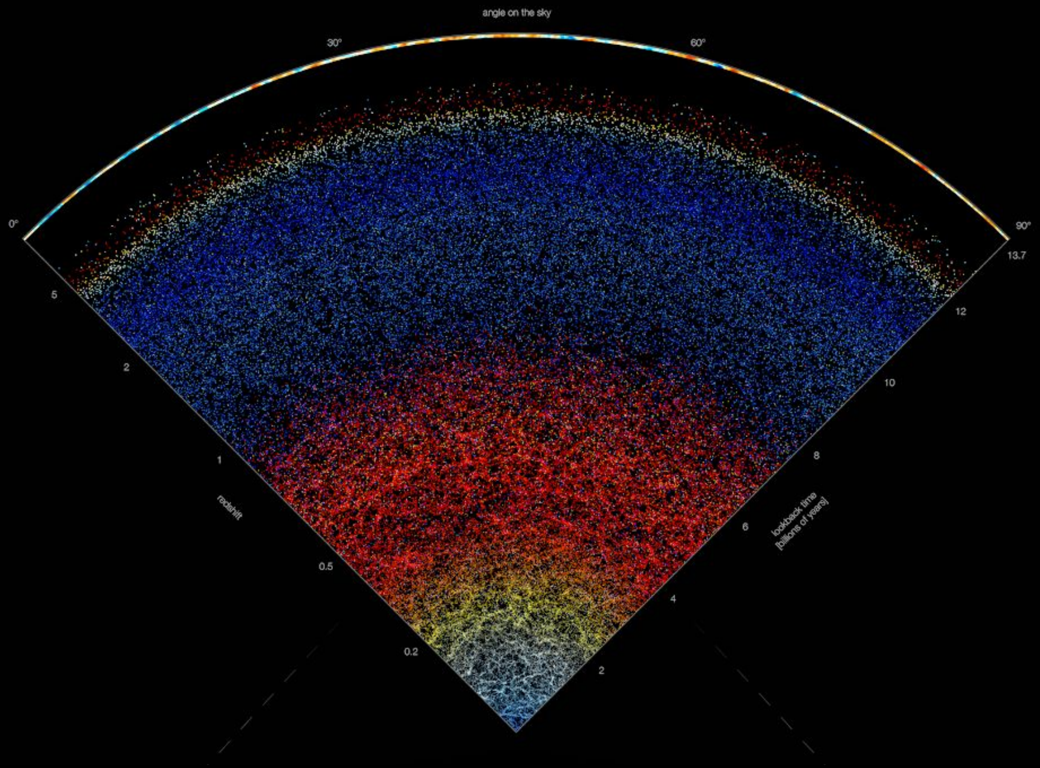
Forward Model the Universe in the Era of Deep Learning

Yin Li (Peng Cheng Laboratory)

