北京国际数学研究中心
BEIJING INTERNATIONAL CENTER FOR MATHEMATICAL RESEARCH

北京大学人工智能研究院
Academy for Artificial Intelligence, Peking University

北京大学未来技术学院
COLLEGE OF FUTURE TECHNOLOGY PEKING UNIVERSITY

大数据分析与应用技术国家工程实验室

Learning and Learning to Solve PDEs

Bin Dong (董彬)
Peking University

OUTLINE

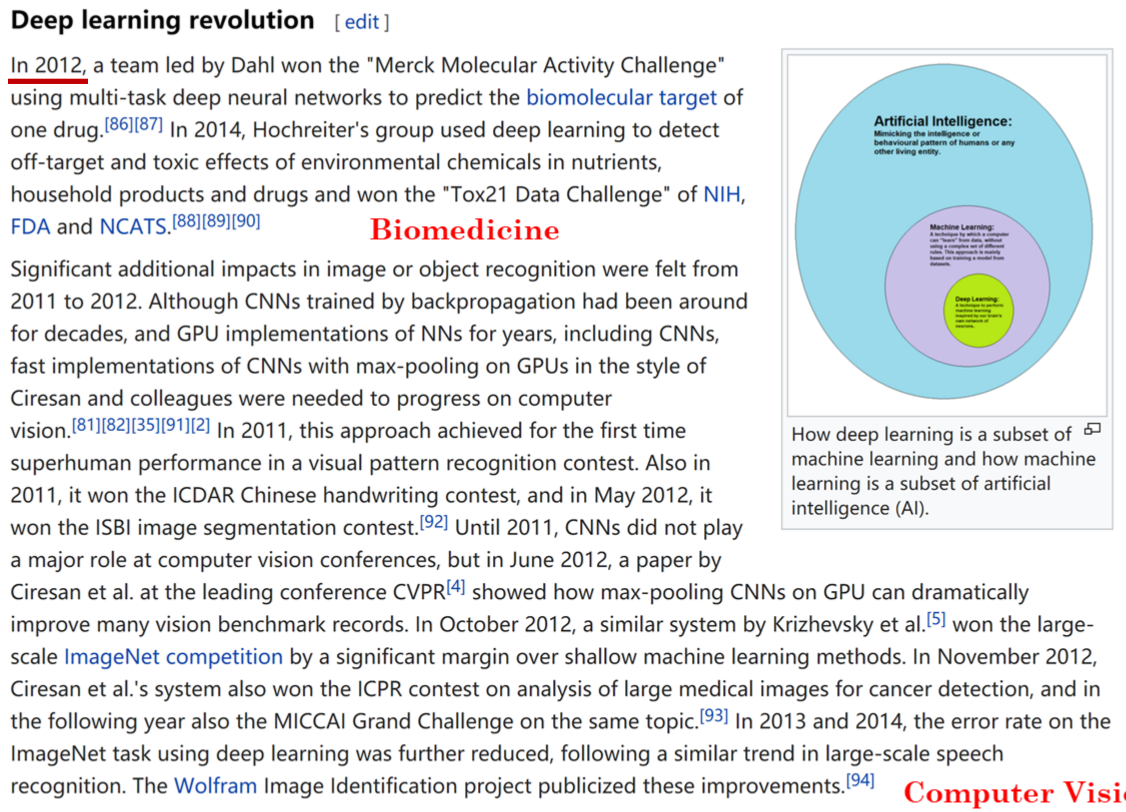
- Overview and Motivations
  - Deep Learning and Optimal Control
  - Exploring Structural Similarities between PDEs/ODEs and CNNs
- Learning PDEs (Model Reduction)
  - Learning Moment Closure for Boltzmann-BGK Equation
- Learning to Solve PDEs
  - Solving Linear Parametric PDEs: A Meta-Learning Approach
  - Electromagnetic Simulation
    - Joint work with Huawei MindSpore AI + Scientific Computing team
- Conclusions and Future Work

OVERVIEW

Motivations and Intuitions

Deep Learning Revolution

- It changes the research landscape of machine learning and artificial intelligence in general.



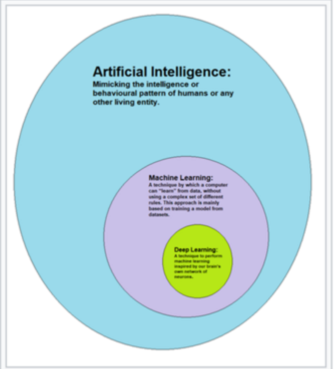
Deep learning revolution [edit]

In 2012, a team led by Dahl won the "Merck Molecular Activity Challenge" using multi-task deep neural networks to predict the biomolecular target of one drug.[86][87] In 2014, Hochreiter's group used deep learning to detect off-target and toxic effects of environmental chemicals in nutrients, household products and drugs and won the "Tox21 Data Challenge" of NIH, FDA and NCATS.[88][89][90]

Biomedicine

Significant additional impacts in image or object recognition were felt from 2011 to 2012. Although CNNs trained by backpropagation had been around for decades, and GPU implementations of NNs for years, including CNNs, fast implementations of CNNs with max-pooling on GPUs in the style of Ciresan and colleagues were needed to progress on computer vision.[81][82][35][91][2] In 2011, this approach achieved for the first time superhuman performance in a visual pattern recognition contest. Also in 2011, it won the ICDAR Chinese handwriting contest, and in May 2012, it won the ISBI image segmentation contest.[92] Until 2011, CNNs did not play a major role at computer vision conferences, but in June 2012, a paper by Ciresan et al. at the leading conference CVPR[4] showed how max-pooling CNNs on GPU can dramatically improve many vision benchmark records. In October 2012, a similar system by Krizhevsky et al.[5] won the large-scale ImageNet competition by a significant margin over shallow machine learning methods. In November 2012, Ciresan et al.'s system also won the ICPR contest on analysis of large medical images for cancer detection, and in the following year also the MICCAI Grand Challenge on the same topic.[93] In 2013 and 2014, the error rate on the ImageNet task using deep learning was further reduced, following a similar trend in large-scale speech recognition. The Wolfram Image Identification project publicized these improvements.[94]

Computer Vision


Artificial Intelligence:
Mimicking the intelligence or behavioural pattern of humans or any other living entity.
Machine Learning:
A technique by which a computer can "learn" from data, without using a complex set of different rules. This approach is mainly based on training a model from datasets.
Deep Learning:
A technique to perform machine learning inspired by our brain's own network of neurons.

How deep learning is a subset of machine learning and how machine learning is a subset of artificial intelligence (AI).

- Its impact has now spread to many fields in natural science, social science and engineering
- This talk's focus: ML + Scientific Computing

Challenges of Deep Learning

- Deep learning has been a great success
  - It is particularly good at fitting complex mapping and making fast inference with about right accuracy.
- However, there are still many pending issues
  - Generalization (inner workings of stochastic training)
  - Model robustness (adversarial, distribution shift)
  - Interpretability (can we learn from the model)
- A promising perspective:
  - ✓ Combine handcraft and data-driven models
  - ✓ Optimal control is a suitable framework

DEEP LEARNING FROM CONTROL PERSPECTIVE

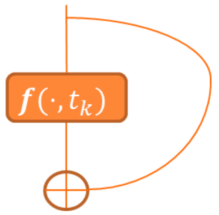
- Control Perspective:

Dynamics:

$$d\boldsymbol{X}_t = \boldsymbol{f}(\boldsymbol{X}_t, u(t))dt + \boldsymbol{\sigma}(\boldsymbol{X}_t, t)d\boldsymbol{W}_t, \quad \boldsymbol{X}_0 = \boldsymbol{x}, \quad t \in (0,1]$$

Training loss:

$$J(\boldsymbol{u}) = \mathbb{E}_{(\boldsymbol{x},y)\sim \boldsymbol{P}}\ \ell(g(\boldsymbol{X}_1), y) + \int_0^1 R\big(s, \boldsymbol{X}_s, \boldsymbol{u}(s)\big)\, ds$$



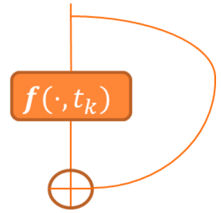
$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \boldsymbol{f}(\boldsymbol{x}^k, t_k)$$

- **ResNet = Forward-Euler ($\boldsymbol{\sigma}$ = 0)**

- Han, E, NIPS DRL Workshop, 2016
- E, CMS, 5(1):1–11, 2017.
- Haber, Ruthotto, IP, 34(1), 2017.

Deep Learning from Control Perspective

- Control Perspective:



$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \boldsymbol{f}(\boldsymbol{x}^k, t_k)$$

Dynamics:

$$\boldsymbol{X}^{k+1} = \tilde{\boldsymbol{f}}(\boldsymbol{X}^k, \boldsymbol{u}(k), \boldsymbol{w}_k), \quad \boldsymbol{X}_0 = \boldsymbol{x}, \quad 0 \le k \le K-1$$

- **ResNet = Forward-Euler ($\boldsymbol{\sigma} = 0$)**
- **Discrete ODEs = Architecture**

Training loss:

$$J(\boldsymbol{u}) = \mathbb{E}_{(\boldsymbol{x},y)\sim \boldsymbol{P}}\ \ell(g(X^K), y) + \sum_k R_k\left(\boldsymbol{X}^k, \boldsymbol{u}(k)\right)$$

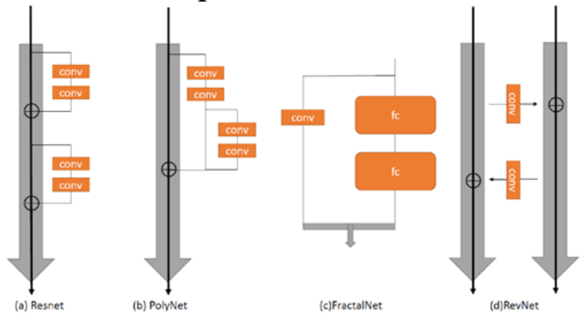
- Han, E, NIPS DRL Workshop, 2016
- E, CMS, 5(1):1–11, 2017.
- Haber, Ruthotto, IP, 34(1), 2017.
- Lu, Zhong, Li, **Dong**, ICML 2018.

Connection with stochastic optimal control

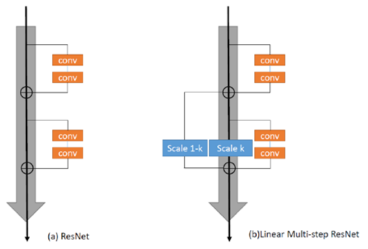
$$\min \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim \mathcal{P}}\left(\mathbb{E}(\ell(X_T, \boldsymbol{y}) + \int_0^T r(s, X_s, \theta_s))ds\right)$$

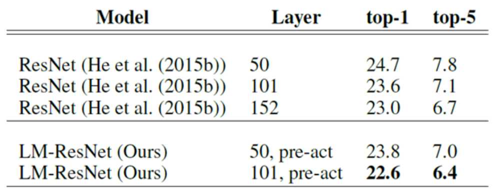
$$\text{s.t. } dX_t = f(X_t, \theta_t)dt + g(X_t, \theta_t)dB_t, \quad X_0 = \boldsymbol{x}.$$

More examples other than ResNet.



