



HAL
open science

Acceleration of Numerical Simulations with Deep Learning: Application to Thermodynamic Equilibrium Calculations

Jingang Qu

► **To cite this version:**

Jingang Qu. Acceleration of Numerical Simulations with Deep Learning: Application to Thermodynamic Equilibrium Calculations. Systems and Control [cs.SY]. Sorbonne Université, 2023. English. NNT: 2023SORUS530 . tel-04630096

HAL Id: tel-04630096

<https://theses.hal.science/tel-04630096v1>

Submitted on 1 Jul 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Thèse de doctorat de Sorbonne Université

Spécialité — Informatique

École Doctorale Informatique, Télécommunications et Électronique (Paris)

ACCELERATION OF NUMERICAL SIMULATIONS
WITH DEEP LEARNING

Application to Thermodynamic Equilibrium Calculations

JINGANG QU

Pour obtenir le grade de
docteur de Sorbonne Université

devant le jury composé de :

M. Massih-Reza AMINI	Université Grenoble-Alpes	Rapporteur
M. Wei YAN	Université technique du Danemark	Rapporteur
Mme. Mathilde MOUGEOT	ENS Paris-Saclay	Examinatrice
M. Pascal MORIN	Sorbonne Université	Examineur
M. Fabian JIRASEK	Technische Universität Kaiserslautern	Examineur
M. Thibault FANEY	IFP Energies nouvelles	Examineur
M. Jean-Charles DE HEMPTINNE	IFP Energies nouvelles	Examineur
M. Patrick GALLINARI	Sorbonne Université	Directeur de thèse

ABSTRACT

Numerical simulations are a powerful tool for analyzing dynamic systems, but they can be computationally expensive and time-consuming for complex systems with high resolution. Over the past decades, researchers have been striving to accelerate numerical simulations through algorithmic improvements and high-performance computing (HPC). More recently, artificial intelligence (AI) for science is on the rise and involves using AI techniques, specifically machine learning and deep learning, to solve scientific problems and accelerate numerical simulations, having the potential to revolutionize a wide range of fields.

The primary goal of this thesis is to speed up thermodynamic equilibrium calculations by means of techniques used to accelerate numerical simulations. Thermodynamic equilibrium calculations are able to identify the phases of mixtures and their compositions at equilibrium and play a pivotal role in many fields, such as chemical engineering and petroleum industry. We achieve this goal in two aspects. On the one hand, we use deep learning frameworks to rewrite and vectorize algorithms involved in thermodynamic equilibrium calculations, facilitating the use of diverse hardware for HPC. On the other hand, we use neural networks to replace time-consuming and repetitive subroutines of thermodynamic equilibrium calculations, which is a widely adopted technique of AI for science.

Another focus of this thesis is to address the challenge of domain generalization (DG) in image classification. DG involves training models on known domains that can effectively generalize to unseen domains, which is crucial for deploying models in safety-critical real-world applications. DG is an active area of research in deep learning. Although various DG methods have been proposed, they typically require domain labels and lack interpretability. Therefore, we aim to develop a novel DG algorithm that does not require domain labels and is more interpretable.

RÉSUMÉ

Les simulations numériques sont un outil puissant pour analyser les systèmes dynamiques, mais peuvent être coûteuses en termes de calcul et consommatrices de temps pour les systèmes complexes à haute résolution. Au cours des dernières décennies, les chercheurs ont cherché à accélérer les simulations numériques grâce à des améliorations algorithmiques et à l'informatique haute performance (HPC). Plus récemment, l'intelligence artificielle (IA) pour la science est en plein essor et implique l'utilisation de techniques d'IA, en particulier l'apprentissage automatique et l'apprentissage profond, pour résoudre des problèmes scientifiques et accélérer les simulations numériques, ayant le potentiel de révolutionner un large éventail de domaines.

L'objectif principal de cette thèse est d'accélérer les calculs d'équilibre thermodynamique au moyen de techniques utilisées pour accélérer les simulations numériques. Les calculs d'équilibre thermodynamique permettent d'identifier les phases des mélanges et leurs compositions à l'équilibre et jouent un rôle pivot dans de nombreux domaines, tels que le génie chimique et l'industrie pétrolière. Nous atteignons cet objectif sous deux aspects. D'une part, nous utilisons des cadres d'apprentissage profond pour réécrire et vectoriser les algorithmes impliqués dans les calculs d'équilibre thermodynamique, facilitant l'utilisation de divers matériels pour le HPC. D'autre part, nous utilisons des réseaux de neurones pour remplacer les sous-programmes répétitifs et chronophages des calculs d'équilibre thermodynamique, ce qui est une technique largement adoptée par l'IA pour la science.

Un autre axe de cette thèse est de relever le défi de la généralisation de domaine (DG) en classification d'images. DG implique de former des modèles sur des domaines connus qui peuvent généraliser efficacement à des domaines inconnus, ce qui est crucial pour déployer des modèles dans des applications réelles critiques en matière de sécurité. DG est un domaine de recherche actif dans l'apprentissage profond. Bien que diverses méthodes de DG aient été proposées, elles nécessitent généralement des étiquettes de domaine et manquent d'interprétabilité. Par conséquent, nous visons à développer un nouvel algorithme de DG qui ne nécessite pas d'étiquettes de domaine et est plus interprétable.

ACKNOWLEDGMENTS

As I look back on my PhD journey, I realize that there are so many people to thank, without whom this journey would not have been possible. In particular, I started my PhD in the midst of the unprecedented global turmoil brought on by the COVID-19 pandemic.

To begin with, I owe my deepest gratitude to my advisor, Patrick, who opened the door to the realm of deep learning for me. Despite the fact that my background in undergraduate and master's studies was not directly related to deep learning, Patrick did not hesitate to engage with me enthusiastically when I expressed my desire to pursue my PhD research under his guidance. Although my PhD subject did not align precisely with Patrick's main work, Patrick's guidance kept me focused and steered me clear of unnecessary detours. For this, I am profoundly thankful.

I extend my heartfelt appreciation to my mentors at IFPEN: Thibault, Jean-Charles, and Soleiman. Having interned at IFP for half a year before my PhD, I had the privilege of getting acquainted with them. Thibault has been both a mentor and a friend to me, always lending a helping hand just when I needed it, not only in academics but also in life. His weekly check-ins and discussions about the challenges I faced have been invaluable. I am deeply appreciative of his patience, especially when I struggled to express myself in French. Despite my limited exposure to thermodynamics before my PhD, Jean-Charles graciously guided me through this domain, explained complex thermodynamic concepts, and entrusted me with the use of the IFPEN-developed Carnot. Without his assistance, I would have struggled to progress in my work. Although I didn't have many interactions with Soleiman, he never hesitated to share necessary advice and assistance, like helping me polish my papers for publication.

I would like to express my gratitude to each member of the jury. I am grateful to Massih-Reza and Wei for their time and expertise in reviewing my thesis. Massih-Reza is an expert in machine learning and deep learning. I once attended a summer school organized by DTU, where Wei happened to deliver lectures on phase equilibrium calculations. His lectures enriched my understanding of thermodynamics. Pascal is the PhD advisor of my best friend, Ze WANG. I am honored to be involved in Ze's work and to collaborate with Pascal. Fabian's work is dedicated to the integration of machine learning into thermodynamics. I had the opportunity to hear Fabian's presentation at a conference on how to integrate machine learning models into thermodynamic models, which nicely inspired my work. Mathilde is also an expert in deep learning with a broad range of research interests, such as transfer learning and physics-informed deep learning.

I would be remiss if I did not mention the invaluable support of my girlfriend, Yang YANG. Meeting in 2014 and moving together from China to France in 2016, Yang has always been by my side, encouraging, understanding, and caring for me. I am grateful for her unflinching support of my ideas and decisions.

Finally, to my parents, Zhu LI and Shifeng QU, whom I have not seen for more than three years due to the pandemic. The longing for them was perhaps the hardest part of my PhD journey. Thank you for their unwavering support and encouragement, even from thousands of miles away.

Thank you, each and every one of you.

CONTENTS

1	Introduction	1
1.1	Current techniques for accelerating numerical simulations	1
1.2	Thesis objectives	2
1.2.1	Acceleration of thermodynamic equilibrium calculations	2
1.2.2	Efficient learning of heterogeneous patterns in data	4
1.3	Thesis organization	4
I	Preliminaries	
2	Fundamentals of applied thermodynamics and phase equilibrium calculations	9
2.1	Basic concepts	9
2.1.1	System, state and phase	9
2.1.2	Thermodynamic properties	9
2.1.3	Residual properties	10
2.2	Phase equilibrium	11
2.3	Equations of state	12
2.3.1	Cubic equations of state	13
2.3.2	Statistical Associating Fluid Theory	15
2.3.3	Cubic Plus Association equation of state	16
2.4	Isothermal two-phase flash calculation	17
2.4.1	Problem setting	17
2.4.2	Stability analysis	18
2.4.3	Phase split calculations	19
2.4.4	Strategy for the isothermal two-phase flash calculation	21
3	Neural networks and deep learning	23
3.1	Feedforward neural networks	23
3.2	Universal approximation theorem	25
3.3	Training of neural networks	26
3.3.1	Cost function	27
3.3.2	Optimization algorithms	28
3.4	Hyper-parameter tuning	29
II	Speeding up phase equilibrium calculations with deep learning	
4	PTFlash : A deep learning framework for two-phase flash calculation	33
4.1	Introduction	33
4.2	Related work	34
4.3	Data generation	34
4.3.1	Design of experiments (DoE)	34
4.3.2	A new method for sampling multiple variables adding up to 1	35
4.4	Case studies	37
4.5	Vectorization of two-phase flash calculation	39
4.6	Acceleration of flash calculation using neural networks	41
4.6.1	Classifier	42
4.6.2	Initializer	45
4.6.3	Strategy for accelerating flash calculation using neural networks	46

4.7	Results	47	
4.7.1	Vectorized flash calculation	47	
4.7.2	Deep-learning-powered vectorized flash calculation	49	
4.7.3	Discussion	51	
4.8	Conclusion	52	
5	NNEoS : Neural network-based EoS to calculate fugacity coefficients	53	
5.1	Introduction	53	
5.2	Related work	54	
5.2.1	Replace numerical EoS with machine learning models	54	
5.2.2	Regression Clustering and Mixture of Experts	55	
5.3	Case study	56	
5.4	Analysis of the discontinuity of fugacity coefficients	56	
5.5	Clustered regression network	57	
5.6	Neural network-based equation of state	59	
5.7	Results	62	
5.7.1	Comparison between Carnot and NNEoS	63	
5.7.2	Comparison between Carnot and PTFlash using NNEoS	64	
5.7.3	HybridEoS to combine Carnot and NNEoS	65	
5.7.4	Discussion	67	
5.8	Conclusion	68	
III Domain generalization			
6	HMOE: Hypernetwork-based Mixture of Experts for Domain Generalization	71	
6.1	Introduction	71	
6.2	Related work	73	
6.2.1	Domain generalization (DG)	73	
6.2.2	Hypernetworks	73	
6.2.3	Mixture of Experts (MoE)	73	
6.2.4	Application of hypernetworks and MoE in DG	74	
6.3	Method	74	
6.3.1	Problem setting	74	
6.3.2	Overall architecture	75	
6.3.3	Hypernetworks	76	
6.3.4	Routing mechanism	76	
6.3.5	Embedding space	77	
6.3.6	Class-adversarial training on D2V	77	
6.3.7	Semi-/supervised learning on domains	78	
6.3.8	Training and inference	78	
6.4	Toy regression problem	78	
6.5	DomainBed	80	
6.5.1	Datasets and model evaluation	80	
6.5.2	Implementation details	80	
6.5.3	Results	81	
6.5.4	Latent domain discovery	83	
6.5.5	Ablation study	84	
6.5.6	More empirical analysis	86	
6.6	Conclusion	87	
7	Conclusion and perspectives	89	

7.1	Conclusion	89
7.2	Perspectives	89
7.2.1	Improve the generalization of neural networks to components	89
7.2.2	A more interpretable method for domain generalization	90
IV	Appendix	
A	Derivation of the closed-form expression of the probability density function	95
B	Detailed domain generalization results	97
	Bibliography	103

