AI for Science
and Its Implication for Mathematics

Weinan E

Peking University

AI for Science Institute, Beijing

Two major purposes of scientific research:

- Understanding first principles
- Solving practical problems

For practical purposes, the task of finding first principles is basically accomplished after establishing quantum mechanics.

Paul M. Dirac (1929): "The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact application of these laws leads to equations much too complicated to be soluble."

What remains to be done is to solve the mathematical problems that describe these first principles.

First principles

- Newtonian mechanics: Newton's equations
- Gas dynamics: Euler's equations and Boltzmann equation
- Elasticity theory: Lame's equations
- Fluid mechanics: Navier-Stokes equations
- Electromagnetism: Maxwell equations
- Quantum mechanics: Schrödinger equations

They provide the first principles for essentially all of natural sciences and engineering.

They are in the form of differential equations, actually mostly PDEs.

Therefore solving PDEs has been a major theme in computational science and engineering.

Three different phases

- initially (until the 50's): deriving and evaluating formulas
- discretizing PDEs directly: finite difference, finite elements, spectral methods, ...
- multi-scale modeling: dealing with the physical process and principles directly

This has had enormous success!

- structural mechanics, weather forecasting, all kinds of engineering
- basic tool for engineering science

It is the foundation for modern technology!

Many problems remain unsolved from first principles

- material properties and design
- drug design
- combustion engines
- optimal control of complex systems
- ......

These still largely rely on experiences, experiments, trial and error.

A common source of difficulty: Too many degrees of freedom

The quantum many-body problem:

$$-\Delta\psi + V\psi = E\psi$$

where ψ is the wavefunction.

number of degrees of freedom $= 3 \times$ the number of electrons

Curse of dimensionality (CoD)

As dimensionality grows, computational cost grows exponentially fast.

Reason: Polynomials, piecewise polynomials, wavelets, et al. are no longer effective in high dimension.

- too many monomials in high dimension (more then 1 million degree 10 monomials when $d = 10$).
- the mesh always looks too coarse

We have to settle with crude approximations such as the Hartree approximation:

$$f(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_d) \sim f_1(\boldsymbol{x}_1) \cdots f_d(\boldsymbol{x}_d)$$

This is exactly where deep learning can help!

Machine learning can do amazing things

Recognizing images better than average humans

- Given a set of "labeled" images ("label" = the content of the image), find an algorithm that can automatically tell us the content of similar images.

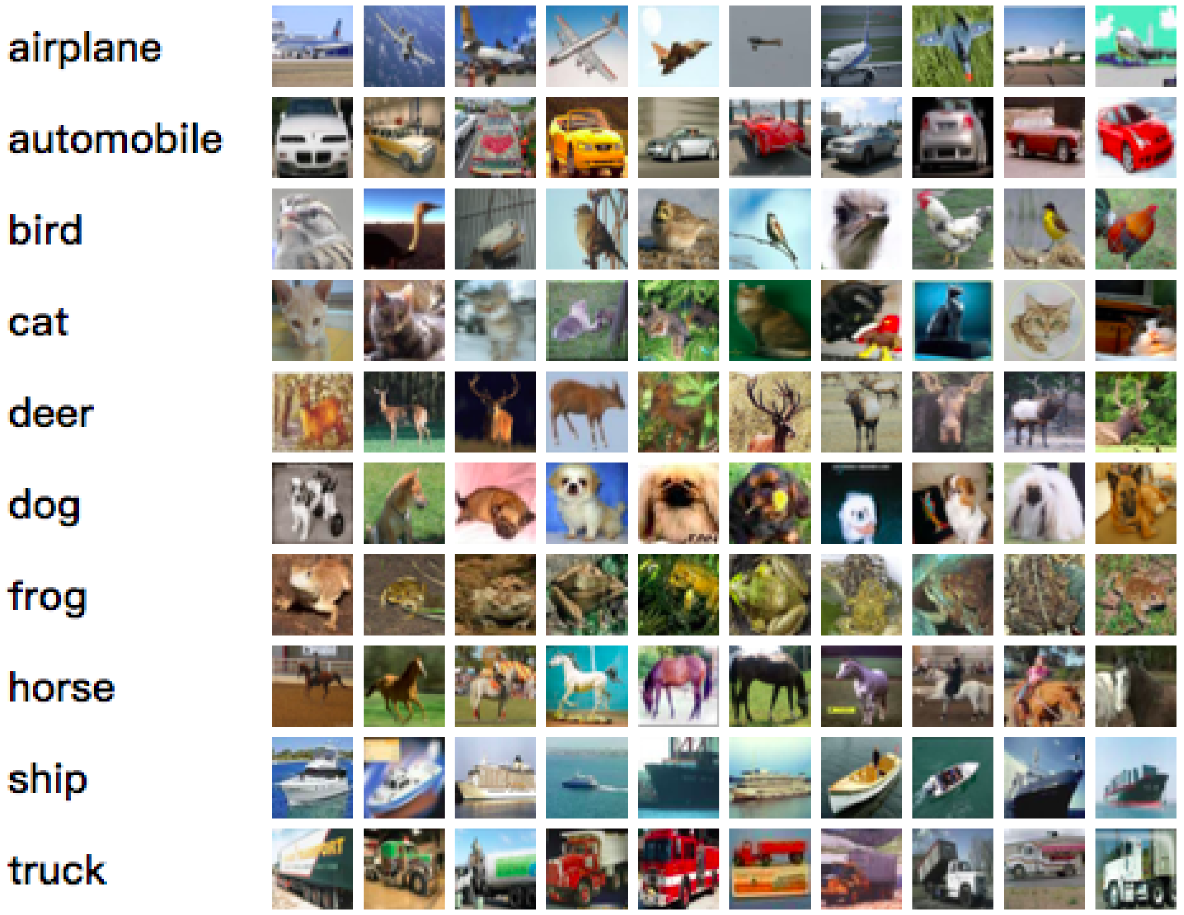


Figure: The Cifar-10 dataset: Each image is assigned a label from the 10 different categories

