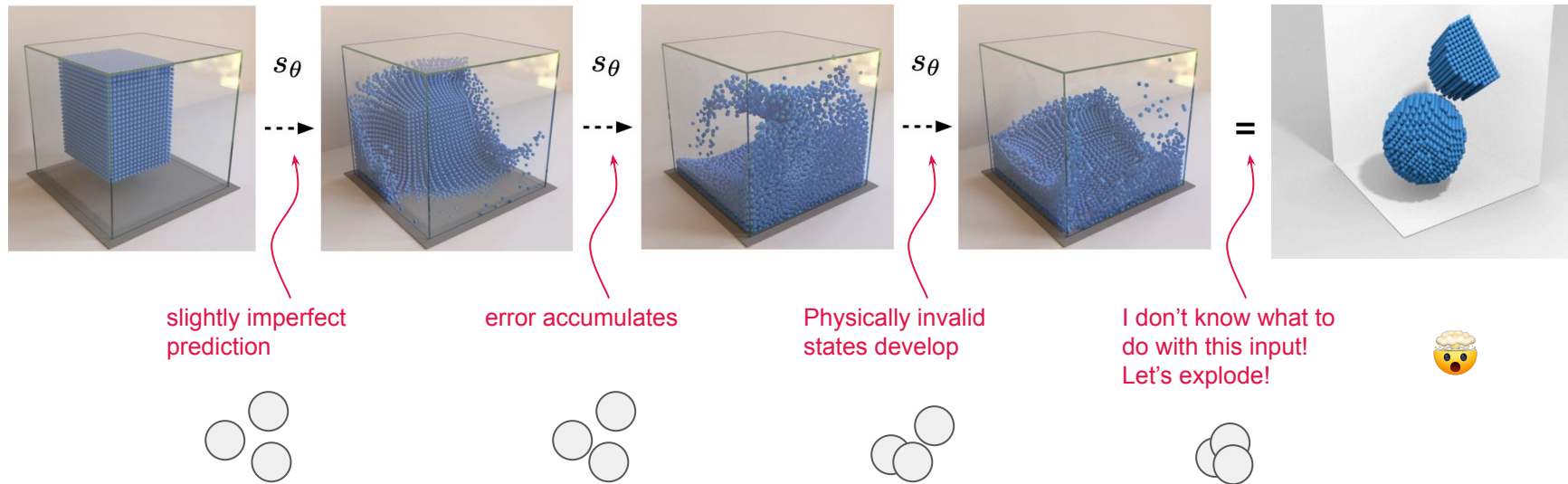
Model rollouts

What usually happens



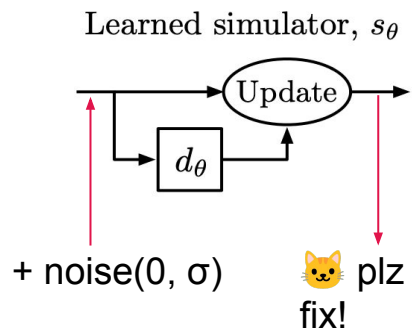
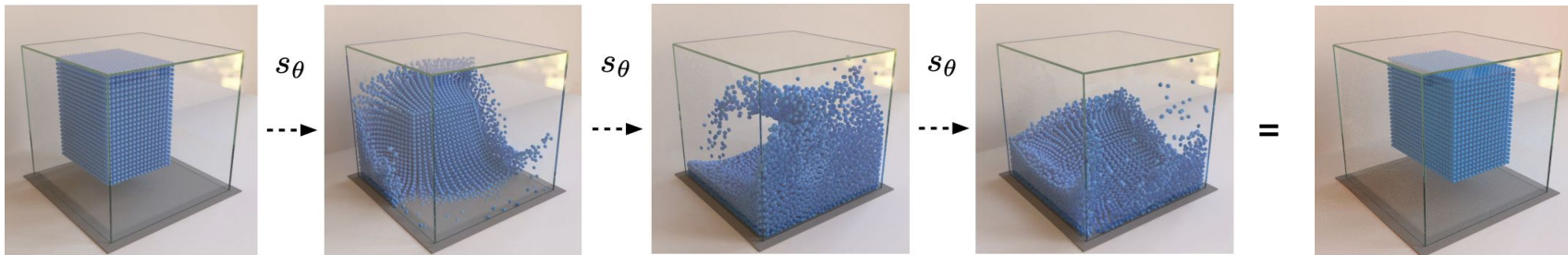
Model rollouts

What usually happens



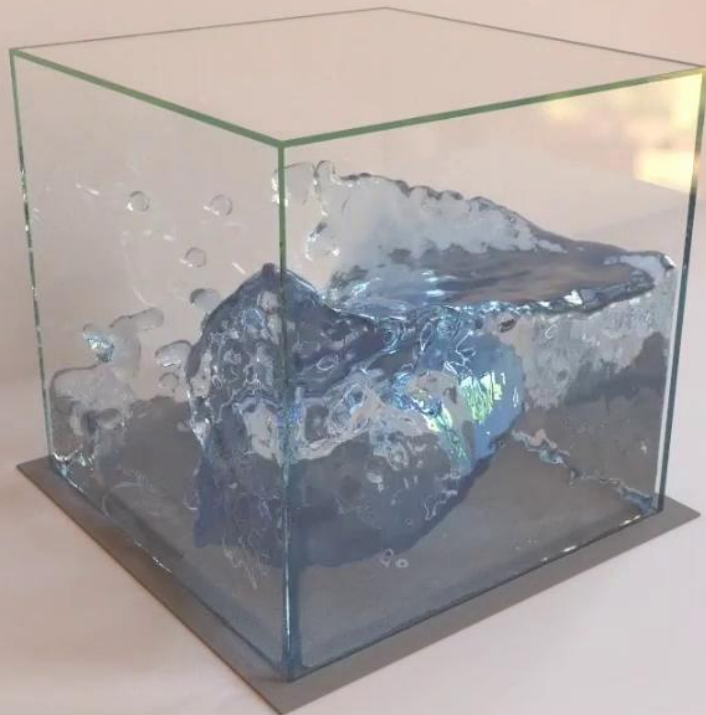
Model rollouts

Training noise



Prediction vs. Ground truth simulation

Ground truth

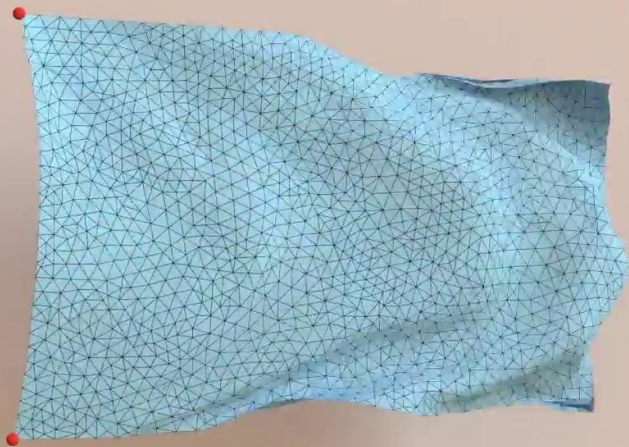


Prediction

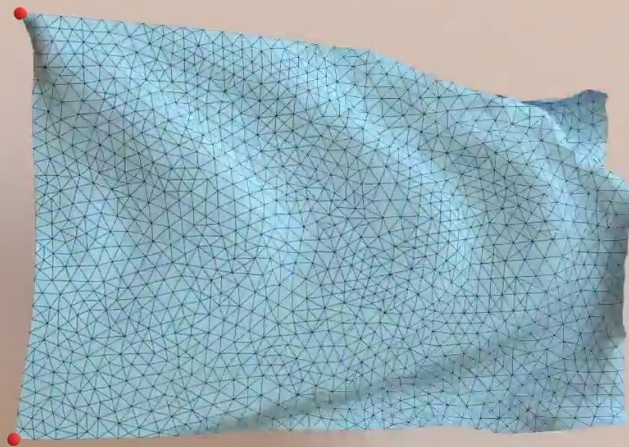


Long-term stability

Ground truth



Prediction



10x

500



Locality, equivariances and generalization

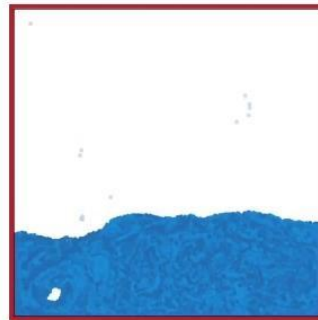
Training

1 x 1 domain
2k particles
600 steps

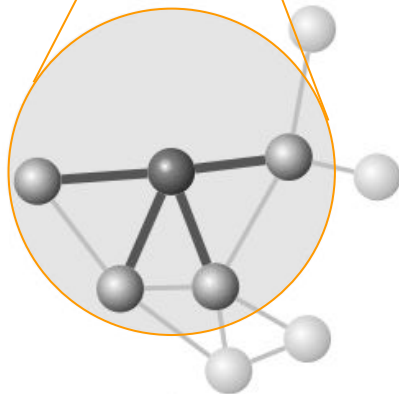
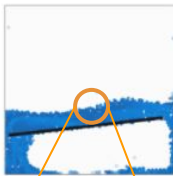
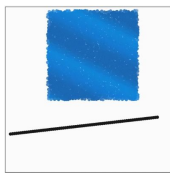


Generalization

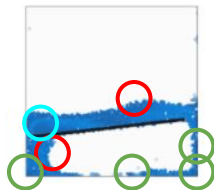
2 x 2 domain, 28k particles, 2500 steps



Locality, Equivariance and Generalization



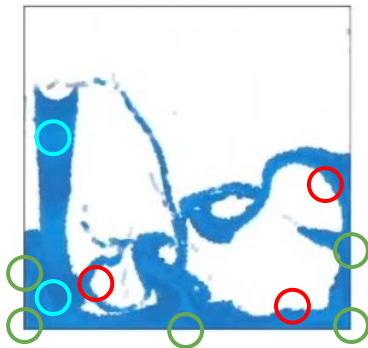
Locality, Equivariance and Generalization



Water surface dynamics

Interaction with box boundaries

Dense blocks of water



Generalization

Ground truth



Prediction
(with learned remeshing)



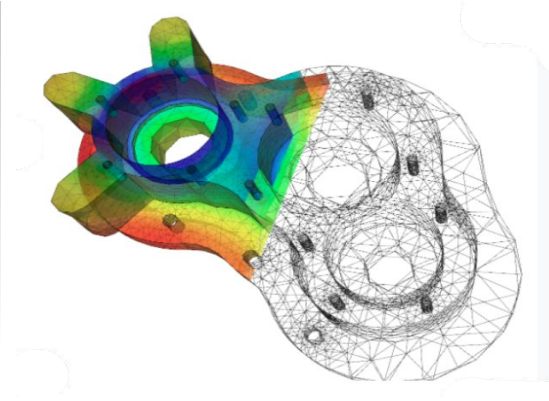
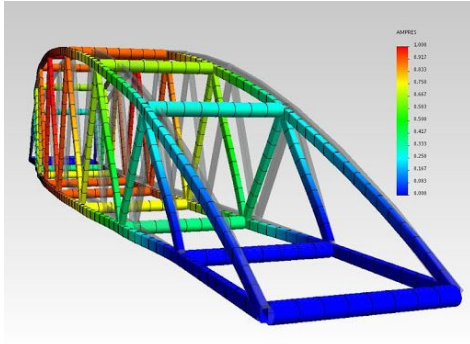
Training:
2k nodes