LIST OF FIGURES

Figure 1.1	Relationship between artificial intelligence, machine learning and deep learning	2
Figure 1.2	Two problems of learning heterogeneous patterns	5
Figure 2.1	Formation of a molecule in the original SAFT EoS	16
Figure 2.2	Successive substitution of phase split calculations	20
Figure 2.3	Flowchart of the isothermal two-phase flash calculation	21
Figure 3.1	Representation of an artificial neuron	24
Figure 3.2	Commonly used activation functions	24
Figure 3.3	Architecture of feedforward neural networks	25
Figure 3.4	Loss landscape of ResNet-56 without shortcut connections for the CIFAR-10 dataset	27
Figure 3.5	Illustration of underfitting, overfitting and good fit	28
Figure 4.1	Comparison between random sampling and Latin Hypercube Sampling	35
Figure 4.2	Sampling the composition using the simple approach	36
Figure 4.3	The Dirichlet distribution with respect to its concentration parameters	37
Figure 4.4	Phase envelopes of four typical reservoir fluids	39
Figure 4.5	Marginal distribution of the molar fraction of component i for black oil	40
Figure 4.6	Architecture of the classifier for the case study containing 9 components	43
Figure 4.7	Hyper-parameter tuning of the classifier	43
Figure 4.8	Cyclic learning rate used to train the classifier	44
Figure 4.9	Contours of the probabilities predicted by the classifier	44
Figure 4.10	Architecture of the initializer	45
Figure 4.11	Acceleration of flash calculation using neural networks	47
Figure 4.12	Comparison between Carnot and PTFlash	48
Figure 4.13	Convergence percentage and running time of each subroutine of PTFlash on GPU	50
Figure 4.14	Closeness to critical points in the execution of PTFlash	50
Figure 5.1	Phase diagram of the mixture of water and methane	56
Figure 5.2	Fugacity coefficients for the mixture of water and methane at different pressures	57
Figure 5.3	Architecture of clustered regression networks	58
Figure 5.4	Sigmoid activation function and its gradient	58
Figure 5.5	Application of CRNet to a toy regression problem	59
Figure 5.6	Four clustering algorithms for the toy regression problem	60
Figure 5.7	Architecture of NNEoS	61
Figure 5.8	Cyclic learning rate schedule to train Net_Z	61
Figure 5.9	Training losses of Net_Z	62
Figure 5.10	The prediction of Net_Z and the outputs of its key parts at $P = 1\text{MPa}$	62
Figure 5.11	Training loss of Net_φ	63

Figure 5.12	Comparison between the true and predicted $\ln \varphi$ at $P = 1\text{MPa}$	63
Figure 5.13	Comparison between the execution time of Carnot and NNEoS for calculating fugacity coefficients	64
Figure 5.14	Comparison between the execution time of Carnot and PTFlash using HybridEoS	67
Figure 6.1	Diagram of classical Mixture of Experts	74
Figure 6.2	Overview of the architecture of HMOE	75
Figure 6.3	A toy regression problem to evaluate HMOE	79
Figure 6.4	Comparison between three inference modes of HMOE	79
Figure 6.5	Average of supervised domain loss over all test domains for each dataset of DomainBed	83
Figure 6.6	The t-SNE visualization of the output of the D2V encoder of HMOE	84
Figure 6.7	Comparison between domain labels and HMOE clusters	85
Figure 6.8	Soft partitioning of HMOE for OfficeHome	85
Figure 6.9	Learning collapse for PACS with $K = 8$	86
Figure 6.10	Train HMOE using OOD for PACS	86
Figure 7.1	Construction of graph data to consider the properties of components	90
Figure 7.2	Prototype of a VAE-based DG method	91

LIST OF TABLES

Table 2.1	State variables expressed as the partial derivatives of energy functions	11
Table 2.2	Relationships between different residual properties	11
Table 2.3	Estimation of the energy and co-volume parameters of pure compounds for cubic EoS	14
Table 3.1	Weight initialization methods for neural networks with different activation functions	29
Table 4.1	Properties of the components involved in three case studies used to evaluate PTFlash	37
Table 4.2	Four fluid types characterized by different compositional ranges	38
Table 4.3	Some typical reservoir fluid compositions	38
Table 4.4	Concentration parameters of the Dirichlet distribution used to sample different fluid types	39
Table 4.5	Performance profiler of PTFlash on GPU	49
Table 4.6	Performance profiler of NN-PTFlash on GPU	51
Table 5.1	Comparison between the results of Carnot and PTFlash using NNEoS	65
Table 5.2	Performance profiler of PTFlash using NNEoS on GPU	66
Table 5.3	Performance profiler of PTFlash using HybridEoS on GPU	66
Table 6.1	Description and visualization of datasets of DomainBed	80
Table 6.2	Domain generalization results on DomainBed	82

Table 6.3	Ablation study for HMOE-DN	85
Table 6.4	Use Swin Transformer as featurizer of HMOE-DN	87
Table B.1	Domain generalization results on ColoredMNIST	97
Table B.2	Domain generalization results on RotatedMNIST	98
Table B.3	Domain generalization results on VLCS	99
Table B.4	Domain generalization results on PACS	100
Table B.5	Domain generalization results on OfficeHome	101
Table B.6	Domain generalization results on TerraIncognita	102

INTRODUCTION

1.1 CURRENT TECHNIQUES FOR ACCELERATING NUMERICAL SIMULATIONS

Numerical simulations of dynamical systems predict their behavior over time by computationally solving mathematical models that represent the underlying dynamics of these systems, e.g., computational fluid dynamics (CFD) [50] to solve Navier-Stokes equations. Numerical simulations are commonly used to test hypotheses, optimize designs, and make decisions across a broad range of fields, including physics, biology, chemistry, engineering, and more. The size and computation time of a numerical simulation can vary significantly depending on the system being studied, the complexity of mathematical models, and the goals of the simulation. For large-scale and complex systems, e.g., weather forecasting, numerical simulations can be particularly time-consuming and computationally expensive. As a result, there is a growing demand to accelerate numerical simulations. Over the past decades, in addition to developing more efficient algorithms, researchers have focused on exploiting the computational power of hardware through high-performance computing (HPC) [43], which involves performing simulations via parallel computing on clusters of central processing units (CPUs) or specialized hardware, such as graphics processing units (GPUs) and tensor processing units (TPUs). In recent years, machine learning [137] and deep learning [104] have become increasingly powerful and achieved impressive results due to the explosive growth of data and dramatically improved computational performance of hardware. This leads to the rise of artificial intelligence (AI) for science [192], which greatly facilitates and accelerates numerical simulations. As a result, the effective utilization of hardware and the application of AI play a crucial role in speeding up numerical simulations nowadays.

The main types of computing hardware used in numerical simulations include CPUs, GPUs, and TPUs. GPUs are commonly used as general-purpose hardware accelerators and are capable of performing a wide range of arithmetic operations in parallel. TPUs, developed by Google, are specifically designed to accelerate deep learning tasks, such as training neural networks, and are particularly proficient in dense matrix multiplication and linear algebra-intensive calculations. To use hardware from different manufacturers, researchers can implement numerical simulation programs using corresponding development tools, such as CUDA [170] for NVIDIA GPUs and ROCm [146] for AMD GPUs. However, this necessitates frequent modifications to programs in order to meet the requirements of different hardware. An emerging alternative approach is to use deep learning frameworks, such as Tensorflow [1], PyTorch [150] and JAX [19], to implement numerical simulations, enabling more flexibility in using and switching between different hardware. For instance, JAX has been used in a number of works for CFD [15, 97, 224], finite element method [224], ocean modeling [71], molecular dynamics [174], and quantum many-body dynamics [138]. In addition, [216] uses Tensorflow to model fluid flows, with a focus on using TPUs. [79] presents a differentiable Eulerian PDE framework for a large class of PDE families with support for Tensorflow, PyTorch, and JAX.

Deep learning frameworks offer several advantages for numerical simulations. First, they provide not only high-level functionalities for building and training neural networks

but also low-level primitives for scientific computing, such as linear algebra toolkits. This facilitates the development of programs for a range of numerical simulation problems. Second, they support various hardware internally and offer a unified interface externally, enabling the use of different hardware without the need for code modification. Third, they support automatic differentiation (AD), which is beneficial for many numerical simulations involving ordinary differential equations (ODEs) or partial differential equations (PDEs). AD also allows for end-to-end optimization since entire numerical simulations are differentiable. Last but not least, they enable the seamless integration of deep learning models into numerical simulations.

In recent years, AI for science has been on the rise and has achieved some breakthroughs, such as the accurate and fast prediction of protein structures [90] and the discovery of more efficient matrix multiplication algorithms [47]. The powerful expressive ability of machine learning and deep learning models has been a major driving factor in these breakthroughs. The relationship between AI, machine learning, and deep learning is shown in Fig. 1.1.

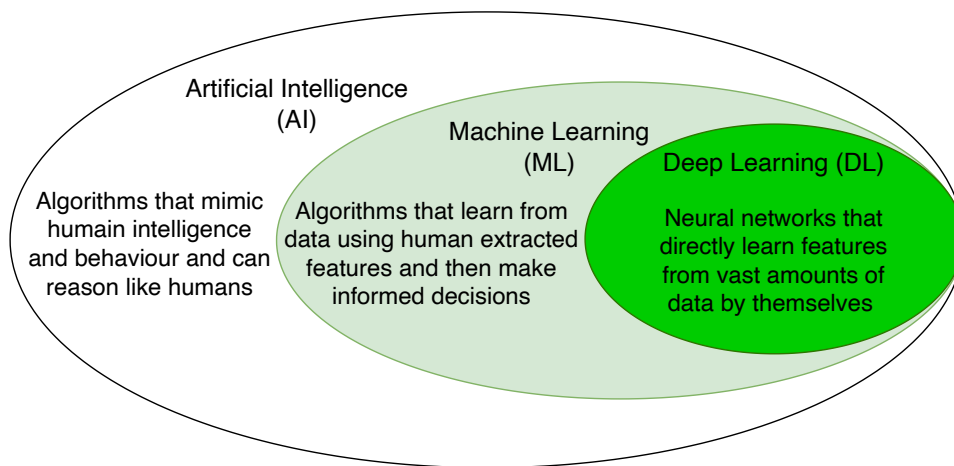


Figure 1.1: Relationship between artificial intelligence, machine learning and deep learning

To speed up numerical simulations of governing dynamics represented as ODEs or PDEs, several deep learning methods are commonly utilized, including data-driven neural networks (DDNNs), physical-informed neural networks (PINNs) [161, 162], neural operators [117, 125], and NeuralODE [26]. DDNNs are trained on data generated by numerical simulators and can make quick predictions. PINNs incorporate ODEs or PDEs into their training as regularization terms. Neural operators use neural networks to solve a family of ODEs or PDEs by learning the mapping between functions (i.e., operators). Lastly, NeuralODE is capable of learning unknown governing ODEs from data.

1.2 THESIS OBJECTIVES

1.2.1 Acceleration of thermodynamic equilibrium calculations

The primary aim of this thesis is to apply the techniques mentioned earlier for accelerating numerical simulations to thermodynamic equilibrium calculations. More specifically, we focus on phase equilibrium calculations, also known as flash calculation. The natural world abounds with mixtures composed of various chemical substances, such as the air we breathe and the water we drink. The phenomenon of multiphase equilibrium is constantly occurring. When different phases of a mixture, such as liquid and vapor, come into contact,

they exchange substances with each other (mass transfer) driven by chemical potential differences. This exchange persists until the different phases reach equilibrium and their compositions no longer change.

At equilibrium, the compositions of different phases tend to vary significantly, allowing us to separate mixtures and purify substances through various operations such as distillation, extraction, absorption, adsorption, leaching, and crystallization. These operations are essential to the fields of chemistry and chemical engineering, highlighting the importance of flash calculation. Moreover, flash calculation has a wide range of applications in other areas. In the petroleum industry, it is used to anticipate the behavior of petroleum reservoirs [33]. In environmental science, it is employed to predict the partitioning of volatile organic compounds between different phases in order to design and optimize the removal of pollutants from the environment [29]. In materials science, it plays a critical role in comprehending the properties and processes of materials by predicting their phase diagrams and phase transitions [3].

For mixtures containing dozens of components, a single flash calculation can be performed in milliseconds. However, in compositional reservoir simulation and multiphase flow simulation, the whole system of interest is discretized into a large number of small blocks. It is necessary to determine the number and type of equilibrium phases for each block at each time step through flash calculations. When dealing with millions of blocks, the time required for flash calculations can become computationally prohibitive and therefore constitutes a significant bottleneck in the simulation process. Consequently, there is a growing need to accelerate flash calculations.

To perform a flash calculation for a mixture of a given composition, two process variables should be specified, such as temperature T , pressure P , volume V , enthalpy H , and entropy S . The selection of these process variables has a significant impact on the complexity of flash calculation [133]. In this thesis, we will focus on flash calculation at specified T and P , also referred to as the isothermal flash calculation [129, 130]. The isothermal flash ensures a unique solution that corresponds to the global minimum of the Gibbs energy. Compared to other specifications, the isothermal flash is relatively straightforward to implement and has been the subject of extensive research. Robust and efficient algorithms have been developed for the isothermal flash based on the pioneering work [129, 130]. Additionally, the isothermal flash can be used to perform flash calculations of alternative specifications, such as the isenthalpic flash at specified (P, H) and the isentropic flash at specified (P, S) . This is achieved by using a nested-loop approach that uses the isothermal flash in the inner loop and adjusts P or T in the outer loop [132]. Although this nested-loop approach is less efficient than the concurrent convergence of all independent variables using Newton's method, it is safer and serves as a fallback when Newton's method fails [136]. Moreover, P and T are frequently given as conditions in many flash calculation applications, making the isothermal flash more practical.