Linear 2-step method: $\boldsymbol{X}^{k+1} = (1-\alpha_n)\boldsymbol{X}^k + \alpha_n \boldsymbol{X}^{k-1} + \Delta t \cdot \boldsymbol{f}(\boldsymbol{X}^k, \theta_k)$



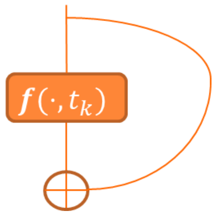


Model	Layer	top-1	top-5
ResNet (He et al. (2015b))	50	24.7	7.8
ResNet (He et al. (2015b))	101	23.6	7.1
ResNet (He et al. (2015b))	152	23.0	6.7
LM-ResNet (Ours)	50, pre-act	23.8	7.0
LM-ResNet (Ours)	101, pre-act	**22.6**	**6.4**

ImageNet (1.28m train, 50k test, 1000 classes)

Deep Learning from Control Perspective

- Control Perspective:



$$x^{k+1} = x^k + f(x^k, t_k)$$

Dynamics:

$$X^{k+1} = \tilde{f}(X^k, u(k), w_k), \quad X_0 = x, \quad 0 \le k \le K-1$$

Training loss:

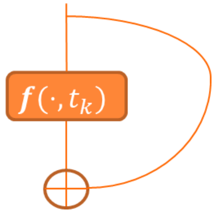
$$J(u) = \mathbb{E}_{(x,y)\sim P}\ \ell(g(X^K), y) + \sum_k R_k\left(X^k, u(k)\right)$$

- **ResNet = Forward-Euler ($\sigma = 0$)**
- **Discrete ODEs = Architecture**
- **Optimization Alg = Architecture**

- Han, E, NIPS DRL Workshop, 2016
- E, CMS, 5(1):1–11, 2017.
- Haber, Ruthotto, IP, 34(1), 2017.
- Lu, Zhong, Li, **Dong**, ICML 2018.
- Gregor, LeCun, ICML 2010.
- Yang, Sun, Li, Xu, NIPS 2016.
- Li, Tai, E, ICML 2017.
- Liu, Theodorou, arXiv:1908.10920.
- Monga, Li, Yonina, IEEE Signal Processing Magazine, 38(2), 2021.

Deep Learning from Control Perspective

- Control Perspective:



$$x^{k+1} = x^k + f(x^k, t_k)$$

Dynamics: $v_t = F(v, Dv), D = \left(\frac{\partial^\alpha}{\partial x^\alpha}\right)_{1\le|\alpha|\le n}$

$\rightsquigarrow$

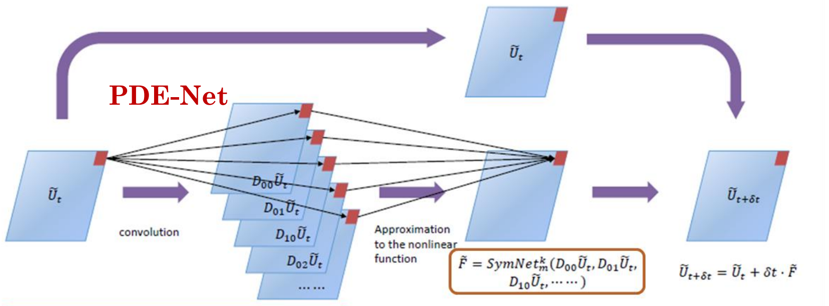
$$X^{k+1} = \tilde{f}(X^k, u(k), w_k), \quad X_0 = x, \quad 0 \le k \le K-1$$

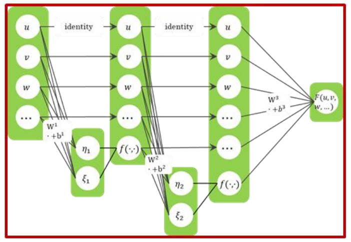
Training loss:

$$J(u) = \mathbb{E}_{(x,y)\sim P}\, \ell(g(X^K), y) + \sum_k R_k\left(X^k, u(k)\right)$$

- ResNet = Forward-Euler ($\sigma = 0$)
- Discrete ODEs = Architecture
- Optimization Alg = Architecture
- Discrete PDEs = Architecture
  - Inverse Problems
  - Solving PDEs

- Chen, Wei, Pock, CVPR 2015.
- Long, Lu, Ma, **Dong**, ICML 2018.
- Long, Lu, **Dong**, JCP, 399, 2019.
- Ray, Hesthaven, JCP, 367, 2018
- Han, Jentzen, E, PNAS, 115(34), 2018.
- Bar-Sinai, Hoyer, Hickey, Brenner,. PNAS,116(31), 2019
- Arridge, Maass, Öktem, Schönlieb, Acta Numerica, 28, 2019.
- Many others … …


PDE-Net
$\tilde{U}_t$
$\tilde{U}_t$
convolution
$D_{00}\tilde{U}_t$
$D_{01}\tilde{U}_t$
$D_{10}\tilde{U}_t$
$D_{02}\tilde{U}_t$
……
Approximation to the nonlinear function
$\tilde{F} = SymNet_m^k(D_{00}\tilde{U}_t, D_{01}\tilde{U}_t, D_{10}\tilde{U}_t, \cdots\cdots)$
$\tilde{U}_{t+\delta t}$
$\tilde{U}_{t+\delta t} = \tilde{U}_t + \delta t \cdot \tilde{F}$



Constraints on filters

$$M(q) = (m_{i,j})_{N\times N}, \text{ where } m_{i,j} = \frac{1}{(i-1)!(j-1)!}\sum_{k\in\mathbb{Z}^2} k_1^{i-1}k_2^{j-1}q[k_1,k_2]$$

Approximating $\frac{\partial}{\partial x}$ for example

$$\begin{pmatrix} 0 & 0 & \star \\ 1 & \star & \star \\ \star & \star & \star \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & \star \\ 0 & \star & \star \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

1st order learnable | 2st order learnable | 1st order frozen

Learning to Close Moment Systems of Boltzmann-BGK Equation

- Zhengyi Li, Bin Dong and Yanli Wang, *Learning Invariance Preserving Moment Closure Model for Boltzmann-BGK Equation*, arXiv:2110.03682, 2021.

Learning to Close Moment Systems of Boltzmann-BGK Equation

- Boltzmann equation describes the dynamics of a rarefied gas and has important applications in various fields of science and engineering.

$$\partial_t f + v \cdot \nabla_x f = Q(f), \qquad (t, x, v) \in [0, \infty) \times \mathbb{R}^3 \times \mathbb{R}^3$$

- Challenge: high-dimensionality
- Numerical methods for Boltzmann
  - Stochastic method: e.g., direct simulation of Monte Carlo method (DSMC)
  - Deterministic method: e.g., discretized velocity method (DVM), spectral method, moment method
- This work focuses on learning neural network based moment closure for 1D Boltzmann-BGK

Learning to Close Moment Systems of Boltzmann-BGK Equation

- 1D Boltzmann-BGK equation

$$\frac{\partial f(t,x,v)}{\partial t} + v\frac{\partial f(t,x,v)}{\partial x} = \frac{1}{Kn}[\mathcal{M}(f) - f], \qquad x \in D \subset \mathbb{R}, \quad v \in \mathbb{R}, \quad t > 0,$$

$$\mathcal{M}(f) = \frac{\rho(t,x)}{\sqrt{(2\pi\theta(t,x)}} \exp\left(-\frac{(v-u(t,x))^2}{2\theta(t,x)}\right)$$

where ρ, u, θ are density, macroscopic velocity, and temperature respectively

- The moment closure problem (following the setting of Cai, Fan, Li, CPAM 2014)

$$f(t,x,v) \approx \sum_{\alpha \leqslant M} f_\alpha(t,x)\mathcal{H}_\alpha(\xi), \qquad \xi = \frac{v-u}{\sqrt{\theta}},$$

$$\mathcal{H}_\alpha(\xi) = \frac{1}{\sqrt{2\pi}}\theta^{-\frac{\alpha+1}{2}} He_\alpha(\xi) \exp\left(-\frac{\xi^2}{2}\right),$$

$$He_\alpha(v) = (-1)^\alpha \exp\left(\frac{v^2}{2}\right)\frac{\mathrm{d}^\alpha}{\mathrm{d}v^\alpha}\exp\left(-\frac{v^2}{2}\right).$$

$$\boxed{\frac{\partial \boldsymbol{\omega}}{\partial t} + \mathbf{A}_M(\boldsymbol{\omega})\frac{\partial \boldsymbol{\omega}}{\partial x} + (M+1)\boxed{\frac{\partial f_{M+1}}{\partial x}}e_{M+1} = G(\boldsymbol{\omega})}$$

$$\boxed{\frac{\partial f_{M+1}(t,\cdot)}{\partial x} = \frac{\partial \mathcal{G}[\omega]}{\partial x} \triangleq \mathcal{G}_x[\rho(t,\cdot), u(t,\cdot), \theta(t,\cdot), f_3(t,\cdot), \cdots, f_M(t,\cdot), Kn].}$$

