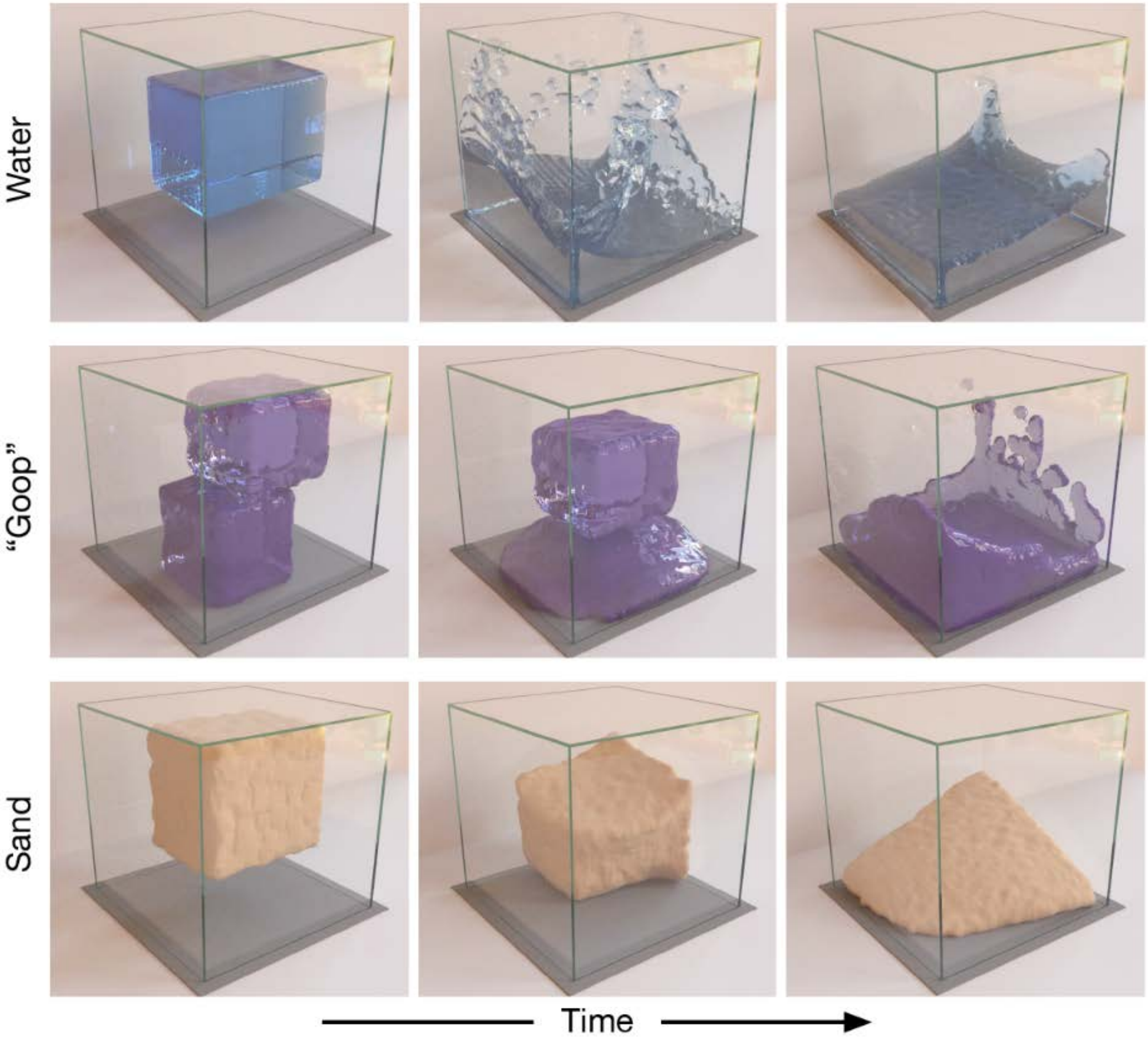
Learning to simulate complex physics

ICML under review

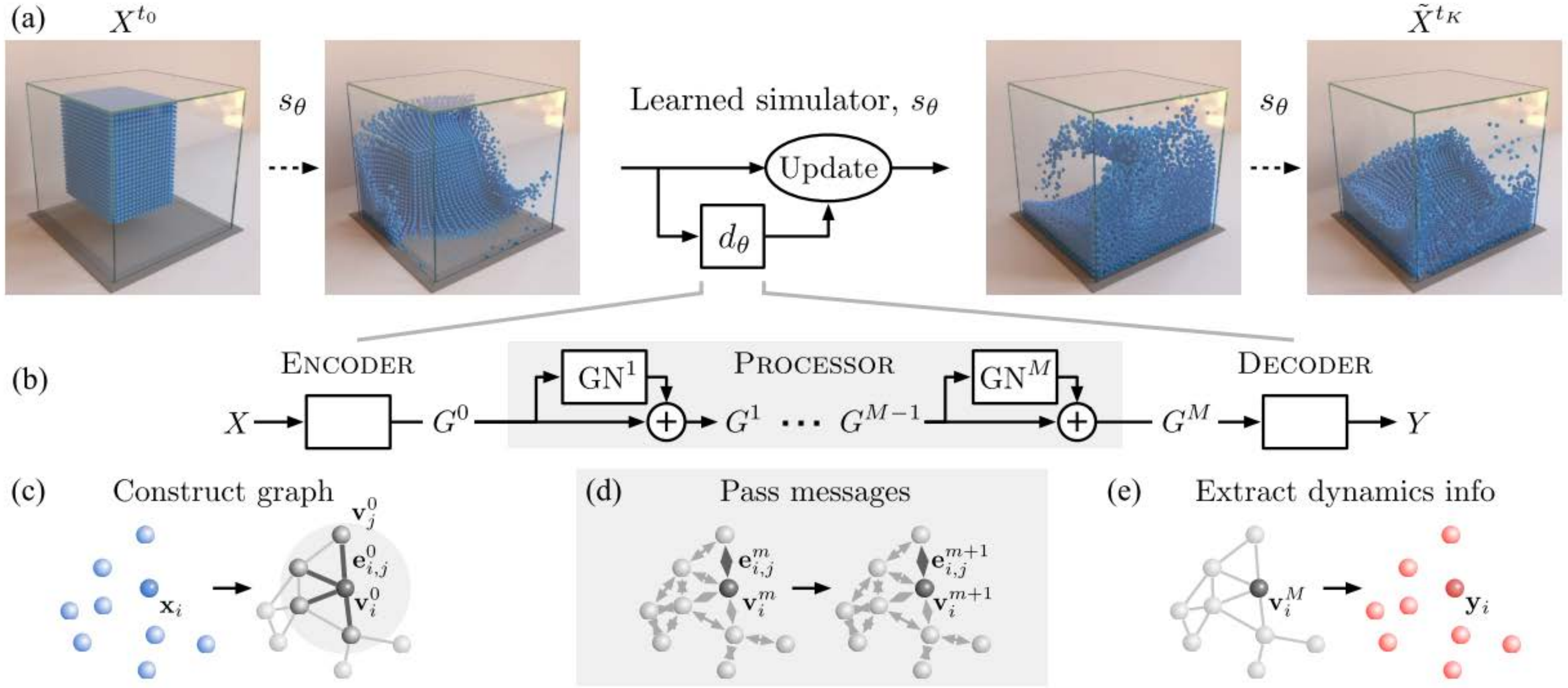
https://arxiv.org/abs/2002.09405

Task

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



Approach



(a)
X^{t_0}
$\tilde{X}^{t_K}$
s_θ
Learned simulator, s_θ
Update
d_θ
s_θ
(b)
ENCODER
PROCESSOR
DECODER
X
G^0
GN^1
G^1
G^{M-1}
GN^M
G^M
Y
(c)
Construct graph
$\mathbf{x}_i$
$\mathbf{v}_j^0$
$\mathbf{e}_{i,j}^0$
$\mathbf{v}_i^0$
(d)
Pass messages
$\mathbf{e}_{i,j}^m$
$\mathbf{v}_i^m$
$\mathbf{e}_{i,j}^{m+1}$
$\mathbf{v}_i^{m+1}$
(e)
Extract dynamics info
$\mathbf{v}_i^M$
$\mathbf{y}_i$

Setup

- $\mathrm{X}^t \in \chi$ is the state of the world at time t
- K timestep trajectory of states $\boldsymbol{X}^{t_{0:K}} = (X^{t_0}, \dots, X^{t_K})$
- A *simulator*, $s: \chi \to \chi$
- A simulated rollout trajectory as $\tilde{\boldsymbol{X}}^{t_{0:K}} = (\tilde{X}^{t_0}, \dots, \tilde{X}^{t_K})$
  - Which is computed iteratively by, $\tilde{X}^{t_{k+1}} = s(\tilde{X}^{t_k})$