Testing:
>20k nodes



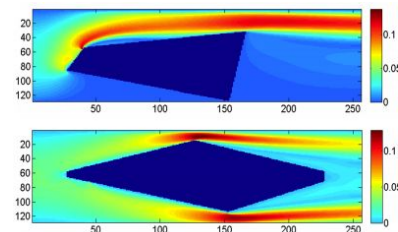
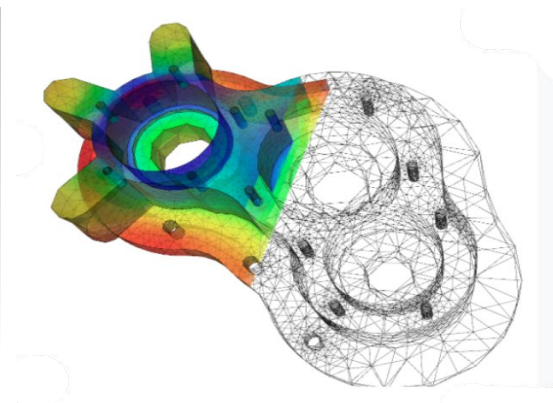
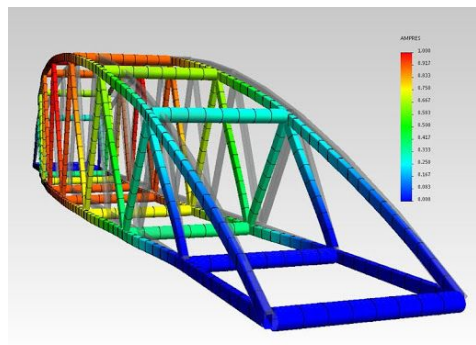
Let's talk about meshes



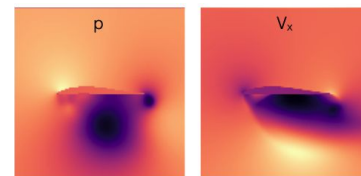
FEM Simulations on meshes



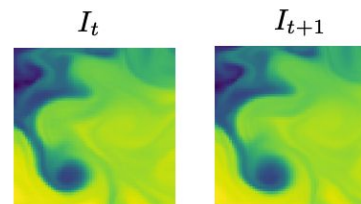
Let's talk about meshes



[Guo et al. 2016]



[Thuerey et al. 2020]



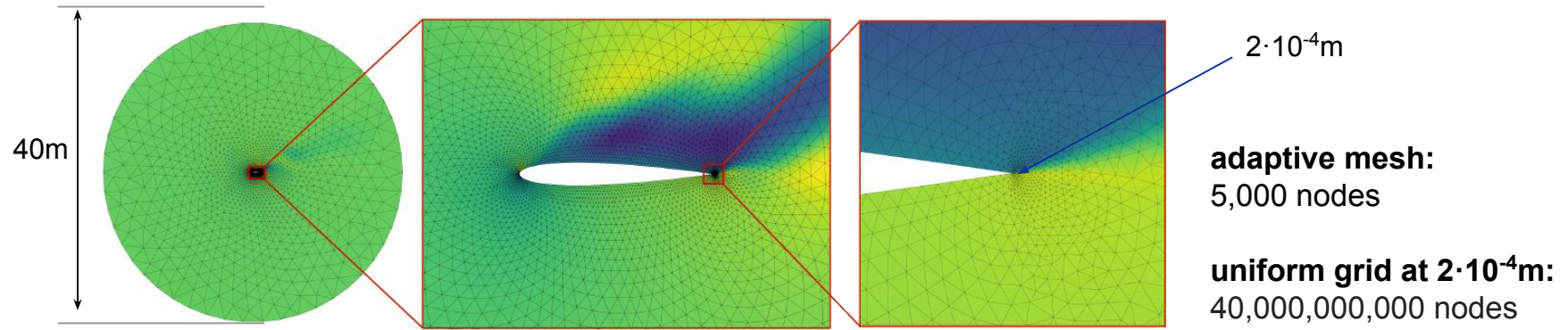
[de Bezenac et al. 2018]

FEM Simulations on meshes

Vast majority of ML/Sim research

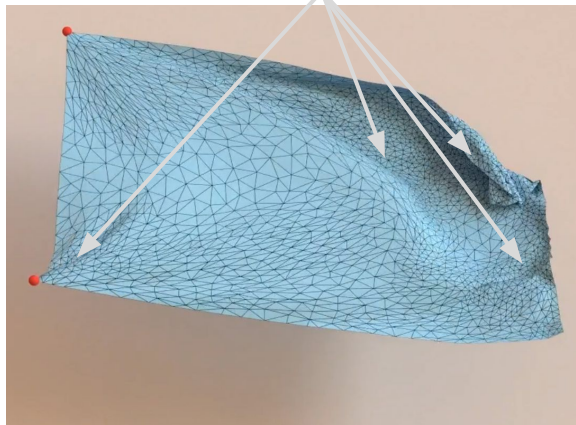


Let's talk about meshes



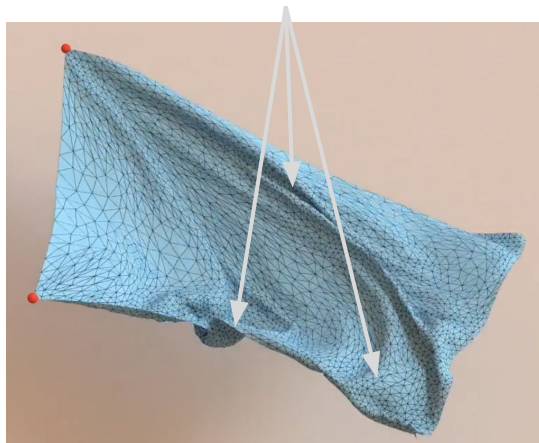
Learned adaptive remeshing

Fine-scale regions at t_i

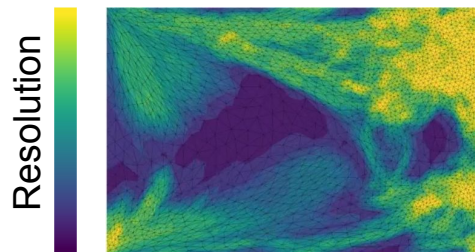


Time
-->

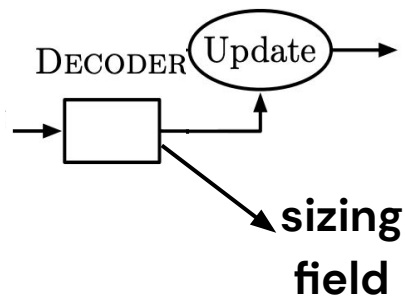
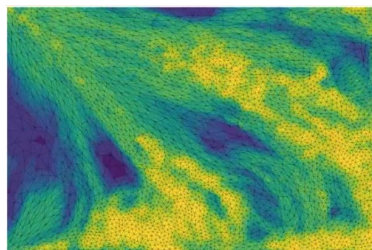
Fine-scale regions at t_j



Sizing field at t_i



Sizing field at t_j

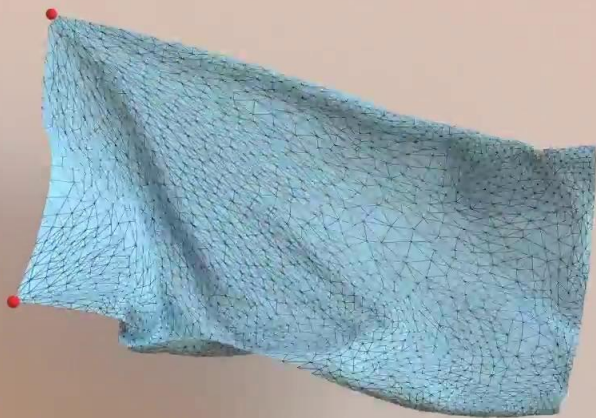


**Predict sizing field
and remesh!**

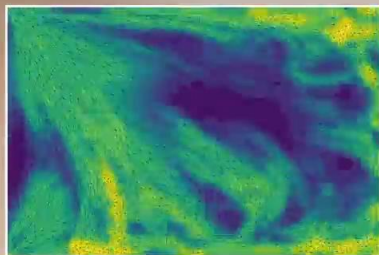


Learned remeshing

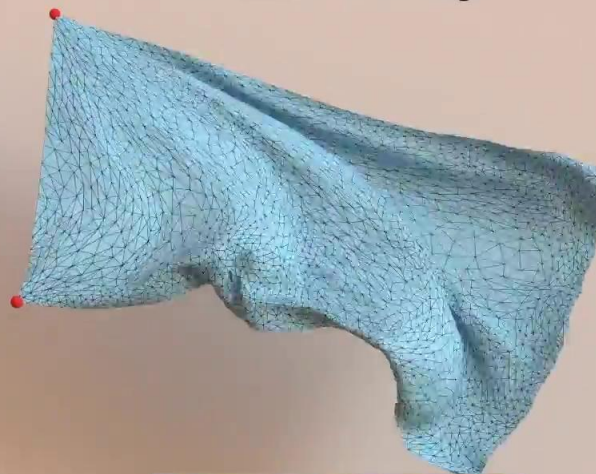
Ground truth



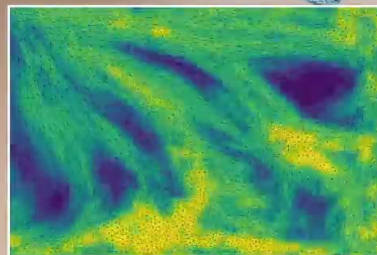
Sizing
field



Prediction
(with learned remeshing)



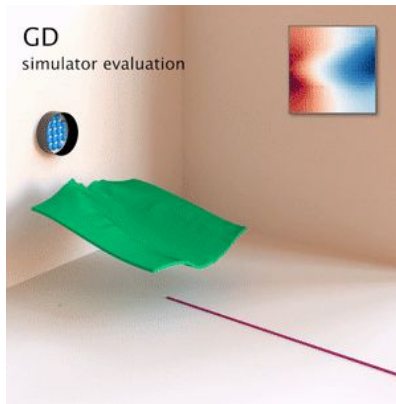
Learned
sizing
field



Up to 300x speed ups compared to solvers!