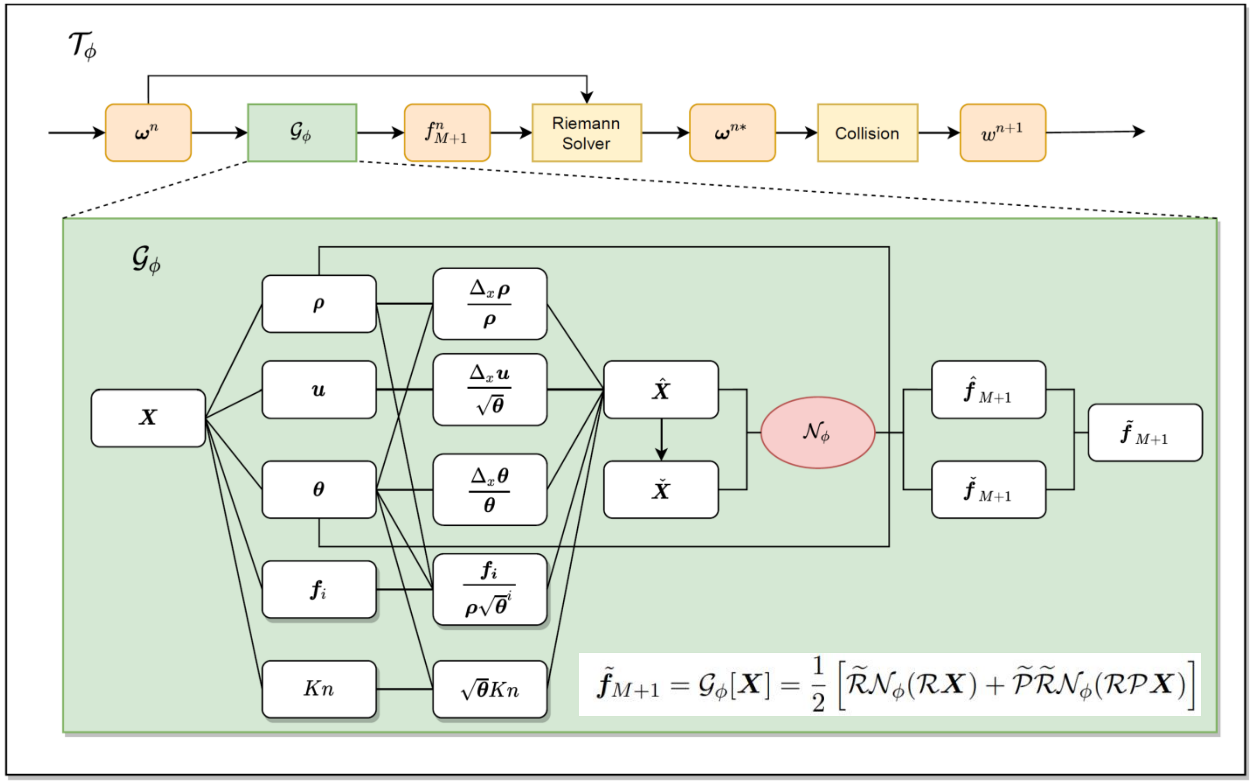
Learning to Close Moment Systems of Boltzmann-BGK Equation

- Mathematical approach
  - Grad method (H. Grad, CPAM 1949)
  - Maximum entropy closure (C. D. Levermore, JSP 1996; M. Junk, JSP 1998; J. McDonald and M. Torrilhon, JCP 2013)
  - Quadrature based closure (R.O. Fox, JCP 2008&2009;)
  - Hyperbolic moment method (Z. Cai, Y. Fan, R. Li., CPAM 2014; J. Koellermeier, R. Schaerer and M. Torrilhon, KRM 2014)
- Machine learning approach
  - Learning closure:
    - J. Han et al., PNAS, 2019.
    - L. Bois et al., arXiv:2011.06242.
    - W. Porteous, M. Laiu, C. Hauck., arXiv:2106.08973.
    - S. Schotthofer et al., arXiv:2106.09445.
  - Solving Boltzmann directly:
    - T. Xiao and M. Frank, JCP, 2021.
    - Q. Lou, X. Meng, G. Karniadakis, JCP, 2021.
  - Closure of RTE:
    - J. Huang et al. arXiv:2105.05690, arXiv:2105.14410, arXiv:2109.00700.

Learning to Close Moment Systems of Boltzmann-BGK Equation

- Invariance preserving neural closure (IPNC)

$$\boldsymbol{\omega}^{n+1} = \mathcal{T}_\phi[\boldsymbol{\omega}^n; Kn, \Delta t] = \mathcal{S}[\boldsymbol{\omega}^n, f^n_{M+1}; Kn, \Delta t] = \mathcal{S}[\boldsymbol{\omega}^n, \mathcal{G}_\phi[\boldsymbol{\omega}^n; Kn]; Kn, \Delta t]$$


$\mathcal{T}_\phi$
$\boldsymbol{\omega}^n$
$\mathcal{G}_\phi$
f^n_{M+1}
Riemann Solver
$\boldsymbol{\omega}^{n*}$
Collision
$\boldsymbol{\omega}^{n+1}$
$\mathcal{G}_\phi$
$\boldsymbol{X}$
$\boldsymbol{\rho}$
$\boldsymbol{u}$
$\boldsymbol{\theta}$
$\boldsymbol{f}_i$
Kn
$\frac{\Delta_x \boldsymbol{\rho}}{\boldsymbol{\rho}}$
$\frac{\Delta_x \boldsymbol{u}}{\sqrt{\boldsymbol{\theta}}}$
$\frac{\Delta_x \boldsymbol{\theta}}{\boldsymbol{\theta}}$
$\frac{\boldsymbol{f}_i}{\boldsymbol{\rho}\sqrt{\boldsymbol{\theta}}^i}$
$\sqrt{\boldsymbol{\theta}}Kn$
$\hat{\boldsymbol{X}}$
$\check{\boldsymbol{X}}$
$\mathcal{N}_\phi$
$\hat{\boldsymbol{f}}_{M+1}$
$\check{\boldsymbol{f}}_{M+1}$
$\tilde{\boldsymbol{f}}_{M+1}$

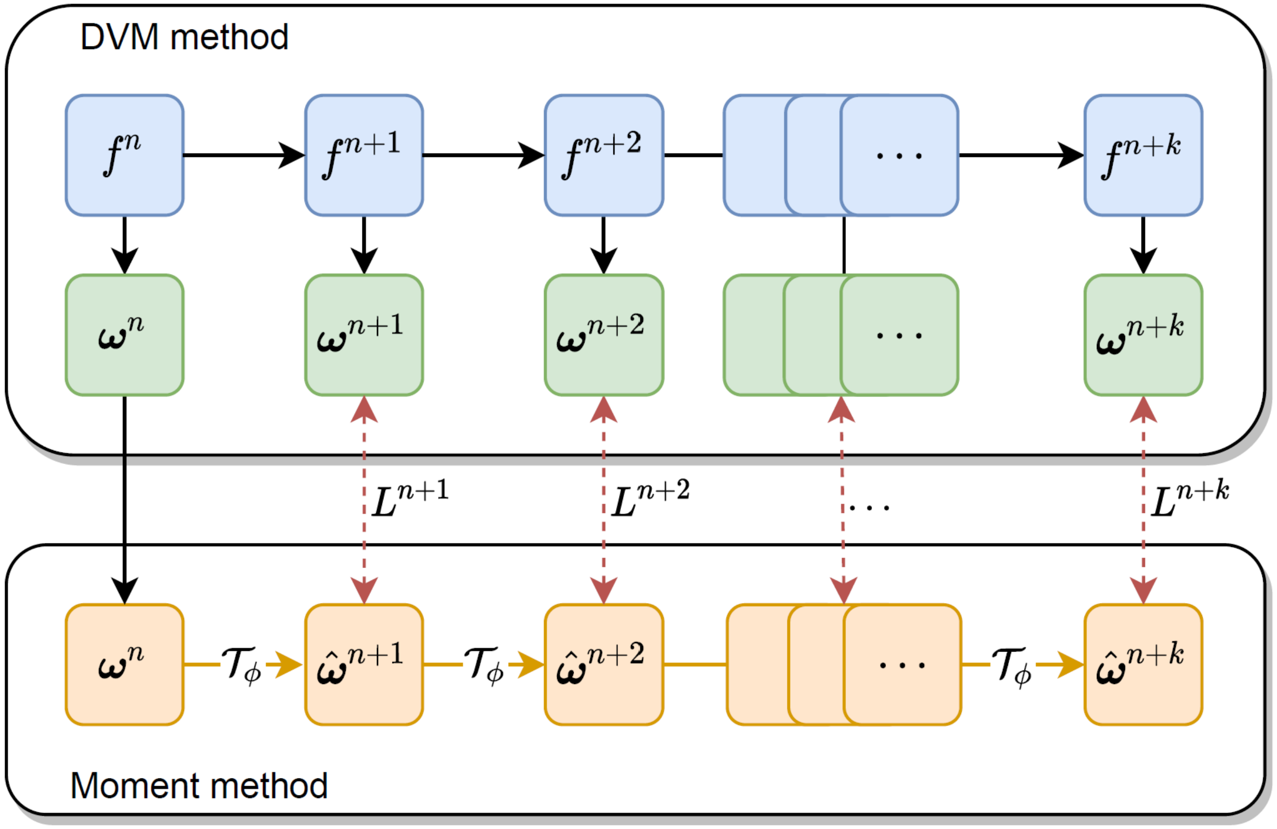
$$\tilde{\boldsymbol{f}}_{M+1} = \mathcal{G}_\phi[\boldsymbol{X}] = \frac{1}{2}\left[\widetilde{\mathcal{R}}\mathcal{N}_\phi(\mathcal{R}\boldsymbol{X}) + \widetilde{\mathcal{P}}\widetilde{\mathcal{R}}\mathcal{N}_\phi(\mathcal{R}\mathcal{P}\boldsymbol{X})\right]$$

Achieving Galilean, reflection and scaling invariance.

Learning to Close Moment Systems of Boltzmann-BGK Equation

- Training of IPNC

$$L^B_{\mathrm{sol},\phi} = \sum_{i\in\mathcal{I}_{\mathrm{test}}} \sum_{j=1}^{N_x} \sum_{b=1}^{B} \|\mathcal{T}^b_\phi(\boldsymbol{w}^i)_j - \boldsymbol{w}_j^{i+b}\|$$



Learning to Close Moment Systems of Boltzmann-BGK Equation

- Experiments: discontinuous initial conditions

$$\boxed{f_{\text{mix}} = \alpha f_{\text{smooth}} + (1-\alpha)\mathcal{M}^{\boldsymbol{U}_{\text{shock}}}(x,v)}$$

$$f_{\text{smooth}} = \frac{\alpha_1 \mathcal{M}^{\boldsymbol{U}_1}(x,v) + \alpha_2 \mathcal{M}^{\boldsymbol{U}_2}(x,v)}{\alpha_1 + \alpha_2 + 10^{-6}}$$

$$\mathcal{M}^{\boldsymbol{U}}(x,v) = \frac{\rho(x)}{\sqrt{2\pi\theta(x)}} \exp\left(-\frac{(v-u(x))^2}{2\theta(x)}\right)$$

$$\begin{cases} \rho(x) = a_\rho \sin(2k_\rho \pi x/L + \phi_\rho) + b_\rho, \\ u(x) = 0, \\ \theta(x) = a_\theta \sin(2k_\theta \pi x/L + \phi_\theta) + b_\theta, \end{cases}$$

$$\boldsymbol{U}^1_{\text{shock}} = \begin{cases} (\rho_l, u_l, \theta_l), & x \in [-0.5, x_1] \text{ or } [x_2, 0.5], \\ (\rho_r, u_r, \theta_r), & x \in [x_1, x_2], \end{cases}$$