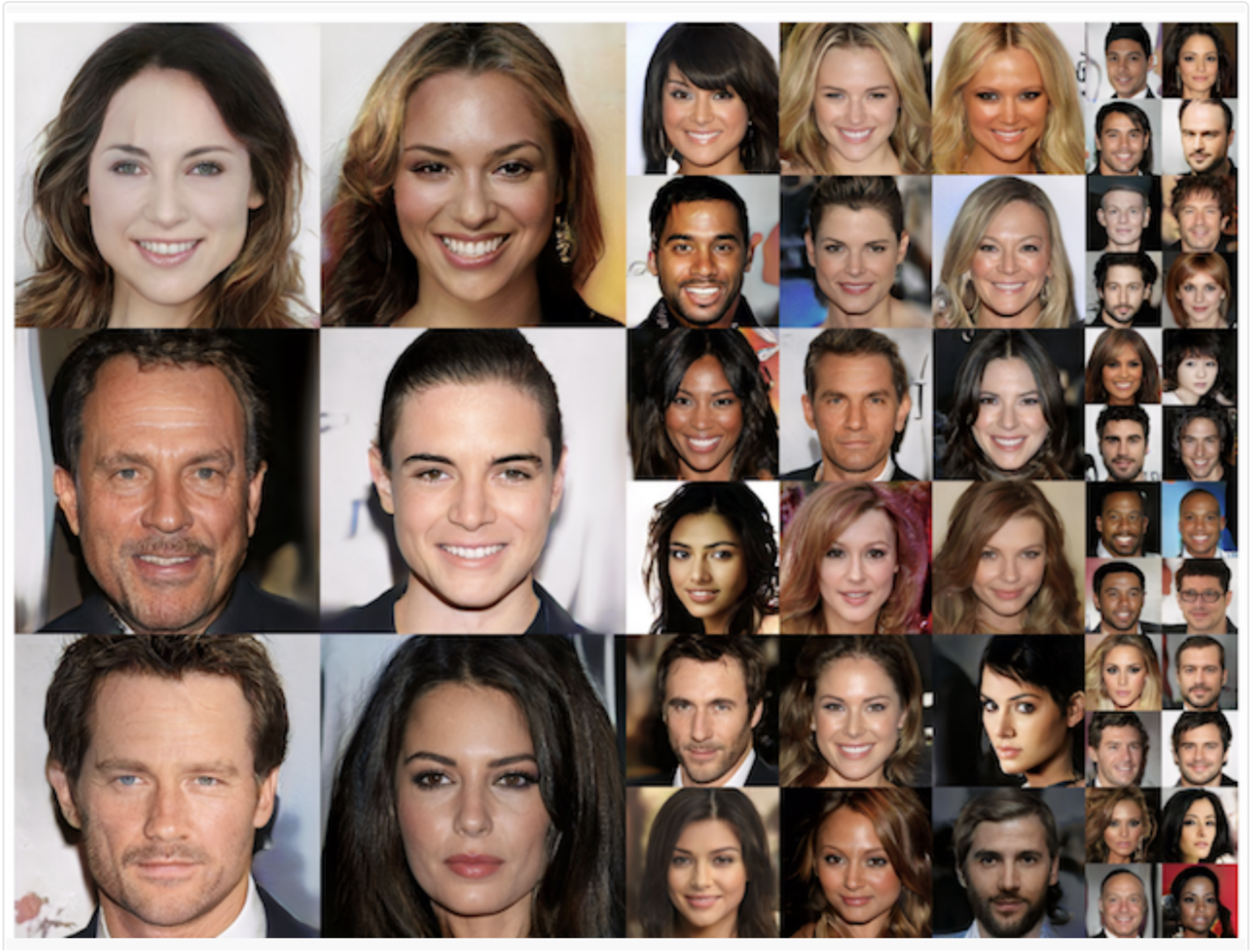
`https://www.cs.toronto.edu/~kriz/cifar.html`

AlphaGo: Playing Go game better than the best humans!

It was done purely by machine learning!

https://www.bbc.com/news/technology-35761246

Generating non-existing data: Pictures of FAKE human faces



https://arxiv.org/pdf/1710.10196v3.pdf

In essence, what's done in all these examples is to solve some standard mathematical problem.

- Image classification: Approximating the function

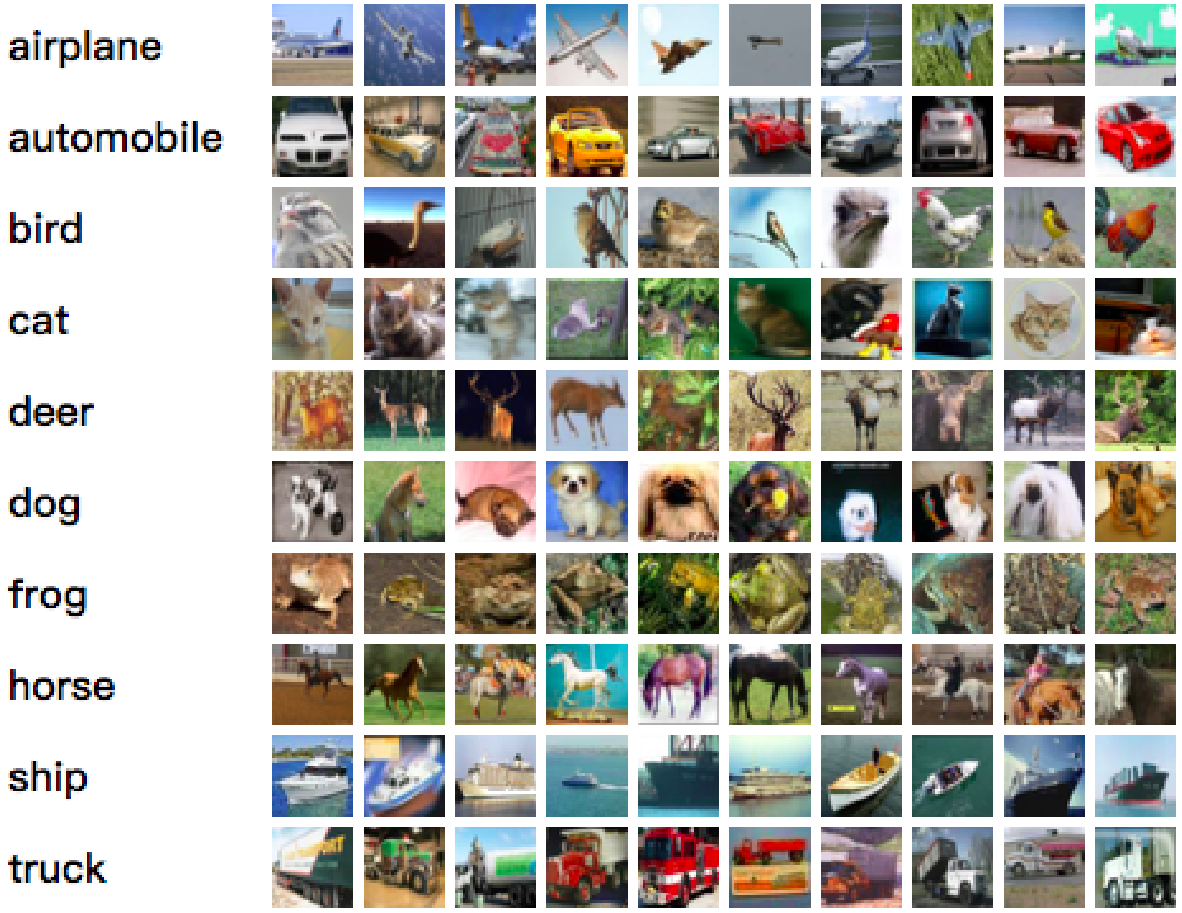
$$f^* : \text{image} \to \text{ its } \text{ content (category)}$$

- Generating pictures of fake human faces: Approximating and sampling an unknown probability distribution, where the random variable is the picture of human faces
- AlphaGo: Solving the Bellman equation in dynamic programming.

These are all classical problems in numerical analysis!

But there is a key difference: Dimensionality!

Dimensionality of the CIFAR-10 problem



Input dimension:

$$d = 32 \times 32 \times 3 = 3072$$

Key observation:

Unlike polynomials, neural networks seem to be able to approximate high dimensional functions much more efficiently.

This is far-reaching, since functions are the most basic objects in mathematics!

Attacking problems in high dimension

- Mathematical theory for neural network-based machine learning
- Scientific computing problems (e.g. PDEs) in high dimension
- Multi-scale modeling

1. Mathematical theory

- d = dimensionality of the problem
- m = the total number of free parameters in the model

Classical approximation theory (say approximating by piecewise linear functions):
mesh-size $h \sim m^{-1/d}$

$$|f^* - f_m| \sim h^2 |\nabla^2 f^*| \sim m^{-2/d} |\nabla^2 f^*|$$

To reduce the error by a factor of 10, we need to increase m by a factor of $10^{d/2}$.

