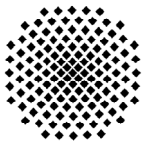


H L R S

Institut für
Höchstleistungsrechnen

FORSCHUNGS- UND ENTWICKLUNGSBERICHTE

HYBRID DEEP LEARNING APPROACHES ON HPC AND QUANTUM COMPUTING FOR DATA ANALYSIS

Li Zhong

ISSN 0941 - 4665 Juli 2024 HLRS-27

Höchstleistungsrechenzentrum
Universität Stuttgart
Prof. Dr.-Ing. Dr. h.c. Dr. h.c. Prof. E.h. Michael M. Resch
Nobelstrasse 19 - 70569 Stuttgart
Institut für Höchstleistungsrechnen

HYBRID DEEP LEARNING APPROACHES ON HPC AND QUANTUM COMPUTING FOR DATA ANALYSIS

von der Fakultät Energie-, Verfahrens- und Biotechnik der Universität Stuttgart zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

vorgelegt von

Li Zhong
aus Jiangsu, China

Hauptberichter:	Prof. Dr.-Ing. Dr. h.c. Dr. h.c. Prof. E.h. Michael M. Resch
Mitberichter:	Prof. Dr. Florina M. Ciorba
Tag der Einreichung:	04. Oktober 2023
Tag der mündlichen Prüfung:	16. Juli 2024

D93

ISSN 0941 - 4665 Juli 2024 HLRS-27

Abstract

In the past decade, machine learning, specifically deep learning methods started to revolutionize several application domains. From the point of view of simulation in engineering this revolution is especially interesting as it might allow to move from human planned simulation to computer analysed data usage and data evaluation for engineering simulations. This can help not only to shorten the simulation testing but also to better understand the simulation data themselves. This thesis will therefore present material science as practical example to better understand the potential of machine learning in engineering simulations. However, deep learning methods are impeded by computational and storage constraints, which quantum computing can provide potential revolutionary breakthroughs. Therefore, this thesis also explores the benefits that quantum technologies can bring to deep learning, and employs image classification and partial differential equation solving as specific case studies to examine the advantages of quantum neural networks (QNNs).

In order to tackle the issue of inadequate ground truth data for deep learning and reduce the demand for expertise, computing resources, and time in determining the validated parameters for simulation, this thesis proposes a novel hybrid data analysis methodology that combines deep learning and simulation on heterogeneous HPC clusters. While deep learning has demonstrated strong abilities at extracting high-level representations of complex processes, the lack of sufficient ground truth data is often a critical issue faced in various areas. In fact, it is almost impossible to generate enough data in real life for supervised learning in many real-world problems, which are limited by scientific instruments, the physical phenomenon itself, or the complexity of modeling. In this thesis, a new approach is proposed to solve such issues, which can improve accuracy, accelerate time to solution and significantly reduce the cost of time and effort. The effectiveness of this workflow is demonstrated through an experiment of material characteristic identification. The experiment shows that the proposed workflow and multi-task deep neural network (DNN) model determine material behaviours with a MSE loss of around 0.03, which is not only more accurate but also reduces the time, effort, and the requirement of deep domain knowledge compared to the state-of-the-art methods.

To address the challenge of lack of computation power and storage for deep learning, this thesis investigates the advantages that quantum computing can bring to DNN. As a practical example, a comparative study is conducted to demonstrate the benefits of QNN over classical DNN in image classification. Quantum computing offers the possibility of revolutionary breakthroughs, particularly in solving problems that are impossible or time-consuming for classical computers due to exponential or sub-exponential time complexity. QNNs integrate the principles of DNNs with the quantum computing

paradigm, thereby conferring several advantages over classical DNNs including quantum parallelism, exponentially increased memory capacity, faster learning capacity, higher stability and reliability, and faster information processing speed. The experiment results indicate that QNNs can achieve a rapid convergence rate without compromising accuracy.

However, the contemporary hardware limitations of quantum computing pose the challenge for effective data compression, particularly for a limited number of qubits. To overcome this issue, an effective data compression approach is proposed that reduces the dimensions of input data to fit in the limited number of qubits while retaining the most important information. The proposed approach develops a novel Transformer-GAN based model to compress information and enhance the performance of QNN on quantum computers. It is demonstrated that the proposed model can reconstruct images into a token representation and restore them almost perfectly to their original state. The method's effectiveness is also exhibited in extreme data compression scenarios for Cityscapes dataset in comparison with state-of-the-art methods.

Finally, this thesis proposes a quantum convolutional neural network to give a potential solution to a long-running problem: partial differential equations (PDEs). The proposed QNN based PDE solver is enabled fast optimisation with reduced memory requirements. The effectiveness of this method is demonstrated through an experiment on two practical problems: Burgers' Equation and Poisson Equation, which shows that a well-designed QNN can solve certain types of PDEs. The success of QNN in this work opens a promising paradigm by complementing physics based DNN model with an emerging new development of quantum computing, as well as providing a vivid reference for fusing quantum computing and deep learning algorithms, which can serve as the basis for further work towards quantum PDE solvers.

In summary, this dissertation explores the impact of hybrid workflows and DL methods on HPC and quantum computers. It proposes approaches that aim to address the challenges of existing deep learning methods, including the scarcity of data and the limitations of computing and storage resources. The methods proposed in this thesis might serve as the basis for further research and design solutions toward hybrid DL, HPC, and quantum methods in data analysis.

Kurzfassung

In den letzten zehn Jahren hat Maschinelles Lernen, insbesondere Methoden des Deep Learnings, zahlreiche Anwendungsbereiche revolutioniert. Für Simulationen aus dem Ingenieurbereich ist diese Revolution besonders interessant, da sie es ermöglichen könnte, die computergestützte Datennutzung und Datenauswertung technische Simulationen zu automatisieren. Dies kann nicht nur zur Verkürzung des Aufwands der Validierung von Simulationen beitragen, sondern auch zum besseren Verständnis der Simulationsdaten selbst. In dieser Arbeit wird die Materialwissenschaft und Werkstofftechnik als praktisches Beispiel herangezogen, um das Potenzial des maschinellen Lernens für ingenieurtechnische Simulationen besser zu verstehen. Trotz stetig zunehmender Rechenleistung sind einige Deep-Learning-Methoden durch Rechen- und Speicherbeschränkungen aktueller Höchstleistungsrechensysteme in ihrem Potenzial limitiert. Eine Möglichkeit, in diesem Kontext einen signifikanten Performanceschub zu erzielen, ist der Einsatz von Quantenrechnern. Daher untersucht diese Arbeit auch Vorteile, die Quantentechnologien für Deep Learning bieten kann. Als Fallstudien werden die Bildklassifizierung und das Lösen partieller Differentialgleichungen betrachtet, um die Vorteile von Quantenneuronalen Netzen (QNNs) zu untersuchen.

Um das Problem der unzureichenden Ground-Truth-Daten für Deep Learning zu lösen und den Bedarf an Fachwissen, Computerressourcen und Zeit bei der Bestimmung der validierten Parameter für die Simulation zu reduzieren, schlägt diese Arbeit eine neuartige hybride Datenanalysemethode vor, die Deep Learning und die Simulation auf heterogenen HPC-Clustern kombiniert. Deep Learning hat zwar gezeigt, dass es in der Lage ist, übergeordnete Repräsentationen komplexer Prozesse zu extrahieren, doch ist das Fehlen ausreichender Ground-Truth-Daten in verschiedenen Bereichen oftmals ein kritisches Problem. In der Tat ist es beinahe unmöglich, genügend Daten für überwachtes Lernen für viele reale Probleme zu generieren, die heutzutage durch wissenschaftliche Instrumente, physikalische Phänomene oder die Komplexität der Modellierung begrenzt sind. In dieser Arbeit wird daher ein neuer Ansatz zur Lösung solcher Probleme vorgeschlagen, der die Genauigkeit verbessern, die Zeit bis zur Lösung beschleunigen und die Kosten für Zeit und Aufwand erheblich reduziert. Die Wirksamkeit dieses Vorgehens wird anhand eines Experiments zur Identifizierung von Materialeigenschaften demonstriert. Das Experiment zeigt, dass der vorgeschlagene Vorgehen und das Multi-Task-Modell des tiefen neuronalen Netzes (DNN) das Materialverhalten mit einem MSE-Verlust von etwa 0,03 bestimmen, was nicht nur genauer ist, sondern auch den Zeitaufwand, die Anstrengungen und die Anforderungen an das tiefe Domänenwissen im Vergleich zu modernsten Methoden reduziert.

Um das zuvor angesprochene Problem der mangelnden Rechenleistung und des fehlenden Speichers für Deep Learning zu lösen, untersucht diese Arbeit die Vorteile, die Quantencomputing für DNN bringen kann. Als praktisches Beispiel wird eine Studie durchgeführt, um die Vorteile von QNN gegenüber klassischen DNN bei der Bildklassifizierung aufzuzeigen. Das Quantencomputing bietet die Möglichkeit eines revolutionären Durchbruchs, insbesondere bei der Lösung von Problemen, die für klassische Computer heutzutage aufgrund der exponentiellen oder subexponentiellen Zeitkomplexität unmöglich oder zeitaufwändig sind. QNNs integrieren die Prinzipien von DNNs mit dem Paradigma des Quantenrechnens und bieten dadurch mehrere Vorteile gegenüber klassischen DNNs: Quantenparallelität, exponentiell erhöhter Speicherkapazität, schnellerer Lernkapazität, höherer Stabilität und Zuverlässigkeit sowie schnellerer Informationsverarbeitungsgeschwindigkeit. Ergebnisse dieser Arbeit belegen, dass QNNs eine schnelle Konvergenzrate erreichen können, ohne die Genauigkeit zu beeinträchtigen.

Die gegenwärtigen Hardwarebeschränkungen des Quantencomputers stellen weiterhin eine Herausforderung für eine effektive Datenkompression dar, insbesondere bei einer begrenzten Anzahl von Qubits. Um dieser Herausforderung zu begegnen, wird in dieser Arbeit ein Ansatz zur Datenkompression vorgeschlagen, der die Dimensionen der Eingabedaten derart reduziert, dass sie in die begrenzte Anzahl von Qubits passen und gleichzeitig die wichtigsten Informationen erhalten bleiben. Der vorgeschlagene Ansatz entwickelt ein neuartiges Transformer-GAN-basiertes Modell, um Informationen zu komprimieren und die Leistung von QNN auf Quantencomputern zu verbessern. Es wird gezeigt, dass das vorgeschlagene Modell Bilder in einer Token-Darstellung rekonstruieren und sie fast perfekt in ihren ursprünglichen Zustand zurückversetzen kann. Die Effektivität der Methode wird auch in extremen Datenkompressionsszenarien im Vergleich zu State-of-the-Art-Methoden demonstriert.

Schließlich wird im Rahmen dieser Arbeit eine hybride Methodik vorgeschlagen, die Quanten-Deep-Learning und die Simulation auf HPC-Clustern kombiniert, um eine mögliche Lösung für ein mathematisches Problem zu finden: partielle Differentialgleichungen (PDEs). Der vorgeschlagene QNN-basierte PDE-Löser ermöglicht eine schnelle Optimierung mit reduzierten Speicheranforderungen. Die Wirksamkeit dieser Methode wird durch ein Experiment an zwei praktischen Problemen demonstriert: Burgersgleichung und die Poissongleichung. Anhand beider Gleichungen kann gezeigt werden, dass ein gut konzipiertes QNNs bestimmte Arten von PDEs lösen kann. Der Erfolg des QNN in dieser Arbeit eröffnet ein vielversprechendes Paradigma, indem es das auf der Physik basierende DNN-Modell mit der neuen Entwicklung des Quantencomputers ergänzt und eine anschauliche Referenz für die Verschmelzung von Quantencomputern und Deep-Learning-Algorithmen liefert, die als Grundlage für weitere Arbeiten an Quanten-PDE-Lösern dienen kann.

Zusammenfassend untersucht diese Dissertation die Auswirkungen von hybriden Workflows und DL-Methoden auf HPC- und Quantencomputer. Es werden Ansätze vorgeschlagen, die darauf abzielen, die Herausforderungen bestehender Deep-Learning-Methoden zu bewältigen, einschließlich der Datenknappheit und der Beschränkungen von Rechen- und Speicherressourcen. Die in dieser Arbeit vorgeschlagenen Methoden könnten als Grundlage für weitere Forschung und Designlösungen für hybride DL-, HPC- und Quantenmethoden in der Datenanalyse dienen.

Acknowledgements

This thesis is the result of the research work conducted at the High Performance Computing Center Stuttgart at the University of Stuttgart. It would not have been possible without the support of many people.

First of all, I would like to express my sincere gratitude to Prof. Michael Resch for his guidance, support, advice, and encouragement. I would like to thank him for introducing me into this interesting topic and guiding me through the whole research work. I would also like to thank Prof. Florina Ciorba for agreeing to be the co-reviewer and co-examiner of this thesis.

I am gratitude to Dennis Hoppe and Oleksandr Shcherbakov, many research works have resulted from the discussions and collaborations with them. I would also like to thank all my colleagues at the High Performance Computing Center Stuttgart including Dr. Bastian Koller, Neriman Emre, Dr. Jing Zhang, Dr. Huan Zhou, Qifeng Pan, Aihong Yin, Sergiy Gogolenko, Kamil Tokmakov, Thomas Beisel, and many others. It has been a great pleasure to know them and work with them.

Finally, I would like to thank my parents, my sister, and my wife - Lei Zhou for the continuous support, spur, as well as encouragement. I would also like to thank my son Taiyang Zhong who asked me several times per day if I had finished my dissertation after I told him that I can only play Pokemon with him when it is finished. Now, I can finally play with him.

Contents

List of Figures

List of Tables

Abbreviations

AutoML	**A**utomatic **M**achine **L**earning
PFS	**P**arallel**F**ile **S**ystem
POSIX	**P**ortable**O**perating **S**ystem **I**nterface
GAN	**G**enerative **A**dversarial **N**etwork
AI	**A**rtificial **I**ntelligence
CNN	**C**onvolutional **N**eural **N**etwork
NLP	**N**atural **L**anguage **P**rocessing
Matmul	Multi-Head Attention Module
Relu	**Re**ctified **L**inear **U**nit
DCT	**D**iscrete **C**osine **T**ransformation
Resnet	Residual Neural Network
RNN	**R**ecurrent **N**eural **N**etwork
CGAN	**C**onditional **GAN**
BPP	**B**it **P**er **P**ixel
SSIM	**S**tructural **S**imilarity **I**ndex **M**easure
PSNR	**P**eak-**S**ignal-to-**N**oise **R**atio
QCNN	**Q**uantum **C**onvolutional **N**eural**N**etwork
QNN	**Q**uantum **N**eural **N**etwork
NAS	**N**eural **A**rchitecture **S**earch
MTL	**M**ulti **T**ask **L**earning
SGD	**S**tochastic **G**radient **D**escent
FEM	**F**inite **E**lement **M**ethod
FEMU	**F**inite **E**lement **M**ethod **U**pdate
DWT	**D**iscrete **W**avelet **T**ransform
DCT	**D**iscrete **C**osine **T**ransform

JPEG	**J**oint **P**hotographic **E**xperts **G**roup
CFD	**C**omputational **F**luid **D**ynamics
MSE	**M**ean **S**quared **E**rror
L-BFGS-B	**L**imited **M**emory **B**royden-**F**letcher-**G**oldfarb-**S**hanno **B**ound Constraints
FFT	**F**ast **F**ourier **T**ransform
NISQ	**N**oisy **I**ntermediate-**S**cale **Q**uantum Computation
LSTM	**L**ong **S**hort-**T**erm **M**emory
GRU	**G**ated **R**ecurrent **U**nits
VAE	**V**ariational **A**uto**E**ncoders
GPU	**G**raphical **P**rocessing **U**nits)
NLP	**N**atural **L**anguage **P**rocessing

Chapter 1

Introduction

Deep learning methods are becoming increasingly important and driving revolutions in various areas. However, the performance of deep learning methods is highly dependent on the model and data size. Therefore, several critical issues are becoming the bottleneck of the development of deep learning, including the lack of ground truth data, the limitation of storage and compute power, *etc.* This thesis at hand proposes approaches to address these challenges and explores the impact and potential development trend of hybrid workflows and DL methods on HPC and quantum computers. In particular, a novel hybrid data analysis methodology is proposed to showcase the potential of deep learning methods in revolutionizing engineering simulations, which can address the problem of data sparsity for deep learning, and reduce the demand for expertise and time in determining validated parameters for simulation. This thesis also investigates the advantages of quantum computing in DNN by conducting a comparative study of QNN and classical DNN. In addition, an effective data compression approach is proposed to address the hardware limitations in current quantum computing. To demonstrate the power of QNN, a novel QNN method is proposed to give a potential solution to partial differential equations (PDEs). This chapter provides an introduction to the context of this thesis, where the motivation, the problem setting, and the main contributions are given.

Chapter outline: This chapter is organized as follows. Firstly the motivation is explained in Section 1.1, where the central topics related to this thesis are also described. Section 1.2 then summarizes the main contributions of this thesis. Finally, the organization of the thesis and the list of corresponding publications are provided in Section 1.4.

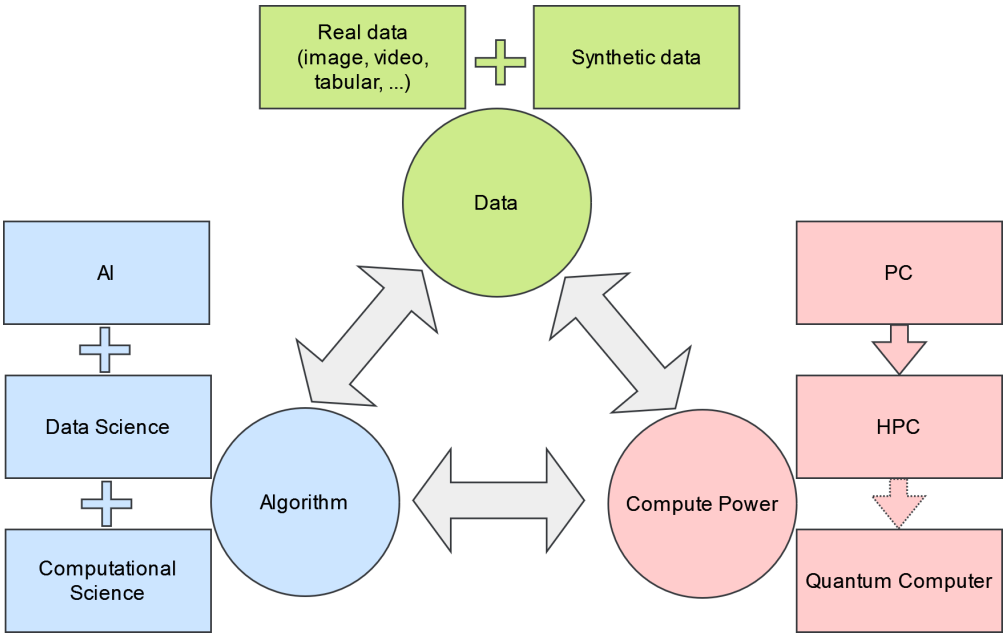


FIGURE 1.1: Data, algorithm, and compute power: the key factors for AI development

1.1 Motivation

Over the past decade, artificial intelligence (AI) has demonstrated strong abilities to extract high-level representations of complex processes, leading to significant improvements in accuracy and generalization. Moreover, among the various AI methods, deep learning methods have made significant advancements in revolutionizing several application domains, including image recognition [8], video analysis [9], natural language processing [10], *etc.* Data, computing power, and algorithm are the three major factors driving the development of artificial intelligence, especially deep learning[11], as is depicted in Figure 1.1. Massive data provides fuel for the development of artificial intelligence, the richness and large-scale nature of datasets are particularly important for algorithm training. Algorithms play a central role in the development of artificial intelligence. Excellent algorithms can improve the performance of the model, making it more accurate and efficient in handling problems. Computing power is also crucial, both the training of algorithms and the processing of massive data pose a high requirement of computing power. Significant computational strength can accelerate the training process of the model, allowing the model to converge and optimize faster. Moreover, the increase in computing power helps to process larger and more complex data.

However, several critical issues are faced by data, computing power, and algorithms, becoming the bottleneck of the development of deep learning.

The first is the limitation of storage and compute power: deep learning methods involve processing extra large datasets, which is not only a big challenge for the machine to

store and process but also makes the model harder to converge in a reasonable amount of time. In addition, training large-scale deep learning models with billions of parameters involves performing billions of mathematical operations, which require enormous computational resources. Furthermore, despite significant breakthroughs in GPU memory, its memory size remains relatively small concerning the model size. To address this issue, researchers have found that supercomputers can be a good fit. HPC systems are designed to provide greater computational power than traditional computing systems, making them well-suited for deep learning tasks that require large amounts of computational resources. By leveraging the parallel computing capabilities of HPC systems, deep learning models can be trained faster and more efficiently, leading to improved performance. Furthermore, HPC systems are capable of handling large and complex datasets that would be impossible to process on traditional computing systems. For example, it would take 355 years to train the GPT-3 model(175 billion parameters) on a single NVIDIA Tesla V100 GPU, while it takes only 34 days on an HPC cluster with 1024 NVIDIA A100 GPUs [12].

Another challenge for deep learning is the lack of sufficient ground truth data. In many real-world contexts, generating massive data needed for supervised learning is almost impossible due to limitations in scientific instruments, physical phenomena, or modeling complexity. Different methods have been developed to solve this problem, *e.g.,* transfer learning, data augmentation, synthetic data usage, new data generation through Generative Neural Networks(GAN), *etc.* Recently, scientists and engineers have begun experimenting with a relatively new approach: training a deep learning model with the virtual dataset produced by simulations. Studies [13–15] have proven that training deep learning models on data generated by simulation can improve accuracy, accelerate time to solution, and significantly reduce the cost.

It is evident and clear that HPC can help accelerate the development of deep learning. However, some challenges still exist in driving the convergence of HPC and deep learning. For example, scaling the deep learning workload on HPC systems is a complex problem. The parallelism strategy, communication, I/O system, *etc.* must be carefully designed to achieve the optimum performance on supercomputers. Another challenge is the difficulty in designing a hybrid simulation and deep learning workflow. It is quite challenging to generate data by simulation that is suitable for training a deep learning model. This often requires careful selection of simulation parameters, ensuring that the simulation produces data that is diverse and representative of the real-world problem. Furthermore, there is the issue of integrating the simulation and deep learning workflows, which often requires a good understanding of both simulation software and deep learning frameworks, as well as experience in designing data pipelines to manage the large volumes of data generated by simulations. Furthermore, there also exists the challenge of optimizing

the overall workflow to ensure that it is computationally efficient and can be scaled for larger simulations and models.

Therefore, this thesis explores the methods that enable the seamless execution of deep learning on the HPC system, which aims at achieving optimum performance and capitalizing on the power of HPC. Further, a novel hybrid workflow that combines a deep learning model and the simulation on HPC clusters with CPU and GPU support is proposed which can address the problem of data sparsity and reduce the demand for expertise, and time in determining the validated parameters for the simulation. The effectiveness of this workflow is demonstrated through a well-designed experiment to determine the material behavior. The experiment shows that the proposed workflow and multi-task deep neural network (DNN) model can accurately and readily identify the material characteristics, significantly reducing the time, effort, and requirement of deep domain knowledge.

However, even with the most advanced supercomputers, many of the problems remain intractable. Therefore, the possibility of quantum computing must be explored, which indicates revolutionary breakthroughs in computing, such as turning computationally inaccessible problems into tractable ones. For example, quantum systems are notoriously difficult to simulate using classical computers since the complexity of the simulation grows exponentially when the number of particles in the system grows, which is impossible to realize when the number of particles is large [16]. On the other hand, quantum computers are naturally suited for simulating quantum systems, as they can represent and manipulate quantum states in a natural way. Furthermore, many real-world optimization problems, such as the Traveling Salesman Problem [17], are difficult for classical computers to solve efficiently. In contrast, quantum computers have been shown to have the potential to speed up optimization. However, many previous quantum algorithms hold promise for the future when large-scale quantum computers exist with enough qubits for quantum error correction but cannot be implemented in the near term due to the required circuit depth. Thus, a crucial question is how to make use of such noisy intermediate-scale quantum (NISQ) computers and whether the implementation of quantum machine learning algorithms on the NISQs can bring benefits to practical problems. An interesting strategy to make use of NISQ devices is to employ variational hybrid quantum-classical algorithms (VHQCAs) [18], which are developed to reduce quantum circuit depth at the expense of additional classical optimization. Specifically, the cost function is evaluated through a short-depth quantum circuit, which is composed of a sequence of quantum gates that depend on the parameters that are optimized by leveraging well-established classical optimizers. Recently, a novel and applicable concept derived from Variational Quantum Machine Learning (VQML) has been proposed, known as quantum neural networks (QNNs). The development of QNNs combines the

basic principles of DNN and the quantum computing paradigm and shows the superiority over classical DNN. Recent research [19–21] finds that well-designed QNNs are able to achieve a higher capacity and faster training ability than comparable classical feedforward neural networks.

However, research has yet to be done to explore the quantum advantages brought by the QNNs through a performance comparison to classical DNNs on the same problem. Therefore, a comparative study has been conducted to show the advantage of QNN over classical DNN in data analysis, especially in the most commonly studied problem: image classification. The experiment results show that QNNs do not show significant advantages regarding model accuracy while they could achieve convergence much faster than the classical DNNs. However, it is also noticed that the number of qubits that quantum computers have is always not enough due to the current hardware limitation, which often raises the issue that the input information/training data can not be fully encoded. Furthermore, the performance of QNNs strongly depends on the methods used for information compression. Thus an effective data compression method is needed to reduce the dimensions of the input data so that the reduced data can fit into the qubits of the quantum computers while retaining the most important information. Therefore, a novel Transformer-GAN based model is developed to do information compression in order to improve the performance of QNNs on quantum computers with the restriction of a limited number of qubits.

Finally, we explore the new possibilities that the combination of quantum computing and deep learning can bring to solving classical supercomputer problems, a novel QNN method is proposed to solve partial differential equations (PDEs) which no previous study has been done. The design of QNN allows for its efficient training and implementation on realistic, near-term quantum devices. The effectiveness of this method is demonstrated through experiments on 1D Burgers' equation and 2D Poisson equation, which shows that a well-designed QNN can solve certain types of PDEs. Further, it proves that the combination of quantum computing and deep learning can bring new solutions and ideas to many problems that are bottlenecked by current algorithms and computing power.

1.2 Contribution

The main contributions of this thesis focus on the conceptual design, the evaluation, and the demonstration of effectiveness through real-world use cases. The details of the contribution and its corresponding chapter are listed below:

1. Different techniques that enable the seamless execution of deep learning applications on HPC are overviewed and taxonomized. Different methods and frameworks for environment setup, orchestration, parallelization, and optimization are also explained and compared.

2. A hybrid simulation and deep learning workflow on HPC is proposed, which solves the problem of lacking data in deep learning and the problem of high demand for expertise, and time-consuming in determining the validated parameters for the simulation. To demonstrate the effectiveness of the hybrid workflow, a multi-task DNN is proposed in combination with finite element method (FEM) simulations on HPC to identify the characteristics of the material. Furthermore, this solution is generalized through the implementation of an automatic machine learning method.

3. A comparative study is conducted to evaluate the performance of classical DNNs and QNNs on the problem of image classification, which presents the advantage of quantum neural networks on processing speed.

4. A novel Transformer-GAN based model is proposed as an effective data compression technique, which can highly reduce the dimension of the input data while retaining the most important information. The proposed method allows the reduced data to fit into the limited qubits of the current quantum computers without sacrificing important information.

5. A quantum neural network is proposed to solve the partial differential equations, which no previous study, to the best of the author's knowledge and through search in peer-reviewed databases, has done before. The effectiveness of the QNN is demonstrated by investigating its performance on two practical problems: 1D Burgers' equation and 2D Poisson equation. The success of QNN opens up a promising paradigm by complementing the physics-based DNN model with the emerging new development of quantum computing, which can serve as the basis for further work towards quantum PDE solvers.

1.3 Experiments Overview

In this thesis, there are multiple experiments conducted from various directions to prove the effectiveness of hybrid workflow and methodology of deep learning, HPC, and quantum computing, which can address the problems of existing methods.

The first two experiments are designed to solve the main problems faced in deep learning and simulation: 1. lack of enough ground-truth data; 2. high demand for expertise, resources, and time in determining the validated parameters for simulation. The whole

workflow is divided into three phases: data generation, training phase, and reverse phase. The material characteristic identification is taken as the use case. In the first experiment, a simulation is designed to generate the data. The generated data from the simulation is then used as the training data for a multi-task learning neural network, which can be further used to determine the material characteristics. In the second experiment, the multi-task learning neural network is replaced with a neural architecture search(NAS) based AutoML method, so that the effort in designing the network structure and hyperparameter is eliminated. The results prove that the combination of deep learning and simulation can help solve complex problems that are not feasible with only computational science or data science. For more details, please refer to Chapter 4.

Although simulation-enhanced deep learning can help solve a lot of problems, still some problems cannot be solved with even the most advanced HPC system due to the physics limitation, therefore the idea of introducing quantum computing is raised. The next experiments are to evaluate the effectiveness of introducing quantum computing in deep learning. Three types of variational quantum neural network are implemented on quantum computers, and their performance on image classification are compared with their corresponding classical neural network. The results show that there exist advantages of QNN over classical DNN for tasks like data analysis, especially in the most commonly studied problem: image classification. More details can be found in Chapter 5

When analyzing and discussing the experiments described in Chapter 5, it is discovered that the performance of the quantum neural network strongly depends on the data compression method, due to the fact that input data must be compressed and encoded to fit into the limited number of the qubits of current quantum computers. Thus, in Chapter 6, an experiment is conducted with the proposed novel data compression method, which is based on the combination of Transformer and GAN. In this experiment, the effectiveness of the method for image compression is tested on the dataset Cifar10 and Cityscape. The training strategies are also discussed for the proposed architecture to achieve the best compression and best performance.

The final experiment aims to demonstrate the potential of combining quantum computing and deep learning in addressing challenging classical HPC problems. Specifically, a QNN is proposed to solve PDEs. To evaluate its effectiveness, the QNN's performance is examined on two real-world problems: the one-dimensional Burgers' equation and the two-dimensional Poisson equation. The success of QNN opens up a promising paradigm by complementing the physics-based DNN model with the emerging new development of quantum computing, which can serve as the basis for further work towards quantum PDE solvers. More details can be found in Chapter 7.

1.4 Organization and Publications

1.4.1 Organization

This thesis is organized in the following way. Chapter 1 provides the basic introduction to the context of this thesis, where the motivation is given. Chapter 2 describes some basic knowledge of deep learning, supercomputing, and quantum computing, which are necessary to understand this thesis. This is followed by the survey of different methods that drive the convergence of supercomputing and deep learning in Chapter 3. Chapter 4 presents the proposed method of hybrid simulation and deep learning workflow, which is used to identify the material characteristics. In Chapter 5, the quantum advantage obtained by quantum neural networks is evaluated through a comparative study of classical DNNs and QNNs on image classification. Chapter 6 then presents a Transformer-GAN based image compression method, which can be used to compress the input data to fit into current quantum hardware. In Chapter 7, the proposed QNN method for solving the PDEs is described. Finally, in Chapter 8, the concluding remarks as well as the future outlook, are provided.

1.4.2 Publication

The contributions presented in Chapter 3 are closely related to the following publications and mainly appear in (1):

(1) **Li Zhong**, Naweiluo Zhou, Dennis Hoppe, Sergiy Gogolenko, and Oleksandr Shcherbakov. "Convergence of HPC and AI: A Survey." In submission to Future Generation Computer System.

(2) **Li Zhong**, Naweiluo Zhou, Dennis Hoppe, Sergiy Gogolenko, and Oleksandr Shcherbakov. "Scaling Deep Learning Workloads on HPC: Approaches and Challenges." In submission to IJCNN: International Joint Conference on Neural Network 2024.

(3) Naweiluo Zhou, Yiannis Georgiou, Marcin Pospieszny, **Li Zhong**, Huan Zhou, Christoph Niethammer, Branislav Pejak, Oskar Marko, and Dennis Hoppe. "Container orchestration on HPC systems through Kubernetes." Journal of Cloud Computing 10, no. 1 (2021): 1-14.

(4) Naweiluo Zhou, Yiannis Georgiou, **Li Zhong**, Huan Zhou, and Marcin Pospieszny. "Container orchestration on HPC systems." In 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), pp. 34-36. IEEE, 2020.

In Chapter 4, the contributions are closely related to the following publications and mainly appear in (5):

(5)**Li Zhong**, Dennis Hoppe, Naweiluo Zhou, and Oleksandr Shcherbakov. "Hybrid workflow of Simulation and Deep Learning on HPC: A Case Study for Material Behavior Determination." In 2021 IEEE International Conference on Cluster Computing (CLUSTER), pp. 698-704. IEEE, 2021.

(6) **Li Zhong**, Oleksandr Shcherbakov, Dennis Hoppe, Michael Resch, and Bastian Koller. "Towards Seamless Execution of Deep Learning Application on Heterogeneous HPC Systems." In Data Science in Applications, pp. 233-252. Cham: Springer International Publishing.

The approaches detailed in Chapter 5 are closely related to the following publications and mainly appear in (7):

(7) **Li Zhong**, Qifeng Pan, and Dennis Hoppe. "Is QNN Really Beneficial in Image Processing? A Comparative Study." In submission to IJCNN: International Joint Conference on Neural Network 2024.

In Chapter 6, the contributions are closely related to the following publications and mainly appear in (9):

(8) **Li Zhong**. A Method for Stream Data Analysis. In: Resch, M.M., Wossough, M., Bez, W., Focht, E., Kobayashi, H. (eds) Sustained Simulation Performance 2019 and 2020. Springer, 2020.

(9) **Li Zhong**, Mark Zwisler. TransGAN: A Transformer-GAN Based Model for Image Compression. In submission to The Asian Conference on Machine Learning (ACML).

In Chapter 7, the contributions mainly appear in:

(10) **Li Zhong**, Qifeng Pan, and Dennis Hoppe. "Physics-informed Quantum Neural Network for Solving Partial Differential Equations." In submission to Nature Physics.

The following publications are not directly part of this thesis, but are generally on the topic of hybrid HPC and deep learning, thus related to this thesis:

(11) Naweiluo Zhou, **Li Zhong**, Dennis Hoppe, Branislav Pejak, Oskar Marko, Javier Cardona, Mikolaj Czerkawski et al. "CYBELE: A Hybrid Architecture of HPC and Big Data for AI Applications in Agriculture." HPC, Big Data, and AI Convergence Towards Exascale: Challenge and Vision (2022): 255.

(12) Ilias Gialampoukidis, Stelios Andreadis, Nick Pantelidis, Sameed Hayat, **Li Zhong**, Marios Bakratsas, Dennis Hoppe, Stefanos Vrochidis, and Ioannis Kompatsiaris. "Parallel DBSCAN-Martingale Estimation of the Number of Concepts for Automatic Satellite Image Clustering." In International Conference on Multimedia Modeling, pp. 95-106. Springer, Cham, 2022.

(13) Tianbai Chen, **Li Zhong**, Naweiluo Zhou, and Dennis Hoppe. "Catch Weight Prediction for Multi-Species Fishing using Artificial Neural Networks." In 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 1545-1552. IEEE, 2021.

(14) Dennis Hoppe, **Li Zhong**, Stefan Andersson, and Diana Moise. "On the Detection and Interpretation of Performance Variations of HPC Applications." In Sustained Simulation Performance 2018 and 2019, pp. 41-56. Springer, Cham, 2020.

(15) Yiannis Georgiou, Naweiluo Zhou, **Li Zhong**, Dennis Hoppe, Marcin Pospieszny, Nikela Papadopoulou, Kostis Nikas et al. "Converging HPC, Big Data and Cloud technologies for precision agriculture data analytics on supercomputers." In International Conference on High Performance Computing, pp. 368-379. Springer, Cham, 2020.

(16) Petros Anastasiadis, Nikela Papadopoulou, Goumas Goumas, Nectarios Koziris, Dennis Hoppe, **Li Zhong** " PARALiA: A Performance Aware Runtime for Auto-tuning Linear Algebra on heterogeneous systems." In ACM Transactions on Architecture and Code Optimization, 2023.

Chapter 2

Background

This thesis describes the research on the hybrid workflow of DL workloads on HPC systems and quantum computers. Thus, it is necessary to understand the basics of deep learning, supercomputing, and quantum computing. To make the research presented from Chapter 3 and onwards accessible to people new to deep learning, HPC, and quantum information, some basic definitions will be given and discussed in this chapter.

Chapter outline: This chapter is structured in 4 sections. Section 2.1 provides the introduction to deep learning basics, including the basics concepts, the network structures, the optimization methods, *etc.* This is followed by the introduction of the basics of supercomputing in Section 2.2, in which the HPC system architectures, the interconnect network, the I/O and file systems, the batch system, and the HPC systems used for this thesis are described. Section 2.3 explains the basics of quantum computing, including qubits and superposition, quantum computation and quantum circuits, quantum algorithms, *etc.* In Section 2.4, the concluding remarks are provided.

2.1 Deep Learning

A system is called intelligent if it can solve problems independently and efficiently. The degree of intelligence depends on the degree of autonomy, the degree of complexity of the problem, and the degree of efficiency of the problem-solving process [22]. Artificial Intelligence (AI) provides systems with the ability to automatically learn and improve from experience to solve complex problems independently [23]. In the past years, AI has shown success in various fields, such as natural language processing (NLP) [24], computer vision (CV) [25] and robotics [26]. The technologies and techniques developed in AI research stem from four main aspects: cybernetics, symbolic and sub-symbolic, and statistical machine learning (ML).

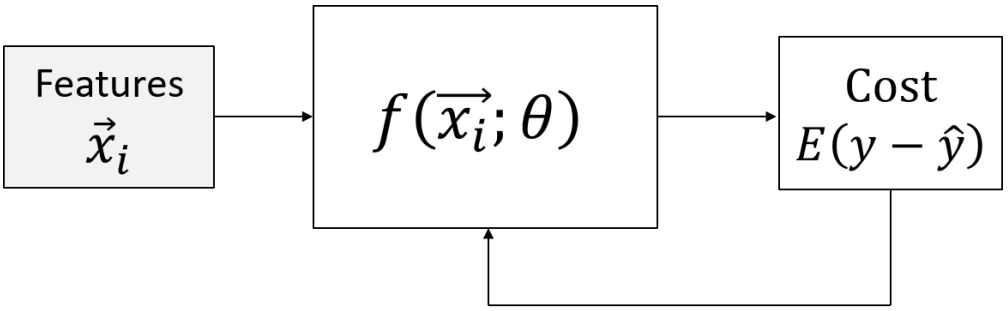


FIGURE 2.1: ML workflow: features are extracted and fed to the model which highly depends on the features and some parameters, then the cost is calculated which is the difference between the real output and the expected output. Afterwards, the model is updated to reduce the cost, so that the accuracy could be improved.

ML, as a subset of AI, is a method that builds a mathematical model to perform predictions or decisions without being explicitly programmed. The basic premise of machine learning is to build algorithms that can receive input data, recognize patterns hidden behind the data, and use statistical analysis to predict an output while updating outputs as new data becomes available. ML algorithms are mainly divided into several categories, *e.g.,* supervised learning, unsupervised learning, reinforced learning, semi-supervised learning, active learning, *etc.* Figure 2.1 illustrates how the ML models work.

Big Data is another term growing in popularity and is often mentioned together with ML. It has been coined to describe the large amount of data generated and collected. Big data analytics usually refers to discovering patterns from a tremendous amount of data, which is often faced with several challenges, *e.g.,* high dimensionality, scalability of algorithms, fast-moving streaming data, noisy and poor quality data, *etc.* Therefore, it is difficult for humans to understand and work on it, while ML algorithms can help overcome these challenges by automatically detecting patterns in the data. ML techniques have been widely deployed to explore the predictive feature of Big Data in many fields, such as medicine, the Internet of Things (IoT), search engines, and much more. To deal with Big Data analytics, a sub-field of machine learning known as deep learning is used to extract useful data from the Big Data.

Deep learning (DL), now often regarded as a new research area, is a class of ML algorithms that uses multi-layer neural networks, *i.e.,* deep neural networks (DNNs), to achieve the goal of ML. The underlying architecture of deep learning was inspired by the structure of the human brain in order to create a system similar to how humans learn. Compared with classical ML algorithms, DL algorithms have several advantages: DL works with both structured and unstructured data, while ML usually works only with sets of structured and semi-structured data; DL can perform much more complex operations efficiently; DL algorithms can analyze the input data and extract features automatically, while humans usually do this work for most ML algorithms. With the

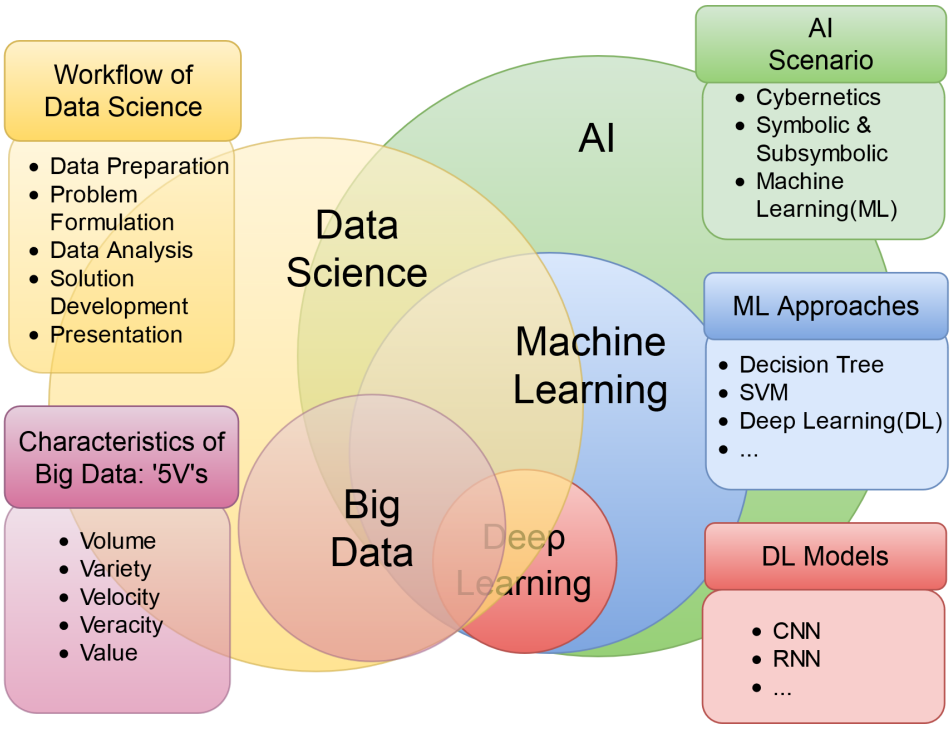


FIGURE 2.2: Relationship among AI, machine learning, deep learning, data science, and big data

great power provided by DL algorithms, DL algorithms have shown superior learning and classification performance in various fields and become a powerful tool to drive innovation and change across all aspects of our lives.