In contrast to numerical simulations for solving ODEs and PDEs, the isothermal flash calculation serves as a building block for other simulations, such as compositional reservoir simulation and multiphase flow simulation. It involves finding the global minimum of the Gibbs energy through an optimization process, which entails substantial knowledge of thermodynamics. With the help of the acceleration techniques discussed in the previous section and by taking into account the characteristics of the isothermal flash, we will use deep learning frameworks to implement the isothermal flash for parallel computing on GPUs. Additionally, we will employ neural networks to replace repetitive and time-consuming subroutines in order to speed up the isothermal flash.

1.2.2 Efficient learning of heterogeneous patterns in data

In addition to accelerating flash calculation, we also aim to address an academic problem in computer vision, namely domain generalization.

When using neural networks to replace the time-consuming subroutines of flash calculation, we face the challenge of learning piecewise continuous functions, i.e., discontinuous functions, as depicted in Fig. 1.2a. This discontinuity leads to the presence of multiple underlying functions that need to be learned from the data. We refer to this problem as the learning of heterogeneous patterns. In machine learning and deep learning tasks, heterogeneity commonly arises from the distribution shift [11, 159]. In supervised learning, it is typically assumed that both the training and test sets consist of independent and identically distributed samples from the same distribution. However, this assumption does not hold in many real-world scenarios, such as when data is collected from multiple sources (e.g., sensors, databases) and multiple modalities (e.g., text, image, audio, video).

From the perspective of causality, the joint distribution of the input X and target Y can be factored as $P_{X,Y}(x, y) = P_X(x) P_{Y|X}(y|x)$. The distribution shift is often manifested as either the domain shift (also known as the covariate shift) or the concept shift [128, 159, 193]. The concept shift is caused by the discrepancies in $P_{Y|X}(y|x)$, suggesting that the functional mapping between the input and output is inconsistent across the data. Therefore, the problem of learning discontinuous functions is more likely associated with the concept shift. The domain shift is due to the differences in $P_X(x)$, indicating that input data is drawn from multiple different distributions. To address the domain shift, a variety of techniques have been proposed and developed, such as multi-task learning [22], transfer learning [148, 218, 240], meta-learning [82, 203, 206], domain adaptation [151, 214], and domain generalization [68, 213, 236].

In this thesis, we focus on domain generalization, which involves training models capable of effectively generalizing to unseen domains. In real-world scenarios, it is often impractical to collect a large and diverse dataset that covers all possible domains. Consequently, models may overfit to the specific patterns of the training domains, making it challenging to generalize well to test domains different from the training domains. For example, a model may predict an image to be a cow or camel mostly due to the presence of green pasture or yellow desert rather than based on animal attributes. Domain generalization is particularly important for safety-critical real-world applications, such as medical diagnosis and autonomous driving. In medical diagnosis, models need to generalize across different hospitals, which may have distinct data collection processes, patient populations, and medical facilities. Similarly, in autonomous driving, models should be able to generalize across different cities, where road conditions, traffic patterns, and weather may vary significantly. A concrete example of domain generalization is provided in Fig. 1.2b.

1.3 THESIS ORGANIZATION

This thesis consists of three parts:

- Part I - **Preliminaries** (Chapters 2 and 3). This part covers the basic concepts of phase equilibrium calculations and deep learning, which pave the way for the work in later chapters.

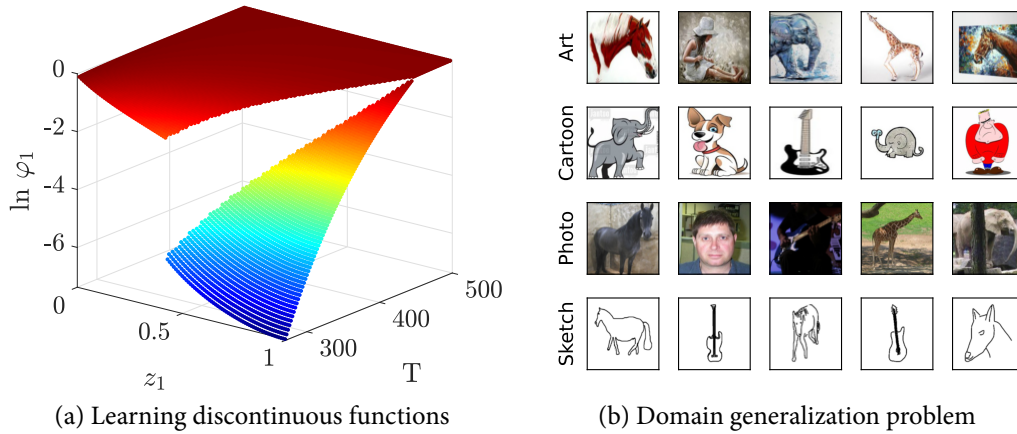


Figure 1.2: These figures show two problems of learning heterogeneous patterns, which are considered in this thesis. Figure (a) depicts the problem of learning discontinuous functions, as exemplified by the logarithmic fugacity coefficient of water $\ln \varphi_1$ in the mixture of water and methane with respect to the molar fraction of water z_1 and temperature T at fixed pressure $P = 1\text{MPa}$. Figure (b) represents one example of the domain shift — domain generalization and shows some images of the PACS dataset [109] consisting of four distinct styles: sketch, photo, cartoon, and art. One style is used as the test domain, and the others are used for training.

Chapter 2: We first introduce the fundamentals of thermodynamics, thermodynamic notation, phase equilibrium conditions, and equations of state (EoS), and then detail how to perform the isothermal two-phase flash calculation.

Chapter 3: We use feedforward neural networks as an example to introduce how to build and train neural networks, and show some commonly used techniques to improve the performance of neural networks.

- Part II - **Speeding up phase equilibrium calculations** (Chapters 4 and 5). This part is the core of the thesis, that is, the application of deep learning techniques to accelerate phase equilibrium calculations.

Chapter 4: We present PTFash, a complete rewrite of the algorithms involved in the isothermal two-phase flash calculation with the Soave-Redlich-Kwong (SRK) EoS [187] using the deep learning framework PyTorch. Implementing algorithms using PyTorch is essentially the process of vectorization, which is a technique used to optimize and accelerate the execution of numerical algorithms by performing operations (e.g., matrix multiplication) on multiple data elements in parallel rather than sequentially. This helps reduce the number of instructions and leverage the parallelism of modern hardware, especially GPUs. However, vectorizing algorithms is challenging because it requires replacing each operation with the corresponding vectorized one, which is further complicated by the intricate control flow of flash calculation. Our experiments demonstrate that PTFash achieves substantial speedups due to parallel computing on GPUs.

Chapter 5: We focus on equations of state, which are the key element of flash calculation. To be more specific, we use neural networks as substitutes for numerical EoS to calculate fugacity coefficients that play a critical role in flash calculation. These neural networks are referred to as NNEoS, which is short for neural-network based equations of state. The goal of NNEoS is to handle complex mixtures (e.g., those involving hydrogen bonding)

using more advanced EoS without vectorizing them. However, fugacity coefficients are discontinuous, and therefore we propose a clustered regression network (CRNet) to address this problem. We also integrate NNEoS into PTFlash. Our experiments show that PTFlash using NNEoS enables fast and accurate flash calculations in most cases.

- Part III - **Domain generalization**

Chapter 6: We draw inspiration from CRNet presented in Chapter 5 and propose HMOE: Hypernetwork-based Mixture of Experts (MoE) to solve the domain generalization (DG) problem. Both hypernetworks [70] and MoE [227] are well-established methods in machine learning and deep learning, and we innovatively combine them. Compared to other DG methods, our proposed HMOE does not require domain labels and can divide the data consisting of mixed unknown domains into separate clusters, which are surprisingly more consistent with human intuition than original domain labels.

Finally, we summarize this thesis and suggest potential avenues for future work in Chapter 7.

Part I

PRELIMINARIES

FUNDAMENTALS OF APPLIED THERMODYNAMICS AND PHASE EQUILIBRIUM CALCULATIONS

This thesis aims to use machine learning and deep learning to accelerate phase equilibrium calculations (flash calculation). In this chapter, we will review the basic concepts of applied thermodynamics and introduce the thermodynamic notation used in the following chapters. Applied thermodynamics is a wide-ranging subject, and we will focus only on the concepts that are essential for flash calculation, such as equilibrium conditions, equations of state (EoS), and the calculation of fugacity coefficients using EoS. After introducing these basic concepts, we will detail how to perform isothermal two-phase flash calculation. This chapter is mainly based on [35, 98, 136].

2.1 BASIC CONCEPTS

2.1.1 System, state and phase

Applied thermodynamics is a branch of physics that studies the relationships between heat, work, and energy in systems that undergo physical and chemical changes. A thermodynamic system is defined as a collection of substances or entities that we aim to investigate, while anything outside of the system is referred to as the surroundings, separated by a boundary. The state of the system, which typically refers to the thermodynamic state at equilibrium rather than at a specific time, is specified by a set of state variables that are independent of both the time and path to reach them. A phase within the system is a homogeneous subset in which all intensive properties are constant, and it can be in the form of a solid, liquid, or gas. At equilibrium, the system may consist of a single phase or multiple phases.

In this thesis, our focus is on a system without homogeneous (i.e., intra-phase) chemical reactions, whereby the quantities of substances remain constant. However, each individual phase within the system is treated as an open subsystem that can exchange energy, work, and matter with other phases.

2.1.2 Thermodynamic properties

A thermodynamic property of a system refers to any measurable property that helps describe the state of the system. Thermodynamic properties can be divided into intensive and extensive properties. The intensive properties are zero-order homogeneous functions that do not depend on the system size (i.e., the quantity of substances of the system). In contrast, extensive properties are first-order homogeneous functions that are proportional to the system size. Next, we present in more detail two sets of thermodynamic properties: state variables and energy functions.

State variables are defined as properties that can be used to derive other thermodynamic properties and fully characterize the state of the system, such as entropy S , volume V , temperature T , pressure P , chemical potentials $\boldsymbol{\mu} = (\mu_1, \dots, \mu_{N_c})$, and mole numbers $\boldsymbol{n} = (n_1, \dots, n_{N_c})$, where μ_i and n_i are the chemical potential and mole number of component i , respectively, and N_c is the number of components. Among these state variables, S ,

V and \mathbf{n} are extensive properties, and T , P and $\boldsymbol{\mu}$ are their conjugated intensive properties, respectively.

Energy functions include the internal energy U , Gibbs energy G , Helmholtz free energy A , and enthalpy H , which are single-valued and differentiable thermodynamic state functions of state variables and are defined as:

$$U(S, V, \mathbf{n}) = TS - PV + \sum_{i=1}^{N_c} \mu_i n_i \quad (2.1a)$$

$$H(S, P, \mathbf{n}) = U + PV = TS + \sum_{i=1}^{N_c} \mu_i n_i \quad (2.1b)$$

$$A(T, V, \mathbf{n}) = U - TS = -PV + \sum_{i=1}^{N_c} \mu_i n_i \quad (2.1c)$$

$$G(T, P, \mathbf{n}) = U + PV - TS = \sum_{i=1}^{N_c} \mu_i n_i \quad (2.1d)$$

Except for \mathbf{n} , the state variables in parentheses are defined as natural variables that govern the corresponding energy function.

The differentials of these energy functions are calculated as follows:

$$dU = TdS - PdV + \sum_{i=1}^{N_c} \mu_i dn_i \quad (2.2a)$$

$$dH = TdS + VdP + \sum_{i=1}^{N_c} \mu_i dn_i \quad (2.2b)$$

$$dA = -SdT - PdV + \sum_{i=1}^{N_c} \mu_i dn_i \quad (2.2c)$$

$$dG = -SdT + VdP + \sum_{i=1}^{N_c} \mu_i dn_i \quad (2.2d)$$

Based on the above differentials, we can express state variables as the partial derivatives of energy functions with respect to the corresponding natural variables, as shown in Tab. 2.1. Therefore, each energy function is able to determine the state of the system by itself and its derivatives.