True for all classical algorithms, e.g. approximating functions using polynomials, or wavelets.

Approximation theory for neural networks

- For random feature models, two-layer neural network model and residual neural network model,
$$\|f^* - f_m(\cdot;\theta)\|_{L^2(\mu)} \lesssim \frac{\|f^*\|_{\mathcal{B}}}{\sqrt{m}}.$$
where the norm $\|\cdot\|_{\mathcal{B}}$ depends on the choice of the model.

  "Monte Carlo"-like convergence rate
- For multi-layer neural network models,
$$\|f^* - f_m(\cdot;\theta)\|_{L^2(\mu)} \lesssim \frac{\|f^*\|_{\mathcal{B}}}{\sqrt{m^{\alpha}}}.$$
where α depends on the number of layers but not on d.

The picture is far from being complete

- convergence of training algorithms, and the test accuracy
  In fact, one can show that gradient descent-based training for these spaces MUST suffer from CoD.
- other settings, such as learning probability distributions, reinforcement learning, etc

We have learned much more than people seem to realize.

2. Using deep learning to solve high dimensional problems in scientific computing

Stochastic control

- Dynamic model:

$$\boldsymbol{z}_{l+1} = \boldsymbol{z}_l + \mathbf{g}_l(\boldsymbol{z}_l, \boldsymbol{a}_l) + \xi_l,$$

  where $\boldsymbol{z}_l$ = state, $\boldsymbol{a}_l$ = control, ξ_l = noise.

- Objective function:

$$\min_{\{\boldsymbol{a}_l\}_{l=0}^{T-1}} \mathbb{E}_{\{\xi_l\}} \Big\{ \sum_{l=0}^{T-1} c_l(\boldsymbol{z}_l, \boldsymbol{a}_l) + c_T(\boldsymbol{z}_T) \Big\},$$

  where $\{c_l\}$ are the running cost, c_T is the terminal cost.

- Look for a feedback control:

$$\boldsymbol{a}_l = \boldsymbol{a}_l(\boldsymbol{z}).$$

The standard approach via solving the Bellman equation suffers from CoD!

Jiequn Han and E (2016)

Why choosing this as the first example?

There is a close analogy between stochastic control and ResNet-based deep learning.

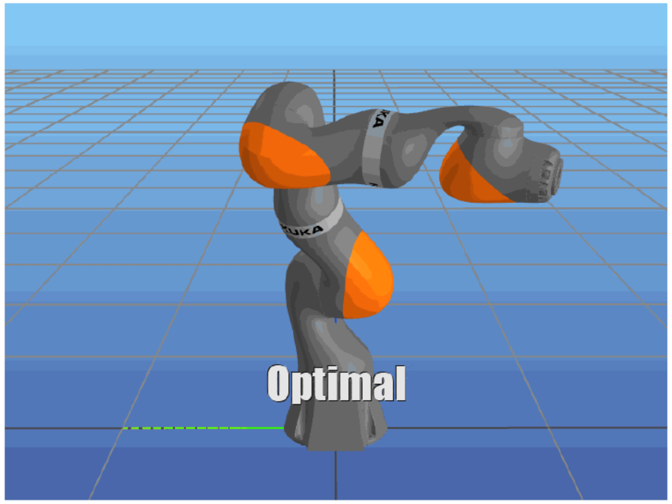
Machine learning approximation:

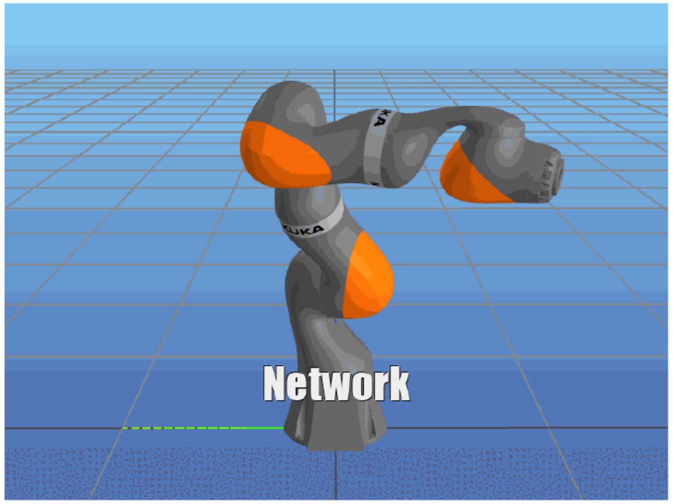
$$\boldsymbol{a}_l(\boldsymbol{z}) = f(\boldsymbol{z}, \theta_l)$$

	ResNet	Stochastic Control
model	$\boldsymbol{z}_{l+1} = \boldsymbol{z}_l + \sigma(W_l \boldsymbol{z}_l)$	$\boldsymbol{z}_{l+1} = \boldsymbol{z}_l + \boldsymbol{g}_l(\boldsymbol{z}_l, f(\boldsymbol{z}_l, \theta_l)) + \xi_l$
loss	$\mathbb{E}\Vert W_L \boldsymbol{z}_L - f^*\Vert^2$	$\mathbb{E}\{\sum c_l(\boldsymbol{z}_l, f(\boldsymbol{z}_l, \theta_l)) + c_T(\boldsymbol{z}_T)\}$
data	$\{(\boldsymbol{x}_j, y_j)\}$	$\xi_0, \ldots, \xi_{T-1}$ (noise)
optimization	SGD	SGD

Table: Analogy between ResNet and stochastic control

The reaching problem on a 7-DoF torque-controlled manipulator, the KUKA LWR iiwa R820 14 with 14-dimension state variable and 7-dimension control variable.





Work of Zhang, Long, Hu, E and Han (2022)

2. Nonlinear parabolic PDEs

$$\frac{\partial u}{\partial t} + \frac{1}{2}\Delta u + \mu \cdot \nabla u + f\big(\nabla u\big) = 0, \quad u(T, \boldsymbol{x}) = g(\boldsymbol{x})$$

Reformulate as a stochastic control problem using backward stochastic differential equations (BSDE, Pardoux and Peng (1990))

$$\inf_{Y_0, \{Z_t\}} \mathbb{E}|g(X_T) - Y_T|^2,$$

$$s.t. \quad X_t = X_0 + \int_0^t \mu(s, X_s)\ ds + \int_0^t dW_s,$$

$$Y_t = Y_0 - \int_0^t f(Z_s)\, ds + \int_0^t (Z_s)^{\mathrm{T}}\, dW_s.$$

The unique minimizer is the solution to the PDE with:

$$Y_t = u(t, X_t) \qquad \text{and} \qquad Z_t = \nabla u(t, X_t).$$

E, Han and Jentzen (Comm Math Stats, 2017); Han, Jentzen and E (PNAS, 2018)

LQG (linear quadratic Gaussian) for $d = 100$ with the cost $J = \mathbb{E}(\int_0^T \|\mathbf{m}_t\|_2^2 \, dt + g(X_T))$

$$dX_t = 2\sqrt{\lambda}\,\mathbf{m}_t \, dt + \sqrt{2}\, dW_t,$$

Hamilton-Jacobi-Bellman equation:

$$\partial_t u + \Delta u - \lambda \|\nabla u\|_2^2 = 0, \; u(T, \boldsymbol{x}) = g(\boldsymbol{x})$$

Using Hopf-Cole transform, one obtains the solution:

$$u(t, \boldsymbol{x}) = -\frac{1}{\lambda} \ln\Big(\mathbb{E}\Big[\exp\Big(-\lambda g(\boldsymbol{x} + \sqrt{2} W_{T-t})\Big)\Big]\Big).$$