32nd Texas Symposium in Shanghai 2023-12-15

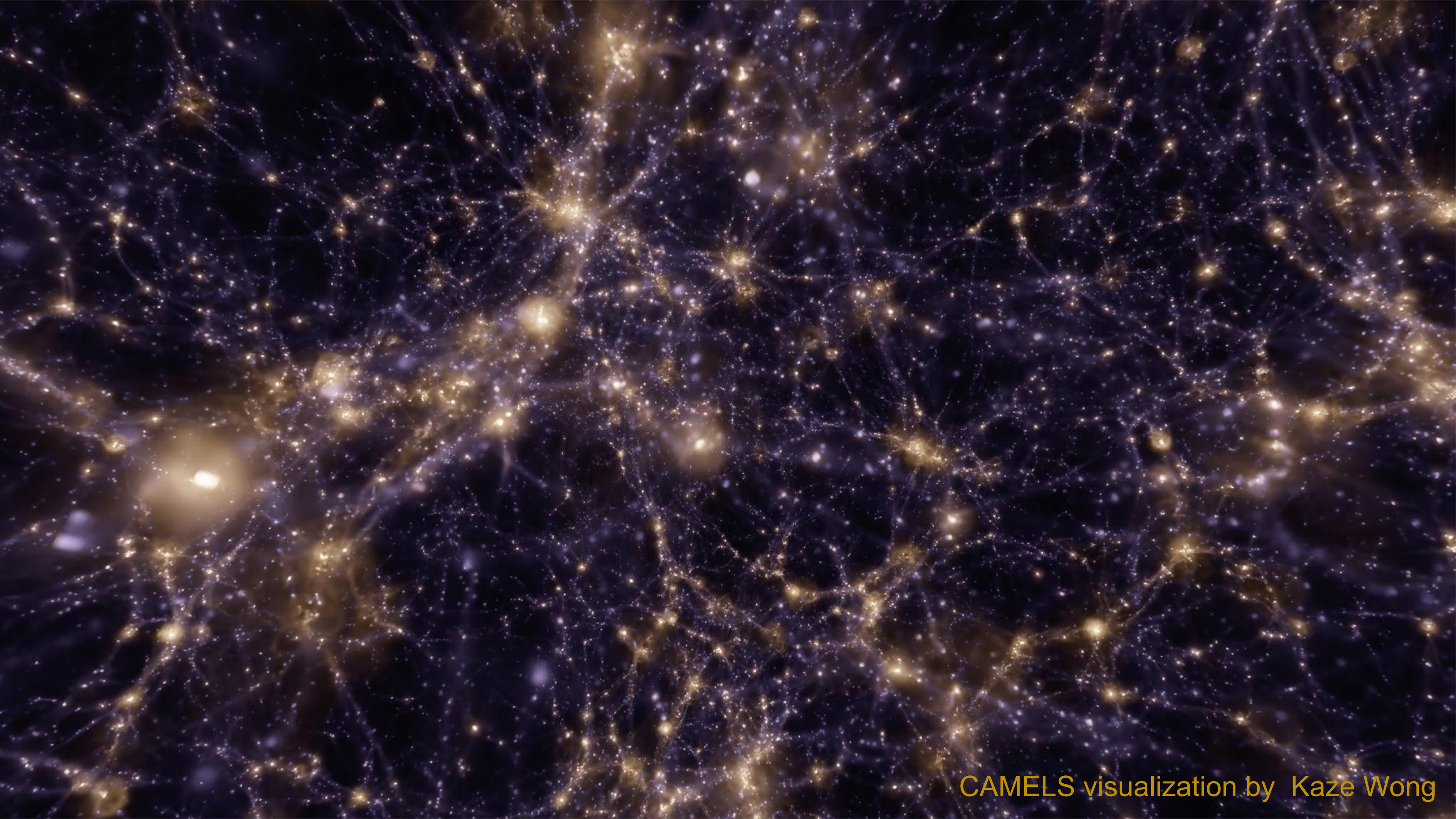
Model

Data

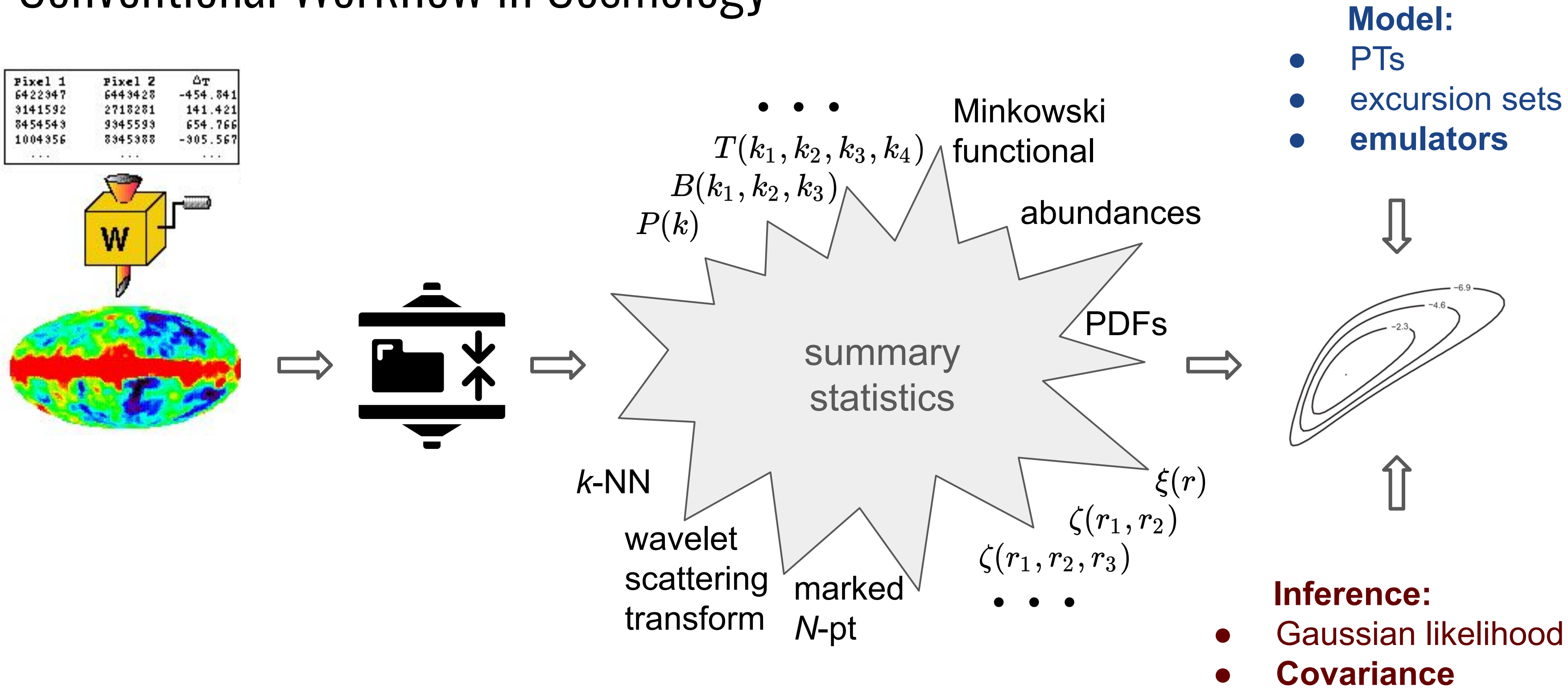


Inference

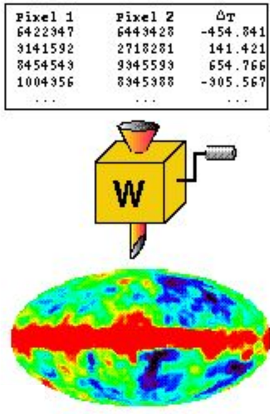
https://mapoftheuniverse.net/

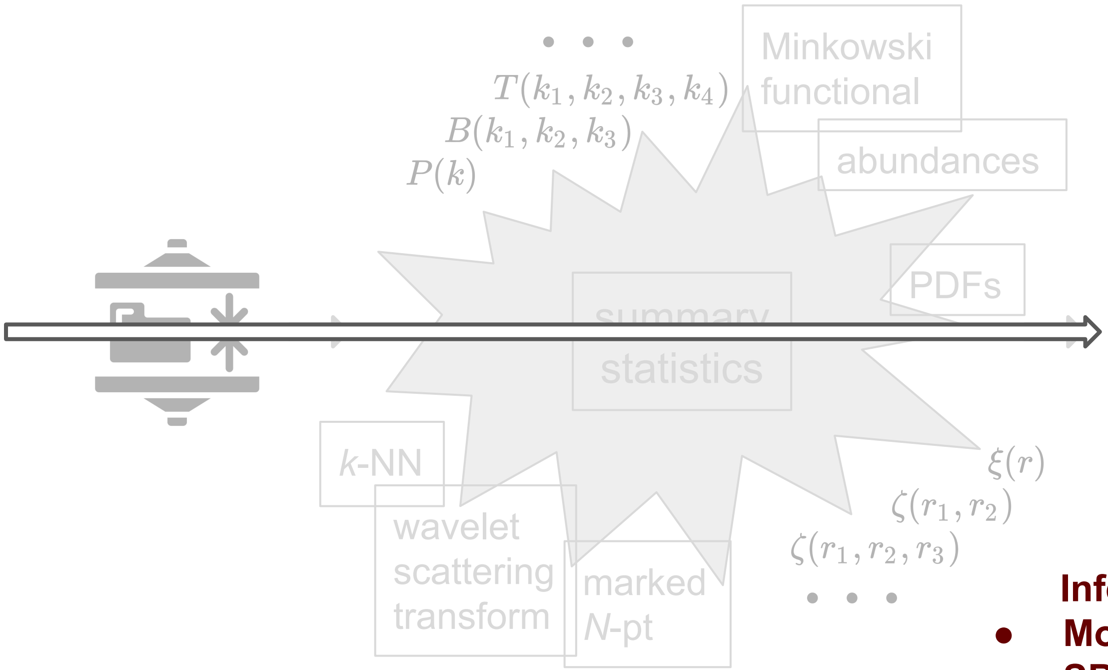

CAMELS visualization by Kaze Wong

Conventional Workflow in Cosmology



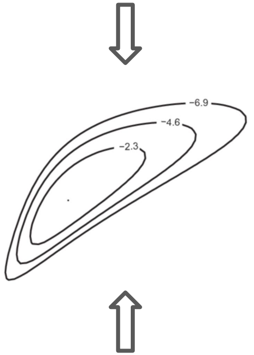
Field-level Modeling & Inference





Model:

- **LPTs**
- **simulations**
- **systematics …**

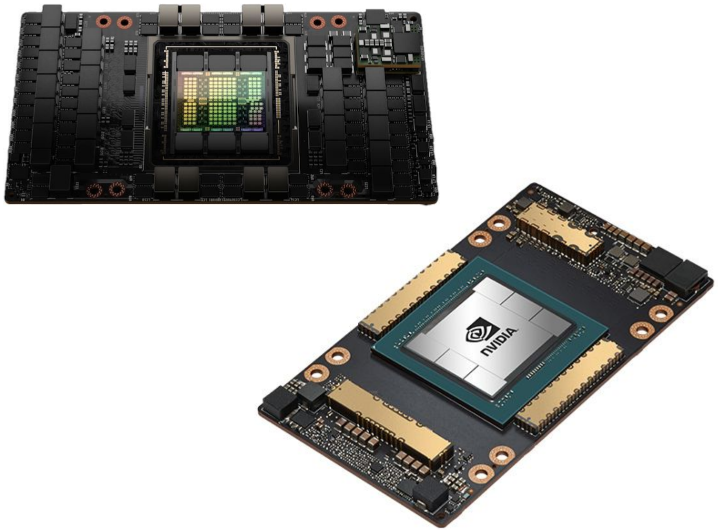


Inference:

- **Model ICs then marginalize**
- **SBI (sim. based inference)**

Hardware & Software Breakthroughs can also help Established Research

Hardware



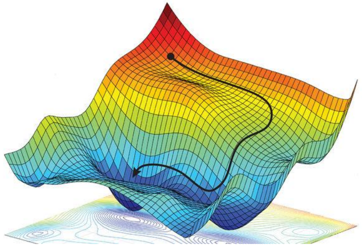
Software

PyTorch

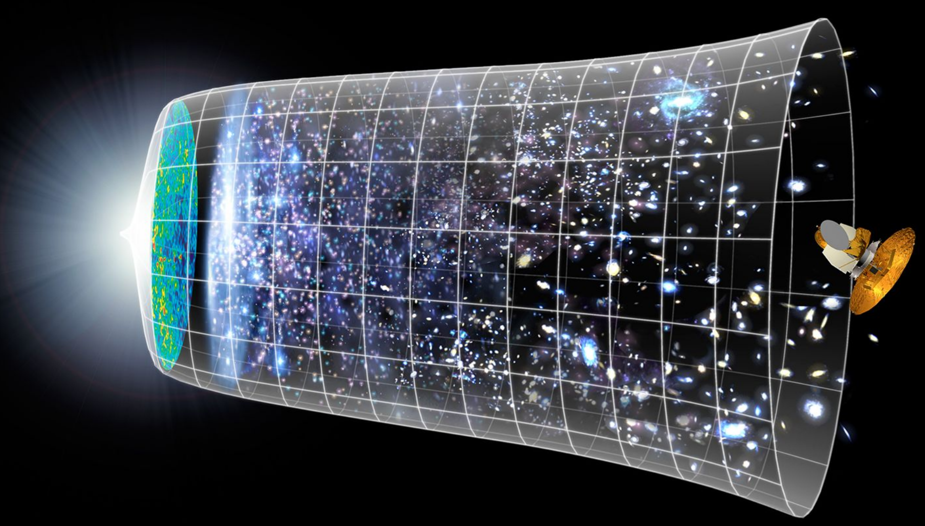
TensorFlow

JAX

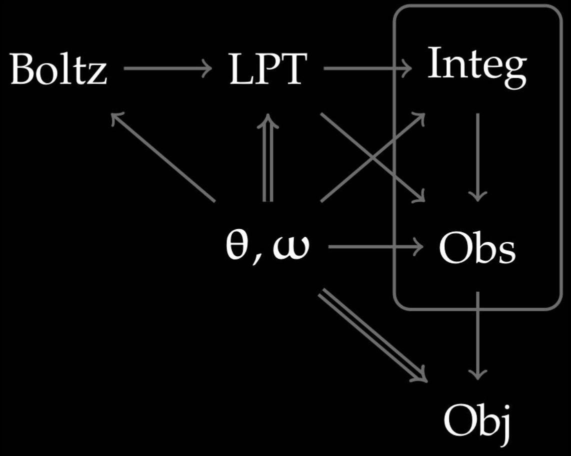
julia



Cosmological Forward Model

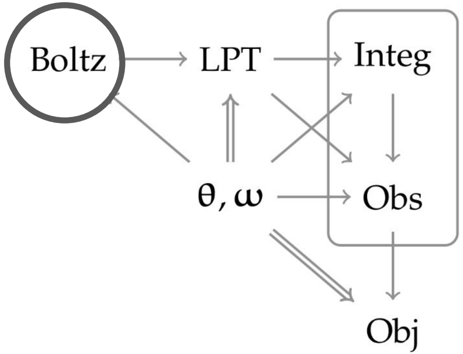


NASA/WMAP Science Team



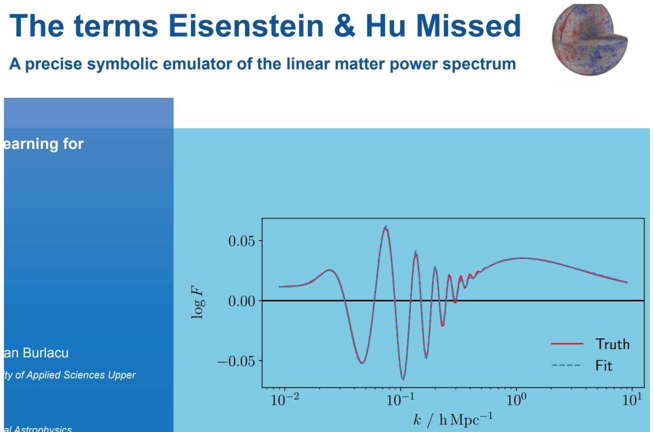
(a) model structure

Boltzmann solvers



(a) model structure

- Fitting formulae (Eisenstein & Hu) 1e-2~1e-1
- Neural network emulators (CosmoPower, CosmicNet, Arico+, …), differentiable by construction; 1e-3~1e-2
- Differentiable Boltzmann solvers, 1e-3
  - DISCO-DJ (Hahn+)
  - Bolt.jl (Z. Li+) only forward
- Symbolic regression (Bartlett+) 1e-3

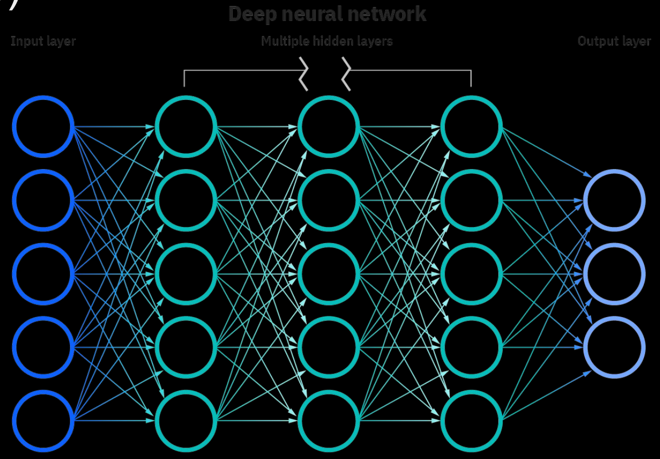


Existing differentiable cosmological simulation codes

- `BORG` (Jasche & Wandelt 2013)
- `ELUCID` (Wang et al. 2014, …)
- `FastPM+vmad` (Feng et al. 2016 & Seljak et al. 2017, …)
- `BORG-PM` (Jasche & Lavaux 2019, …)
- `FlowPM` (Modi et al. 2020)
- ...

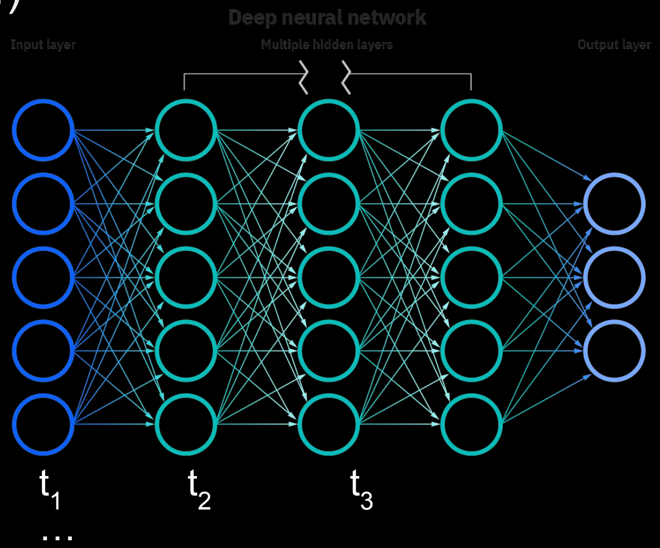
Existing differentiable cosmological simulation codes

- `BORG` (Jasche & Wandelt 2013)
- `ELUCID` (Wang et al. 2014, …)
- `FastPM+vmad` (Feng et al. 2016 & Seljak et al. 2017, …)
- `BORG-PM` (Jasche & Lavaux 2019, …)
- `FlowPM` (Modi et al. 2020)
- …



Existing differentiable cosmological simulation codes

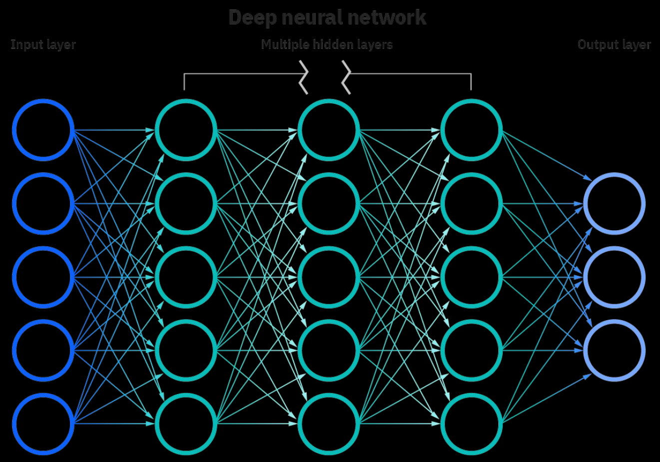
- `BORG` (Jasche & Wandelt 2013)
- `ELUCID` (Wang et al. 2014, …)
- `FastPM+vmad` (Feng et al. 2016 & Seljak et al. 2017, …)
- `BORG-PM` (Jasche & Lavaux 2019, …)
- `FlowPM` (Modi et al. 2020)
- …



Trade-off between spatial and time complexity

Existing differentiable cosmological simulation codes

- `BORG` (Jasche & Wandelt 2013)
- `ELUCID` (Wang et al. 2014, …)
- `FastPM+vmad` (Feng et al. 2016 & Seljak et al. 2017, …)
- `BORG-PM` (Jasche & Lavaux 2019, …)
- `FlowPM` (Modi et al. 2020)
- …

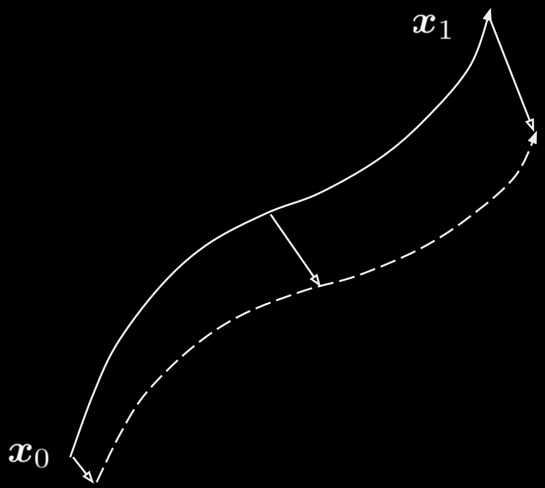


~~Trade-off between spatial and time complexity~~

Adjoint method

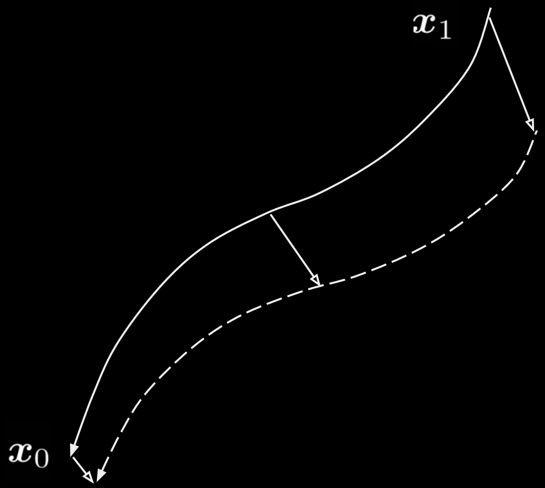
Lev Pontryagin 1956

Algorithm Summary: Forward



$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{F}_k(\boldsymbol{x}_k, \boldsymbol{\theta}), \qquad k = 0, \cdots, n-1,$$