2.1.3 Residual properties

In reality, we can experimentally measure changes in extensive properties. However, to determine their absolute values, we need to specify a reference state, which can be set arbitrarily but is typically chosen as the ideal gas state, which is a hypothetical state governed by the ideal gas law:

$$PV = nRT \quad (2.3)$$

where R is the ideal gas constant and $n = \sum n_i$ is the total number of moles. Subsequently, we can quantify an extensive property by measuring the deviation of its value in a real state

	P	V	T	S	μ_i
$U(S, V, \mathbf{n})$	$-\left(\frac{\partial U}{\partial V}\right)_{S, n_i}$		$\left(\frac{\partial U}{\partial S}\right)_{V, n_i}$		$\left(\frac{\partial U}{\partial n_i}\right)_{S, V, n_{j \neq i}}$
$H(S, P, \mathbf{n})$		$\left(\frac{\partial H}{\partial P}\right)_{S, n_i}$	$\left(\frac{\partial H}{\partial S}\right)_{P, n_i}$		$\left(\frac{\partial H}{\partial n_i}\right)_{S, P, n_{j \neq i}}$
$A(T, V, \mathbf{n})$	$-\left(\frac{\partial A}{\partial V}\right)_{T, n_i}$			$-\left(\frac{\partial A}{\partial T}\right)_{V, n_i}$	$\left(\frac{\partial A}{\partial n_i}\right)_{T, V, n_{j \neq i}}$
$G(T, P, \mathbf{n})$		$\left(\frac{\partial G}{\partial P}\right)_{T, n_i}$		$-\left(\frac{\partial G}{\partial T}\right)_{P, n_i}$	$\left(\frac{\partial G}{\partial n_i}\right)_{T, P, n_{j \neq i}}$

Table 2.1: State variables expressed as the partial derivatives of energy functions

$(T, P$ or $V, \mathbf{n})$ from that in the ideal gas state of the same state variables $(T, P$ or $V, \mathbf{n})$. This deviation is known as residual properties:

$$M^r(T, P, \mathbf{n}) = M(T, P, \mathbf{n}) - M^*(T, P, \mathbf{n}) \quad (2.4a)$$

$$M^r(T, V, \mathbf{n}) = M(T, V, \mathbf{n}) - M^*(T, V, \mathbf{n}) \quad (2.4b)$$

where M denotes an extensive property, the superscript $*$ stands for the ideal gas state, and the superscript r represents a residual property. Because the identity $M(T, V, \mathbf{n}) = M(T, P, \mathbf{n})$ always holds, the difference between two sets of residual properties is:

$$M^r(T, P, \mathbf{n}) - M^r(T, V, \mathbf{n}) = M^*(T, V, \mathbf{n}) - M^*(T, P, \mathbf{n}) \quad (2.5)$$

The relationships between different residual properties are shown in Tab. 2.2.

$A^r(T, V, \mathbf{n}) = -\int_{\infty}^V \left(P - \frac{nRT}{V}\right) dV$	$Z = PV/(nRT)$ $A^r(T, P, \mathbf{n}) = A^r(T, V, \mathbf{n}) - nRT \ln Z$
$S^r(T, V, \mathbf{n}) = -\left(\frac{\partial A^r}{\partial T}\right)_{V, \mathbf{n}}$	$S^r(T, P, \mathbf{n}) = S^r(T, V, \mathbf{n}) + nR \ln Z$
$U^r(T, V, \mathbf{n}) = A^r(T, V, \mathbf{n}) + T S^r(T, V, \mathbf{n})$	$U^r(T, P, \mathbf{n}) = U^r(T, V, \mathbf{n})$
$H^r(T, V, \mathbf{n}) = U^r(T, V, \mathbf{n}) + PV - nRT$	$H^r(T, P, \mathbf{n}) = H^r(T, V, \mathbf{n})$
$G^r(T, V, \mathbf{n}) = A^r(T, V, \mathbf{n}) + PV - nRT$	$G^r(T, P, \mathbf{n}) = G^r(T, V, \mathbf{n}) - nRT \ln Z$ $= RT \sum n_i \ln \varphi_i$

Table 2.2: Relationships between different residual properties

2.2 PHASE EQUILIBRIUM

According to the second law of thermodynamics, an isolated multiphase system is at equilibrium if and only if it reaches a stationary point of maximum entropy. Through some mathematical derivations, the equilibrium condition of maximum entropy can be transformed into a more practical form:

$$T^\alpha = T^\beta \quad \text{and} \quad P^\alpha = P^\beta \quad \text{and} \quad \mu_i^\alpha = \mu_i^\beta \quad (2.6)$$

where α and β represent any two different phases. Therefore, intensive properties are uniform throughout the system at equilibrium. However, Eq. (2.6) involves the chemical potential μ_i , which may be challenging to use in practice because it tends to negative infinity at infinite dilution and is not intuitive to understand [35]. In this case, we introduce fugacity f_i as an alternative, which measures the tendency of a substance to escape and is defined as:

$$RT \ln \frac{f_i}{P_0} = \mu_i(T, P, \mathbf{n}) - \mu_i^*(T, P_0) \quad (2.7)$$

where $\mu_i^*(T, P_0)$ is the chemical potential of component i in the ideal gas state at T and a reference pressure P_0 . If we apply Eq. (2.7) to any two different phases α and β and then subtract one of the yielding equations from the other, we get:

$$RT \ln \frac{f_i^\alpha}{f_i^\beta} = \mu_i^\alpha - \mu_i^\beta \quad (2.8)$$

In accordance with Eq. (2.6), we get another equilibrium condition, i.e., the fugacity of each component must be the same in all phases at equilibrium:

$$f_i^\alpha = f_i^\beta \quad (2.9)$$

We further define the fugacity coefficient φ_i as:

$$\varphi_i = \frac{f_i}{c_i P} \quad (2.10)$$

where $c_i = n_i/n$ is the mole fraction of component i in the mixture. We substitute Eq. (2.10) into Eq. (2.9) and get another equilibrium condition in terms of φ_i :

$$\varphi_i^\alpha c_i^\alpha P = \varphi_i^\beta c_i^\beta P \quad \rightarrow \quad \frac{\varphi_i^\alpha}{\varphi_i^\beta} = \frac{c_i^\beta}{c_i^\alpha} \quad (2.11)$$

The above condition is frequently used in phase equilibrium calculations. To calculate φ_i , we use the residual approach as follows:

$$RT \ln \varphi_i(T, P, \mathbf{n}) = \left(\frac{\partial G^r(T, P, \mathbf{n})}{\partial n_i} \right)_{T,P} \quad (2.12)$$

$$= \left(\frac{\partial A^r(T, V, \mathbf{n})}{\partial n_i} \right)_{T,V} - RT \ln Z \quad (2.13)$$

where G^r denotes the residual Gibbs energy, A^r represents the residual Helmholtz energy, and $Z = PV/(nRT)$ refers to the compressibility factor. As we can see, the calculation of $\varphi_i(T, P, \mathbf{n})$ lies in computing the partial derivative of A^r with respect to n_i and obtaining Z , which requires an equation of state that we will introduce next.

2.3 EQUATIONS OF STATE

An equation of state (EoS) is a thermodynamic equation that describes the relationship between pressure P , volume V , temperature T , and mole numbers \mathbf{n} . The conventional form of an EoS expresses P as a function of T , V and \mathbf{n} , that is, $P(T, V, \mathbf{n})$. However, as recommended by Michelsen and Mollerup in [136], it is more practical to describe an EoS

in terms of the residual Helmholtz energy $A^r(T, V, \mathbf{n})$ in order to facilitate the calculation of fugacity coefficients using Eq. (2.13). The conversion between these two forms of EoS can be carried out as follows:

$$P(T, V, \mathbf{n}) = - \left(\frac{\partial A^r(T, V, \mathbf{n})}{\partial V} \right)_{T, \mathbf{n}} + \frac{nRT}{V} \quad (2.14a)$$

$$A^r(T, V, \mathbf{n}) = - \int_{\infty}^V \left(P(T, V, \mathbf{n}) - \frac{nRT}{V} \right) dV \quad (2.14b)$$

An EoS should satisfy the following thermodynamic consistency involving the partial derivatives of $\ln \varphi_i$ with respect to P , T and \mathbf{n} :

- Pressure P

$$\sum_i n_i \left(\frac{\partial \ln \varphi_i}{\partial P} \right)_{T, \mathbf{n}} = \frac{(Z - 1)n}{P} \quad (2.15)$$

- Temperature T

$$\sum_i n_i \left(\frac{\partial \ln \varphi_i}{\partial T} \right)_{P, \mathbf{n}} = - \frac{H^r(T, P, \mathbf{n})}{RT^2} \quad (2.16)$$

where H^r is the residual enthalpy.

- Mole numbers \mathbf{n}

$$\left(\frac{\partial \ln \varphi_i}{\partial n_j} \right)_{T, P} = \left(\frac{\partial \ln \varphi_j}{\partial n_i} \right)_{T, P} \quad (2.17a)$$

$$\sum_i n_i \left(\frac{\partial \ln \varphi_i}{\partial n_j} \right)_{T, P} = 0 \quad (2.17b)$$

Next, we introduce some examples of EoS.

2.3.1 Cubic equations of state

Cubic EoS are widely used in the petroleum and chemical industries and are particularly useful for calculating vapor-liquid equilibrium of mixtures containing hydrocarbons (e.g., CH_4 , C_2H_6) and non-polar gases (e.g., N_2 , CO_2). The commonly used cubic EoS include the Soave-Redlich-Kwong (SRK) EoS [187] and the Peng-Robinson (PR) EoS [153], which can be unified in the following forms:

$$P_{cubic}(T, V, \mathbf{n}) = \frac{nRT}{V - B} - \frac{D(T)}{(V + \delta_1 B)(V + \delta_2 B)} \quad (2.18a)$$

$$\frac{A^r_{cubic}(T, V, \mathbf{n})}{RT} = -n \ln \left(1 - \frac{B}{V} \right) - \frac{D(T)}{RTB(\delta_1 - \delta_2)} \ln \left(\frac{V + \delta_1 B}{V + \delta_2 B} \right) \quad (2.18b)$$

where $D(T)$ and B represent the temperature-dependent energy parameter and the co-volume parameter of the mixture, respectively, and δ_1 and δ_2 are two EoS-dependent parameters. For the SRK EoS, $\delta_1 = 1$ and $\delta_2 = 0$. For the PR EoS, $\delta_1 = 1 + \sqrt{2}$ and

$\delta_2 = 1 - \sqrt{2}$. The estimation of D and B is based on the van der Waals one-fluid (vdW1f) mixing rules and the classical combining rules:

$$D = n^2 a = \sum_{i=1}^{N_c} \sum_{j=1}^{N_c} n_i n_j a_{ij} \quad (2.19a)$$

$$a_{ij} = (1 - k_{ij}) \sqrt{a_i a_j} \quad (2.19b)$$

$$B = nb = \sum_{i=1}^{N_c} n_i b_i \quad (2.19c)$$

where k_{ij} is the binary interaction parameter and a_i and b_i are the energy and co-volume parameters of component i , respectively, and are calculated based on the critical pressure P_c , the critical temperature T_c and the acentric factor ω , as shown in Tab. 2.3.

SRK EoS	PR EoS
$a_c = 0.42748R^2 T_c^2/P_c$	$a_c = 0.45724R^2 T_c^2/P_c$
$m = 0.48 + 1.574\omega - 0.176\omega^2$	$m = 0.37464 + 1.54226\omega - 0.26992\omega^2$
$a(T) = a_c \left[1 + m \left(1 - \sqrt{T/T_c}\right)\right]^2$	$a(T) = a_c \left[1 + m \left(1 - \sqrt{T/T_c}\right)\right]^2$
$b = 0.08664R T_c/P_c$	$b = 0.0778R T_c/P_c$

Table 2.3: Estimation of the energy and co-volume parameters of pure compounds for cubic EoS

We rewrite Eq. (2.18b) as:

$$\frac{A_{cubic}^r(T, V, \mathbf{n})}{RT} = F = -nh(V, B) - \frac{D(T)}{T} f(V, B) \quad (2.20)$$

$$h(V, B) = \ln(1 - B/V)$$

$$f(V, B) = \frac{1}{RB(\delta_1 - \delta_2)} \ln\left(\frac{V + \delta_1 B}{V + \delta_2 B}\right)$$

And then we have:

$$\frac{\partial F}{\partial n_i} = \frac{\partial F}{\partial n} \frac{\partial n}{\partial n_i} + \frac{\partial F}{\partial B} \frac{\partial B}{\partial n_i} + \frac{\partial F}{\partial D} \frac{\partial D}{\partial n_i}$$

$$= F_n + F_B B_i + F_D D_i \quad (2.21)$$

where $\partial n / \partial n_i = 1$ and other partial derivatives of F , B and D are:

$$F_n = -h = \ln(1 - B/V) \quad (2.22a)$$

$$F_B = -ng_B - Df_B/T \quad (2.22b)$$

$$h_B = -1/(V - B) \quad (2.22c)$$

$$f_B = -(f + Vf_V)/B \quad (2.22d)$$

$$f_V = -\frac{1}{R(V + \delta_1 B)(V + \delta_2 B)} \quad (2.22e)$$

$$F_D = -f/T \quad (2.22f)$$

$$B_i = b_i \quad (2.22g)$$

$$D_i = 2 \sum_j n_j a_{ij} \quad (2.22h)$$

After integrating the above derivatives into Eq. (2.21) and reorganizing Eq. (2.13), we can get the final equation to calculate $\ln \varphi_i$ as follows:

$$\begin{aligned} \ln \varphi_i(P, T, \mathbf{c}) = & \ln \frac{RT}{P(v-b)} + \frac{b_i}{b}(Z-1) \\ & + \frac{a}{(\delta_1 - \delta_2)bRT} \left(\frac{2 \sum_{j=1}^{N_c} a_{ij}c_j}{a} - \frac{b_i}{b} \right) \ln \frac{v + \delta_2 b}{v + \delta_1 b} \end{aligned} \quad (2.23)$$

where \mathbf{c} is the mixture composition and $v = V/n$ is the molar volume. In addition, the partial derivatives of $\ln \varphi_i$ with respect to n_i are needed in phase equilibrium calculations and can be calculated analytically (see [136, Section 3.4] for more details).