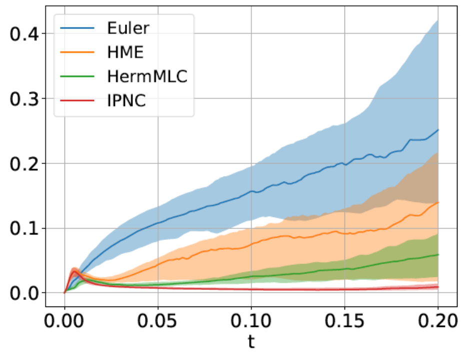
$$\boldsymbol{U}^2_{\text{shock}} = \begin{cases} (\rho_r, u_r, \theta_r), & x \in [-0.5, x_1] \text{ or } [x_2, 0.5], \\ (\rho_l, u_l, \theta_l), & x \in [x_1, x_2], \end{cases}$$

- Parameters randomly generated
- Training set generated by DVM for $x \in [-0.5, 0.5]$, $t \in [0, 0.1]$ (same as in Han et al. PNAS 2019)

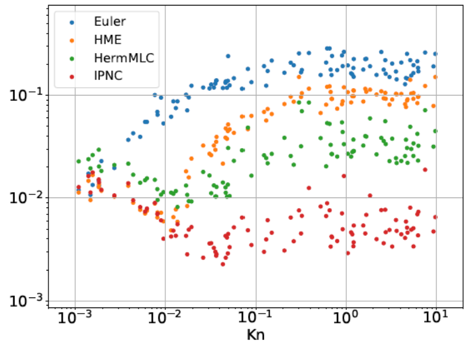
Learning to Close Moment Systems of Boltzmann-BGK Equation

- Experiments: discontinuous initial conditions

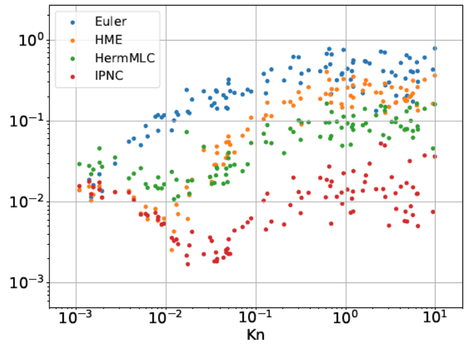
	$T = 0.1$					$T = 0.2$				
Kn	0.001	0.01	0.1	1.0	10	0.001	0.01	0.1	1.0	10
Euler	1.30	7.60	15.73	18.20	18.51	1.58	10.80	31.79	42.78	44.35
HME	**0.78**	0.75	6.49	9.55	9.95	**0.91**	0.58	10.24	21.26	23.13
HermMLC	1.67	0.89	2.15	3.19	3.35	2.33	1.47	4.96	9.55	10.40
IPNC	0.79	**0.40**	**0.49**	**0.67**	**0.74**	**0.91**	**0.35**	**1.29**	**5.22**	**6.00**



(a) error quarterlies



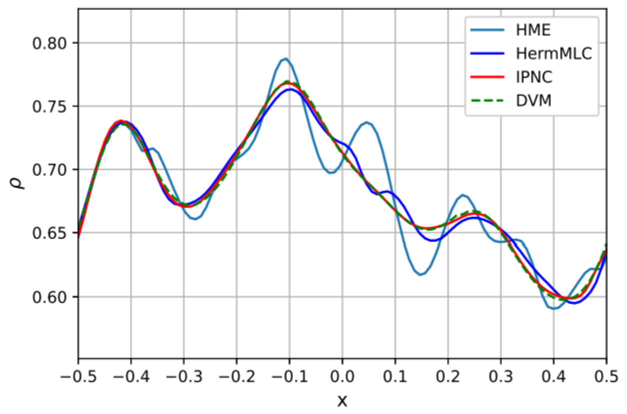
(b) different Kn at $t = 0.1$



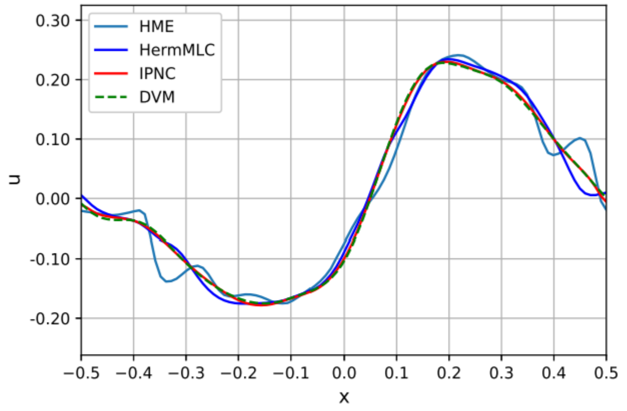
(c) different Kn at $t = 0.2$

Learning to Close Moment Systems of Boltzmann-BGK Equation

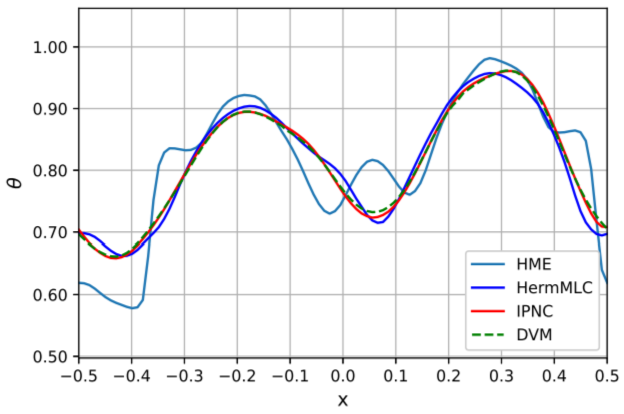
- Experiments: discontinuous initial conditions



(a) ρ



(b) u



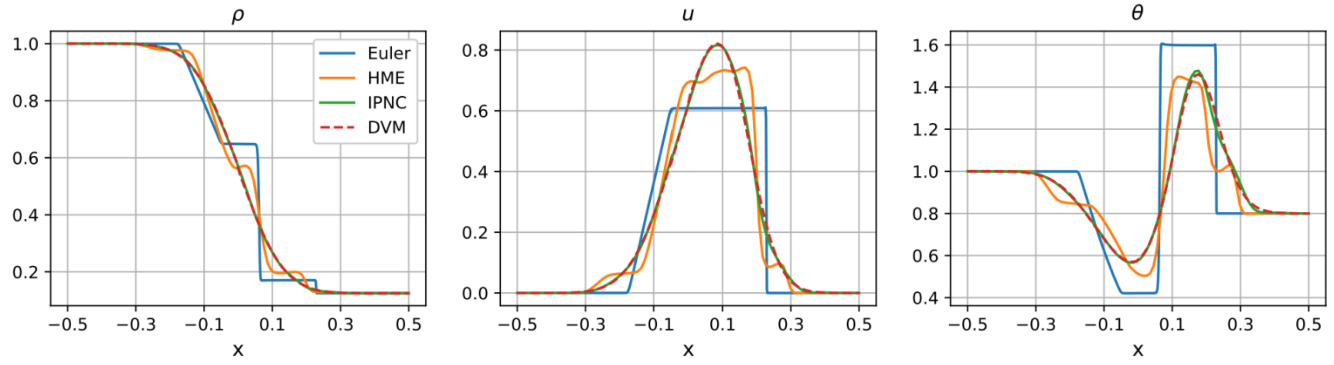
(c) θ

$$\boldsymbol{K_n = 6.2; t = 0.1}$$

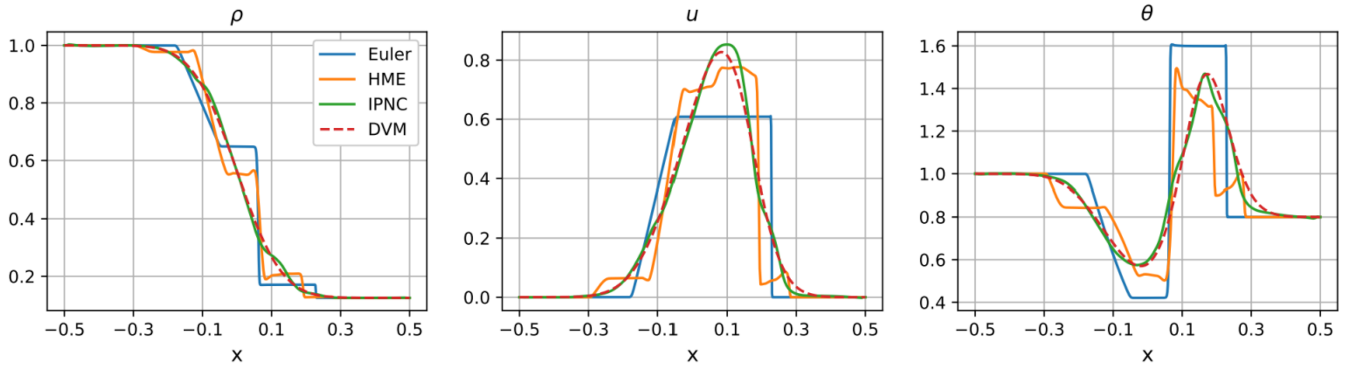
Learning to Close Moment Systems of Boltzmann-BGK Equation

- Generalization: trained on discontinuous initial conditions and generalize to Sod's shock tube *without* retraining.