Dataset	Time per step		Speed-ups
	GNN	Solver	
FLAGSIMPLE	19	4166	200x
FLAGDYNAMIC	837	26199	30x
SPHEREDYNAMIC	140	1610	11x
DEFORMINGPLATE	33	2893	100x
CYLINDERFLOW	23	820	40x
AIRFOIL	38	11015	300x



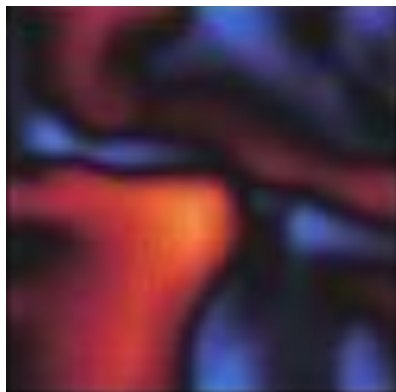


Physical Design using Differentiable Learned Simulators **arXiv (ICML 2022 submission)**

arxiv.org/pdf/2202.00728.pdf
sites.google.com/view/learning-to-simulate

Constraint-based graph network simulator **arXiv (ICML 2022 submission)**

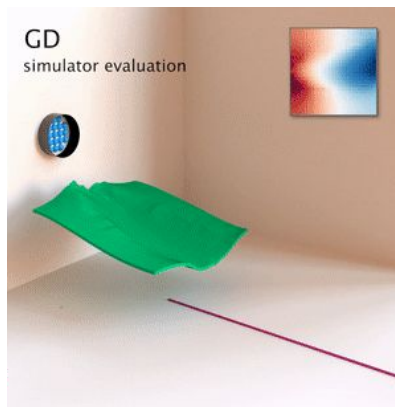
arxiv.org/pdf/2112.09161.pdf
sites.google.com/view/constraint-based-simulator



Learned Coarse Models for Efficient Turbulence Simulation **ICLR 2022**

arxiv.org/pdf/2112.15275.pdf
sites.google.com/view/learning-to-simulate





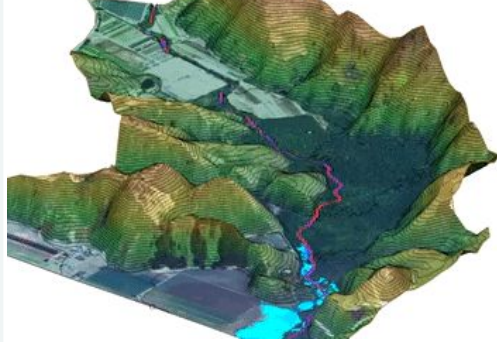
Physical Design using Differentiable Learned Simulators arXiv (ICML 2022 submission)

arxiv.org/pdf/2202.00728.pdf
sites.google.com/view/learning-to-simulate

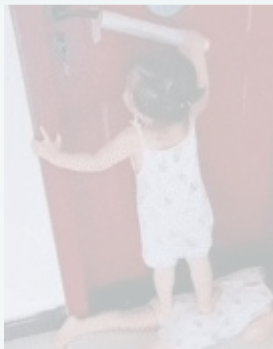
Kelsey R. Allen^{*1} Tatiana Lopez-Guevara^{*1} Kimberly Stachenfeld^{*1} Alvaro Sanchez-Gonzalez¹
Peter Battaglia¹ Jessica Hamrick¹ Tobias Pfaff¹



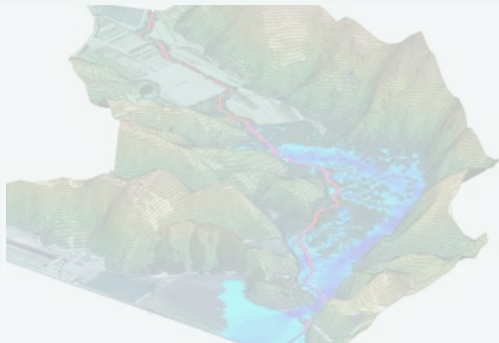
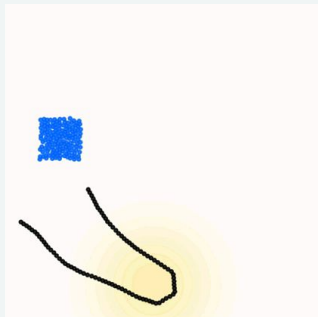
Designing tools is a hallmark of intelligence ...



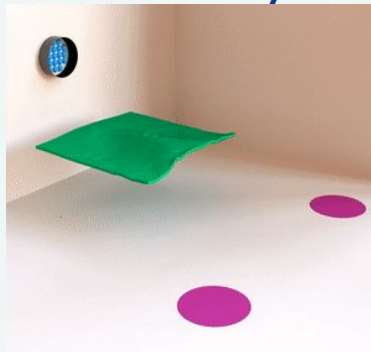
Can we automatically design these structures within a general-purpose framework?



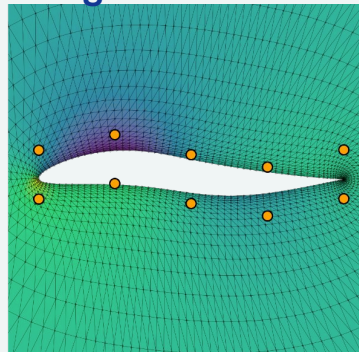
Simple tool creation



Scalability

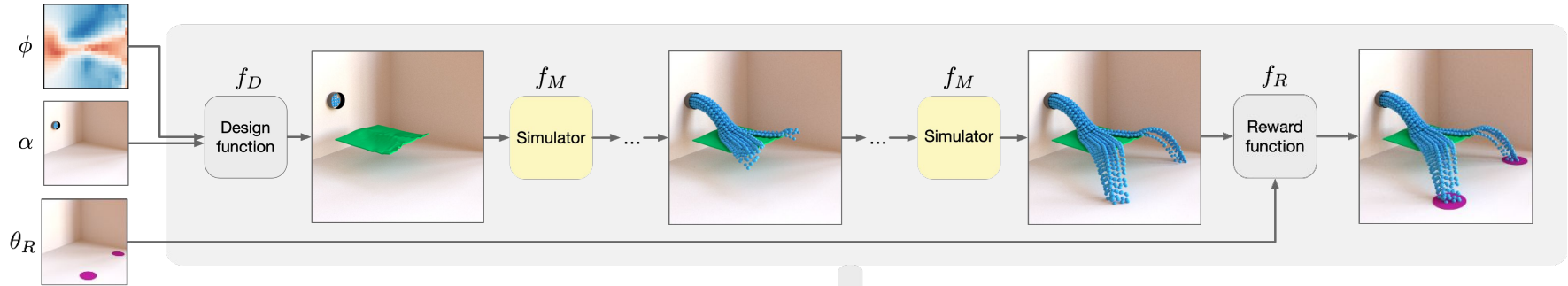


High Precision

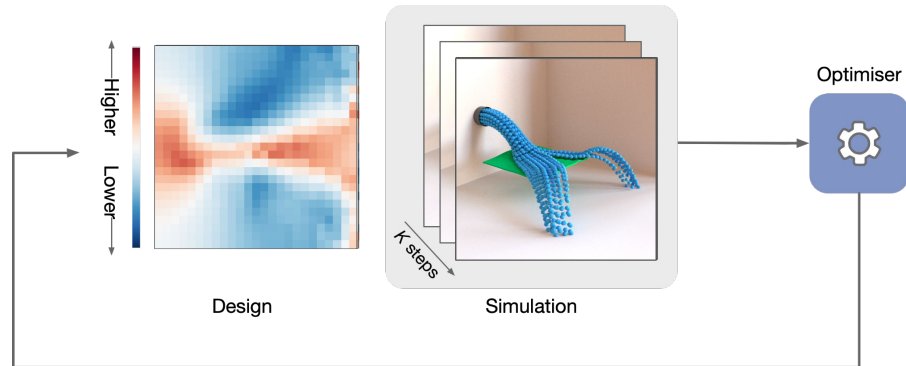


General Inverse Design Framework

Inner loop: forward model rollout



Outer loop: design optimization process

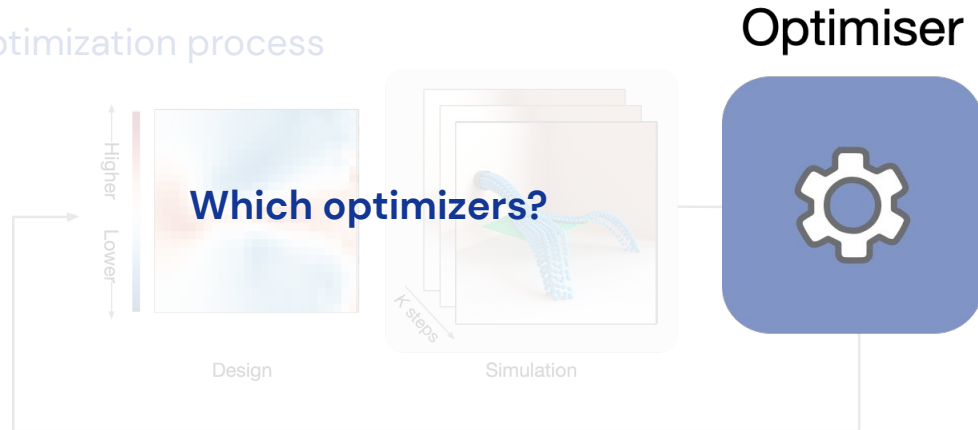


General Inverse Design Framework

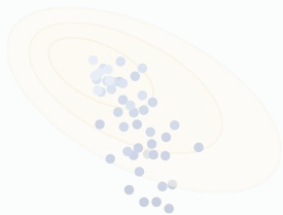
Inner loop: forward model rollout f_M



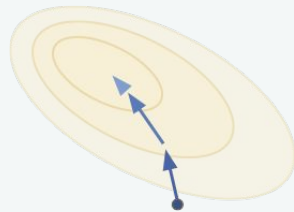
Outer loop: design optimization process



Can we leverage Pre-trained GNN models for Inverse Design?



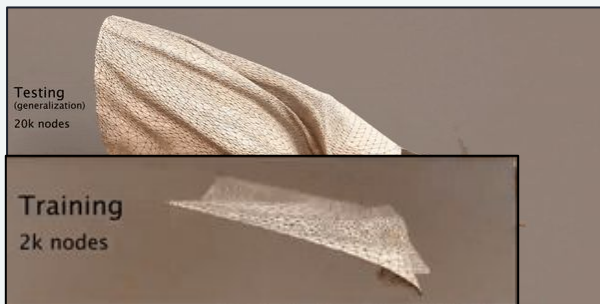
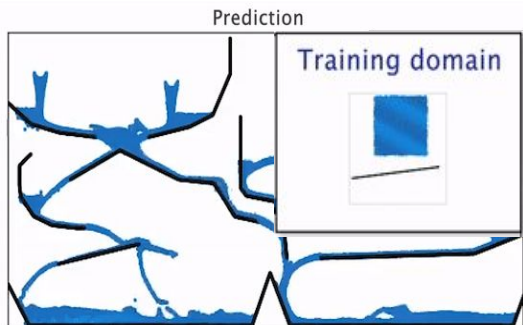
Sampling-based
black-box models



Gradient-based with
learned models



Gradient-based with
analytical models




Can we use a **GNN** based model
pre-trained on physical
dynamics for **inverse design**?




Discovered designs

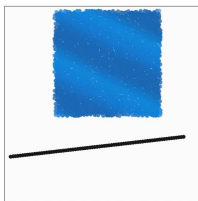
2D Fluid Tools

100 – 1000 particles,
16 – 36 design dimensions

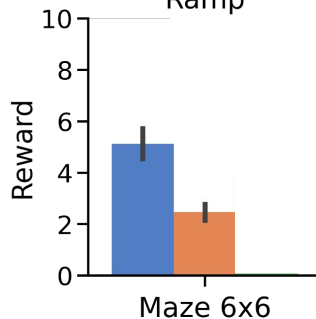
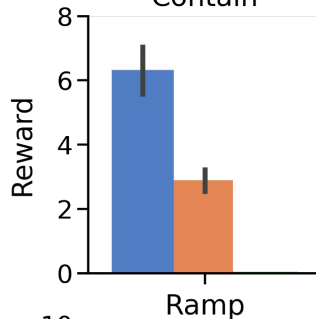
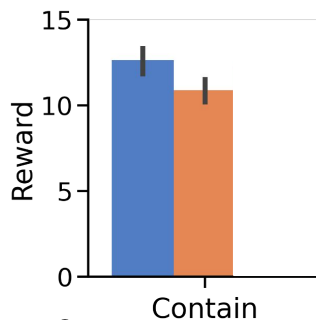
 Gradient
based*
(ours)

 Sampling
based

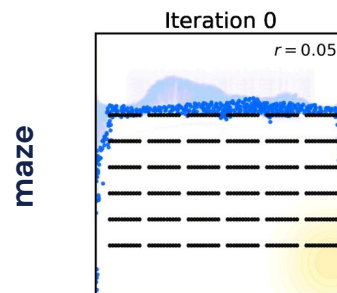
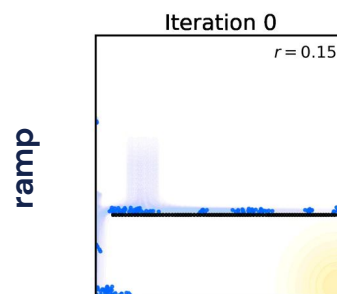
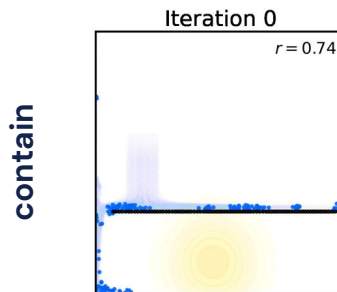
Training
distribution