The relationship between AI, ML, DL and Big Data can be visualized in Figure 3.4. In the following, the DNN will be introduced by explaining the building blocks and different architectures of NNs in Section 2.1.1 and 2.1.2. Further, Section 2.1.3 will discuss the optimization techniques used for training the DNN architectures.

2.1.1 Artificial Neurons

In the early stage of AI, much research was conducted to create systems to imitate how humans learn based on the structure of the brain. A human brain is composed of billions of interconnected nerve cells, namely neurons, which are involved in the processing and transmitting of chemical and electrical signals. During the learning process, multiple signals arrive at the dendrites and are then integrated into the cell body, and if the accumulated signal exceeds a certain threshold, an output signal is generated that will be passed on by the axon. Based on this, the fundamental terminologies within DL were derived, and the computational unit within DL that allows the modelling of nonlinear functions was developed. In 1943, Warren McCullock and Walter Pitts published their first concept of a simplified brain cell, namely McCullock-Pitts (MCP) neuron, which

can be described as a simple logic gate with binary outputs [27]. Fourteen years later, perceptron was introduced by Frank Rosenblatt in 1957 based on the original MCP neuron [28], which is referred to as an artificial neuron today and is still used for DNN as a basic component.

An artificial neuron is a mathematical function based on a model of biological neurons, as illustrated in Figure 2.3, where each neuron takes N inputs $\{x_1, x_2, ..., x_n\}$ with separate weights $\{w_1, w_2, ..., w_n\}$, where $w_i \in \mathbb{R}$. In addition, the neuron is equipped with a bias $b \in \mathbb{R}$. They are summed up together, and then passed through a nonlinear function to produce the output:

$$y = \kappa \sum_i w_i x_i + b \tag{2.1}$$

where the nonlinear function is $\kappa(z)$ is called activation function.

Similar to the way that a neuron works described above, the artificial neuron takes the inputs as N arguments with different importance w_i. Then the activation function $\kappa((z))$ and the bias b define the threshold of the artificial neuron. If the weighted sum $\sum_i w_i x_i + b$ is above the threshold, then the neuron is 'activated', and the output is propagated to the next neuron. Otherwise, it is 'deactivated', and the propagation stops here.

It can be easily drawn out that the activation function has a significant impact on the output of the artificial neuron, even the same input and bias but with different activation functions can have totally different outputs for a neuron. Therefore, choosing the right activation is crucial for an expected training result. In the following, some of the most commonly used activation functions will be discussed.

Binary Step Function: The binary step function was the activation function initially adopted by Rosenblatt in perceptron [28]. As shown in Figure 2.4 (A) and Equation 2.2, the output of the step function is binary, either 0 or 1.

$$\kappa(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.2}$$

However, the step function is non-differentiable at $x = 0$ and the derivative is always 0 when $x \neq 0$, which means that that gradient descent won't be able to make a progress in updating the weights, so gradient-based optimization methods (which is how most neural networks are trained, see Section 2.1.3) would not work. Furthermore, since the main objective of the neural network is to learn the values of the weights and biases so

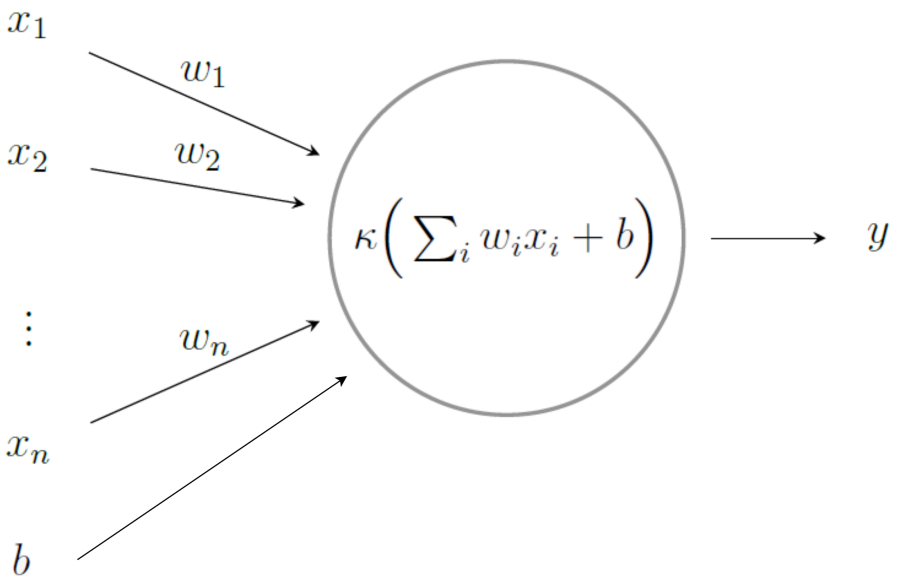


FIGURE 2.3: An artificial neuron which takes the sum of n input parameters and a bias as input of the activation function κ, the output of the activation function is the output of the neuron y

that the model could produce a prediction as close as possible to the real value, it is required that a small change in the weight or bias to cause only a small corresponding change in the output from the network. However, with step function, a small change in the weights or bias would lead to a complete change of the output, which damages the training process. Therefore, nowadays step function is rarely used as the activation function, there exist several options to choose from.

Linear Function: The equation for a linear activation function is shown in Figure 2.4 (B), typically written as:

$$\kappa(x) = \alpha x \tag{2.3}$$

where α is the weight. It is quite obvious that the derivative of a linear function is constant, which is not able to reflect how the model is changed according to the input data when doing back-propagation. Besides, taking the linear function as activation function does not introduce any non-linear change in the model, which means that the model can only perform like a classical linear regression/classification algorithm.

Sigmoid Function: Sigmoid activation functions can map any real-valued number to a value between 0 and 1, which is non-linear and able to capture more complex relationships than linear functions. As shown in Figure 2.4 (C), the equation for a linear

activation function is typically written as:

$$\kappa(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

However, Sigmoid activation functions suffer from the "vanishing gradient problem", meaning that the output of the function can become very small as the input increases. This can cause difficulty for the neural network to learn and adapt to new data. In addition, Sigmoid functions can also be computationally expensive to compute as they involve exponential.

hyperbolic tangens(TanH) Function: Tanh is another non-linear function commonly used in DL, which produces values ranging from -1 to 1. The equation for TanH activation function is typically written as:

$$\kappa(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.5}$$

As shown in Figure 2.4 (D), it has a smooth gradient making it easier to backpropagate errors during training. Further, it can help to prevent the issue of vanishing gradients. However, Tanh can suffer from saturation, where the function's output gets stuck at either -1 or 1, leading to slow learning.

Softmax Function: The softmax function is an activation function that takes a vector of real-valued inputs and squashes them to a range between 0 and 1, such that all the outputs sum to 1. As shown in Figure 2.4 (E), the equation for Softmax activation function is typically written as:

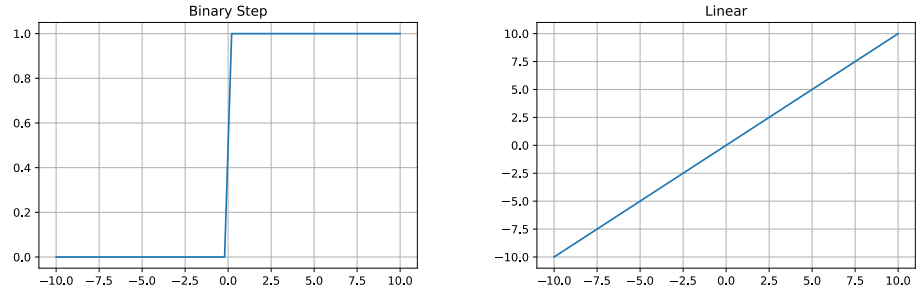
$$\kappa(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.6}$$

It is invariant to monotonic transformation of inputs, meaning that the result of the function does not depend on the scaling or shifting of the inputs. Softmax function is often used as the activation function of output layers in classification problems. However, it is prone to overfitting due to its ability to produce probability distributions.

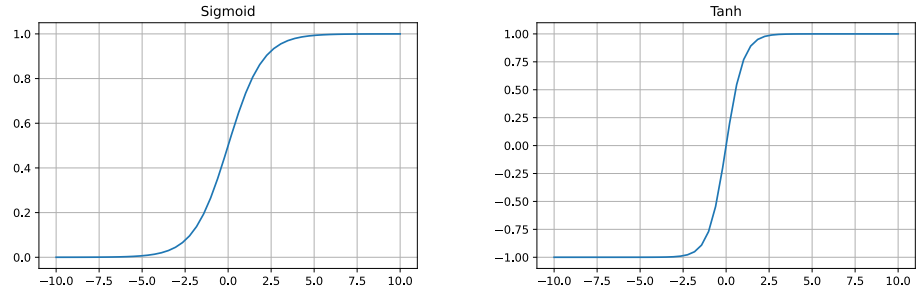
ReLu Function: ReLu function deactivates the neuron if the output is smaller than zero, otherwise performs as a linear function. The equation for ReLu activation function is typically written as:

$$\kappa(x) = max(0, x) \tag{2.7}$$

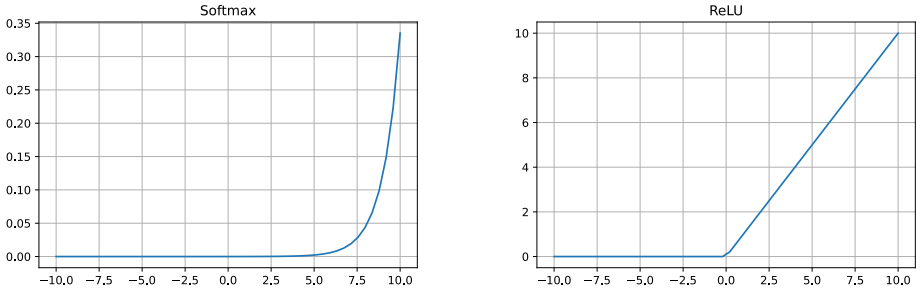
ReLu function is computationally efficient as compared to other activation functions. However, as shown in Figure 2.4 (F), ReLU can result in a dead neuron if the input is



(A) Binary Step (B) Linear



(C) Sigmoid (D) Tanh



(E) Softmax (F) ReLU

FIGURE 2.4: Different kinds of activation functions

negative, which makes it difficult to learn in certain situations. In addition, it is not suitable for the output layer in cases where a probability value is needed.

2.1.2 Network Architecture

Perceptron represents how a single neuron works. When a series of perceptrons are stacked in a row and piled in different layers, it becomes a multi-layer Neural Network (NN). The neurons in NN are arranged layerwise, as shown in Figure 3.4, where the first layer of neurons is the input layer which, as the name indicates, takes the input data. The last layer of neurons is named the output layer, which is the final output of the network. The layers in between are called hidden layers. In this section, some popular methods and architectures for building the multi-layer NN is introduced.

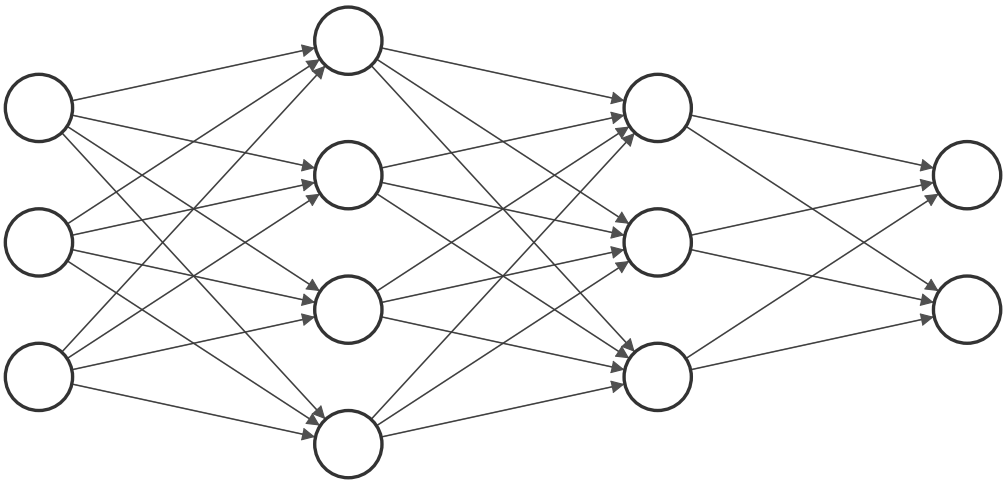


FIGURE 2.5: A feed-forward deep neural network

Feedforward Neural Network(FFNN) Feedfoward neural network is the most straightforward architecture of DNN. It is called a feedforward network because the data flows in only one direction, from the input layer to the output layer, through the hidden layer(s). The structure of the feedforward network is such that the information processed in each layer is fed to the next layer, and it is not processed again in the previous layer.

Convolutional Neural Network(CNN) is one of the most common FFNN architectures, which are widely used in computer vision and have become the state of the art for many visual applications. The key idea behind CNNs is to extract features from an image using multiple filters that scan the image in a sliding window manner and perform a convolution operation. A convolution operation involves element-wise multiplication of the values in the filter with the values in the image and then summing them up, which generates a new feature map that highlights certain aspects of the image. Pooling is another operation in CNNs that is used to down-sample the feature maps and reduce the spatial dimensions while retaining the most important features. It also has the effect of making the network more robust to small changes in the input, as well as reducing the computational cost of processing large images.

Recurrent Neural Network(RNN) is specifically designed to handle sequential data, such as time series data, speech signals, and text. Different from FFNN, the key feature of RNNs is that they have a hidden state that is passed from one time step to the next, allowing them to capture information from previous steps and use it to inform their predictions. RNNs can be unrolled over time to form a deep network to capture long-term dependencies in the sequential data. However, this can also lead to the gradient vanishing or gradient exploding, where the gradients become very small or very large during backpropagation, making it difficult for the network to learn. To overcome this issue, variants of RNNs, such as LSTMs (Long Short-Term Memory) [29] and GRUs

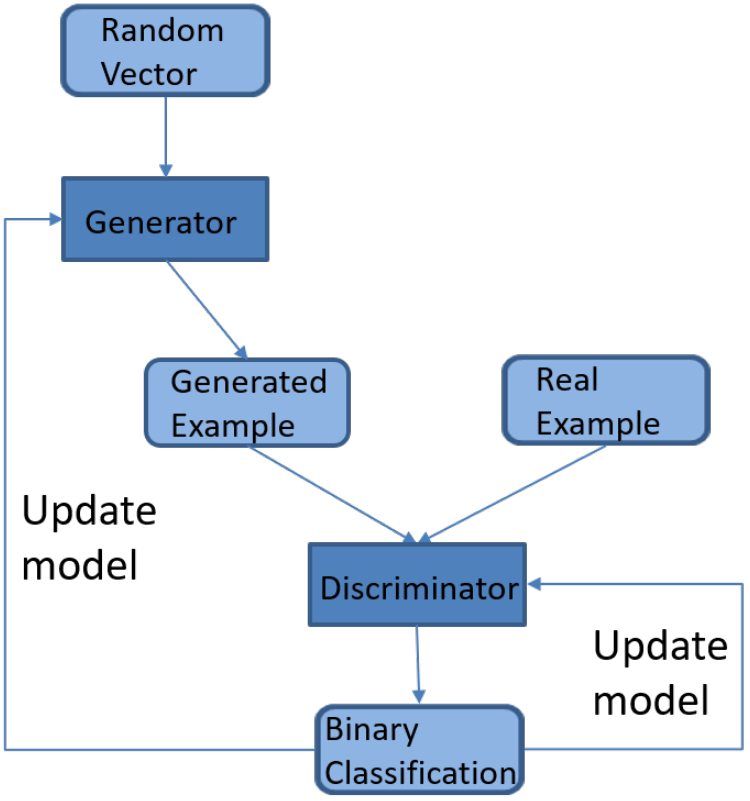


FIGURE 2.6: Architecture of generative adversarial networks

(Gated Recurrent Units) [30], have been developed to use gates to control the flow of information in the network.

Generative Adversarial Networks (GAN) is a generative model that was first introduced by Ian J. Goodfellow in 2014 [31, 32]. The model is built to bypass the difficulties of deep learning models with tasks of approximating many probabilistic computations, and tackle problems using the advantage of piece-wise linear units in a generative context. As illustrated in Figure 2.6, GAN contains two parts: the generative model, namely generator, and the discriminative model, which is called the discriminator. The generator tries to fool the discriminator by generating real-like images, and the discriminator tries to figure out if it is a real image or an image built by the generator, and gives feedback accordingly. Mathematically the generator is described as the joint probability distribution of the input variable and the output variable as seen in Equation 2.8.

$$\begin{aligned} \min_G \max_D V(G,D) &= \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z))] \\ \max_D V(D) &= \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z))] \\ \min_G V(G) &= \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z))] \end{aligned} \tag{2.8}$$

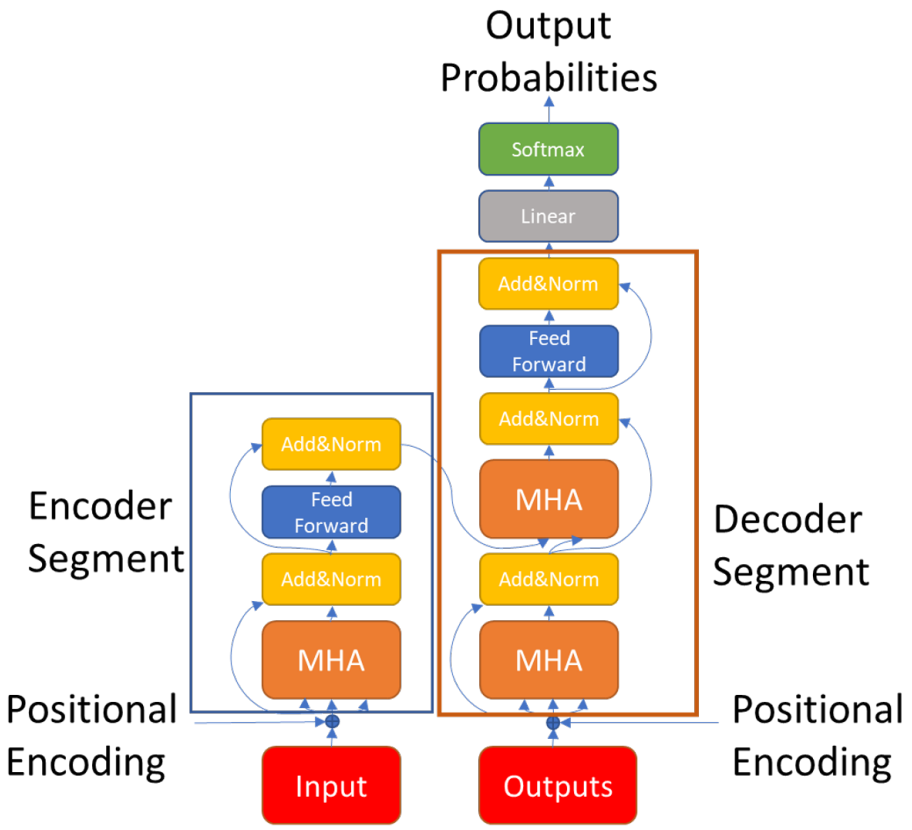


FIGURE 2.7: Multi-head attention module

Transformer is a deep learning architecture initially used in natural language processing tasks, such as machine translation, text generation, and text classification. It was introduced in 2017 by Vaswani et al. in the paper "Attention is All You Need" [33]. The key component of the Transformer is the Multi-Head Attention mechanism, which is used to model relationships between different elements in a sequence of data, such as words in a sentence. As illustrated in Figure 2.7

Mathematically, the attention mechanism can be represented as:

$$\begin{aligned} MultiHead(Q,K,V) &= Concat(head_1,...,head_h)W_0 \\ head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.9}$$

where Q is the query matrix, K is the key matrix and V is the value matrix.

2.1.3 Optimization

2.1.3.1 Loss Functions

For supervised tasks, the target of training DNN is to minimize the error between the real result and predicted result, which is called $loss function$. There are several loss functions that are commonly used in deep learning, depending on the type of problem being solved. Some of the most commonly used loss functions are:

Mean Squared Error (MSE): This is a regression loss function that measures the mean squared difference between the predicted and actual values.

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{D}(y_i - \tilde{y}_i)^2 \tag{2.10}$$

where N is the number of samples, y_i is the true value, and $\tilde{y}_i$ is the predicted value.

Cross-Entropy (CE): This is a classification loss function that measures the difference between predicted probabilities and actual labels. Binary Cross-Entropy(BCE) is commonly used for binary classification problems:

$$\mathcal{L} = \frac{-1}{N} * \sum \left(y \log(\tilde{y}_i) + (1 - y_i) \log(1 - \tilde{y}_i)\right) \tag{2.11}$$

And Categorical Cross-Entropy(CCE) is commonly used for multi-class classification problems:

$$\mathcal{L} = \frac{-1}{N} * \sum\sum y_{i,j} \log(\tilde{y}_{i,j}) \tag{2.12}$$

Hinge Loss: This is a binary classification loss function that is used for maximum-margin classification:

$$\mathcal{L} = \frac{1}{N} * \sum max(0, 1 - y_i * \tilde{y}_i) \tag{2.13}$$

Kullback-Leibler Divergence (KL divergence): This is a loss function that measures the difference between two probability distributions. KL divergence is commonly used for problems such as generative adversarial networks (GANs) and variational autoencoders (VAEs):

$$\mathcal{L} = D_K L(p||q) = \sum p_i * log(p_i/q_i) \tag{2.14}$$

where p and q are the true and predicted probability distributions.

These are just some of the most commonly used loss functions, and there are many other loss functions that have been developed for specific types of problems. The choice of loss

function will depend on the type of problem being solved. For more information about different loss functions, please refer to [34].

2.1.3.2 Algorithms

Optimization algorithms are widely used during training to adjust the weights in the network so as to minimize a loss function. There are several optimization algorithms that are commonly used to train DNNs, including:

Stochastic Gradient Descent (SGD): SGD is a optimization algorithm that updates the weights using the gradient of the loss with respect to the weights. The gradient is estimated using a single randomly selected sample from the training data, which is why the algorithm is called "stochastic". The weights are updated using the following rule:

$$w^* = w - \eta * \bigtriangledown \tag{2.15}$$

where w is the weight, η is the learning rate, and $\bigtriangledown$ is the gradient of the loss with respect to the weight.

Mini-batch Gradient Descent: Mini-batch gradient descent is a variant of SGD that updates the weights using a small batch of samples from the training data, rather than a single sample. The gradient is estimated using the average of the gradients for each sample in the batch.

Momentum: Momentum is an optimization algorithm that uses the past gradient to give more importance to the recent updates. This can help the optimization converge faster and avoid getting stuck in local minima. The weights are updated using the following rule:

$$\begin{aligned} v^* &= momentum * v - \eta * \bigtriangledown \\ w* &= w + v \end{aligned} \tag{2.16}$$

where v is the momentum and w is the weight.

Adagrad: Adagrad is an optimization algorithm that adapts the learning rate for each weight based on its historical gradient. The idea is to give more importance to the weights that have had larger gradients in the past, and reduce the learning rate for the weights that have had smaller gradients. The weights are updated using the following rule:

$$\begin{aligned} cache* &= cache + \bigtriangledown^2 \\ w* &= w - \eta * \bigtriangledown / (\sqrt{cache} + \epsilon) \end{aligned} \tag{2.17}$$

where *cache* is the accumulated gradient square and ϵ is a small constant added to prevent division by zero.

Adam: Adam is an optimization algorithm that combines the ideas of Momentum and Adagrad to give an adaptive and efficient optimization algorithm. The algorithm keeps track of both the average of the past gradients and their exponential moving average, and uses these to adapt the learning rate for each weight.

These are just a few of the most popular optimization algorithms used in training DNNs. The choice of optimization algorithm can have a significant impact on the performance of the network, and the best algorithm for a particular problem can vary depending on the specific details of the network and the data. For more details of different optimization algorithms, please refer to [35].

2.2 High Performance Computing

In the 1960s, Seymour Roger Cray built the CDC6600 at the Control Data Corporation(CDC), which outperformed all other contemporary computers and was considered as the first supercomputer in the world. Since then, supercomputers have been developing with a very high speed, the HPC systems are becoming larger(hundreds of thousands of compute nodes), more heterogeneous(CPU, GPU, FPGA, *etc.*), and thus more complex. Till now, HPC systems and their applications are playing a vital role in both scientific research and industry.

In this section, an overview of such HPC systems and the environments is given, so that the required background knowledge for this thesis is available.

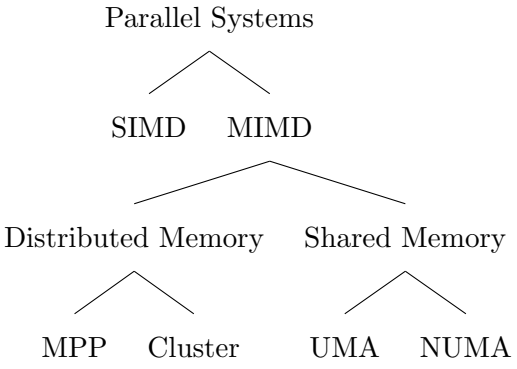
2.2.1 HPC System Design Architectures

During the past two decades, many different conceptual architectures of supercomputers have constantly evolved. HPC computer architectures are parallel computer architectures. A parallel computer is built out of compute units, main memory, and a high speed network.

For compute unit, CPUs are traditionally adopted in HPC platforms. Recently, GPUs (Graphical Processing Units) or GPGPUs (General Purpose Graphical Processing Units), which were originally used for image processing and displaying images, have been widely utilized on HPC systems to provide very high peak floating-point operations per second (FLOPs), especially for AI tasks. In addition, special devices like FPGAs (Field-Programmable Gate Arrays) which were traditionally used for fixed-point digital signal

processing are also used in HPC systems. Although, FPGAs operate with a lower clock frequency and have lower peak performances, it can be hardware optimized for each specific application thus can achieve better performance efficiencies. Another point is that FPGAs achieve in general higher power efficiencies compared to GPUs and CPUs.

Based on the main memory architecture which refers to an abstract level where the high speed network connects compute units and main memory, several parallel computer architectures have been developed.



- Single instruction, multiple data (SIMD): It is a single processor system which was developed at the early stage of supercomputing, also known as array processors or vector processors. SIMD based HPC systems are mainly used for the fast execution of similar computing operations on several simultaneously arriving or available input data streams. As its name indicates, one instruction is applied to a bunch of information or distinct data at given moment. It can be easily drawn out that SIMD is less efficient.

- Multiple Instruction Multiple Data (MIMD): Unlike SIMD, MIMD design applies multiple directions over totally different information at the same time, thus is more efficient in terms of performance than SIMD.

- Symmetric multiprocessor (SMP): It is a shared memory system that all compute units can directly access the whole main memory. This implies that memory access is symmetric and the time it takes to access any memory address is the same for all compute units. Therefore, the relative gain of performance keeps decreasing when the number of CPUs increase due to the memory-bus bottleneck.

- Non-Uniform Memory Architecture (NUMA): To avoid the problems of SMPs, NUMA was developed. It is built out of nodes that have their own memory while enable to directly access memory of other nodes.

- Massively Parallel Processings (MPP): In a distributed memory system each compute node can only access its own memory directly, while different nodes can

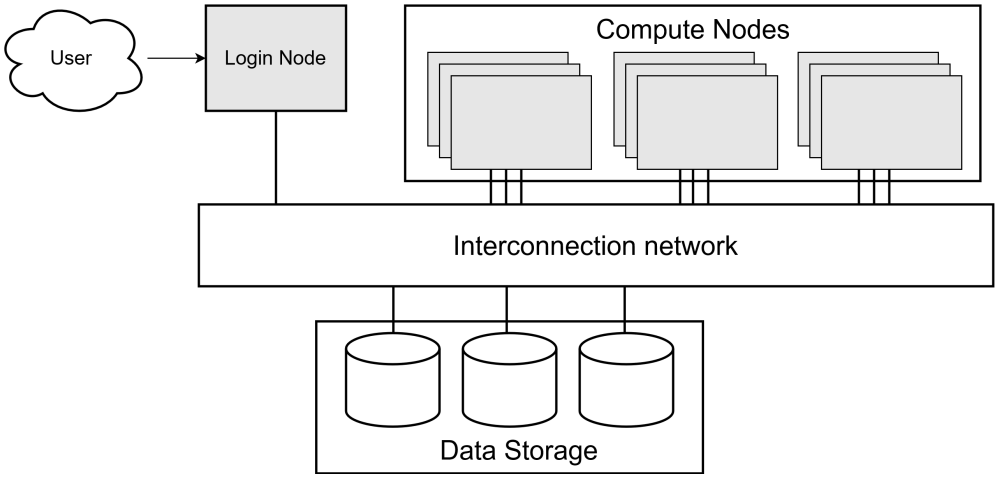


FIGURE 2.8: HPC cluster architecture

exchange data via the network. Compared with cluster systems, MPP systems are more tightly-integrated since individual nodes cannot run on their own and they are connected by a specialized network.

- Cluster: A cluster system is more standardized compared to MPP. It is a parallel computer system comprising an integrated collection of independent nodes, each of which is a system in its own right, capable of independent operation and derived from products developed and marketed for other stand-alone purposes [36]. A typical cluster HPC system is illustrated in Figure 2.8, where the details of each component are described in Section 2.2.2 to Section 2.2.4.

- Constellation: It is a specialized system which is usually based on a blade center structure: there are several nodes in one chassis and the whole chassis is considered as a single unit.

Figure 2.9 reveals the trend of the design concepts development for the Top500 HPC systems in the past 30 years, from 1993 to June, 2023. As can be seen, SMP, SIMD, and single processor systems all disappeared from the list in the 1990s. Constellations systems were quite popular in the 2000s, but have not been adopted by the Top500 systems any more since 2007. Nowadays, it is clearly visible that only MPP and cluster systems are in the TOP 500, while the percentages of MPP continues declining and most of the Top500 systems are clusters.

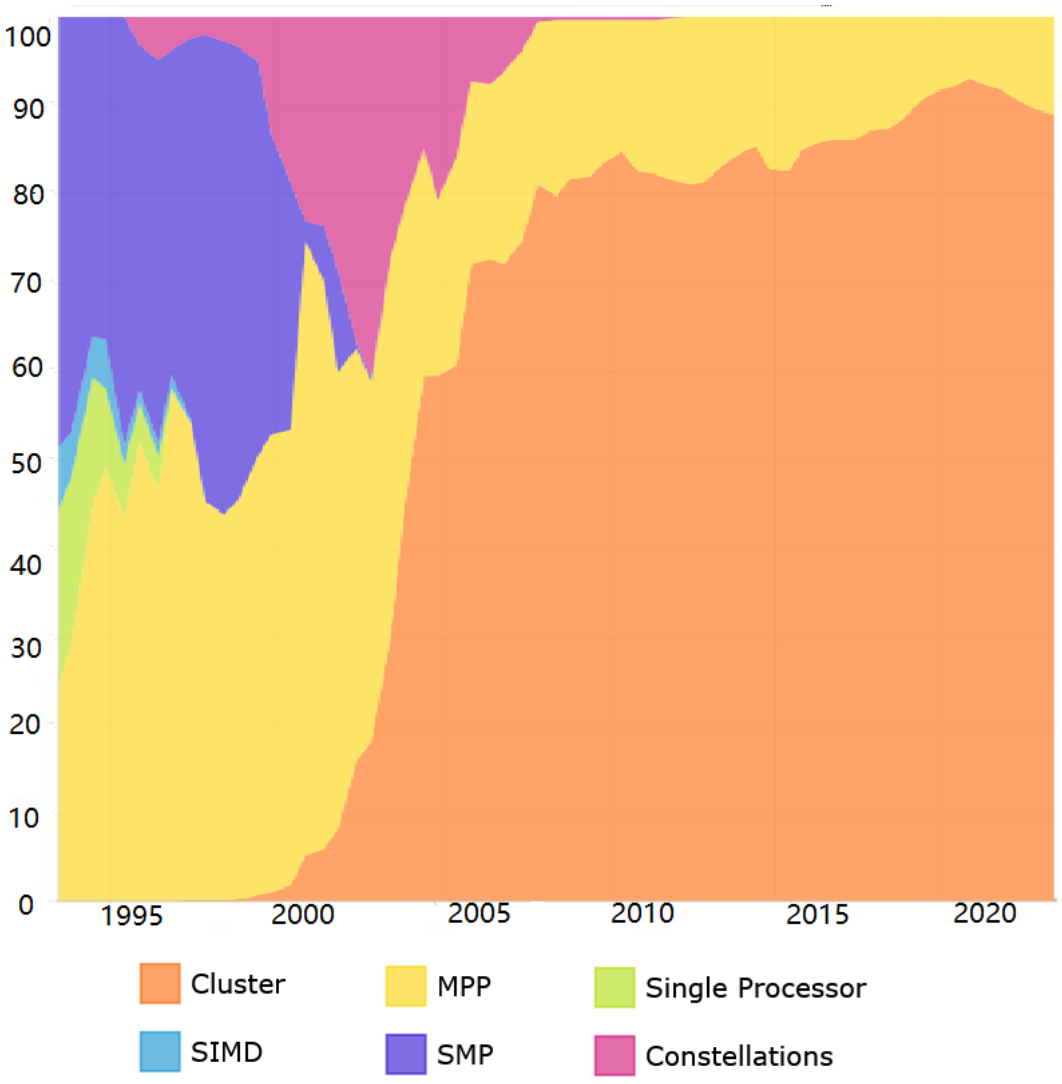


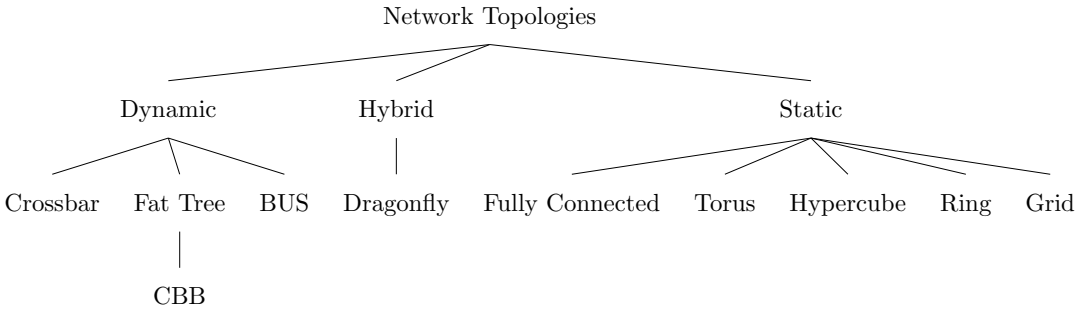
FIGURE 2.9: The development of design architectures of HPC systems within the TOP 500 List from 1993 to 2023. Image source: Top500 [1]

2.2.2 Interconnection Network

High-performance interconnection network is a critical part of the HPC system. With continuous development of HPC systems, the scale of computing cores and nodes continues to increase and the network scale of the interconnection network of HPC systems also continue to expand. For HPC systems with an enormous number of cooperating computing cores and nodes, the performance of large-scale interconnection networks inevitably become the bottleneck to limit the overall performance of HPC systems, because the execution time depends more and more on the communication time rather than on the calculation time. Therefore, the interconnection network has become the key to realizing high-speed, collaborative, parallel computing in a HPC system.

To evaluate the usefulness and performance of the interconnection network, a huge set of metrics can be adopted. Among them, the main performance properties can be network bandwidth, latency, switch radix, reliability, and power consumption. Bandwidth is the maximum amount of data transmitted over an internet connection in a given amount of time. Latency measures the amount of time required for a packet to travel from a source node to a destination node. Switch radix is the number of switch ports through which it connects to other nodes. Reliability describes the security in case of failure of individual components. It is obvious that additional resources are needed for redundancy to increase the reliability of a network. With the increased number of nodes in the petascale era, the reliability has become more important, and all interconnection network are required to tolerate the faults of individual nodes without any significant performance degradation [37]. And the power consumption measures how much power the connection network consumes since the power consumed on the inter-chip connection has become a significant portion of the total power usage of an exascale computing system. An ideal interconnect should maximizes the FLOPs spent doing useful application work, and minimizes the number spent waiting for data or performing processor communication.

The performance of a high-performance interconnection network is dependent on various parameters, among which the most significant are the network topology, the routing, and the flow-control algorithms [37]. While the routing and flow-control algorithms have advanced to a state where efficient techniques are known and used, it is still not reached at an equal level of sophistication in the development of topologies [38]. Therefore, only the network topologies will be discussed here and the taxonomy of network topologies [39] is listed as below:



- Ring network topology is a static topology, where each node is only connected to its neighbours, thus forming a closed ring, as depicted in Figure 2.10 (A). Messages from one node to another then travel from origin to destination via the set of intermediate nodes. The intermediate nodes serve as active repeaters for messages intended for other nodes, thus no switching components are needed.
- Fully connected network topology is a static topology where each node is connected to each other, as depicted in Figure 2.10 (B). Messages can be transferred from

the origin to the destination through the direct connection. When a connection is blocked and fails, the message can be redirected and the network is still operational. This mechanism ensures very high reliability with the cost that a node has $n-1$ networking cards, where n is the number of the nodes in the network. So this topology is only practical for a very small number of nodes.

- Fat Tree network topology is a universal dynamic network for very efficient communication, as illustrated in Figure 2.10 (F). It is one of the most widely used topologies, since it provides low latency and enables a variety of throughput options. However, for large scale clusters, it is relatively costly due to the large number of switches and links.

- Torus network topology is a switch-less static network topology, which directly interconnects a node to several of its neighbors in a k-dimensional lattice, as illustrated in Figure 2.10 (E). It can provide low network throughput for adversary traffic patterns. A torus is suitable for stencil applications, but due to its blocking nature and higher latency, it is not a preferred option for supercomputers that need to support a variety of applications.

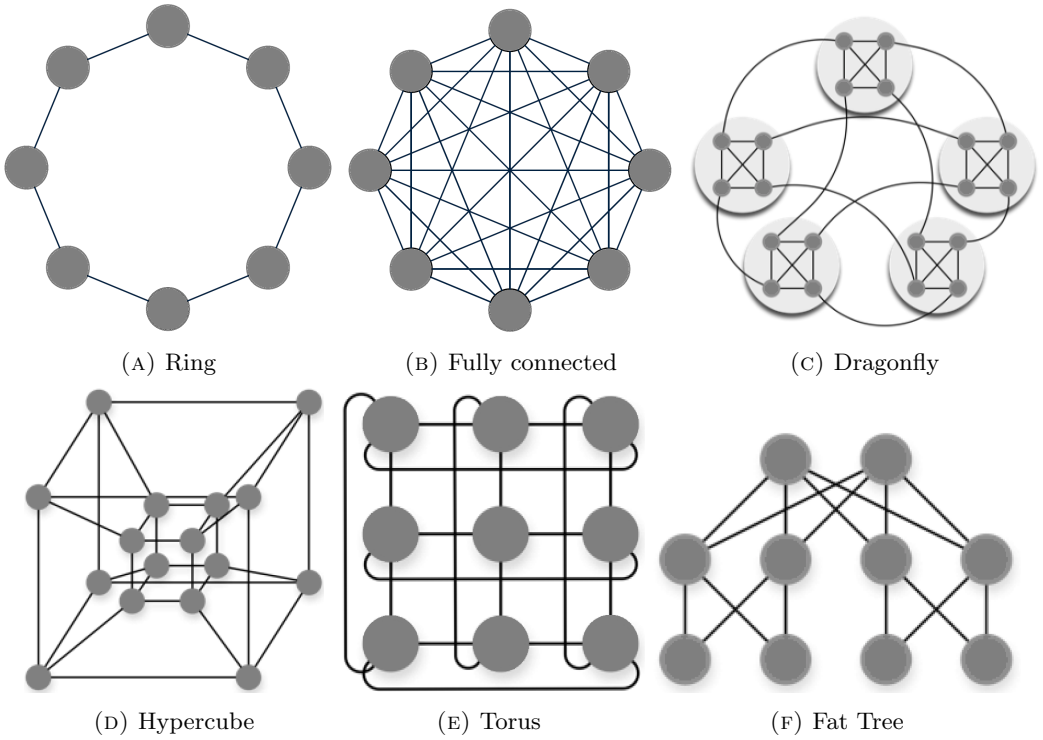
- Hypercube network consists of 2^n nodes, which form the vertices of squares to create an inter-network connection, see Figure 2.10 (D). A generalization of the k-ary n-cube [40] is the generalized hypercube where rather than being connected to the immediate neighbors in each dimension, each node is connected to every node in each dimension.

- Dragonfly is based on groups of connected compute elements, where all the groups are connected in a full graph, see Figure 2.10 (C). One can create any inner-group structure, such as a full graph (Dragonfly), a generalized hypercube (GHC), or a Fat-Tree. Dragonfly provides good performance for a variety of applications, specifically, it reduces network costs compared to other topologies, by reducing the number of long links.

Nowadays, the most frequently used interconnection network topologies in the Top500 include direct k-ary n-cubes, fat tree, torus and mesh, and dragonfly.

2.2.3 I/O and File Systems

For HPC systems, I/O usually refers to the storage and retrieval of persistent data to and from the file system. There are three main file system architectures used in clustered HPC systems: local file system, distributed file system, and parallel file system.



(A) Ring (B) Fully connected (C) Dragonfly

(D) Hypercube (E) Torus (F) Fat Tree

FIGURE 2.10: Different kinds of network topologies

Local file systems can be provided on nodes that are equipped with disks. Only the node itself can access its disks. Local file systems are useful for (heavy) scratch I/O. The advantage of local disks is that I/O performance scales perfectly with the number of nodes. The disadvantage is that for an overall view data has to be collected from all the nodes it is stored on. Because disks are a major source of failures, many clusters have diskless nodes.

