ML in Computational Sciences and HPC

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

***Abstract*—This seminar paper is an introduction for two interrelated subjects: i)Recent data-approach innovations as a solution for traditional computationally heavy approaches. ii)Point out the underlying oppotunites and difficulties of the usage of machine learning for HPC.**

***Index Terms*—HPC, machine learning, surrogate, reinforcement learning, PDE solver, matrix multiplication**

I. Introduction

Old fashion simulation-based solutions has confronted challeges as the fact that we could foresee the obsolescence of Moore's law and Dennard scaling. The emerge of machine learning and big data enveil the possibilities of solving traditional intractable problems by sidesteping. The usage of machine learning can not only solve some computational problems or discover underlying problems more efficiently but also will improve the general performance of HPC system as you can see in the later chapters. In this paper we will firstly focus on two computational breakthroughs with the usage of maching learning as innovative solotions, namely solving matrix multiplication and PDE. Secondly we will elaborate the idea of MLaroundHPC which involves different interactions between machine learning and HPC. First and formost, lets briefly explain some key concepts and terminologies.

- **Machine Learning and Deep Learning.** Machine learning is a technology which generates the underlying information ouf of given big amount of Data without explicit instruction; Deep learning is a machine learning algorithm that used multiple layers (called neural network) to extract hiddle features from input data.
- **Hign Performance Computing.** HPC can perfrom complex calculations using high speed, enable execution of large-scale simulation and computation, which are essential to solve scientific problems.
- **Surrogate Model** also known as a surrogate or metamodel, is a simplified and computationally efficient representation of a more complex and computationally expensive simulation or optimization model. The purpose of using a surrogate model is to approximate the behavior of the original model while significantly reducing the computational cost. As you can see in the later chapter. creating a surrogate model using machine learning is one of the typical usage of machine learning.
- **Tensor** is a generalization of matrix and vector. Matrix can be viewed as 2D tensor.Vector can be viewed as 1D tensor.Scalar as 0D. 3Dtensor can be considered as a vector of matrices as entry instead of numbers.
- **Reinforcement Learning** is a type of machine learning that learns the optimal decision of given enviornment. An agent, which is a decision maker, will receive reward or punishment after its decision based on our evaluation. A famous example would be AlphaZero, which beats top human player in shogi and chess.
- In computational physics nad numerical simulations, a **Mesh** is a discretized representation of a physical domain. Each element of the mesh represents a small part of the physical structure.
- A **Campaign** typically refers to a set of coordinated and planned computational tasks or simulations conducted with specific objectives.

Identify applicable funding agency here. If none, delete this.

II. Discovering Faster Matrix Multiplication Algorithms With Reinforcement Learning

A paper from 2022(Fawzi,A.,Balog,M., Huang, A. et al.) created an agent **AlphaTensor** using AlphaZero and deep reinforcement learning to search for provably correct and efficient matrix multiplication algorithms.

A. Previous Challeges in Matrix Multiplication

High order matrix multiplication is one of the most frequent and painful computation in science and engineering.Even though matrix multiplications can be formalized as low-rank decompositions of a specific three-dimensional(3D)tensor(c), called matrix multiplication tensor. Finding low-rank decompositions of 3D tensors(and beyond) is still NP-hard(c) and also hard in practice. Previously, matrix multiplication has been often rely on human-designed heuristics, which are probably suboptimal.Here Fawzi and others use deep reinforcement learning to learn and generalize patterns in tensors, and the use the learned agent to predict efficicent decomposition.

B. The Idea of AlphaTensor and Formulation

We can formulate the matrix multiplication algorithm discovery as a single-player game, called TensorGame. At each step of TensorGame, the player selects how to combine different entries of the matrices to multiply. A score will be assigned based on the number of selected operations required to reach

the correct multiplicaiton result. In order to find "good" matrix multiplication algorithms, DRL agent AlphaTensor has been developed, which is based on AlphaZero. Figure1 demostrates using 3D tensor of the size $4 \times 4 \times 4$ to represent the multiplication of two 2×2 matrices.
In general, we decompose tensor $\mathcal{T}_{n,m,p}$ into $\boldsymbol{R}$ rank-one terms:

$$\mathcal{T}_n = \sum_{r=1}^{\boldsymbol{R}} \mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)} \tag{1}$$