Algorithm Summary: Backward



$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{F}_k(\boldsymbol{x}_k, \boldsymbol{\theta}), \qquad k = 0, \cdots, n-1,$$

$$\boldsymbol{\lambda}_{k-1} = \boldsymbol{\lambda}_k + \boldsymbol{\lambda}_k \cdot \frac{\partial \boldsymbol{F}_{k-1}}{\partial \boldsymbol{x}_{k-1}}, \qquad \boldsymbol{\lambda}_n = -\frac{\partial \mathcal{J}}{\partial \boldsymbol{x}_n},$$

$$\frac{\mathrm{d}\mathcal{J}}{\mathrm{d}\boldsymbol{\theta}} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}} - \boldsymbol{\lambda}_0 \cdot \frac{\partial \boldsymbol{x}_0}{\partial \boldsymbol{\theta}} - \sum_{k=1}^{n} \boldsymbol{\lambda}_k \cdot \frac{\partial \boldsymbol{F}_{k-1}}{\partial \boldsymbol{\theta}}.$$

pmwd (particle mesh with derivatives)

- A differentiable particle-mesh library based on **JAX**
  - ✓ fast sim on GPU
  - ✓ 2x memory & 3~4x runtime with derivatives
  - ✓ open sourced at github.com/eelregit/pmwd

code	OSS	gradient	mem efficient	hardware
ELUCID		analytic		CPU
BORG-PM		analytic		CPU
`FastPM-vmad`	✓	AD		CPU
`FlowPM`	✓	AD		GPU/CPU
pmwd	✓	adjoint	✓	GPU/CPU

Memory efficient: on 1 GPU (say A100)

- FlowPM: 128^3 particles for 10 time steps
- pmwd: 512^3 particles for any # of steps

2211.09815 (ApJS accepted), 2211.09958

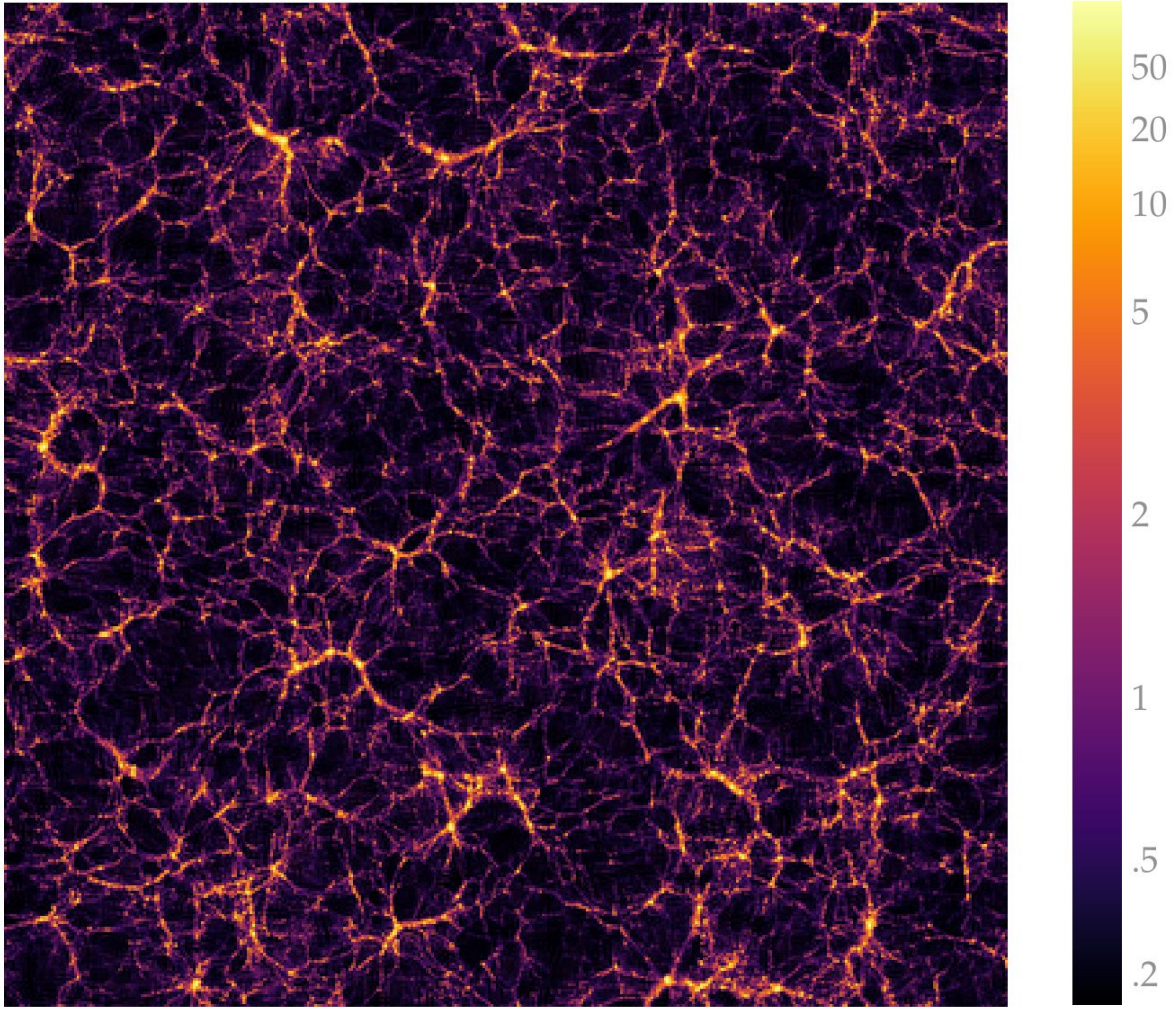
13s to run this:

512^3 particles

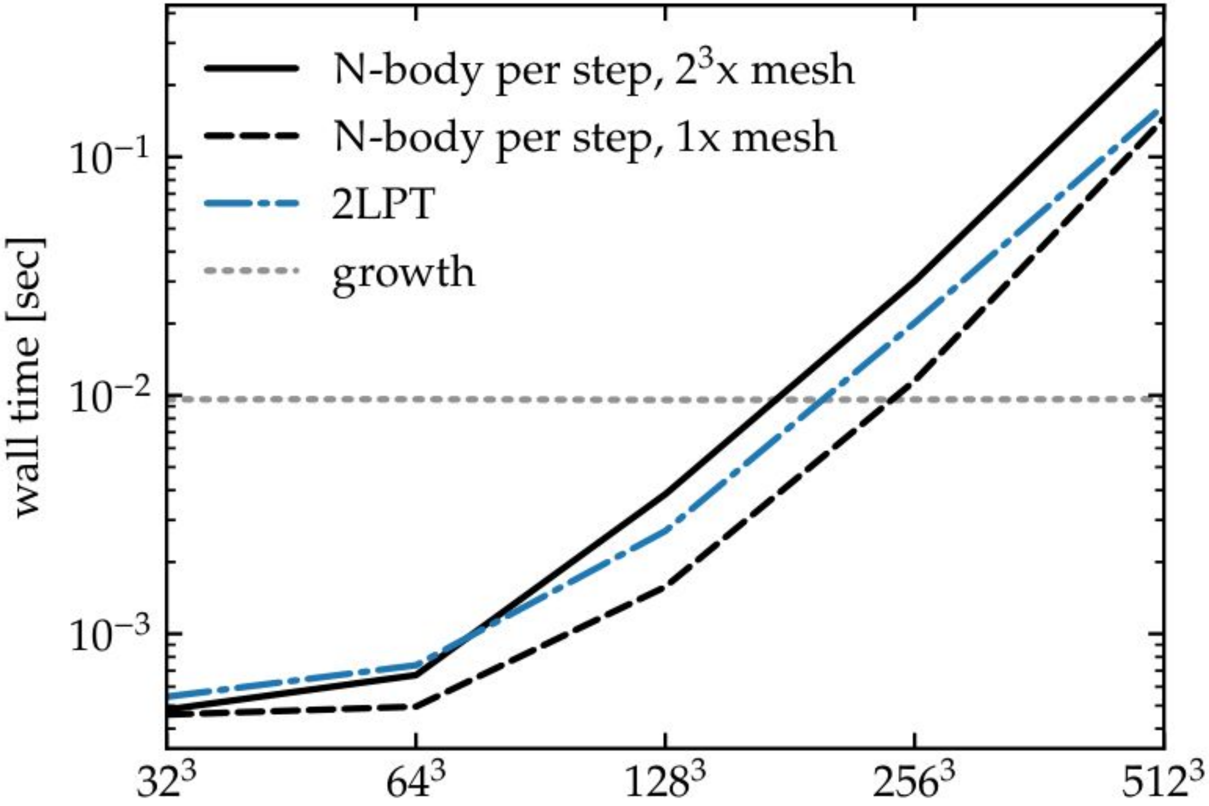
1024^3 mesh

63 time steps

1 H100 PCIe GPU



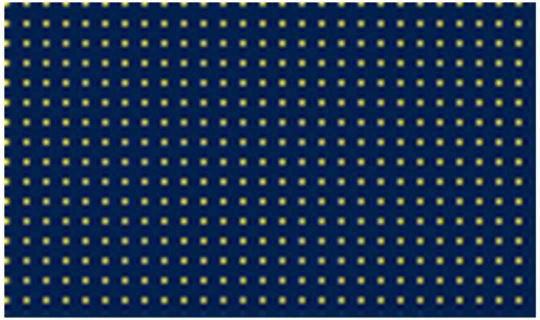
Performance



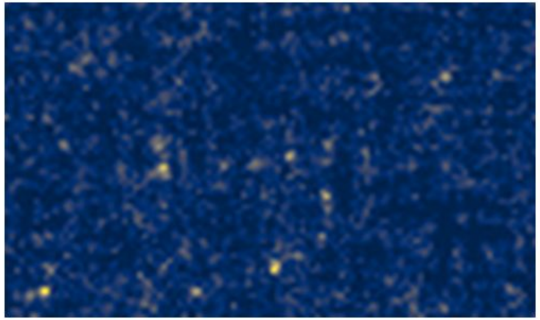
Computation efficient: 256^3 particles per step

- FastPM on 1 CPU: 40s
- FastPM on 32 CPU: 3s
- FlowPM on 1 GPU: 1s
- FlowPM on 32 GPU: 0.4s
- pmwd on 1 GPU: 0.01s
- pmwd on n GPU: in dev.