To obtain the system volume V , we reformulate $P - P_{cubic}(T, V, \mathbf{n}) = 0$ as a cubic equation of the compressibility factor Z as follows:

$$Z^3 + \rho_2 Z^2 + \rho_1 Z + \rho_0 = 0 \quad (2.24)$$

For the SRK EoS, $\rho_2 = -1$, $\rho_1 = s - t(1+t)$ and $\rho_0 = -st$. For the PR EoS, $\rho_2 = t - 1$, $\rho_1 = s - 2t - 3t^2$ and $\rho_0 = t^3 + t^2 - st$, where $s = aP/(R^2T^2)$ and $t = bP/(RT)$. One simple way to find the roots of Eq. (2.24) is to use the analytical solution of the cubic equation, e.g., the Cardano's formula, which, however, may be prone to numerical errors in certain edge cases [234]. Instead, we solve Eq. (2.24) using Halley's method [37], which is a modification of Newton's method and takes into account the second derivative of the function to improve the convergence. Halley's method starts with a liquid-like guess and converges to a real root Z_0 , and then we deflate Eq. (2.24) as follows:

$$(Z - Z_0)(Z^2 + pZ + q) = 0 \quad (2.25)$$

where $p = Z_0 - 1$ and $q = pZ_0 + \rho_1$. If $p^2 < 4q$, only one real root Z_0 exists. Otherwise, there are three real roots and the other two are $-p/2 \pm \sqrt{p^2 - 4q}/2$. In the latter case, we assign the smallest root to the liquid phase and the largest root to the vapor phase. Subsequently, we choose the root associated with the lowest Gibbs energy, which corresponds to the most stable system according to the principle of minimum energy.

2.3.2 Statistical Associating Fluid Theory

The Statistical Associating Fluid Theory (SAFT) EoS is a theoretically derived model based on the Wertheim perturbation theory [219] [222] [220] [221] [24] [83]. The SAFT EoS is particularly suited for hydrogen-bonding systems, in which hydrogen bonds can occur through the self-association between molecules of the same type, such as water, alcohols, and amines, or through the cross-association between molecules of different types, e.g., between water and methanol.

Fig. 2.1 illustrates the principle of the SAFT EoS by showing how to form a molecule. First, the fluid is assumed to consist of hard spheres of equal size to consider the repulsion effect. Next, a dispersion potential (e.g., the square-well or Lennard-Jones potential) is added to account for attractive interactions between the spheres. Then, chains are formed between the spheres. Finally, association sites are introduced, which allow the chains to be associated with each other through hydrogen bonds.

Considering the effects of repulsion, dispersion, chain formation, and association, the SAFT EoS is expressed in the following form:

$$A_{SAFT}^r = A_{hs} + A_{disp} + A_{chain} + A_{assoc} \quad (2.26)$$

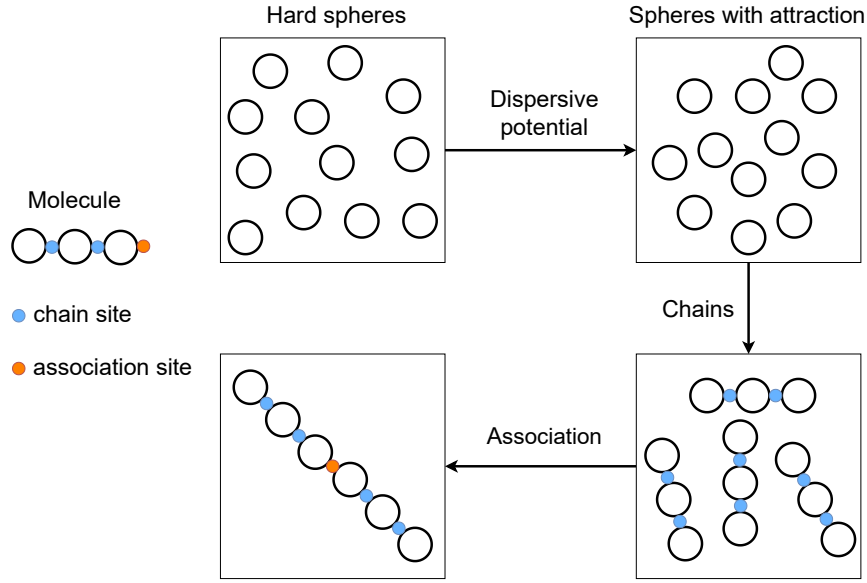


Figure 2.1: Formation of a molecule in the original SAFT EoS (Adapted from [51])

where A_{hs} describes the repulsive interactions between hard spheres, A_{disp} is the dispersion contribution, A_{chain} is the contribution from chain formation, and A^{assoc} refers to the contribution from association between molecules. A range of variants of the original SAFT EoS are proposed, such as the simplified SAFT EoS [51], the soft-SAFT EoS [17], the SAFT-VR EoS [62], and the PC-SAFT [67, 242]. These variants generally have small differences in the chain and association terms but differ largely in the dispersion and repulsion terms. For more details about the SAFT EoS, refer to [98, Chapter 8].

2.3.3 Cubic Plus Association equation of state

The Cubic Plus Association (CPA) EoS [99] [100] combines the SRK EoS and the SAFT EoS by leveraging the SRK EoS to describe the dispersion and repulsion terms and borrowing the association term from the SAFT EoS to take into account the effect of hydrogen bonding, as follows:

$$A_{CPA}^r = A_{SRK}^r + A_{assoc}^r \quad (2.27)$$

where A_{SRK}^r is given by Eq. (2.18b) with $\delta_1 = 1$ and $\delta_2 = 0$:

$$\frac{A_{SRK}^r(T, V, \mathbf{n})}{RT} = -n \ln \left(1 - \frac{B}{V} \right) - \frac{D(T)}{RTB} \ln \left(1 + \frac{B}{V} \right) \quad (2.28)$$

And A_{assoc}^r denotes the association contribution and is defined as:

$$\frac{A_{assoc}^r(T, V, \mathbf{n})}{RT} = \sum_i n_i \sum_{A_i} \left(\ln X_{A_i} - \frac{1}{2} X_{A_i} + \frac{1}{2} \right) \quad (2.29)$$

where X_{A_i} refers to the proportion of A sites on molecule i that do not form bonds with other active sites and is calculated as:

$$\begin{aligned} X_{A_i} &= \frac{V}{V + \sum_j n_j \sum_{B_j} X_{B_j} \Delta^{A_i B_j}} \\ \Delta^{A_i B_j} &= g(V, n) \left[\exp\left(\frac{\varepsilon^{A_i B_j}}{RT}\right) - 1 \right] b_{ij} \beta^{A_i B_j} \quad \text{with} \quad b_{ij} = \frac{b_i + b_j}{2} \\ g(V, n) &= \frac{1}{1 - 1.9\eta} \quad \text{with} \quad \eta = \frac{B}{4V} \end{aligned} \quad (2.30)$$

where $\Delta^{A_i B_j}$ is the association strength, $\varepsilon^{A_i B_j}$ is the association energy, $\beta^{A_i B_j}$ is the association volume, and $g(V, n)$ is the radial distribution function. By substituting A_{CPA}^r into Eq. (2.14a), we can obtain another form of the CPA EoS in terms of P as follows:

$$P_{CPA}(T, V, \mathbf{n}) = \frac{nRT}{V - B} - \frac{D(T)}{V(V + B)} + \frac{RT}{2V} \left(1 - V \frac{\partial \ln g}{\partial V} \right) \sum_i n_i \sum_{A_i} (1 - X_{A_i}) \quad (2.31)$$

The partial derivative of A_{CPA}^r with respect to n_i is calculated as:

$$\frac{\partial A_{CPA}^r}{\partial n_i} = \frac{\partial A_{SRK}^r}{\partial n_i} + \frac{\partial A_{assoc}^r}{\partial n_i} \quad (2.32)$$

The first term on the right-hand side was already discussed, and the second term is calculated as:

$$\begin{aligned} \frac{\partial A_{assoc}^r}{\partial n_i} &= \sum_{A_i} \ln X_{A_i} - \sum_i n_i \sum_{A_i} (1 - X_{A_i}) \frac{\partial \ln g}{\partial n_i} \\ \frac{\partial g}{\partial n_i} &= \frac{\partial g}{\partial B} B_i \quad \text{with} \quad \frac{\partial g}{\partial B} = \frac{0.475V}{(V - 0.457B)^2} \end{aligned} \quad (2.33)$$

Substituting Eq. (2.32) into Eq. (2.13) yields the final equation for calculating fugacity coefficients. To obtain V for the CPA EoS, we need to solve the following equation:

$$P - P_{CPA}(T, V, \mathbf{n}) = 0 \quad (2.34)$$

The difficulty in solving the above equation arises from the fact that the site fractions X_{A_i} are the implicit function of V , as defined by Eq. (2.30). One approach to solving Eq. (2.34) is to implement a nested-loop optimization process. In the outer loop, we fix X_{A_i} and adjust V to satisfy Eq. (2.34). In the inner loop, Eq. (2.30) is solved given the obtained V to update X_{A_i} . More details can be found in [134].

2.4 ISOTHERMAL TWO-PHASE FLASH CALCULATION

In this section, we introduce isothermal two-phase flash calculation. In the following, without loss of generality, we consider the equilibrium between the liquid and vapor phases.

2.4.1 Problem setting

We consider a mixture of N_c components. Given pressure P , temperature T and feed composition $\mathbf{z} = (z_1, \dots, z_{N_c})$, where z_i represents the molar fraction of component i

in the whole mixture. The objective of flash calculation is to determine the system state at equilibrium: single phase or coexistence of two phases. In the latter case, we need to additionally compute the molar fraction of the vapor phase θ_V , the composition of the liquid phase \mathbf{x} and that of the vapor phase \mathbf{y} . These properties are constrained by the following mass balance equations:

$$x_i(1 - \theta_V) + y_i\theta_V = z_i, \quad \text{for } i = 1, \dots, N_c \quad (2.35a)$$

$$\sum_{i=1}^{N_c} x_i = \sum_{i=1}^{N_c} y_i = 1 \quad (2.35b)$$

In addition, the equilibrium condition that we introduced in Sec. 2.2 should also be satisfied:

$$\frac{\varphi_i^L(P, T, \mathbf{x})}{\varphi_i^V(P, T, \mathbf{y})} = \frac{y_i}{x_i} \quad (2.36)$$

where the superscripts L and V refer to the liquid and vapor phases, respectively, and φ_i denotes the fugacity coefficient of component i and is calculated by Eq. (2.13) using a specific EoS.

Eq. (2.35) and Eq. (2.36) form a non-linear system, which is generally solved in a two-stage procedure. First, we establish the stability of a mixture via stability analysis (Sec. 2.4.2). If the mixture is stable, only one phase exists at equilibrium, and its composition is equal to the feed composition \mathbf{z} . Otherwise, two phases coexist. Second, we determine θ_V , \mathbf{x} and \mathbf{y} at equilibrium through phase split calculations (Sec. 2.4.3).

2.4.2 Stability analysis

A mixture of composition \mathbf{z} is stable at specified P and T if and only if its total Gibbs energy is at the global minimum, which can be verified through the reduced tangent plane distance [130]:

$$tpd(\mathbf{e}) = \sum_{i=1}^{N_c} e_i (\ln e_i + \ln \varphi_i(\mathbf{e}) - \ln z_i - \ln \varphi_i(\mathbf{z})) \quad (2.37)$$

where $\mathbf{e} = (e_1, \dots, e_{N_c})$ is the composition of a trial phase. If $tpd(\mathbf{e})$ is non-negative for any \mathbf{e} with $\sum e_i = 1$, the mixture is stable. This involves a constrained minimization problem, which is generally reframed as an unconstrained one:

$$tm(\mathbf{E}) = 1 + \sum_{i=1}^{N_c} E_i (\ln E_i + \ln \varphi_i(\mathbf{E}) - \ln z_i - \ln \varphi_i(\mathbf{z}) - 1) \quad (2.38)$$

where tm is the modified tangent plane distance and \mathbf{E} is the mole numbers of a trial phase. To locate the minima of tm , we first use the successive substitution method accelerated by the Dominant Eigenvalue Method (DEM) [145], which iterates:

$$\ln E_i^{(k+1)} = \ln z_i + \ln \varphi_i(\mathbf{z}) - \ln \varphi_i(\mathbf{E}^{(k)}) \quad (2.39)$$

It is customary to initiate the above iteration with two sets of estimates, namely a vapor-like estimate $E_i = K_i z_i$ and a liquid-like estimate $E_i = z_i / K_i$, where K_i is the distribution coefficients, defined as y_i / x_i and initialized via the Wilson approximation [187] as follows:

$$\ln K_i = \ln \left(\frac{P_{c,i}}{P} \right) + 5.373(1 + \omega_i) \left(1 - \frac{T_{c,i}}{T} \right) \quad (2.40)$$

where $T_{c,i}$ and $P_{c,i}$ refer to the critical temperature and pressure of component i , respectively, and ω_i is the acentric factor. Once converging to a stationary point (i.e., $\max(|\partial tm/\partial \mathbf{E}|) < 1.0\text{e-}6$) or a negative tm is found, the successive substitution stops.