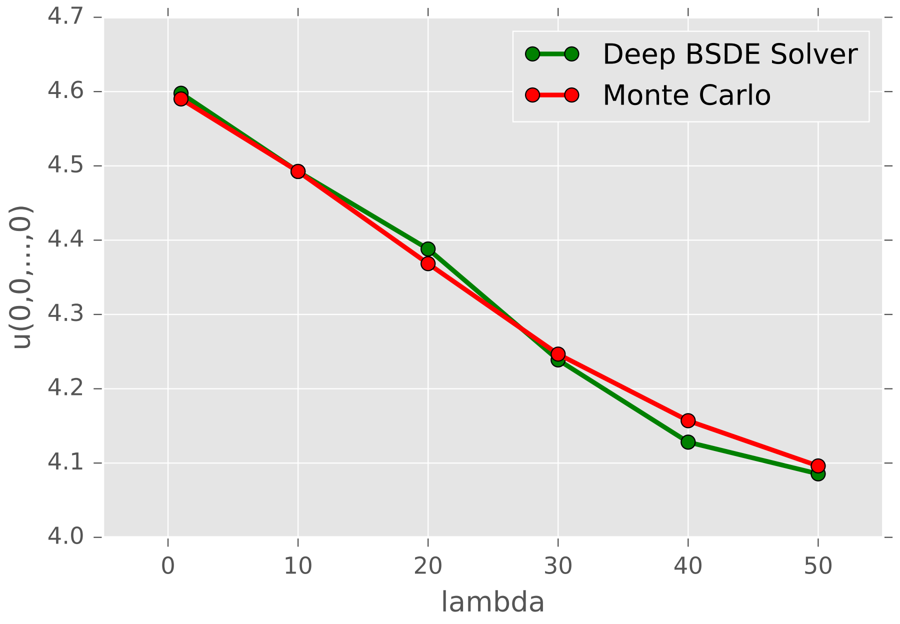


Figure: Optimal cost $u(t{=}0, \boldsymbol{x}{=}(0, \ldots, 0))$ for different values of λ.

3. Deep learning-based algorithms for multi-scale modeling

DeePMD: Molecular dynamics with *ab initio* accuracy

Modeling the dynamics of atoms in a material or molecule using Newton's equation:

$$m_i \frac{d^2 \boldsymbol{x}_i}{dt^2} = -\nabla_{\boldsymbol{x}_i} V, \quad V = V(\boldsymbol{x}_1, ..., \boldsymbol{x}_N),$$

Key question: $V = ?$ The origin of V lies in quantum mechanics (QM).

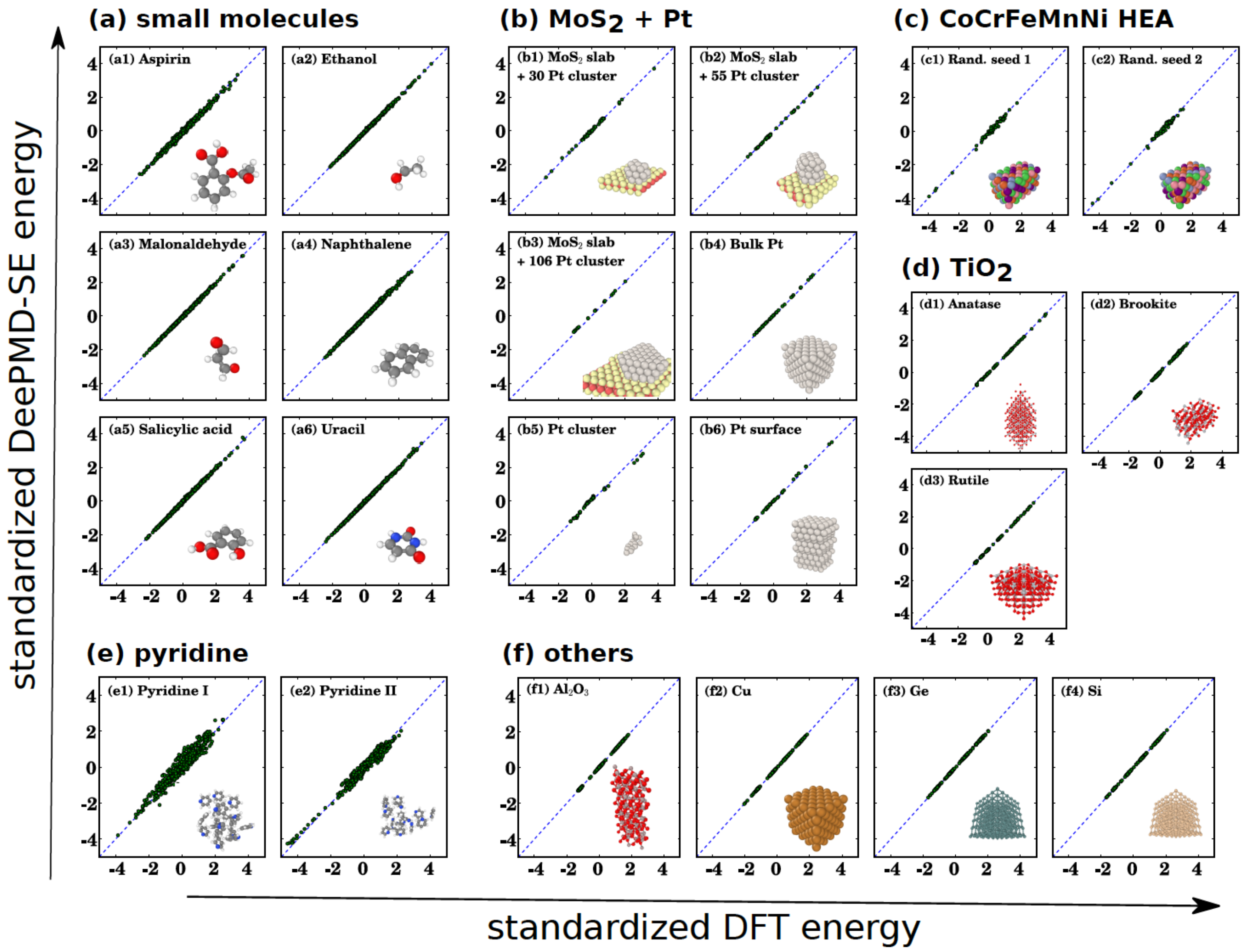
- Empirical potentials: basically guess what V should be.
  Unreliable.
- Compute the forces on the fly using QM models (Car and Parrinello (1985)).
  As reliable as the QM model but expensive (limited to about 1000 atoms).