Distributed (or network) file systems are provided by a file server which is integrated into the cluster. These file systems are available on all nodes. A typical example is the /home file system. While all nodes can read concurrently from a distributed file system without interference, care must be taken in concurrent writing. In general, a file should only be written or modified by a single process at a time. A classic distributed file system is the Network File System (NFS), which is not designed for high I/O loads.

For most HPC applications, it is difficult to run a program on several distributed computers with their own local storage, if the processes have to work on shared data. Furthermore, a lot of HPC applications require a very high I/O load from the file system. Hence, clustered file systems with parallel I/O are designed, which are supposed to abstract from this problem of coordinating accesses and consistency problems. The file system is usually a parallel file system(PFS), which is designed for parallelism, scalability, and high-performance. PFS enables concurrent access to individual data chunks from tens of thousands clients through efficient N-to-1 and N-to-N access pattern with

high-performance interconnect and protocol stack. Although the files and storage elements are distributed, parallel file systems abstract from this distribution and provide a Portable Operating System Interface (POSIX) [41]. Logically, a PFS is composed of there main components:

- PFS Client: it is provided the access to the file system and often available as a kernel module. Further, it is allowed to mount the global file system through the client software.
- metadata services: it manages all metadata operations on the file system, including providing client the access to the file system namespace using metadata such as directory hierarchy, file names, and access permissions.
- I/O server: it provides the I/O services connecting to the Object Storage Targets(OST), where OST can be almost anything from local disks to shared storage to high-end SAN fabric.

Some of the most commonly used PFSs by HPC centers are listed below:

- Lustre [42] is an open source parallel file system that provides high throughput and low latency storage for tightly coupled HPC workloads. In addition to the standard POSIX mount points in Linux, Lustre also supports data and I/O libraries such as NetCDF, HDF5, and MPI-IO, enabling parallel I/O to various application domains.
- BeeGFS [43] is a hardware-independent POSIX parallel file system developed with a strong focus on performance and designed specifically to manage I/O intensive workloads in performance-critical environments. BeeGFS is a software-defined storage based on the POSIX file system interface, which means applications do not have to be rewritten or modified to access the data inside the file system, communicate with the storage servers via network, via any TCP/IP based connection or via RDMA-capable networks.
- GPFS [44] is high-performance clustered file system software developed by IBM. It provides concurrent high-speed file access to applications executing on multiple nodes of clusters. GPFS provides a global namespace, shared file system access among GPFS clusters, simultaneous file access from multiple nodes, high recoverability and data availability through replication, the ability to make changes while a file system is mounted, and simplified administration in large environments.

The above mentioned PFSs can provide HPC applications general access directly via the POSIX file system API, however, for some applications, it is far from enough. Therefore, some higher-level I/O libraries are specially designed for scientific I/O. For example, MPI-IO [45] supports collective I/O operations, where groups of processes use the API to concurrently execute a read or write operation. As an optimization, MPI-IO servers can perform global write buffering and aggregation to match the I/O pattern of clients to the layout of data on the data servers. Libraries such as HDF5 [46] and ADIOS [47] provide higher-level storage management capabilities. For instance, HDF5 provides its own directory structure, with files being replaced with typed, multidimensional numerical arrays called dataset. I/O libraries may be layered, *e.g.,* HDF5 can be layered on top of MPI-IO to enable collective access to datasets; and, in turn, MPI-IO can be layered on top of POSIX. This layering generates complex I/O access patterns that may differ greatly from the I/O access patterns one would deduce from examining the scientific application code.

2.2.4 Scheduling and Resource Management

A scheduler is software that implements a batch system on a HPC cluster, which is responsible for receiving and parsing the user submitted jobs. HPC users usually cannot not run their jobs directly and interactively, instead they submit non-interactive batch jobs to the scheduler. The scheduler stores the batch jobs, evaluate their resource requirements and priorities, and distributes the jobs to suitable compute nodes. Without a scheduler, an HPC cluster would just be a bunch of servers with different jobs interfering with each other. When you have a large cluster and multiple users, each user is not aware of which compute nodes and CPU cores to use, nor how much resources are available on each node. To solve this, cluster batch control systems are used to manage jobs on the system using HPC schedulers. They are essential for sequentially queueing jobs, assigning priorities, distributing, parallelizing, suspending, killing, or otherwise controlling jobs cluster-wide.

The resource manager is the part of a batch system, which is typically used to control resource usage policies, such as the number of cores or requested memory. It allows users and administrators to access and manage the status of various computing resources like processors, memory, network, and storage. The resource manager provides the infrastructure to control and monitor the job and collect the statistics of all the processes running all of the tasks in the job. These statistics are gathered, aggregated, and ultimately saved to a database which will contain a record of that job's run. Moreover, a mechanism called job prologue and epilogue is designed to enable the resource manager

to prepare the nodes for the next job, so that a resource manager is able to to clean, prepare or test the full functionality of the allocated nodes.

There exist various software which performs scheduling and resource management for HPC clusters, some of them are listed as below:

- The Simple Linux Utility for Resource Management (SLURM) [48] is an open-source, fault-tolerant, and highly scalable cluster management and job scheduling system for clusters of all sizes. Slurm requires no kernel modifications for its operation and is relatively self-contained. As a cluster workload manager, it provides a rich set of features for managing resources, scheduling jobs, and tracking job status. It also supports plugins for customizing its behavior and integrating with other tools.

- Moab HPC Suite [49] is a workload and resource orchestration platform that automates the scheduling, managing, monitoring and reporting of HPC workloads on massive scale. It includes tools for managing resources, scheduling jobs, and monitoring performance. It also provides features for managing policies and quotas, enforcing service-level agreements, and optimizing resource utilization.

- TORQUE (Tera-scale Open-source Resource and QUEue manager) [50] is a resource manager providing control over batch jobs and distributed compute nodes. It is a community effort based on the original PBS project and has incorporated significant advancements in the areas of scalability, fault tolerance, and feature extensions. It can be integrated with other HPC tools, such as Moab.

- The Portable Batch System(PBS) [51], is another workload management solution for HPC systems and Linux clusters. From the initial design forward, PBS has included innovative new approaches to resource management and job scheduling, such as the extraction of scheduling policy into a single separable, completely customizable module. PBS provides a simple interface for submitting jobs, managing resources, and monitoring job status. It also supports advanced features such as job arrays, checkpoint/restart, and resource reservations

2.2.5 HPC Systems at HLRS

The Hawk system at HLRS [52] is composed of 5,632 CPU nodes and 24 GPU nodes (Apollo 6500 nodes with 8 NVIDIA A100 GPUs per node), deploys an Infiniband HDR based interconnect with a 9-dimensional enhanced hypercube topology. Infiniband HDR has a bandwidth of 200 Gbit/s and a MPI latency of 1.3us per link. Hawk has a peak

TABLE 2.1: Technical Specification of Hawk and Vulcan

System	HPE Apollo (Hawk)	Apollo 6500	Cray CS-Storm
Number of compute nodes	5632	24	8
Peak performance	26 Pflops	120 Pflops (AI)	8 Pflops (AI)
CPU type	AMD EPYC 7742	AMD EPYC 7702	Intel CLX 6240
GPU type	-	Nvidia A100	Nvidia Tesla V100
Number of cores	720,896 (CPU)	1,327,104 (CUDA)	327,680 (CUDA)
CPU frequency	2.25 GHz	2.0 GHz	2.6 GHz
Interconnect	InfiniBand HDR200	InfiniBand HDR200	InfiniBand HDR100

performance of 26 Petaflops and its GPU accelerator extension has a peak performance of 120 Petaflops for DL training.

In addition, Cray CS-Storm [53] (part of Vulcan cluster) is also provided to users. It is composed of 8 NVIDIA Tesla V100 GPU nodes for DL workloads and 8 Intel Xeon Gold 6230 CPU nodes (CS-500) for big data workloads. To address the demand for processing-intensive applications in the realms of ML and DL, the Vulcan and Hawk-AI partitions support a wide variety of well-known and established AI frameworks and tools, such as Apache Spark, Python-based data science libraries like scikit-learn, and frameworks steered toward DL like TensorFlow and PyTorch. The detailed configuration of Hawk and Vulcan are listed in Table 2.4.

The storage of HPC systems is available globally through distributed parallel file systems like Lustre or Network File System (NFS). The Lustre at HLRS, which provides about 25 PB of storage to its users, is accelerated with DDN IME to achieve highest performance especially when dealing with large amount of small files. And the ability of providing high performance solution for small files is utmost important for DL applications.

2.3 Quantum Computing

As described in the section 2.2, with the great power of storage and computation, supercomputers have been a significant tool to address the most challenging problems faced by scientists, business leaders and governments. However, even with the most advanced supercomputers, many problems remain intractable because of the great requirement of storage and computation power. There are many approaches adopted to innovate the infrastructure and to meet the ever increasing requirements: one popular method is the memory-driven computing [54], where each processor in a system is

given access to an enormous reservoir of shared memory; another approach is to offer greater flexibility, control and customization at the hardware level through chiplets and field-programmable gate arrays (FPGAs) [55], which are used to accelerate HPC computation. However, these methods are still within the realm of classical computers and limited by the intrinsic boundaries of classical computers. The space for storage and the time for computation are restricted by the architecture and physical implementation of the classical computers, which is incapable of solving many problems. For instance, the description of more sizeable quantum systems is still impossible on today's classical supercomputers since the dimension of quantum systems scales exponentially with the number of basic building blocks.

A completely new approach is an entirely different paradigm: quantum computing. Compared to the modest performance gains expected from the above approaches, quantum computing offers the possibility of revolutionary breakthroughs such as transforming computationally unsolvable problems into manageable ones. For example, problems that would take too long to compute on a classical computer due to exponential or subexponential time complexity can be computed on a quantum computer in a reasonable amount of time (polynomial time) [56].

The history of quantum computing dates back to the early 20th century when physicists such as Max Planck and Albert Einstein introduced the concept of quantum mechanics. In the 1930s, Paul Dirac developed the mathematical foundations of quantum mechanics, which enabled the development of quantum theory [57]. In 1982, physicist Richard Feynman proposed the idea of using quantum computers to simulate quantum systems, which are difficult to simulate using classical computers [58]. Since then, the field of quantum computing evolved in the last decades and is now one of the most rapidly developing research areas. In 1985, physicist David Deutsch developed the first quantum algorithm, known as the Deutsch-Jozsa algorithm [59]. In 1994, mathematician Peter Shor developed a quantum algorithm for factoring large numbers, which would have significant implications for cryptography [60]. In 1996, Grover developed the Grover's algorithm which can search an unsorted database by using quantum parallelism and interference to amplify the amplitude of the desired state in the quantum state [61].

Till now, quantum computing is still in its infancy and the key developments remain on the more theoretical side. Even the fastest quantum computers today have no more than 1000 qubits, and are plagued by random errors. These first quantum computers are called noisy intermediate-scale quantum (NISQ) devices, comprise up to a few hundred quantum bits and give the opportunities to test quantum algorithms for their behaviour

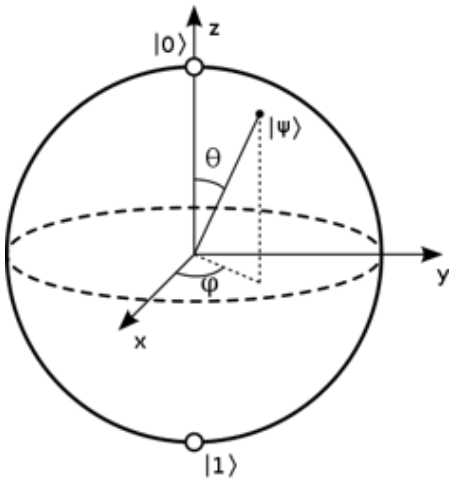


FIGURE 2.11: Bloch sphere: qubit states can be represented as points on the surface of the sphere

under high noise levels. In 2019, Google demonstrated that its 54-qubit quantum computer could solve in minutes a problem that would take a classical machine 10,000 years, roughly 158 million times faster [62].

To make the research presented this work accessible to people new to quantum information, the basic definitions will be discussed in this section, including quantum bit and superposition, quantum computation and quantum circuits, quantum algorithms, *etc.* For a more detailed introduction to quantum information and quantum computing, please refer to [63].

2.3.1 Qubits and Superposition

In classical computation and classical information, the *bit* is the fundamental concept which can only have one of the two values, either 0 or 1. This is because of the limitation of the implemented device, since classical hardware devices have only two states. Different from classical computation and classical information, quantum computation and information is built upon an analogous concept, namely *quantum bit*, or *qubit* for short. Similar to classical bit which has a state(either 0 or 1), a qubit also has a state, which is often denoted in terms of $|0\rangle$ and $|1\rangle$. The difference between a bit and a qubit is that a qubit can be in a coherent superposition of both levels $|0\rangle$ and $|1\rangle$ simultaneously, while a classical bit can either be 0 or 1. For more mathematical description of *Dirac Notation*, please refer to [64].

A qubit can be described as a normalised vector $|\psi\rangle$ in a two-dimensional complex Hilbert space $\mathcal{H} = \mathbb{C}^2$, which can be expressed as superposition:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.18}$$

where α and β are complex numbers and $|\alpha|^2+|\beta|^2 = 1$. Since this description is a vector with norm 1, it is easy to depict a qubit state on the surface of a sphere. Therefore, a qubit state can be rewritten as:

$$|\psi\rangle = e^{i\gamma}(\cos(\frac{\theta}{2})\,|0\rangle) + e^{i\varphi}\sin(\frac{\theta}{2})\,|1\rangle) \tag{2.19}$$

As is visualized in Figure 2.11, which is named as *Bloch Sphere*.

For multiple qubits, n qubit state can be expressed as the superposition

$$|\psi\rangle = \sum_{q_1,\ldots,q_n \in 0,1} \alpha_{q_1,\ldots,q_n}\,|q_1 \ldots q_n\rangle \tag{2.20}$$

in $\mathcal{H} = \mathbb{C}^{2n}$ with complex coefficients $\alpha_{q_1,\ldots,q_n}$.

2.3.2 Quantum Computation and Circuits

Quantum gates are basic building blocks of quantum algorithms and quantum circuits. They manipulate the quantum states of qubits and are used to encode quantum algorithms and perform quantum computations. In Table 2.2, some of the commonly used quantum gates are listed. For example, applying the Pauli-X gate on a quantum state of ψ gives:

$$X\,|\psi\rangle = X(\alpha\,|0\rangle + \beta\,|1\rangle) = \beta\,|0\rangle + \alpha\,|1\rangle = |\psi'\rangle \tag{2.21}$$

Since $X\,|\psi\rangle = |\psi'\rangle$, and for every gate U that fulfills $U\,|\psi\rangle = |\psi'\rangle$, specifically $\langle\psi|\psi\rangle = 1 = \psi'|\psi'$, the gate has to be unitary defined through $U^\dagger U = 1$, where $U^\dagger$ denotes the adjoint of the matrix U . For more information about unitary gates, please refer to [65].

Except from the quantum gates listed above, there are also some commonly used quantum gates which can be adjusted by the parameters, namely parameterized quantum gates. In Table 2.3 a selection of parameterized quantum gates are listed and explained. Here is a brief introduction to them:

- P gate: The P gate, also known as the "phase gate," adds a phase to the state of a qubit. It is represented by the unitary matrix:

$$P(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix} \tag{2.22}$$

  where i is the imaginary unit and θ is the phase angle.

Gates	Operators	Matrix
—[I]—	Identity	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
—[X]—	Pauli-X	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
—[Y]—	Pauli-Y	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
—[Z]—	Pauli-Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
—[S]—	S	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
—[H]—	Hadamard	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
	CNOT	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Table 2.2: A selection of quantum gates

Gates	Operators	Matrix
—[$P[\theta]$]—	$P(\theta)$	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$
—[$RX[\theta]$]—	$R_X(\theta)$	$\begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$
—[$RY[\theta]$]—	$R_Y(\theta)$	$\begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$
—[$RZ[\theta]$]—	$R_Z(\theta)$	$\begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$

Table 2.3: Parameterized Gates

- RX gate: The RX gate, also known as the "rotation-X gate," rotates the state of a qubit around the X-axis in the Bloch sphere representation. It is represented by the unitary matrix:

$$RX(\theta) = \cos(\theta/2)I - i\sin(\theta/2)X \tag{2.23}$$

  where I is the identity matrix and X is the Pauli-X matrix.

- RY gate: The RY gate, also known as the "rotation-Y gate," rotates the state of a qubit around the Y-axis in the Bloch sphere representation. It is represented by the unitary matrix:

$$RY(\theta) = \cos(\theta/2)I - i\sin(\theta/2)Y \tag{2.24}$$

  where Y is the Pauli-Y matrix.

- RZ gate: The RZ gate, also known as the "rotation-Z gate," rotates the state of a qubit around the Z-axis in the Bloch sphere representation. It is represented by the unitary matrix:

$$RZ(\theta) = \cos(\theta/2)I - i\sin(\theta/2)Z \tag{2.25}$$

  where I is the identity matrix and Z is the Pauli-Z matrix.

The quantum gates stated above describe the basic operations on quantum state, then the questions arise: how can complex operations be done? Can a complex operation be approximated by breaking down into sequences of elementary gates? The Solovay-Kitaev theorem [66] states that every unitary U can be approximated with an accuracy of ε by a sequence of gates from $\mathcal{G}$ of length $(O)(\log^c(\frac{1}{\varepsilon}))$, where $\mathcal{G}$ is finite universal gate set and c is a constant based on the choice of U.

The quantum operations built from sets of gates can be represented in a quantum circuit. Figure 2.12 illustrates a sample quantum circuit composed of a Hardamad gate and two CNOT gates, where every horizontal line depicts a qubit and following the lines from left to right depicts the evolution in time.

2.3.3 Quantum Algorithms

Quantum algorithms are a class of algorithms that can be run on quantum computers, taking advantage of the unique properties of quantum mechanics to solve problems more efficiently than classical algorithms, or cannot be solved in any feasible time by today's classical computer. One advantage of quantum algorithms lie in the exponential memory capacity, since the state space grows with 2^b, where b is the number of computational bits on classical computers, while in quantum computers it grows in a polynomial manner.

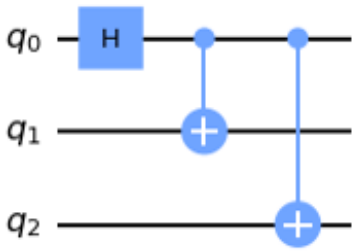


FIGURE 2.12: A sample quantum circuit with three qubits, one Hardamard gate and two CNOT gates

Another quantum advantage is the quantum parallelism, which can be achieved by quantum superposition and inference.

Here are some of the most well-known quantum algorithms:

Deutsch-Jozsa algorithm [59]: This is one of the earliest quantum algorithms and is designed to solve a specific type of problem known as the Deutsch-Jozsa problem. This problem can be formulated as follows: given a function f that takes a string of bits as input and returns either 0 or 1, determine whether the function is constant or balanced. The Deutsch-Jozsa algorithm works by exploiting the properties of quantum superposition and interference. It takes advantage of the fact that a quantum computer can evaluate a function on many inputs simultaneously by creating a superposition of all possible inputs. The algorithm then applies a series of quantum gates that depend on the function being evaluated to create interference between the different inputs in the superposition. Finally, it measures the output of the quantum circuit and uses it to determine whether the function is constant or balanced.

Shor's algorithm [60]: This is a quantum algorithm designed to solve the problem of factoring large numbers into their prime factors, which is believed to be computationally intractable for classical computers. The Shor algorithm works by exploiting the properties of quantum superposition and entanglement to evaluate the period of a function called the modular exponentiation function. This function is used in classical factoring algorithms and plays a key role in the Shor algorithm. Shor's algorithm can factor an $N - bit$ integer in $O((\log N)^3)$ time, which is much faster than the best known classical algorithms, which require $O(e^{(\log N)^{\frac{1}{3}} * (\log \log N)^{\frac{2}{3}}})$ time.

Grover's algorithm [61]: This is a quantum search algorithm that can be used to find a particular item in an unsorted database. The key to the algorithm's speed is that it uses a combination of quantum superposition and interference to search the database for the desired item. It does this by initializing a quantum register in a superposition of all

TABLE 2.4: Technical Specification of IBM Quantum System One at Ehningen

Index	Values
Number of qubits	27
Coherence time	$\approx 150us$
Single qubit gate error	$\approx 0.025\%$
Two qubit gate error	$\approx 0.7\%$
Operation time of 2 qubit gate	$\approx 300ns$ for CNOT
Quantum Volume	QV 64

possible states, applying a series of quantum gates to the register, and then measuring the register to obtain the desired item with high probability. The algorithm requires a number of iterations that scales with the square root of the number of items in the database. Each iteration amplifies the amplitude of the item being searched for while suppressing the amplitudes of the other items. This process continues until the desired item is found with high probability. Grover's algorithm can search through a database of N items With $O(\sqrt{N})$ queries, which is much faster than classical search algorithms that require $O(N)$ queries.

2.3.4 IBM Quantum System One at Ehningen

The experiments in this dissertation related to quantum computing are conducted on the IBM Quantum System One at Ehningen [67], which is the world's first integrated gate-based quantum computer system. The specification of IBM Quantum System One can be found at Table 2.4.

2.4 Conclusion

In conclusion, this chapter has provided a comprehensive overview of the context and key factors that are relevant to the topic being studied. Overall, this chapter has set the stage for the research by providing a solid understanding of the historical, theoretical, and empirical context in which the study takes place. This understanding will be vital for interpreting the findings and drawing meaningful conclusions from the study.

Chapter 3

Methods for Scaling Deep Learning Workloads on HPC

This chapter reviews the techniques that bridge the gap between HPC and deep learning workloads, *e.g.,* environment setup, orchestration/scheduling, frameworks, *etc.* Furthermore, various optimization methods are described and compared from different aspects, which enable the seamless execution of deep learning applications on HPC, including parallelization methods, communication optimization, *etc.*

Chapter outline: This chapter is divided into sixsections. In Section 3.1, the introduction to the research scenario is provided. Section 3.2 describes the key technologies that are commonly used for setting up the environment in the HPC system for DL workloads. In Section 3.3, different parallelization methods that enable distributed training of DL workloads on HPC are described, followed by the review of communication optimization methods in Section 3.4. In Section 3.5, different tools and frameworks which are used for distributed deep learning training are summarized and compared. Lastly, the conclusion of this chapter is given in Section 3.6.

3.1 Introduction

HPC systems are traditionally employed to perform large-scale financial, engineering and scientific simulations. The efficient implementation of such simulations requires good knowledge of the underlying hardware architectures, network topologies, programming environments and libraries to exploit the systems. With the recent advent of AI, specifically High-Performance Data Analytics (HPDA) and DL, there is an increasing

TABLE 3.1: Comparison of HPC and DL stack

Type	HPC	Deep Learning
Scheduling	Batch Scheduler: PBS/Torque, Slurm	VM/Container Management
Frameworks	OpenMP, MPI, OpenFoam, ..	Tensorflow, Pytorch, ...
Programming Language	C/C++, Fortran	Python, C++, Java
Network	Infiniband,OPA fabrics	Ethernet
Storage	Storage&I/O nodes, NAS	Local Storage, NAS/SAN
Processors	CPUs, GPUs, FPGAs	CPUs, GPUs, TPUs, FPGAs

demand for HPC infrastructures [68]. Challenges such as climate simulation, green energy, smart transportation and autonomous driving require a close interplay between AI and simulations. For instance, climate simulation needs a fine mesh resolution and incorporates many complex atmospheric components to accurately predict cloud behaviour that can be learnt with given physical constraints. A trained AI model thus can act as a surrogate to replace the traditional physics-based model, which reduces model complexity and improves prediction accuracy; meanwhile, execution time on HPC systems is significantly reduced to a small fraction of the simulation with the physics-based model.

Conventionally, HPC focuses on complex and compute-intensive tasks that are highly parallelizable. It has grown its own legacy software stacks powered by programming languages such as C++/Fortran and parallel programming paradigms such as OpenMP [69], and Message Passing Interface (MPI) [70]. *Per contra*, AI applications are usually developed with high-level scripting languages or frameworks, *e.g.,* TensorFlow [71], and PyTorch [72], which require flexible development environments.

To scale DL model training and fully utilize the computation and storage resources on HPC is often challenging because there exist several gaps between the two workflows. For example, the schedulers of HPC are often batch schedulers, *e.g.,* PBS, Slurm, *etc.* which evaluates a job's resource requirements and priorities and distributes the job to suitable compute nodes. While in DL training, such work is often done by VM/container management, and DL schedulers often refer to how to map the DL training processes to the processing nodes in the infrastructure. A detailed description of the difference between HPC and DL is shown in Table. 3.1.

There have been several surveys that are complementary to this chapter which focus on more specific aspects such as containerization [73, 74], container orchestration [75–77], cloud orchestration[78], scaling up machine learning[79], scaling up DL [80], high-performance big data [81–83]. In particular, Liao et al. [74] and Bhatia et al. [73] provide

a review of Docker technologies, one of the most used containers. Casalicchio [75] surveys the state-of-the-art solutions and research challenges in the autonomic orchestration of containers. Bengio [80] surveys scaling deep learning from various perspectives, focusing on models, optimization algorithms and datasets. Furthermore, it also overviews some aspects of distributed computing, including asynchronous and sparse communication.

Contribution: There is no study yet provides a broad view of the interplay and convergence of AI and HPC. Thus, the convergence of AI and HPC, including the techniques used to deploy AI applications on HPC has been studied in this chapter. Moreover, the benefits of utilizing HPC for AI applications with respect to parallelism, communication and scheduling are summarized. These aspects are crucial when applying HPC for AI applications.

3.2 Environment Setup

Configuring and maintaining installed software on HPC clusters is often difficult because of the complex dependencies among libraries, particularly, updating and installing new packages can often cause dependency conflicts. When training of DL models on HPC clusters, such complexity becomes increasingly evident, especially considering the different programming languages, libraries, frameworks and workflows. Solutions to address the above issues should meet the following requirements:

- give users the freedom to customise working environments without affecting the host systems. This is often not an option on HPC systems where users often have limited access privileges, (*e.g.,* no root access).
- make it possible to move around the applications easily among HPC clusters.

This section depicts different technologies which can circumvent the above issues and facilitate implementation of DL processes on HPC systems.

3.2.1 Containerization

A container encapsulates programs together with their libraries, data and configuration files [84] in an isolated environment, ensuring package compatibility and enabling users to move and deploy programs easily among clusters. Containerization is a virtualization technology [85]. Rather than simulating the holistic kernel as in a Virtual Machine (VM), a container shares its host's underlying kernel. Different from the VM, the host merely

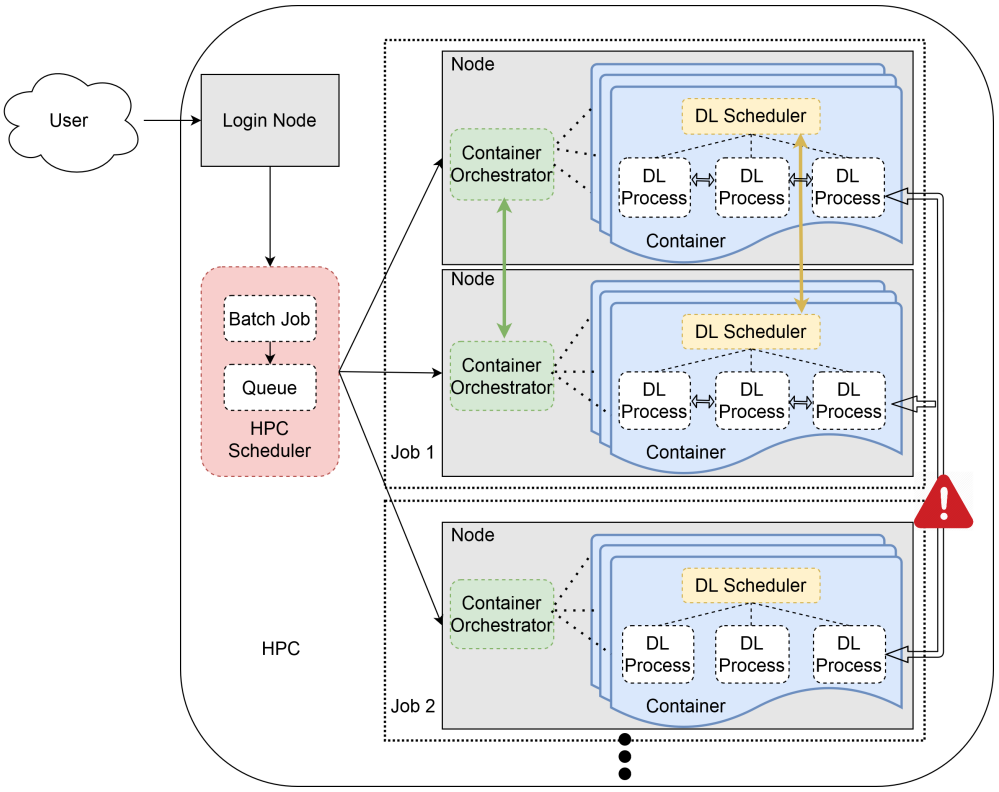


FIGURE 3.1: Scalable deep neural network training on HPC through container and container orchestrator

needs to start new processes that are isolated from its own to boot a new container [86], thus making start-up time of a container similar to that of a native application. Apart from their portability, containers can also guarantee reproducibility.

Figure 3.1 illustrates how scalable DL workloads are executed on HPC cluster. Users prepare and submit their workloads as jobs to the HPC scheduler; based on request, policies, and availability of resources, HPC scheduler reserves certain compute nodes and executes the job script submitted by user; job script prepares required environment and deploys a container orchestrator on (all) reserved nodes; last one deploys containers with DL processes managed by DL scheduler. HPC cluster are usually multi-tenant systems where some resources (especially network) are shared between tenants. Depending on security restrictions and available features container orchestrator can be represented by a simple script for deploying containers, or by a more complex orchestrator like Kubernetes or its more restricted implementations (k3s[87], usernetes[88]).

There are multiple technologies for implementing the concept of containers, *e.g.,* Docker [89], Singularity [90], Linux LXC [91], Shifter [92], *etc.* AI models can be easily implemented in containers like Singularity [93] with its dependencies prepared for running on HPC clusters. However, when running on HPC infrastructure, container's nature

of hardware- and platform-agnostic prevents the containerized application from detecting and leveraging any specialized hardware and software in the system, such as GPU accelerators, vendor-specific MPI implementations that unlock the full performance of interconnect technologies like InfiniBand and Cray's AriesTM. Therefore, containers which specifically target at deploying AI applications on HPC need to be researched.

NVIDIA offers NGC container registry [1] which provides fully integrated DL framework containers that are optimized for NVIDIA GPUs. Compared with other containers, NGC containers provides monthly DL container releases offering latest features and superior performance on NVIDIA GPUs [94]. The supported DL frameworks include TensorFlow, PyTorch, MXNet, NVIDIA TensorRT, *etc.* Besides, it supports Docker and Singularity runtimes and can run flexibly in different HPC environments. In addition, NGC offers HPC containers supporting visualization tools, *e.g.,* ParaView with NVIDIA IndeX volume renderer.

Benedicic et al. [95] presented an extension to the container runtime of Shifter that provides containerized applications with a mechanism to access GPU accelerators and MPI resources from the host system, effectively enabling performance portability of containers across HPC resources. Their experiment of containerzed TensorFlow on HPC systems proved that the Shifter extension allows accelerated and distributed DL applications to achieve native performance when executed from a container.

Charliecloud [96], developed at Los Alamos National Laboratory (LANL), is a lightweight open source user-defined software stacks (UDSS) implementation based on the Linux user namespace for HPC sites with strict security requirements. Brayford et al. [97] described a mechanism that employs Charliecloud to deploy TensorFlow and trained a complex neural network at scale on a secure HPC system with utilization of MPI and Horovod.

Regarding the impact of container technologies on AI workloads running on conventional HPC systems, different studies have proven that HPC users and AI experts should feel free to containerize their AI applications without concern about performance degradation, regardless of the container technology used [93, 98, 99]. It is an encouraging development on the path towards greater adoption of user-defined software stacks to increase the portability and reproducibility of HPC.

[1] https://ngc.nvidia.com

3.2.2 Container Orchestration

Containers are dedicated to run micro-services, with one container typically hosting just one application. Nevertheless, containerized applications can become complex, as thousands of separate containers can sometimes be required in production. The traditional HPC workload managers, *e.g.,* Slurm [100], Torque [101], lack efficiencies in container scheduling and management, and often do not provide integrated support for environment provisioning [102]. Moreover, HPC applications are hardware specific, and their applications are specifically optimized for the nodes. This is not the case for containerized applications. Regarding the deployment of AI applications on HPC, an AI application needs to parallelize its workloads and scale to more HPC compute nodes in order to speed up the whole process [97].

The state-of-art trend is to optimize the containerized AI frameworks to better suit HPC infrastructure, *e.g.,* containerize the DL framework Horovod [103] (see Section 3.5.2), as Horovod adopts the concepts of MPI that is a typical model for writing parallel applications for HPC. Considering that performance is the *sine qua non* for HPC applications, it poses the key question for massive usage of containerized applications on HPC clusters [104–106]

Kubernetes [107] is a widely-adopted container orchestrator on cloud clusters. Some tools are developed which enables scheduling of containers from cloud to HPC systems. WLM-operator [108] enables submitting and monitoring container jobs to HPC clusters managed by Slurm from cloud clusters controlled by Kubernetes, while takes advantages of Kubernetes features, such as smart scheduling and volumes. Zhou et al. [102] implemented a Torque-Operator that extended WLM-operator with Torque support and proposed an architecture which can provide a unique interface for job submission and management for HPC and cloud.

With the development of container and orchestration, it is noticed that more and more domains of science begin to use container technology to deploy AI applications on HPC systems, such as climate analytics [109], cancer detection [110], computational fluid dynamics [95], *etc.*

3.2.3 Other Methods

Conda [111]: Conda focuses on quick installation of software and ease of use, and enables users create a conda environment in which they can install one or more packages. These packages are usually pre-built generic binaries, which may significantly impact the performance of the installations. Despite wide adoption in the scientific community

conda is not a good fit for HPC systems for a number of reasons, including poor support for multi-user environments, a lack of focus on performance, heavily relying on the home directory (which usually is limited in size on HPC systems), *etc.* There is also no guarantee that it will install libraries that are compatible with the hardware of the cluster, so the Conda-installed software may not always work properly with the cluster interconnect or resource manager.

Virtualenv [112]: Virtualenv is a tool for creating isolated Python environments. With virtualenv, users can install specific versions of packages and dependencies without affecting the system Python installation. Virtualenv is a lightweight tool and can be useful in HPC environments where users have limited permissions to install software.

EasyBuild [113]: EasyBuild is a tool for automating the installation and management of software on HPC clusters. With EasyBuild, users can easily install and manage complex software stacks and dependencies. EasyBuild supports a wide range of software packages and HPC systems, making it a versatile tool for managing software in HPC environments.

Spack [114]: Spack is a free, open-source package manager designed to make it easy to install and manage software on a wide range of computing platforms. Its support for multiple architectures, compilers, and operating systems, as well as its ability to create and manage environments, make it an ideal tool for scientific computing, high-performance computing, and data science applications. One of the key features of Spack is its ability to create and manage software environments. Users can create isolated environments for different projects, each with its own set of installed packages and dependencies. This makes it easy to switch between different software stacks, and to ensure that different projects do not interfere with each other. Spack also supports the creation of custom packages, which can be shared with others or kept private. Packages can be created using a simple configuration file format, and can be versioned and published to online repositories for easy sharing.

3.3 Parallelization

In deployment of distributed deep learning, there exist many possibilities for parallelization, especially when used in conjunction with HPC. In order to fully utilize the computational resources in HPC, different parallelization algorithms and strategies have been developed. Here, the four predominant parallelization methods in ML & DL, namely data, model, pipeline and local parallelism, as well as hybrid forms of parallelism are introduced.

3.3.1 Model Parallelization

Model parallelization [115, 116] is a commonly used strategy where parameters of DNN models are partitioned to different workers and undergo concurrent iterative updates. Here, a subset of parameters on each worker is updated in parallel using either all data or different subsets of the data [117]. The updates from each subset must be highly compatible in order to ensure correctness. However, because model parameters are distributed to different workers, and these arbitrary parameter subsets are dependent on each other, independent updates will break the simplex constraint and lead to incorrect estimates. To solve such problems, algorithms that support sophisticated constraint and consistency satisfiability mechanisms have been developed: Bradley et al. [117] proposed model-parallel coordinate descent, which allows updating of multiple parameters in parallel. Scherrer et al. [118] updated groups of independent parameters in parallel to avoid interference and loss of correctness during concurrent parameter updates.

More specifically, there are two main challenges for the parallelization of DL models. The first is splitting the model into partitions assigned to the parallel workers [119]. A common approach to find a good model splitting is to use reinforcement learning [120, 121]: Starting from some initial partitioning, permutations on that partitioning are performed, and performance is measured(*e.g.,* for one training iteration). If performance is improved, the permutation is maintained, and further permutations are performed, until the measured performance converges [122].

The second challenge is how to reduce communication costs between workers. Muller et al. [123] proposed an idea to introduce redundant computations to neural networks. In particular, this method partitions a DNN such that each processor is responsible for twice the neurons (with overlap), and would thus need to compute more but communicate less. Another proposal is to use Cannon's matrix multiplication algorithm to reduce communication in fully connected layers, modified for DNNs [124]. It is reported that Cannon's algorithm outperforms simple partitioning on small-scale multilayer fully connected networks in both efficiency and speed.

The main advantage of model parallelism is to conserve memory. Because the entire model is not stored in one place and the model is split into multiple workers, less memory is needed for each worker. This is useful when the complete model is too large to fit on a single device, which can occur, for example, when the device consists of specialized hardware such as GPUs or TPUs. The disadvantages of model parallelism are the heavy communication that is needed between workers. This is a critical issue, especially when training of DL models, as complex neural networks are hard to split effectively, and

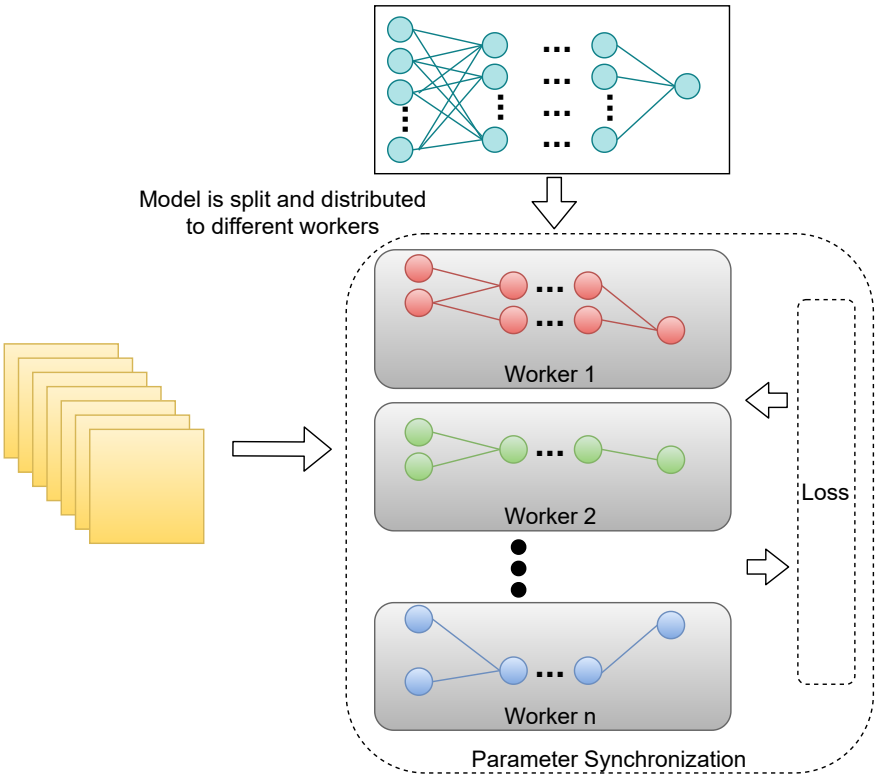


FIGURE 3.2: Model parallelism

stalling of workers can occur because of communication overhead and synchronization delays.

3.3.2 Data Parallelization

Different from model parallelization, data parallelization [125, 126] implies that the training data is split into non-overlapping chunks and fed into the model replicas of the workers for training, while an identical copy of the DNN model is loaded in the workers. Each worker performs the training independently on its shard of the batch, leading to updates of the model parameters. Two main methods are employed in data parallelism: synchronous training and asynchronous training. As the name infers, synchronous training pools the parameter updates after each iteration; in asynchronous training, the models are updated on each work completely independently from each other. In data parallelism, several challenges in synchronization and communication exist. Especially when the model has many parameters, synchronization becomes the bottleneck [115, 127].

The scaling of data parallelism is naturally defined by the minibatch size. When performing stochastic gradient descent (SGD), it is common to decrease the number of weight updates by computing the sample loss in minibatches, averaging the gradient

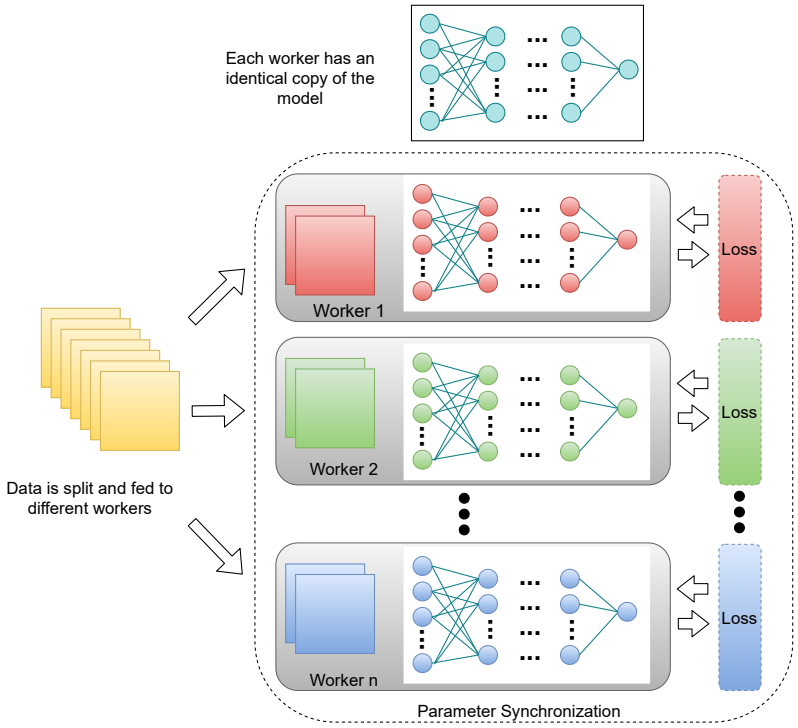


FIGURE 3.3: Data parallelism

concerning subsets of the data [128]. The minibatch method aims to achieve a balance between SGD and batch methods [129], where the whole dataset is used in each iteration, suggesting that the use of minibatches could be regarded as a form of data parallelism. Additional approaches for data parallelism using MapReduce are proposed [130][131][132][133] .