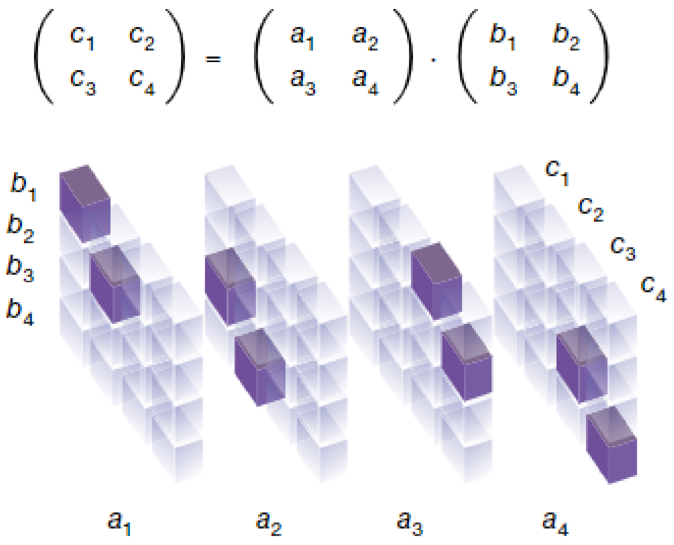

Fig. 1. $4 \times 4 \times 4$ Tensor $\mathcal{T}$ represents the muliplication of two 2×2 matrics. Purple box means tensor entries equal to 1 and transparent box means 0. For instance $c_1 = a_1 b_1 + a_2 b_3$, means (a_1, b_1, c_1) and (a_2, b_3, c_1) are set to 1.

where $\mathbf{u}^{(r)}$, $\mathbf{v}^{(r)}$ and $\mathbf{w}^{(r)}$ are vectors and $\boldsymbol{R}$ is the upper bound of the rank of $\mathcal{T}$. we denote $\otimes$ as the outer product. The underlying idea is simple: Tensor with rank n cannot be decomposed into $k < n$ rank 1 matrices. Once we have the decomposition, we can compute matrix multiplications conveniently.

We formalize finding decompositions as a reinforcement learning problem, modelling the environment as a single-player game, TensorGame. The procedure will be described as follows:

- We set our target tensor $\mathcal{T}$to our initial state $\mathcal{S}_0$.
- In each step t of the game, the player selects a triplet $\mathbf{u}^{(r)}, \mathbf{v}^{(r)}, \mathbf{w}^{(r)}$ and update the tensor $\mathcal{S}_t$ by substracting the tensor we computed from the triplet:

  $$\mathcal{S}_t \leftarrow \mathcal{S}_{t-1} - \mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}$$

  Our goal is try to find the smallest step the agent takes to acquire $\mathcal{S}_t = 0$, which will satisfy (1). We use $\boldsymbol{R}$ to limit the maximum computation times.

C. *Optimizations and Performance*

In fact, the whole idea is relatively straightforward. The significance lies in the engineering. A transformer-based(citation) architecture has been developed and deployed. The main part of the model comprises a sequence of attention operations, each applied to a set of features belonging to a pair of grid. This generalized axial attention to multiple grids, which is more efficient and yielding better results than naive self-attention. Apart from achitecture, the use of data argumentation and change of basis also has improved the general performance.

III. Fourier Neural Operator For Parametric Partial Differential Equations

Just like matrix multiplication, solving partial derivitive equation is one of the most common and intricate problems in science and engineering. A complex traditonal PDE systems, normally require very hign level of discretization, which is extremely time inefficient. A data-driven approach has been proven to be more efficient and accurate than traditional discretizational solvers. Here, we introduce **Fourier Neural Operator**, a novel deep learning architecture, which consists mostly of two previous works:

- Recent works of **Neural Operators** (citation), enable to transfer solutions between meshes. Moreover, the neural operator needs to be trained only once, which means it has only passed one time through the neural network. Lastly, the neural operator requires no knowledge of the underlying PDE, only data.
- **The Fourier Transform** is frequently used in spectral methods for solving differential equations, since differentiation is equivalent to multiplication in the Fourier domain. Fourier transforms have also played an important role in the development of deep learning. Empirically, they have been used to speed up convolutional neural networks. Recently, some spectral methods for PDEs have been extended to neural networks. This paper builds on these works by proposing a neural operator architecture defined directly in Fourier space with quasi-linear time complexity and state-of-the-art approximation capabilities.