New paradigm:

- use QM to supply the data
- use neural network model to find accurate approximation of V

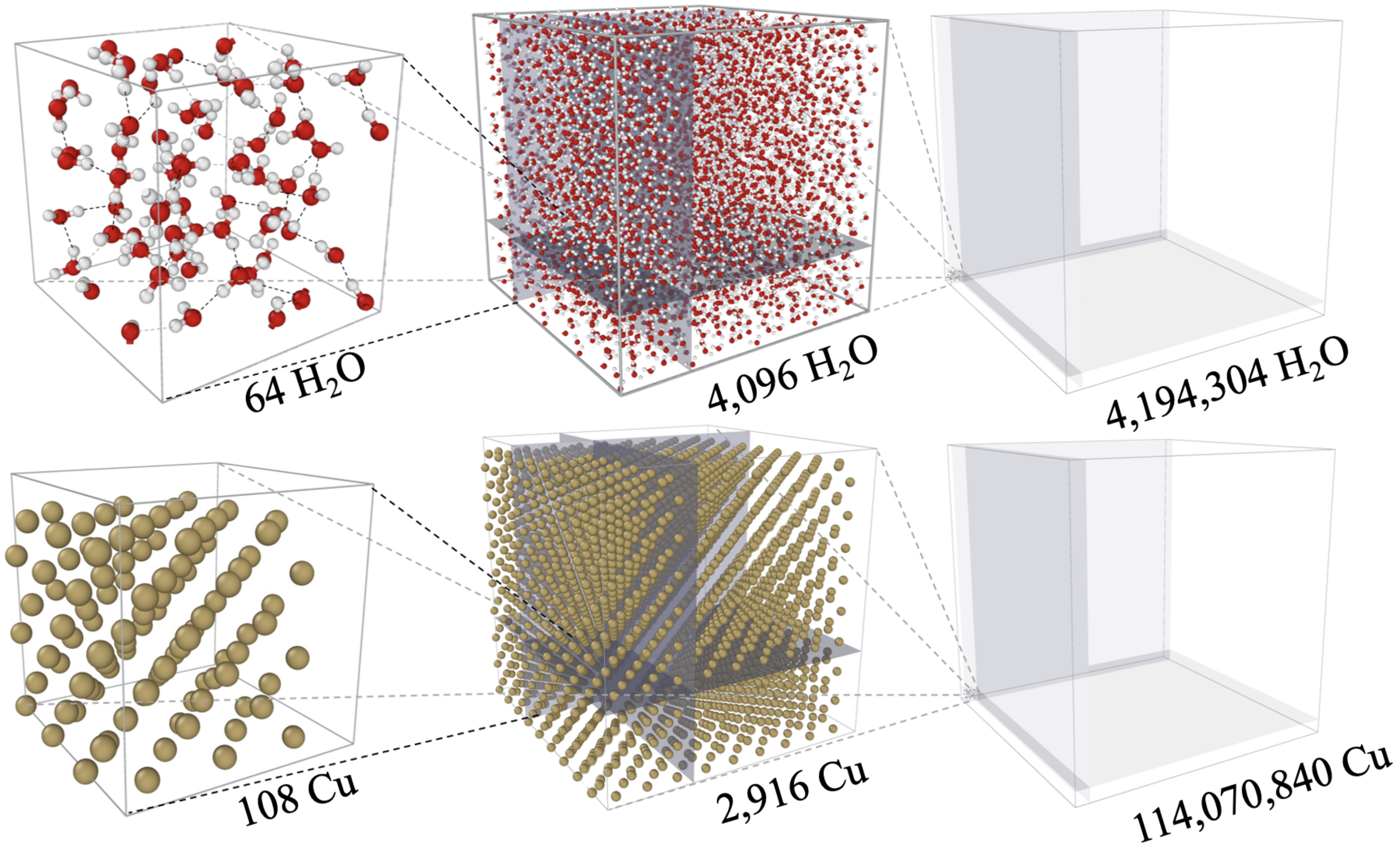
Behler and Parrinello (2007), Jiequn Han et al (2017), Linfeng Zhang et al (2018).

Accuracy comparable to QM for a wide range of materials and molecules



Linfeng Zhang, Jiequn Han, et al (2018)

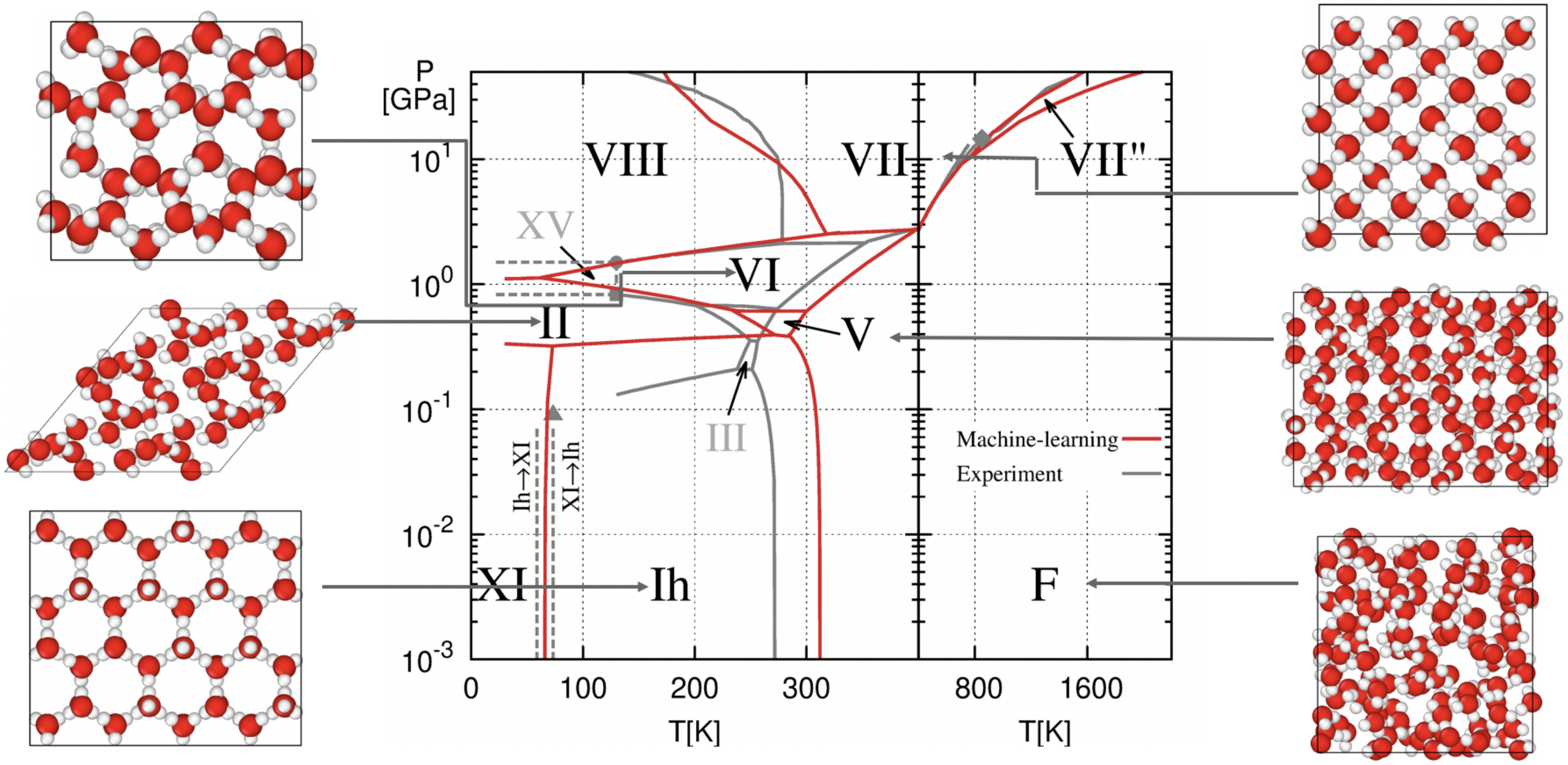
DeePMD simulation of 100M atoms with *ab initio* accuracy



(a) AIMD + HPC; (b) DeePMD+1 GPU @ Home; (c) DeePMD+27360 GPUs @ Summit.