When trying to achieve data parallelism in HPC, several works have shown how high-performance communication interfaces, *e.g.,* MPI can be used to implement fine-grained parallelism features. Gaunt et al. [134] revealed how to reduce latencies via asynchronous execution and pipelining. Further, Zhao et al. [72] and Renggli et al. [135] introduced how to utilize *allreduce* in Kylix [72] and SparCML [135] to achieve better performance through sparse communication. In Kylix, a nested, heterogeneous-degree butterfly *allreduce* network is implemented to synchronize models, maintain distributed datasets, and perform operations on distributed data. In SparCML, sparse *allreduce* operations are generalized by allowing processes to contribute arbitrary sparse input data vectors. In addition, it is able to switch a sparse and dense representation automatically by a simple performance model. Oyama et al. [136] and Zlateski et al. [137] studied how to exploit parallelism within a given computational resource through decomposing [136] or computing [137] micro-batches (fragmented from minibatches). Doing so makes it possible to reduce the required memory footprint and enable further hybrid CPU-GPU inference.

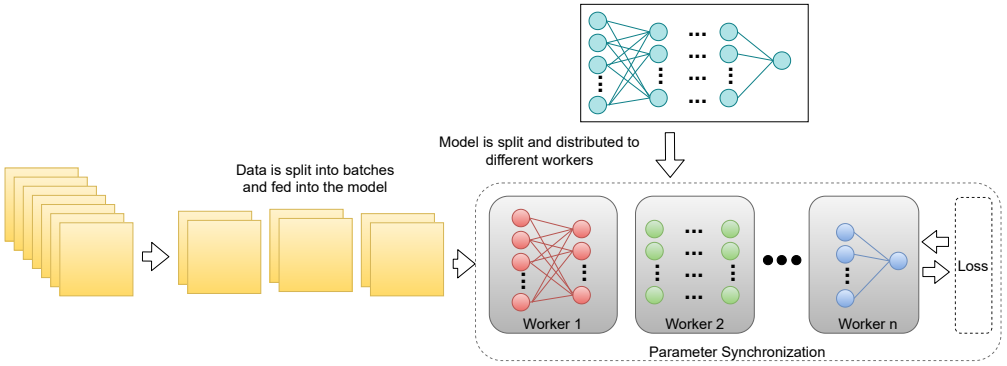


FIGURE 3.4: Pipeline parallelism

3.3.3 Pipeline Parallelization

Pipeline parallelism [138][139][140] can be viewed as a combination of model parallelism and data parallelism, as the model is split into different workers for training and the training data is also split into micro batches. In particular, when training of DL model, data are processed through the network in parallel (data parallelism) and the length of the pipeline is determined by the DNN structure (model parallelism) [141]. This methodology is widely adopted by most DL frameworks, *e.g.,* TensorFlow [142], Torch [143] and Caffe [144] through overlapping forward evaluation, backpropagation, and weight updates. Colin et al. [145] proved that using randomized smoothing [146][147] in pipeline parallelism which replaces the usual gradient information with an average of gradients sampled around the current parameter, is particularly effective in specific fields such as few-shot learning [148], deep reinforcement learning [121][120] or adversarial learning [149][150].

Gpipe [138] and Pipedream [151][152] are scalable pipeline parallelization framework that provide a solid and efficient way of applying pipelining techniques to neural network training. Here, network layers are partitioned and training samples flow across them, only waiting for the next layer to be free, increasing the overall efficiency in a nearly linear way [145].

3.3.4 Local Parallelization

The parallelism methods mentioned above are trying to make the backpropagation more efficient, which also brings the limitation that the network parameters of each layer can only be updated in turn after completing the full forward pass(backward locking). This backward locking results in increased memory overhead, and precludes efficient parallel processing across layers [153]. To overcome these challenges and scale more efficiently with compute than backpropagation, local parallelization was proposed by Laskin et al.

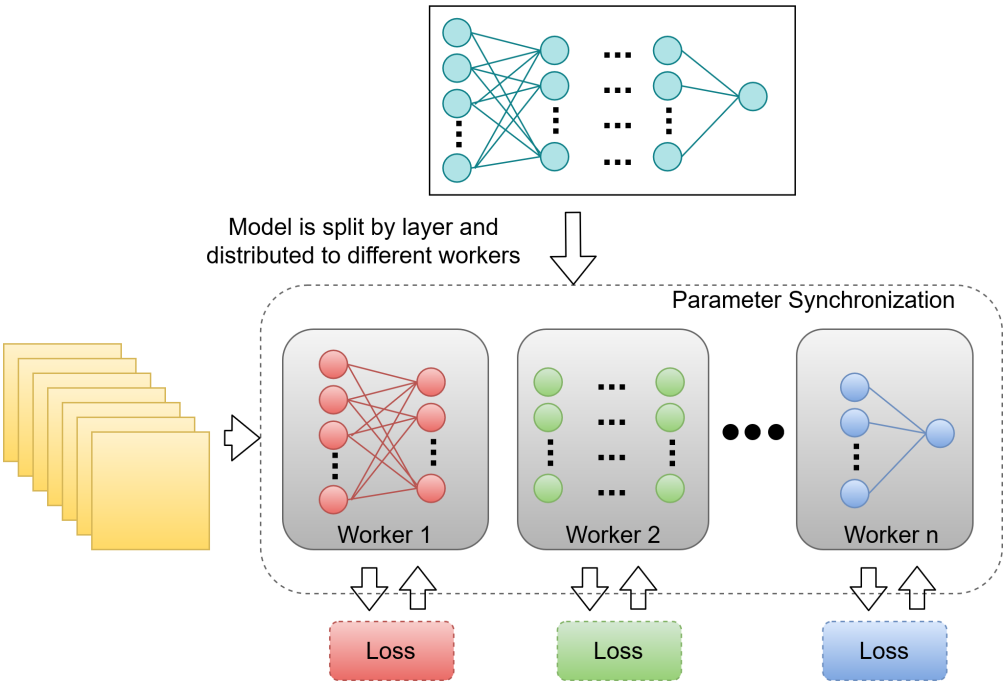


FIGURE 3.5: Local parallelism

[154], which parallelizes the training of individual layers in deep networks by replacing global backpropagation with truncated layer-wise backpropagation. As illustrated in Figure 3.5, the layers of a deep neural network are divided into a sequence of blocks, which may contain one or more layers. Each block is trained independently with an auxiliary objective on an individual worker, and receives the output of the activation by the previous block as input. Their study shows that local parallelism enables fully asynchronous layer-wise parallelism with a low memory footprint and requires little communication overhead compared with model parallelism.

3.3.5 Hyrbid Parallelization

DNN models are complex and different DNN models usually require different parallelization methods. To overcome the drawbacks of each scheme, a variety of combinations of multiple schemes for parallelization have been introduced. Below, hybrid approaches that mix model, data and pipeline parallelism are surveyed.

Different hybrid parallelism systems have been developed for distributed machine learning. For example, SystemML [155][156][157] is a systematic approach for combining different kinds of parallelism for large-scale machine learning on top of MapReduce. The core idea is to employ a dedicated ParFOR (Parallel FOR) construct, as known from HPC, to create optimal parallel execution plans. Angel [158] is a hybrid, large-scale machine learning system deployed by Tencent to accelerate performance while reducing the

TABLE 3.2: Comparison of different parallelism strategies

Strategies	Data Parallelism (DP)	Model Parallelism (MP)	Pipeline Parallelism (PP)
Data Split	x		x
Model Split		x	x
Communication	Synchronization of model parameters	Very high	Less communication overhead than MP
CPU/GPU Resources	High memory usage	Less memory usage	The more pipelines, the less memory usage on each worker
Downsides	Not applicable for large models	High communication cost; device underutilization; hard to split model effectively	Device underutilization

network latency by overlapping parameter pulling and update computing, and utilizing the sparseness of data to avoid the pulling of unnecessary parameters.

In addition to the algorithms mentioned above, several frameworks that utilize hybrid parallelism to empower the training of large-scale DL models have been proposed. Gaunt et al. [134] proposed a framework called AMPNet which implements asynchronous parallelism; tasks between layers are scheduled asynchronously and dynamic control flow is executed asynchronously. DistBelief [159] combines all three parallelism strategies, performing training on multiple model replicas simultaneously. Based on this idea, a distributed system called Adam [160], designed by Microsoft, exploits asynchrony throughout the system to improve not only the performance but also the accuracy of trained models.

3.4 Communication

In the training of DNN models on large-scale CPU and GPU clusters, communication is considered the most important bottleneck of performance as it requires much more time than computation. This is particularly true when using a data parallelism strategy on a large dataset or model parallelism for extremely large models. In HPC environments, such communication problems can be optimized due to HPC techniques for stencil computations such as MPI Neighborhood Collectives [161] or optimized Remote Memory Access programming [162]. However, due to the data-intensive nature of DNN training, traffic between nodes is hefty, and the performance-optimized interconnect of HPC cannot be fully utilized. For example, TensorFlow enables distributed training through Google's GRPC protocol [163], which is unsuitable for HPC as it requires at least one server and utilizes network protocols such as TCP/IP. Messages from such protocols

require more bandwidth than HPC interconnect messages, which downgrades the performance of the overall I/O pipeline in HPC.

Therefore, different technologies have been developed to offer good distributed training performance at minimum implementation overhead for DNN on HPC. Hoefler et al. [164] studied the communication requirements for training modern DNNs on HPC and developed an open-source library called Aluminum to enable optimized, asynchronous, GPU-aware communication on HPCs with distributed GPU systems using CUDA and MPI. Techniques that enable overlapping of communication and computation, which is a standard approach to reduce communication overheads, are employed. Additionally, a solution is proposed for solving the semantic mismatch between MPI and CUDA that leads to overheads increase and asynchrony limits. The NVIDIA Collective Communications Library(NCCL) [165] implements multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs. NCCL provides routines such as all-gather, all-reduce, broadcast, reduce, and reduce-scatter that are optimized to achieve high bandwidth and low latency over PCIe and NVLink high-speed interconnect. Intel(R) Machine Learning Scaling Library(MLSL) [166] is a library developed by Intel that provides an efficient implementation of communication patterns used in deep learning.

3.5 Tools and Frameworks

3.5.1 Deep Learning Frameworks

Since the rise of DL, many different DL frameworks and tools have been developed, in which different concepts of parallelization and distribution are implemented. Table 6.1 provides a detailed overview of DL frameworks.

TensorFlow [167], developed by Google and community contributors, is one of the most widely used ML/DL frameworks. It natively supports distributed and parallel training through both model parallelism and data parallelism. In data parallelism, it supports the centralized approach via parameter servers, using either asynchronous or synchronous training. TensorFlow can efficiently work with mathematical expressions involving multi-dimensional arrays. And it shows high scalability of computation across machines on huge data sets.

PyTorch [168], developed by Facebook and community contributors, is another popular DL framework. In PyTorch, distributed, data-parallel training as well as model-parallel training are supported out-of-the-box. For data-parallel training, PyTorch supports both

synchronous and asynchronous training through implementation of its both centralized and decentralized architecture. Compared with TensorFlow, Pytorch is easier to use, still it is more popular amongst researchers, not regularly used as a production framework in industry.

CNTK (Microsoft Cognitive Toolkit) [169] is a DL framework developed by Microsoft and community contributors. It provides a built-in mechanism for data-parallel and distributed training. CNTK has a centralized architecture with parameter servers, using asynchronous training. Currently it does not support model parallelism.

Deeplearning4j [170] is a DL framework developed by Skymind and community contributors organized by the Eclipse Foundation. Its data-parallel training is realized using both decentralized asynchronous and centralized synchronous approach. Model parallelism is currently not supported by Deeplearning4j. Distributed and parallel training are supported by using Spark/Hadoop, thus Deeplearning4j is able to process a huge amount of data without sacrificing speed.

MXNet [171] is a DL framework and an Apache project (incubating). MXNet supports model parallelism for multiple GPUs on a single node, although model parallelism for multi-node is not supported though. Data parallelism is implemented though the centralized architecture. MXNet supports synchronous and asynchronous training. To be noted here, MXNet has a dynamic dependency scheduler which enables auto parallelism. And its very good computational scalability with multiple GPUs and CPUs makes it shine with big, industrial style projects.

SINGA [172] is a DL framework and Apache project (incubating) which is developed by community contributors. SINGA supports distributed, data-parallel and model-parallel training, as well as hybrid parallelism out-of-the-box. Data parallelism is realized through a centralized approach. However, the decentralized architecture can also be emulated. Both synchronous and asynchronous training are supported. Although it is less popular compared with other DL frameworks, different studies have shown that SINGA has a great training scalability for big deep learning models over large datasets [173].

With so many frameworks mentioned above, it is important to bear in mind that there is no clear winner for all use cases due to the sensitiveness to different choices and problem settings [174]. And it is always not so easy to decide which one suits you better from the myriad of frameworks. Users need to be cautious as the performance of DL frameworks are highly dependent on the applicable area, the datasets as well as the hardware. Although hardware like CPU, GPU, TPU, FPGA and ASIC all can support DL training, some of them only support one specific framework, for example,

TABLE 3.3: Comparison of open source DL frameworks and libraries

Property	TensorFlow [167]	PyTorch [168]	CNTK [169]	DL4j [170]	MXNet [171]	SINGA [172]
Written in	C++, Python	C, Lua	C++	C, C++	C++	C++, Python
Supported languages	C++, Python, Go, JavaScript, Swift, Java	C++, Python	C++, Python, C#, BrainScript	Java	C++, JavaScript, Julia, Matlab, Python, Scala, R, Wolfram, Go, Perl	C++, Python
Hardware Support	CPU, GPU, TPU, Mobile	CPU, GPU	CPU, GPU	CPU, GPU	CPU, GPU, Mobile	CPU, GPU
Synchronized	x	x	x	x	x	x
Asynchronized	x	x	x	x	x	x
Centralized	x	x	x	x	x	x
Decentralized		x	x	x		x
Parallelism Mode	Data, Model, Pipeline	Data, Model, Pipeline	Data	Data	Data, Model, Pipeline	Data, Model, Hybrid
CUDA Support	x	x	x	x	x	x
OpenMP Support		x		x	x	
Communication	gRPC, NCCL	MPI, NCCL	MPI , NCCL	Spark, Aeron	NCCL	MPI, NCCL, ZeroMQ
ONNX Support		x	x		x	x

TPU only supports TensorFlow. One good thing is that CNTK, PyTorch, MXNet all support Open Neural Network Exchange (ONNX) [175] format, which is co-developed by Microsoft and Facebook and allows to easily transform models between them and other DL tools.

3.5.2 Scaling DL Frameworks on HPC Systems

Although the frameworks mentioned above offer native support for distributed computation, most of them are not suited for HPC architectures. Mathuriya et al. [163] showed that native support of distributed training from TensorFlow was accompanied by a decrease in the efficiency on the worker when it is scaled up to 128 nodes. To

overcome this obstacle and achieve better performance, several frameworks that can be regarded as distributed training middleware sitting between the DL framework and the communication runtime have been developed.

Horovod[103], developed at Uber, is seamlessly integrated into TensorFlow and PyTorch programming. It uses MPI as a mechanism for communication to allow multi-node training, enabling benefits from optimizations made in the underlying MPI library, such as *allgather* and *allreduce* during handling of cross-replicas communication and weight updates. This is different from the TensorFlow hierarchical architecture in which a centralized parameter server is utilized to pass parameter updates. Mesh-TensorFlow [176] is a language extension for TensorFlow which targets at specifying a general class of distributed tensor computations. It allows for combining data parallelism and model parallelism. Through Mesh-TensorFlow, batch-splitting problems that occur when scaling TensorFlow in HPC including the inability to train very large models (due to memory constraints), high latency, and inefficiency at small batch sizes are solved through the combination of data parallelism and model parallelism. With MPI-allreduce across mesh dimensions, the operations can be particularly efficient. Cray Programming Environment (CPE) ML Plugin [177] is another solution for scaling DL frameworks to HPC, especially on Cray XC supercomputers. This deep learning framework portable communication plugin based on MPI. It couples itself to a TensorFlow graph with a provided custom TensorFlow operation. Similar to Horovod, it is able to directly operate on in-graph memory. HyPar-Flow [178] is a scalable, practical, and user-transparent system for hybrid-parallel training on HPC that exploits MPI, Keras, and TensorFlow. Data, model, and hybrid parallel training of any Keras model at scale can be provided through a single API. Further, it exploits pipelining to improve performance and leverage efficient MPI primitives for scalable communication. HPDL [179] and HeAT [180] are other frameworks for high-performance distributed ML and DL that are compatible with existing frameworks and adaptive to HPC.

Except from the frameworks mentioned above, Jacobs et al. [181] introduced work related to the development of the Livermore Big Artificial Neural Network (LBANN), an open-source deep learning toolkit for training DNNs at scale on HPC resources. Appropriate DL algorithms can be selected, trained and optimized through LBANN so as to achieve the best possible results in network quality and parallel performance. It supports model parallelism using distributed matrix operations, and data parallelism using distributed mini-batches.

Some recent work and frameworks for distributed deep learning training is also reviewed and summarized in Table 3.4.

TABLE 3.4: An overview of recent researches on distributed DL frameworks and libraries

Property	Data Parallelism	Model Parallelism	Pipeline Parallelism
Pytorch DDP [182]	x	-	-
DeepSpeed ZeRO-3 [183]	x	-	-
Megatron [184]	x	x	-
Pytorch Gpipe [185]	-	-	x
DeepSpeed 3D [186]	x	x	x
FlexFlow [187]	x	x	-
PipeDream [188]	x	-	x
DAPPLE [189]	x	-	x
Unity [190]	x	x	x
Alpa [191]	x	x	x
Galvatron [192]	x	x	x

3.6 Conclusion

This chapter profoundly reviews the techniques that bridge the gap between HPC and deep learning workloads. It compares the differences between the typical HPC and DL workloads, which is followed by the description of different technologies that are commonly used for setting up the environment for DL workloads on the HPC system. Furthermore, various optimization methods are described and compared from different aspects, which enable the seamless execution of deep learning applications on HPC, including parallelization methods, communication optimization, *etc.* In addition, it evaluates and compares different frameworks and tools for distributed training of DL workloads which are widely utilized in both academia and industry.

Chapter 4

Hybrid Workflow of HPC and Deep Learning for Material Characteristic Identification

This chapter proposes a novel hybrid workflow combining a multi-task neural network and the simulation on HPC systems, which can address the problem of data sparsity for deep learning workloads and reduce the demand for expertise, resources, and time in determining the validated parameters for the simulation. Nowadays, machine learning (ML), especially deep learning(DL) methods, provide promising solutions in many real-life problems. However, the lack of training data is often a crucial issue for these learning algorithms, the performance accuracy of which relies on the amount and the quality of the available data. This is particularly true when applying DL-based methods for specific areas *e.g.,* material characteristics identification, as it requires a huge cost of time and manual power to get observational data from real life. In the meanwhile, simulations on HPC have already been commonly used in computational science due to the fact that it has the ability to generate sufficient and noise-free data, which can be used for training the DL-based models. However, to achieve accurate simulation results, the input parameters usually have to be determined and validated by many tests. Furthermore, the evaluation and validation of such input parameters for the simulation often require a deep understanding of the domain-specific knowledge, software and programming skills, which can, in turn, be solved by DL-based methods. This work is demonstrated through experiments on the identification of material characteristics, and the results prove a promising performance (MSE = 0.0386) through such a workflow.

Chapter outline: This chapter is organized into five sections. In Section 4.1, the introduction to the problem is provided. Section 4.2 reviews the related work. The proposed

methodology, including the workflow, the simulation, the DNN model, the experiment, and the optimization, is explained in Section 4.3. In Section 4.4, a general approach which is implemented through the AutoML, namely multi-task network architecture search, is described, which is followed by the concluding remarks in Section 4.5

4.1 Introduction

Machine learning, specifically deep learning methods, revolutionized several application domains in the past decade. While deep learning has demonstrated strong abilities at extracting high-level representations of complex processes, the need for sufficient ground truth data is often a critical issue faced in various areas. In fact, it is almost impossible to generate enough data in real life for supervised learning in many real-world problems, which are limited by scientific instruments, the physical phenomenon itself, or the complexity of modelling. Nowadays, different methods have been developed to solve this problem, *e.g.,* transfer learning, data augmentation, usage of synthetic data, generation of new data through GAN, *etc.* Recently, scientists and engineers have begun experimenting with a relatively new approach to understanding complex systems using DNNs, trained by the virtually unlimited data sets produced by simulations [193]. Studies have proven that these "synthesis models," combining ML and traditional simulation, can improve accuracy, accelerate time to solution, and significantly reduce costs [194].

Input parameters have to be determined and validated by a large number of tests to expect accurate simulation results [195]. Furthermore, the evaluation and validation of such input parameters for the simulation often require a deep understanding of the domain-specific knowledge, software and programming skills. Thus, how to efficiently define and validate the input parameters for the simulation becomes the key factor in the development and design of numerical models. While simulation can solve the data sparsity problem for DNN models, DL-based methods can, in turn, solve the difficulty in the determination and validation of the input parameters for simulation by training DNN models. However, both pieces of training of DNN model and the running of simulations are compute-intensive tasks in which the supercomputers can manifest their computation efficiency.

Contributions: In this chapter, a hybrid workflow of deep learning and simulations on HPC is proposed to address the problem: i) lack of sufficient ground-truth training data for deep learning; ii) difficulties in definition and validation of the input parameters for the simulation; The effectiveness of the workflow is demonstrated through the experiment on the identification of material characteristics. Further, the time consumption of each process within the deep learning workload is analyzed, based on which several

optimization approaches are adopted to improve the compute efficiency significantly. In addition, a Neural Architecture Search (NAS) based AutoML approach is implemented, which generalizes the workflow and enables it to be data- and use-case- independent.

4.2 Related Work

Virtual data generated by simulations have been widely used for improving the performance of DNN models in various areas: education [196], medical [197], computational imaging [198], computational mechanics [199], climate modeling [200], *etc.* In the other way round, the ML/DL aided simulation has also developed very fast, many studies and researches have been done. Hu et al. [201] proposed a Long Short-Term Memory (LSTM) network to help improve the simulation of rainfall-runoff, so that potential flood can be predicted. Yeo et al. [202] developed a DNN model, DE-LSTM, to model the nonlinear dynamics so as to be used for the simulation of stochastic processes. Other researches on ML/DL aided simulations have also been reported [203–205].

In the area of computational materials science, it is more and more common to train learning algorithms with such virtually generated data due to the fact that Finite Element Method(FEM) [206] simulation can provide as much noise-free data as required [207, 208]. Traditional methods *e.g.,* finite element model updating (FEMU) [209], Virtual Field Method (VFM) [210], iDIC [211], *etc.* have been used for a long time to determine and validate the input parameters for FEM simulation. Recently, due to the swift development of machine learning especially deep learning, methods based on machine learning algorithms have also been developed . Gorji et al. [212] proposed a three hidden layer neural network model to reproduce the force-displacement curves of the tensile tests with a prediction accuracy around 93%. Koch et al. [213] designed a simple neural network model for the estimation of yield curve parameters with the data from a tensile test. Chheda et al. [214] proposed a neural network based method for predicting forming limit curves based on chemical composition and rolling process parameters of sheet metals. Mozaffar et al. [215] presented a recurrent neural network model to model the complicated path-dependent plasticity behavior, which demonstrated a very promising performance.

4.3 Methodology

In this section, it will be discussed how the simulation and the DNN model for prediction are defined. As illustrated in Figure 4.2, the proposed method is consisted of three

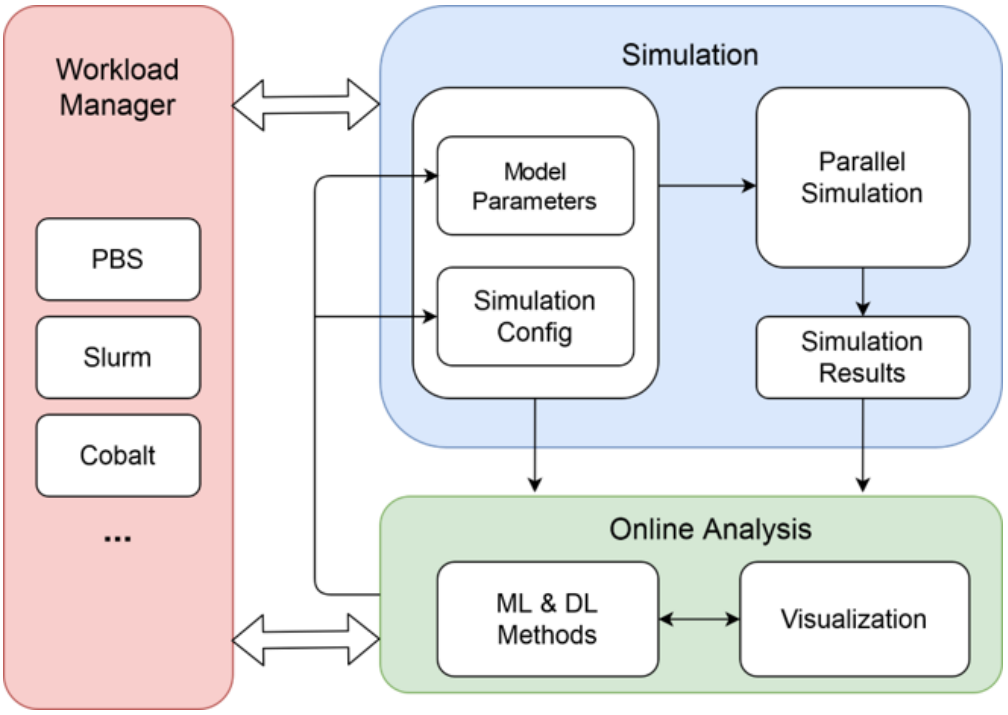


FIGURE 4.1: General workflow of deep learning aided simulation on HPC

phases: data generation phase, training phase and inverse phase. In the data generation phase, a set of material tests were simulated using a variety of material parameters as inputs and recorded the results of the simulation. In the second phase, a DNN model is trained with the simulation outputs as inputs and the material parameters which are simulation inputs as outputs. In the third phase, the prediction values and the real input values are compared to improve the DNN model and the simulation performance. Therefore, the FEM simulation can address the problem of data sparsity and the introduction of machine learning methods can reduce the high demand of expertise, furthermore, both the performance of simulation and DNN model are improved.

4.3.1 Simulation

The Barlat-3 parameter model [216] was employed in the data generation phase to define the anisotropic material behavior. In combination with the three anisotropy values (r0, r45, r90), other 5 parameters (M, Enorm, Sratio, c_mult, n_pow) are also included to set up the simulation, where the value ranges of these material parameters were defined in such a way that the calculation effort is manageable and the generated data set contains sufficient information for the feasibility study of the approach. It has to be noted here that all FE simulations in this study were calculated using a predefined flow curve. The specimens were stretched by 3 mm, giving a maximum value of about 0.1 for the equivalent plastic strain. Three tensile specimens were considered in rolling, transverse

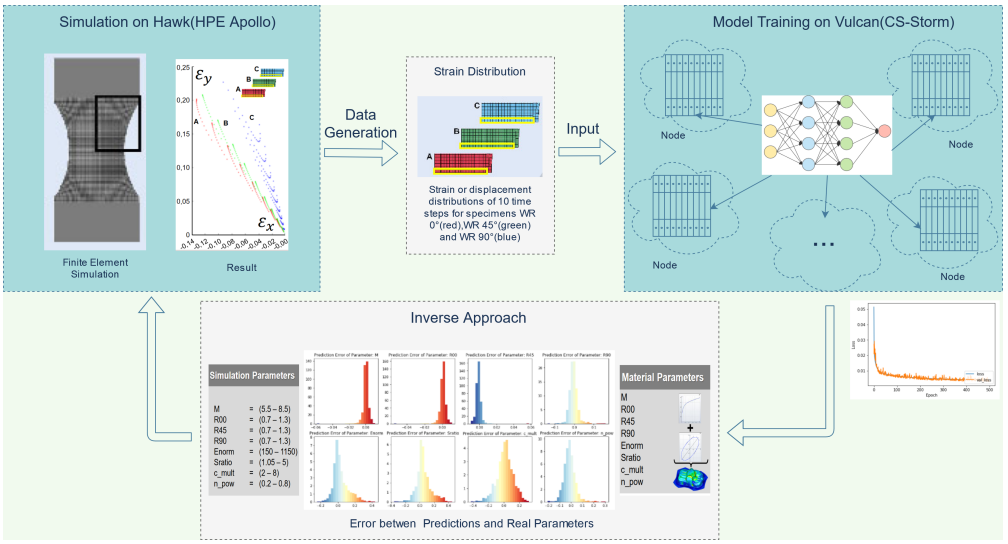


FIGURE 4.2: Hybrid workflow of simulation and distributed deep learning on HPC for material characterization

and diagonal directions. Due to the orthogonal symmetry of the material behaviour, only a quarter of the tensile specimen was modelled. The strain values of the elements located in the yellow area of the tensile specimen were extracted. This narrowest region of the tensile specimen is stretched the most. Details of the selected finite element model can be obtained from [217].

4.3.2 Multi-Task Neural Network

As the input parameters for the simulation are individual values and the prediction of such individual values can be regarded individual tasks, a multi-task neural network is proposed based on the fact that multi task learning (MTL) can help improve the performance by introducing an inductive bias, leveraging the domain-specific information [218], increasing the sample size, and focusing its attention on those features that actually matters. The network structure is designed by hard parameter sharing, which is the most commonly used approach to MTL in neural networks that can greatly reduce the risk of overfitting [219]. The overall structure of the model is depicted in Figure 4.3. The whole network is composed of two main parts: the shared network and individual network for each parameter output. In the shared network, 1D CNN layer and max-pooling layers are used to extract the global features. When designing the part of individual network for each output, it is noticed that for the outputs(M, R00, R45, R90), 1D CNN layers and max-pooling layers followed by two fully connected layers could give out best prediction performance, while for individual network of the other outputs(Sratio, c_mult, n_pow, Enorm), a more complicated network which have more convolution and max-pooling layers should be designed to better learn the features. This is also reflected in Figure

4.5, where the loss for the outputs(Sratio, c_mult, n_pow, Enorm) are higher than for the outputs(M, R00, R45, R90) even with the more complicated network structure. In the mean while, Batch Normalization is employed here to stabilize the learning process and Dropout layers are used to avoid overfitting.

In terms of training the network, the primary loss function used is the Mean Squared Error between the between the real field y rand the predicted field $\hat{y}$. To train the networks, a weighted MSE is employed as the loss function. As the 8 outputs are weighted equally, the loss function is described as:

$$L = \sum_{1}^{N} \sum_{i=1}^{I} \frac{1}{I} (y_i - \hat{y}_i)^2 \tag{4.1}$$

where N denotes the number of outputs to be predicted.

4.3.3 Experiment

The training dataset was created using FEM simulations of tensile tests, which contains the values of the x-strains in the first half and the values of the y-strains in the second half . Each third of these halves refers to a specimen with 18 elements. For each element, 10 consecutive strain values from 10 time steps are taken. For each finite element, the longitudinal and transverse strains were exported for 10 time steps. A total of 1080 strain values were thus obtained per FE simulation, which means that each record of the FM simulation output is a 1080 size vector, and the total dataset is around 3 TB composed of 4,941,258 records, where train and test dataset are split from with a ratio (0.9,0.1). One record of the simulation output is as illustrated in Figure 4.4, which can be regarded as a sequence of features from 6 dimensions coming in 10 time steps.

In this section, the performance of the model is evaluated by inspecting the change of losses during training and testing it on the validation dataset. The losses of each output and as a whole during the training process are shown in Figure 4.5, the total training loss is around 0.0343 and the total validation loss is around 0.0386. The error histogram shown in Figure 4.6 indicates that the normalized maximum error of the values for outputs (R90, Enorm, Sratio, c_mult and n_pow) are about 0.2, whereas about 90 % of the errors are below 0.1. The maximum error of the values for outputs (M, R00, R45) of the error histogram is only 0.02, whereas about 90 % of the errors are below 0.01. It was shown that the input material parameters for simulation can be approximated relatively well by our MTL model, and the performance of our MTL model keeps improving along with the increase of simulation data, which leads to the fact that both the simulation and DNN performance can be improved through such workflow.

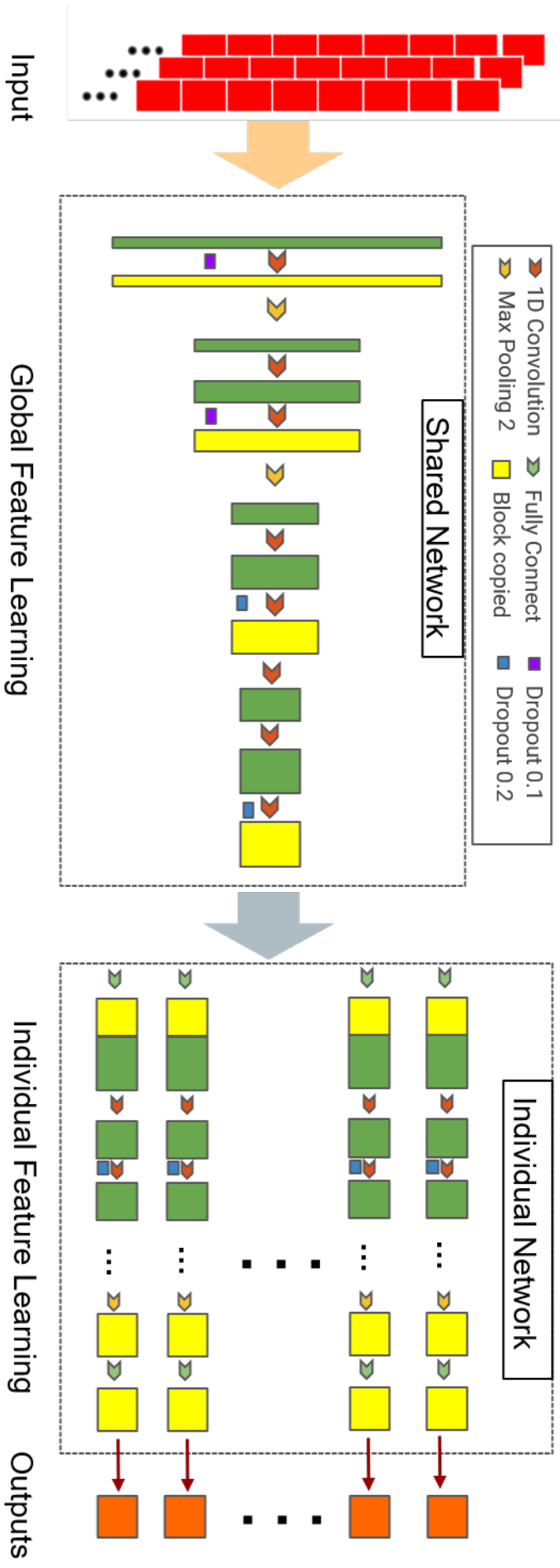


FIGURE 4.3: DNN model structure

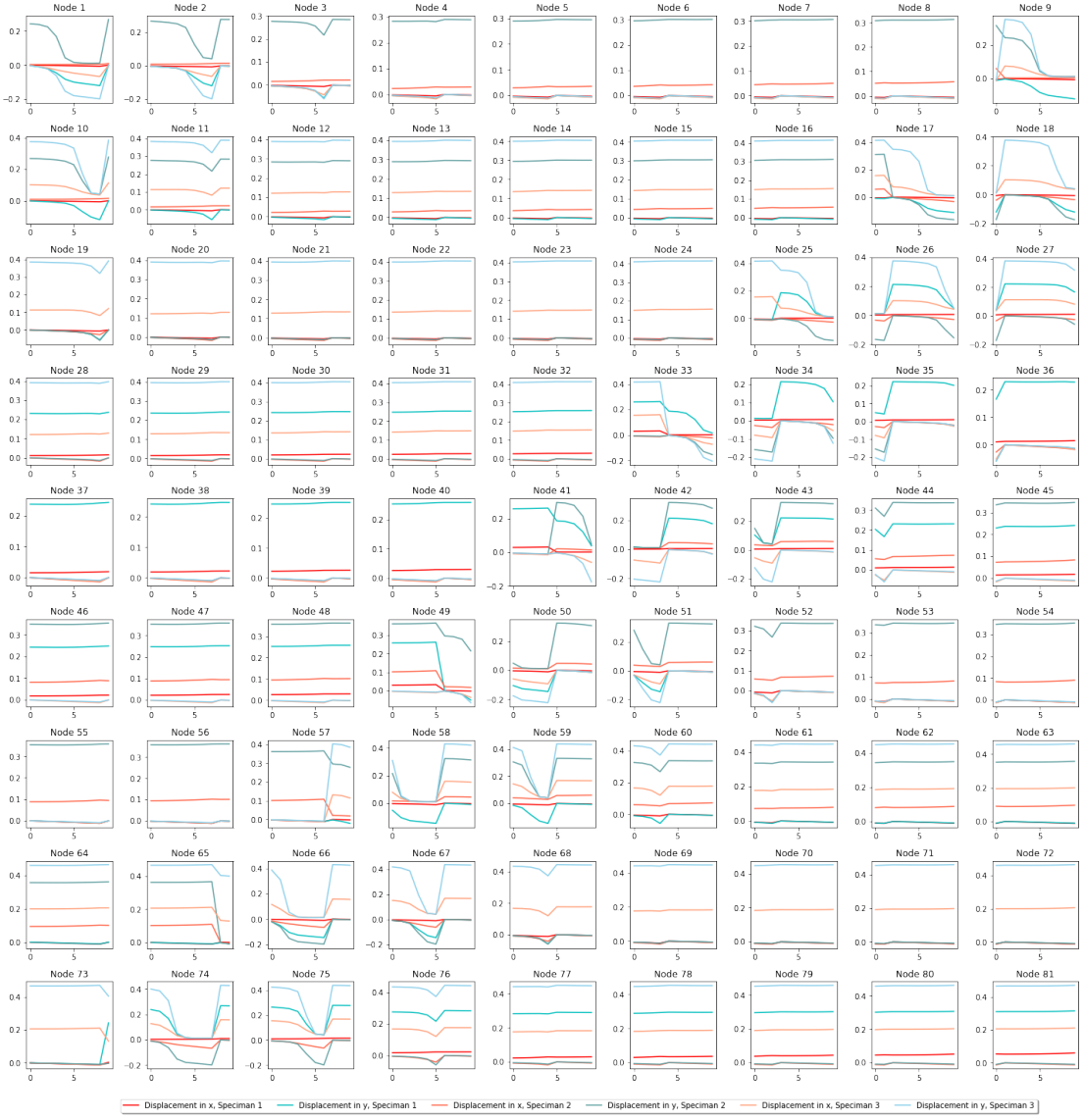
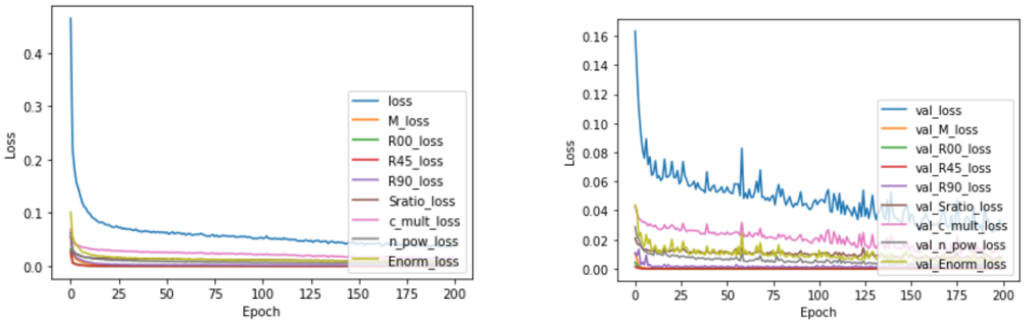


FIGURE 4.4: Visualization of the dataset generated by FEM simulation



(A) Training loss (B) Validation loss

FIGURE 4.5: The training and validation loss of the multi-task regression dnn model

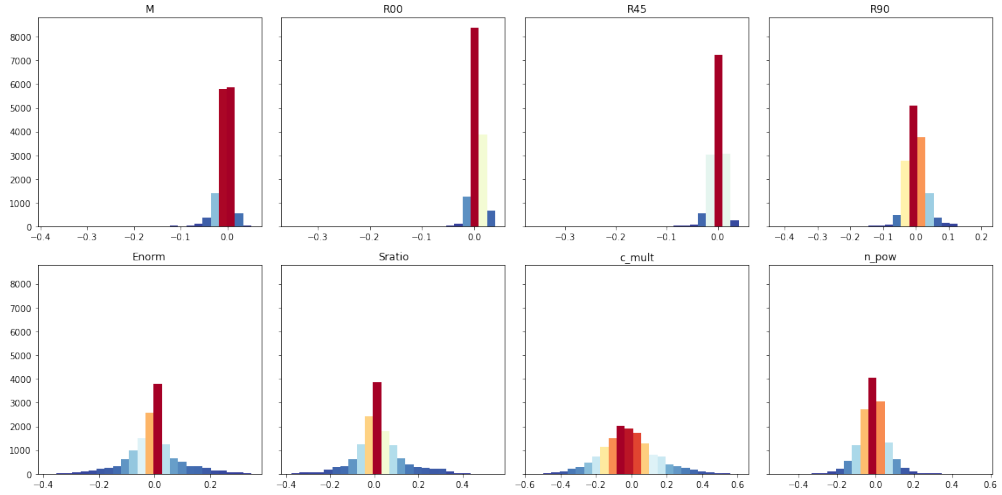
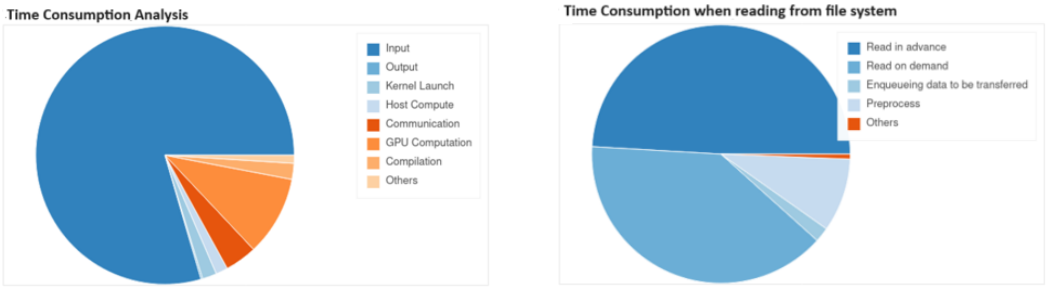


FIGURE 4.6: Errors between the predicted values and the real value of the 8 parameters

4.3.4 Optimization

As the whole dataset is around 3 TB and the DNN is trained on multiple nodes in a distributed manner, different optimization methods are required to improve the training performance, *e.g.,* learning rate schedule, data I/O optimization, *etc.*



(A) Time consumption analysis of each process

(B) Time consumption analysis when reading from file system

FIGURE 4.7: Profile of DNN model training process

4.3.4.1 Learning Rate Schedule

In the training of DNN, most of the models use stochastic gradient descent (SGD) for optimization. It is usually difficult to decide a adequate learning rate, as a large learning rate will lead to overshooting loss minimum and a small learning rate will lead to a extremely slow convergence. Therefore, a well designed learning rate scheduler is gaining popularity in gradient based optimization, which has the ability of achieving optimal asymptotic convergence rate and escaping from poor local minima [220]. Compared with static learning rate, a learning rate scheduler usually uses larger learning rate values at

the beginning of the training procedure which makes large changes, and decreases the learning rate such that a smaller rate and therefore smaller training updates are made to weights later in the training procedure. In this work, a exponential decay sine wave learning rate [221] is applied, which scans a range of learning rate in each epoch and varies in sine wave way in the training process. The control of the learning rate is according to:

$$lr(t) = lr_0 e^{\frac{-\alpha t}{T}}(sin(\beta \frac{t}{b2\pi}) + e^{\frac{-\alpha t}{T}} + 0.5) \tag{4.2}$$

where t is the number of epochs, lr_0 is the initial learning rate, T denotes the total number of eopchs and b is the number of batches. α and β can control the decay and oscillation nature of the learning rate. So that the learning rate would vary in a sine way during the training process, while the maximum value of sine wave would decay exponentially along with training epochs.