$K_n = 1$
$t = 0.1$



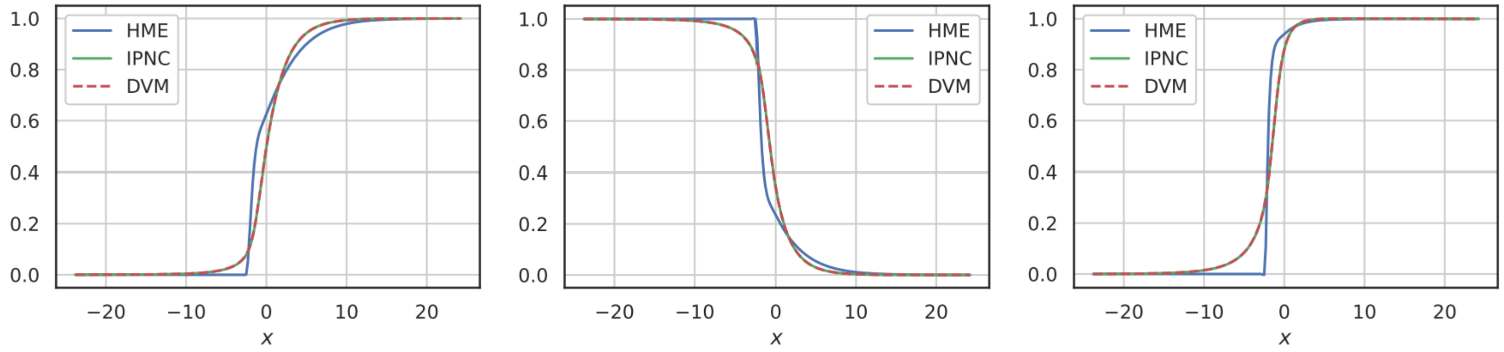
$K_n = 10$
$t = 0.1$



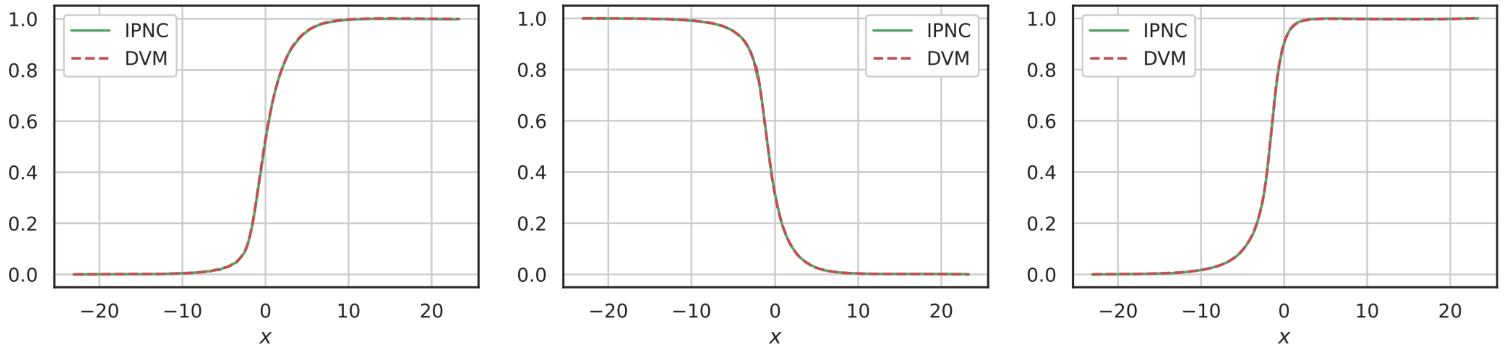
Learning to Close Moment Systems of Boltzmann-BGK Equation

- Shock structure: trained on small $Ma < 7$ and generalize to larger $Ma = 21$ *without* retraining.

$K_n = 0.1; Ma = 7$



$K_n = 0.1; Ma = 21$



Learning to Close Moment Systems of Boltzmann-BGK Equation

- Enforcing invariances help with generalization: both in-distribution (ID) and out-of-distribution (OoD).

GI	SI	RI	Train on Wave				Train on Mix			
			Wave	Wave-OoD	Mix	Mix-OoD	Wave	Wave-OoD	Mix	Mix-OoD
			0.63	2.16	4.33	2.44	1.50	-	0.62	-
		✓	0.61	1.89	4.25	2.12	1.38	-	0.60	-
✓			0.63	2.9	1.53	3.43	1.62	-	0.61	-
	✓		0.61	**0.25**	1.53	0.86	0.96	0.32	0.61	**0.52**
	✓	✓	**0.58**	**0.25**	1.48	**0.85**	**0.92**	**0.31**	**0.59**	**0.52**
✓	✓		0.62	0.26	1.51	0.91	1.11	0.36	0.63	0.53
✓	✓	✓	0.59	**0.25**	**1.47**	0.90	1.05	0.33	**0.59**	**0.52**

- iD: train on wave & test on wave; train on Mix & test on Mix
- OoD: all other cases

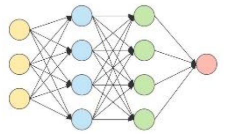
Learning to Solve PDEs

A brief review

Neural Networks (NNs) and Numerical PDEs – A Highly incomplete List

- NN as a new ansatz of solution u:
  - C. Beck, W. E, A. Jentzen. JNS, 1–57, 2017.
  - J. Han, A. Jentzen, W.E. CMS, 5, 349–380, 2017.
  - W. E and B. Yu, CMS, 6(1), 1-12, 2018.
  - Sirignano and Spiliopoulos, JCP, 375:1339-1364, 2018.
  - M. Raissi et al., JCP, 378:686–707, 2019.
  - Y. Zhang et al., arXiv:1907.08272.
  - D. Pfau et al., arXiv:1909.02487.
  - W. Cai and Z. Xu, arXiv:1910.11710.
  - Z. Liu, W. Cai, Z. Xu, arXiv:2007.11207.

- Consider $\mathcal{L}(u) = 0$.
- Use ansatz $u \approx u_\Theta(\boldsymbol{x}, t)$
- Loss: $\|\mathcal{L}(u_\Theta)\|_2^2$, $E(u_\Theta)$, etc.

$(\boldsymbol{x}, t)$  $u_\theta(\boldsymbol{x}, t)$

- NN as a new ansatz of solution mapping $\mathcal{S}$:
  - Integration with classical solvers
    - D. Ray, J. S Hesthaven. JCP, 367:166–191, 2018.
    - Y. Bar-Sinai et al., PNAS, 116 (31), 15344–15349, 2019
    - N. Discacciati et al., JCP, p. 109304, 2020.
    - Y. Feng, T. Liu, K. Wang, JSC, 83(21), 2020.
    - Y. Wang et al., CiCP, 28, 2158-2179, 2020.
  - Direct approximation
    - Y. Khoo, J. Lu, L. Ying. arXiv:1707.03351.
    - Z. Long et al., ICML 2018.
    - L. Lu, P. Jin, G. E. Karniadakis, arXiv:1910.03193.
    - K. Wu and D. Xiu, arXiv:1910.06948.
    - Z. Li et al., arXiv:2010.08895.
    - S. Cao, arXiv:2105.14995.

- Consider $\mathcal{L}(u; \boldsymbol{\eta}) = 0$.
- Start with a PDE solver $\mathcal{S}: \boldsymbol{\eta} \mapsto u; \mathcal{S} = \mathcal{S}_1 \circ \mathcal{S}_2 \circ \cdots \circ \mathcal{S}_K$
- Replace $\{\mathcal{S}_{k_j}\}$ by NNs

- Consider $\mathcal{L}(u; \boldsymbol{\eta}) = 0$.
- Classical solver: $\mathcal{S}: \boldsymbol{\eta} \mapsto u$
- Approximation: $\mathcal{S} \approx \mathcal{F}(\eta; \Theta)$

A Meta-Learning Approach for Parametrized PDEs

- Yuyan Chen, Bin Dong and Jinchao Xu, *Meta-MgNet: Meta Multigrid Networks for Solving Parameterized Partial Differential Equations*, arXiv:2010.14088, 2020.

Parameterized Partial Differential Equations (PDEs)

- General parameterized PDEs:

$$\mathcal{L}(u, \boldsymbol{x}, t; \boldsymbol{\eta}) = 0, \qquad x \in \Omega \subset \mathbb{R}^d, t \geq 0$$

  - Boundary condition: $\mathcal{B}(u, \boldsymbol{x}_{BC}, t; \boldsymbol{\eta}) = 0$
  - Initial condition: $u_0 = u_{IC}(\boldsymbol{x}; \boldsymbol{\eta})$
  - Others
    - State variable: $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x}, t; \boldsymbol{\eta}) \in \mathbb{R}^m$
    - Parameter vector: $\boldsymbol{\eta} \in \boldsymbol{D} \subset \mathbb{R}^p$

- Linear steady parameterized PDEs:

$$\boxed{\begin{cases} \underset{\sim}{\mathcal{A}_\eta}\, \underset{\sim}{u} = \underset{\sim}{f}, & \text{in } \Omega, \\ \underset{\sim}{u} = \underset{\sim}{u_b}, & \text{on } \partial\Omega. \end{cases}}$$

e.g.

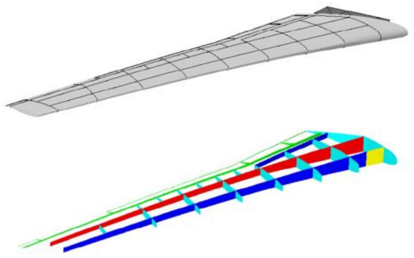
2D anisotropic diffusion equation with $\boldsymbol{\eta} = (\varepsilon, \theta)$

$$\begin{cases} -\nabla \cdot (C \nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases}$$

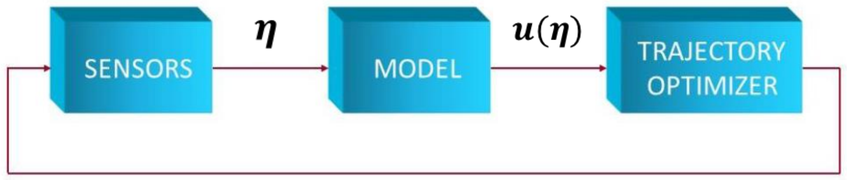
$$C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

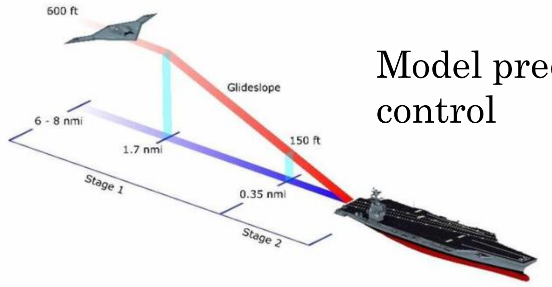
Applications That Require Efficient Solvers For Parameterized PDEs

- Typical parameters of interest
  - Shape parameters
  - Material (properties) parameters
  - Operation parameters (e.g. flight conditions, cruise conditions, etc.)
  - Initial and boundary conditions



- Scenarios require solving $\boldsymbol{u}(\boldsymbol{\eta})$ for multiple $\boldsymbol{\eta}$
  - Inverse problems
  - Uncertainty quantification
  - Design optimization
  - Optimal control
  - Model predictive control