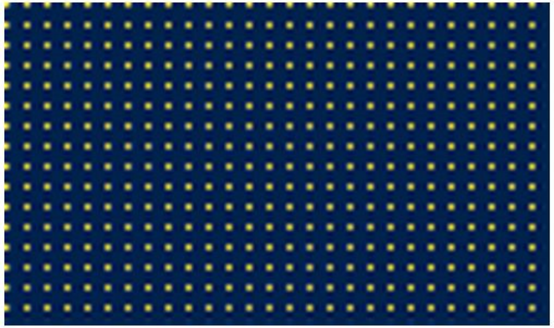
A Toy Optimization Problem



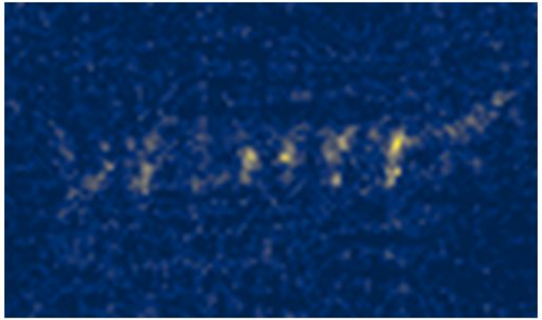
(a) initial conditions at $a = 1/64$



(b) final snapshot at $a = 1$



(c) reverse evolution back to $a = 1/64$

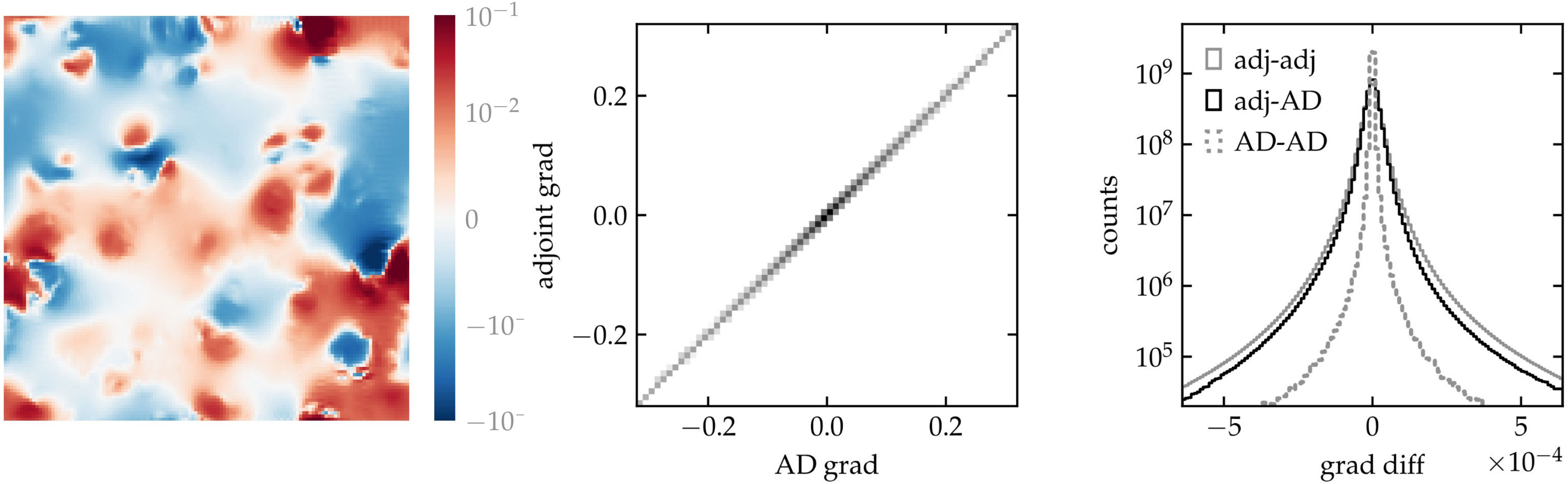


(d) optimization for 10 iterations at $a = 1$

(e) optimization for 100 iterations at $a = 1$

(f) optimization for 1000 iterations at $a = 1$

Accuracy of the Gradients: Comparison with Automatic Differentiation



Optimize spatial resolution (at the same computational cost)

"Sharpen" the PM Poisson solver kernel, taking symmetry and dimensional analysis into account:

$$\frac{ik_i}{k^2} \to \frac{ik_i}{k^2} f(k_i;\vartheta) g(k_1, k_2, k_3;\vartheta),$$

the most general form due to symmetry, with f being a neural network, and g being an permutation equivariant neural network; both f and g are even functions

$$f(k_i;\vartheta) = f(k_i/k_P, kR_{TH}, \cdots; D, f, \cdots).$$

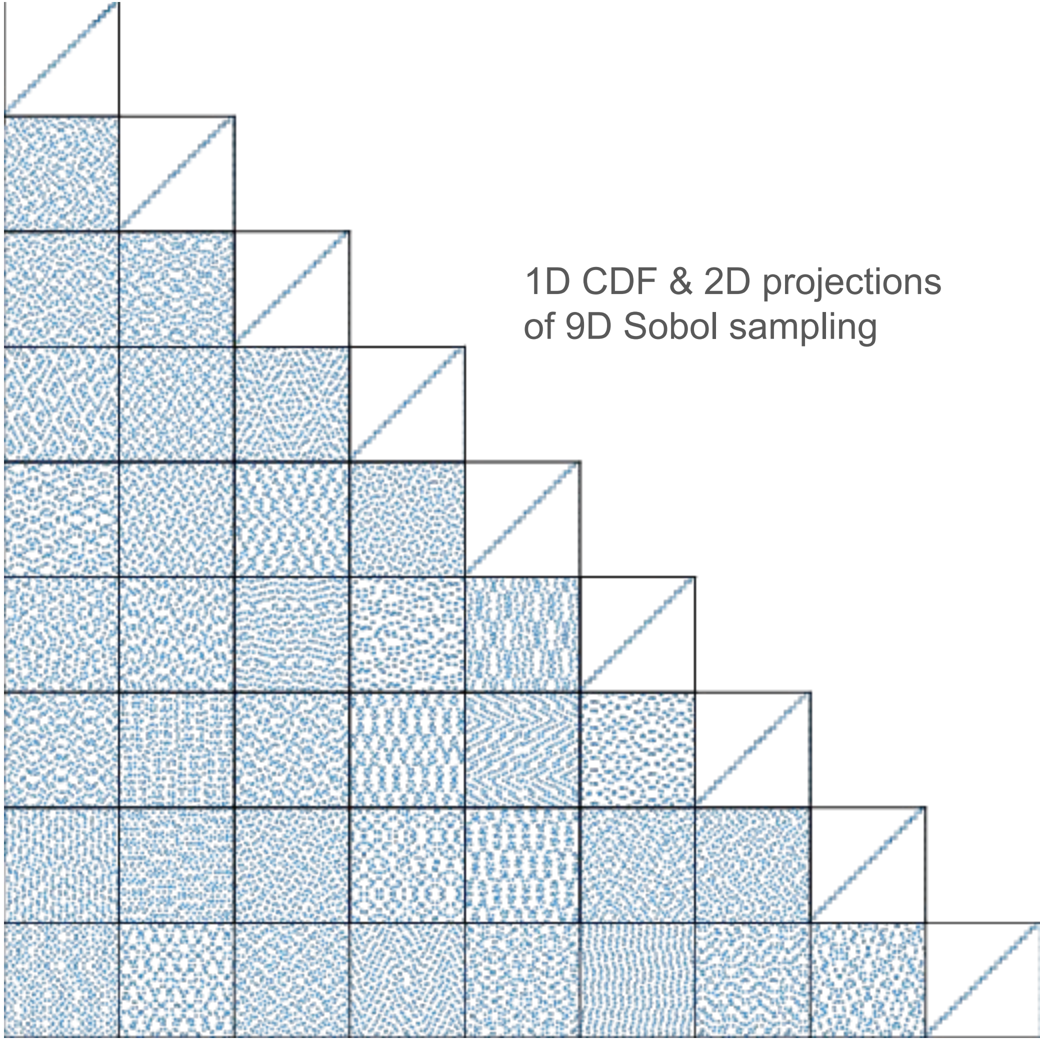
k and R are characteristic scales (eg the nonlinear scales);