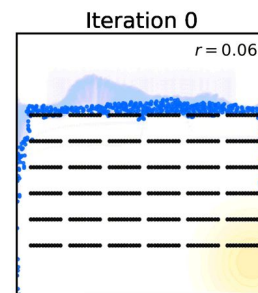
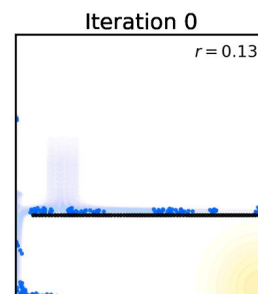
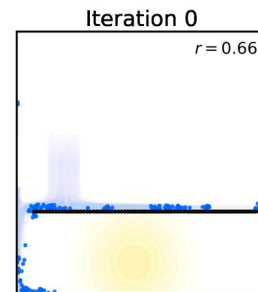
Overall reward
(higher is better)



Gradient-based
optimization

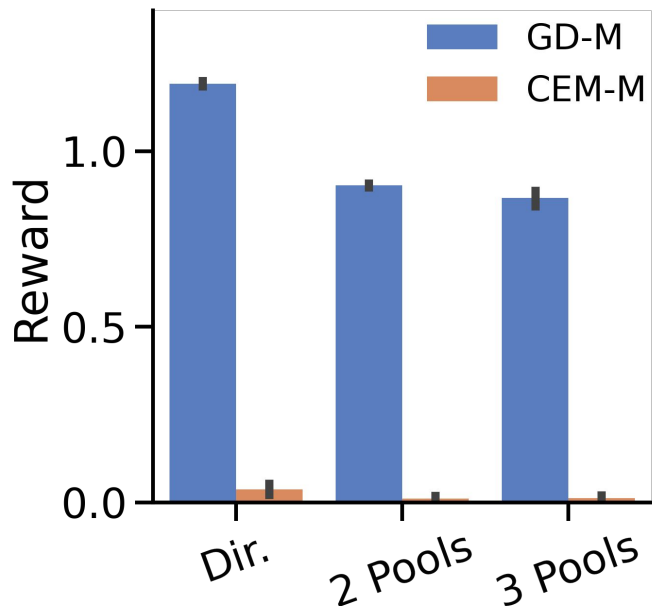


Sampling-based
optimization

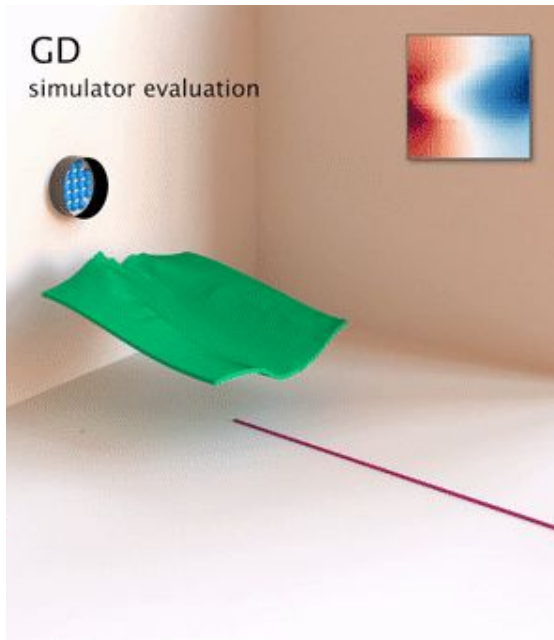


Discovered designs - 3D WaterCourse

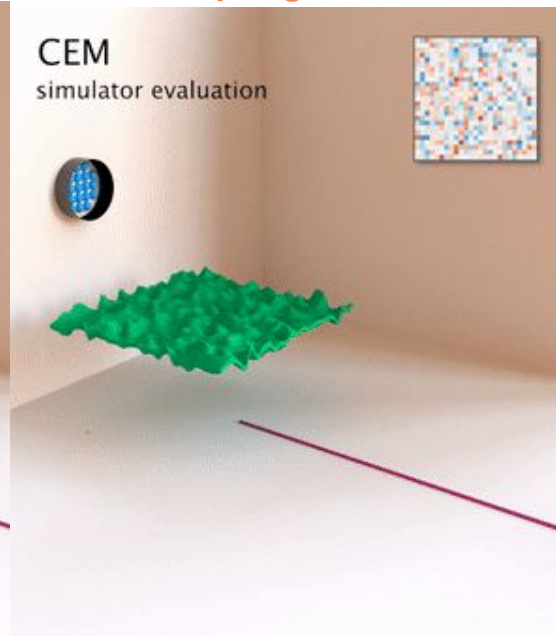
2k - 4k particles, 625 design dimensions



Gradient-based



Sampling-based

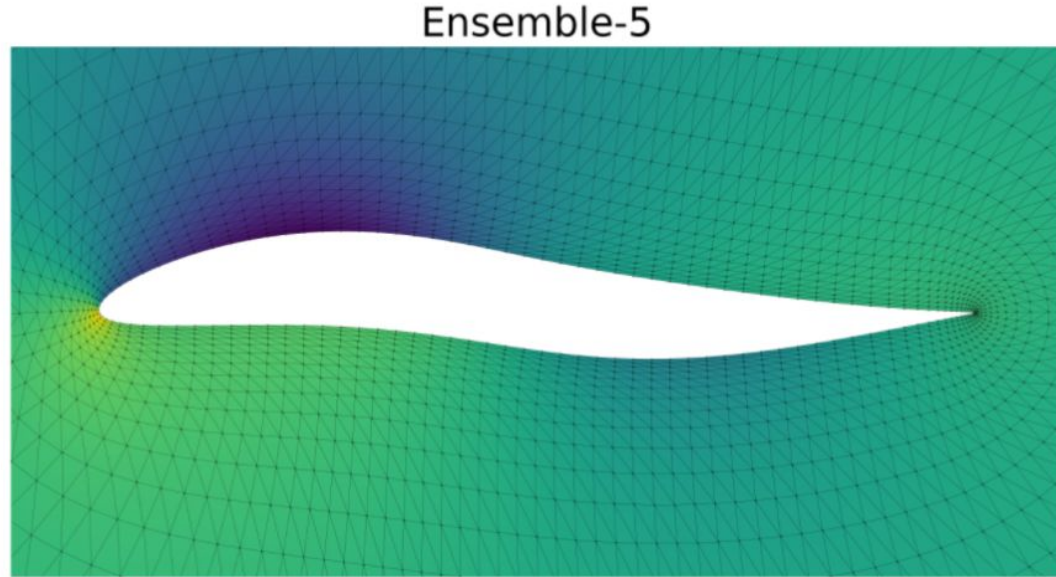


- Gradient descent through a learned simulator (GD-M) outperforms a sampling-based approach (CEM-M) by 1-2 orders of magnitude.



Discovered designs - Airfoil

4158 node mesh, 10 design dimensions



- Airfoil designs converges fairly quickly in about **150 steps**
- Very **similar drag coefficients** achieved with the true simulator and learned models





Constraint-based graph network simulator
arXiv (ICML 2022 submission)

arxiv.org/pdf/2112.09161.pdf

sites.google.com/view/constraint-based-simulator

Yulia Rubanova^{* 1} Alvaro Sanchez-Gonzalez^{* 1} Tobias Pfaff¹ Peter Battaglia¹

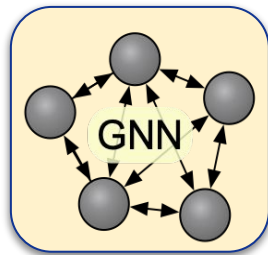


Explicit GNN simulator*

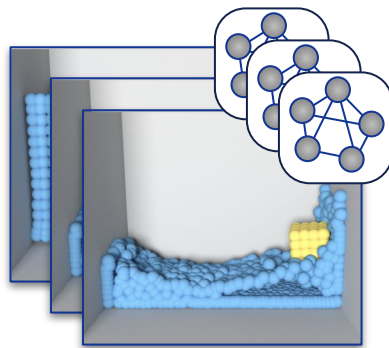
$$\mathbf{X}_{\leq t}$$



The next state
is predicted
explicitly

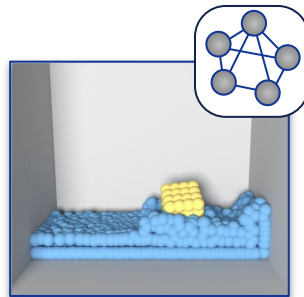


$$\hat{\mathbf{X}}_{t+1}$$



Nodes = {pos, vel} per particle

Edges = particle interactions



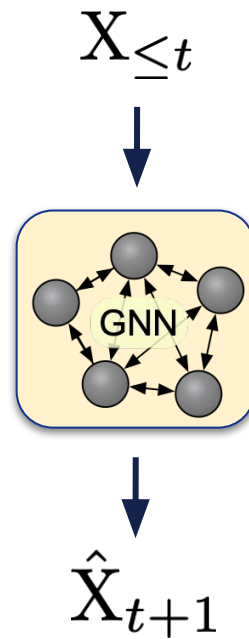
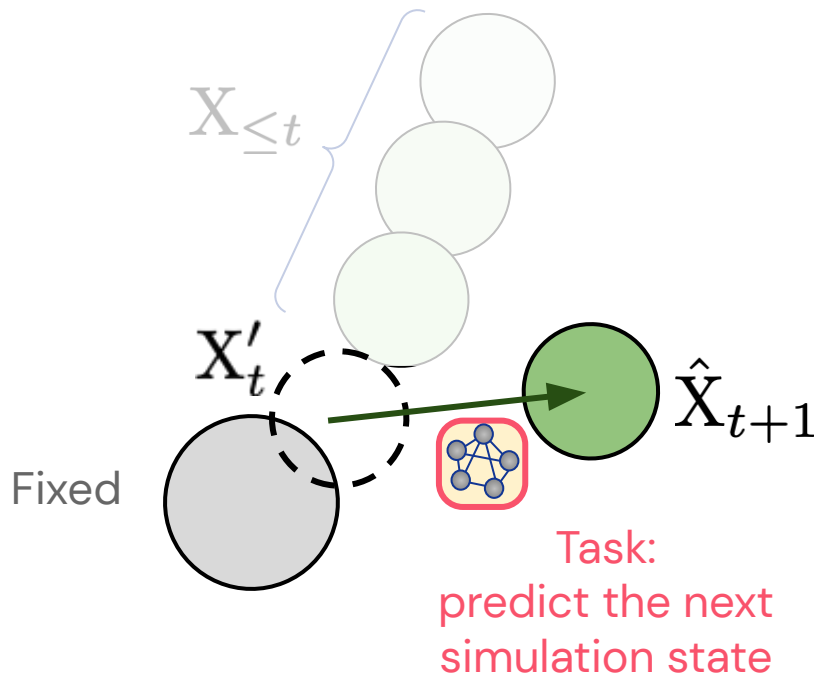
Task:
predict {pos, vel} at
the next time step