If the successive substitution fails to converge quickly, particularly around critical points for which liquid and vapor phases are almost indistinguishable, we switch to a second-order optimization technique — the trust-region method with restricted steps [75] to achieve faster convergence. The trust-region method is accomplished by iterating the following equations:

$$\begin{aligned} \boldsymbol{\beta}^{(k)} &= 2\sqrt{\mathbf{E}^{(k)}} \\ (\mathbf{H}^{(k)} + \zeta^{(k)}\mathbf{I}) \cdot \Delta\boldsymbol{\beta} + \mathbf{g}^{(k)} &= \mathbf{0} \quad \text{s.t.} \quad \|\Delta\boldsymbol{\beta}\| \leq \Delta_{max}^{(k)} \\ \boldsymbol{\beta}^{(k+1)} &= \boldsymbol{\beta}^{(k)} + \Delta\boldsymbol{\beta} \\ \mathbf{E}^{(k+1)} &= \left(\frac{\boldsymbol{\beta}^{(k+1)}}{2} \right)^2 \end{aligned} \quad (2.41)$$

where \mathbf{I} is the identity matrix, \mathbf{g} and \mathbf{H} are the gradient and Hessian matrices of tm with respect to $\boldsymbol{\beta}$, respectively, and are calculated as follows:

$$g_i = \sqrt{E_i}(\ln E_i + \ln \varphi_i(\mathbf{E}) - \ln z_i - \ln \varphi_i(\mathbf{z})) \quad (2.42a)$$

$$H_{ij} = \sqrt{E_i E_j} \frac{\partial \ln \varphi_i}{\partial E_j} + \sigma_{ij} \left(1 + \frac{g_i}{\beta_i} \right) \quad \text{where } \sigma_{ij} = 1 \Leftrightarrow i = j \quad (2.42b)$$

In addition, ζ is the trust-region size used to guarantee the positive definiteness of $\mathbf{H} + \zeta\mathbf{I}$ and to tailor the step size to meet $\|\Delta\boldsymbol{\beta}\| \leq \Delta_{max}$, where Δ_{max} is adjusted during iterations depending on the match between the actual reduction $\delta_{tm} = tm^{(k+1)} - tm^{(k)}$ and the predicted reduction based on the quadratic approximation $\hat{\delta}_{tm} = \Delta\boldsymbol{\beta}^T \mathbf{g} + \frac{1}{2} \Delta\boldsymbol{\beta}^T \mathbf{H} \Delta\boldsymbol{\beta}$, using the following heuristic rules:

$$\Delta_{max}^{(k+1)} = \begin{cases} \frac{\Delta_{max}^{(k)}}{2}, & \text{if } \left| \delta_{tm}/\hat{\delta}_{tm} \right| \leq 0.25 \\ 2\Delta_{max}^{(k)}, & \text{if } \left| \delta_{tm}/\hat{\delta}_{tm} \right| \geq 0.75 \\ \Delta_{max}^{(k)}, & \text{otherwise} \end{cases} \quad (2.43)$$

The convergence criterion of Eq. (2.41) is $\max(|\mathbf{g}|) < 1.0\text{e-}6$.

Furthermore, based on the results of stability analysis, we can re-estimate K_i more accurately as z_i/E_i^L if $tm^L < tm^V$ or E_i^V/z_i otherwise, where the superscripts V and L denote the results obtained using the vapor-like and liquid-like estimates, respectively.

2.4.3 Phase split calculations

Substituting $K_i = y_i/x_i$ into Eq. (2.35) yields the following well-known Rachford-Rice equation [160]:

$$f_{RR}(\theta_V, \mathbf{K}) = \sum_{i=1}^{N_c} \frac{(K_i - 1)z_i}{1 + (K_i - 1)\theta_V} = 0 \quad (2.44)$$

Given \mathbf{K} , the solution of the above equation amounts to finding an appropriate zero yielding all non-negative phase compositions. Concretely, we adopt the method proposed

by [106], which involves transforming f_{RR} into a helper function h_{RR} that is more linear in the vicinity of the zero:

$$h_{RR}(\theta_V, \mathbf{K}) = (\theta_V - \alpha_l) \cdot (\alpha_r - \theta_V) \cdot f_{RR}(\theta_V) = 0 \quad (2.45)$$

where $\alpha_l = 1/(1 - \max(K_i))$ and $\alpha_r = 1/(1 - \min(K_i))$. The above equation is solved by alternating between the Newton method and the bisection method, of which the latter is used when the Newton step causes θ_V to fall outside the bounds containing the zero and becoming narrower during iterations. When the Newton step size is smaller than $1.0\text{e-}8$, the iteration stops.

To obtain θ_V , \mathbf{x} and \mathbf{y} at equilibrium, phase split calculations start with the accelerated successive substitution method, as illustrated in Fig. 2.2, and the corresponding convergence criterion is $\max(|K_i^{(k+1)}/K_i^{(k)} - 1|) < 1.0\text{e-}8$.

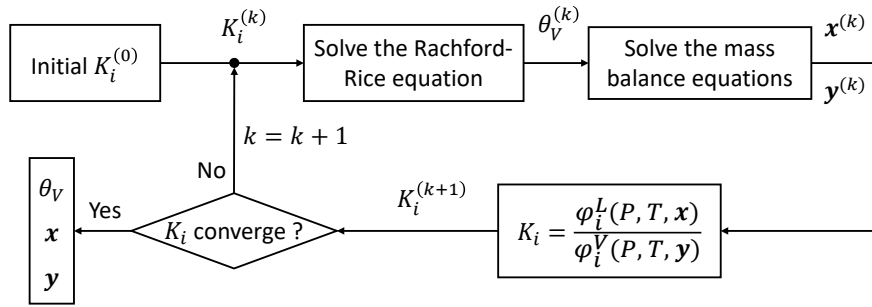


Figure 2.2: Successive substitution of phase split calculations

If the successive substitution fails to converge after a few iterations (9 in our work), we use the trust-region method to minimize the following reduced Gibbs energy:

$$G^{re} = \sum_{i=1}^{N_c} n_i^L (\ln x_i + \ln \varphi_i^L) + \sum_{i=1}^{N_c} n_i^V (\ln y_i + \ln \varphi_i^V) \quad (2.46)$$

where $n_i^L = x_i(1 - \theta_V)$ and $n_i^V = y_i\theta_V$ are the mole numbers of liquid and vapor phases, respectively. We choose n_i^V as the independent variable and perform the following iteration:

$$\left(\tilde{\mathbf{H}}^{(k)} + \tilde{\zeta}^{(k)} \cdot \text{diag} \left(\frac{\mathbf{z}}{\mathbf{xy}} \right) \right) \cdot \Delta \mathbf{n}^V + \tilde{\mathbf{g}}^{(k)} = \mathbf{0} \quad s.t. \quad \|\Delta \mathbf{n}^V\| \leq \Delta_{max}^{(k)} \quad (2.47)$$

$$\mathbf{n}^{V,k+1} = \mathbf{n}^{V,k} + \Delta \mathbf{n}^V$$

where $\text{diag}(\cdot)$ is a diagonal matrix with diagonal entries in parentheses, and $\tilde{\mathbf{g}}^{(k)}$ and $\tilde{\mathbf{H}}^{(k)}$ are the gradient and hessian matrices of G^{re} with respect to n_i^V , respectively, and are calculated as follows:

$$\tilde{g}_i = \ln y_i + \ln \varphi_i^V - \ln x_i - \ln \varphi_i^L \quad (2.48a)$$

$$\tilde{H}_{ij} = \frac{1}{\theta_V(1 - \theta_V)} \left(\frac{z_i}{x_i y_i} \sigma_{ij} - 1 + \theta_V \frac{\partial \ln \varphi_i^L}{\partial n_j^L} + (1 - \theta_V) \frac{\partial \ln \varphi_i^V}{\partial n_j^V} \right) \quad (2.48b)$$

Here, the trust-region method for phase split calculations is implemented in the same way as in stability analysis, and Eq. (2.47) stops if $\max(|\tilde{\mathbf{g}}|) < 1.0\text{e-}8$.

2.4.4 Strategy for the isothermal two-phase flash calculation

We basically adopt the rules of thumb proposed by Michelsen and Mollerup in [136] to implement the isothermal two-phase flash calculation, as shown in Fig. 2.3. In the flowchart, we first initialize the distribution coefficients K_i using the Wilson approximation. Subsequently, in order to avoid computationally expensive stability analysis, we carry out the successive substitution of phase split calculations 3 times, which will end up with 3 possible cases: (1) θ_V is out of bounds (0, 1) during iterations. (2) None of ΔG , $tpd(\mathbf{x})$ and $tpd(\mathbf{y})$ are negative, where $tpd(\mathbf{x})$ and $tpd(\mathbf{y})$ are reduced tangent plane distances using current vapor and liquid phases as trial phases, and $\Delta G = \theta_V \times tpd(\mathbf{x}) + (1 - \theta_V) \times tpd(\mathbf{y})$. (3) Any of ΔG , $tpd(\mathbf{x})$ and $tpd(\mathbf{y})$ is negative.

For the first two cases, we cannot be sure of the stability of the given mixture, thus continuing with stability analysis. For the third case, we can conclude that the given mixture is unstable, thereby sidestepping stability analysis. Finally, if two phases coexist, we perform phase split calculations to get θ_V , \mathbf{x} and \mathbf{y} at equilibrium.

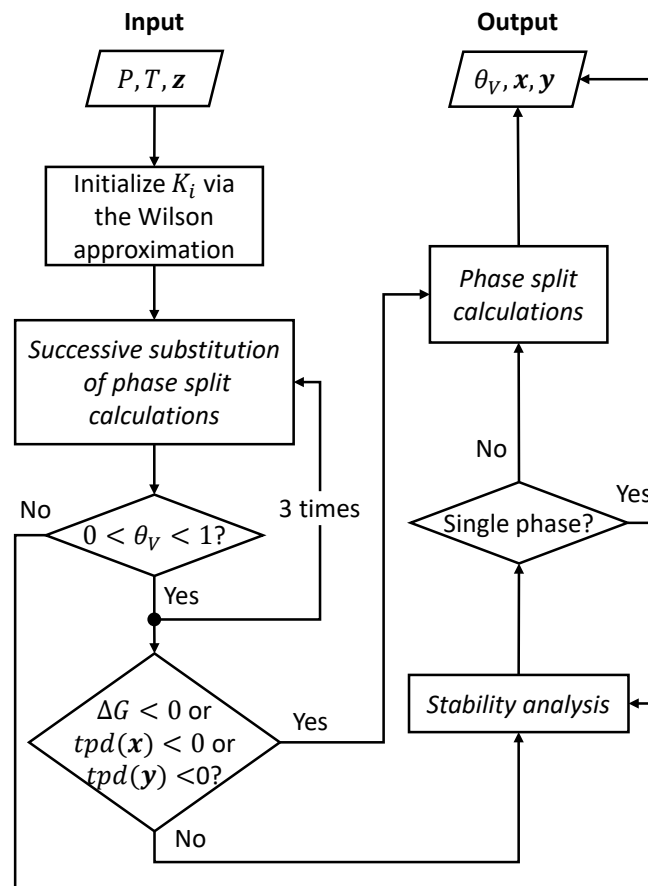


Figure 2.3: Flowchart of the isothermal two-phase flash calculation

In this chapter, we present the basic concepts of neural networks and deep learning to pave the way for the application of neural networks in later chapters. Artificial neural networks (ANN) refer to a series of mathematical models inspired by the structure and function of the brain's nervous system. We have learned that the brain's nervous system consists of a large number of interconnected neurons, with one neuron transmitting the signals it receives from its dendrites to its axon terminals and finally to the next neuron. While our understanding of brain mechanisms is still limited, ANN draw on the idea of neuronal connectionism. Over the past decades, a variety of neural networks have been developed depending on the type of data to be processed [173], including feedforward neural networks (FNN) for structured data, convolutional neural networks (CNN) for image processing [105], recurrent neural networks (RNN) for sequences [168], and graph neural networks (GNN) for unstructured graph data [172]. ANN have been ubiquitous and gained great success in a wide range of fields, such as speech recognition, image recognition and generation, machine translation, and natural language processing.