4.3.4.2 Distributed Strategy

In order to take advantage of the great computation power provided by Vulcan, it is necessary to do the training distributedly. To achieve this goal, a data parallelism strategy is adopted, where the whole dataset is split into different batches and assigned to different GPUs. In the meanwhile, a replica of the model is created per GPU, each variable in the model is mirrored across all the replicas. All variables are synchronized by applying identical updates. As the code is implemented in Tensorflow 2, efficient all-reduce algorithms implemented in NVIDIA Collective Communication Library(NCCL) are used to do the communication across all GPUs which can reduce the overhead of the synchronization significantly.

4.3.4.3 Data Pipeline Optimization

As is shown in Figure 4.7, input pipeline of dataset from file system takes the longest to execute during the whole training process. The input pipeline performs actual I/O, decoding and pre-processing, among which the read and pre-process take the longest to execute. Thus, to achieve the peak performance, it is necessary to implement an efficient input pipeline, especially a efficient read and pre-process, that delivers data for the next step before the current step has finished. The common approach is to optimize this process by overlapping the input pipeline with the computation pipeline [222]. In this work, further optimization methods have been adopted:

- Parallel I/O: Input files are read and pre-processed individually with individual outputs as a embarrassingly parallel process. The parallelization of the I/O of

many files (343 in our case) can be done by mapping the list of the file names for transformation. The I/O will be executed by threads that are spawned by the runtime, where the number of threads is specified manually.

- Prefetching: Prefetch is used here to ensure there will be a specified number of batches ready for the consumption, where the data from the previous pipeline is 'prefetched'. The prefetcher runs as a background thread and a consumption function, which contains an infinite loop waiting for a condition variable.

- Caching: The cache of data in memory or on local storage can save some of the operations like file opening and data reading from being executed during training. By applying the cache method, the transformations before the cached one are executed only during the first epoch, the following epochs will reuse the data cached.

The comparison of execution time between the training before optimization and after optimization is shown in Figure 4.8. As can be seen that, by applying the optimization methods listed above, the execution time of the whole process is decreased from around 179 minutes on 1 GPU, 25 minutes on 32 GPUs to 35 minutes on 1 GPU and 40 seconds on 32 GPUs.

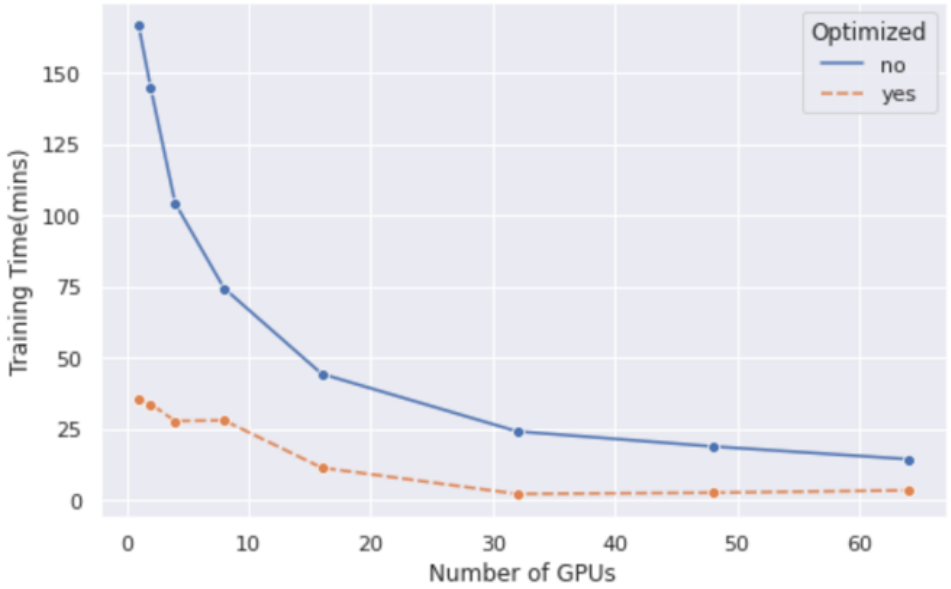


FIGURE 4.8: Comparison of the training time before and after data pipeline optimization

4.4 AutoML through NAS

In the sections above, a hybrid workflow of multi-task neural network and FEM simulation for identifying the material characteristics is introduced. Although the result is promising and demonstrates the effectiveness of the workflow, it still requires a lot of effort and expertise for manual designing and definition of the neural network structure, optimization, hyperparameters, *etc.* Therefore, further studies are required to bridge the gap between the expertise of mechanics and machine learning. In this section, an AutoML approach is described to replace the manual designed multi-task neural network in the workflow, which eliminates the need for manual tuning.

AutoML, which aims to automate a machine learning system, has attracted a lot of attention in the research community. Compared with traditional machine learning, AutoML automates many of the time-consuming and repetitive tasks involved in traditional ML, such as feature engineering and model selection, thus it is more time-saving. Moreover, AutoML algorithms can automatically search for the best hyperparameters, eliminating the need for manual tuning, therefore requires less technical expertise and makes it easier for non-experts to apply ML algorithms to their data.

To fulfil this, an multi-task-learning neural architecture search(MTL-NAS) which developed by Gao et al. [6] is implemented here to replace the multi-task NN in the workflow. NAS aims to automatically design well-performing neural network architectures for specific target task, to explore neural network architectures that outperform the ones designed by human experts, and largely reduce the human efforts [223]. The MTL-NAS incorporates NAS into general-purpose multi-task learning (GP-MTL), which are disentangled into single-task backbones to adapt to different task combinations. In addition, hierarchical and layer-wise features sharing/fusing scheme is extricated across single-task backbones to enable a general task-agnostic search space, which inserts cross-task edges into fixed single-task network backbones. Some description about the MTL-NAS can be found in Appendix A, for more information please refer to [6].

The error histogram shown in Figure 4.9 indicates that the normalized maximum error of the values for outputs (Enorm, Sratio, c_mult and n_pow) are about 0.2, whereas about 90 % of the errors are below 0.1. The maximum error of the values for outputs (M, R00, R45, R90) of the error histogram is only 0.05, whereas about 90 % of the errors are below 0.01.

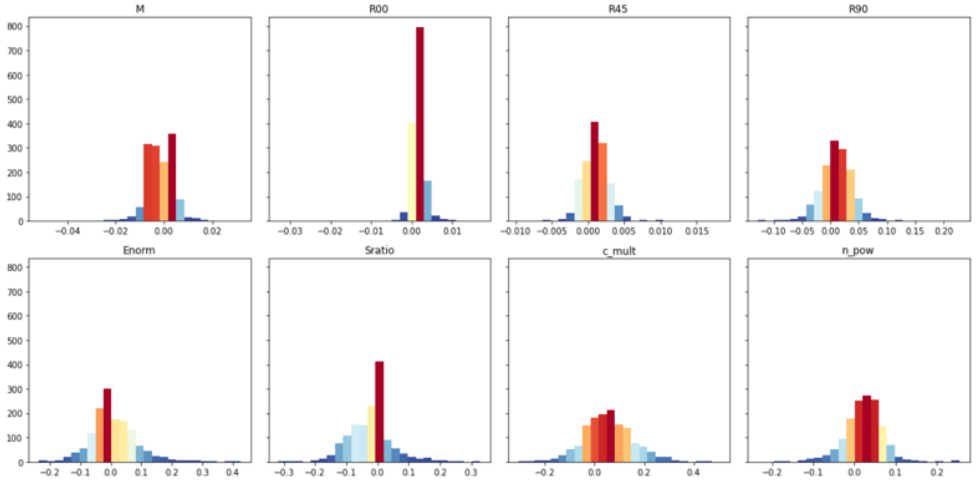


FIGURE 4.9: Errors between the predicted values given by the MTL-NAS model and the real value of the 8 parameters

4.5 Conclusion and Outlook

This chapter proposes and investigates a hybrid workflow composed of a FEM simulation and a DNN model on HPC. For the FEM simulation, the Barlat-3 parameter model is adopted to generate virtual data. Afterwards, the simulation outputs would then be fed as inputs to the DNN model which can predict the simulation input parameters in return. The DNN model is designed with a multi-task network architecture and scaled to multi nodes to accelerate the training. In order to improve the performance, several optimization methods are also discussed here. The conducted experiments successfully prove that such a workflow can improve the performance of the simulation and the DNN model.

In future work, more work will be carried on to improve the performance of the neural network, especially when scaling on an HPC environment. In addition, although the performance of MTL-NAS is promising, it is also noticed that the training time for NAS is extremely long. Therefore, more work should be done to further optimize the NAS algorithm, especially for the distributed environment, to accelerate the training.

Chapter 5

Evaluation of Variational Quantum Neural Networks for Image Classification

In the previous chapter, the description of the hybrid workflow of simulation and DL on HPC systems have been provided. However, even with the most advanced supercomputers, many problems in computational fields still need to be addressed. The need to solve these problems, combined with the ever-increasing complexity of issues, is faced by researchers. A radically new innovative approach is through a completely different paradigm: quantum computing. Quantum computing offers the possibility of revolutionary breakthroughs, such as turning computationally inaccessible problems into tractable ones.

This chapter evaluates the effectiveness of introducing quantum computing in DL through a comparative study. Experiments are conducted to show the advantage of QNN over classical DNN in data analysis, especially in the most commonly studied problem: image classification. The experiment results present that the convergence speed is highly improved with the involvement of a quantum layer or quantum neural networks, while there is no significant proof that it can improve the accuracy. Furthermore, due to the current hardware limitation, the number of qubits that quantum computers have is not always enough, which often raises the issue that the input information/training data can not be fully encoded.

Chapter Outline: This chapter is structured into four sections. Section 5.1 provides the introduction to the problem and explains the contributions. This is followed by the introduction of variational quantum machine learning in Section 5.2, which is followed by

Section 5.3, giving an overview of different quantum data encoding methods. Section 5.4 describes the comparative experiment and result analysis of several popular quantum neural network models and their corresponding classical DNNs. In Section 5.5, the concluding remarks are provided.

5.1 Introduction

In the past decades, quantum computing is anticipated to outperform classical computing. For example, problems that would take too long or are impossible to calculate on a classical computer could be calculated on a quantum computer within a reasonable amount of time. Therefore, many algorithms have been developed to obtain quantum advantages even with the current noisy intermediate-scale quantum computation (NISQ). Among them, Variational quantum machine learning (VQML) has drawn much attention since it can achieve linear or sublinear complexity as opposed to the polynomial complexity of conventional ML. In addition, a novel, useful and applicable concept has been proposed recently, known as quantum neural network (QNN). QNN has been developed by combining the basics of DNN with a quantum computation paradigm which is superior to the traditional DNN. Recent research finds that well-designed QNNs can achieve a higher capacity and faster training ability than comparable classical feedforward neural networks [224]. Moreover, compared to classical DNN, QNNs have various advantages [225], including 1) quantum parallelism; 2) memory capacity increased exponentially as compared to classical counterpart; 3)capacity to learn is faster as compared to traditional one; 4) higher and good stability and reliability; 5) performance is high with less number of hidden neurons; 6) single layer network solution of linearly inseparable problems; 7) information processing speed is high.

Although variational QNNs have become a leading strategy to obtain a near-term quantum advantage, important challenges in trainability, accuracy, and efficiency still exist. Accounting for all constraints imposed by NISQ computers with a single strategy requires an optimization-based or learning-based approach. Variational QNNs use parametrized quantum circuits that run on the quantum computer and then outsource the parameter optimization to a classical optimizer [56]. Compared to quantum algorithms developed for the fault-tolerant era, this approach makes quantum circuits shallower, thereby mitigating noise. Another challenge imposed by the QNNs is the barren plateaus, a notorious problem in QNN that occurs when the number of qubits increases. The barren plateaus vanish the gradients of the QML, making it impossible to guarantee trainability [226]. As a solution, Pesah et al. [227] proved that utilizing a quantum convolutional neural network (QCNN) with proper initialization can reduce the barren plateaus.

However, no research has been done before to thoroughly explore the quantum advantages brought by the QNNs in image processing, especially image classification. Therefore, in this chapter, the idea of QNNs is described, and experiments are conducted to evaluate the performance of QNNs compared to classical DNNs.

Contributions: In this chapter, several hybrid quantum-classical deep neural networks are evaluated and compared with corresponding classical DNN models. The quantum transfer learning based on Resnet18 is implemented to compare its performance with the original Resnet18. A Quanvolutioanl Neural Network is implemented, which adds a quanvolutioanl layer before CNN to encode the input image into four channels. Its performance is compared with the classical CNN model without this quanvolutional layer. In addition, a quantum CNN is implemented, which uses entanglement to fulfil the concept of convolution and pooling in classical CNNs. Their performance is evaluated and compared through experiments.

5.2 Variational Quantum Machine Learning

VQML [228] is a subfield of quantum machine learning that combines the principles of quantum computing with the techniques of variational optimization. It involves the use of quantum algorithms and quantum circuits to find approximate solutions to complex machine learning problems.

In VQML, a quantum circuit is designed to represent a parameterized quantum state, and the parameters are optimized to minimize a cost function that measures the quality of the solution. The optimization is performed using classical algorithms, such as gradient descent, and the quantum circuit is executed on a quantum computer or quantum simulator, as illustrated in Figure 5.1

VQML can be mathematically represented as an optimization problem. The main idea behind VQML is to find the parameters of a parameterized quantum state that minimize a cost function. Let's consider a cost function J that measures the quality of a quantum state $|\psi(\theta)\rangle$ represented by a parameterized quantum circuit. The goal of VQML is to find the parameters θ that minimize J, such that:

$$\theta^* = \arg\min_{\theta} J(|\psi(\theta)\rangle) \tag{5.1}$$

The cost function J can be defined based on the specific problem being solved, *e.g,* it could be the mean squared error (MSE) in a regression problem, or the cross-entropy loss in a classification problem. The optimization problem can be solved using classical

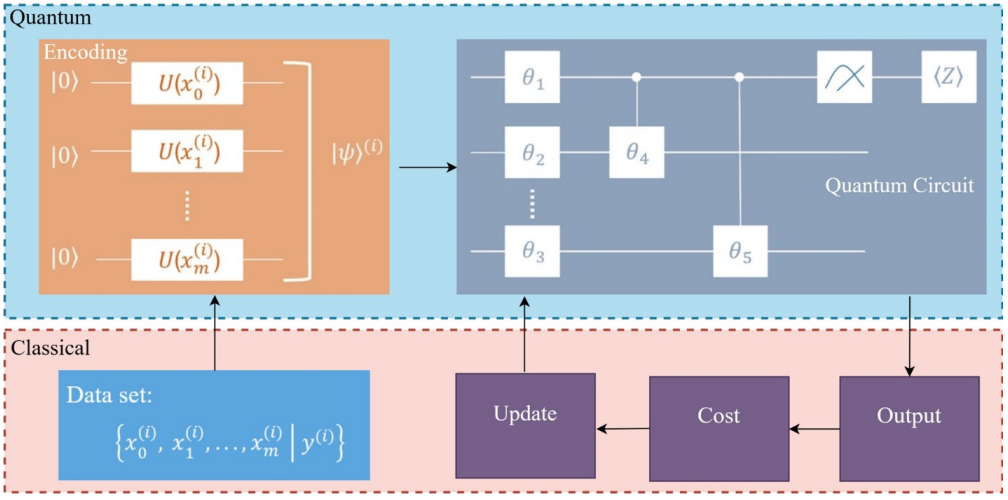


FIGURE 5.1: Variational quantum machine learning

optimization algorithms, such as gradient descent or stochastic gradient descent. The gradient of the cost function with respect to the parameters can be estimated using techniques such as finite differences or the quantum gradient, and the parameters can be updated in the direction of the negative gradient. In practice, the optimization is performed iteratively until convergence, or a stopping criterion is reached. The final parameters θ^* represent the optimal solution to the optimization problem, and the corresponding quantum state $|\psi(\theta)\rangle$ can be used to make predictions or perform other tasks, such as quantum feature extraction or quantum generative modeling.

VQML is a powerful technique for solving complex machine learning problems, as it combines the expressiveness of quantum states with the efficiency of classical optimization algorithms. One of the main benefits of VQML is that it enables the representation of quantum states that can capture complex and high-dimensional relationships between variables, which can be difficult to model with classical neural networks. Additionally, VQML algorithms are designed to be scalable and can be executed on current and future quantum hardware, making them a promising approach for solving real-world problems. By leveraging the unique features of quantum mechanics, such as superposition and entanglement, VQML has the potential to solve problems that are intractable for classical machine learning algorithms.

5.3 Data Encoding

As illustrated in Figure 5.1, a classical input information must first be encoded into quantum state to be processed by quantum computers. Data encodings for quantum computing define how data is represented by the state of a quantum system. Loading

data is not a trivial task in quantum computing as a variety of data encodings can be used depending on the requirements of the proper unitary transform of the algorithm. Every data encoding is essentially a trade-off between three major forces [229]:

- the number of qubits needed for the encoding should be minimal because of the limited qubits that current device can provide
- the number of parallel operations needed to realize the encoding should be minimal to minimize the width of the quantum circuit
- the data must be represented in a suitable manner for further calculations

In this section, an overview of different patterns for data encoding in quantum computing is presented.

Basis Encoding [230]: Basis Encoding represents data by encoding it as basis states of a quantum system. By this, each data point is associated with a specific basis state, and the quantum state is represented as a superposition of these basis states, with the coefficients of the superposition being the parameters that need to be optimized. In basis encoding, the quantum state is described by the probabilities of measuring each basis state, and these probabilities are used as the features of the data.

Basis encoding has the advantage of being simple and intuitive, as it directly maps data points to quantum states. It also has the advantage of being well-suited for encoding discrete data, as each data point is associated with a specific basis state. However, basis encoding also has the limitation that it may not be able to represent complex relationships between data points, as it only considers the probabilities of measuring each basis state and does not take into account the correlations between basis states.

Amplitude Encoding [230]: Amplitude encoding represents data by encoding it as amplitudes of quantum states. In this method, each data point is mapped to a quantum state, and the probability of measuring a particular state is proportional to the magnitude of the corresponding amplitude. The quantum state can be represented as a superposition of basis states, with the amplitudes of each basis state being the parameters that need to be optimized.

Angle Encoding [231]: Angle encoding represents data by encoding it as rotation angles of quantum states. In this method, each data point is mapped to a quantum state, and the quantum state is then rotated by a specified angle to encode the data. The angle encoding can be performed by applying unitary quantum gates to the quantum state, and the rotation angles are the parameters that need to be optimized.

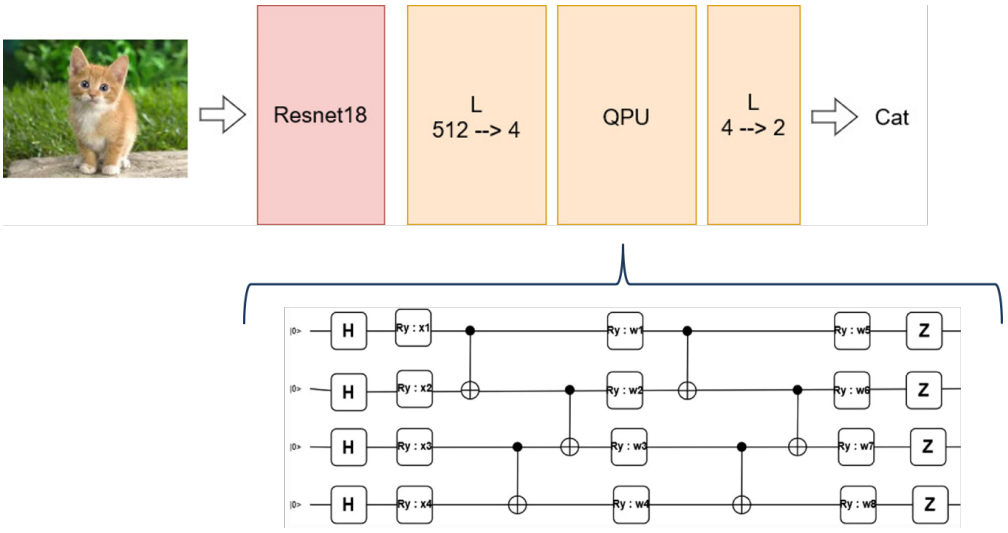


FIGURE 5.2: Quantum transfer learning with Resnet18

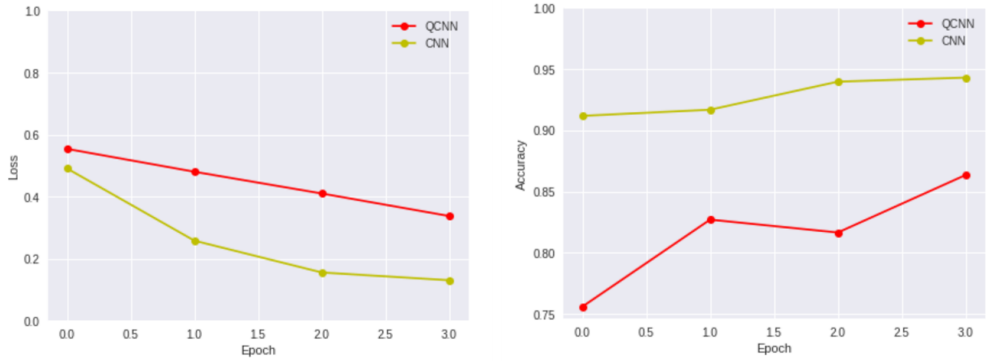
Quantum Random Access Memory(QRAM) Encoding [232]: QRAM encoding represents data by encoding it in a QRAM structure. In this method, data is stored in a quantum state that can be accessed efficiently by a quantum algorithm, and the quantum state is used as the input to a quantum circuit.

And there exist many more quantum encoding methods, for more information, please refer to [230].

5.4 Experiment

5.4.1 Quantum Transfer Learning

Transfer learning is a well-established technique for training artificial neural networks [233], which is based on the general intuition that if a pre-trained network is good at solving a given problem, then, with just a bit of additional training, it can be used to solve a different but related problem. Based on this idea, a hybrid transfer learning which introduces quantum neural layers into classical DNNs is developed by Mari et al. [234]. In their work, a hybrid DL model is described, which consists of using exactly those classical pre-trained models as feature extractors and then post-processsing such features on a quantum computer. Such hybrid approach can process high-resolution images since the quantum computer is applied only to a limited number of abstract features, which is extracted by the pre-trained classical DNN model from the original input. In this experiment, a Resnet-18 [235] is adopted as the pretrained classical DNN model, through which the initial input images from Cifar10 [236] (restricted to the classes of cats and dogs) dataset can be downscaled from 128*128*3 to 4 qubits. After that, a



(A) Comparison of training loss between quantum transfer learning and Resnet18

(B) Comparison of accuracy between quantum transfer learning and Resnet18

FIGURE 5.3: Comparison between the performance of transfer learning and quantum transfer learning based on Resnet18

quantum circuit is applied, which is composed of Hardamard, CNOT, and Ry gates. In the last part of the model, it is a classical dense layer which will outputs the result and give the prediction whether the input image is a cat or dog. The details of the model is depicted in Figure 5.2

The experiment result in Figure 5.3 suggests that the hybrid model which consists of pre-trained DNN model and QDNN layer is capable of classifying highly non-linear dataset. However, it is remarkable that the accuracy of the classification given by the hybrid model does not show any advantages over the classical model. However, this is only a particular example, any general and rigorous comparison would require a much more complex and detailed analysis.

5.4.2 Quanvolutional Neural Network

A quanvolutional neural network is a type of neural network that incorporates the principles of quantum computing to perform convolutional operations, which were first proposed by Henderson et al. [2]. The idea behind Quanvolutional networks is exactly the same as for the classical convolution: to extract relevant features from a localized space in an image. Instead of applying learner filters, quanvolutional neural network adopts a quantum circuit to access a higher-dimensional space. Thus, such QNN consists of two parts: a quanvolutional layer and CNN.

A quanvolutional layer is designed to be able to replace the classical convolution and be stacked on top of the classical layers. The key difference is that quanvolutional filters extract features from input data by transforming spatially-local subsections of data using random quantum circuits [237].

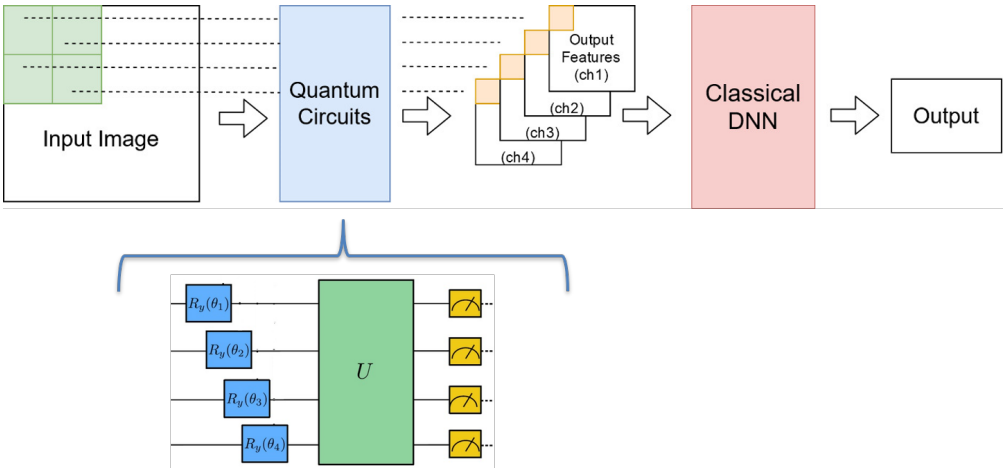


FIGURE 5.4: Quanvolutional neural network, revised from [2]

The workflow of quanvolutional neural network can be divided into 3 main steps: The same as described in Section 5.2, in order to enable quantum computer process the input image, the input classical images must be first encoded into quantum state. Some simple methods can be found in Section 5.3. After the input data transformed into quantum state, the quantum circuit is applied to fully explore the potential of quantum computing. As is shown in Figure 5.4, some parameterizable and trainable unitary gates can be deployed to extract specific features. In the last step, the quantum state is converted back to a scalar output

In the work done by Wilson et al. [238], it is proved that quantum transformations feeding into a linear model could give a performance enhancement over linear models built on the data directly. In this experiment, rather than a linear model, a DNN model is applied after the quanvolutional circuit which will output the enhanced four channel feature map of input. Some sample of enhanced four channel feature map input images are shown in Figure 5.5.

The results of Figure 5.6 extend the scope of quantum features applicability further. However, it is also obvious that the introduction of the quanvolutional layer does not show any advantages over classical DNN when comparing the accuracy of their prediction. Still, it is observed that the involvement of the quantum circuit could accelerate the convergence, since Figure 5.6 B reveals that the QNN model could achieve the same level of training loss with less number of epochs.

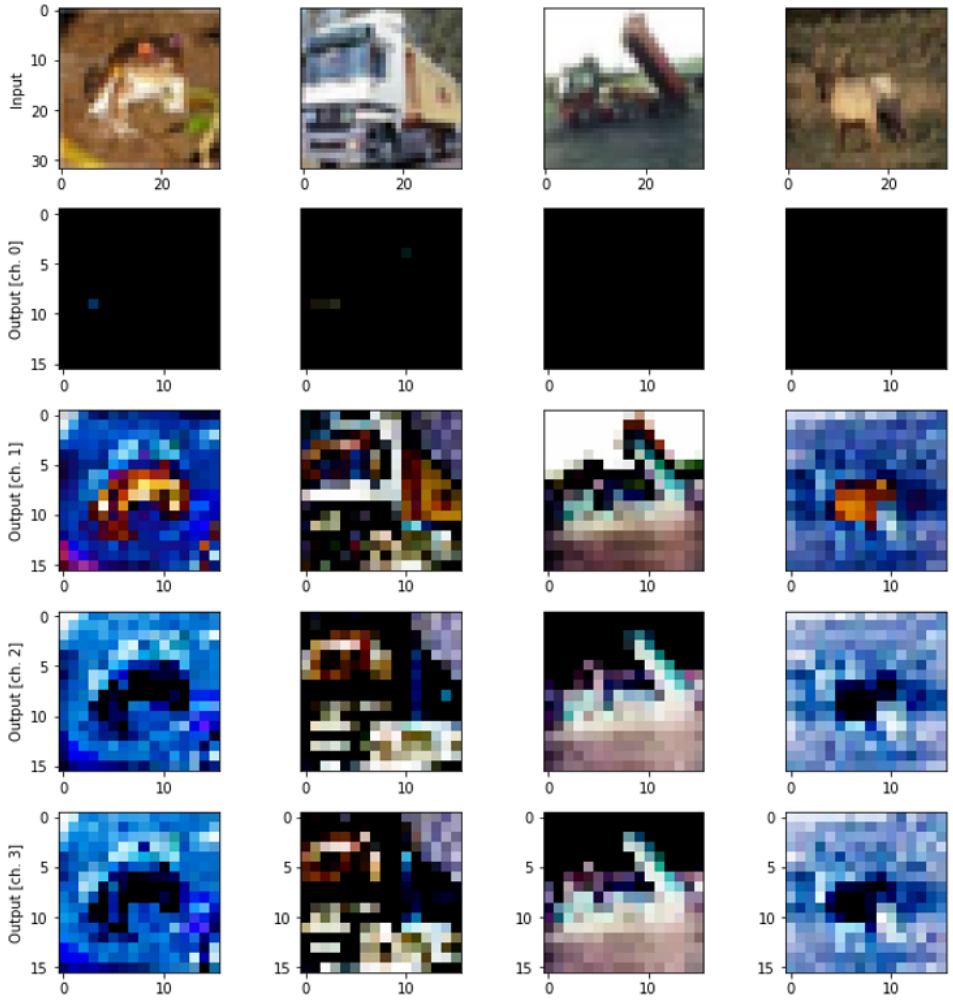
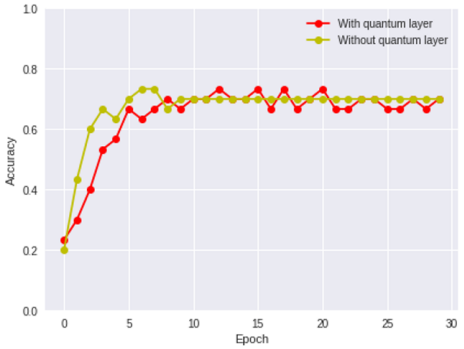
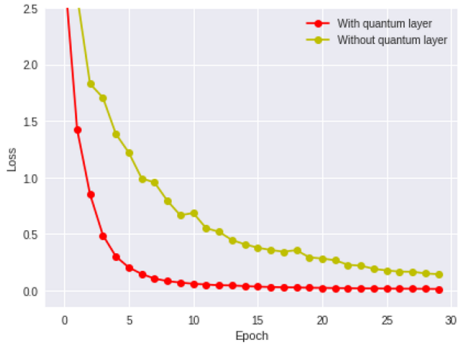


FIGURE 5.5: The output of the quanvolutional layer, which are four channel feature maps



(A) Comparison of accuracy between quanvolutional neural network and classical CNN



(B) Comparison of training loss between quanvolutional neural network and classical CNN

FIGURE 5.6: Comparison between the performance of classical CNN and quanvolutional neural network

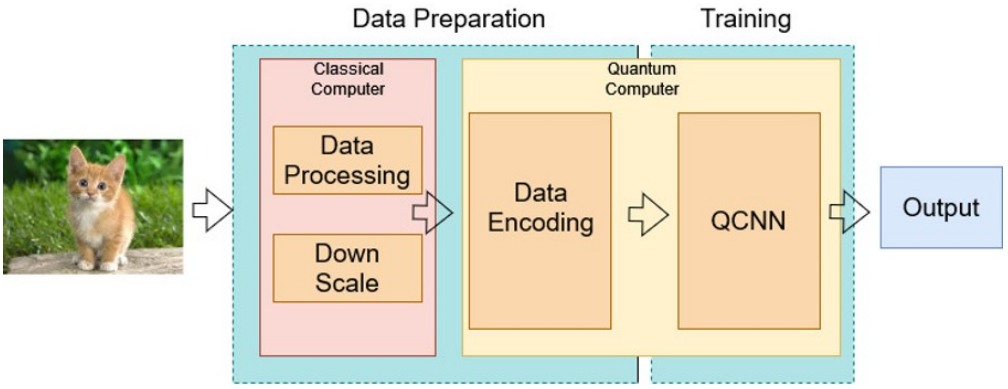


FIGURE 5.7: Workflow of quantum convolution neural network

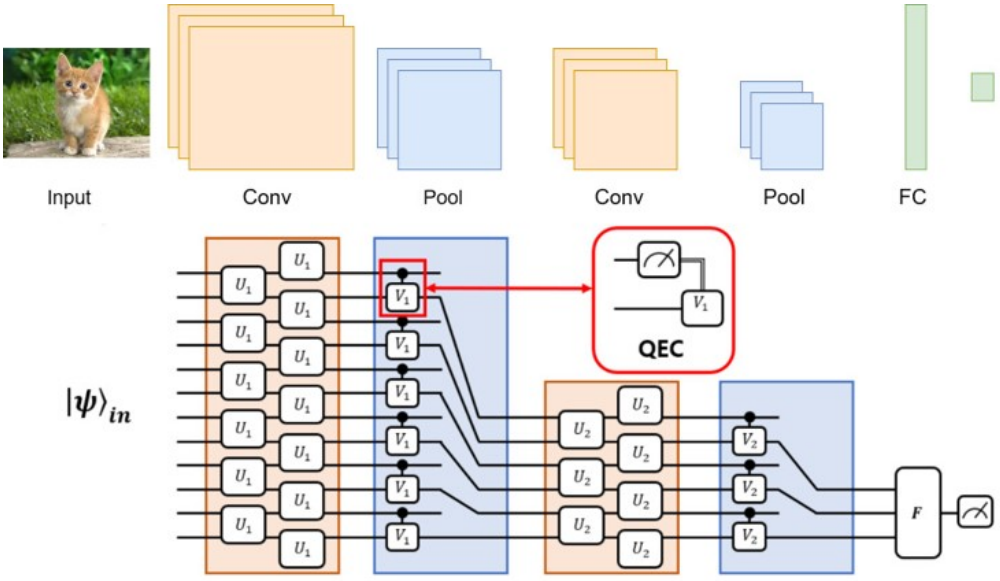
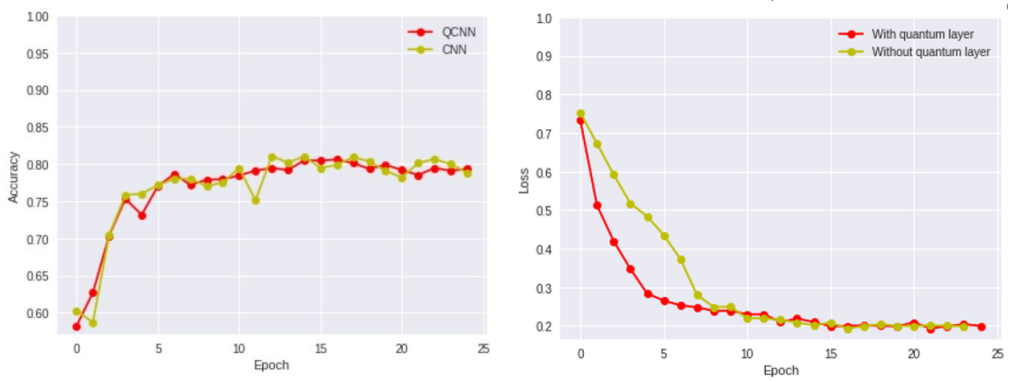


FIGURE 5.8: Quantum convolution neural network, image revised from [3]

5.4.3 Quantum Convolution Neural Network

The above mentioned two QNN models are hybrid, which means that the model is a combination of quantum layers and classical layers. In this section, Quantum Convolutional Neural Networks (QCNN) [239] are described, which is composed of only quantum gates and layers, as is shown in Figure 5.7.

QCNNs are a type of neural network architecture that incorporates the principles of quantum computing to perform convolutional operations on data. In traditional CNNs, convolutional operations are performed using digital computing techniques. However, in QCNNs, these operations are performed using quantum circuits, which involve the manipulation of quantum states. As illustrated in Figure 5.8, the basic building block of a QCNN is a quantum convolution layer and a quantum pooling layer. The quantum convolution layer consists of a quantum circuit that performs the convolution operation on the input data. This circuit typically includes quantum gates, such as the Hadamard



(A) Comparison of accuracy between QCNN and classical CNN

(B) Comparison of training loss between QCNN and classical CNN

FIGURE 5.9: Comparison between the performance of classical QCNN and CNN

gate, CNOT gate, and rotation gates, and uses entanglement to perform the convolution operation in parallel. The input data is encoded into a quantum state, which can be represented as a vector of qubits. The quantum convolution layer applies a series of quantum gates to this state to produce a new quantum state that represents the output of the convolution operation.

To perform the pooling operation, which is used to reduce the size of the feature map. Quantum pooling layers are designed to extract useful information from the data, while reducing the number of qubits that need to be processed in subsequent layers. This can be achieved through a variety of methods, such as entangling qubits or applying specific quantum gates. QCNNs also adopts a quantum activation function, which is used to introduce non-linearity into the network. This activation function is typically based on the threshold gate, which introduces a non-linear transformation to the quantum state.

The model is evaluated by experimenting on the Cifar10 dataset. As is shown in Figure 5.9, it can be drawn out that the QCNN and CNN have almost the same explicit results regarding accuracy. However, it can also be seen that compared to the corresponding classical CNN model, QCNN can achieve the same level of training loss with less epochs.

5.5 Conclusion

In this chapter, three hybrid quantum DNN models, namely the quantum transfer learning (Resnet18 based), the Quanvolutional network, and the quantum convolution network, are evaluated by comparing their performance on the Cifar10 dataset to their corresponding classical DNN models. It is observed that adding a quantum circuit to a model could not significantly improve its accuracy or reduce its loss value. However, it is also proved that with the involvement of quantum circuits, DNN models can

get convergence with fewer epochs compared to the models without quantum circuits, which indicates that the quantum advantage could be obtained at some level without sacrificing the accuracy of DNN models. However, these are only experiments on a specific dataset. Any general and rigorous comparison would require a more complex and detailed analysis.

Looking into the future, developing more application-specific quantum network models is important. Better ansatzes shall be designed to enhance gradient magnitudes to improve trainability and enable promising performance for a specific problem, *e.g.,* image processing, *etc.* Hybrid quantum-classical models are a natural extension of VQMLs where one parameterizes both a classical and quantum ansatz, and such models could also facilitate near-term applications [240]. In addition, analytical and heuristic scaling analysis of VQML, especially VQNNs will be increasingly important, and better methods of scalability to show the quantum advantage are anticipated in the future.

Chapter 6

TransGAN: A Transformer-GAN Based Model for Image Compression

A novel DNN model based on GAN and Transformer for data lossy compression is presented in this chapter. Since the amount of data is increasing faster than ever, new and better compression methods must be researched and implemented continuously. Further, due to the limitation of current quantum technology, a more effective compression method is needed to reduce the input data dimension while retaining sufficient information to fit the limited number of the qubits of current quantum computers. Therefore, a novel DNN model based on Transformer and GAN is developed, which takes both advantages of the Transformer and GAN to obtain visually pleasing results at bitrates. The approach is tested on the dataset Cifar10 and discussed training strategies for the proposed architecture to achieve the best compression and best performance, including the loss function, quantization, number of residual blocks, *etc.* This proposed method is then evaluated on the benchmark dataset Cityscape [241] to compare with other data compression methods based on compression ratio, computational resources, and quality of the rendered images. The results prove that this method has advantages in computational resources and recovered image quality in the range of extreme compression.

Chapter outline: This chapter is structured in 5 sections. Section 6.1 introduces the problem and explains the contributions. This is followed by the review of related work in Section 6.2. Section 6.3 explains the proposed method, which is followed by the experiment and result analysis in Section 6.4. In Section 6.5, the concluding remarks are provided.