D, …, stand for dimensionless parameters;

Therefore, f and g networks are nonlinear functions of dimensionless combinations (Buckingham's Pi theorem)

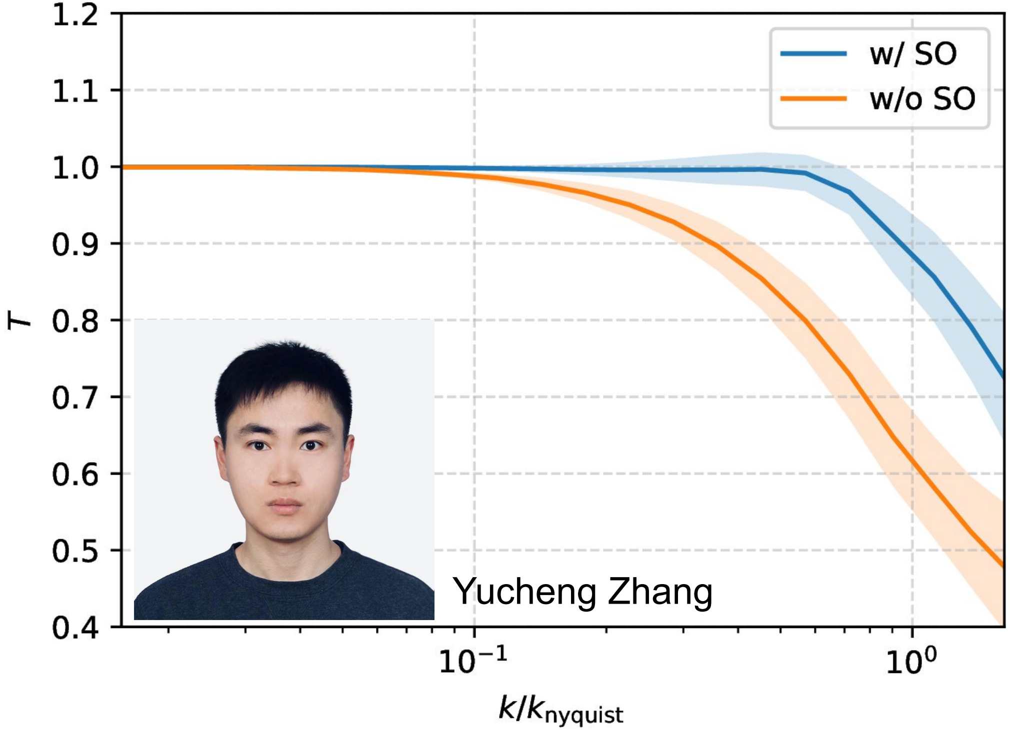
Training Data

- 512 simulations (128^3 particles)
- 9D parameter space uniformly sampled with Sobol sequence (quasi-Monte Carlo)



1D CDF & 2D projections of 9D Sobol sampling

Preliminary results of spatial optimization (SO)



quantify accuracy by:

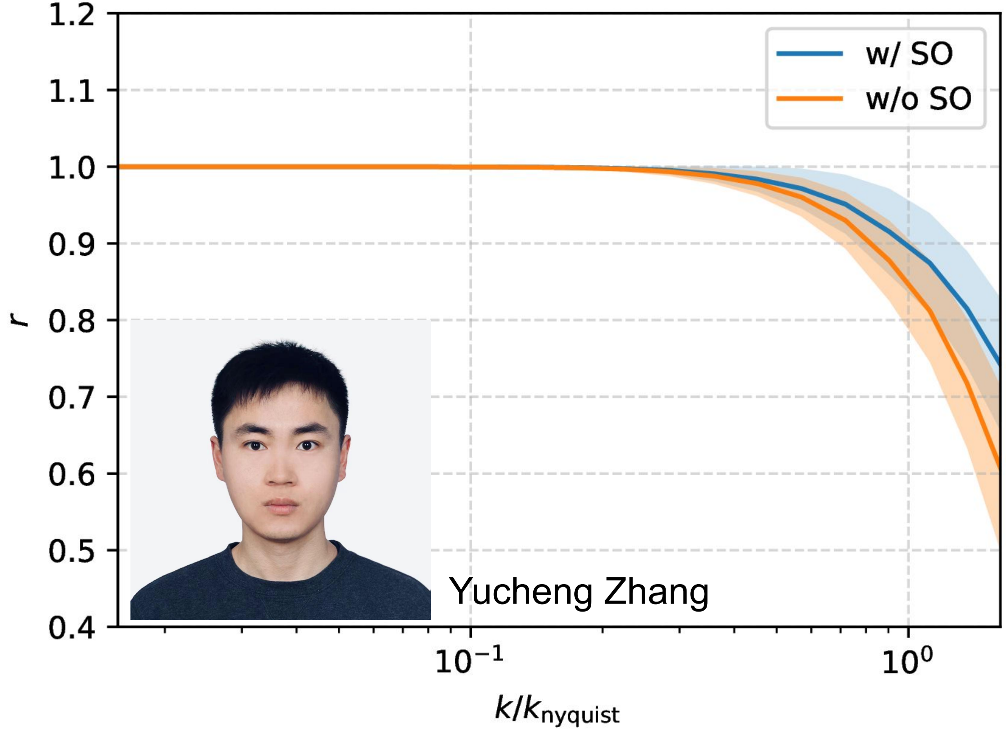
$$T(\mathbf{k}) \triangleq \sqrt{\frac{|\hat{\delta}(\mathbf{k})|^2}{|\delta(\mathbf{k})|^2}}, \quad r(\mathbf{k}) \triangleq \frac{\Re\left[\hat{\delta}(\mathbf{k})\delta^*(\mathbf{k})\right]}{\sqrt{|\hat{\delta}(\mathbf{k})|^2|\delta(\mathbf{k})|^2}}.$$

2000 CPU-hours vs O(0.1) of GPU-sec

$$\frac{ik_i}{k^2} \to \frac{ik_i}{k^2} f(k_i;\vartheta) g(k_1, k_2, k_3;\vartheta),$$

Symbolic regression of the trained networks

Preliminary results of spatial optimization (SO)



quantify accuracy by:

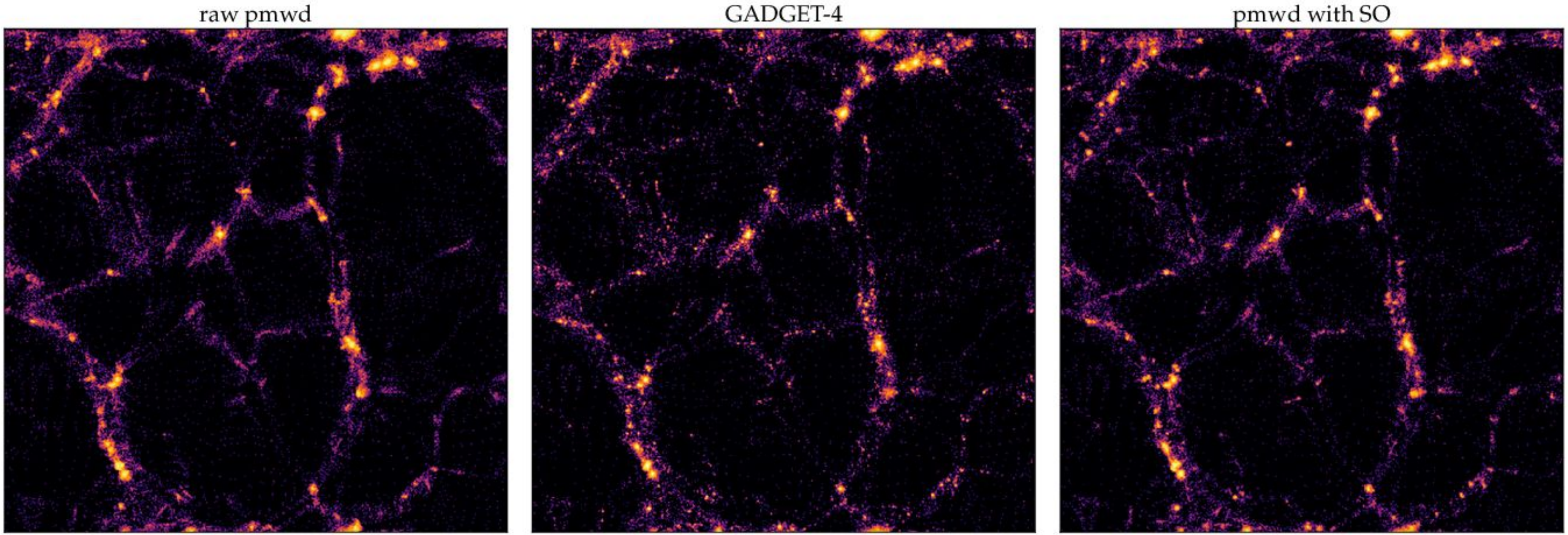
$$T(\mathbf{k}) \triangleq \sqrt{\frac{|\hat{\delta}(\mathbf{k})|^2}{|\delta(\mathbf{k})|^2}}, \quad r(\mathbf{k}) \triangleq \frac{\Re\left[\hat{\delta}(\mathbf{k})\delta^*(\mathbf{k})\right]}{\sqrt{|\hat{\delta}(\mathbf{k})|^2|\delta(\mathbf{k})|^2}}.$$