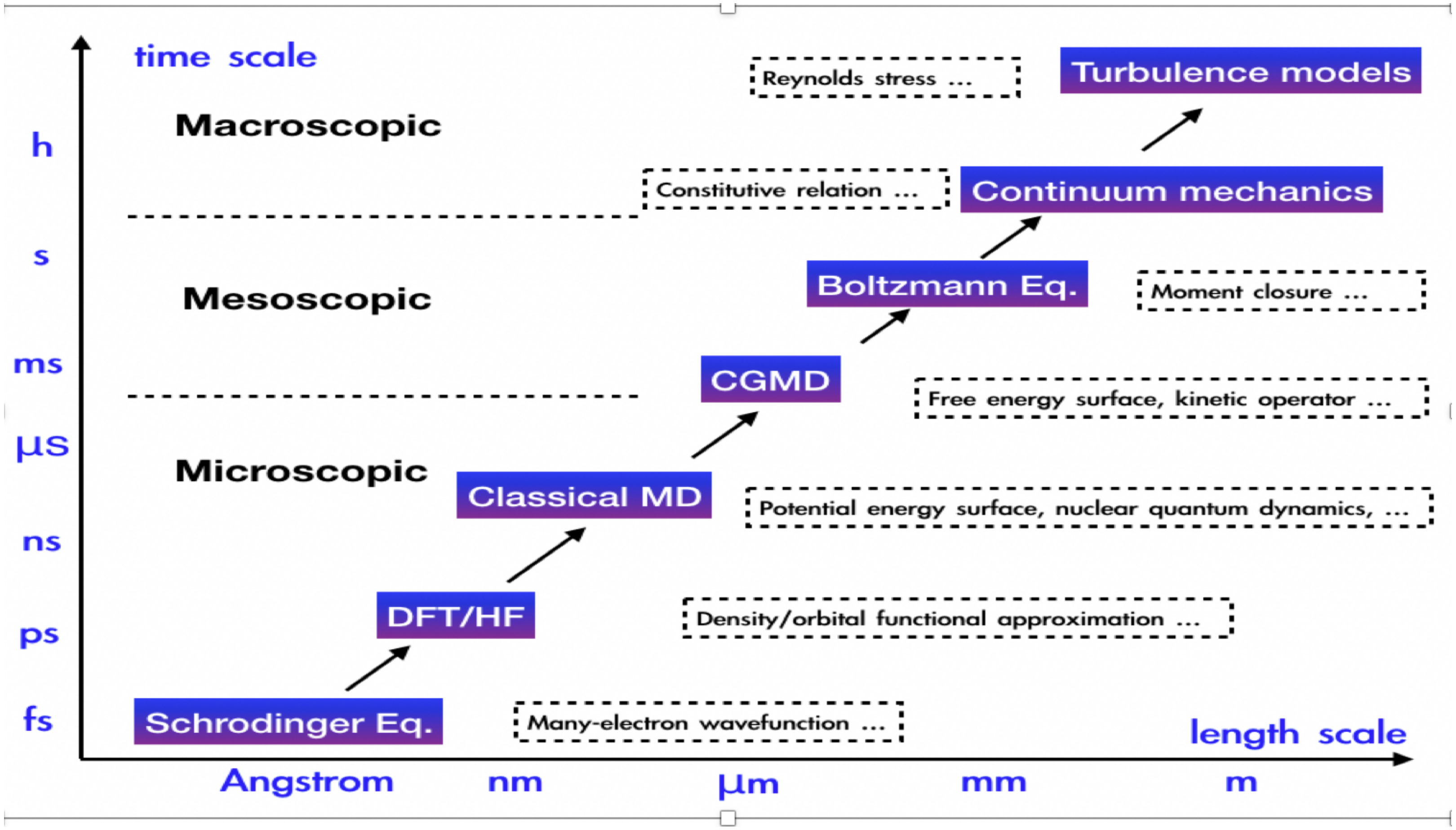
Weile Jia, et al, SC20, 2020 ACM Gordon Bell Prize

Phase diagram of water



Linfeng Zhang, Han Wang, et al. (2021)

The hierarchy of physical models



AI for Science: Making first principles truly reliable and useful

Machine learning provides the missing tool:

- *Quantum many-body problem*: RBM (2017), DeePWF (2018), FermiNet (2019), PauliNet (2019), ...
- *Density functional theory*: DeePKS (2020), NeuralXC (2020), DM21 (2021), ...
- *Molecular dynamics*: DeePMD (2018), ...
- *Coarse-grained molecular dynamics*: DeePCG (2019)
- *Kinetic equation*: machine learning-based moment closure (Han et al. 2019)
- *Continuum mechanics*: DeePN^2 (2020)
- ......

With applications in:

- New tools for drug design
- More systematic ways of battery design
- New ways to model combustion engine
- New ways to perform inversion using spectroscopy and other experimental tools

What about solving low dimensional PDEs?

Traditional methods: Finite difference, finite elements, etc

- accuracy can be systematically improved
- still not easy for complex problems such problems with complex geometry

We will see that machine learning-based algorithms behave in an opposite way.

Machine learning-based algorithms for PDEs

- Variational form: **Deep Ritz Method** (DRM, E and Yu (2018))
- Strong form (least squares):
  Physics-Informed Neural Networks (PINN, Raissi et al. (2019)),
  Deep Galerkin Method (DGM, Sirignano and Spiliopoulos (2018))
- Weak form (test function): **Weak Adversarial Network** (WAN, Bao et al. (2020))

PINN and DGM

- very general
- uses least square formulation for both the PDE and the boundary condition, $m = n$ is no longer necessary
- mesh-free, so easier for complex geometry
- accuracy cannot be systematically improved, since the optimization problem is highly non-convex

Bridging classical and ML-based algorithms: The random feature method

- Keeping least square formulation using collocation points.
- Replacing NNs by *multi-scale* random feature models (RFM)

Random feature model vs. (two-layer) NNs.

$$u_M(\boldsymbol{x}) = \sum_{m=1}^{M} u_m \sigma(\boldsymbol{k}_m \cdot \boldsymbol{x} + b_m)$$

where σ is the activation function.

- random feature model: inner parameters are fixed (but random). The model is linear and training is much easier.
- two-layer NNs: the inner parameters are also trained. The model is more adaptive but training is much harder.

See also "extreme learning machines" (Huang et al. (2006)).

Multi-scale random feature models

- fine scale features captured using a partition of unity, with an independent RFM for each component in the partition
- macro-scale features captured using a global RFM

$$u_m(\boldsymbol{x}) = u_g(\boldsymbol{x}) + \sum_{k=1}^{m_p} \psi_k(\boldsymbol{x}) \sum_{j=1}^{J_k} u_{nj}\phi_{kj}(\boldsymbol{x})$$

Similarity with "local extreme machines" (Dong et al. (2021)).
Difference: (1) the way locality is enforced; (2) loss function.