Input and output representation

- Input state vector
  - Current position (absolute or relative)
  - Sequence of C previous velocities
  - Material properties F (water, sand, rigid, boundary particle, ⋯)
  - Global properties g (external forces, ⋯)

$$\mathrm{X}_i^{t_k} = [\mathrm{P}_i^{t_k}, \dot{\mathrm{P}}_i^{t_{k-C+1}}, \dots, \dot{\mathrm{P}}_i^{t_k}, F_i]$$

Input and output representation

- Output
  - For all decoder state vector y_i (per particle)
  - It outputs accelerations $\ddot{P}_i$

- Update
  - Euler integration
  - Assume $\Delta t = 1$

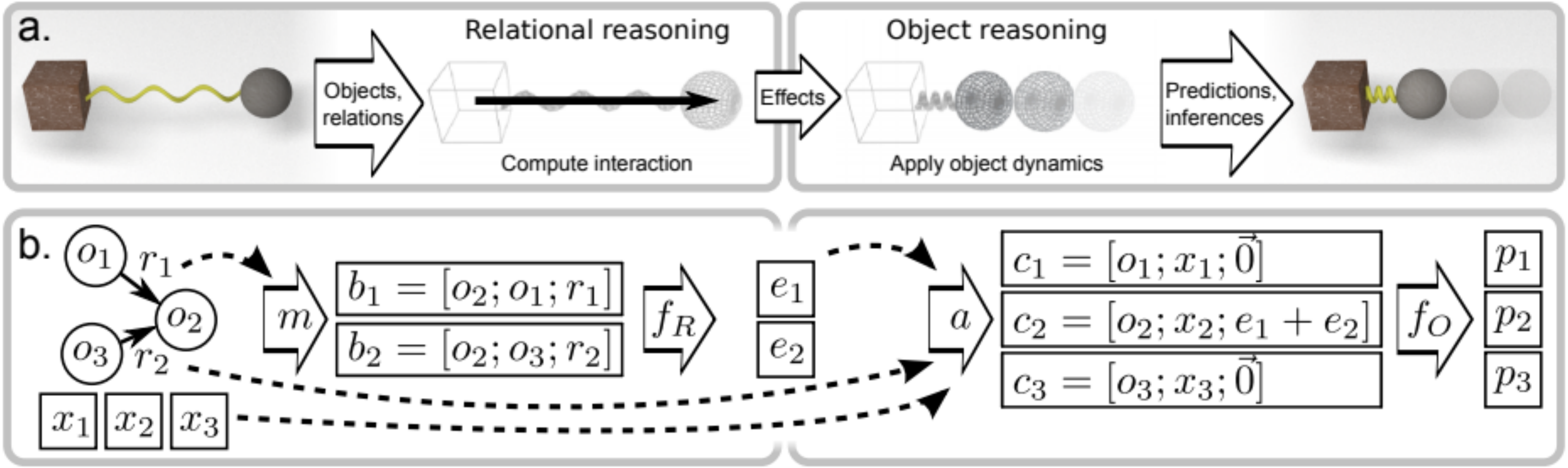
$$\dot{P}^{t_{k+1}} = \dot{P}^{t_k} + \Delta t \cdot \ddot{P}^{t_k}$$
$$P^{t_{k+1}} = P^{t_k} + \Delta t \cdot \dot{P}^{t_{k+1}}$$

Encoder / Decoder

- Construct the graph structure G_0
    - Connectivity radius R (K-NN from K-d tree)
    - ε^v and ε^e as MLP

Processor

- M stack of Battaglia et al., 2016
  - Interaction Networks for Learning about Objects, Relations and Physics



Loss

- Sample particle state pairs $(X_i^{t_k}, X_i^{t_{k+1}})$ randomly
  - From training trajectories

- Calculated target accelerations $\ddot{P}_i^{t_k}$
  - L_2 loss on the predicted per-particle accelerations