*Sanchez-Gonzalez, Godwin, Pfaff et al., ICML 2020

*Pfaff, Fortunato, Sanchez-Gonzalez et al., ICLR 2021

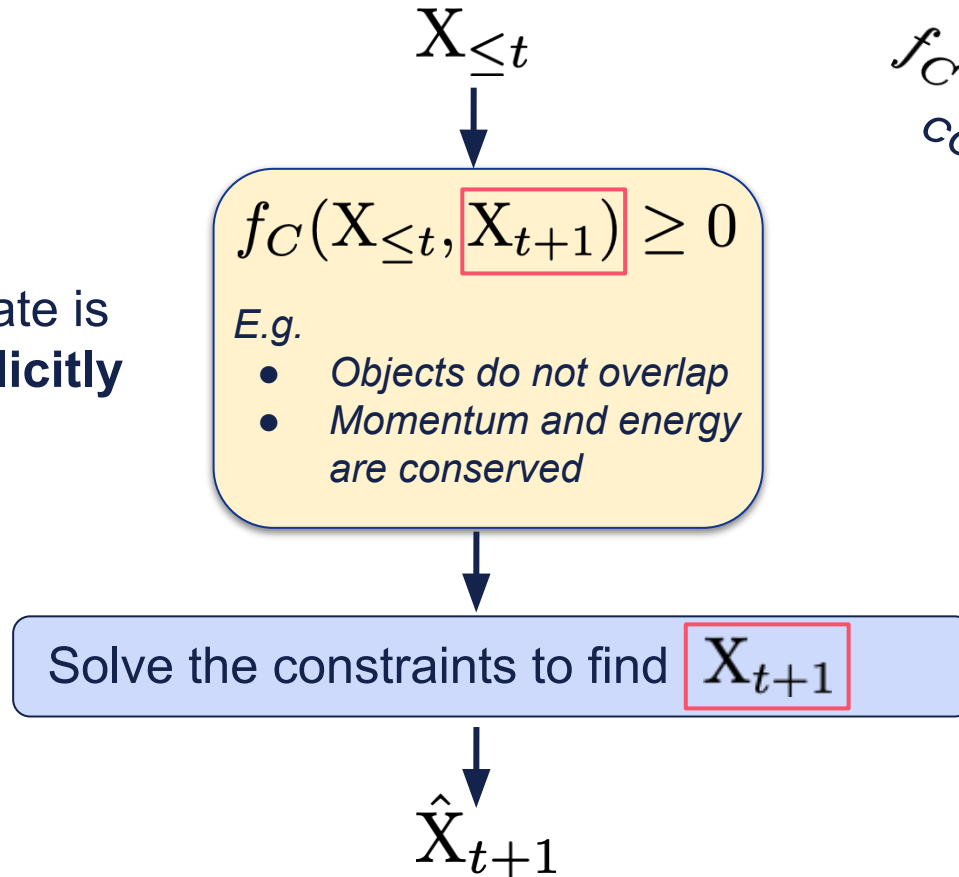


How do we use explicit GNN simulator?



Many physical simulators don't work like that!

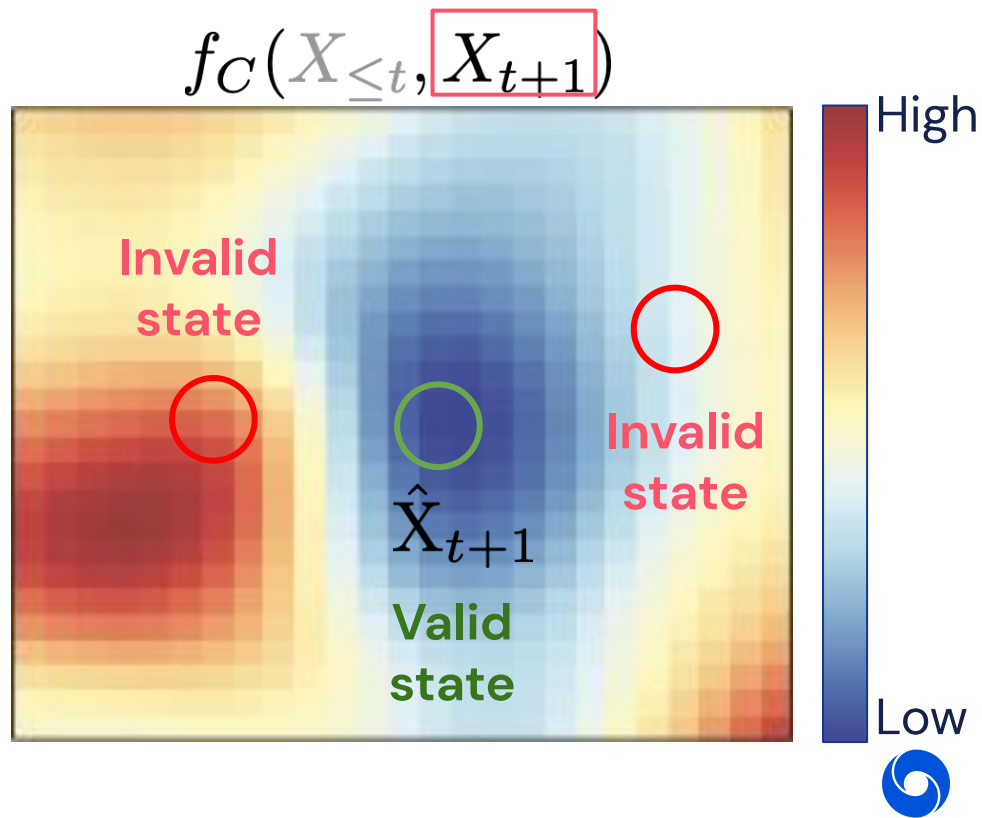
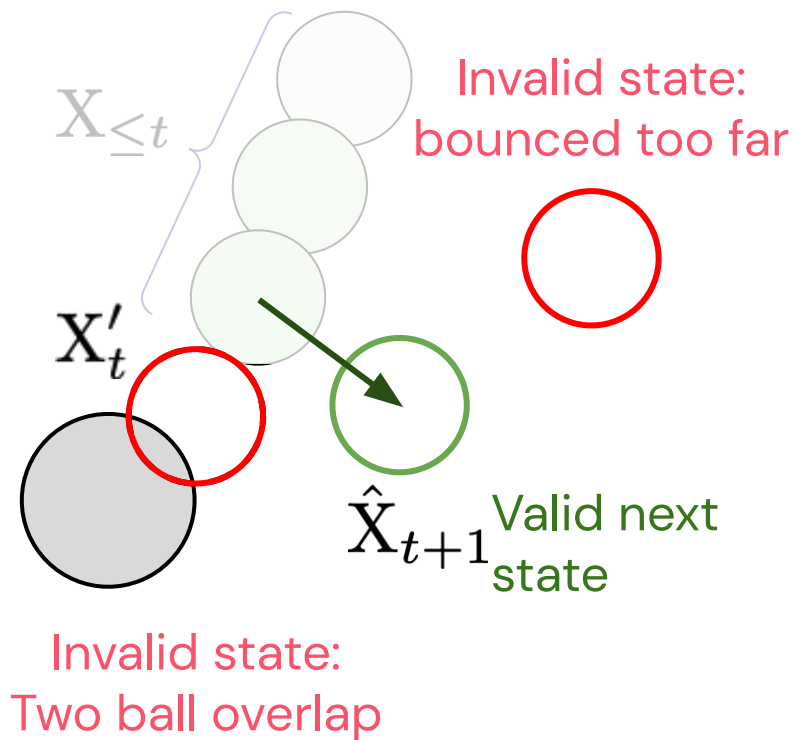
The next state is defined **implicitly**



f_C : Is proposed X_{t+1} consistent with $X_{\leq t}$?