Accuracy can be systematically improved

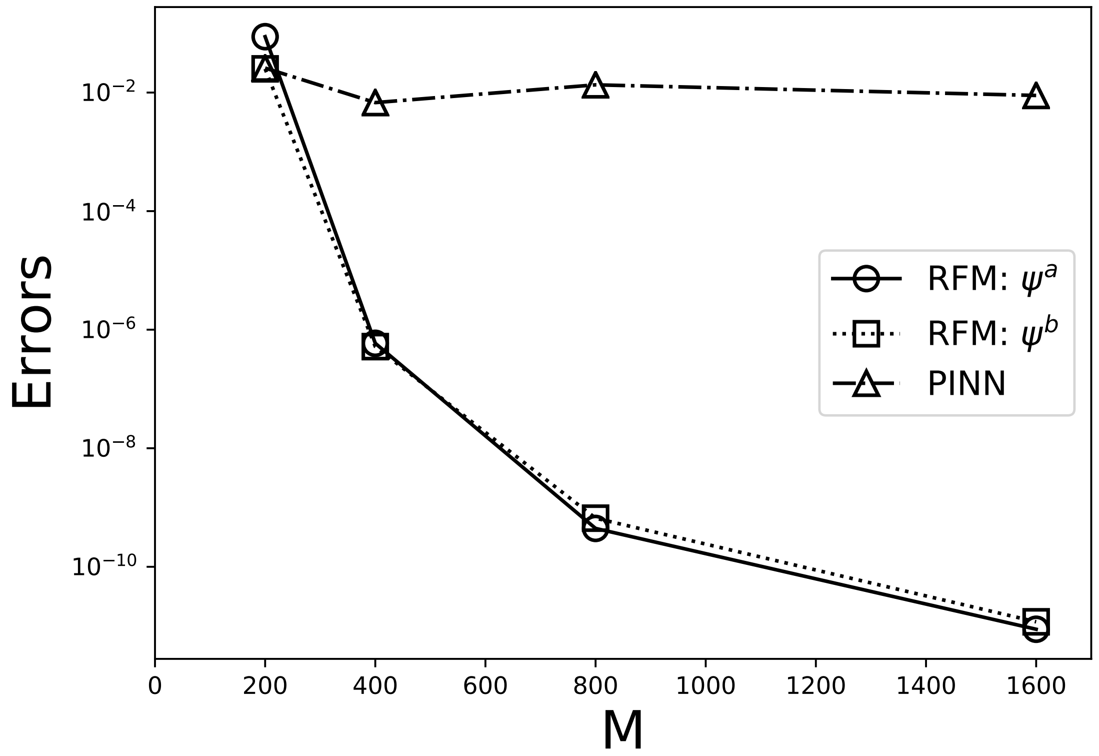
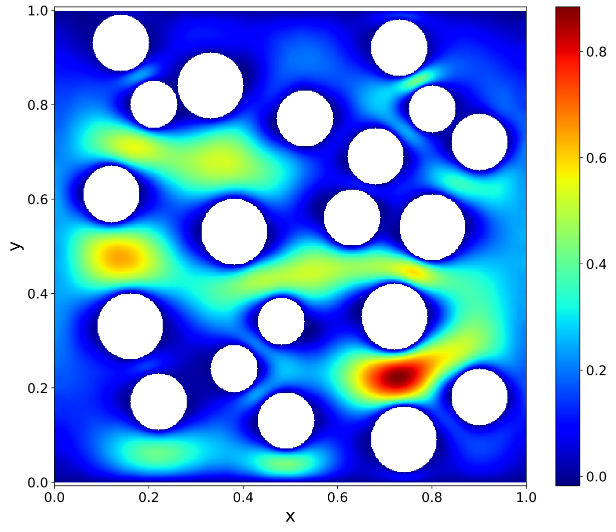


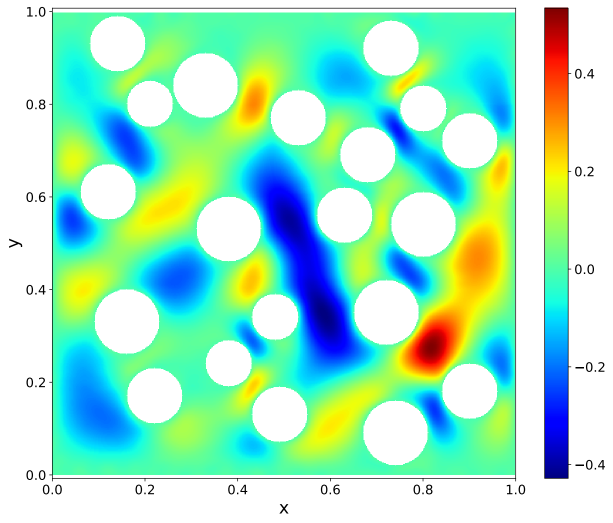
Figure: Convergence of RFM and PINN for Helmholtz equation in the semi-log scale

Easy to handle complex geometry: Stokes flow

$$(u, v)|_{\partial\Omega} = \begin{cases} (y(1-y), 0) & \text{if } x = 0 \\ (y(1-y), 0) & \text{if } x = 1 \\ (0, 0) & \text{otherwise} \end{cases}$$



