ML in Computational Sciences and HPC

1st Ece Öztürk
Faculty of Mathematics, Computer Science and Statistics
Ludwig Maximilian University of Munich
Munich, Germany
Oeztuerk.Ece@campus.lmu.de

2nd Yan Liu
Faculty of Mathematics, Computer Science and Statistics
Ludwig Maximilian University of Munich
Munich, Germany
Yan.Liu@campus.lmu.de

***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-fashioned simulation-based solutions have confronted challenges as we could foresee the obsolescence of Moore's law [1] and Dennard scaling. The emergence of machine learning and big data unveils the possibilities of solving traditionally intractable problems by sidestepping. The usage of machine learning can not only solve some computational problems or discover underlying problems more efficiently but will also improve the HPC system's general performance as you can see in the later chapters. In this paper, we will first focus on two computational breakthroughs using machine learning as innovative solutions, namely solving matrix multiplication and PDE. Secondly, we will elaborate on the idea of MLaroundHPC which involves different interactions between machine learning and HPC. First and foremost, lets briefly explain some key concepts and terminologies.

- **Machine Learning and Deep Learning.** Machine learning is a technology that generates the underlying information ouf of a given big amount of Data without explicit instruction; Deep learning is a machine learning algorithm that uses multiple layers (called neural networks) to extract hiddle features from input data.
- **Hign Performance Computing.** HPC can perform complex calculations using high speed, and enable the 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 uses of machine learning.
- **Tensor** is a generalization of matrix and vector. A matrix can be viewed as 2D tensor. Vector can be viewed as 1D tensor. Scalar analogly as 0D. 3Dtensor can be considered as a vector of matrices as an entry instead of numbers.
- **Reinforcement Learning** is a type of machine learning that learns the optimal decision of a given environment. An agent, which is a decision maker, will receive a reward or punishment after its decision based on our evaluation. A famous example would be AlphaZero, which beats the top human player in shogi and chess.
- In computational physics and 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.

II. Background

A. Historical Developments and Approaches in ML around HPC

First of all, examining the historical development of approaches and applications in the fields of machine learning around HPC will allow the subject to be analyzed better. The history can be examined in different ways but we want to focus on significant milestones and evolving methodologies [2]. Understanding historical developments provides us with a versatile perspective to analyze the reasons for the difficulties and needs that arise over time and how they should be solved.

- **Early Integration (1950s-1980s):** The first stages of machine learning in HPC were characterized by rudimentary algorithms and limited computational capabilities. Early efforts focused primarily on rule-based systems and symbolic reasoning, with important work by researchers such as Marvin Minsky and John McCarthy.
- **Knowledge-Based Systems (1980s-1990s):** Knowledge-based systems that combine expert knowledge and heuristics are gradually coming to the fore. These systems aimed to capture domain-specific expertise and judgment. The integration of rule-based inference engines and symbolic representations formed the basis for the first applications of artificial intelligence in HPC.
- **Emergence of Neural Networks (1990s):** The increasing trend towards neural networks, supported by advances in computational power and algorithmic innovations, has led

to the scientific revolution. Researchers such as Geoffrey Hinton have significantly contributed to the development of deep learning architectures, leading to neural network applications in HPC.

- **Parallel and Distributed Computing (2000s):** The 2000s ushered in a major shift in machine learning with the emergence of parallel and distributed computing on HPC platforms. The increasing availability of clustered computing environments has made training larger and more complex models easier. Parallel algorithms have become widespread and leverage the power of distributed computing resources to train neural networks and ensemble methods.
- **Scalable Machine Learning Frameworks (2010s):** The growing need for big data and the growing trend towards scalable machine learning solutions have led to the development of designs that take advantage of the parallelism inherent in HPC architectures. Apache Hadoop and Spark have emerged as popular platforms that enable efficient processing of large data sets using distributed computing clusters.
- **Accelerated Computing with GPUs (2010s):** The introduction of Graphics Processing Units (GPUs) for machine learning tasks was a major milestone. GPUs designed for parallel processing have proven effective in speeding up training times for deep neural networks. This era has increased the computational efficiency of machine learning algorithms with the emergence of GPU-accelerated libraries such as CUDA.
- **Exascale Computing and ML (2020s and Beyond):** The current era can be summarized as the search for high-level computing capabilities that require unprecedented computing power. Machine learning in hyperscale systems simultaneously poses challenges and opportunities that require the development of algorithms and frameworks optimized for extreme parallelism. Research efforts are focused on addressing scalability, fault tolerance, and energy efficiency in machine learning applications in the context of exascale HPC.
- **Interdisciplinary Collaborations and Domain-Specific ML (Ongoing):** Contemporary developments require interdisciplinary collaborations between domain experts and computer scientists. Machine learning applications tailored to specific scientific fields, such as computational sciences, genomics, and climate modeling, showcase the versatility of machine learning in tackling complex challenges in HPC environments.