2000 CPU-hours vs O(0.1) of GPU-sec

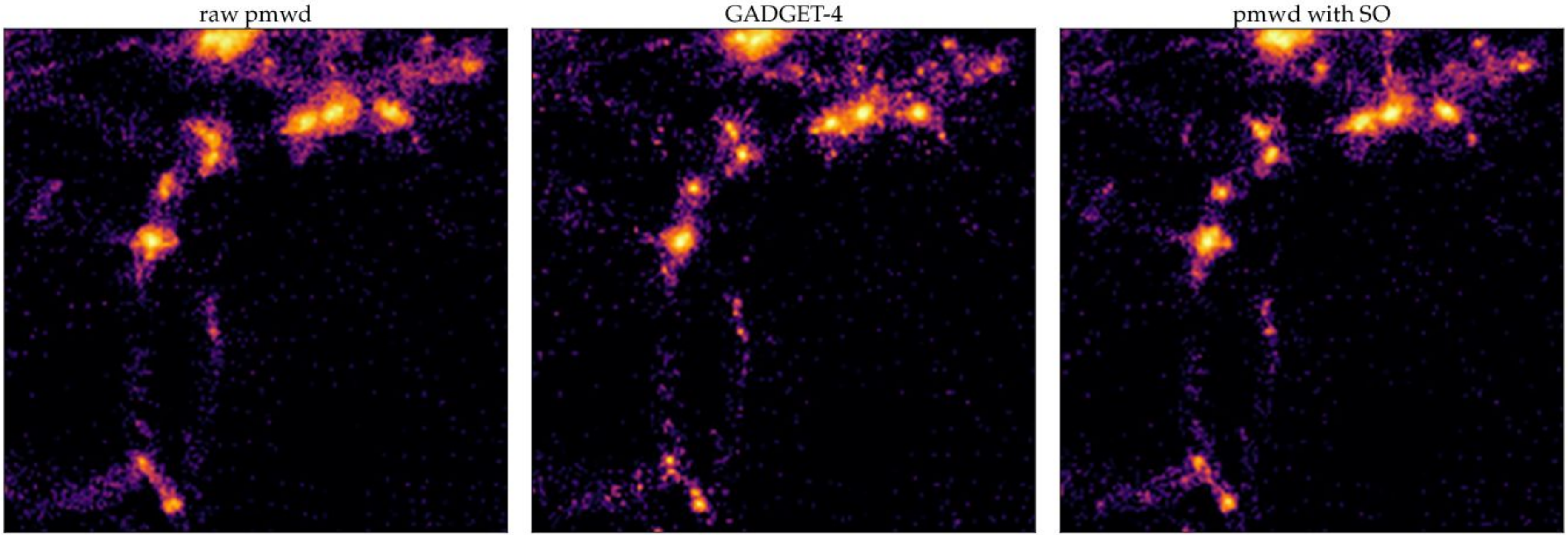
$$\frac{ik_i}{k^2} \rightarrow \frac{ik_i}{k^2} f(k_i;\vartheta) g(k_1, k_2, k_3;\vartheta),$$

Symbolic regression of the trained networks

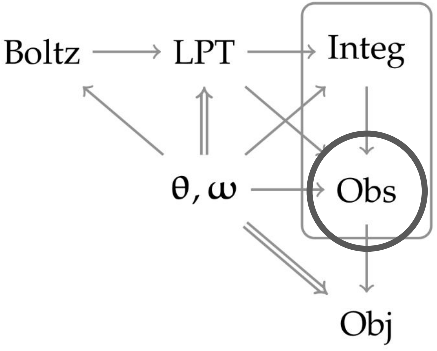
Preliminary results of spatial optimization (SO)



Preliminary results of spatial optimization (SO)