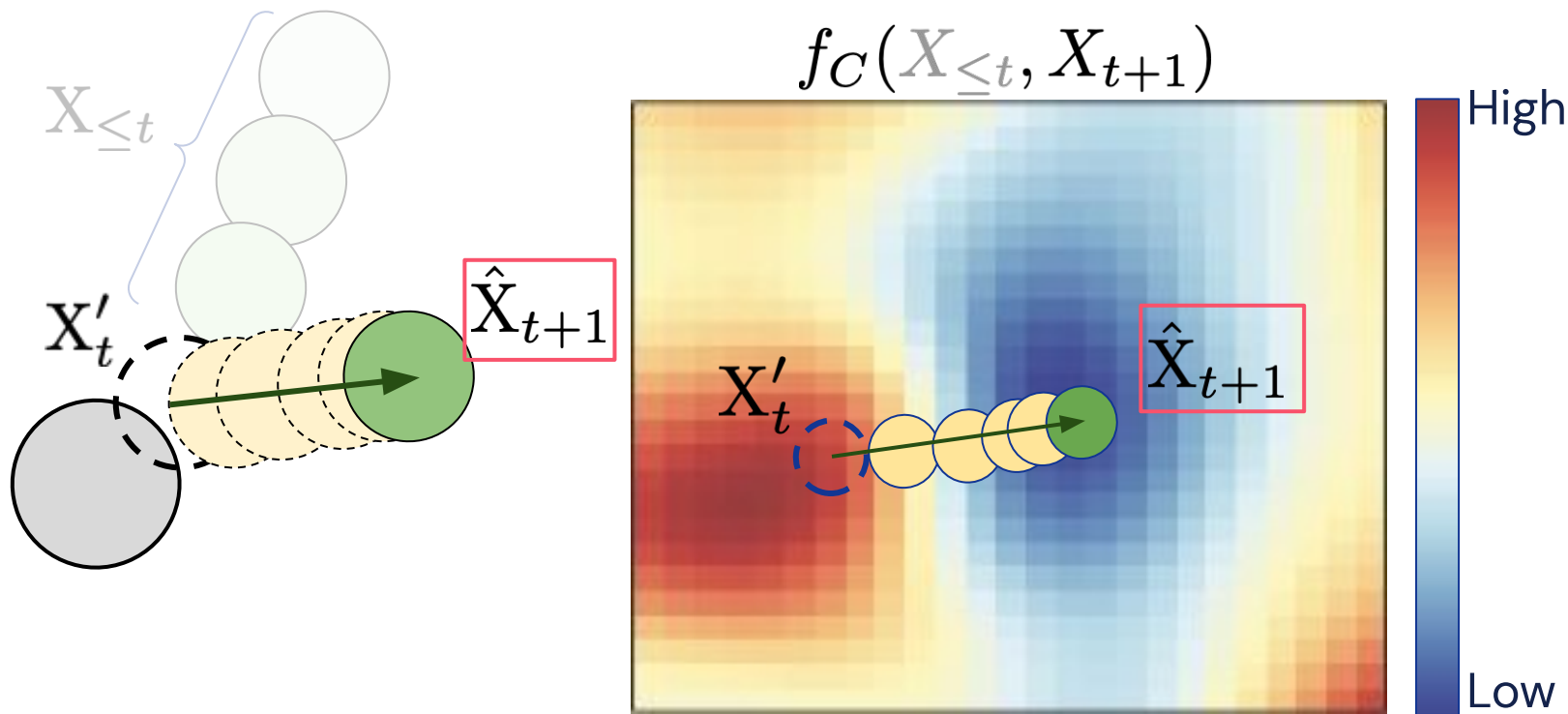


Constraint function

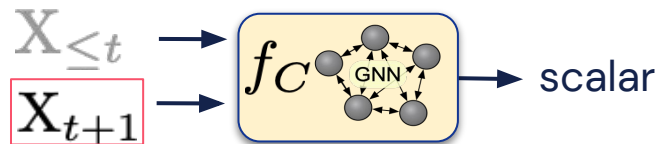


Forward pass

Run gradient descent on f_C to find \hat{X}_{t+1}



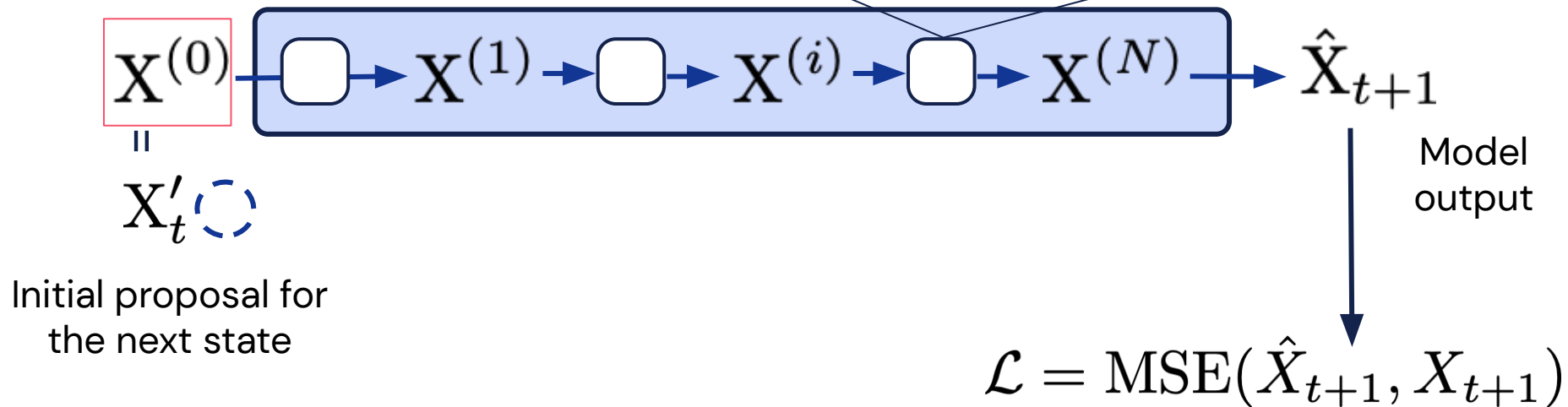
Forward pass



Gradient descent step

$$X^{(i+1)} = X^{(i)} - \lambda \nabla f_C$$

Diagram illustrating the gradient descent step. The update equation $X^{(i+1)} = X^{(i)} - \lambda \nabla f_C$ is shown, where f_C is the function from the forward pass. A callout box shows the GNN structure inside f_C .

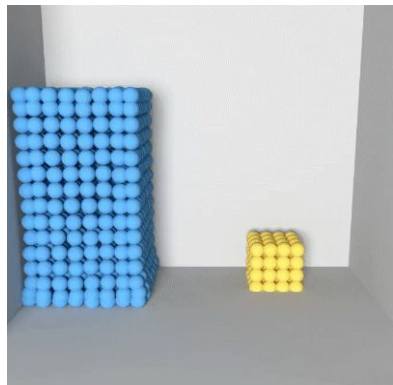
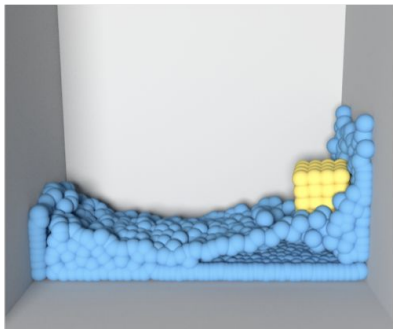


Trainable end-to-end: Don't need constraint labels!

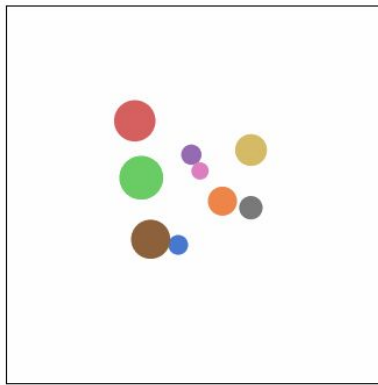


Domains

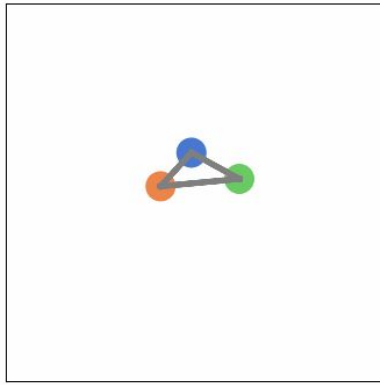
BoxBath



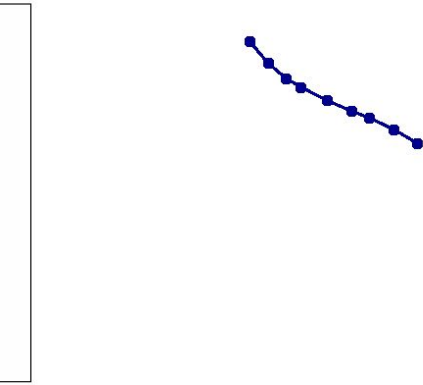
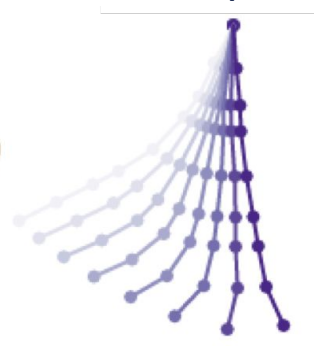
Bouncing Balls



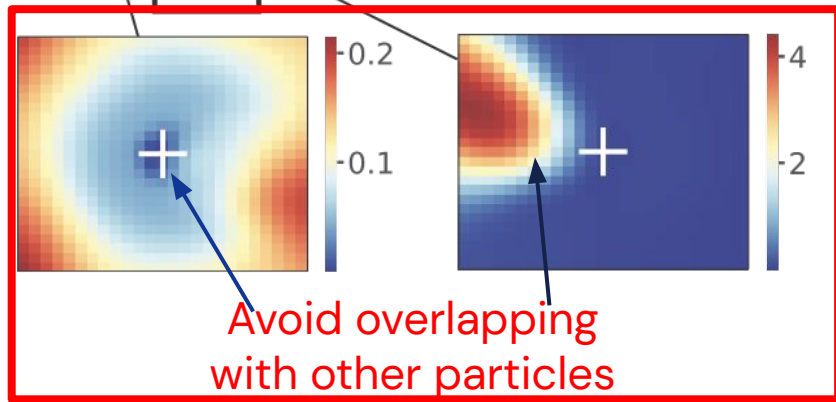
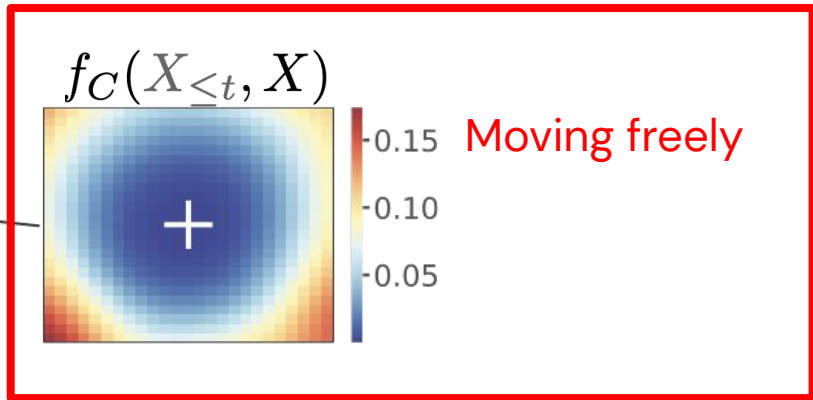
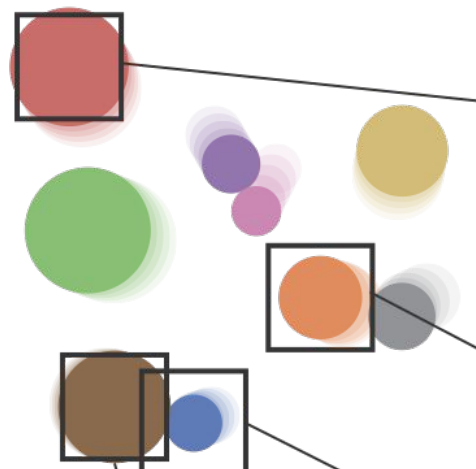
Bouncing Rigid



Rope



Interpreting the constraints



+ Ground-truth next state



Combining constraint functions

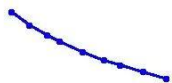
At test time optimize $f_C(\mathbf{X}_{\leq t}, \mathbf{X}_{t+1}) + f_{\text{obstacle}}(\mathbf{X}_{t+1})$

a learned constraint

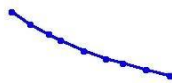
a user-defined constraint
(e.g. new obstacle)

No collisions were ever
observed at training time!

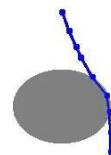
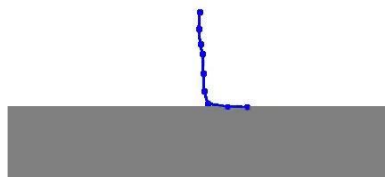
Ground Truth



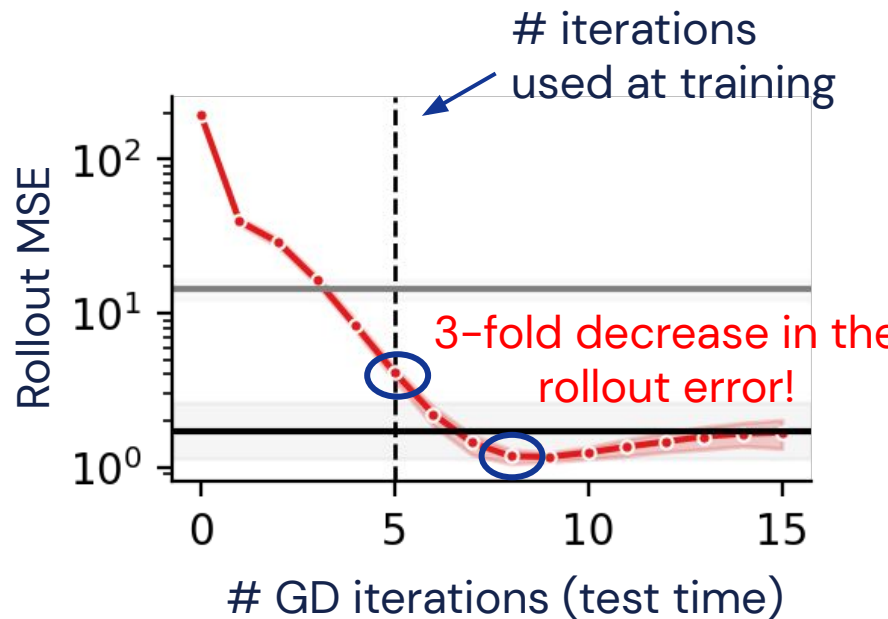
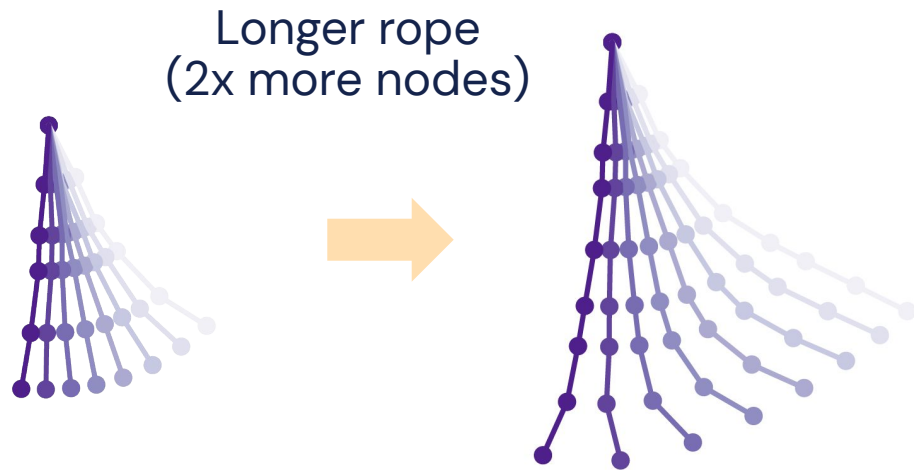
C-GNS



C-GNS with additional spatial constraints

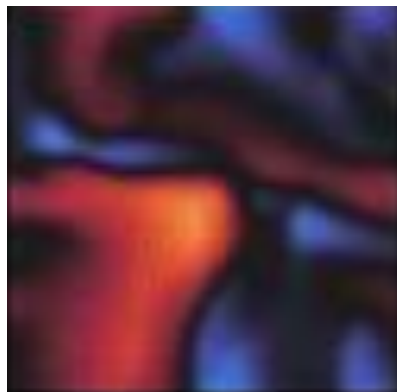


Improving accuracy on larger systems



Better generalization than explicit models!