III. Discovering Faster Matrix Multiplication Algorithms With Reinforcement Learning

A paper [3] created an agent **AlphaTensor** using AlphaZero [4] and deep reinforcement learning to search for provably correct and efficient matrix multiplication algorithms.

A. Previous Challenges in Matrix Multiplication

High-order matrix multiplication is one of the most frequent and painful computations in science and engineering. Even though matrix multiplications can be formalized as low-rank decompositions of a specific three-dimensional(3D) tensor [5], called matrix multiplication tensor. Finding low-rank decompositions of 3D tensors(and beyond) is still NP-hard [6]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 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 multiplication 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:

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

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, modeling 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 subtracting 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 to 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

The whole idea is relatively straightforward. The significance lies in the engineering. A transformer-based [7] 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 grids. This generalized axial attention [8] to multiple grids is more

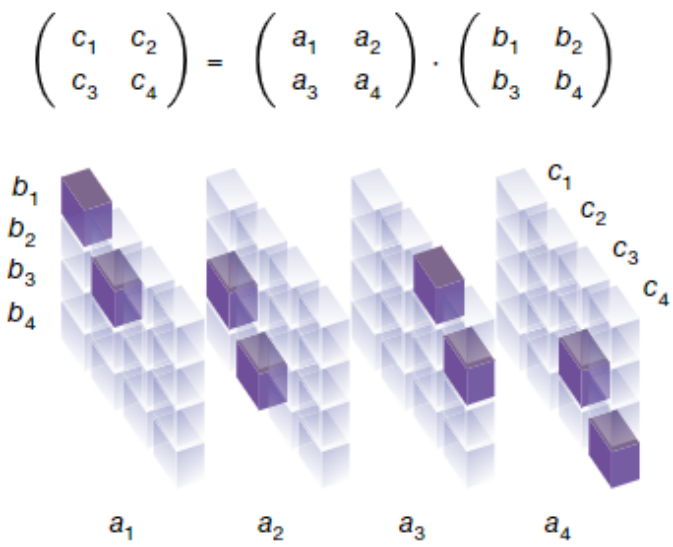

Fig. 1. $4 \times 4 \times 4$ Tensor $\mathcal{T}$ represents the muliplication of two 2×2 matrics. The purple box means tensor entries equal to 1 and the transparent box means 0. For instance $c_1 = a_1 b_1 + a_2 b_3$, means (a_1, b_1, c_1) and (a_2, b_3, c_1) are set to 1.

efficient and yields better results than naive self-attention. Apart from architecture, the use of data argumentation and change of basis also has improved the general performance.

IV. 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** [9], a novel deep learning architecture, which consists mostly of two previous works:

- Recent works of **Neural Operators** [10]–[14], 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. In theory, they appear in the proof of the universal approximation theorem [15]. Empirically, they have been used to speed up convolutional neural networks [16]. Recently, some spectral methods for PDEs [17], [18] 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 $G^\dagger : \mathcal{A} \to \mathcal{U}$ 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 $\mu_j = G^\dagger(a_j)$ will contain noises. Parametric map will be defined as follows:

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

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 $C : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ and also the minimizer:

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

B. Neural Operator and Fourier Neural Operator

The neural operator, proposed by [19], is an iterative learning architecture as presented in Figure2. P and Q are local transformations 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