A. *Formal Problems Formulation*

The methodology we present here is fairly convensional in machine learning modelling. We built a mapping between two infinite dimensional spaces from a finite collection of observed input-output paris. We parameterize the input and train a model to map to the desired output. Thereafter, we define a cost function and minimizer. Detailed formulizations are presented as follows:

Let $D \supset \mathbb{R}^d$ be a bounded, open set. Input $\mathcal{A} = \mathcal{A}\left(D; \mathbb{R}^{d_a}\right)$ and output $\mathcal{U} = \mathcal{U}\left(D; \mathbb{R}^{d_u}\right)$ be separable Banach spaces of function taking values in $\mathbb{R}^{d_a}$ and $\mathbb{R}^{d_u}$ respectively. We denote $G^\dagger : \mathcal{A} \rightarrow \mathcal{U}$ as our desired solution for parametric PDE. Suppose we have observations $\{a_j, u_j\}_{j=1}^N$ where $a_j \sim \mu$ is an independent and identically distributed sequence from the probability measure μ supported on $\mathcal{A}$ and normally $\mu_j = G^\dagger\left(a_j\right)$ will contain noises. Parametric map will be defined as follows:

$$G_\theta : \mathcal{A} \times \Theta \rightarrow \mathcal{U} \tag{2}$$

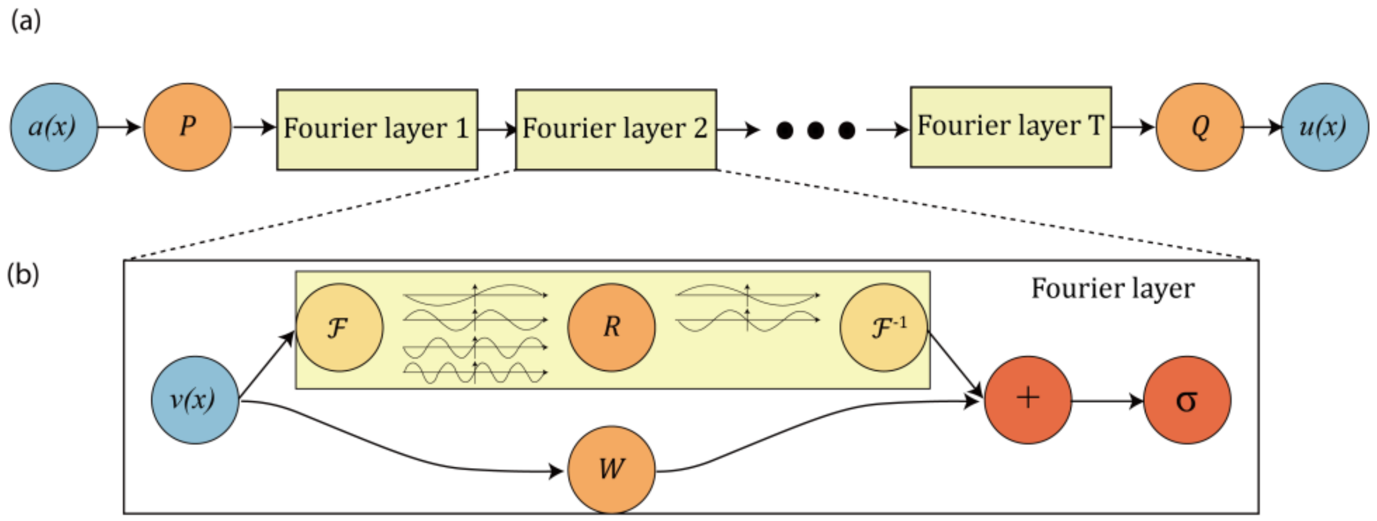

Fig. 2. **(a)The full architecture of neural operator:** start from input a. 1. Lift to a higher dimension channel space by a neural network P. 2. Apply four layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u.**(b) Fourier layers:** Start from input v. On top: apply the Fourier transform $\mathcal{F}$; a linear transform R on the lower Fourier modes and filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. On the bottom: apply a local linear transform W.

We want to achieve $G(\cdot,\theta) = G_{\theta^\dagger} \approx G^\dagger$ by finding the optimal parameter space Θ. And we define the cost function in convensional manner $C : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}$ and also the minimizer:

$$\min_{\theta\in\Theta} \mathbb{E}_{a\sim\mu}[C\left(G\left(a,\theta\right), G^\dagger\left(a\right)\right)] \tag{3}$$

B. Neural Operator and Fourier Neural Operator

The neural operator, proposed in (Li et al., 2020b), is an iterative learning architecture as presented in Figure2. P and Q are local transformation P which are usually parameterized by a shallow fully-connected neural network. Between P and Q there are multiple fourier layers $v_t \mapsto v_{t+1}$which are formal defined below:

Definition of Iterative Updates

$$v_{t+1}\left(x\right) := \sigma\left(W_{v_t}\left(x\right) + \left(\mathcal{K}\left(a;\phi\right)v_t\right)\left(x\right)\right), \quad \forall x \in D \tag{4}$$

where $\mathcal{K} : \mathcal{A} \times \Theta_{\mathcal{K}} \rightarrow \mathcal{L}\left(\mathcal{U}\left(D;\mathbb{R}^{d_v}\right), \mathcal{U}\left(D;\mathbb{R}^{d_v}\right)\right)$ maps to bounded linear operators on $\mathcal{U}\left(D;\mathbb{R}^{d_v}\right)$ and is parameterized by $\phi \in \Theta_{\mathcal{K}}$. $\mathcal{K}$ is a convolution operator defined in Fourier space, here is the definition:

Definition of Fourier Integral Operator $\mathcal{K}$

$$\left(\mathcal{K}\left(a;\phi\right)v_t\right)\left(x\right) = \mathcal{F}^{-1}\left(R_\phi \cdot \left(\mathcal{F}v_t\right)\right)\left(x\right), \quad \forall x \in D \tag{5}$$

where R_ϕ is the Fourier transform of a periodic function $\mathsf{K} : \overline{D} \rightarrow \mathbb{R}^{d_v\times d_v}$ parameterized by $\phi \in \Phi_{\mathsf{K}}$. More intuitive demonstration can be found in Figure 2 (b).

The idea of (5) is come from the original **Kernel Integral Operator** in (Li et al., 2020b). which is defined in (6):

$$\left(\mathcal{K}\left(a;\phi\right)v_t\right)\left(x\right) := \int_D \mathsf{K}\left(x,y,a\left(x\right),a\left(y\right);\phi\right)v_t\left(y\right)dy, \tag{6}$$

for $\forall x \in D$, where $\mathsf{K} : \mathbb{R}^{2(d+d_a)} \rightarrow \mathbb{R}^{d_v\times d_v}$ is a neural network parameterized by $\phi \in \Theta_{\mathsf{K}}$. In fact (5) can be seen as a optimization of (6) . To see the heuristical intuition of (5), we firstly denote $\mathcal{F}$ as the Fourier transform of a function $f : D \rightarrow \mathbb{R}^{d_v}$ and $\mathcal{F}^{-\infty}$ as its inverse and

$$\left(\mathcal{F}f\right)_j\left(k\right) = \int_D f_j\left(x\right)e^{-2i\pi\langle x,k\rangle}dx$$

$$\left(\mathcal{F}^{-1}f\right)_j\left(x\right) = \int_D f_j\left(k\right)e^{-2i\pi\langle x,k\rangle}dk$$

for $j = 1, ..., d_v$ where $i = \sqrt{-1}$ is the imaginary unit. By letting $\mathsf{K}\left(x,y,a\left(x\right),a\left(y\right);\phi\right) = \mathsf{K}\left(x-y;\phi\right)$, and applying the convolution theorem, we can conclude that

$$\left(\mathcal{K}\left(a;\phi\right)v_t\right)\left(x\right) = \mathcal{F}^{-1}\left(\mathcal{F}\left(\mathsf{K}_\phi\right)\cdot\mathcal{F}\left(v_t\right)\right)\left(x\right)$$

which is our definition in (5).

C. Performance and Contributions

The Fourier neural operator is the groud-breaking work that can manage to learn resolution-invariant solution operator for the family of Navier-Strokes equation in the turbulent regime. Despite from that, the Fourier neural operator also outperforms all existing deep learning methods in a relatively large scale even in fixing resolution 64×64: It achieves error rates that are 30% lower on Burgers'Equation, 60% lower on Darcy Flow, and 30% lower on Navier Stokes, which are the typical PDE scenarios. It also worth to mention that Fourieer neual operator achieve tremendous robustness. Despite its speed advantage, the method does not suffer from accuracy degradation when used in downstream applications such as solving the Bayesian inverse problem.

IV. MLAROUNDHPC: UNDERSTANDING ML DRIVEN HPC

Hign Performance Computing is essential for supporting machine learning in computation. The HPC communities have previously mistakenly assumed that as long as performance gains from hardware are possible, traditional simulation-based methods will continue to provide increased scientific insight. As we mentioned in the beginning: At the time that hardware progress doesn't seem to be promising in the future, it is necessary to reexamine our methodologies. Here we will deliver the big picture of various interaction and opputunities between machine learning and HPC.