Learned Coarse Models for Efficient Turbulence Simulation **ICLR 2022**

arxiv.org/pdf/2112.15275.pdf

sites.google.com/view/learning-to-simulate

**Kimberly Stachenfeld,¹ Drummond B. Fielding,² Dmitrii Kochkov,³
Miles Cranmer,⁴ Tobias Pfaff,¹ Jonathan Godwin,¹ Can Cui,²
Shirley Ho,² Peter Battaglia,¹ Alvaro Sanchez-Gonzalez¹**



Turbulence Simulation

Engineering



Forecasting



Science



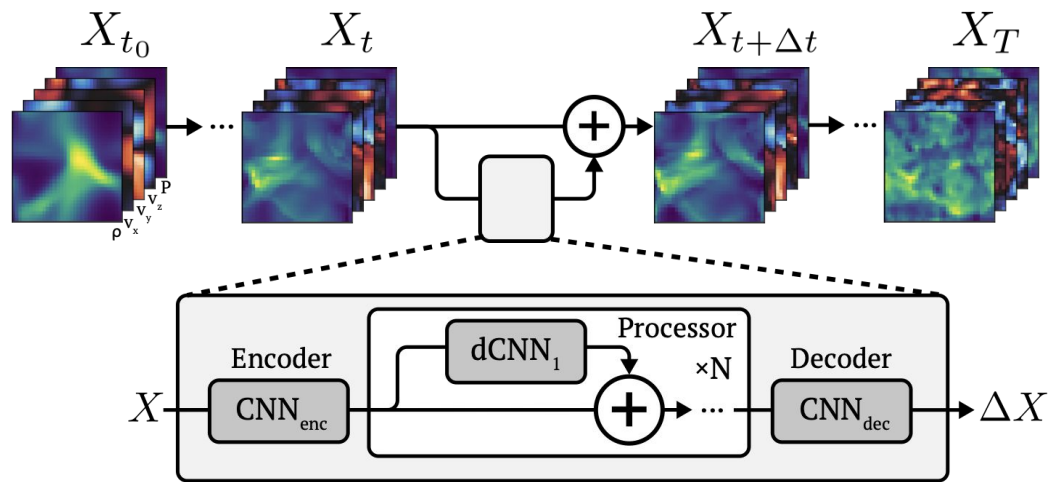
Classical numerical solvers are powerful but computationally expensive

Can fully-learned simulators capture complex, chaotic turbulence accurately at faster speeds?



Our Approach

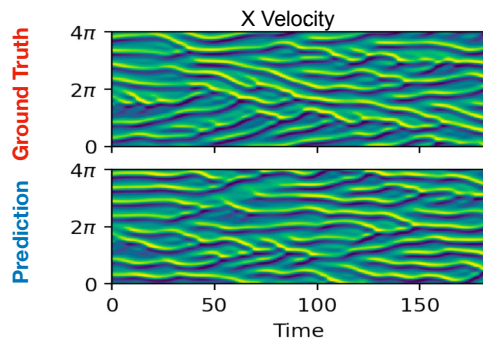
1. Use classical physics solvers to produce **high-resolution trajectories**
2. Downsample these trajectories in **space** and **time** to produce training data
3. Train a neural network to do **next-step prediction** on **low-resolution** frames



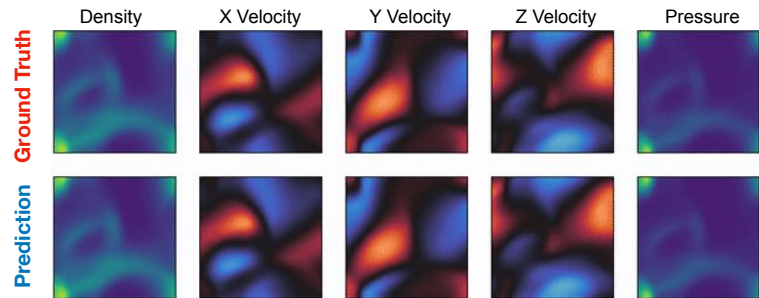
Domain Generality

One model \rightarrow 4 different domains

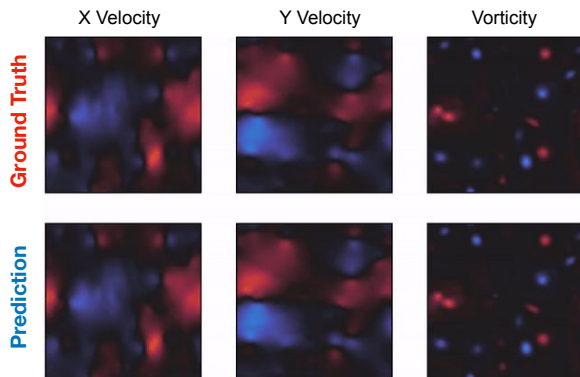
1D Kuramoto–Sivashinsky (KS) Equation



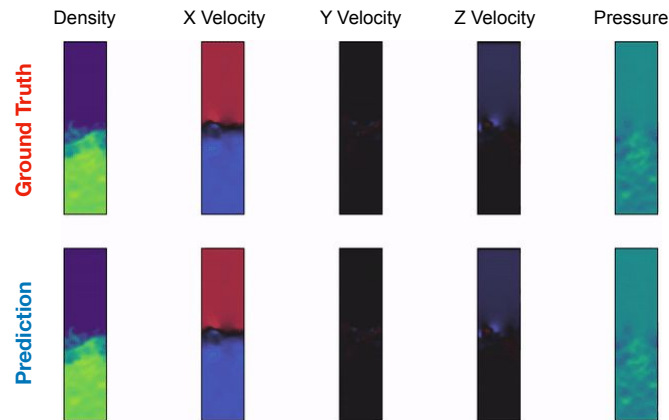
3D Uniform Compressible Decaying Turbulence



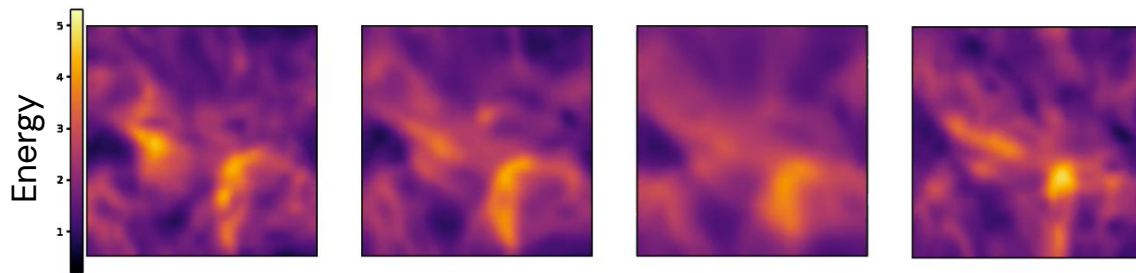
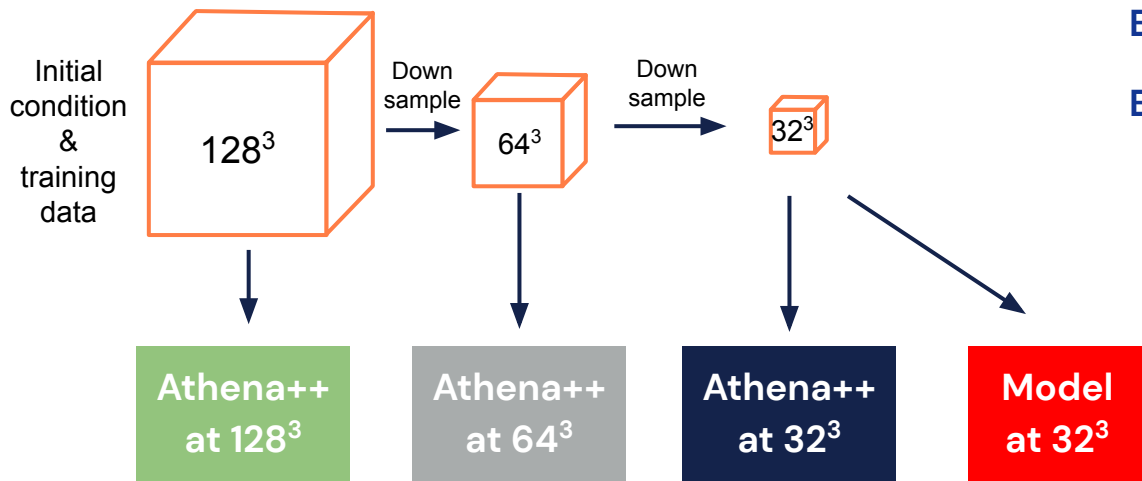
2D Incompressible Turbulence



3D Mixing Layer Turbulence with Radiative Cooling



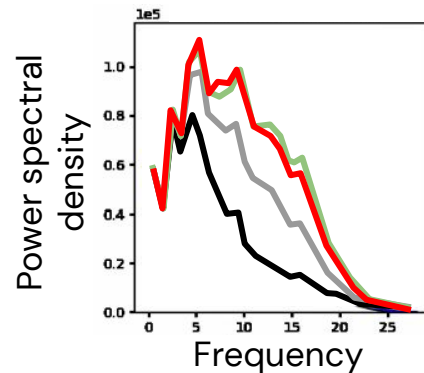
Spatial Coarsening



Better RMSE than Athena at 32^3



Better spectrum than Athena at 64^3

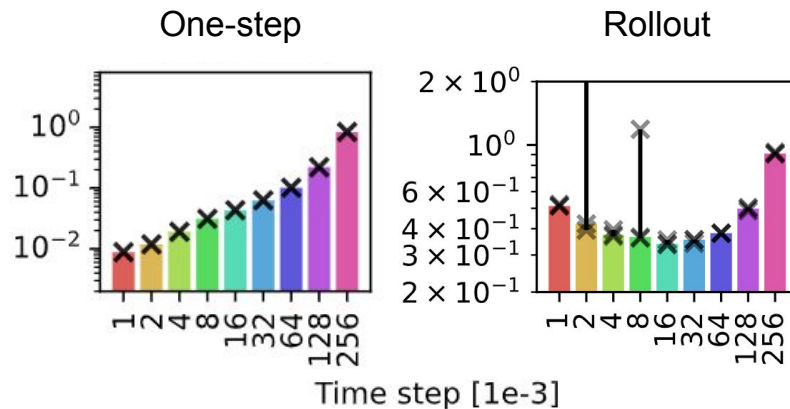


Temporal Coarsening

Learned simulators can be trained on larger timesteps.