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

where $\mathcal{K} : \mathcal{A} \times \Theta_{\mathcal{K}} \to \mathcal{L}\left(\mathcal{U}\left(D; \mathbb{R}^{d_v}\right), \mathcal{U}\left(D; \mathbb{R}^{d_v}\right)\right)$ 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 $\mathcal{K}$

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

where R_ϕ is the Fourier transform of a periodic function $\mathsf{K} : \overline{D} \to \mathbb{R}^{d_v \times d_v}$ parameterized by $\phi \in \Phi_{\mathsf{K}}$. More intuitive demonstration can be found in Figure 2 (b). The idea of (5) comes from the original **Kernel Integral Operator** in (Li et al., 2020b). which is defined in (6):

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

for $\forall x \in D$, where $\mathsf{K} : \mathbb{R}^{2(d+d_a)} \to \mathbb{R}^{d_v \times d_v}$ 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

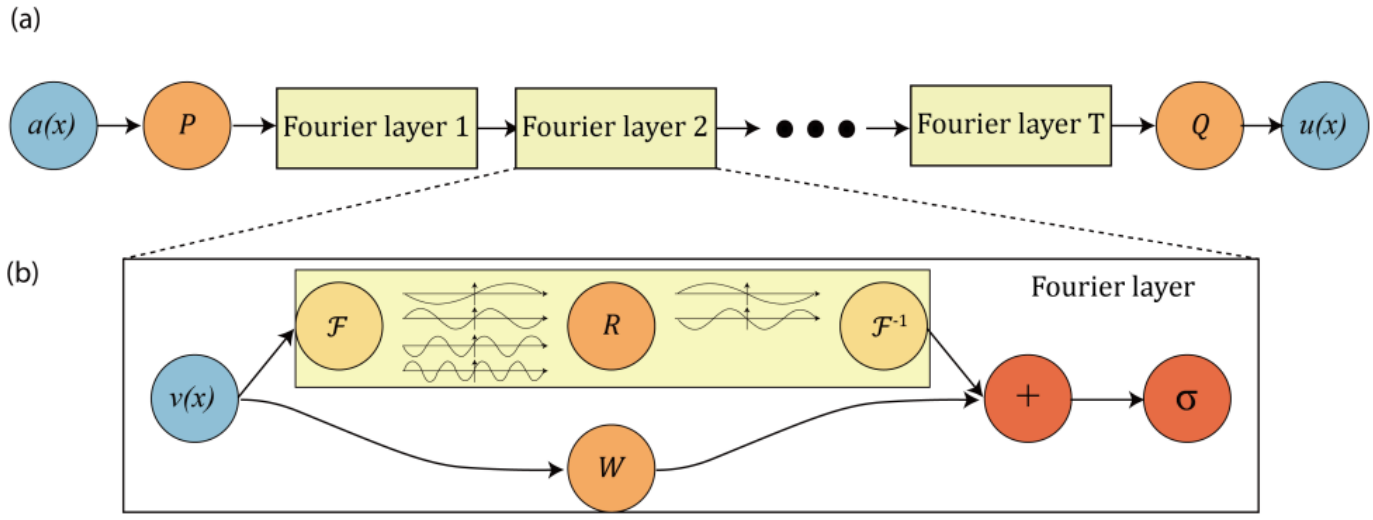

Fig. 2. **(a)The full architecture of neural operator:** start from input a. 1. Lift to a higher dimension channel space by a neural network P. 2. Apply four layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u.**(b) Fourier layers:** 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.

(5), we firstly denote $\mathcal{F}$ as the Fourier transform of a function $f : D \to \mathbb{R}^{d_v}$ and $\mathcal{F}^{-\infty}$ as its inverse and

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

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

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

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

which is our definition in (5).

C. Performance and Contributions

The Fourier neural operator is the ground-breaking work that can manage to learn the resolution-invariant solution operator for the family of Navier-Strokes equation in the turbulent regime. Despite that, the Fourier neural operator also outperforms all existing deep learning methods on 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 is also worth mentioning that the Fourier neural operator achieves 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.