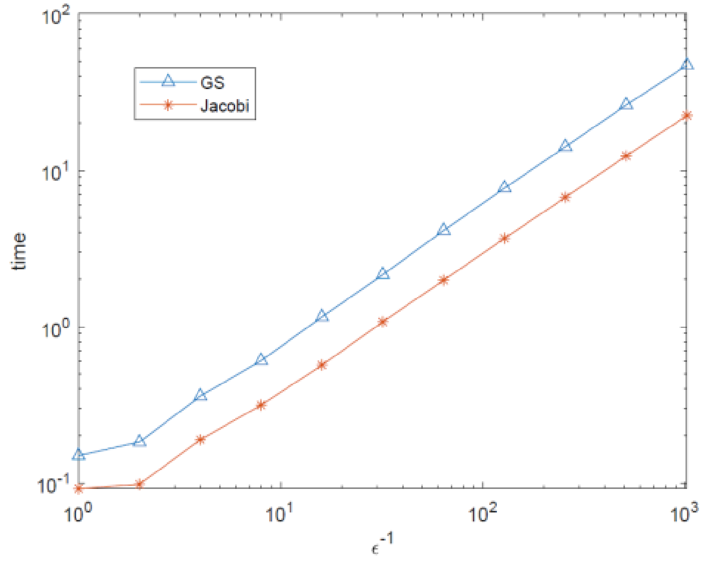
Model predictive control

Multigrid Method for Linear Problems

- We focus on the linear problem: $\boldsymbol{A_\eta u = f}$
- Multigrid method (MG) has linear complexity
- However, CPU time for MG can go up significantly when $\boldsymbol{\eta}$ is within certain range
- For example

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases}$$

$$C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$



Meta-Learning - Motivation

- Single task: learning a solver for a given $\boldsymbol{\eta}$
- Multi-task: learning a solver for a set of $\boldsymbol{\eta}$
- Key difference from supervised learning strategy:
  - Leveraging **common structures** hidden in the tasks!
- Effective approach: Meta-Learning
  - Finding a good initialization for all tasks

  (Finn, Abbeel and Levine, 2017; Nichol, Achiam and Schulman, 2018)
  - Designing a hypernetwork to infer suitable parameters for each task

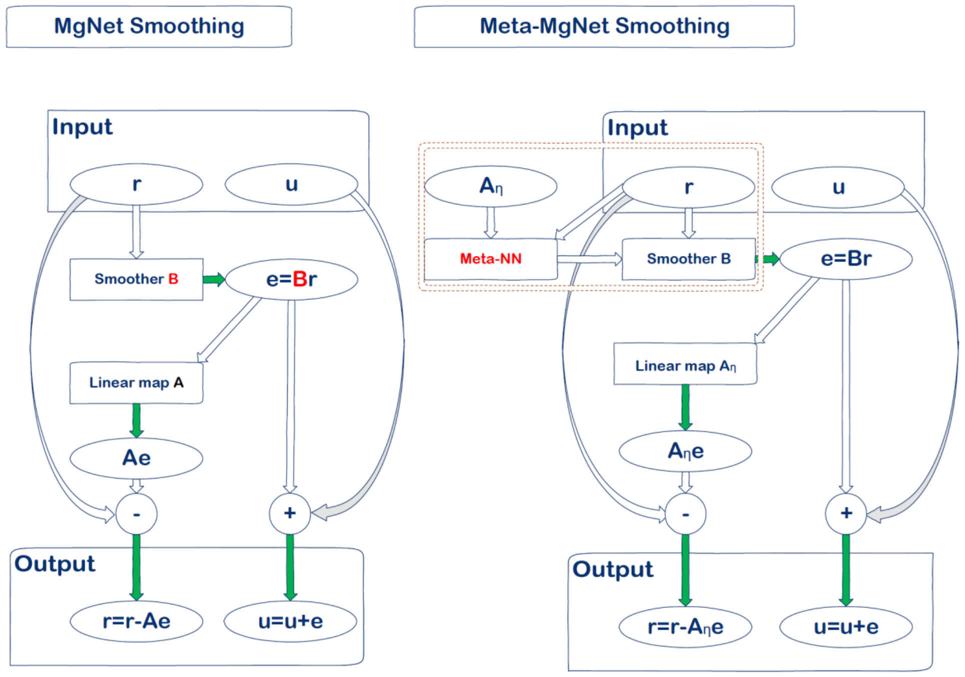
  (Ha, Dai and Le, 2016; Lorraine and Duvenaud, 2018; Brock, Lim, Ritchie and Weston, 2017;Zhang, Liu, Yu and Dong, 2020)

Meta-MgNet

- Introducing a hypernetwork (Meta-NN) in MgNet (Xu & He 2019)

$$u_{t+1} = u_t + \text{Meta-MgNet}(f - A_\eta \star u_t, A_\eta),$$

- Meta-NN induces a smoother adaptive to $\boldsymbol{\eta}$

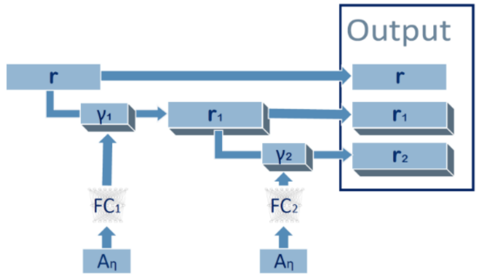


Meta-NN: $\mathcal{G}_\theta(\boldsymbol{r}, \boldsymbol{A}_\eta)$

Input:

- Kernel of the operator $\boldsymbol{A}_\eta$
- Residual $\boldsymbol{r}$

Output: a set of vectors that spans a subspace for subspace correction



Meta-MgNet

- Introducing a hypernetwork (Meta-NN) in MgNet (Xu & He 2019)

$$\mathsf{u}_{t+1} = \mathsf{u}_t + \text{Meta-MgNet}(\mathsf{f} - \mathsf{A}_\eta \star \mathsf{u}_t, \mathsf{A}_\eta),$$

- Meta-NN induces a smoother adaptive to $\boldsymbol{\eta}$
- Motivation: Krylov subspace

$$\mathbb{G}_\mathrm{K} = \{\mathbf{f}_0(\mathbf{A})\mathbf{r}, \mathbf{f}_1(\mathbf{A})\mathbf{r}, ..., \mathbf{f}_k(\mathbf{A})\mathbf{r}\}, \text{ and } \mathbf{f}_i(\mathbf{A}) = \mathbf{A}^i.$$

- Meta-NN: $\mathcal{G}_{\boldsymbol{\theta}}(r, A_\eta) = \mathcal{N}_{FC_{\boldsymbol{\theta}}(A_\eta)}(r)$,
  - $\mathcal{N}_{\boldsymbol{\gamma}}$ is a 3-layer dense-net block (Huang et al. 2017) with parameter $\boldsymbol{\gamma}$
  - $FC_{\boldsymbol{\theta}}$ is a 2-layer fully connected neural network with parameter $\boldsymbol{\theta}$
- Convergence guarantee for Poisson problem √

Experiments

- 2D anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \qquad C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

- Training:
  - $\theta = 0,\ \lg\frac{1}{\varepsilon} \sim \mathcal{U}[0,5]$
  - randomly generate 100 right-hand-side function $f \sim \mathcal{N}(0,1)$.
- Testing:
  - $\theta = 0,\ \varepsilon = 10^{-l}, l = 0,1,\ldots,5$ (in-distribution generalization)
  - randomly generate 10 right-hand-side function $f \sim \mathcal{N}(0,1)$
  - select stopping criteria

$$\frac{||\mathbf{f} - \mathbf{A}_\eta \mathbf{u}_t||_2}{||\mathbf{f}||_2} < 10^{-6}.$$

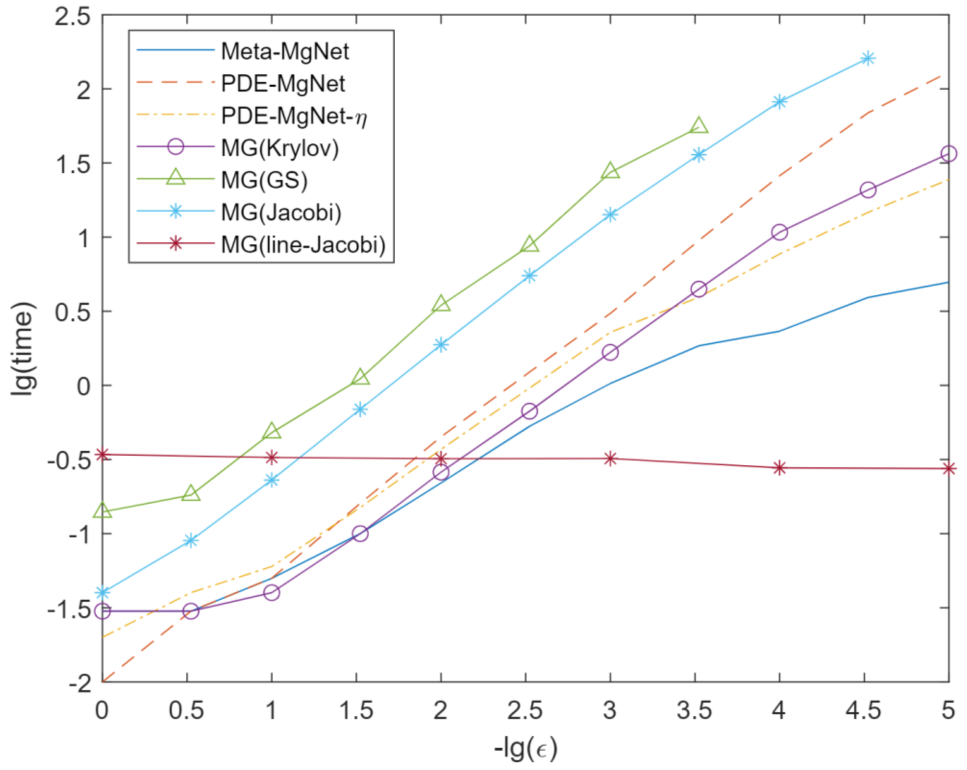
- Report mean$\pm$std of number of iterations and computation time for each experiment with each compared algorithm

Experiments

- 2D anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \qquad C = C(\epsilon,\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

- Results:



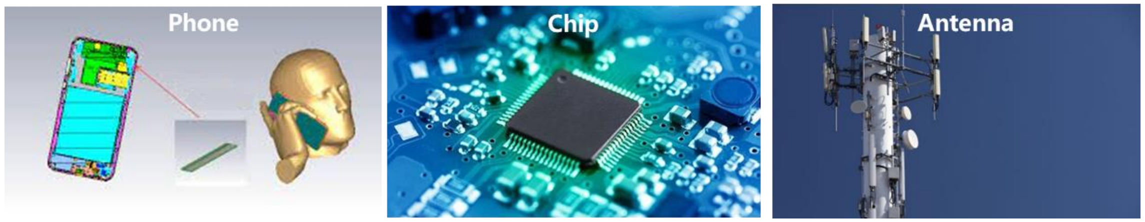
Meta-MgNet
×7 speedup

Electromagnetic simulation

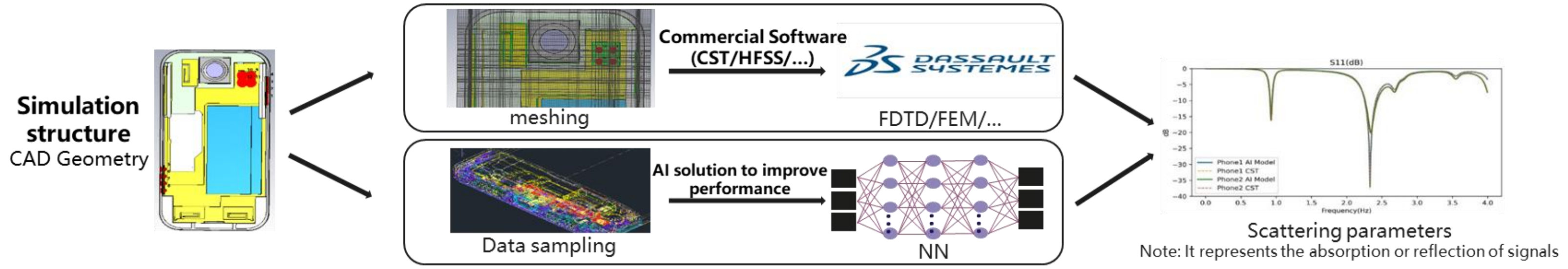
- Jointly with Huawei MindSpore AI + Scientific Computing team

MindSpore AI Electromagnetic Simulation: Background

- Motivations: Electromagnetic simulation (solution of Maxwell equations) is widely used in the design of mobile phones and chips.