6.1 Introduction

Due to the rapid progress in digitalization, more and more data are generated yearly. Between 2016 and 2021 alone, the amount of data processed via the Internet increased almost by a factor of three [242]. This leads to heavy pressure on storage space, computing power and network bandwidth, as they can grow at different rates. Further, due to the limitation of current quantum technology, a more effective compression method is needed to reduce the input data dimension while retaining sufficient information to fit the limited number of the qubits of current quantum computers. As a result, data compression, especially image compression, is becoming increasingly important to diminish the size of the files required to be stored, transmitted, and processed.

Traditional compression algorithms, *e.g.,* JPEG, JPEG2000, WebP, *etc.* rely on handcrafted encoder/decoder pairs(codecs), which lack adaptability and have poor performance against complex textures and unexpected signals. Furthermore, since they are designed to faithfully reproduce signals based on the peak signal-to-noise ratio (PSNR), they suffer from visual artefacts such as blurring, ringing, and blocking at low bit rates [243]. To overcome this obstacle, new methods are continuously being researched. One of these new approaches is to use CNNs with parts of traditional compression algorithms. There exist several methods based on CNNs perform end-to-end optimization using flexible nonlinear analysis by CNN-based autoencoders, and it has been reported that they show higher performance than conventional image compression systems. However, CNNs do have some limitations. For example, the convolutional filter can only characterize short-range spatial correlation within the receptive field. At the same time, the global positional information is also significant, which is very difficult for CNNs to learn. Some recent work [244, 245] proved that doing so may potentially constrain the image reconstruction performance since the stacked convolutional layers have natural constraints on the effectiveness of the receptive field and over-parameterized redundant filter issues.

To overcome these drawbacks, methods using GANs have been developed to improve subjective image quality. Motivated by the great success that GANs have achieved in image processing, several GAN-based methods have been proposed for image/video compression. And these methods have been shown to outperform traditional codecs in various aspects [246]. GANs can produce high-quality compressed images that closely resemble the original images. This is because GANs learn to generate images that are visually similar to the training images, which helps in maintaining the quality of the compressed image [247]. In addition, GANs can achieve variable compression, which means that the compression rate can be adjusted to achieve a desired trade-off between image quality and compression ratio. This is particularly useful in scenarios where the available bandwidth is limited or fast image transfer is required [7].

However, these GAN-based methods are mostly centred around CNN and still suffer from its inbuilt disadvantages. In recent work, Transformers, which excel in NLP, have been shown to perform better than CNN on a variety of computer vision tasks. Moreover, compared to CNN, Transformers have several advantages [33]. Transformer models can attend to specific parts of an image using a self-attention mechanism, which helps the model to focus on the important features of an image. Transformers are permutation-invariant, which means they are able to handle input sequences of different lengths, making them well-suited for image processing tasks where the size of the image can vary. In addition, Transformer models can better handle long-range dependencies, which is useful when the relationship between pixels in the image is complex. Moreover, it can handle inputs of varying sizes, thus eliminating the need for many customized building blocks common in CNN-based pipelines.

Although the Transformer has shown multi advantages in computer vision tasks, it is mainly used for tasks like image classification. No studies have been done to explore the possibility of using GAN and Transformer in combination to do image compression. Therefore, this chapter describes an image compression method which is based on a combined mechanism of Transformer, GAN, and conventional compression methods. The model takes both advantages of Transformer and GAN to gain high-performance reconstruction for images and obtain visually pleasing results at bitrates. The experimental results demonstrate that the proposed method obtains superior reconstruction quality and noise robustness compared with other methods.

Contributions: The main contributions of this chapter are the following:

- A novel Transformer-GAN based model is proposed for image compression
- Experiments on the Cifar10 and Cityscape dataset are conducted to show the effectiveness of the proposed method
- The effects of compression for the proposed method and other methods are compared and analyzed

6.2 Related Work

Classical methods: The majority of compression methods are composed of similar stages, which include color transformations, block splitting, scalar quantization, and coding [248]. The JPEG (Joint Photographic Experts Group) [249] method is a widely used lossy compression technique for digital images. It uses a discrete cosine transform (DCT) to transform the image into frequency components, which are then quantized

to reduce the amount of data. The compressed image is stored as a series of blocks. JPEG2000 [250] is an image compression standard that was developed as an improvement over the original JPEG method. It uses a discrete wavelet transform(DWT) to transform the image into frequency components, which are then quantized and compressed. JPEG2000 offers higher compression ratios than JPEG, while maintaining better image quality. WebP codec [251] is a image compression format developed by Google, which uses a combination of techniques, including prediction, variable-length coding, and a lossless compression method called WebP-Lossless. WebP offers high compression ratios and good image quality, and is especially useful for web-based applications where image loading times are critical.

DNN based methods: With the concept of data compression in combination with DL, several models have been developed in recent years. One of the more successful models are compression models based on GAN architectures [247, 252, 253]. Especially for extremely low bit rates (below 0.1 bpp) these models can be used successfully [246]. Agustsson et al. [7] trained a GAN to generate compressed images, which were then decompressed using a decoder network. The results showed that their approach outperformed existing image compression methods in terms of image quality and compression ratio. Song et al. [254] demonstrated the use of unified binary GAN (BGAN+) for image compression and image retrieval. The result exhibited that BGAN+ can achieve better visual quality of the reconstructed image than JPEG and JPEG-2000 at low bit-rate (0.15 bpp).

6.3 Methodology

The proposed method adopts the AutoEncoder architecture. For the Encoder part, it starts with the preparation of the images. Since words can be represented as one-dimensional tokens while images are two-dimensional vectors, directly processing images with Transformer can lead to problems due to the complexity of the attention, which is $O(n^2 * d)$ with d as representation dimension and n as the sequence length. This ends up in large matrices with bigger sequence lengths n by bigger two-dimensional inputs [33]. These large matrices then require enormous computing power which is expensive and inefficient. To solve this problem, the input image must be processed to meet the requirement. One method for this is to tokenize the image into a projected representation, through which, the image is firstly divided to small parts or patches. These patches are then labelled and paired with positional information [255, 256].

For extracting patches, the original image x with the resolution H(Height), W(Width) and the number of channels(C) must be transformed into the flattened extracted patches

$$\begin{gathered} x_p \in \mathbb{R}^{N*(p^2*C)} \leftarrow x \in \mathbb{R}^{H*W*C} \\ N = HW/P^2 \end{gathered} \tag{6.1}$$

The number of patches N can be calculated through H, W, and the resolution of the Patches P. To connect the shaped patches with the positional embeddings. For this purpose, for each patch a suitable positional information must be created and added to the patch. Firstly, the number of patches is determined by:

$$Num_{Patches} = (Size_{Image}//Size_{Pathes})^2 \tag{6.2}$$

The following operation is the embedding of the patches and the addition of the embeddings to the original patches. After this preparation, the dimension of the vector is reduced and the transformer layer is enabled to process the image[257].

Motivated by the work done by Wu et al. [258], a connection of convolutions and transformer architectures is introduced in the proposed method. Since convolutions are especially good at identifying and extracting low level features, such as corners and edges, but they are usually overwhelmed by spatially-distant concepts. This is where transformer architectures have their strengths [256]. After forming the patches, transformer layer is connected before and after to ensure best performance [259]. It uses the first step of the previous vision transformer [258], in addition it uses image segmentation for a bigger variety of data [260]. With this connection, a small number of visual tokens can then be sufficient to describe high-level concepts in an image. These tokens can then be reused as a representation or in the form of an attention map [259]. After an attention map is created, the tensor can be further processed into a token representation.

The transformer layer consists of a combination of normalizations, multi-head attention modules, and multilayer perceptron [33]. Multi-head attention is an attention mechanism module that passes an attention mechanism several times in parallel. The independent attentional outputs are then linked and linearly transformed into the expected dimension. Intuitively, multiple attention heads allow for different treatment of parts of the sequence. The learnable parameters Query(Q), Key(K), and Value(V) are represented in the parameter matrices W as

$$\begin{gathered} Multihead(Q, K, V) = [head_0, head_1, ..., head_h]W_0 \\ head_i = Attention(QW_i^Q, KW_iQ, W_i^K, VW_i^V,) \end{gathered} \tag{6.3}$$

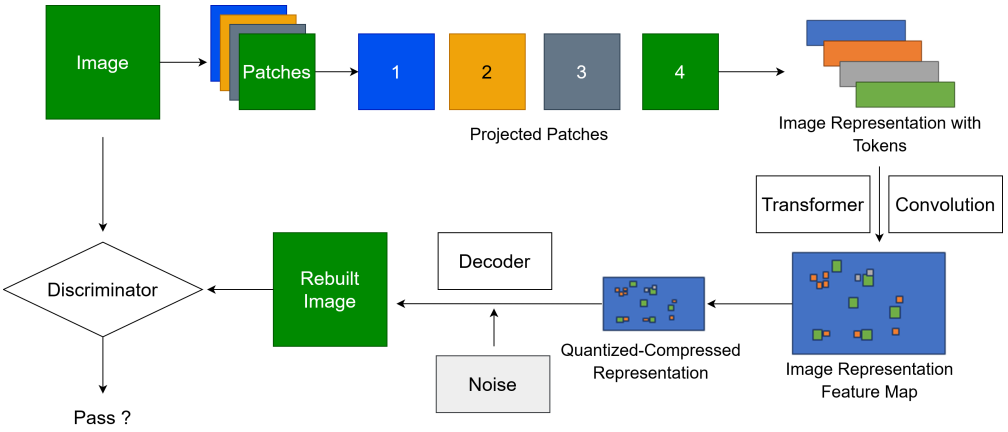


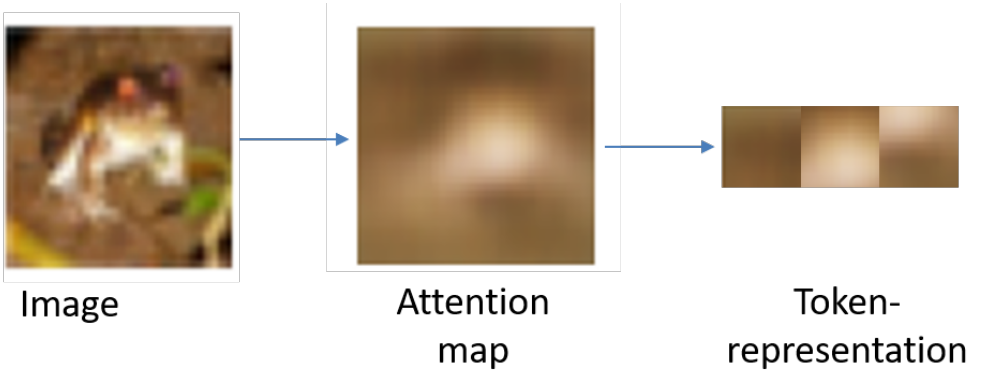
FIGURE 6.1: The workflow of the proposed TransGAN model

Image representation now takes place in between of a series of transformer operations. For this, an attention map is formed, which is then converted into a token representation. After this step, the encoding process is complete, and the elements representing the image have been significantly reduced. At this point, conventional compression methods, such as vector quantization [261], can also be applied.

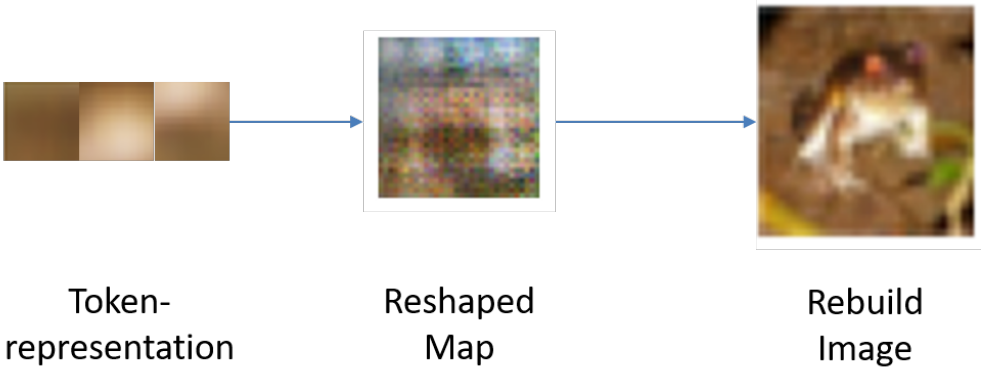
To rebuild the image from the compressed representation, a decoder is applied. The first step for the initial decoding is to get a form from which the ground state can be restored. This consists of transposed convolutional and normalization layers [7]. In addition, residual blocks are inserted, which improve the performance of the model and make it more robust [261]. These consist of weight layer with relu activation which feeds information x into the next layer as well as into the layer two or three steps further, to improve performance and robustness. In some methods, stochastic noise is added to the feature maps. Since when the tensor offseted with trainable, stochastic noise, it provides a greater flexibility and can increase the performance [32]. After the performance enhancing residual blocks, the image can be rebuilt from the representation with a series of transposed convolutions.

The last part of this method is the discriminator, which is used to distinguish whether the image receives, is a real one or one reproduced by the generator and gives feedback accordingly [32]. In the proposed model, convolutions and normalizations are used, which have a dense or convolution layer with only one filter at the end, adopted from the idea introduced by Agustsson et al. [7].

The whole workflow is illustrated in Figure 6.1 and the configuration of the designed model can be found in Appendix A.



(A) Encoding



(B) Decoding

FIGURE 6.2: Data encoding and decoding process

6.4 Experiment and Analysis

6.4.1 Experiment Settings

The proposed method is implemented on top of Tensorflow and evaluated on the Cifar10 and Cityscapes dataset. The Cityscapes dataset consists of 2975 annotated images for training, 500 annotated images for validation and 1525 annotated images for testing.

To evaluate the presented approach, several metrics have been adopted. Peak signal-to-noise ratio (PSNR), which is the pixel-wise MSE between the original and the reconstructed image in decibel. Thus, this metric is very sensitive to variations in single image pixels. Another metric to be used is the structural similarity (SSIM), which aims at predicting the human-perceived image quality. It compares the luminance, the contrast, and the structure of the original and the reconstructed image. Another metric adopted is the bits per pixel (bpp) to measure the bit rate, which refers to the sum of the "number of bits per color channel" *i.e.,* the total number of bits required to code the color information of the pixel. For comparison, the image compression standards JPEG has been used as the baseline.

The experiment is conducted on the HPC system Vulcan, for more technical specifications of this system, please refer to Section 2.2.5.

6.4.2 Result and Analysis

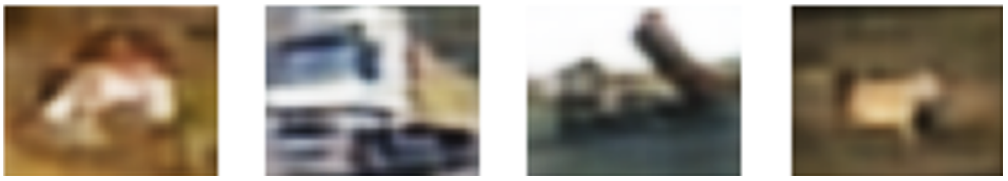
This section describes the experimental results and hyperparameter tuning of the described architecture. The parameters that will be examined are whether it makes sense to add noise, what the optional quantization means in terms of system performance, and what number of residual blocks and tokens are optimal. The model is measured according to the parameters, quality of the recovered image in the form of SSIM method. In addition, the computational resources are compared with each other by comparing the number of elements between the ground tensor and the image representation tensor. The basic settings for experimental training are 50000 training images with the size (32*32*3). The architecture is trained over 100 epochs with the Adam Optimizer.

Noise The first experiment is to test whether to add stochastic noise to the token representation and whether this can improve the restored image quality. When adding stochastic noise to an input of a GAN, it can have a variety of effects on the output image, such as potentially introducing new variations into the generated images, making them less predictable and potentially more diverse. In some cases, adding stochastic noise to the input of a GAN can improve the stability and convergence of the training process, as it can help to break symmetry and prevent the generator from getting stuck in a local minimum. However, if the added noise is too strong, it may cause the generated images to become distorted or unrealistic. One common way to add stochastic noise to the input of a GAN is through the use of a random noise vector, which is concatenated with the input image and then fed into the generator network.

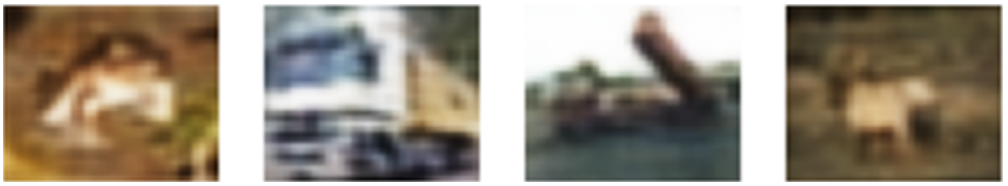
The first image sequence shown in Figure 6.3 is the base image, the second row is the generated images without noise and the bottom row is the generated images with noise added. By comparing the SSIM and PSNR values in the Table 6.2, it is evident that the variability introduced by the addition of stochastic noise greatly improves the quality without creating disadvantages in the other categories.

Quantization Quantization is the process of reducing the number of bits used to represent the values of an input vector in GAN, which can have an impact on the quality of the generated images and the stability of the training process. Although quantization can reduce the accuracy of the model and increase the level of noise in the generated images, quantization can also have some benefits, such as reducing the memory and computational requirements of the GAN. This can make the GAN more efficient to train and deploy.

(A) Original images



(B) Rebuilt image with noise added



(C) Rebuilt image with noise added

FIGURE 6.3: Comparison of the rebuilt image with and without adding stochastic noise to the iuput

As is shown in Figure 6.4, the first series of images are the original images, the images in the row below are quantized input vectors without added noise. and the images in the last row are quantized input vectors with added noise. By comparing the SSIM and PSNR values in the Table 6.2, it is obvious that that quantization causes a strong degradation of the reproduced images. Also, it is observed that in this case the addition of noise causes a deterioration of the quality with otherwise unchanged parameters.

Residual Blocks Residual blocks can be used to improve the quality of the generated images by allowing for the creation of deeper generator networks, which have the potential to capture more complex relationships in the data, leading to more detailed and realistic generated images. Residual blocks can also help alleviate the vanishing gradients problem, which can occur when training deep networks, by allowing the gradients to flow more easily through the network. Additionally, residual blocks can also be used in the discriminator network to improve its ability to distinguish between real and reproduced images. This is because residual blocks can allow for the creation of deeper discriminator networks, which can capture more nuanced and complex features in the images.

In this experiment, the influence of different numbers of residual blocks on the final rebuilt image is studied. The first row of the displayed images in Figure 6.5 are the baseline images, the rows below are generated images with one, three and six residual

(A) Original images

(B) Quantization without noise

(C) Quantization with noise

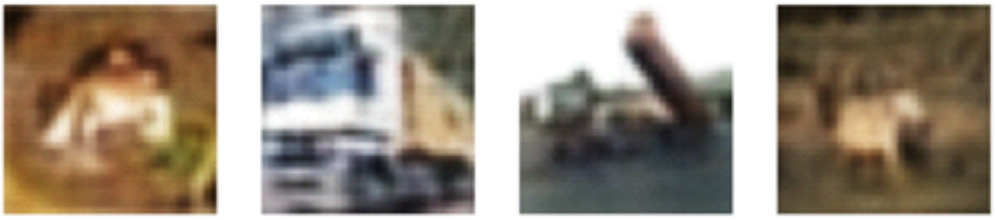
FIGURE 6.4: The effect of quantization with and without adding noise

blocks. Table 6.2 describes that the highest quality performance is obtained with the quantity of three residual blocks. These also consume a considerably shorter time compared to six or nine blocks. The lowest calculation power is needed with only one block, but this is at the expense of the quality. Based on this result, the optimal number of residual blocks for dataset cifar10 is three.

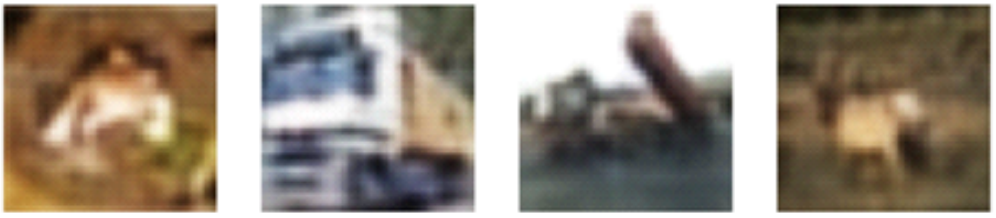
Tokens The number of tokens used in a Transformer and a GAN can influence the quality of the generated images. For a Transformer, the number of tokens can impact its capacity to represent and generate complex relationships in the data. A larger number of tokens can allow for the creation of a more powerful model, which can better capture the structure and patterns in the data. However, a larger number of tokens can also increase the computational and memory requirements of the model, making it more challenging to train and deploy. For a GAN, a larger number of tokens can allow for the creation of a more powerful generator, which can capture more complex patterns in the data and generate more detailed images. Similarly, a larger number of tokens in the discriminator can allow to better distinguish between real and fake images by capturing more nuanced and complex features.

In this experiment, the influence of different numbers of tokens on the final rebuilt image

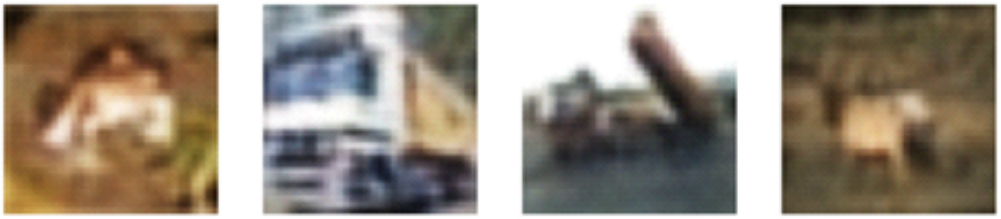
(A) Original images



(B) 1 Residual Block: SSIM 93.898



(C) 3 Residual Block: SSIM 94.902



(D) 6 Residual Block: SSIM 93.862

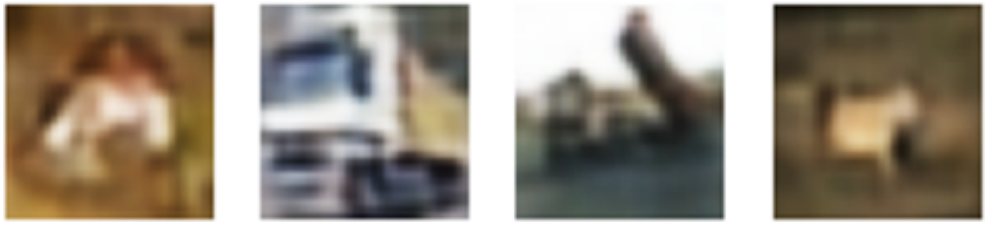
FIGURE 6.5: Comparison of the proposed method with different residual blocks

is researched. The first row of the displayed images in Figure 6.6 are the baseline images, the rows below are generated images with 8, 2 and 1 tokens. The representation with 8 tokens is almost indistinguishable from the original image to the naked eye and with an SSIM value of 97.932% coming closest to the maximum value in this experiment as seen in Table 6.2. However, this is at the expense of the compression ratio, which is three, the lowest of all the variations examined in this experiment. it is also noticed that a downward trend of the quality with higher compression, but despite a strongly increased compression rate the quality did not decrease to the same extent.

The proposed model is furthered evaluated with the dataset Cityscape [262]. This provides a good basis for comparison with other compression methods, since it is more detailed and has a larger data volume. The size it is mapped with is (256*256*3) and it is trained with a number of 2688 images.

(A) Original images

(B) 8 tokens: SSIM 97.932 bpp 0.33333



(C) 2 tokens:SSIM 89.083 bpp 0.08333

(D) 1 token:SSIM 86.361 bpp 0.04166

FIGURE 6.6: Comparison of quantity representation with different numbers of tokens

TABLE 6.1: Performance of the predicted model with different model configurations

Characteristic Features	Quality SSIM (%)	Time per epoch (s)	Compression rate (bpp)
No Noise	86..099	189	0.16666
Noise	93.861	190	0.16666
Quantized without noise	43.339	191	0.16666
Quantized with noise	28.454	197	0.16666
1 Residual Block	93.898	90	0.16666
3 Residual Blocks	94.902	120	0.16666
6 Residual Blocks	94.862	156	0.16666
8 Tokens	97.932	118	0.33333
2 Tokens	89.083	118	0.08333
1 Token	86.361	118	0.04166

TABLE 6.2: Comparison of the performance of the proposed model, Deep Generative Model [7], and JPEG

Characteristic Features	Quality SSIM (%)	Compression rate (bpp)
Proposed model with 32 tokens	53.178	0.0208
Deep Generative Model [7]	10	0.1
JPEG(4:2:0)	<10	0.3

The rebuilt image from the proposed model is compared with two other commonly used methods: the conventional JPEG [263] method and a lossy method based on an DNN codec principle presented by Minnen et al. [264], which has been widely used to do comparative study in other generative compression models [7].

In order to fit the proposed method into different dataset, several changes have been done: 1. Additional upsample layers are added to the decoder to get to the desired size. 2. Data papallelism has been adopted to work efficiently in HPC. 3. In addition, the loss functions must be equipped with a reduction. Based on the experiments, a variant must be found that is well suited for a comparison with the other methods. To get a similar bpp rate two token numbers are experimented, 64 tokens would give a bpp value of 0.04166 and 32 would give a value of 0.0208.

Since this network is primarily designed for extreme compression, compression is firstly tested with 32 tokens and trained on 50 epochs following the strategy described in [7]. And quality of the rebuilt image, compression rate, and computational resources are measured.

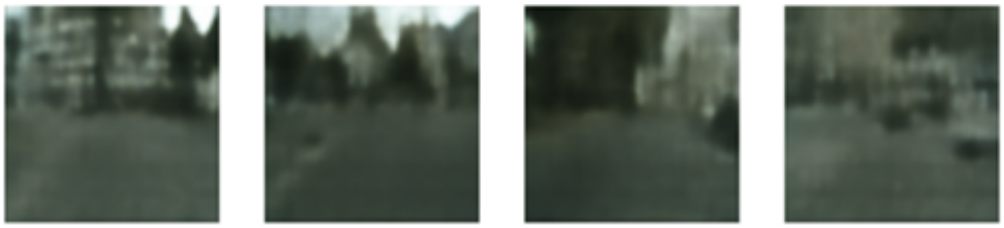
It can be seen from Figure 6.7 that the proposed method has a good performance in extreme compression. Especially in the SSIM values, the comparison in Table 6.2 shows that this model possesses advantages over the model proposed in [7] and JPEG. The comparison with JPEG makes little sense in these areas of extreme compression since the performance decreases significantly from a compression rate of 10 or 0.5 bpp

6.5 Conclusion and Discussion

The Transformer and GAN architectures are steadily gaining popularity in image compression due to their performance. More and more research has been done to efficiently take advantage of the Transformer and GAN structure. However, work has yet to be done before to develop a model which combines the GAN and Transformer for image compression. This work has developed a model that inserts Transformer into a GAN architecture, which creates an image compression codec by combining Transformers and

(A) Original image

(B) SSIM 71.176 bpp 0.0833



(C) SSIM 65.784 bpp 0.0417

(D) SSIM 53.178 bpp 0.0208

FIGURE 6.7: Comparison between the original and rebuilt image with different model configurations

convolutions. The encoder is implemented by preparing the image information and applying it to Transformer architectures. Afterwards, the image information is processed into an attention map and further into a compressed token representation. Experiments were also conducted to investigate if conventional compression-type vector quantization could be employed to improve the performance. The decoder is consisted of a set of convolutions to rebuild the image. And the discriminator is well designed to distinguish between original and reconstructed images. The result has proven that it is possible to rebuild the images into a token representation and to restore them almost perfectly to their original state. Overall, the architecture has advantages, especially in extreme data compression.

Looking into the future, further optimization in model training could be done, especially

in parallelization. In addition, different compression rates would be worthwhile to investigate since it is presented in the experimental chapter how to change certain metrics for the desired result. Conventional methods, such as quantization, could also be better integrated into the system for better compression. For future appliances, bringing more and more Transformer and attention modules into the model will be the way to go, as they prove to have the potential to relieve bandwidth and make data traffic more efficient.

Chapter 7

Quantum Neural Network for Solving Partial Differential Equations

In this chapter, a well-designed Quantum Neural Network (QNN) is introduced, which uses automatic differentiation to solve partial differential equations (PDEs) on quantum computers (QC). The design of QNN allows for its efficient training and implementation on realistic, near-term quantum devices. Its performance is investigated on two practical problems to explicitly demonstrate its capabilities: 1D Burgers' equation and 2D Poisson equation. The potential experimental realizations and generalizations of QNN are also discussed.

Chapter outline: This chapter is organized as follows: Section 7.1 provides the introduction to the problem and explains the contributions. In Section 7.2, the state of the art of DNN solver for PDEs will be briefly reviewed. The corresponding methods and algorithms of QNN will be explained in Section 7.3 and 7.4. The QNN solution to two specific problems, namely Burger's equation and Poisson equation, will be discussed in Section 7.5. The experimental setup and corresponding result will be discussed in detail in this section. The last section summarizes the results, and an outline for future work will be presented.

7.1 Introduction

In the field of science and engineering, various mathematical models have been developed to describe physical phenomena. Many of these models are naturally expressed in

differential equation form, mostly as time partial differential equations. Solving these differential equations is of enormous importance for problems in all numerical disciplines, such as weather prediction, astronomical simulations, molecular modelling, or jet engine design.

However, solving PDEs numerically is difficult and has long been a computational challenge. Solving most of the important equations is analytically difficult and usually based on numerical approximation methods. Hand-crafted solvers are required and always tailored to the equation at hand to obtain accurate solutions to bounded errors with minimal computational effort. Recently, research has been conducted on DNN-based solvers, where DNNs are trained to satisfy the differential operator, initial condition and boundary conditions using stochastic gradient descent on randomly sampled spatial points. This approach transforms the PDE solution problem into a machine learning problem, usually data-driven rather than numerical. Therefore, training deep neural networks is computationally intensive and requires a lot of storage and communication. Exciting new opportunities are emerging due to the imminent advent of quantum computing devices that directly exploit the laws of quantum mechanics to circumvent the technological and thermodynamic limitations of classical computation.

Therefore, a novel method based on QNN is proposed to solve the partial differential equations, which enables fast optimization with reduced memory requirements. The designed QNN is inspired by the Quantum Convolutional Network (QCNN) [239], where a convolutional layer applies a single quasi-local unit in a translation-invariant manner for finite depth, and a pooling layer applies a fraction of qubits. Convolution and pooling layers are performed until the system size is sufficiently small. And the circuit result is obtained by measuring a fixed number of output qubits, which are trained by learning the data generated on HPC systems.

The main contributions of this work are as follows:

- A QNN is proposed and developed for solving PDEs, which no previous study, to the best of author's knowledge and through search in peer-reviewed databases, has done before.
- The effectiveness of the QNN is demonstrated by investigating its performance on two practical problems: 1D Burgers' equation and 2D Poisson equation.
- The success of QNN in this work opens up a promising paradigm by complementing the physics-based DNN model with an emerging new development of quantum computing, as well as providing a vivid reference for fusing QC and DL algorithms, which can serve as the basis for further work towards quantum PDE solvers.

7.2 Related Work

7.2.1 DNN Solver for PDE

In recent years, there has been growing interest in using DNNs to solve PDEs due to their ability to approximate complex nonlinear functions. The idea is to use DNNs to represent the solution to a PDE as a mapping from the spatial and temporal domains to the solution space. This representation can be trained using data generated from either numerical simulations or experimental observations. Yu et al. [265] proposed the DNN-based Deep Ritz Method(DRM) to approximate the solutions of PDEs by minimizing the energy function associated with the PDE. Li et al. [266] described the DNN-based Neural Operator Method that approximates the solution of PDEs by learning the underlying differential operator. Another work is to use the GAN to solve PDE [267], which is composed of a generator and a discriminator. The generator is trained to produce solutions that are consistent with the governing equation, while the discriminator is trained to identify solutions that are inconsistent with the equation. A recent work [268] demonstrates that U-Net, which is a CNN-based model, can be trained using flow fields computed by computational fluid dynamics (CFD) simulations to approximate the solution of PDEs. Rassi et al. [269, 270] introduced the physics-informed neural network (PINN), which enforce physical constraints during the training process for inferring latent solution to a PDE system. [268]

7.2.2 QNN

Quantum neural networks are a subclass of variational quantum algorithms comprising of quantum circuits that contain parameterized gate operations [271]. In QNN, a variational model containing parameterized gates is applied and optimized for a particular task [272–274]. Farhi et al. [275] defined the formalism of a gate-model quantum neural network to classify the classical data sets consisting of bitstrings with binary labels. Wiebe et al. [276] found that the application of quantum computing can reduce the time required to train a deep-restricted Boltzmann machine and lead to significant performance improvements in comparison to classical computing. Kwok et al. [277] defined a quantum generalization of feedforward neural networks, which can be trained efficiently using gradient descent to perform quantum generalizations of classical tasks. Cong et al. [239] proposed a quantum convolutional neural network(QCNN) which combines the multi-scale entanglement renormalization ansatz and quantum error correction. The QCNN only makes use of $O(log(N))$ variational parameters for input sizes of N qubits, allowing for its efficient training and implementation on realistic, near-term quantum

devices. Beer et al. [20] proposed a truly quantum analogue of classical neurons, which form quantum feedforward neural networks capable of universal quantum computation. Abbas et al. [21] demonstrated numerically that a class of quantum neural networks is able to achieve a considerably better effective dimension than comparable feedforward networks and train faster, suggesting an advantage for quantum machine learning.

7.3 Mathematical Preliminaries

7.3.1 Data Driven PDE Solver

The general form of a PDE system can be given as:

$$\frac{\partial^k u}{\partial t^t}(x,t) + D[u(x,t);\lambda] = 0,\ x \in \Omega,\ t \in [0,T] \tag{7.1}$$

where the spatial domain $\Omega \in \mathbb{R}^d$, D is the nonlinear operator parameterized by λ. The PDE system is subject to the boundary condition:

$$B[u(x,t)] = h(x,t),\ x \in \partial\Omega \tag{7.2}$$

where where $\partial\Omega$ is the domain boundary, B is the differential operator for boundary conditions, $h(x,t) : \mathbb{R}^{(d+1)} \to \mathbb{R}$ is the given function and $u(x,t)$ is the latent solution of this PDE system.

To solve the equation through DNN, this equation can be rewritten as

$$f(u;t,x) = \frac{\partial^k u}{\partial t^t}(x,t) + D[u] = 0 \tag{7.3}$$

where $u = u_\theta(t,x)$; this gives the a physics-informed neural network parameterized by θ

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_u + \mathcal{L}_f, \\ \mathcal{L}_u &= \frac{1}{N_u}\sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i))|^2, \\ \mathcal{L}_f &= \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i, x_f^i))|^2 \end{aligned} \tag{7.4}$$

$\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ denote the initial and boundary training data on $u(t,x)$, $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ specify the collocation points for $f(u;t,x)$, the solution of Equation 7.4 can be obtained by employing a DNN as an approximated solution, *i.e.*, $\tilde{u} = u_{NN}(\vec{x};\vec{w^*},\vec{b^*})$, where u_{NN} is a neural network output with respective weights and biases $\vec{w}, \vec{u}$. Then the target is to

find:

$$\tilde{u} = u_{NN}(\vec{x}; \vec{w^*}, \vec{b^*}) \tag{7.5}$$

such that

$$\vec{w^*}, \vec{b^*} = \arg\min(\mathcal{L}(\vec{w}, \vec{b})) \tag{7.6}$$

7.3.2 QNN

A QNN implemented on a gate-model quantum computer with a quantum gate structure usually contains quantum links between the unitaries and classical links for the propagation of classical side information. The classical information is encoded into quantum information, and then the quantum information is propagated forward to the output.

A QNN can be formulated as a collection of N unitary gates, and the i-th unitary gate is defined as:

$$U_i(\theta_i) = exp(-i\theta_i P) \tag{7.7}$$

where P is a generalized Pauli operator formulated by a tensor product of Pauli operators X, Y, Z, while θ is the gate parameter. Therefore, the QNN can be defined as a sequence of unitary operators:

$$U(\vec{\theta}) = \prod_{i=1}^{L} U^i(\theta_i) \tag{7.8}$$

where $\vec{\theta}$ represents the gate parameter vectors. Acting the unitary gates on a particular input $|\psi, \phi\rangle$, the output is

$$|Y\rangle = U(\vec{\theta}) |\psi, \phi\rangle \tag{7.9}$$

The goal is, therefore, to find the optimal gate parameters $\vec{\theta^*}$:

$$\vec{\theta^*} = \arg \min_{\theta \in \mathcal{C}} \mathcal{L}(\vec{\theta}, \mathcal{F}) \tag{7.10}$$

where $\mathcal{C} \subseteq \mathbb{R}^d$ is a constraint set, $\mathcal{L}(\vec{\theta}, \mathcal{F})$ is the loss and $\mathcal{F}$ refers to the objective function. In the simplest form of the gradient-descent method, the parameters are updated according to the following:

$$\theta := \theta - \eta \bigtriangledown_\theta \mathcal{L}(\psi, \phi, \theta) \tag{7.11}$$

where η is the learning rate.

7.4 Method

Data Encoding In order to apply quantum algorithms and process the data in quantum computers, it is necessary to transform the classical data into a quantum representation at first. Thus, feature maps are adopted to represent the features of the data in a quantum state. A feature map can be thought of as a mapping from the original data to a new space where the features of the data can be more easily manipulated so that quantum algorithms can perform operations more efficiently.

In order to embed n-dimensional classical data on n qubits, the unitary gates introduced in [278] are adopted, which is defined as:

$$\begin{aligned} F_{\phi(\vec{x})} &= exp(i \sum_{S \subseteq [n]} \phi_S(\vec{x}) \prod_{i \in S} Z_i), \\ \phi_i(x) &= x_i, \\ \phi_{i,j}(s) &= (\pi - x_0)(\pi - x_1) \end{aligned} \tag{7.12}$$

the quantum circuits illustrated in Table 7.1 is utilized, which encodes 2-dimensional classical information into quantum state. First, the featuremap applies Hadamard gate on each of the input qubits, followed by a layer of phase gate. Then a phase gate between two CNOT gates is implemented. For an input vector $\vec{x}$, after encoding through the feature map, the output is:

$$F_{\phi(\vec{x})} = exp(ix_0 Z_0 + ix_1 Z_1 + i(\pi - x_0)(\pi - x_1) Z_0 Z_1) \tag{7.13}$$

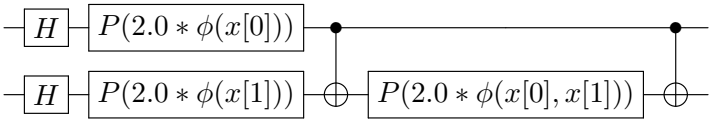


TABLE 7.1: Two qubit unitary circuit F, which is used to encode the classical information into quantum information

Quantum Convolutional Layer In order to extract the features from the encoded data(quantum state), additional circuits appended to the feature map need to be designed. Motivated by the optimal two qubits gates proposed by Vatan et al. [279], the definition of the unitary gate for constructing a quantum convolutional layer is:

$$\begin{aligned} U &= (A_1 \otimes A_2) \cdot N(\alpha, \beta, \gamma) \cdot (A_3 \otimes A_4), \\ A_i &\in SU(2) \\ N(\alpha, \beta, \gamma) &= \exp(i[\alpha \delta_x \delta_x + \beta \delta_y \delta_y + \gamma \delta_z \delta_z]) \end{aligned} \tag{7.14}$$

The construction of the optimal two qubits gates is based on an optimal circuit for computing $N(\alpha, \beta, \gamma)$. As proved by Bullock et al. [280], at least 15 elementary one-qubit gates are needed to implement a generic two-qubit gate, which is able to span the whole Hilbert space. Thus, it implies that each unitary in our QCNN must contain 15 parameters. Tuning this large amount of parameters would be difficult and would lead to long training times. To overcome this problem, the ansatz is restricted to a particular subspace of the Hilbert space and define the two qubit unitary gate as depicted in Table 7.2. These two-qubit unitaries are applied to all neighbouring qubits in each of the layers in the QCNN, as illustrated in Figure C.3.

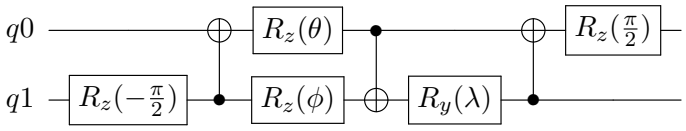


TABLE 7.2: Two-qubit unitary circuit U, which is used to construct the quantum convolutional layer

Quantum Pooling Layer In addition to the quantum convolutional layer described above, the pooling operation is also performed using quantum gates that can be designed to capture different properties of the input data and to reduce the dimension.

The pooling layer applies parameterized quantum gates to two qubits and traces out one of the qubits to reduce the two-qubit states to one-qubit states. Similar to the choice of ansatz for the convolutional filter, there exists a variety of choices of two-qubit circuits of the pooling layer. In this work, a simple form of a two-qubit circuit consisting of three free parameters is chosen for the pooling layer. Table 7.3 illustrates the unitary gate used to build the pooling layer. It applies the rotation to the Z axis, which is followed by a $CNOT$ gate. After that, Rz gates, are applied to do rotation about the Z axis. Then the output is given out after operation in another $CNOT$ and Ry gate.

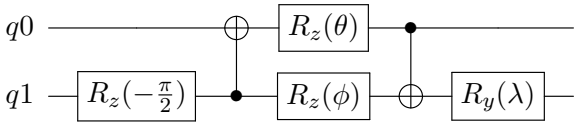


TABLE 7.3: Two-qubit unitary circuit V, which is used to construct the quantum pooling layer

Both the unitary gate of convolution and pooling are used to build the QCNN. A distinct feature of the QCNN architecture is the translational invariance, which forces the blocks of parameterized quantum gates to be identical within a layer. Considering the unitary gate composed of quantum convolution U_i and the gate part of pooling V_i, the quantum state resulting from an ith layer of QCNN can be expressed as:

$$|\psi_i(\theta_i)\rangle \langle\psi_i(\theta_i)| = Tr_{B_i}(U_i(\theta_i)V_i(\theta_i)) |\psi_{i-1}\rangle \langle\psi_{i-1}| (U_i(\theta_i)V_i(\theta_i))^\dagger \tag{7.15}$$

where $Tr_{B_i}(\cdot)$ is the partial trace operation over subsystem $B_i \in \mathbb{C}^{\frac{n}{2^i}}$, Ui is the parameterized convolution unitary gate operation, and V_i is the parameterized pooling unitary gate operation; and $|\psi_i(0)\rangle = |0\rangle^{\otimes n}$.