One term tightly related to ANN is deep learning, which is a methodology for effective representation learning from data by leveraging the hierarchy of concepts. These concepts are built on top of each other and form a deep cascade that transforms raw data into higher-level and more abstract representations, which largely evades laborious hand-designed feature engineering. From this point of view, ANN are quintessential deep learning models.

In the following, we present feedforward neural networks from the perspective of supervised learning, in which both input and output data are available.

3.1 FEEDFORWARD NEURAL NETWORKS

The basic unit of feedforward neural networks is an artificial neuron, as depicted in Fig. 3.1. A neuron processes an input vector $\mathbf{x} = (x_1, \dots, x_n)$ in the following way:

$$a = f_a \left(\sum_{i=1}^n (w_i \cdot x_i) + b \right) = f_a(\mathbf{w}^T \mathbf{x} + b) \quad (3.1)$$

where $\mathbf{w} = (w_1, \dots, w_n)$ and b refer to the weights and bias of the neuron, respectively, and f_a denotes an activation function. Fig. 3.2 shows some commonly used activation functions, including the logistic function (sigmoid), the hyperbolic tangent function (tanh), the rectified linear unit (ReLU) [64, 142], and the sigmoid linear unit (SiLU or Swish) [45, 163, 164].

The expressiveness of a single neuron is far from satisfactory, and feedforward neural networks combine many neurons to realize and perform complex functions, as shown in Fig. 3.3. Neurons are divided into different layers based on the order in which they receive information. The first layer is the input layer, the last layer is the output layer, and the other intermediate layers are called hidden layers.

The neurons of the l th layer take as input the output of the previous layer \mathbf{a}^{l-1} and process it as follows:

$$\mathbf{a}^l = f_a(\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l) \quad (3.2)$$

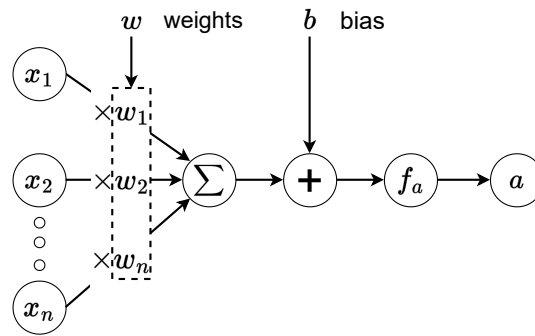
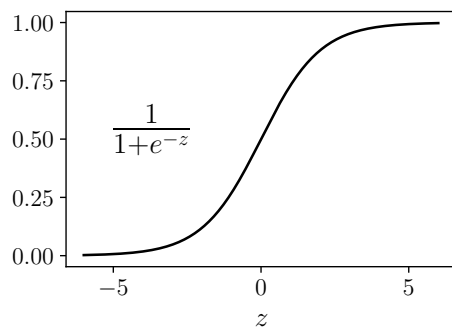
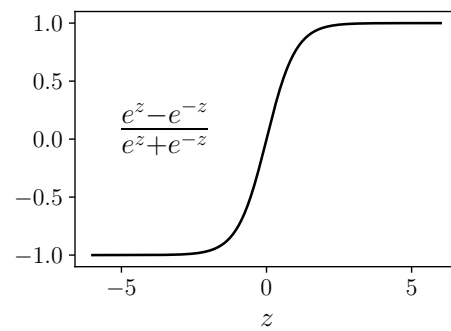


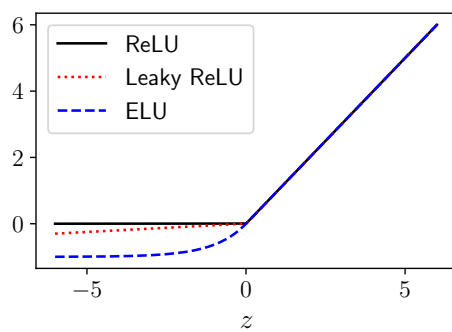
Figure 3.1: Representation of an artificial neuron



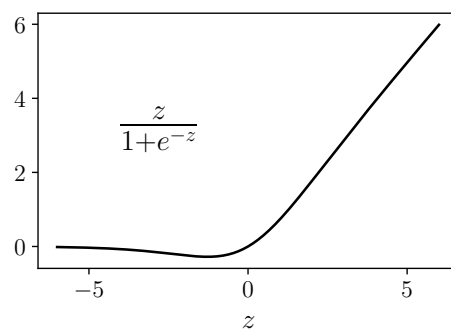
(a) sigmoid



(b) tanh



(c) ReLU and its variants



(d) SiLU (swish)

Figure 3.2: Commonly used activation functions

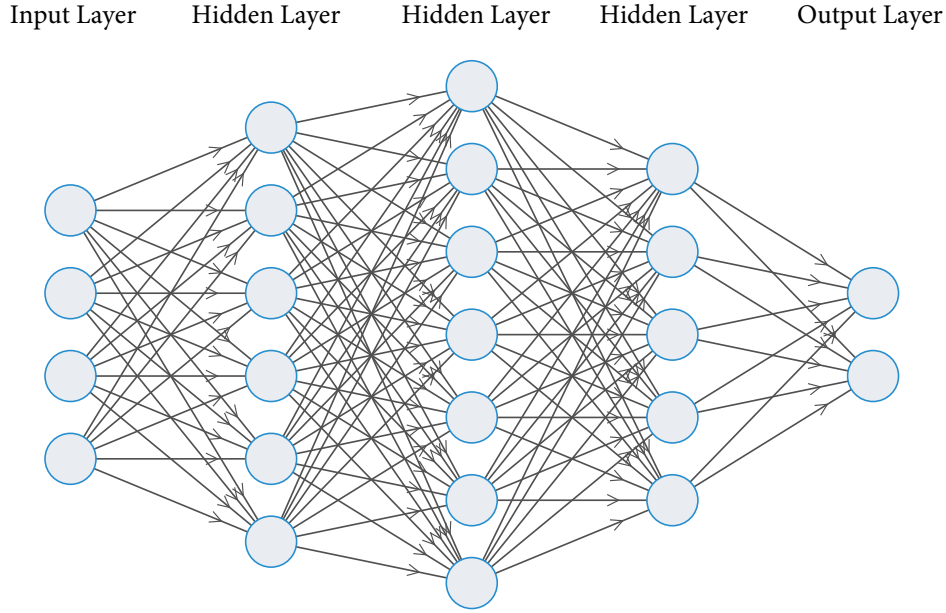


Figure 3.3: Architecture of feedforward neural networks

where \mathbf{W}^l is the weights of all neurons of the l th layer and is a matrix $\in \mathbb{R}^{N_l \times N_{l-1}}$ (N_l is the number of neurons), \mathbf{b}_l is the bias vector $\in \mathbb{R}^{N_l}$, and $\mathbf{a}^l \in \mathbb{R}^{N_l}$ is the output. In fact, Eq. (3.2) can be seen as a vectorized version of Eq. (3.1). Feedforward neural networks propagate information in one direction without feedback, which can be represented by a directed acyclic graph:

$$\mathbf{x} = \mathbf{a}_0 \rightarrow \mathbf{a}_1 \rightarrow \cdots \rightarrow \mathbf{a}_{L-1} \rightarrow \mathbf{a}_L = \mathbf{y} = f_{NN}(\mathbf{x}; \mathbf{W}, \mathbf{b}) \quad (3.3)$$

where \mathbf{W} and \mathbf{b} stand for the weights and biases of all layers of a neural network, L is the total number of layers, and \mathbf{y} is the final output. We refer to N_l and L as the hyperparameters of neural networks, which should be specified before the training of neural networks, and we refer to \mathbf{W} and \mathbf{b} as the learnable or trainable parameters, which are learned during the training of neural networks.

3.2 UNIVERSAL APPROXIMATION THEOREM

The theoretical foundation of feedforward neural networks is underpinned by the universal approximation theorem [31], which states that any feedforward neural network with a single hidden layer containing a sufficient number of neurons with sigmoidal activation functions (e.g., sigmoid, tanh) is able to approximate any continuous function to any desired degree of accuracy. [81] expanded the range of activation functions and proved that the universal approximation theorem holds for any continuous, bounded, and non-constant activation function. More recently, [188] further relaxed the restrictions on activation functions and demonstrated that feedforward neural networks with unbounded activation functions (e.g., relu, SiLU) are still universal approximators.

While the universal approximation theorem demonstrates the existence of a neural network to approximate any continuous function, it does not provide guidance on how to construct such a neural network, including the number of neurons and layers, weights,

and biases. [18] showed that finding a neural network that is precisely fitted to a specific set of training examples is an NP-complete problem.

3.3 TRAINING OF NEURAL NETWORKS

Although training neural networks involves minimizing a high-dimensional, non-convex cost function, in practice it is easier than anticipated, and gradient-based optimization methods such as stochastic gradient descent (SGD) can be used to find good local minima that achieve the desired accuracy in a limited time. Algorithm 1 outlines the general procedure for training neural networks, which involves a nested-loop process. In the outer loop (i.e., epoch), the entire training set is randomly divided into mini-batches. In the inner loop (i.e., iteration), neural networks are trained on each mini-batch.

Algorithm 1: General procedure for training neural networks

```

Input: Training set  $\mathcal{D}_{tr} = \{(\mathbf{x}_i; \mathbf{y}_i)\}_{i=1}^N$ 
      Neural network  $f_{NN}(\mathbf{x}; \Theta)$ 
      Cost function  $\mathcal{L}$ 
      Optimizer
      Learning rate  $\eta$ 
      Batch size  $m$ 
      Maximum number of epochs  $K$ 
      Optional validation set  $\mathcal{D}_{va}$ 
1 Parameter initialization
2 | Initialize  $\Theta$ 
3 Training
4 | for  $k = 1$  to  $K$  do
5 |   Randomly split  $\mathcal{D}_{tr}$  into  $\lfloor \frac{N}{m} \rfloor$  mini-batches of  $m$  examples
6 |   for  $i = 1$  to  $\lfloor \frac{N}{m} \rfloor$  do
7 |     Perform the forward pass for the mini-batch  $\mathcal{B}_i$  and use Eq. (3.4) to
8 |       evaluate the cost function  $\mathcal{L}^{(i)} = \mathcal{L}(\mathcal{B}_i, \Theta^{(i)})$ 
9 |     Perform the backpropagation to compute the gradient  $\mathbf{g}^{(i)} = \nabla_{\Theta^{(i)}} \mathcal{L}^{(i)}$ 
10 |    Update  $\Theta^{(i+1)} \leftarrow \text{optimizer.update}(\Theta^{(i)}, \mathbf{g}^{(i)}, \eta)$ 
11 |    /* Assuming the optimizer has an "update" method */
12 |     $i \leftarrow i + 1$ 
13 |   Evaluate and monitor the performance of the neural network on  $\mathcal{D}_{va}$ 
14 |    $k \leftarrow k + 1$ 
Output: Neural network with its parameters  $\Theta$  trained on  $\mathcal{D}_{tr}$ 

```

There are two main challenges in training neural networks. On the one hand, training neural networks involves a high-dimensional, non-convex optimization problem (Fig. 3.4), which is troubled by how to get rid of local minima and saddle points [34]. On the other hand, training neural networks may experience issues of vanishing and exploding gradients. The vanishing gradient problem occurs when the gradient is so small that training is unable to further reduce the cost function. The exploding gradient problem arises when the gradient is so large that training becomes unstable and oscillates. However, after decades of ongoing efforts by the deep learning community, these problems have been largely alleviated.

3.3.1 Cost function

Given a training set $\mathcal{D}_{tr} = \{(\mathbf{x}_i; \mathbf{y}_i)\}_{i=1}^N$, where $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$, and $(\mathbf{x}; \mathbf{y})$ is drawn from a joint distribution $p_r(\mathbf{x}, \mathbf{y})$ defined on $\mathcal{X} \times \mathcal{Y}$, a neural network is trained to minimize the following cost function:

$$\mathcal{L}(\mathcal{D}, \Theta) = \underbrace{\frac{1}{N} \sum_{i=1}^N \ell(\mathbf{y}_i, \hat{\mathbf{y}}_i)}_{\text{empirical risk}} + \Omega(\Theta) \quad (3.4)$$

where Θ denotes all learnable parameters of the neural network, $\hat{\mathbf{y}} = f_{NN}(\mathbf{x}; \Theta)$ is the prediction calculated by Eq. (3.3), and $\ell(\cdot)$ is a loss function used to measure the deviation of the prediction from the ground truth, such as mean squared error (MSE) for regression problems and cross entropy loss (CE) for classification problems. Fig. 3.4 intuitively shows how \mathcal{L} varies with Θ and reflects to some extent the difficulty of training neural networks.