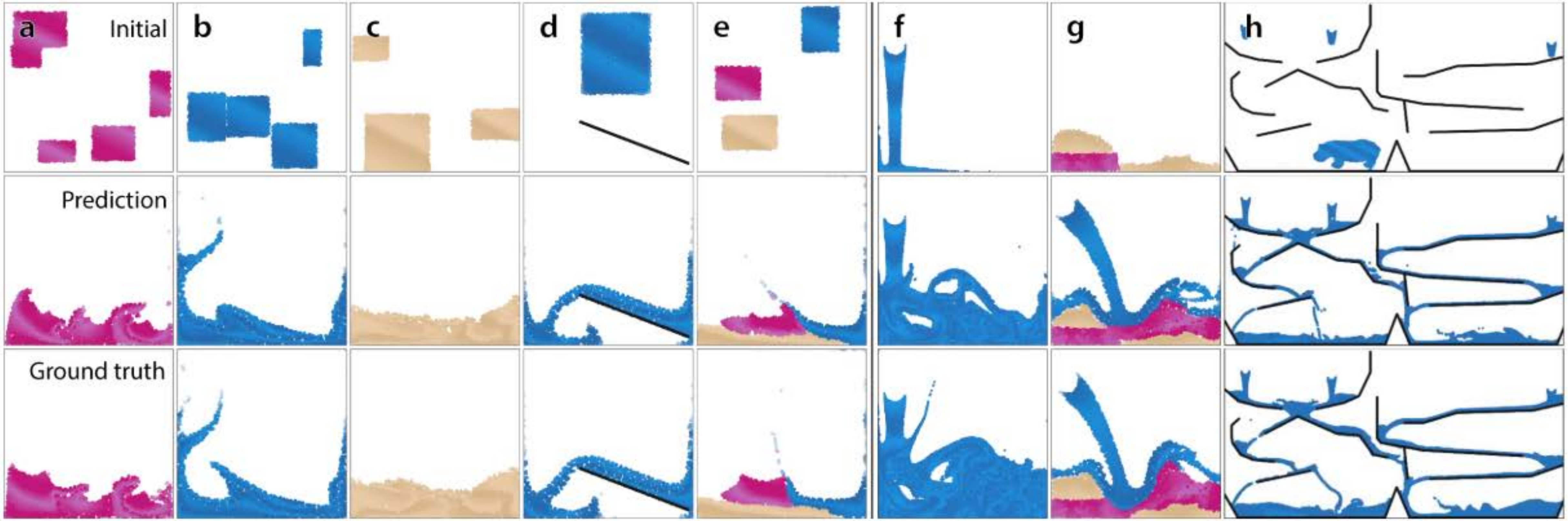
$$L\left(X_i^{t_k}, X_i^{t_{k+1}}; \theta\right) = \left\| d_\theta\left(X_i^{t_k}\right) - \ddot{P}_i^{t_k} \right\|^2$$

Tricks

- Training noise
  - To avoid error propagation
- Normalization
  - On input and target vectors
  - Elementwise to zero mean and unit variance
- Stopping criteria
  - Stop training when we observed negligible decrease in MSE
  - Training usually takes a few hours for small dataset
  - Training usually takes up to a week for complex dataset