Cost Function The choice of a cost function of QCNN solver for PDEs is a crucial factor in the performance of the solver. The cost function determines how the network is trained and, ultimately, the accuracy of the solution it produces. In this study, mean squared error (MSE), a simple and commonly used cost function that measures the average difference between the predicted output and the true output, is chosen. Moreover, the loss of the QCNN can be expressed as:

$$\mathcal{L} = \frac{1}{2N} \sum_{i=1}^{M} (y_i - f_{\{F,U_i,V_j\}}(|\psi_i\rangle))^2 \tag{7.16}$$

Here, $f_{\{F,U_i,V_j\}}(\cdot)$ refers to the operation of QCNN that includes feature map, quantum convolution and pooling. $f_{\{F,U_i,V_j\}}(|\psi_i\rangle))$ denotes the expected QCNN output value for input $|\psi_i\rangle$. The learning process is done by firstly initializing all unitaries, and successively optimizing them until convergence by iteratively updating the parameters based on optimization methods.

Optimization The optimization of the gate parameters can be carried out by iteratively updating the parameters based on the gradient of the cost function until some condition for the termination is reached. In this study, Limited Memory Broyden-Fletcher-Goldfarb-Shanno Bound Constraints (L-BFGS-B) is adopted as the optimization. L-BFGS-B is a limited-memory quasi-Newton code for bound-constrained optimization, *e.g.,* for problems where the only constraints are of the form $l <= x <= u$. It is intended for problems in which information on the Hessian matrix is difficult to obtain or for large dense problems. It is used to minimize a function that is expressed as the sum of a differentiable function and a convex, lower-semicontinuous function.

However, like other quasi-Newton methods, it can be sensitive to the choice of initial conditions and may converge slowly for some functions. Furthermore, as a BFGS method, it can only ensure their convergence if their functions appear with nearly optimum quadratic Taylor expansion.

7.5 Experiments

In this section, some examples of PDEs that have been addressed in recent articles are considered and treated as benchmark problems in order to compare against the proposed algorithms with respect to approximation accuracy and computation time.

7.5.1 Burgers' Equation

Burgers' equation is a fundamental PDE, which is often regarded as the simplification of a more complex and sophisticated model, *e.g.,* it can be derived from the Navier-Stokes equations for the velocity field by dropping the pressure gradient term. Therefore, it is used here to understand the work and demonstrate the performance of the proposed QCNN method.

The Burgers' equation is considered:

$$\frac{\partial u}{\partial t} + \lambda u \frac{\partial u}{\partial v} = v \frac{\partial^2 u}{\partial x^2},\ x \in [0, L],\ t \in [0, \tau] \tag{7.17}$$

$$u(x, 0) = \phi(x),\ x \in [0, L] \tag{7.18}$$

$$u(0, t) = \zeta_1(t),\ u(L, t) = \zeta_2(t),\ t \in [0, \tau] \tag{7.19}$$

where u, x, t and v are the velocity, spatial coordinate, time and kinematic viscosity, respectively. ϕ, ζ_1 and ζ_2 are prescribed functions of variables depending upon the specific conditions for the problem to be solved. For the QCNN, the solution of Equation 7.17 is given by:

$$u(x; \phi) = f_{F,U_i,V_j}(\int \theta; \phi(x')) \tag{7.20}$$

In this experiment, it is considered to generate the training dataset by configuring the following parameters: $\lambda = 0.5$, $\mu = 0.01$, $\lambda = 0.5$, $\mu = 0.02$, $\lambda = 0.5$, $\mu = 0.03$, $\lambda = 0.5$, $\mu = 0.04$, $\lambda = 0.5$, $\mu = 0.05$, and it is defined that $x = 20, t = 20$, so the dataset generated on the $20 * 20$ grid, thus it is composed of $20 * 20 = 400$ data points.

In the model for solving 1D Burgers' equation, an input size of $s_{in} = 2$ is adopted, output size $s_{out} = 1$ and $d = 20$ trainable parameters. The model is trained for 70 training iterations. The details of the model adopted for training are depicted in Figure C.5. To examine the performance of the proposed model, an analytic solution is also generated by using the method proposed by Sacha et al. [4], where all the process of calculation is based on pseudo-spectral method and, more particularly, by using the Fast Fourier Transform(FFT).

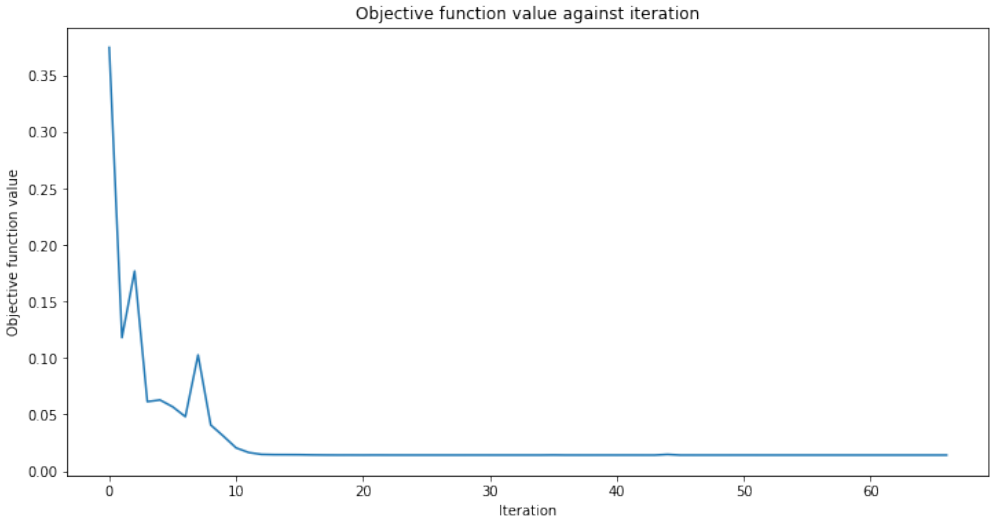


FIGURE 7.1: Observable loss over training iterations for 1D Burgers' equation

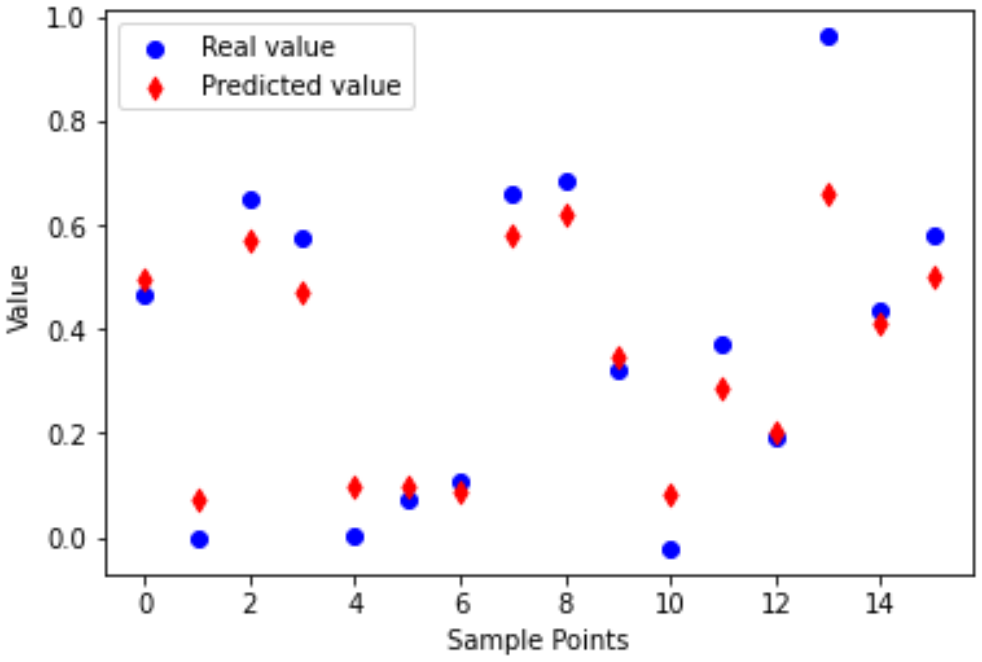


FIGURE 7.2: Comparison between the analytical solution [4] and the predicted solution by QCNN of the 1D Burgers' equation on some random points.

The predicted value given by QCNN and the analytical solution on some random sampled data points are studied and compared, and the result is shown in Figure 7.2. Figure 7.1 shows the change of loss during the training process, as can be easily seen that the QCNN gets convergence at epoch 10 with a loss around 0.01. For better visualizing the performance of the proposed QCNN, it is shown that the analytic solution and QCNN solution on 20∗20 equidistributed isometric grid points by colour, as shown in Figure 7.4. To compare these two solutions more intuitively, the difference between the two solutions at each point is calculated, and the error is visualized by a heatmap, *i.e.,* as the error

increases, the colour changes from red to blue. As can be seen, the predicted solution given by QCNN and the analytical solution have little deviations, which demonstrates that the proposed QCNN has the potential to give an accurate solution to 1D Burgers' equation.

7.5.2 Poisson Equation

As another fundamental PDE, the Poisson equation is widely adopted in mechanical engineering, electrostatics, *etc.* The theoretical solution of Poisson equation was also well studied in many previous researches [281, 282]. Therefore, it is utilized to evaluate the performance of the proposed QCNN model.

The general form of the 2D Poisson equation can be written as:

$$-\nabla \cdot (a(x,y)\nabla u(x,y)) = f(x,y) \text{ in } \Omega \tag{7.21}$$

$$u = g(x,y) \text{ on } \partial\Omega_D \tag{7.22}$$

$$\frac{\partial u}{\partial n} = h(x,y) \text{ on } \partial\Omega_N \tag{7.23}$$

The diffusivity parameter a is a spatial dependent variable, and the right-hand side term f is the source term or force term. $\partial\Omega_D$ and $\partial\Omega_N$ represent the Dirichlet and Neumann boundaries of the domain Ω respectively. In the experiment setup, two sets of experiments will be performed: the first one assumes the diffusivity parameter is constant within the whole domain, while in the second set of experiments, the diffusivity is homogeneous through the whole domain.

It is considered solving the Equation 7.21 in domain $\Omega = [0,1] \times [0,1]$ under the following boundary condition:

$$f(x,y) = sin(\pi * x)sin(\pi * y) \tag{7.24}$$

$$u(0,y) = u(1,y) = u(x,0) = 0 \tag{7.25}$$

$$\frac{\partial u}{\partial n} = 0 \text{ for } y = 1 \tag{7.26}$$

In the first experiment, the diffusivity is set as $a = 1$ in the whole domain. Furthermore, it is considered to generate the training dataset by configuring the following parameters: $grid = 5 * 5$, $grid = 7 * 7$, $grid = 9 * 9$ and $grid = 11 * 11$.

In the model for solving 2D Poisson equation, an input size of $s_{in} = 5$, output size $s_{out} = 1$ and $d = 59$ are used as trainable parameters. Moreover, the model is trained

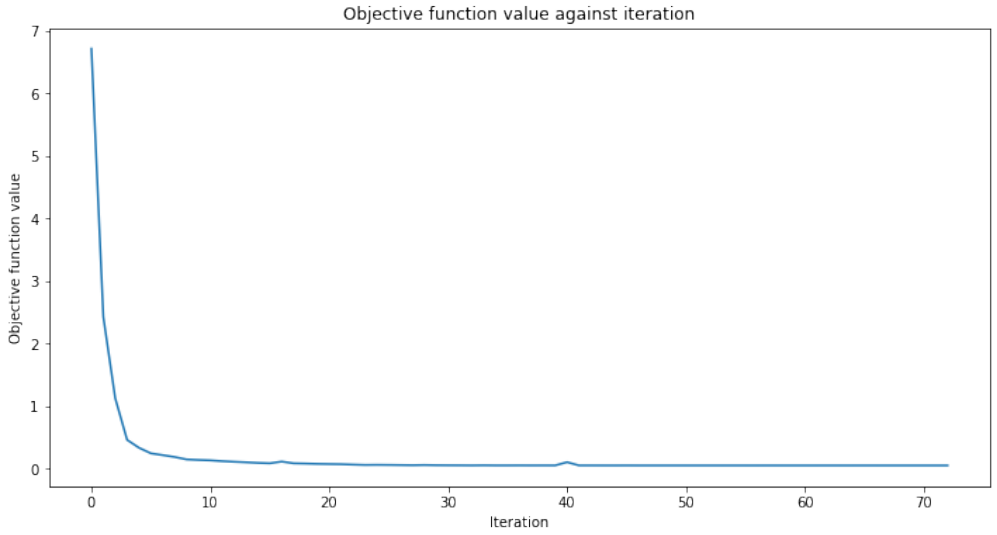


FIGURE 7.3: Observable loss over training iterations for 2D Poisson equation

for 70 training iterations. The details of the model adopted for training are depicted in Figure C.6.

Figure 7.3 shows the change of loss during the training process, as can be easily seen that the QCNN gets convergence at epoch 10 with a loss around 0.1. For better visualizing the performance of the proposed QCNN, the analytic solution and QCNN solution are visualized in 3D by colour in Figure 7.5. As can be seen, the predicted solution given by QCNN and the analytical solution have little deviations, which demonstrates that the proposed QCNN has the potential to give an accurate solution to 2D Poisson equation.

7.5.3 Trainability

The proposed QCNN architecture can be efficiently implemented on several state-of-the-art experimental platforms. The experiments were first conducted on the simulator to examine the trainability of the proposed models. Further, the hardware experiment is conducted on IBM Q System One Ehningen 27-qubit device [67] via Qiskit [283].

7.6 Conclusion and future work

Fully parameterized quantum convolutional neural networks pave promising avenues for near-term quantum machine learning and data science applications. In this work, a QCNN model is proposed for solving PDEs. The advantage of the QCNN is that it can better handle complex and nonlinear data by processing data in a higher-dimensional feature space, which enables them to capture more complex patterns and relationships

in the data. It is speculated that the advantage of QCNN lies in the ability to exploit entanglement, which is a global effect, while a classical neural network is only capable of capturing local correlations. For illustration, QCNN approach is used for solving Burgers' equation and Poisson equation, and it is shown that the QCNN can be well-trained to approximate the correct solution. Moreover, the QCNN can guide us on taking advantage of the combination of quantum computing and deep learning to solve the physical PDEs.

Though the good results are shown above, there are many important problems with efficiently using QNN to solve PDE for future works. Firstly, how to generalize the QNN model to fit into more PDEs is still under research, and a careful design and building of the architecture are needed. For the data part, whether noisy data will have the same good regularization effect is also significant work to be done. Another important work that remains to be done is to decide how much data should be used to enable an accurate solution while fitting in the current limited number of qubits that the current quantum computer has. In addition, another interesting future work is to optimize the data encoding. Moreover, understanding the underlying principle for the quantum advantage demonstrated remains to be done.

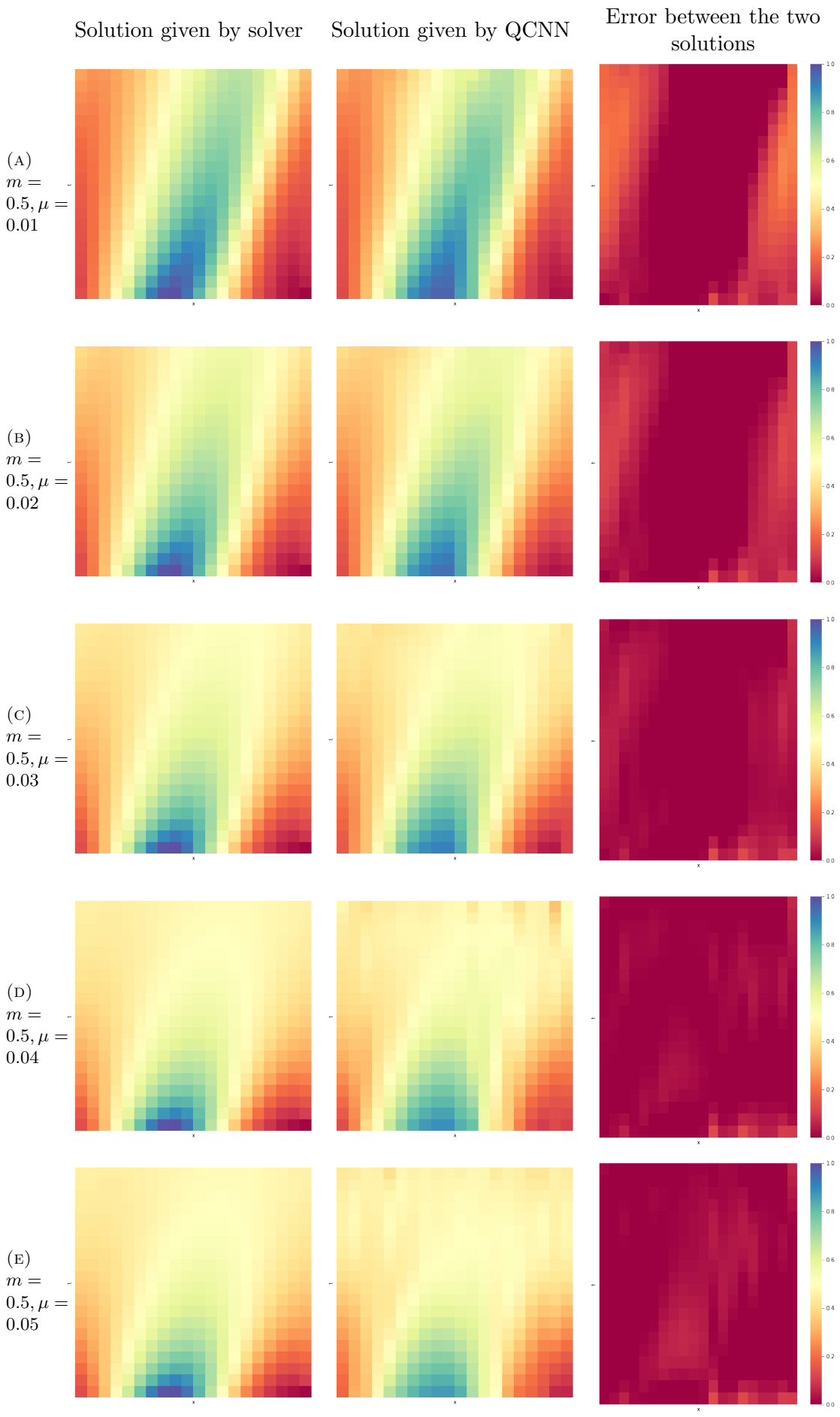


FIGURE 7.4: Comparison between the solution given by analytical solver and proposed QCNN model for 1D Burgers' Equation

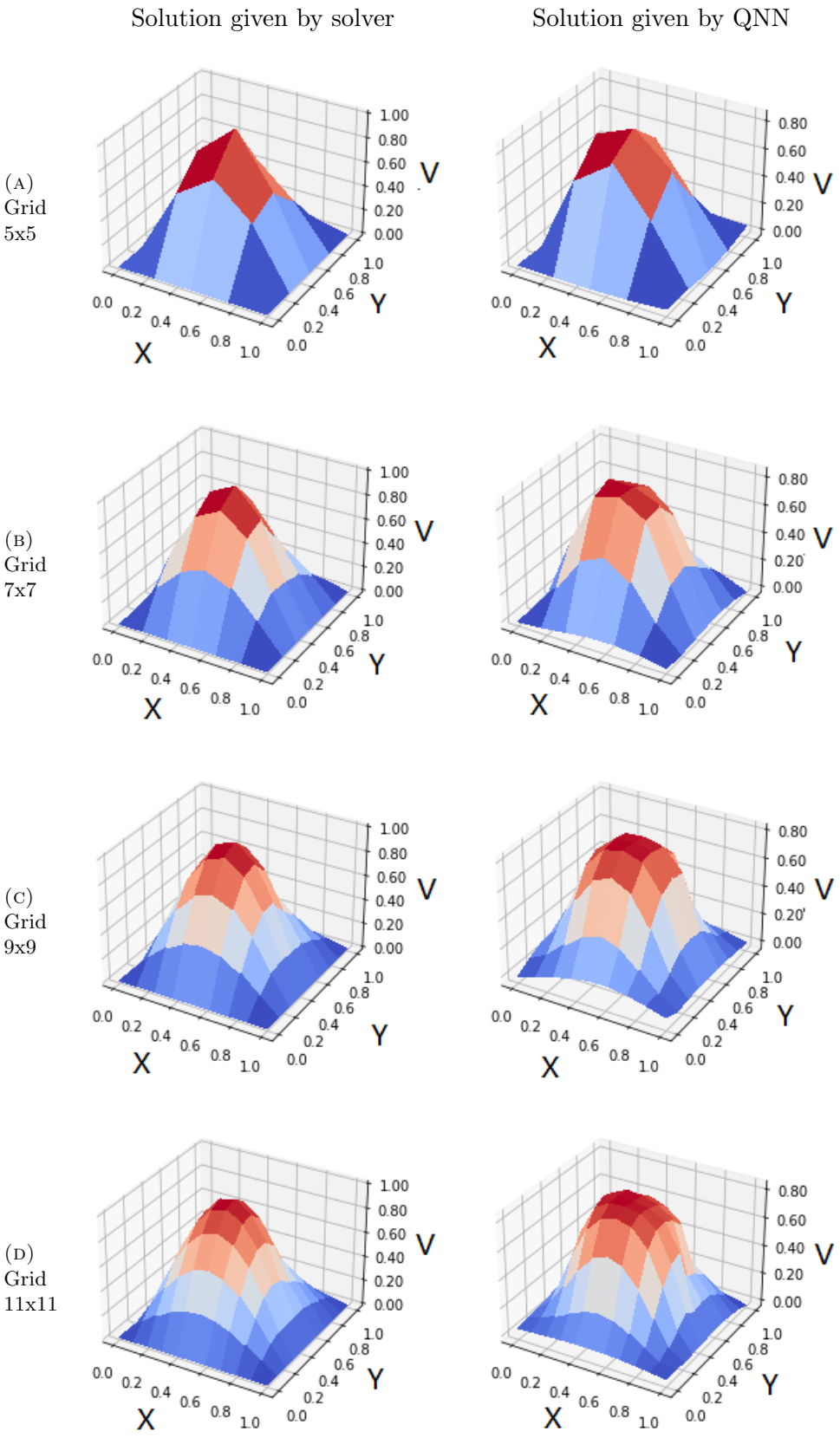


FIGURE 7.5: Comparison between the solution given by analytical solver [5] and QCNN. The left column is the visualization of the solution give by analytical solver and the right column is the solution given by the proposed QCNN

Chapter 8

Concluding Remarks

There is no doubt that deep learning methods will continue to drive revolutions in various areas, where HPC and quantum computing will play more and more important roles in these revolutions due to the fast-increasing model and data size. This dissertation explores the impact and potential development trend of hybrid workflows and DL methods on HPC and quantum computers, with the objective of addressing the main challenges encountered by current deep learning methods, namely the scarcity of data and the constraints of computing and storage resources. At first, the potential of deep learning methods in revolutionizing engineering simulations is explored, using material science as a practical example. A novel hybrid data analysis methodology that combines deep learning and simulation on HPC clusters is proposed to address the problem of data sparsity for deep learning and reduce the demand for expertise and time in determining validated parameters for simulation. The thesis also investigates the advantages of quantum computing in DNN by conducting a comparative study of QNN and classical DNN, showing that QNN can achieve higher capacity and faster training ability. To address hardware limitations in quantum computing, an effective data compression approach is proposed that reduces the dimensions of input data while retaining the most important information. In addition, a novel QNN method is proposed to give a potential solution to PDEs. The work presented in this thesis can open up a promising paradigm of hybrid DL methods on HPC systems and quantum computers and provide an explicit reference for fusing quantum and DL algorithms, which can serve as the basis for further work.

In this chapter, the chapters in this thesis are summarized. Moreover, the key takeaways and lessons from this thesis are described, and finally, ideas that could not be explored due to time limitations but will be pursued as future work are presented.

8.1 Summary

The main contributions of this dissertation are:

- A profound review is done in Chapter 3 on the techniques that drive the interplay and convergence of HPC and deep learning workloads, *e.g.,* environment setup, orchestration/scheduling, frameworks, *etc.* Furthermore, various optimization methods are described and compared from different aspects, which enable the seamless execution of deep learning applications on HPC, including parallelism methods, communication optimization, *etc.*

- In Chapter 4, a novel hybrid workflow combining a multi-task neural network and the simulation on HPC system is proposed, which can address the problem of data sparsity for deep learning and reduce the demand for expertise, resources, and time in determining the validated parameters for the simulation. To demonstrate the effectiveness of the hybrid workflow, a multi-task deep neural network is developed to identify the characteristics of the material, where the model is trained based on the data generated by the FEM simulation on HPC. Further, the time consumption of each process within the deep learning workload is analyzed, based on which several optimization approaches are adopted to improve the computing efficiency. In addition, a NAS-based AutoML approach is implemented, which generalizes the workflow and enables it to be data- and use-case-independent.

- The quantum advantage is evaluated through a comparative study of classical neural network and quantum neural network on the problem of image classification in Chapter 5. Three popular hybrid quantum DNN models, namely the quantum transfer learning (Resnet18 based), the Quanvolutional network, and the quantum convolution network, are studied by comparing their performance on the Cifar10 dataset to their corresponding classical DNN models. It is observed that adding a quantum circuit to a model could not significantly improve its accuracy or reduce its loss value. However, it is also noticed that with the involvement of quantum circuits, DNN models can converge with fewer epochs than those without quantum circuits. This indicates that the quantum advantage could be obtained at some level without sacrificing the accuracy of DNN models.

- A novel model is presented in Chapter 6 that inserts Transformer into a GAN architecture, which combines Transformers and convolutions to create an image compression codec. The encoder was implemented by preparing the image information and making it applicable to Transformer architectures. Afterward, the image information is processed into an attention map and further into a compressed token representation. The result has proven that it is possible to rebuild

the images into a token representation and to restore them almost to their original state.

- A novel QNN method is proposed for solving PDEs in Chapter 7. The effectiveness of the QNN is demonstrated by investigating its performance on two practical problems: 1D Burgers' equation and 2D Poisson equation. The success of QNN in this work provides a definitive reference for fusing QC and DL algorithms, which can serve as the basis for further work toward quantum PDE solvers.

In conclusion, this dissertation comprehensively examines different hybrid deep learning methods on HPC and quantum computers. The findings of this study give valuable insights for integrating different technologies to create a hybrid approach that can take advantage of the strengths of each technology. These results have important implications for DL, HPC, and quantum computing convergence and provide a foundation for future research.

8.2 Discussion and Future Work

In the future, with a convergence of high-performance computing, quantum computing, and deep learning, it is possible to tackle currently intractable problems. However, realizing this will require significant hardware, software, and algorithm breakthroughs. Towards this, emerging trends like DL-aided simulation, variational quantum DL, and hybrid simulation on quantum computers and supercomputers will likely continue and gather momentum. This thesis provides several solutions that drive the convergence of HPC, DL, and quantum computing in different areas addressing several requirements that arise from or are strengthened by these developing trends. Therefore, these solutions are relevant and might serve as the basis for further work towards designing future hybrid systems. The possible extension and future works can be:

- Optimization of DL training on HPC system: HPC systems may have a variety of hardware configurations, including different processors, memory sizes, and storage devices. This heterogeneity can pose challenges for scaling deep learning; thus, deep learning frameworks and libraries must be designed to take advantage of the heterogeneous hardware, and workloads must be efficiently scheduled and parallelized. In addition, load-balancing techniques shall be developed to help distribute the workloads more evenly and optimize resource usage since DL workloads are often imbalanced, which can lead to inefficient resource utilization. Communication among nodes is also critical when scaling deep learning on HPC systems. As the

number of nodes increases, the communication overhead can become a bottleneck. Efficient communication protocols and parallelization techniques are required to minimize communication overhead and achieve scalable performance.

- Performance improvement of DNN: Regarding the hybrid workflow proposed in Chapter 4, the neural network's performance shall be further improved due to the increasing demand for high accuracy of model predictions. In addition, although the performance of MTL-NAS is promising, it is also noticed that the training time for NAS is extremely long. Therefore, further optimization of the NAS algorithm should be done, especially for the distributed environment, to accelerate the training.

- Better information compression method: In Chapter 6, a Transformer-GAN based DNN model is developed to compress the data, so that the input data to a quantum computer can be dimension reduced to fit into the limited number of qubits of current quantum computers. However, the Transformer-GAN method involves complex encoding and decoding algorithms that can be computationally intensive, which can be a challenge for applications that require real-time compression, such as video streaming or live video conferencing. It is also noted that different applications have different requirements for image compression in terms of compression ratio, image quality, and computational complexity. Thus, a more generalized method that can be used for data compression without losing much information shall be developed.

- Improving quantum scalability: One of the challenges with QNNs is that they are difficult to scale up to larger problems due to a phenomenon known as quantum decoherence [284], which causes errors and instability in quantum computations as the number of qubits increases. Future work in this area could focus on developing more efficient and scalable methods for training and using QNNs, which is essential to unlocking the full potential of quantum computing and enabling new applications in areas such as cryptography, machine learning, and materials science.

- Development of quantum hardware: One of the biggest challenges with QNNs is requiring specialized hardware to run. However, many challenges are associated with quantum hardware, *e.g.,* quantum noise, the limited number of qubits, calibration and error correction, and cost. One major future work is to develop new and more accessible quantum hardware, which could make it easier to use and study QNNs.

Appendix A

Multi-Task Neural Architecture Search

Figure A.1 illustrates the problem formulation of the general-purpose MTL-NAS developed by Gao et.al. It disentangles the GP-MTL networks into fixed task-specific single-task backbones and general feature fusion schemes between them. This allows to define a general task-agnostic search space compatible with any task combinations, as shown in the leftmost subfigure. The right-top subfigure illustrates the inter-task fusion operation. The initialization of the fusion operation is shown in the right-bottom subfigure. New edges are inserted between the fixed and well-trained single-task network backbones, in order to make a minimal impact on the original output at each layer at initialization.

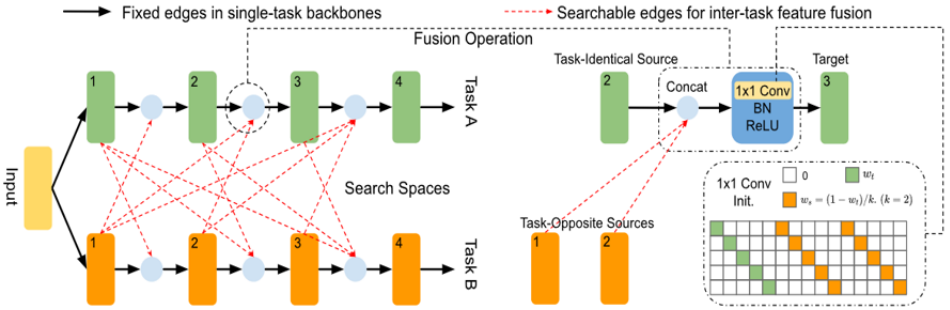


FIGURE A.1: Task-agnostic neural architecture search towards general-purpose multi-task learning[6]

Appendix B

TransGAN Structure

```
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 128, 128, 64)      3136
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 128, 128, 64)      0
_________________________________________________________________
layer_normalization_8 (Layer (None, 128, 128, 64)      128
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 64, 64, 128)       131200
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 64, 64, 128)       0
_________________________________________________________________
layer_normalization_9 (Layer (None, 64, 64, 128)       256
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 32, 32, 256)       524544
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
layer_normalization_10 (Laye (None, 32, 32, 256)       512
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 16, 16, 512)       2097664
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 16, 16, 512)       0
_________________________________________________________________
layer_normalization_11 (Laye (None, 16, 16, 512)       1024
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 1)         8193
_________________________________________________________________
size2 (size2)                (None, 16, 16, 1)         0
=================================================================
Total params: 2,766,657
Trainable params: 2,766,657
Non-trainable params: 0
```

FIGURE B.1: Configuration of discriminator part of TransGAN for image compression

Layer (type)	Output Shape	Param #
project_patches (project_pat	(None, 1024, 128)	24704
position_embedding (position	(None, 1024, 128)	131072
dropout (Dropout)	(None, 1024, 128)	0
transformer (transformer)	(None, 1024, 128)	165248
transformer_1 (transformer)	(None, 1024, 128)	165248
transformer_2 (transformer)	(None, 1024, 128)	165248
layer_normalization_3 (Layer	(None, 1024, 128)	256
prep_att (prep_att)	(None, 32, 32, 128)	0
attention_map (attention_map	(None, 32, 128)	55552
noise (noise)	(None, 64, 128)	0
transformer_3 (transformer)	(None, 64, 128)	165248
transformer_4 (transformer)	(None, 64, 128)	165248
transformer_5 (transformer)	(None, 64, 128)	165248
dense_12 (Dense)	(None, 64, 128)	16384
reshape_1 (Reshape)	(None, 4, 4, 512)	0
residual_block (residual_blo	(None, 4, 4, 512)	2361344
residual_block_1 (residual_b	(None, 4, 4, 512)	2361344
residual_block_2 (residual_b	(None, 4, 4, 512)	2361344
upsample (upsample)	(None, 8, 8, 512)	2361344
upsample_1 (upsample)	(None, 16, 16, 256)	1180672
upsample_2 (upsample)	(None, 32, 32, 128)	295424
upsample_3 (upsample)	(None, 64, 64, 64)	73984
upsample_4 (upsample)	(None, 128, 128, 32)	18560
upsample_5 (upsample)	(None, 256, 256, 16)	4672
conv2d_10 (Conv2D)	(None, 256, 256, 3)	3075
size (size)	(None, 256, 256, 3)	0

Total params: 12,241,219
Trainable params: 12,236,131
Non-trainable params: 5,088

FIGURE B.2: Configuration of generator part of TransGAN for image compression

Appendix C

Details of QDNN PDE Solver Model

In the following, the chosen models Chapter 7 is explained in more detail.

Figure C.1 illustrates a parametrized two qubit unitary circuit, which will be applied to all neighboring qubits as is shown in Figure C.3, where the two qubit unitary gates are appliedto all even pairs of qubits followed by applying to odd pairs of qubits in a circular coupling manner, *i.e.,* the as well as neighboring qubits being coupled, the first and final qubit are also coupled through a unitary gate. Thus the convolutional layer is constructed.

Figure C.2 shows the two qubit unitary, which transforms the two qubit system to one. After applying this two qubit unitary circuit, the first qubit (q0) is neglected in future layers and only use the second qubit (q1) in our QCNN. Further, this two qubit pooling layer is applied to different pairs of qubits to create our pooling layer for N qubits, which is shown in Figure C.4 . This layer therefore has the overall effect of 'combining' the information of the two qubits into one qubit by first applying the unitary circuit, encoding information from one qubit into another, before disregarding one of qubits for the remainder of the circuit and not performing any operations or measurements on it.

The quantum neural networks considered in Chapter 7 are of the form given in Figure C.5 and Figure C.5, which are composed of the quantum convolutional layers and quantum pooling layers.

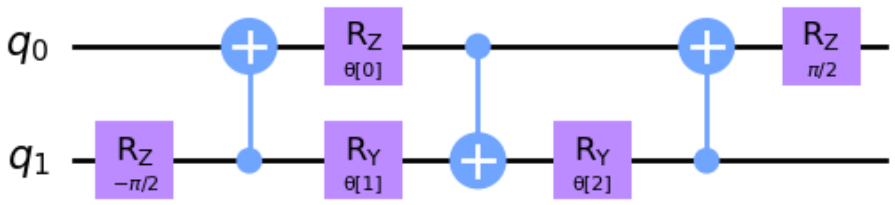


FIGURE C.1: Details of the convolutional unitary gate

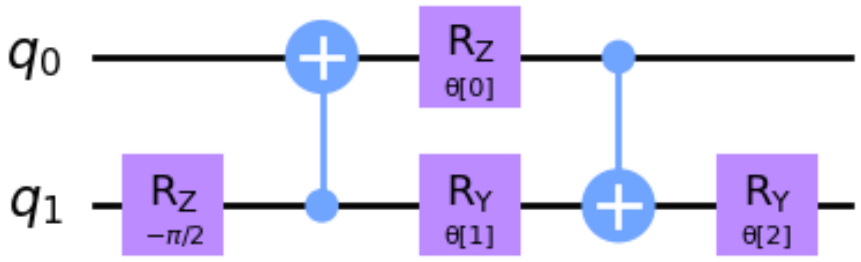


FIGURE C.2: Details of the pooling unitary gate

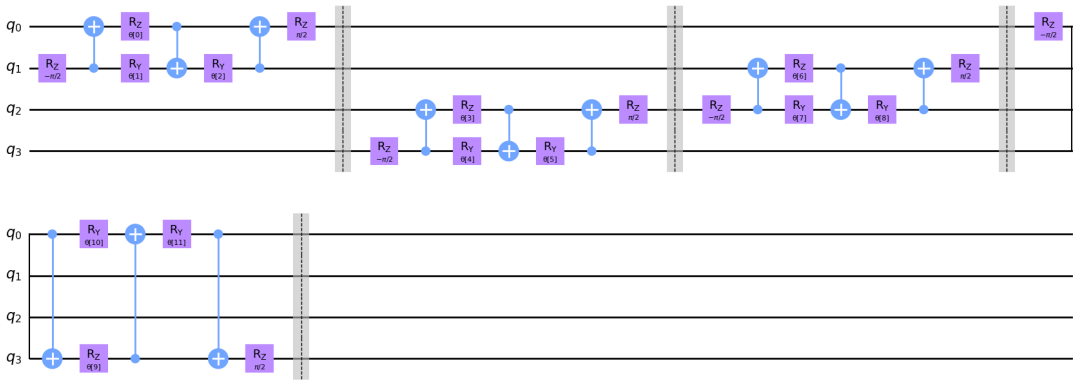


FIGURE C.3: Details of the quantum convolutional layer

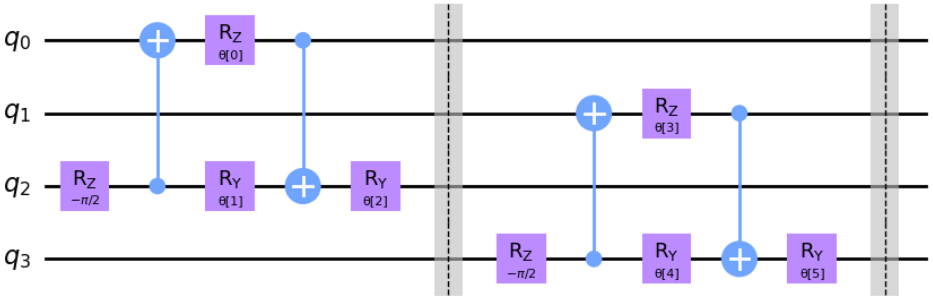


FIGURE C.4: Details of the quantum pooling layer

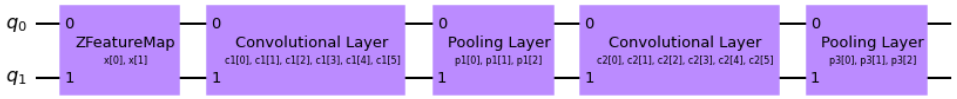


FIGURE C.5: The QNN structure adopted for solving the 1D Burgers' Equation

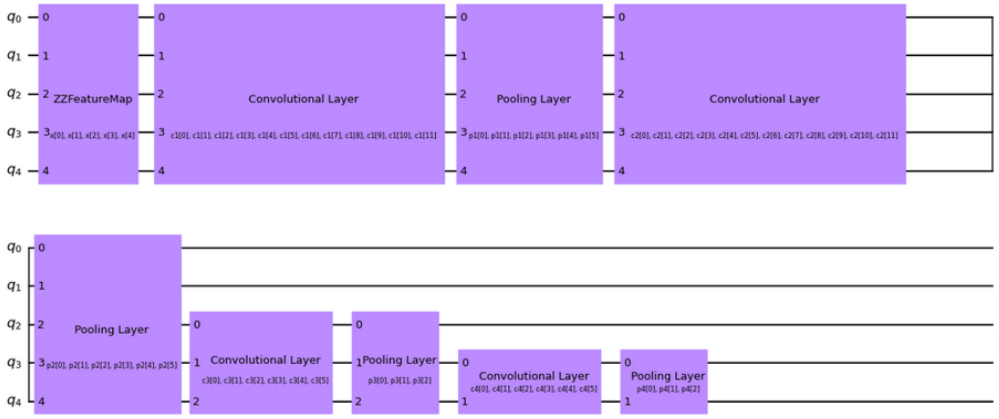


FIGURE C.6: The QNN structure adopted for solving the 2D Poisson Equation

Bibliography

[1] Top500, 2022. URL https://top500.org/.

[2] Maxwell Henderson, Samriddhi Shakya, Shashindra Pradhan, and Tristan Cook. Quanvolutional neural networks: powering image recognition with quantum circuits. *Quantum Machine Intelligence*, 2(1):2, 2020.

[3] Seunghyeok Oh, Jaeho Choi, and Joongheon Kim. A tutorial on quantum convolutional neural networks (qcnn). In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 236–239. IEEE, 2020.

[4] Sacha BINDER. tude de lobservation et de la modlisation des ondes de surface en eau peu profonde. *Physique-Informatique*, 2021.

[5] Alexander Heinlein, Axel Klawonn, and Oliver Rheinbach. A parallel implementation of a two-level overlapping schwarz method with energy-minimizing coarse space based on trilinos. *SIAM Journal on Scientific Computing*, 38(6):C713–C747, 2016.

[6] Yuan Gao, Haoping Bai, Zequn Jie, Jiayi Ma, Kui Jia, and Wei Liu. Mtl-nas: Task-agnostic neural architecture search towards general-purpose multi-task learning. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 11543–11552, 2020.

[7] Eirikur Agustsson, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool. Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 221–231, 2019.

[8] Myeongsuk Pak and Sanghoon Kim. A review of deep learning in image recognition. In *2017 4th international conference on computer applications and information processing technology (CAIPT)*, pages 1–3. IEEE, 2017.

[9] Keerthana Rangasamy, Muhammad Amir As'ari, Nur Azmina Rahmad, Nurul Fathiah Ghazali, and Saharudin Ismail. Deep learning in sport video analysis: a review. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 18(4):1926–1933, 2020.