- Traditional v.s. machine learning approaches



Challenges for machine learning methods:

a) Efficient data conversion: Domain data needs to be efficiently converted into AI-affinity tensor data.

b) Memory and computation: A single sample requires several GB video memory, and a single convolutional layer computation exceeds 400 GFLOPs.

c) Generalization: The model needs to handle different structures and materials and produce errors under 10% compared with commercial software.

MindSpore AI Electromagnetic Simulation: Typical Scenario I

- Meta Auto-Decoder (MAD) for solving parametric PDEs
  - Parameterized PDEs: $\boxed{\mathcal{L}_x^\eta u = \phi_\eta \quad x \in \Omega \subset \mathbf{R}^d}$ $\boxed{\mathcal{B}_x^\eta u = 0 \quad x \in \partial\Omega}$
  - Pretrain stage:

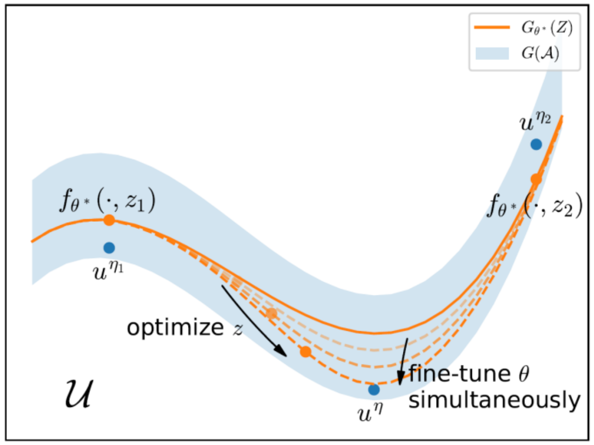
$$\arg\min_{\theta,\{z_i\}_{i\in\{1,\dots,N\}}} \sum_{i=1}^{N} \left(\hat{L}^{\eta_i}[f_\theta(\cdot, z_i)] + \frac{1}{\sigma^2}\|z_i\|^2 \right)$$

  - Fine-tune stage:

$$(z^\eta, \theta^\eta) = \arg\min_{z,\theta} \hat{L}^\eta[f_\theta(\cdot, z)] + \frac{1}{\sigma^2}\|z\|^2$$

- *Fixed θ: Park et al., CVPR 2019*

- Manifold learning interpretation of MAD

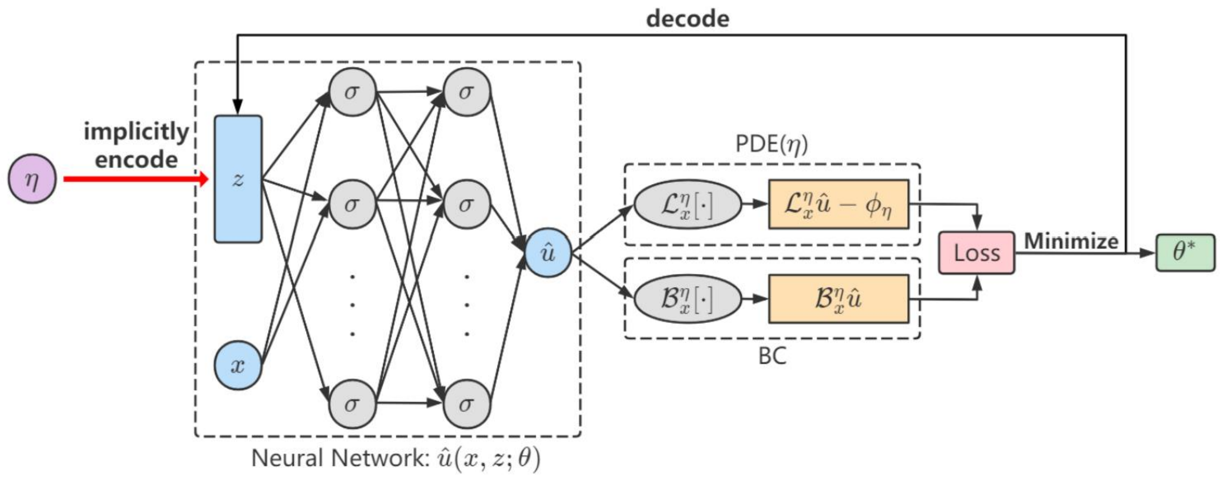


- $\boldsymbol{\mathcal{U}}$ is a function space contains all solutions of the parametric PDEs
- $\boldsymbol{\mathcal{A}}$ is the space of all parameters η
- $\boldsymbol{\mathcal{Z}}$ is the space of latent variables
- $G(\boldsymbol{\mathcal{A}}) = \{u^\eta : \eta \in \boldsymbol{\mathcal{A}}\}$ is the set of all solutions
- $G_\theta(\boldsymbol{\mathcal{Z}}) = \{f_\theta(x,z) : z \in \boldsymbol{\mathcal{Z}}\}$ approximates $G(\boldsymbol{\mathcal{A}})$
  - For any η, there exists z, such that $\|G_\theta(z) - G(\eta)\|_{\mathcal{U}} < \delta$ (*Assumption*)

arXiv:2111.08823

MindSpore AI Electromagnetic Simulation: Typical Scenario I

- Network architecture of MAD: $f_\theta(x, z)$


decode
implicitly encode
η
z
x
σ
$\hat{u}$
PDE(η)
$\mathcal{L}_x^\eta[\cdot]$
$\mathcal{L}_x^\eta \hat{u} - \phi_\eta$
$\mathcal{B}_x^\eta[\cdot]$
$\mathcal{B}_x^\eta \hat{u}$
BC
Loss
Minimize
θ^*
Neural Network: $\hat{u}(x, z; \theta)$

- PINN loss (Raissi, Perdikaris, Karniadakis, 2019):

$$\begin{aligned}\hat{L}^\eta[u] &= \hat{L}_{\mathrm{r}}^\eta[u] + \lambda_{\mathrm{bc}} \hat{L}_{\mathrm{bc}}^\eta[u] \\ &= \frac{1}{M_{\mathrm{r}}} \sum_{j=1}^{M_{\mathrm{r}}} \left\| \mathcal{L}_x^\eta u(x_j^{\mathrm{r}}) - \phi_\eta(x_j^{\mathrm{r}}) \right\|_2^2 + \frac{\lambda_{\mathrm{bc}}}{M_{\mathrm{bc}}} \sum_{j=1}^{M_{\mathrm{bc}}} \left\| \mathcal{B}_x^\eta u(x_j^{\mathrm{bc}}) \right\|_2^2\end{aligned}$$

arXiv:2111.08823

MindSpore AI Electromagnetic Simulation: Typical Scenario I

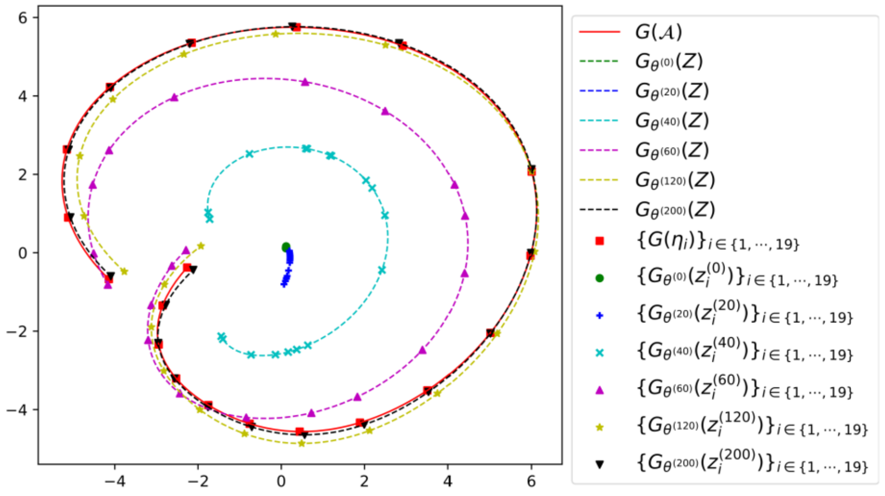
- Demonstration of MAD on a simple example

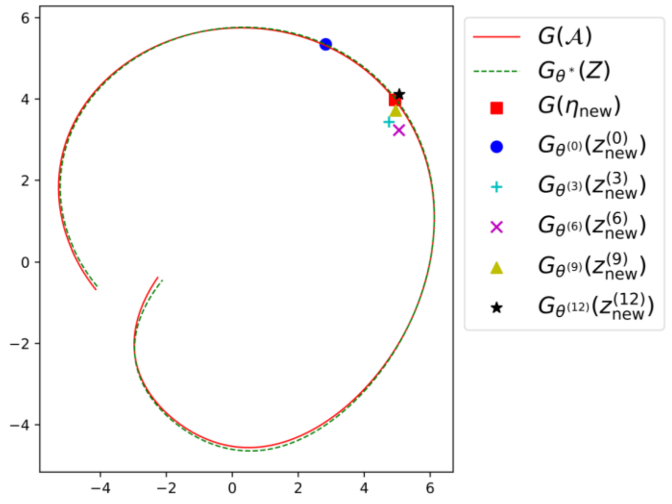
$$\frac{\mathrm{d}u}{\mathrm{d}x} = \phi_\eta(x) = 2(x-\eta)\cos\big((x-\eta)^2\big), \qquad \boxed{u^\eta(x) = \sin\big((x-\eta)^2\big)}$$

$$u(-\pi) = \sin\big((\pi+\eta)^2\big),$$

$$u(\pi) = \sin\big((\pi-\eta)^2\big) \qquad L^\eta[u] = \int_{-\pi}^{\pi} \big|u'(x) - \phi_\eta(x)\big|^2 \,\mathrm{d}x + \big|u(-\pi) - u^\eta(-\pi)\big|^2 + \big|u(\pi) - u^\eta(\pi)\big|^2.$$

- Pre-training and fine-tuning of MAD

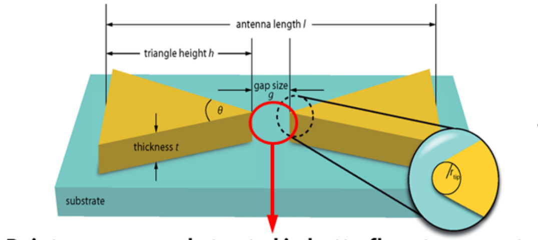




arXiv:2111.08823

MindSpore AI Electromagnetic Simulation: Typical Scenario I

- Descriptions on typical scenario I
  - **Scenario introduction:** point excitation source electromagnetic simulation + parameterized PDE solution.