Energy RMS error
(trained with noise)



Ground Truth

Learned model timestep

1x

2x

4x

8x

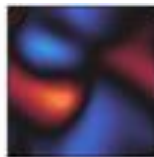
16x

32x

64x

128x

256x



Running time

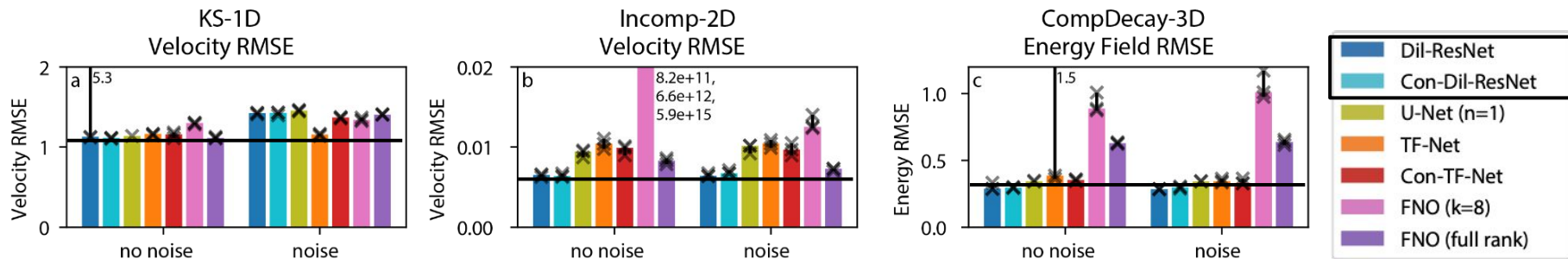


- Athena++
 - Scales $O(\text{resolution}^4)$
 - CPU only
- Learned model:
 - **Up to 1000x faster** than Athena at 128

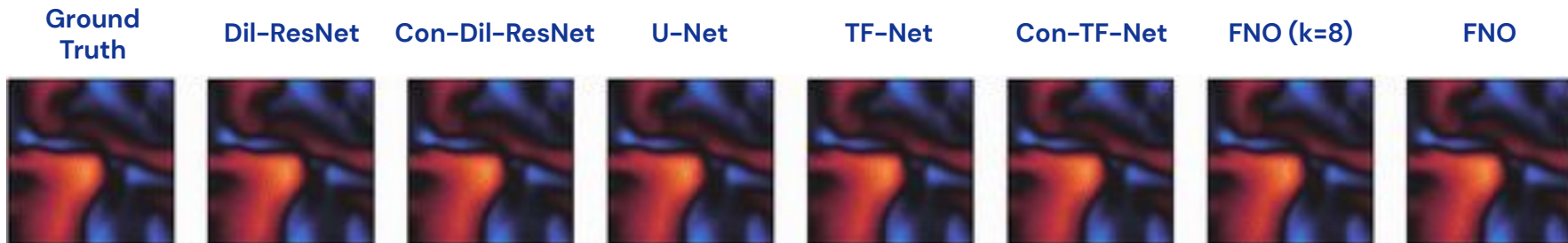
Simulator	Time (s)
Athena++ 32^3	~4
Athena++ 64^3	~60
Athena++ 128^3	~1000
Model $128^3 \rightarrow 32^3$	~20-30
Model $128^3 \rightarrow 32^3$ (GPU)	~1



Learned Model Comparison



Our models quantitatively outperform other, more specialized, parameterized models



However, most learned models do qualitatively pretty well



Constraints satisfaction as function of time

1D KS
total momentum
conesevation



2D Incompressible
velocity field
divergence



3D turbulence
mass, energy,
momentum
conservation



Generalization out of the training distribution

Generalization to longer trajectories:



Does not generalize to more developed turbulence

Generalization to different initial conditions:



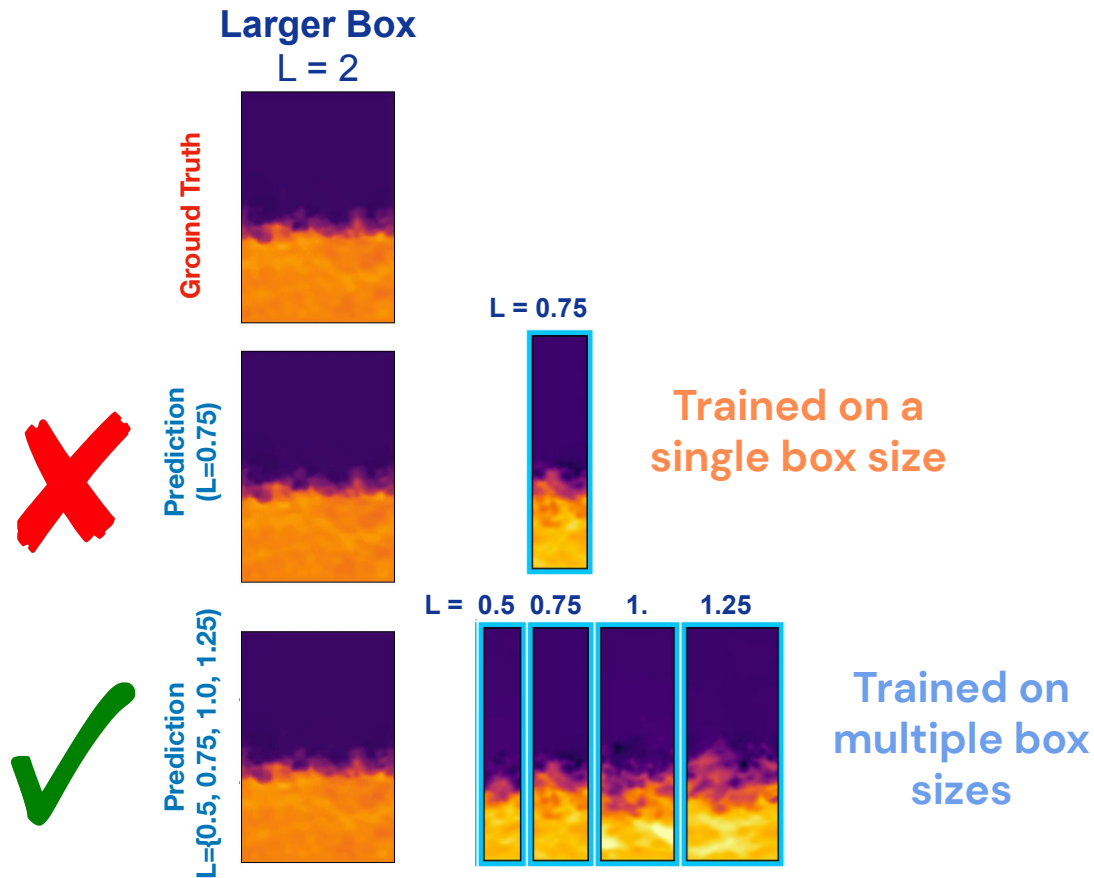
Generalizes to higher solenoidal components



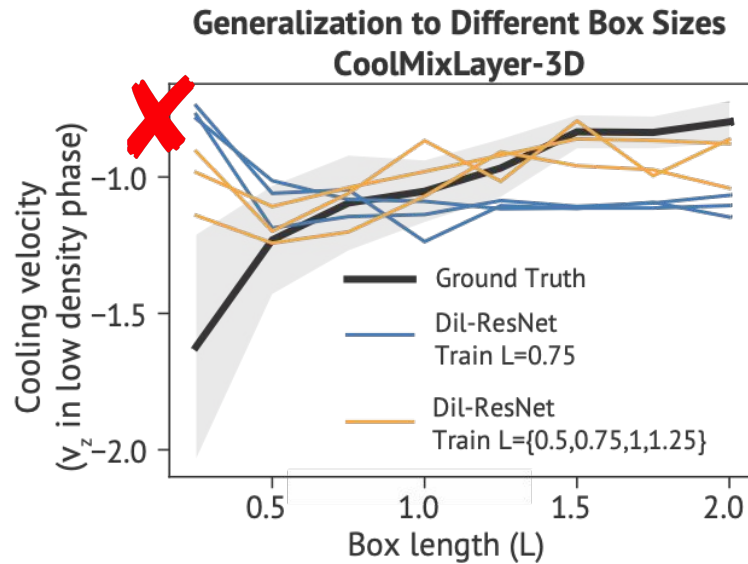
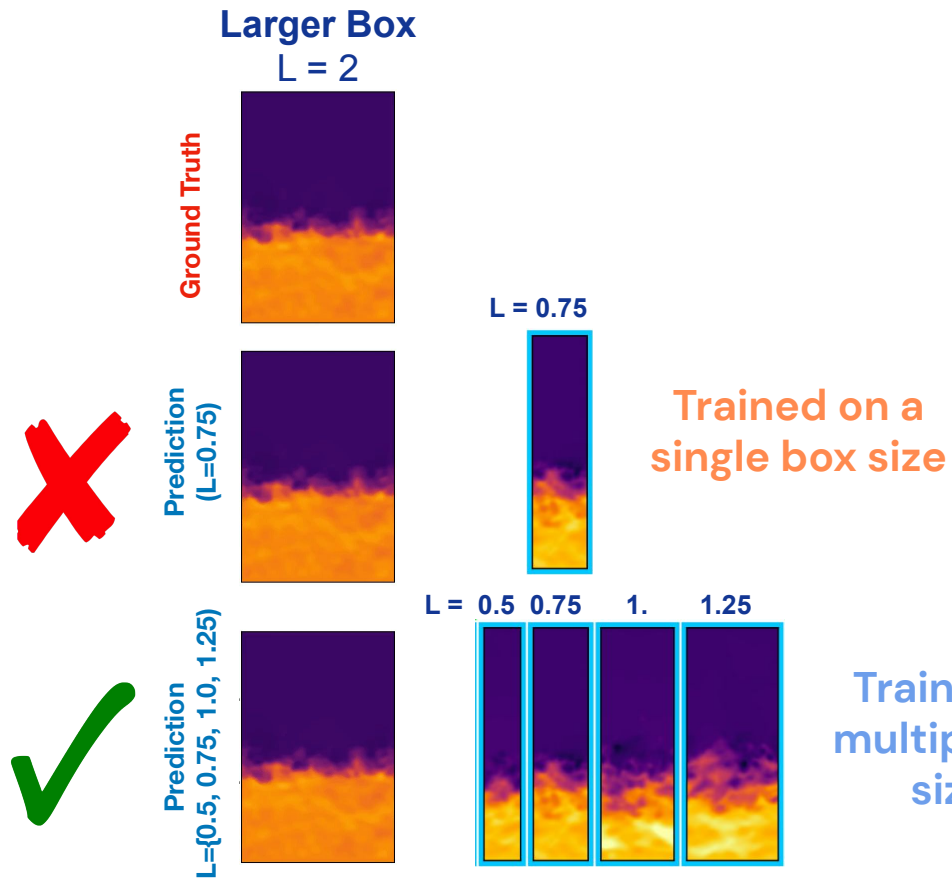
Fails to generalize to higher compressive components



Generalization to different box lengths



Generalization to different box lengths



Quantitative generalization remains a challenge



Conclusions

1. Learning reusable knowledge and inductive biases are key to generalization
2. Graph & Mesh representations for ML do scale, and are worth considering
3. Learned simulators can bring unique advantages
 - a. Accelerated predictions
 - b. Gradients and inverse design.
 - c. Interpretability
 - d. ...



DeepMind

Learning general purpose physical simulators

Thanks for your attention!

Question time?

Presenter: Alvaro Sanchez-Gonzalez

April 19, 2022

Workshop on Representation Learning from
Heterogeneous/Graph-Structured Data

Learning to Discover – Institut Pascal Paris–Saclay