[10] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[11] Yongjun Xu, Xin Liu, Xin Cao, Changping Huang, Enke Liu, Sen Qian, Xingchen Liu, Yanjun Wu, Fengliang Dong, Cheng-Wei Qiu, et al. Artificial intelligence: A powerful paradigm for scientific research. *The Innovation*, 2(4), 2021.

[12] Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694, 2020.

[13] Fei Wang, Hao Wang, Haichao Wang, Guowei Li, and Guohai Situ. Learning from simulation: An end-to-end deep-learning approach for computational ghost imaging. *Optics express*, 27(18):25560–25572, 2019.

[14] Andreas Tolk. The next generation of modeling & simulation: integrating big data and deep learning. In *Proceedings of the conference on summer computer simulation*, pages 1–8, 2015.

[15] Fei Wang, Hao Wang, Haichao Wang, Guowei Li, and Guohai Situ. Learning from simulation: An end-to-end deep-learning approach for computational ghost imaging. *Opt. Express*, 27(18):25560–25572, Sep 2019. doi: 10.1364/OE.27.025560. URL https://opg.optica.org/oe/abstract.cfm?URI=oe-27-18-25560.

[16] Iulia Buluta and Franco Nori. Quantum simulators. *Science*, 326(5949):108–111, 2009.

[17] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

[18] Carlos Bravo-Prieto, Ryan LaRose, Marco Cerezo, Yigit Subasi, Lukasz Cincio, and Patrick J Coles. Variational quantum linear solver. *arXiv preprint arXiv:1909.05820*, 2019.

[19] Siddhant Garg and Goutham Ramakrishnan. Advances in quantum deep learning: An overview. *arXiv preprint arXiv:2005.04316*, 2020.

[20] Kerstin Beer, Dmytro Bondarenko, Terry Farrelly, Tobias J Osborne, Robert Salzmann, Daniel Scheiermann, and Ramona Wolf. Training deep quantum neural networks. *Nature communications*, 11(1):1–6, 2020.

[21] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, 2021.

[22] Klaus Mainzer and Mainzer. *Künstliche Intelligenz-wann übernehmen die Maschinen?* Springer, 2016.

[23] D. M. Hutton. *The quest for artificial intelligence: A history of ideas and achievements*. Kybernetes, 2011.

[24] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[25] David Forsyth and Jean Ponce. *Computer vision: A modern approach*. Prentice Hall, 2011.

[26] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724, 2015.

[27] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[28] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[29] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[30] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[31] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, oct 2020. ISSN 0001-0782. doi: 10.1145/3422622. URL https://doi.org/10.1145/3422622.

[32] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[34] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017.

[35] Derya Soydaner. A comparison of optimization algorithms for deep learning. *International Journal of Pattern Recognition and Artificial Intelligence*, 34(13):2052013, 2020.

[36] J. Dongarra, T. Sterling, H. Simon, and E. Strohmaier. High-performance computing: clusters, constellations, mpps, and future directions. *Computing in Science Engineering*, 7(2):51–59, 2005. doi: 10.1109/MCSE.2005.34.

[37] Roman Trobec, Radivoje Vasiljević, Milo Tomašević, Veljko Milutinović, Ramon Beivide, and Mateo Valero. Interconnection networks in petascale computer systems: A survey. *ACM Computing Surveys (CSUR)*, 49(3):1–24, 2016.

[38] Jose Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection networks*. Morgan Kaufmann, 2003.

[39] Ananth Grama, Vipin Kumar, Anshul Gupta, and George Karypis. *Introduction to parallel computing*. Pearson Education, 2003.

[40] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE transactions on Computers*, 39(6):775–785, 1990.

[41] Stephen R Walli. The posix family of standards. *StandardView*, 3(1):11–17, 1995.

[42] Lustre file system, 2022. URL https://www.lustre.org/.

[43] Beegfs, the leading parallel file system, 2022. URL https://www.beegfs.io/c/.

[44] Ibm general parallel file system documentation, 2022. URL https://www.ibm.com/docs/en/gpfs.

[45] Peter Corbett, Dror Feitelson, Sam Fineberg, Yarsun Hsu, Bill Nitzberg, Jean-Pierre Prost, Marc Snirt, Bernard Traversat, and Parkson Wong. Overview of the mpi-io parallel i/o interface. In *Input/Output in Parallel and Distributed Computer Systems*, pages 127–146. Springer, 1996.

[46] Mike Folk, Albert Cheng, and Kim Yates. Hdf5: A file format and i/o library for high performance computing applications. In *Proceedings of supercomputing*, volume 99, pages 5–33, 1999.

[47] William F Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, et al. Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561, 2020.

[48] Slurm workload manager, 2022. URL https://slurm.schedmd.com/documentation.html.

[49] Moab hpc suite, 2022. URL https://adaptivecomputing.com/moab-hpc-suite/.

[50] Torque, 2022. URL https://github.com/adaptivecomputing/torque.

[51] Openpbs, 2022. URL https://github.com/openpbs/openpbs.

[52] Hpe apollo (hawk), 2022. URL https://www.hlrs.de/systems/hpe-apollo-hawk/.

[53] Cray cs-storm, 2022. URL https://www.hlrs.de/systems/cray-cs-storm/.

[54] Kimberly Keeton. Memory-driven computing. In *FAST*, 2017.

[55] Wim Vanderbauwhede and Khaled Benkrid. *High-performance computing using FPGAs*, volume 3. Springer, 2013.

[56] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021.

[57] Paul AM Dirac. The lagrangian in quantum mechanics. In *Feynman's Thesis—A New Approach To Quantum Theory*, pages 111–119. World Scientific, 2005.

[58] Richard P Feynman. Quantum mechanical computers. *Found. Phys.*, 16(6):507–532, 1986.

[59] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

[60] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. doi: 10.1109/SFCS.1994.365700.

[61] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917855. doi: 10.1145/237814.237866. URL https://doi.org/10.1145/237814.237866.

[62] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[63] Michael A. Nielsen and Isaac Chuang. Quantum computation and quantum information. *American Journal of Physics*, 70(5):558–559, 2002. doi: 10.1119/1.1463744. URL https://doi.org/10.1119/1.1463744.

[64] Roderich Tumulka. Dirac notation. In *Compendium of Quantum Physics*, pages 172–174. Springer, 2009.

[65] Johannes Jisse Duistermaat and Johan AC Kolk. *Lie groups*. Springer Science & Business Media, 2012.

[66] Christopher M Dawson and Michael A Nielsen. The solovay-kitaev algorithm. *arXiv preprint quant-ph/0505030*, 2005.

[67] Ibm quantum system one at ehningen germany, 2023. URL https://www.fraunhofer.de/content/dam/zv/de/institute-einrichtungen/Kooperationen/kompetenznetzwerk-quantencomputing/brochure_fraunhofer-v10.pdf.

[68] Sardar Usman, Rashid Mehmood, and Iyad Katib. Big data and hpc convergence for smart infrastructures: A review and proposed architecture. *Smart Infrastructure and Applications: Foundations for Smarter Cities and Societies*, pages 561–586, 2020.

[69] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1): 46–55, 1998.

[70] William Gropp, William D Gropp, Ewing Lusk, Anthony Skjellum, and Argonne Distinguished Fellow Emeritus Ewing Lusk. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[71] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 265–283, USA, 2016. USENIX Association. ISBN 9781931971331.

[72] Huasha Zhao and John Canny. Kylix: A sparse allreduce for commodity clusters. In *2014 43rd International Conference on Parallel Processing*, pages 273–282, 2014. doi: 10.1109/ICPP.2014.36.

[73] Gaurav Bhatia, Arjun Choudhary, and Vipin Gupta. The road to docker: a survey. *International Journal of Advanced Research in Computer Science*, 8:8, 2017.

[74] Weidong Liao and Jesse Draper. Cloud computing and docker containerization: A survey. In *Proceedings of the West Virginia Academy of Science*, volume 91. 91, 2019.

[75] Emiliano Casalicchio. Container orchestration: a survey. *Systems Modeling: Methodologies and Tools*, pages 221–235, 2019.

[76] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[77] Maria A Rodriguez and Rajkumar Buyya. Container-based cluster orchestration systems: A taxonomy and future directions. *Software: Practice and Experience*, 49(5):698–719, 2019.

[78] Denis Weerasiri, Moshe Chai Barukh, Boualem Benatallah, Quan Z Sheng, and Rajiv Ranjan. A taxonomy and survey of cloud resource orchestration techniques. *ACM Computing Surveys (CSUR)*, 50(2):1–41, 2017.

[79] Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. A survey of methods for distributed machine learning. *Progress in Artificial Intelligence*, 2(1):1–11, 2013.

[80] Yoshua Bengio. Deep learning of representations: Looking forward. In *International conference on statistical language and speech processing*, pages 1–37. Springer, 2013.

[81] Xue-Wen Chen and Xiaotong Lin. Big data deep learning: challenges and perspectives. *IEEE access*, 2:514–525, 2014.

[82] Giang Nguyen, Ján Astaloš, and Ladislav Hluchỳ. Considerations about data processing, machine learning, hpc, apache spark and gpu. In *11th Workshop on Intelligent and Knowledge Oriented Technologies in conjunction with 35th conference Data and Knowledge*, pages 241–247, 2016.

[83] HamidReza Asaadi, Dounia Khaldi, and Barbara Chapman. A comparative survey of the hpc and big data paradigms: Analysis and experiments. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 423–432. IEEE, 2016.

[84] Malik Khan, Tobias Becker, Permural Kuppuudaiyar, and Anne C. Elster. Container-based virtualization for heterogeneous hpc clouds: Insights from the eu h2020 cloudlightning project. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 392–397, 2018. doi: 10.1109/IC2E.2018.00074.

[85] Maria A. Rodriguez and Rajkumar Buyya. Container-based cluster orchestration systems: A taxonomy and future directions. *Software: Practice and Experience*, 49(5):698–719, 2019. doi: https://doi.org/10.1002/spe.2660. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2660.

[86] Jack S. Hale, Lizao Li, Christopher N. Richardson, and Garth N. Wells. Containers for portable, productive, and performant scientific computing. *Computing in Science Engineering*, 19(6):40–50, 2017. doi: 10.1109/MCSE.2017.2421459.

[87] RANCHER, 2022.

[88] Akihiro Suda. Usernetes: Kubernetes without the root privileges. URL https://github.com/rootless-containers/usernetes.

[89] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), mar 2014. ISSN 1075-3583.

[90] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. *PLoS ONE*, 12(5), 5 2017. doi: 10.1371/journal.pone.0177459.

[91] S Senthil Kumaran. *Practical LXC and LXD: linux containers for virtualization and orchestration*. Springer, 2017.

[92] Lisa Gerhardt, Wahid Bhimji, Shane Canon, Markus Fasel, Doug Jacobsen, Mustafa Mustafa, Jeff Porter, and Vakho Tsulaia. Shifter: Containers for HPC. *Journal of Physics: Conference Series*, 898:082021, oct 2017. doi: 10.1088/1742-6596/898/8/082021. URL https://doi.org/10.1088/1742-6596/898/8/082021.

[93] Marvin Newlin, Kyle Smathers, and Mark E. DeYoung. Arc containers for ai workloads: Singularity performance overhead. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, PEARC '19, New York, NY, USA, 2019. Association for Computing Machinery.

ISBN 9781450372275. doi: 10.1145/3332186.3333048. URL https://doi.org/10.1145/3332186.3333048.

[94] Scott McMillan. *Making Container Easier with HPC Container Maker*. HPCSYSPROS18: HPC System Professionals Workshop, Dallas, TX, 2018.

[95] Felipe A. Cruz Alberto Madonna Benedicic, Lucas and Kean Mariotti. Portable, high-performance containers for hpc. preprint, 2017.

[96] Reid Priedhorsky and Tim Randles. Charliecloud: Unprivileged containers for user-defined software stacks in hpc. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351140. doi: 10.1145/3126908.3126925. URL https://doi.org/10.1145/3126908.3126925.

[97] David Brayford, Sofia Vallecorsa, Atanas Atanasov, Fabio Baruffa, and Walter Riviera. Deploying ai frameworks on secure hpc systems with containers. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2019. doi: 10.1109/HPEC.2019.8916576.

[98] David Brayford and Sofia Vallecorsa. Deploying scientific ai networks at petaflop scale on secure large scale hpc production systems with containers. preprint, 2020.

[99] Alfred Torrez, Timothy Randles, and Reid Priedhorsky. Hpc container runtimes have minimal or no performance impact. In *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, pages 37–42, 2019. doi: 10.1109/CANOPIE-HPC49598.2019.00010.

[100] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60. Springer, 2003.

[101] Garrick Staples. Torque resource manager. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, pages 8–es, 2006.

[102] Naweiluo Zhou, Yiannis Georgiou, Li Zhong, Huan Zhou, and Marcin Pospieszny. Container orchestration on hpc systems. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 34–36, 2020. doi: 10.1109/CLOUD49709.2020.00017.

[103] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[104] Miguel G. Xavier, Marcelo V. Neves, Fabio D. Rossi, Tiago C. Ferreto, Timoteo Lange, and Cesar A. F. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 233–240, 2013. doi: 10.1109/PDP.2013.41.

[105] Max Plauth, Lena Feinbube, and Andreas Polze. A performance survey of lightweight virtualization techniques. In *European Conference on Service-Oriented and Cloud Computing*, pages 34–48. Springer, 2017.

[106] Jie Zhang, Xiaoyi Lu, and Dhabaleswar K. Panda. Is singularity-based container technology ready for running mpi applications on hpc clouds? UCC '17, page 151–160, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351492. doi: 10.1145/3147213.3147231. URL https://doi.org/10.1145/3147213.3147231.

[107] Brendan Burns, Joe Beda, and Kelsey Hightower. *Kubernetes: up and running: dive into the future of infrastructure*. O'Reilly Media, 2019.

[108] V. Pisaruk and Sasha. Yakovtseva. Wlm-operator, October 2020. URL https://github.com/sylabs/wlm-operator.

[109] Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, Prabhat, and Michael Houston. Exascale deep learning for climate analytics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18. IEEE Press, 2018.

[110] Paola Buitrago, Nicholas Nystrom, Rajarsi Gupta, and Joel Saltz. *Delivering Scalable Deep Learning to Research with Bridges-AI*, pages 200–214. 02 2020. ISBN 978-3-030-41004-9. doi: 10.1007/978-3-030-41005-6_14.

[111] Amiya K Maji, Lev Gorenstein, and Geoffrey Lentner. Demystifying python package installation with conda-env-mod. In *2020 IEEE/ACM International Workshop on HPC User Support Tools (HUST) and Workshop on Programming and Performance Visualization Tools (ProTools)*, pages 27–37. IEEE, 2020.

[112] Virtualenv, 2022. URL https://virtualenv.pypa.io/en/latest/.

[113] Kenneth Hoste, Jens Timmerman, Andy Georges, and Stijn De Weirdt. Easybuild: Building software with ease. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 572–582. IEEE, 2012.

[114] Todd Gamblin, Matthew LeGendre, Michael R Collette, Gregory L Lee, Adam Moody, Bronis R De Supinski, and Scott Futral. The spack package manager: bringing order to hpc software chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2015.

[115] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[116] Seunghak Lee, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A Gibson, and Eric P Xing. On model parallelization and scheduling strategies for distributed machine learning. *Advances in neural information processing systems*, 27, 2014.

[117] Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for l1-regularized loss minimization. *arXiv preprint arXiv:1105.5379*, 2011.

[118] Chad Scherrer, Ambuj Tewari, Mahantesh Halappanavar, and David Haglin. Feature clustering for accelerating parallel coordinate descent. *Advances in Neural Information Processing Systems*, 25, 2012.

[119] Ruben Mayer, Christian Mayer, and Larissa Laich. The tensorflow partitioning and scheduling problem: it's the critical path! In *Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning*, pages 1–6, 2017.

[120] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *International Conference on Machine Learning*, pages 2430–2439. PMLR, 2017.

[121] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V Le, and Jeff Dean. A hierarchical model for device placement. In *International Conference on Learning Representations*, 2018.

[122] Ruben Mayer and Hans-Arno Jacobsen. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. *ACM Computing Surveys (CSUR)*, 53(1):1–37, 2020.

[123] UA Muller and A Gunzinger. Neural net simulation on parallel computers. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 6, pages 3961–3966. IEEE, 1994.

[124] Ludvig Ericson and Rendani Mbuvha. On the performance of network parallel training in artificial neural networks. *arXiv preprint arXiv:1701.05130*, 2017.

[125] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, 2014.

[126] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[127] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 1–13, 2019. URL https://proceedings.mlsys.org/paper/2019/file/c74d97b01eae257e44aa9d5bade97baf-Paper.pdf.

[128] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 873–880, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553486. URL https://doi.org/10.1145/1553374.1553486.

[129] Stephen J. Nocedal, Jorgeand Wright, editor. *Sequential Quadratic Programming*, pages 526–573. Springer New York, New York, NY, 1999. ISBN 978-0-387-22742-9. doi: 10.1007/0-387-22742-3_18. URL https://doi.org/10.1007/0-387-22742-3_18.

[130] Markus Weimer Lihong Li Zinkevich, Martin and Alex Smola. Parallelized stochastic gradient descent. *In Advances in neural information processing systems*, 23: 2595–2603, 2010.

[131] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. Heterogeneity-aware distributed parameter servers. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 463–478, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450341974. doi: 10.1145/3035918.3035933. URL https://doi.org/10.1145/3035918.3035933.

[132] Quoc V. Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y. Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 265–272, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

[133] Kunlei Zhang and Xue-Wen Chen. Large-scale deep belief nets with mapreduce. *IEEE Access*, 2:395–403, 2014. doi: 10.1109/ACCESS.2014.2319813.

[134] Matthew A. Johnson Maik Riechert Daniel Tarlow Ryota Tomioka Dimitrios Vytiniotis Gaunt, Alexander L. and Sam Webster. Ampnet: Asynchronous model-parallel training for dynamic neural networks. preprint, 2017.

[135] Cedric Renggli, Saleh Ashkboos, Mehdi Aghagolzadeh, Dan Alistarh, and Torsten Hoefler. Sparcml: High-performance sparse communication for machine learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362290. doi: 10.1145/3295500.3356222. URL https://doi.org/10.1145/3295500.3356222.

[136] Yosuke Oyama, Tal Ben-Nun, Torsten Hoefler, and Satoshi Matsuoka. Accelerating deep learning frameworks with micro-batches. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 402–412, 2018. doi: 10.1109/CLUSTER.2018.00058.

[137] Aleksandar Zlateski, Kisuk Lee, and H. Sebastian Seung. Znni: Maximizing the inference throughput of 3d convolutional networks on cpus and gpus. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 854–865, 2016. doi: 10.1109/SC.2016.72.

[138] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[139] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[140] Chi-Chung Chen, Chia-Lin Yang, and Hsiang-Yun Cheng. Efficient and robust parallel dnn training through model parallelism on multi-gpu platform. arxiv. preprint, 2018.

[141] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Comput. Surv.*, 52(4), aug 2019. ISSN 0360-0300. doi: 10.1145/3320060. URL https://doi.org/10.1145/3320060.

[142] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard,

Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016. URL https://arxiv.org/abs/1603.04467.

[143] Ronan Collobert, Koray Kavukcuoglu, and Clement Farabet. Torch7: A matlab-like environment for machine learning. 01 2011.

[144] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, MM '14, page 675–678, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450330633. doi: 10.1145/2647868.2654889. URL https://doi.org/10.1145/2647868.2654889.

[145] Igor Colin, Ludovic Dos Santos, and Kevin Scaman. *Theoretical Limits of Pipeline Parallel Optimization and Application to Distributed Deep Learning*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[146] Peter L. Bartlett Duchi, John C. and Martin J. Wainwright. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22(2):674–701, 2012.

[147] Francis Bach Sébastien Bubeck Laurent Massoulié Scaman, Kevin and Yin Tat Lee. Optimal algorithms for non-smooth distributed optimization in networks. *In Advances in Neural Information Processing Systems*, pages 2740–2749, 2018.

[148] Kevin Swersky Snell, Jake and Richard Zemel. Prototypical networks for few-shot learning. *In Advances in neural information processing systems*, pages 4077–4087, 2017.

[149] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, page 641–647, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 159593135X. doi: 10.1145/1081870.1081950. URL https://doi.org/10.1145/1081870.1081950.

[150] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015. URL https://arxiv.org/abs/1511.06434.

[151] Aaron Harlap, Harlap, Deepak Narayanan, Amar Phanishayee, Greg Ganger Vivek Seshadri, Nikhil Devanur, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. arxiv. preprint, 2018.

[152] Deepak Narayanan Amar Phanishayee Vivek Seshadri Gregory R. Ganger Harlap, Aaron and Phillip B. Gibbons. Pipedream: Pipeline parallelism for dnn training. In *Conference on Systems and Machine Learning*, 2018.

[153] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1627–1635. JMLR.org, 2017.

[154] Michael Laskin, Luke Metz, Seth Nabarro, Mark Saroufim, Badreddine Noune, Carlo Luschi, Jascha Sohl-Dickstein, and Pieter Abbeel. Parallel training of deep networks with local updates, 2020. URL https://arxiv.org/abs/2012.03837.

[155] Shirish Tatikonda Tian, Yuanyuan and Berthold Reinwald. Scalable and numerically stable descriptive statistics in systemml. In *InIEEE 28th International Conference on Data Engineering*, pages 1351–1359. IEEE, April 2012.

[156] Rajasekar Krishnamurthy Edwin Pednault Berthold Reinwald Vikas Sindhwani Shirish Tatikonda Yuanyuan Tian Ghoting, Amol and Shivakumar Vaithyanathan. Systemml: Declarative machine learning on mapreduce. pages 231–242. IEEE 27th International Conference on Data Engineering, April 2011.

[157] Shirish Tatikonda Berthold Reinwald Prithviraj Sen Yuanyuan Tian Douglas R. Burdick Boehm, Matthias and Shivakumar Vaithyanathan. Hybrid parallelization strategies for large-scale machine learning in systemml. In *Proceedings of the VLDB Endowment*, pages 553–564, 2014. 7(7).

[158] Lele Yu Jiawei Jiang Yuhong Liu Jiang, Jie and Bin Cui. Angel: a new large-scale machine learning system. *National Science Review*, 5(2):216–236, 2018.

[159] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf.

[160] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system.

In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 571–582, Broomfield, CO, October 2014. USENIX Association. ISBN 978-1-931971-16-4. URL https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chilimbi.

[161] Torsten Hoefler and Jesper Larsson Traff. Sparse collective operations for mpi. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–8, 2009. doi: 10.1109/IPDPS.2009.5160935.

[162] Vinod Tipparaju Manojkumar Krishnan Nieplocha, Jarek and Dhabaleswar K. Panda. High performance remote memory access communication: The armci approach. *The International Journal of High Performance Computing Applications*, 20(2):233–253, 2006.

[163] Amrita Mathuriya, Thorsten Kurth, Vivek Rane, Mustafa Mustafa, Lei Shao, Debbie Bard, Prabhat, and Victor W Lee. Scaling grpc tensorflow on 512 nodes of cori supercomputer, 2017. URL https://arxiv.org/abs/1712.09388.

[164] Nikoli Dryden, Naoya Maruyama, Tim Moon, Tom Benson, Andy Yoo, Marc Snir, and Brian Van Essen. Aluminum: An asynchronous, gpu-aware communication library optimized for large-scale training of deep neural networks on hpc systems. pages 1–13, 11 2018. doi: 10.1109/MLHPC.2018.8638639.

[165] NVIDIA. Nvidia collective communications library (nccl), 2022. URL https://developer.nvidia.com/nccl.

[166] Intel. Intel machine learning scalability library (mlsl), October 2021. URL https://github.com/intel/MLSL.

[167] TensorFlow Developers. Tensorflow. *Zenodo*, 2022.

[168] Nikhil Ketkar, Jojo Moolayil, Nikhil Ketkar, and Jojo Moolayil. Introduction to pytorch. *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*, pages 27–91, 2021.

[169] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2135–2135, 2016.

[170] KONDUIT. Eclipse deeplearning4j (dl4j), 2022. URL https://github.com/deeplearning4j/deeplearning4j.

[171] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient

machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[172] Beng Chin Ooi, Kian-Lee Tan, Sheng Wang, Wei Wang, Qingchao Cai, Gang Chen, Jinyang Gao, Zhaojing Luo, Anthony KH Tung, Yuan Wang, et al. Singa: A distributed deep learning platform. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 685–688, 2015.

[173] Wei Wang, Gang Chen, Haibo Chen, Tien Tuan Anh Dinh, Jinyang Gao, Beng Chin Ooi, Kian-Lee Tan, Sheng Wang, and Meihui Zhang. Deep learning at scale and at ease. *ACM Trans. Multimedia Comput. Commun. Appl.*, 12 (4s), nov 2016. ISSN 1551-6857. doi: 10.1145/2996464. URL https://doi.org/10.1145/2996464.

[174] Jiayi Liu, Jayanta Dutta, Nanxiang Li, Unmesh Kurup, and Mohak Shah. Usability study of distributed deep learning frameworks for convolutional neural networks. In *Deep Learning Day at SIGKDD Conference on Knowledge Discovery and Data Mining*, 2018.

[175] ONNX. Onnx: Open standard for machine learning interoperability, 2022. URL https://github.com/onnx/onnx.

[176] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake Hechtman. Mesh-tensorflow: Deep learning for supercomputers. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 10435–10444, Red Hook, NY, USA, 2018. Curran Associates Inc.

[177] Peter Mendygral, Nick Hill, Krishna Chaitanya Kandalla, Diana Moise, Jacob Balma, and Marcel Schöngens. High performance scalable deep learning with the cray programming environments deep learning plugin. 2018.

[178] Ammar Ahmad Awan, Arpan Jain, Quentin Anthony, Hari Subramoni, and Dhabaleswar K Panda. Hypar-flow: exploiting mpi and keras for scalable hybrid-parallel dnn training with tensorflow. In *International Conference on High Performance Computing*, pages 83–103. Springer, 2020.

[179] Dongsheng Li, Zhiquan Lai, Keshi Ge, Yiming Zhang, Zhaoning Zhang, Qinglin Wang, and Huaimin Wang. Hpdl: Towards a general framework for high-performance distributed deep learning. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1742–1753, 2019. doi: 10.1109/ICDCS.2019.00173.

[180] Markus Gotz, Charlotte Debus, Daniel Coquelin, Kai Krajsek, Claudia Comito, Philipp Knechtges, Bjorn Hagemeier, Michael Tarnawa, Simon Hanselmann, Martin Siggel, Achim Basermann, and Achim Streit. HeAT – a distributed and GPU-accelerated tensor framework for data analytics. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, dec 2020. doi: 10.1109/bigdata50022.2020.9378050. URL https://doi.org/10.1109%2Fbigdata50022.2020.9378050.

[181] S A Jacobs, N Dryden, T Moon, B Van Essen, S He, and J Allen. Scaling deep learning for cancer drug discovery on hpc systems. 2 2018. URL https://www.osti.gov/biblio/1459129.

[182] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

[183] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[184] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[185] Chiheon Kim, Heungsub Lee, Myungryong Jeong, Woonhyuk Baek, Boogeon Yoon, Ildoo Kim, Sungbin Lim, and Sungwoong Kim. torchgpipe: On-the-fly pipeline parallelism for training giant models. *arXiv preprint arXiv:2004.09910*, 2020.

[186] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020.

[187] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 553–564. IEEE, 2017.

[188] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream:

Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.

[189] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. Dapple: A pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 431–445, 2021.

[190] Colin Unger, Zhihao Jia, Wei Wu, Sina Lin, Mandeep Baines, Carlos Efrain Quintero Narvaez, Vinay Ramakrishnaiah, Nirmal Prajapati, Pat McCormick, Jamaludin Mohd-Yusof, et al. Unity: Accelerating {DNN} training through joint optimization of algebraic transformations and parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 267–284, 2022.

[191] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 559–578, 2022.

[192] Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. Galvatron: Efficient transformer training over multiple gpus using automatic parallelism. *arXiv preprint arXiv:2211.13878*, 2022.

[193] K. Kadupitige. *Intersection of hpc and machine learning.* Digital Science Center, 2017.

[194] Á Kerestély. High performance computing for machine learning. 2020.

[195] M. Abspoel, M. E. Scholting, M. Lansbergen, Y. An, and H. Vegter. A new method for predicting advanced yield criteria input parameters from mechanical properties. *Journal of Materials Processing Technology*, 248:161–177, 2017.

[196] V. Jadhao and J. C. S. (2020 Kadupitiya. November). *Integrating machine learning with hpc-driven simulations for enhanced student learning*, pages 25–34, 2020.

[197] A. A. Sekh, I. S. Opstad, R. Agarwal, A. B. Birgisdottir, T. Myrmel, and B. S. Ahluwalia. ... & prasad, d. K. Simulation-supervised deep learning for analysing organelles states and behaviour in living cells. arXiv preprint, 2020.

[198] Fei Wang, Hao Wang, Haichao Wang, Guowei Li, and Guohai Situ. Learning from simulation: An end-to-end deep-learning approach for computational ghost imaging. *Opt.*, 27:25560–25572, 2019.

[199] J. B. Hamrick. Analogues of mental simulation and imagination in deep learning. *Current Opinion in Behavioral Sciences*, 29:8–16, 2019.

[200] S. Partee, M. Ellis, A. Rigazzi, S. Bachman, G. Marques, A. Shao, and B. Robbins. Using machine learning at scale in hpc simulations with smartsim: An application to ocean climate modeling. arxiv. preprint, 2021.

[201] C. Hu, Q. Wu, H. Li, S. Jian, N. Li, and Z. Lou. Deep learning with a long short-term memory networks approach for rainfall-runoff simulation. *Water*, 10 (11):1543, 2018.

[202] K. Yeo and I. Melnyk. Deep learning algorithm for data-driven simulation of noisy dynamical system. *Journal of Computational Physics*, 376:1212–1231, 2019.

[203] M. S. B. Othman and G. (2018 Tan. Machine learning aided simulation of public transport utilization. In *In IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–2. IEEE, october 2018.

[204] B. Wörrlein, S. Bergmann, N. Feldkamp, S. Straßburger, M. Putz, and A. Schlegel. Deep-learning-basierte prognose von stromverbrauch für die hybride simulation. *Simulation in Produktion und Logistik*, pages 121–131, 2019.

[205] B. Moseley, A. Markham, and T. Nissen-Meyer. Fast approximate simulation of seismic waves with deep learning. arxiv. preprint, 2018.

[206] Klaus-Jürgen Bathe. Finite element method. *Wiley encyclopedia of computer science and engineering*, pages 1–12, 2007.

[207] D. Lorente, F. Martínez-Martínez, M. J. Rupérez, M. A. Lago, M. Martínez-Sober, and J Escandell-Montero, P.... & Martín-Guerrero. A framework for modelling the biomechanical behaviour of the human liver during breathing in real time using machine learning. *Expert Systems with Applications*, 71:342–357, 2017.

[208] R. Luo, T. Shao, H. Wang, W. Xu, K. Zhou, and Y. Yang. Deepwarp: Dnn-based nonlinear deformation. preprint, 2018.

[209] J. Kajberg and G. Lindkvist. Characterisation of materials subjected to large strains by inverse modelling based on in-plane displacement fields. *International Journal of Solids and Structures*, 41(13):3439–3459, 2004.

[210] F. Pierron and M. Grédiac. *The virtual fields method: extracting constitutive mechanical parameters from full-field deformation measurements.* Springer, Science & Business Media, 2012.

[211] F. Mathieu, H. Leclerc, F. Hild, and S. Roux. Estimation of elastoplastic parameters via weighted femu and integrated-dic. *Experimental Mechanics*, 55(1): 105–119, 2015.

[212] M. B. Gorji and D. (2019 Mohr. November). towards neural network models for describing the large deformation behavior of sheet metal. *In IOP Conference Series: Materials Science and Engineering*, 651(1).

[213] D. Koch and A. Haufe. An investigation of machine learning capabilities to identify consti-tutive parameters in yield curves. *International Deep Drawing Research Group*, 2019, 2019.

[214] A. M. Chheda, L. Nazro, F. G. Sen, and V. (2019 Hegadekatte. November). prediction of forming limit diagrams using machine learning. *In IOP Conference Series: Materials Science and Engineering (Vol.*, 651(1):012107.

[215] M. Mozaffar, R. Bostanabad, W. Chen, K. Ehmann, J. Cao, and M. A. Bessa. Deep learning predicts path-dependent plasticity. *Proceedings of the National Academy of Sciences*, 116(52):26414–26420, 2019.

[216] F. Barlat, H. Aretz, J. W. Yoon, M. E. Karabin, J. C. Brem, and R. E. Dick. Linear transfomation-based anisotropic yield functions. *International Journal of Plasticity*, 21(5):1009–1039, 2005.

[217] A. G"uner, C. Soyarslan, A. Brosius, and A. E. Tekkaya. Characterization of anisotropy of sheet metals employing inhomogeneous strain fields for yld2000-2d yield function. *International Journal of Solids and Structures*, 49(25):3517–3527, 2012.

[218] R. Caruana. *Multitask learning.* autonomous agents and multi-agent systems., 1998.

[219] M. Crawshaw. Multi-task learning with deep neural networks: A survey. arxiv. preprint, 2020.

[220] C. Darken, J. Chang, and J. Moody. Learning rate schedules for faster stochastic gradient search. *In Neural networks for signal processing*, 2, August 1992.

[221] W. An, H. Wang, Y. Zhang, and Q. Dai. December). exponential decay sine wave learning rate for fast deep neural network training. *In IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4, 2017.

[222] S. W. Chien, S. Markidis, C. P. Sishtla, L. Santos, P. Herman, S. Narasimhamurthy, and E. (2018 Laure. Characterizing deep-learning i/o workloads in tensorflow. *In 2018 IEEE/ACM 3rd International Workshop on Parallel*

Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS), 3: 54–63, November 2018.

[223] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1946–1956, 2019.

[224] Maria Schuld and Nathan Killoran. Quantum machine learning in feature hilbert spaces. *Physical review letters*, 122(4):040504, 2019.

[225] Alexandr A Ezhov and Dan Ventura. Quantum neural networks. *Future Directions for Intelligent Systems and Information Sciences: The Future of Speech and Image Technologies, Brain Computers, WWW, and Bioinformatics*, pages 213–235, 2000.

[226] Jarrod R McClean, Sergio Boixo, Vadim N Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature communications*, 9(1):4812, 2018.

[227] Arthur Pesah, Marco Cerezo, Samson Wang, Tyler Volkoff, Andrew T Sornborger, and Patrick J Coles. Absence of barren plateaus in quantum convolutional neural networks. *Physical Review X*, 11(4):041011, 2021.

[228] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Physical Review A*, 103(3):032430, 2021.

[229] Manuela Weigold, Johanna Barzen, Frank Leymann, and Marie Salm. Expanding data encoding patterns for quantum algorithms. In *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, pages 95–101, 2021. doi: 10.1109/ICSA-C52384.2021.00025.

[230] Manuela Weigold, Johanna Barzen, Frank Leymann, and Marie Salm. Data encoding patterns for quantum computing. In *Proceedings of the 27th Conference on Pattern Languages of Programs*, pages 1–11, 2020.

[231] Frank Leymann and Johanna Barzen. The bitter truth about gate-based quantum algorithms in the nisq era. *Quantum Science and Technology*, 5(4):044007, 2020.

[232] Eric R Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia. *Programming Quantum Computers: essential algorithms and code samples.* O'Reilly Media, 2019.

[233] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, pages 759–766, 2007.

[234] Andrea Mari, Thomas R Bromley, Josh Izaac, Maria Schuld, and Nathan Killoran. Transfer learning in hybrid classical-quantum neural networks. *Quantum*, 4:340, 2020.

[235] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[236] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[237] Korn Sooksatra, Pablo Rivas, and Javier Orduz. Evaluating accuracy and adversarial robustness of quanvolutional neural networks. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 152–157, 2021. doi: 10.1109/CSCI54926.2021.00097.

[238] CM Wilson, JS Otterbach, Nikolas Tezak, RS Smith, AM Polloreno, Peter J Karalekas, S Heidel, M Sohaib Alam, GE Crooks, and MP da Silva. Quantum kitchen sinks: An algorithm for machine learning on near-term quantum computers. *arXiv preprint arXiv:1806.08321*, 2018.

[239] Iris Cong, Soonwon Choi, and Mikhail D Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, 2019.

[240] Guillaume Verdon, Jacob Marks, Sasha Nanda, Stefan Leichenauer, and Jack Hidary. Quantum hamiltonian-based models and the variational quantum thermalizer algorithm. *arXiv preprint arXiv:1910.02071*, 2019.

[241] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Scharwächter, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset. In *CVPR Workshop on the Future of Datasets in Vision*, volume 2. sn, 2015.

[242] Laurie Rich. Cisco visual networking index, 2021. URL https://www.cisco.com/c/dam/global/pt_br/assets/docs/whitepaper_VNI_06_09.pdf.

[243] Shinobu Kudo, Shota Orihashi, Ryuichi Tanida, Seishi Takamura, and Hideaki Kimata. Gan-based image compression using mutual information for optimizing subjective image similarity. *IEICE TRANSACTIONS on Information and Systems*, 104(3):450–460, 2021.

[244] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

[245] Aaditya Prakash, James Storer, Dinei Florencio, and Cha Zhang. Repr: Improved training of convolutional filters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10666–10675, 2019.

[246] Fabian Mentzer, George D Toderici, Michael Tschannen, and Eirikur Agustsson. High-fidelity generative image compression. *Advances in Neural Information Processing Systems*, 33:11913–11924, 2020.

[247] Lirong Wu, Kejie Huang, and Haibin Shen. A gan-based tunable image compression system. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2334–2342, 2020.

[248] Jonas Löhdefink, Andreas Bär, Nico M Schmidt, Fabian Hüger, Peter Schlicht, and Tim Fingscheidt. Gan-vs. jpeg2000 image compression for distributed automotive perception: Higher peak snr does not mean better semantic segmentation. *arXiv preprint arXiv:1902.04311*, 2019.

[249] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

[250] Michael W Marcellin, Michael J Gormish, Ali Bilgin, and Martin P Boliek. An overview of jpeg-2000. In *Proceedings DCC 2000. Data Compression Conference*, pages 523–541. IEEE, 2000.

[251] Zhanjun Si and Ke Shen. Research on the webp image format. In *Advanced graphic communications, packaging technology and materials*, pages 271–277. Springer, 2016.

[252] Xi Zhang and Xiaolin Wu. Near-lossless l-infinity constrained multi-rate image decompression via deep neural network. *CoRR*, 2018.

[253] Byeongkeun Kang, Subarna Tripathi, and Truong Q Nguyen. Toward joint image generation and compression using generative adversarial networks. *arXiv preprint arXiv:1901.07838*, 2019.

[254] Jingkuan Song, Tao He, Lianli Gao, Xing Xu, Alan Hanjalic, and Heng Tao Shen. Unified binary generative adversarial network for image retrieval and compression. *International Journal of Computer Vision*, 128:2243–2264, 2020.

[255] Boyu Wang, Kevin Yager, Dantong Yu, and Minh Hoai. X-ray scattering image classification using deep learning. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 697–704. IEEE, 2017.

[256] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. *arXiv preprint arXiv:1911.03584*, 2019.

[257] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[258] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Zhicheng Yan, Masayoshi Tomizuka, Joseph Gonzalez, Kurt Keutzer, and Peter Vajda. Visual transformers: Token-based image representation and processing for computer vision. *arXiv preprint arXiv:2006.03677*, 2020.

[259] Michael Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: Adaptive space-time tokenization for videos. *Advances in Neural Information Processing Systems*, 34:12786–12797, 2021.

[260] Lei Zhang, Xun Wang, Nicholas Penwarden, and Qiang Ji. An image segmentation framework based on patch segmentation fusion. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 2, pages 187–190. Ieee, 2006.

[261] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[262] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[263] Alain M Leger, Takao Omachi, and Gregory K Wallace. Jpeg still picture compression algorithm. *Optical Engineering*, 30(7):947–954, 1991.

[264] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. *Advances in neural information processing systems*, 31, 2018.

[265] Bing Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[266] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[267] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411:109409, 2020.

[268] Matthias Eichinger, Alexander Heinlein, and Axel Klawonn. Stationary flow predictions using convolutional neural networks. In *Numerical Mathematics and Advanced Applications ENUMATH 2019: European Conference, Egmond aan Zee, The Netherlands, September 30-October 4*, pages 541–549. Springer, 2020.

[269] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[270] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[271] Maria Schuld, Alex Bocharov, Krysta M Svore, and Nathan Wiebe. Circuit-centric quantum classifiers. *Physical Review A*, 101(3):032308, 2020.

[272] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. Quantum generative adversarial networks for learning and loading random distributions. *npj Quantum Information*, 5(1):103, 2019.

[273] Jonathan Romero, Jonathan P Olson, and Alan Aspuru-Guzik. Quantum autoencoders for efficient compression of quantum data. *Quantum Science and Technology*, 2(4):045001, 2017.

[274] Vedran Dunjko and Hans J Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics*, 81(7):074001, 2018.

[275] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002*, 2018.

[276] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. Quantum deep learning. *arXiv preprint arXiv:1412.3489*, 2014.

[277] Kwok Ho Wan, Oscar Dahlsten, Hlér Kristjánsson, Robert Gardner, and MS Kim. Quantum generalisation of feedforward neural networks. *npj Quantum information*, 3(1):36, 2017.

[278] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019.

[279] Farrokh Vatan and Colin Williams. Optimal quantum circuits for general two-qubit gates. *Physical Review A*, 69(3):032315, 2004.

[280] Stephen S Bullock and Igor L Markov. An arbitrary twoqubit computation in 23 elementary gates or less. In *Proceedings of the 40th Annual Design Automation Conference*, pages 324–329, 2003.

[281] Parviz Ghadimi, Abbas Dashtimanesh, and Hossein Hosseinzadeh. Solution of poisson's equation by analytical boundary element integration. *Applied Mathematics and Computation*, 217(1):152–163, 2010.

[282] Selçuk Yıldırım. Exact and numerical solutions of poisson equation for electrostatic potential problems. *Mathematical problems in engineering*, 2008, 2008.

[283] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, F Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, et al. Qiskit: An open-source framework for quantum computing. *Accessed on: Mar*, 16, 2019.

[284] Maximilian Schlosshauer. Quantum decoherence. *Physics Reports*, 831:1–57, 2019.