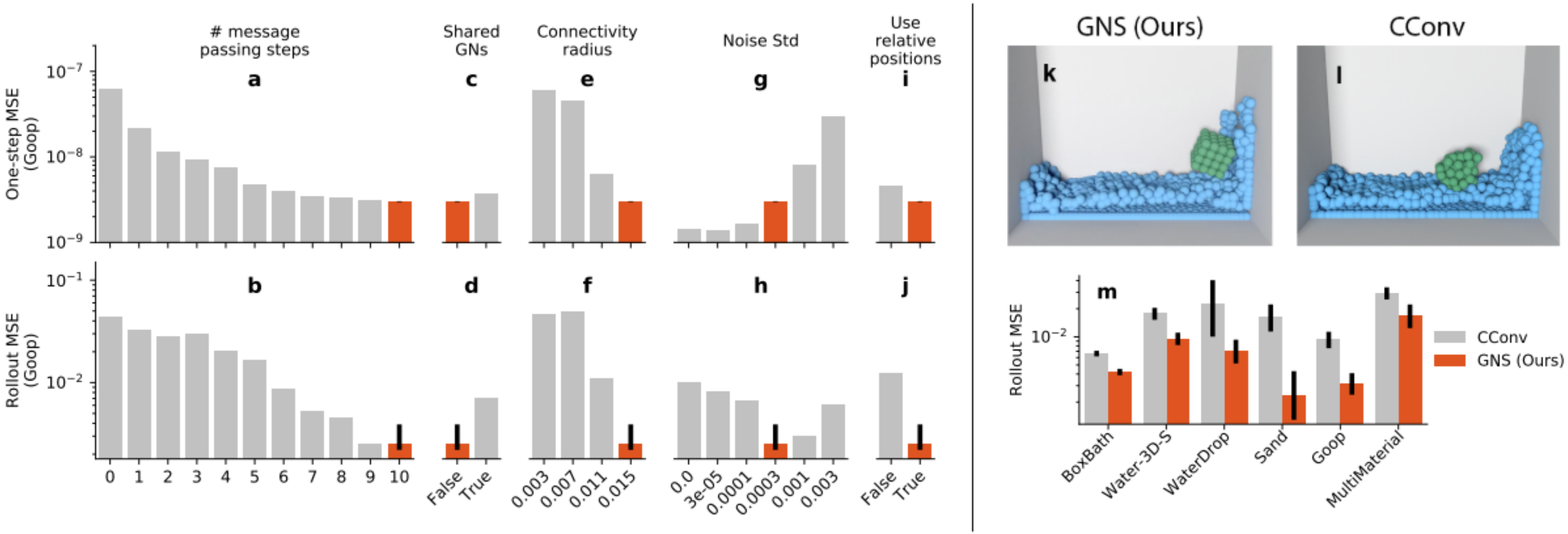
Results

- Model can generalize the physical processes



Results

- Model can generalize the physical processes

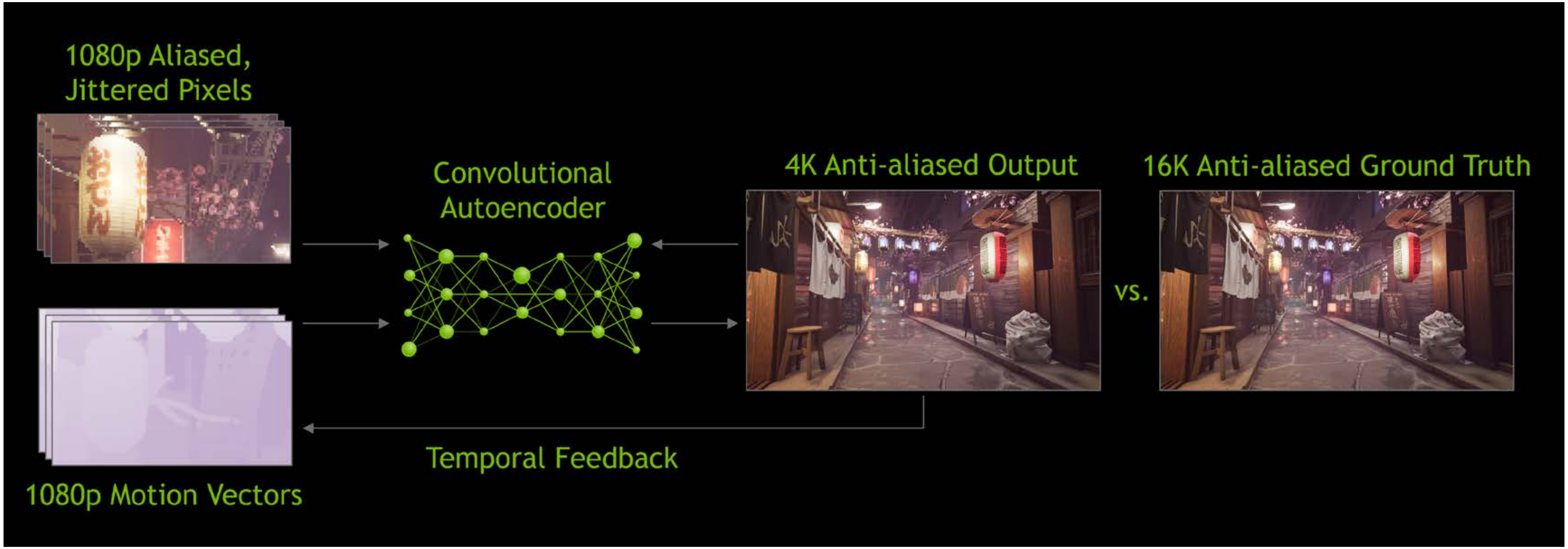


Conclusion

- General purpose ML framework for simulating complex systems
- Learned, differentiable simulators

Etc

- NVIDIA DLSS



Questions

- Why we use deep learning on physical simulation?
  - Unified framework? Yes
  - Accuracy? No
  - Speed? No

- Computer vision?
  - Point cloud?
  - Graph neural network?