(a) u



(b) v

Figure: Velocity field (u, v) generated by the random feature method

Two-dimensional elasticity problem with a complex geometry

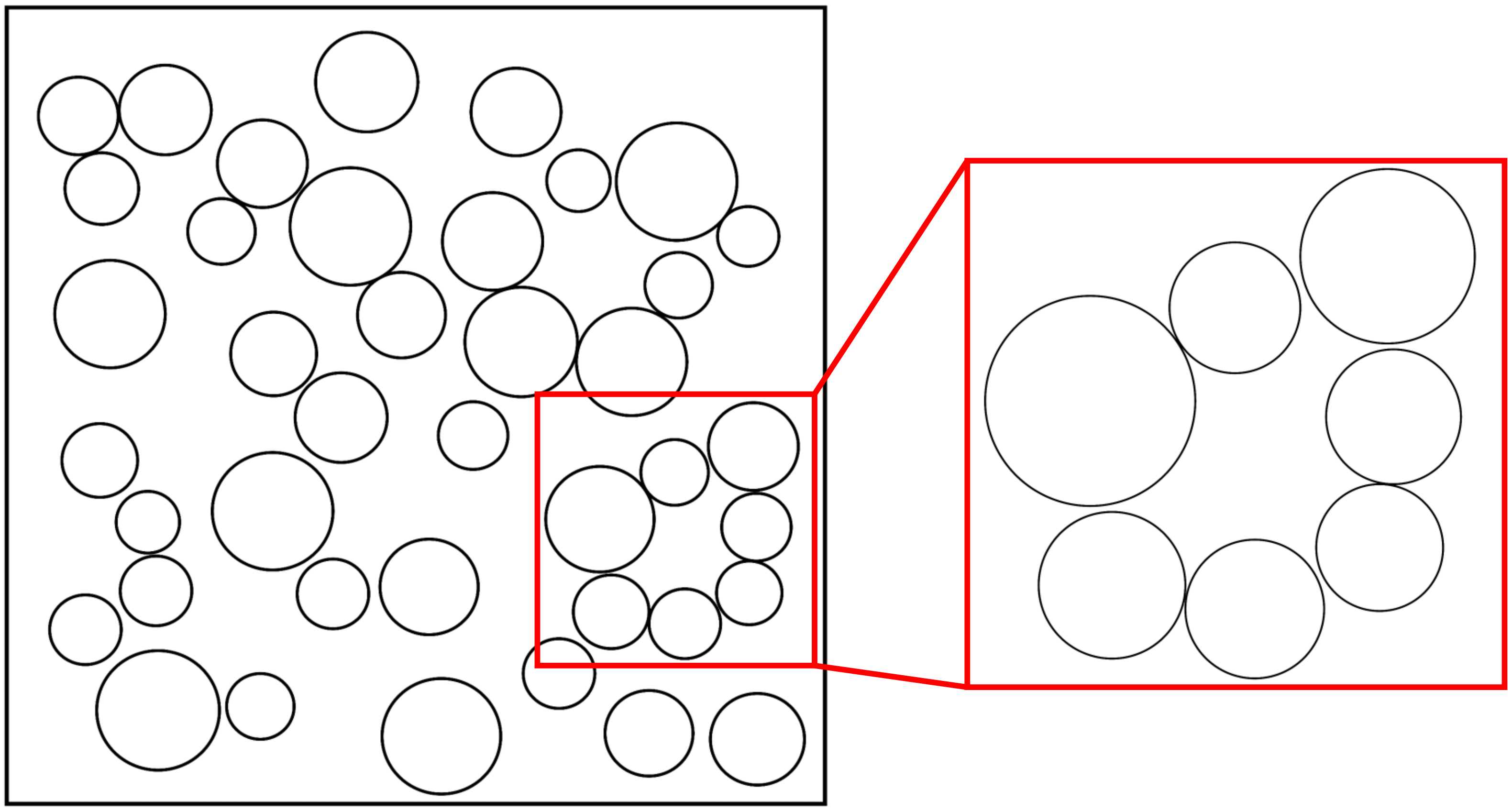
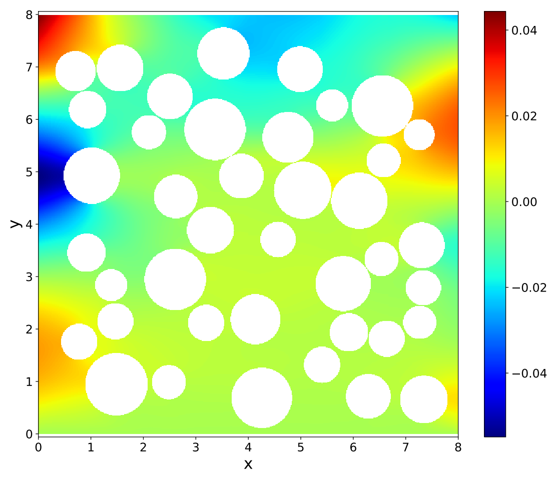
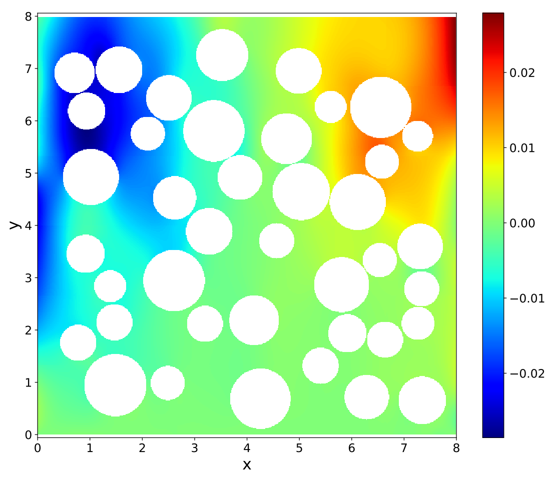
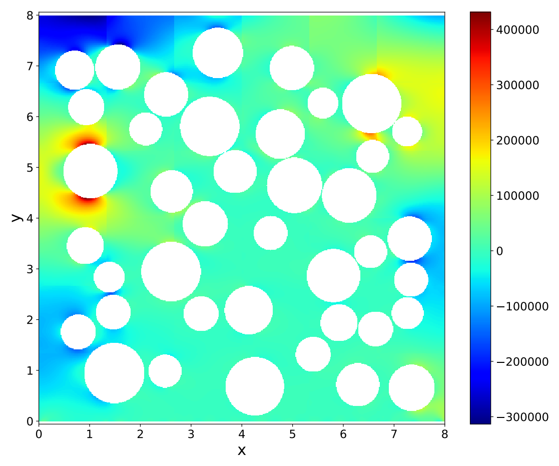
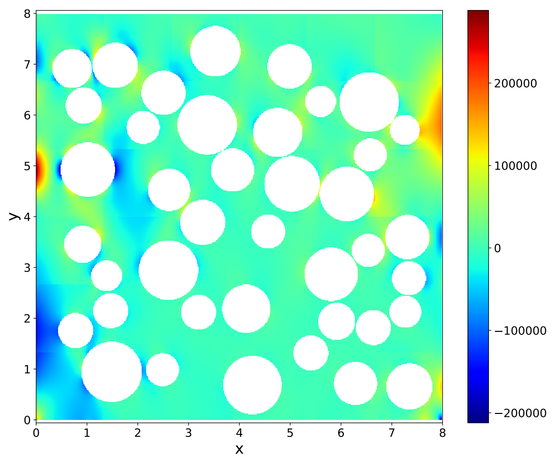


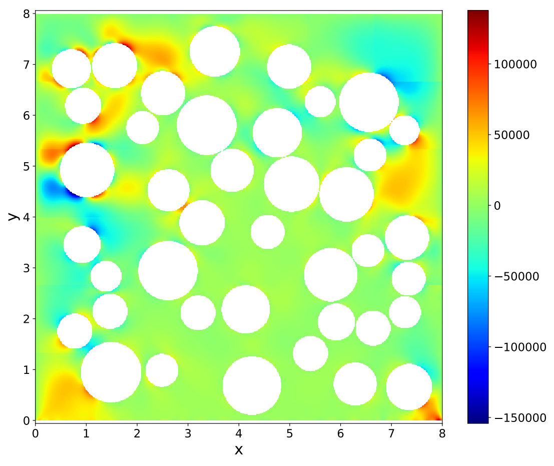
Figure: Complex domain with a cluster of holes that are nearly touching


(a) u


(b) v


(c) σ_x


(d) σ_y


(e) τ_{xy}

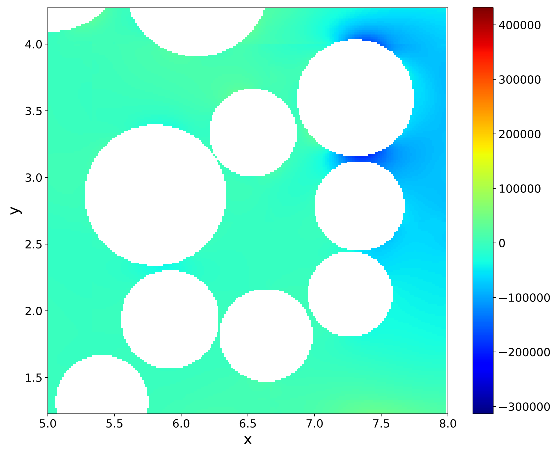

(f) σ_x over a cluster

Figure: Numerical solution by the random feature method for the two-dimensional elasticity problem over a complex geometry

Promising but still a lot to be understood

Jingrun Chen et al., *Bridging Traditional and Machine Learning-based Algorithms for Solving PDEs: The Random Feature Method*, **Journal of Machine Learning**, vol 1. No 3. pp. 268-298.

- What is the best way to select the basis functions? (Random basis seems to perform better, but why?)
  PINN/DGM is fully adaptive
  RFM uses preselected basis functions.
  The optimal solution must be somewhere in-between.
- Least squares formulation makes the iterative training process very slow.
- What is the best (convenient, robust, accurate) way to describe 3-D geometry?

Summary

- Deep neural networks seem to be able to approximate high dimensional functions efficiently, and this opens doors to a lot of new possibilities.
- "**AI for Science**" will change the way we do science in a fundamental way.
- Even for very well-studied problems such as solving PDEs, ML might change the way we do things.
- It makes the task of connecting with real applications much easier.

Thanks you for your attention!