V. 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 following four categories

MLaroundHPC

- Learning Outputs from Inputs: Use performed simulation to train an AI system directly [20], [21].
- 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 and model [22], [23].
- 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.

VI. Discussion

The trajectory of machine learning in High Performance Computing is emerging as multifaceted, with the methodological methods we describe in this article, and depicts a complex landscape that requires rigorous research. The discussion delves into the key opportunities and challenges inherent in the integration of machine learning into High Performance Computing. First, let's take a look at the opportunities it offers:
- **Scalability and Efficiency:** Properly applied ML techniques exhibit inherent scalability, allowing large datasets

and complex simulations to be processed with improved computational efficiency.

- **Adaptability to Diverse Domains:** The versatility of machine learning gives it wide application across a wide range of scientific disciplines. From climate modeling to materials science, machine learning techniques can adapt to the complexities of various data types and scientific problems and produce effective solutions to them.
- **Predictive Modeling and Optimization:** Machine learning algorithms are an excellent way to create predictive models, allowing optimization of simulations, resource allocation, and decision-making processes. This capability has the potential to revolutionize the way HPC resources are used.
- **Future Aspects:** Future research in HPC will prioritize improving ML algorithms for improved scalability, parallelization and efficiency. Hybrid approaches that integrate physics-based models with machine learning techniques are expected to gain even more importance in interdisciplinary studies. In particular, addressing the inherent uncertainty in ML predictions will be crucial and will require the development of probabilistic models and advanced uncertainty measurement techniques.

The Integration of ML with HPC has also many challenges:

- **Computational Overheads and Scalability:** The inherent computational demands of advanced machine learning models present challenges in terms of scalability, especially when dealing with large-scale simulations and large data sets. Future research should also consider computational overheads and optimize for efficient algorithms in HPC architectures.
- **Data Quality and Quantity:** Unfortunately, ML models' reliance on the quality and quantity of training data remains a persistent challenge. Obtaining representative datasets in scientific fields is often difficult and requires innovative strategies for data augmentation, synthesis, and curation.
- **Interpretability and Explainability:** The current era can be summarized as the search for high-level computing capabilities that require unprecedented computing power. ML in hyperscale systems simultaneously poses challenges and opportunities that require the development of algorithms and frameworks optimized for extreme parallelism. Research efforts are focused on addressing scalability, fault tolerance, and energy efficiency in machine learning applications in the context of exascale HPC.
- **Ethical Considerations:** The ethical implications of deploying ML in HPC contexts require meticulous attention. Future research should actively address issues of bias, fairness, and transparency and ensure that machine learning models do not perpetuate or exacerbate existing inequalities in scientific research. In addition, maximum data security should be ensured in the processing of large data that is difficult to control.

VII. Conclusion

Many different processor architectures have been designed as the consequences of hardware and architectural trends, such as the end of Dennard scaling and Moore scaling. As it becomes more difficult to control the increasing data size, achieving performance gains becomes increasingly important. This situation underlines the need for unsustainable software investment. In other words we can reach the limits of both hardware and methodological performance gains.It seems that this situation may oblige us to use ML driven HPC applications in the future, which paints an optimistic picture for the most difficult problems.

The recent exploration of data-centric innovations as alternatives to traditional computationally intensive approaches highlights a significant paradigm shift in scientific computing. The quest to increase efficiency and performance has led to an increased reliance on data-driven methodologies by leveraging the power of ML in the HPC context. This paper has highlighted important advances in data-centric approaches and underlined their potential as solutions to computationally heavy paradigms. The important mathematical foundations required for the HPC concepts and terminology behind the methods are also explained with various models in this paper.

The use of recent data-driven innovations, including but not limited to machine learning, shows promise in mitigating the challenges associated with traditional computational methods. Integration of these approaches has demonstrated improved scalability, adaptability, and computational efficiency, providing viable alternatives for solving complex scientific problems. Nuanced exploration of data-driven techniques such as neural networks, deep learning, and statistical modeling sheds light on their applicability in various scientific fields.