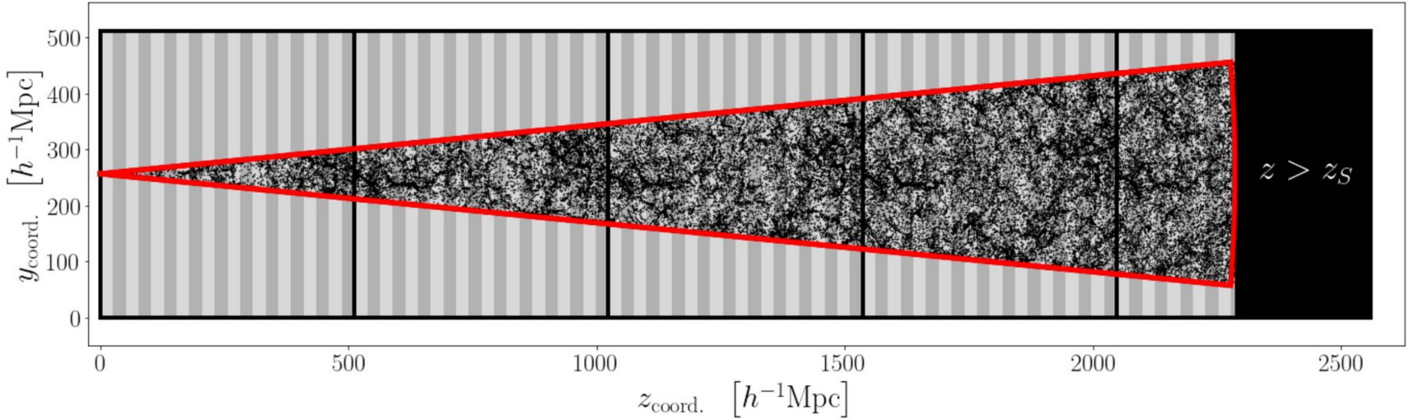


Simulating observables on-the-fly with post-Born ray tracing

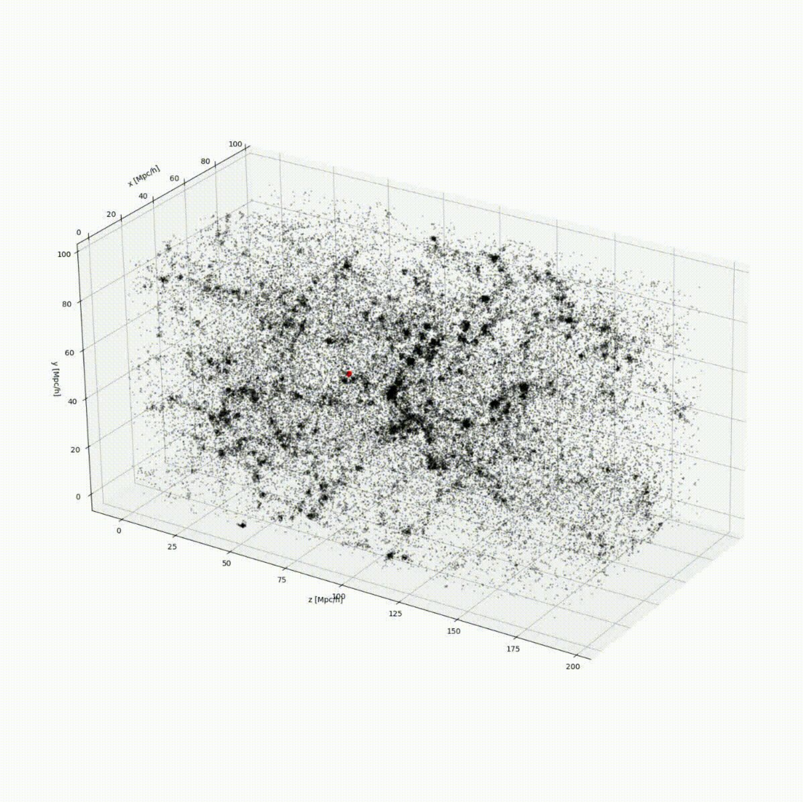


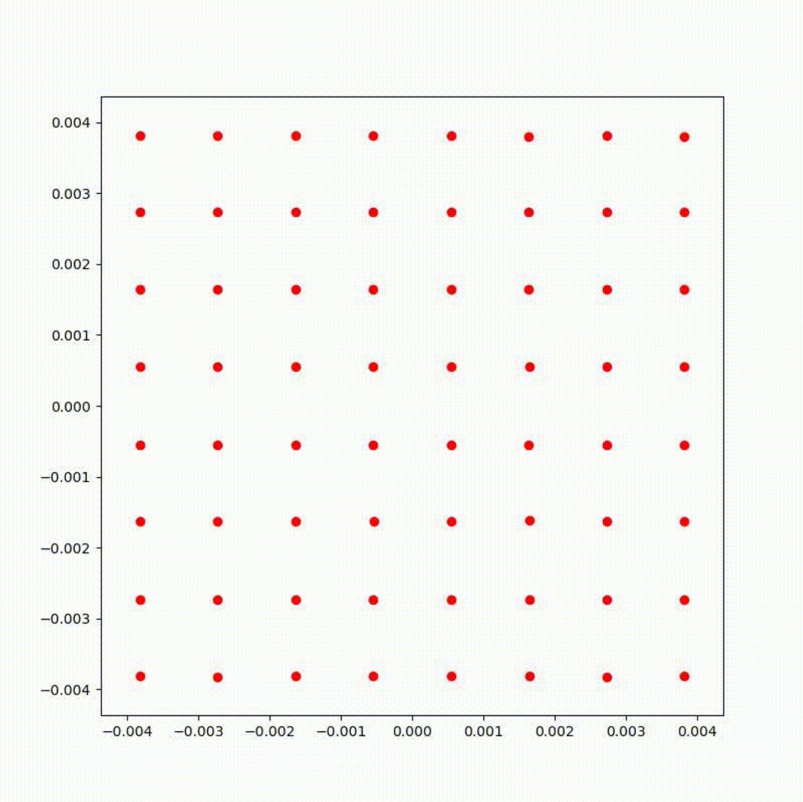
(a) model structure

Junzhe (Alan) Zhou



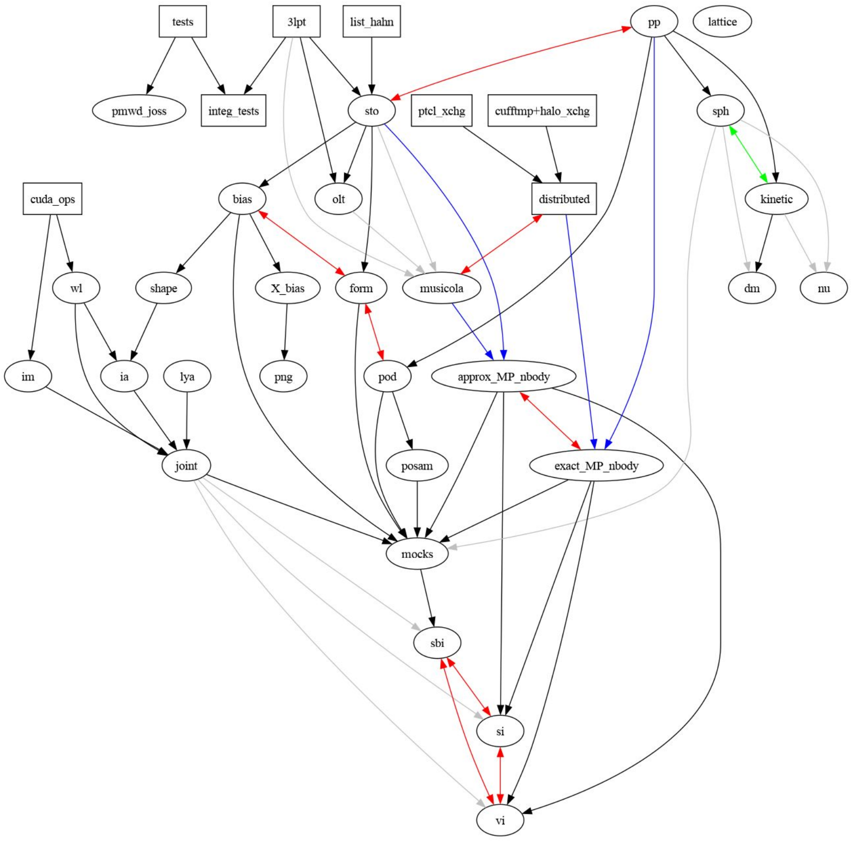
Simulating observables on-the-fly with post-Born ray tracing





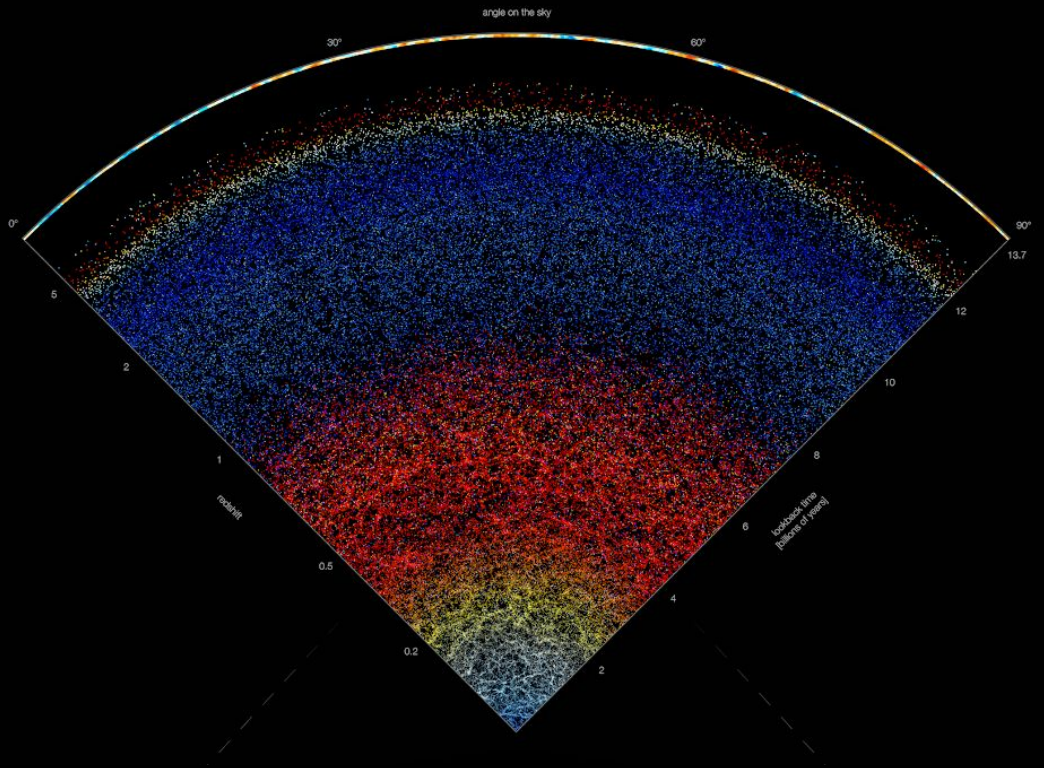
Unified Framework for Cosmology

- Unified model for all observables:
  - weak lensing
  - galaxies, number density & intrinsic alignment
  - Lya skewers
  - intensity mapping
  - foregrounds & systematics
- Baseline model for new physics extensions:
  - neutrinos
  - early Universe lattice simulation
  - other dark matter interactions
- Performance
  - small scales: short range force (PP or ML correction)
  - large scales: approx or exact parallelization



pmwd Universe

Model



Data

Inference

Reversibility

precision	cell/ptcl	ptcl mass [$10^{10} M_\odot$]	time steps	disp rel diff	vel rel diff
single	8	1	63	5.2×10^{-2}	7.1×10^{-2}
single	8	1	126	2.1×10^{-2}	3.6×10^{-2}
single	8	8	63	3.3×10^{-3}	7.6×10^{-3}
single	8	8	126	3.7×10^{-3}	7.0×10^{-3}
single	1	1	63	1.4×10^{-3}	2.2×10^{-3}
single	1	1	126	1.3×10^{-3}	1.7×10^{-3}
single	1	8	63	4.3×10^{-4}	7.3×10^{-4}
single	1	8	126	4.4×10^{-4}	6.3×10^{-4}
double	8	1	63	5.4×10^{-11}	1.3×10^{-10}
double	8	1	126	3.5×10^{-11}	7.0×10^{-11}
double	8	8	63	5.8×10^{-12}	1.4×10^{-11}
double	8	8	126	6.4×10^{-12}	1.2×10^{-11}
double	1	1	63	2.2×10^{-12}	3.4×10^{-12}
double	1	1	126	2.1×10^{-12}	2.9×10^{-12}
double	1	8	63	7.2×10^{-13}	1.2×10^{-12}
double	1	8	126	6.9×10^{-13}	9.4×10^{-13}