A. Learning Everywhere

There are various way to interact ML and HPC. Using ML to learn from simulations and produce learned surrogates for the simulations. This increases effective performance for strong scaling. Here we will outline some common interactions in four categories

MLaroundHPC

- Learning Outputs from Inputs: Use performed simulation to train an AI system directly.
- Learning Simulation Behavior: Use ML surrogates to learn the system/model behavior. HPC system usually is computationally heavy. Once we have built the ML surrogates, we can apply them to learning the behaviors of the HPC system/model.
- Faster and Accurate PDE Solutions: As we discussed in the previous chapter. Applying ML algorithms to reduce the time costs on high dimensional PDE computations on HPC.
- New approach to Multi-Scale Modelling: Multi-Scale Modelling is a modeling method that uses multiple scales instead of one. The deployment of machine learning can simplify the process of finding potential Multi-Scale.

ML Control

- Experiment Control: Using ML to build the simulation surrogates for simulations, when simulations are used in the control of experiments and objective-driven computational campaigns.
- Experiment Design: Model-based design of experiments with new ML assistance can identify the optimal conditions for stimuli and measurements that yield the most information about the system given practical limitations on realistic experiments. In other words, it helps the model design by giving more information about the structures and parameters of ongoing projects.

ML Auto-Tunning

- ML can be used for a range of tuning and optimization objectives.

ML After HPC

- ML analyzing results of HPC as in trajectory analysis and structure identification in biomolecular simulations.

B. ML around HPC Classification and Exemplars

There are two major questions emerging from ML and HPC interaction:

- How to use the acquired data?
- How to acquire the data we want?

Therefore we have the urge to distinguish different modes and mechanics of how learning is integrated with HPC simulations. In other words, we need to distinguish different interactions of learning and HPC. The three primary modes and mechanisms for integrating learning with HPC simulations are:

- **Substitution:** A surrogate model is used to substitute an essential element of the original simulation (method). The surrogate model is used to create multi-scale or coarse-grained surrogate modeling, which could either learn the structure or theory of the original simulation. For example: Using a trained Neural Network from the original simulation can reduce the cost significantly compared to running it in the original simulation.
- **Assimilation:** In this mode, data from simulations, offline external constraints, or real-time experiments are integrated into physics-based models, which are then assimilated into traditional simulations. For example: In weather prediction, we constantly assimilate new, time-dependent, based on observations corrected simulations into the traditional simulation model.
- **Control and Adaptive Execution**In this mode, the simulation is controlled towards important and interesting parts of the simulation phase space. Sometimes this involves determining the parameters of the next stage (iteration) of simulations based on intermediate data. Sometimes the entire campaign can be adaptively steered towards an objective, which in turn could involve getting better data via active learning based upon an objective function. In other words, we steer the simulation towards the objectives so that we might have more fitting data for the campaign.

C. ML Around HPC Cyber Infrastructure

Analysis of ML Around HPC will be divided into three categories: (i) algorithms, benchmarks and methods; (ii) system software and runtime, and (iii) hardware.

Algorithms, Benchmarks and Methods

- Identification of Underlying Projects: Which traditional HPC simulations can be redesigned with ML? And if HPC simulations are going to serve as data sources, is there an opportunity to devise new ML algorithms so we can use those generated data better?
- Possible Improvement: Simulations are simply 4D time-series data. However ML doesn't have to be like that. There is space for improvement of better interaction between these two.
- Main Problem of a Project: How to understand which learning methods work. why and for which problems? How do we develop benchmarks? Furthermore, how do we develop proxy apps to represent the applications?
- Understanding Performance: What are the metrics that represent the performance of machine learning and simulations? How to define those metrics?

System Software and ML-HPC Runtime Systems
The interactions of ML and HPC simulations can be intertwined, so it is important to understand the control and coupling between Learning elements(L), and HPC Simulation (S). In many cases, a third general component: experiments or observations (E) may also be needed. There are some run-time requirements that have to be fulfilled. For example: switching actions depending on data size; Learning algorithms depending on remaining training time; And the need for scaling, since we work on two (possibly three) components; The concurrency of the whole system. Therefore a run-time system and software have to be accordingly developed.
Hardware and Platform Configuration

- Role and importance of heterogeneous accelerators: (i) Future generations of accelerators might not work perfectly for simulation. (current GPU accelerators are fine with both ML and simulation) (ii) When should RNN need different accelerators from CNN? (the importance of RNN is increasing, but now many accelerators are for CNN)
- Requirements of fast I/O and accordingly, fast and large disks.
- Latent optimization methods like fog computing.

D. Authors and Affiliations

Acknowledgment

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

References

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[4] K. Elissa, "Title of paper if known," unpublished.
[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.
[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.