Moreover, the change towards data-centric methodologies has not only provided computational advantages but also opened avenues for interdisciplinary collaboration. The collaboration between domain-specific knowledge and advanced data-driven approaches has resulted in new insights and methodologies that transcend the limitations of traditional computational paradigms.

References

[1] T. N. Theis and H.-S. P. Wong, "The End of Moore's Law: A New Beginning for Information Technology," in *Computing in Science and Engineering*, vol. 19, no. 2, pp. 41-50, Mar.-Apr. 2017, doi: 10.1109/MCSE.2017.29.

[2] A. Jung, "Machine Learning: The Basics," *ArXiv:1805.05052*, 2018.

[3] A. Fawzi, M. Balog, A. Huang, et al., "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, 610, 47–53 (2022), https://doi.org/10.1038/s41586-022-05172-4.

[4] D. Silver, et al., "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, 362, 1140–1144 (2018).

[5] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, 13, 354–356 (1969).

[6] C. J. Hillar and L.-H. Lim, "Most tensor problems are NP-hard," *J. ACM*, 60, 1–39 (2013).

[7] A. Vaswani, "Attention is all you need," In *International Conference on Neural Information Processing Systems*, Vol 30, 5998–6008 (Curran Associates, 2017).

[8] J. Ho, N. Kalchbrenner, D. Weissenborn, and T. Salimans, "Axial attention in multidimensional transformers," Preprint at https://arxiv.org/abs/1912.12180 (2019).

[9] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier Neural Operator for Parametric Partial Differential Equations," *ArXiv:2010.08895*, 2020.

[10] Lu Lu, Pengzhan Jin, and George Em Karniadakis, "Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators," *arXiv:1910.03193*, 2019.

[11] Kaushik Bhattacharya, Nikola B. Kovachki, and Andrew M. Stuart, "Model reduction and neural networks for parametric pde(s)," preprint, 2020.

[12] N. H. Nelsen and A. M. Stuart, "The random feature model for input-output maps between Banach spaces," *arXiv preprint arXiv:2005.10224*, 2020.

[13] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Neural operator: Graph kernel network for partial differential equations," *arXiv preprint arXiv:2003.03485*, 2020.

[14] Ravi G. Patel, Nathaniel A. Trask, Mitchell A. Wood, and Eric C. Cyr, "A physics informed operator regression framework for extracting data-driven continuum models," *Computer Methods in Applied Mechanics and Engineering*, 373:113500, 2021.

[15] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. "Multilayer feedforward networks are universal approximators." *Neural Networks*, 2(5):359–366, 1989.

[16] Michael Mathieu, Mikael Henaff, and Yann LeCun. "Fast training of convolutional networks through ffts." 2013.

[17] Yuwei Fan, Cindy Orozco Bohorquez, and Lexing Ying. "Bcr-net: A neural network based on the nonstandard wavelet form." *Journal of Computational Physics*, 384:1–15, 2019.

[18] Karthik Kashinath, Philip Marcus, et al. "Enforcing physical constraints in CNNS through differentiable PDE layer." In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[19] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. "Neural operator: Graph kernel network for partial differential equations." *arXiv preprint* arXiv:2003.03485, 2020.

[20] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. "Boltzmann generators – sampling equilibrium states of Many-Body systems with deep learning." December 4, 2018.

[21] Katsuhiro Endo, Katsufumi Tomobe, and Kenji Yasuoka. "Multi-step time series generator for molecular dynamics." In *Thirty-Second AAAI Conference on Artificial Intelligence*. aaai.org, 2018.

[22] Wolfgang Gentzsch. "Deep learning for fluid flow prediction in the cloud." https://www.linkedin.com/pulse/deep-learning-fluid-flow-prediction-cloud-wolfgang-gentzsch/, December 2018. Accessed: 2019-3-1.

[23] Jiequn Han, Arnulf Jentzen, and Weinan E. "Solving high-dimensional partial differential equations using deep learning." *Proceedings of the National Academy of Sciences of the United States of America*, 115(34):8505–8510, 21 August 2018.