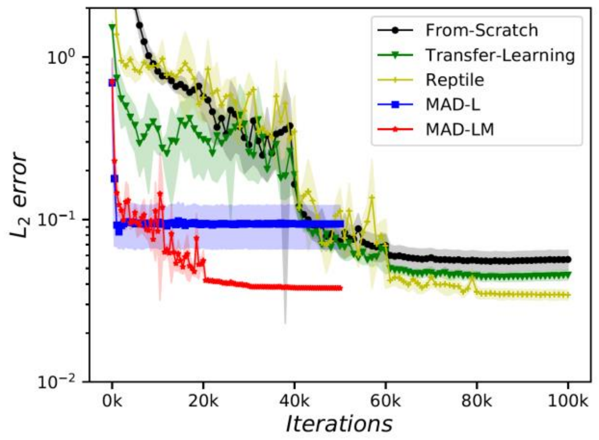
Point sources are abstracted in butterfly antenna ports

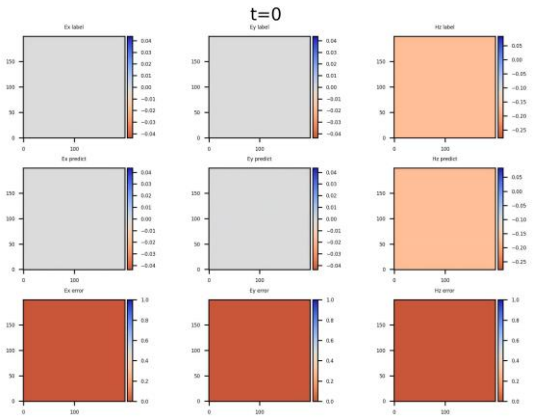
Maxwell's Equation

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon_0 \epsilon_r} \frac{\partial H_z}{\partial y},$$
$$\frac{\partial E_y}{\partial t} = -\frac{1}{\epsilon_0 \epsilon_r} \frac{\partial H_z}{\partial x},$$
$$\frac{\partial H_z}{\partial t} = -\frac{1}{\mu_0 \mu_r}\left(\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} + J\right).$$

$$J(x,y,t) = e^{-(\frac{t-d}{\tau})^2}\delta(x-x_0)\delta(y-y_0).$$

- Results: Implemented on MindSpore and NPU/GPU. The relative error of the electromagnetic field is $\approx$ **10%** with $\times$ **20** speedup over PINN.





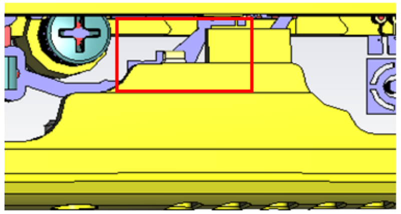
FDTD

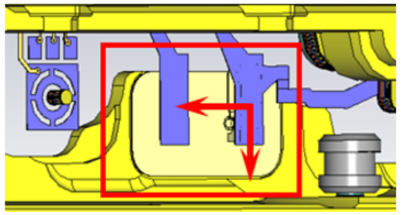
Auto-decoder

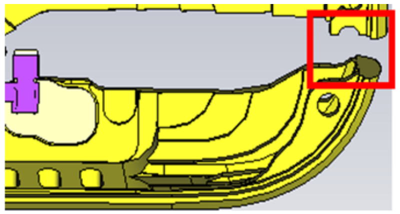
Error map

MindSpore AI Electromagnetic Simulation: Typical Scenario II

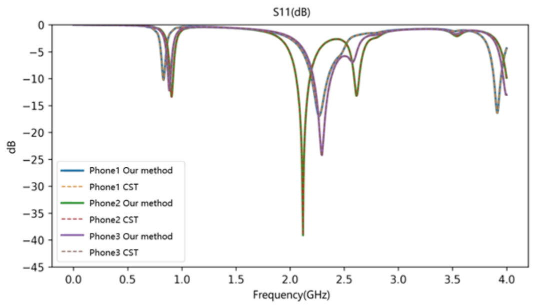
- Task: Learn a direct mapping between mobile device structures and S-parameters based on historical data gathered by Huawei Device Co., Ltd.


Change of metal plate


Change of spring position


Change of gap structure

- Methodology: efficient data sampling, grid data compression, and neural architecture design to facilitate structural/material generalization.
- Results: Implemented on MindSpore and NPU/GPU. The relative error of the S-parameter is far less than **10%** with × **30** speedup.



- Time benefit: 32x improvement

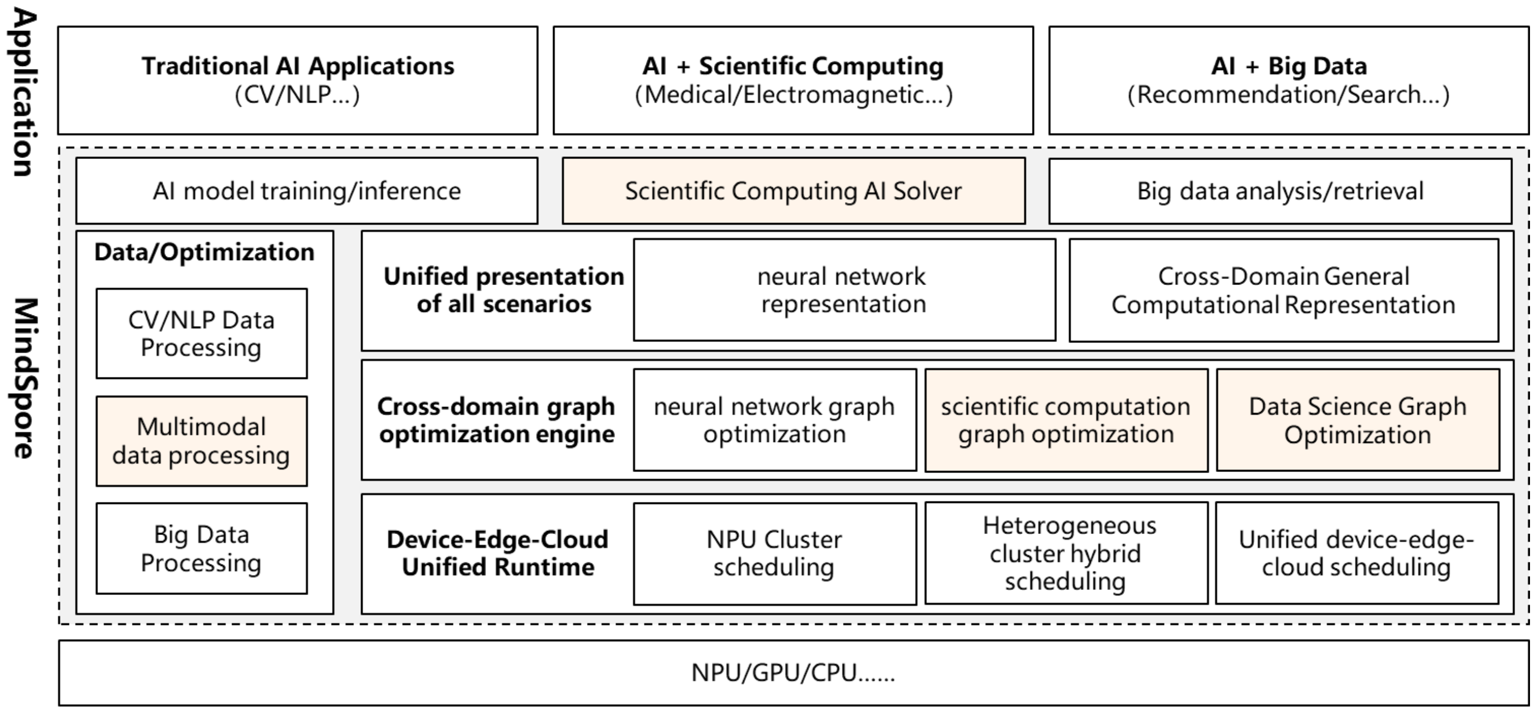
Method	Time(s)
Predicting Scattering Parameters with CST	2820
Predicting Scattering Parameters with our method	88

- Accuracy: The error is much less than 10%

Working condition	Structural change
Relative error of Scattering parameters compared to CST	1.71%

MindSpore AI Electromagnetic Simulation: MindSpore

- MindSpore architecture (open source):
  - https://www.mindspore.cn
  - https://gitee.com/mindspore



Application
MindSpore
Traditional AI Applications (CV/NLP…)
AI + Scientific Computing (Medical/Electromagnetic…)
AI + Big Data (Recommendation/Search…)
AI model training/inference
Scientific Computing AI Solver
Big data analysis/retrieval
Data/Optimization
CV/NLP Data Processing
Multimodal data processing
Big Data Processing
Unified presentation of all scenarios
neural network representation
Cross-Domain General Computational Representation
Cross-domain graph optimization engine
neural network graph optimization
scientific computation graph optimization
Data Science Graph Optimization
Device-Edge-Cloud Unified Runtime
NPU Cluster scheduling
Heterogeneous cluster hybrid scheduling
Unified device-edge-cloud scheduling
NPU/GPU/CPU……

CONCLUSIONS AND FUTURE DIRECTIONS

- Deep learning ≈ optimal control.
- This grants a framework to combine learning and handcraft modeling
  - Start with your favorite dynamics;
  - Identify the component(s) that is hard to handcraft;
  - Replace it by a deep neural network;
  - Pick a loss function and training algorithm;
  - Hope for the best and prepare for the worst.
- End-to-end mapping: significant speed up for a given task, but requires data and incorporate domain knowledge.

Thanks for Your Attention!

My Webpage:

HTTP://BICMR.PKU.EDU.CN/~DONGBIN