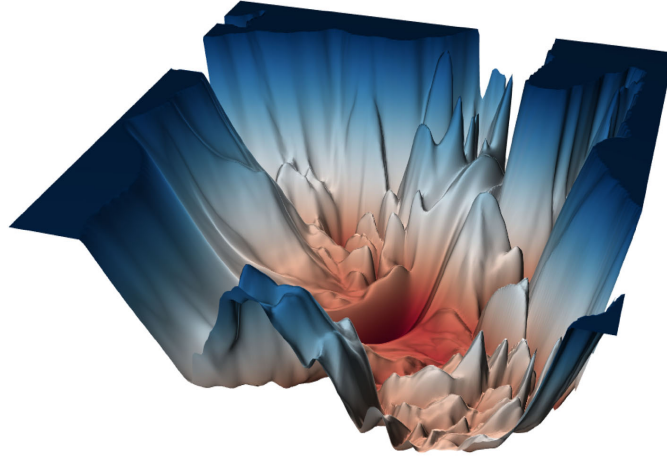


Figure 3.4: This figure shows the loss landscape of ResNet-56 [74] without shortcut connections for the CIFAR-10 dataset. The loss landscape represents how the cost function changes with the parameters of neural networks. The visualization of the loss landscape is attributed to [112]. We can observe that the loss landscape is extremely uneven, reflecting the high non-convexity and difficulty of training neural networks.

The cost function consists of two parts: the empirical risk and $\Omega(\Theta)$ which is the regularization on Θ . The empirical risk is an approximation of the expected risk defined as $\mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim p_r}[\ell(\mathbf{y}, \hat{\mathbf{y}})]$, and it converges to the expected risk when the number of training examples $|\mathcal{D}_{tr}|$ is large enough based on the law of large numbers. Minimizing the expected risk is expected to lead to low generalization errors on unseen data. However, when $|\mathcal{D}_{tr}|$ is small and neural networks are overly complex (e.g., with too many neurons and layers), generalization errors may be great. This problem is called overfitting, as shown in Fig. 3.5b. To avoid overfitting, we can limit the complexity of a neural network by imposing $\Omega(\Theta)$ on its parameters. We can define $\Omega(\Theta)$ explicitly, such as the ℓ_1 regularization $\lambda \|\Theta\|_1$ and the ℓ_2 regularization $\lambda \|\Theta\|_2$ where λ is the penalty strength, and we can also employ some regularization techniques for which $\Omega(\Theta)$ works implicitly, such as EarlyStopping [157], Dropout [190] [55] [56] and Batch Normalization [87]. However, if we excessively penalize the ability of neural networks, generalization errors are also likely to be high due to insuffi-

cient ability. This is known as underfitting, as shown in Fig. 3.5a. Therefore, the setting of $\Omega(\Theta)$ has a significant impact on the generalization performance of neural networks.

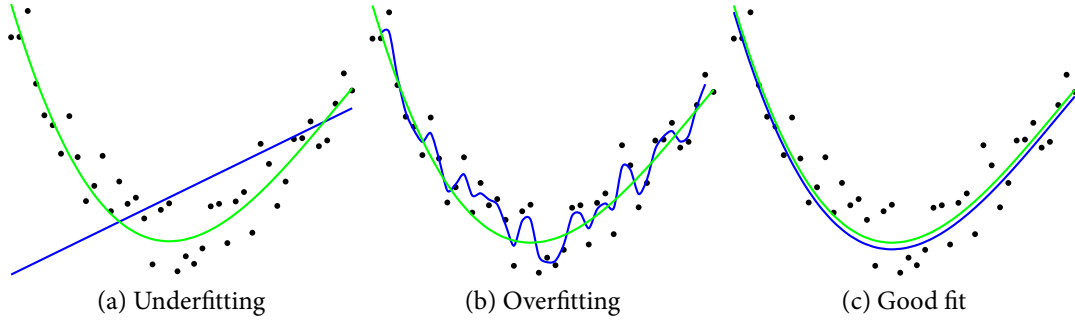


Figure 3.5: The green line is the target function and the black dots are the training set with some random noise. Figure (a): A simple linear model is not able to fit the data well. Figure (b): An overly complex model pays too much attention to the noise, resulting in overfitting. Figure (c): An appropriate model is robust to the noise and also well approximates the target function.

3.3.2 Optimization algorithms

We need an optimization algorithm (also known as an optimizer) to update the learnable parameters Θ during training, and the mainstream optimization algorithms in deep learning are based on stochastic gradient descent (SGD) as follows:

$$\Theta^{(i+1)} = \Theta^{(i)} - \eta \mathbf{g}^{(i)} \quad (3.5)$$

where the superscript i is the number of iterations, η denotes the learning rate, and $\mathbf{g}^{(i)} = \nabla_{\Theta^{(i)}} \mathcal{L}^{(i)}$ represents the gradient of the cost function \mathcal{L} with respect to Θ and is computed through the backward propagation of errors [168], also known as the backpropagation. The learning rate η is a crucial hyper-parameter, and the training of neural networks converges slowly with a small η and may diverge with a great η . Current deep learning frameworks, such as Tensorflow [1] and PyTorch [150], support Automatic Differentiation (AD) to calculate $\mathbf{g}^{(i)}$, which decomposes a composite function into basic operations ($+$, $-$, \times , \div) and elementary functions (\sin , \cos , \exp , \log) and then creates a computational graph to automatically compute the gradient of the composite function based on the chain rule.

Modern optimization algorithms primarily improve SGD in two ways: the adjustment of learning rate and the correction of gradient estimation. Here, we elaborate on Adaptive Moment Estimation Algorithm (Adam) [95], which has become the de-facto optimization algorithm and will be frequently used in later chapters. Adam updates Θ as follows:

$$\begin{aligned} v^{(i+1)} &= \beta_1 v^{(i)} + (1 - \beta_1)^{(i)} g^{(i)} \\ s^{(i+1)} &= \beta_2 s^{(i)} + (1 - \beta_2) g^{(i)} \odot g^{(i)} \\ \hat{v}^{(i+1)} &= \frac{v^{(i)}}{1 - (\beta_1)^i} \\ \hat{s}^{(i+1)} &= \frac{s^{(i)}}{1 - (\beta_2)^i} \\ \Theta^{(i+1)} &= \Theta^{(i)} - \frac{\eta}{\sqrt{\hat{s}^{(i+1)} + \epsilon}} \hat{v}^{(i+1)} \end{aligned} \quad (3.6)$$

where $v^{(i)}$ and $s^{(i)}$ denote the biased first and second moment estimates of gradients ($v^{(0)} = s^{(0)} = 0$), respectively, β_1 and β_2 refer to the decay rates for the moving averages of moment estimates and are generally set to 0.9 and 0.999, $\hat{v}^{(i)}$ and $\hat{s}^{(i)}$ are used to correct the bias of $v^{(i)}$ and $s^{(i)}$, and ϵ is a small constant to prevent division by zero.

Despite the popularity of Adam, the no free lunch theorem states that there is no optimization algorithm that can be effective for all problems within a limited search space [223]. Each optimization algorithm has its own strengths and limitations. Therefore, we cannot compare optimization algorithms in isolation from concrete problems.

Optimization algorithms necessitate an initial guess of Θ to initiate the training of neural networks. It is common practice to initialize the biases of neural networks with zeros and randomly draw the weights of neural networks from either a uniform distribution $\mathcal{U}(-r, r)$ or a normal distribution $\mathcal{N}(0, \sigma^2)$. Instead of fixed r and σ , it is better to adjust them based on the numbers of neurons of the previous and current layers (N_{l-1} and N_l), which is a technique known as variance scaling. Tab. 3.1 presents two commonly used weight initialization methods for different activation functions, namely the Glorot (Xavier) initialization method [63] and the He (Kaiming) initialization method [73].

Method	Activation function	Uniform distribution	Normal distribution
		$\mathcal{U}(-r, r)$	$\mathcal{N}(0, \sigma^2)$
Glorot (Xavier) [63]	logistic	$r = 4\sqrt{\frac{6}{N_{l-1}+N_l}}$	$\sigma = 4\sqrt{\frac{2}{N_{l-1}+N_l}}$
	tanh	$r = \sqrt{\frac{6}{N_{l-1}+N_l}}$	$\sigma = \sqrt{\frac{2}{N_{l-1}+N_l}}$
He (Kaiming) [73]	relu	$r = \sqrt{\frac{6}{N_{l-1}}}$	$\sigma = \sqrt{\frac{2}{N_{l-1}}}$

Table 3.1: Weight initialization methods for neural networks with different activation functions

3.4 HYPER-PARAMETER TUNING

The process of training neural networks involves numerous hyper-parameters, which can be broadly divided into three classes:

- Architecture-related hyper-parameters
Number of neurons and layers, activation function, etc.
- Regularization-related hyper-parameters
Penalty strength of ℓ_1 and ℓ_2 regularization, patience of EarlyStopping, probability of a neuron being discarded in Dropout, etc.
- Optimization-related hyper-parameters
Loss function, optimizer, learning rate, batch size, maximum number of epochs, optimizer-related hyper-parameters, etc.

Hyper-parameter tuning is a combinatorial optimization problem that cannot be optimized using gradient-based methods. Additionally, evaluating a set of hyper-parameters typically requires the complete training of neural networks, which may be time-consuming.

As a result, an exhaustive search of hyper-parameters is computationally prohibitive and impractical. Commonly used hyper-parameter tuning methods include grid search, random search, and Bayesian optimization:

- Grid search

Grid search is the simplest way to tune hyper-parameters. We select a set of promising values for each hyper-parameter and create a grid containing all possible combinations of hyper-parameters, and then we evaluate each combination and choose the best one. Grid search is only suitable for a limited number of hyper-parameters and is typically used as a rough tuning.

- Random search

Random search involves sampling each hyper-parameter from a predefined distribution [13], such as the uniform or log-uniform distribution. Random search is more efficient than grid search.

- Bayesian optimization

Grid search and random search do not take into account the information from previous trials when choosing the next set of hyper-parameters to evaluate. On the contrary, Bayesian optimization is an adaptive derivative-free method that suggests the next promising set of hyper-parameters based on the results of previous trials. The basic idea of Bayesian optimization is to prioritize experimentation in regions that have produced high-performing trials. Commonly used Bayesian optimization methods include Sequential Model-Based Optimization [85] [186] for numerical hyper-parameters and Tree-Structured Parzen Estimator for categorical hyper-parameters [12] [14].

Part II

SPEEDING UP PHASE EQUILIBRIUM CALCULATIONS WITH DEEP LEARNING

PTFLASH : A DEEP LEARNING FRAMEWORK FOR TWO-PHASE FLASH CALCULATION

In this chapter, we present our first contribution to accelerating flash calculation and introduce a fast and parallel framework, PTFlash, that uses PyTorch to vectorize algorithms required for two-phase flash calculation and can facilitate a wide range of downstream applications. Vectorization promotes parallelism and consequently leads to attractive hardware-agnostic acceleration. In addition, to further accelerate PTFlash, we design two task-specific neural networks, one for predicting the stability of mixtures and the other for providing estimates of distribution coefficients, which are trained offline and help shorten computation time by sidestepping stability analysis and reducing the number of iterations required for convergence.

This chapter is based on our published work in the journal Fuel:

Qu, Jingang, Thibault Faney, Jean-Charles de Hemptinne, Soleiman Yousef, and Patrick Gallinari. "PTFlash: A vectorized and parallel deep learning framework for two-phase flash calculation." Fuel 331 (2023): 125603.

4.1 INTRODUCTION

Numerical simulation of multi-component multi-phase flow in porous media is an essential tool for many subsurface applications, from reservoir simulation to long term CO_2 storage. A core element of simulators for such applications is to determine the phase distribution of a given fluid mixture at equilibrium, also known as flash calculation. Starting from the seminal work of [129, 130], researchers have developed robust and efficient algorithms for isothermal two-phase flash calculation, which have been well implemented in IFPEN's thermodynamic C++ library — Carnot.

Nonetheless, flash calculations still account for the majority of simulation time in a large variety of subsurface applications [10, 215]. In most simulators, flash calculations are performed for each grid cell at each time step. Moreover, since modern simulators tend to require higher and higher grid resolutions up to billions of grid cells [40], the share of computing time devoted to flash calculations is expected to increase as well. In this context, speeding up flash calculations has drawn increasing research interest.

In this chapter, we introduce PTFlash, a framework for two-phase flash calculation based on the SRK equation of state [187]. PTFlash is built on the deep learning framework PyTorch [150] and consists of two main elements, namely the vectorization of algorithms and the use of neural networks. First, we perform a complete rewrite of two-phase flash calculation algorithms of Carnot using PyTorch. This enables the systematic vectorization of the complex iterative algorithms implemented in Carnot, allowing in turn to efficiently harness modern hardware with the help of, e.g., Advanced Vector Extensions AVX for Intel CPUs [123] and CUDA for NVIDIA GPUs [170]. Note that vectorization of complex iterative algorithms with branching is not straightforward and needs specific care. Second, we replace repetitive and time-consuming subroutines of flash calculation with deep neural networks trained on the exact solution. More specifically, one neural network is used

to predict the stability of mixtures, and the other is used to provide initial estimates for iterative algorithms. Once well trained, neural networks can be seamlessly incorporated into PTFlash. These two elements allow PTFlash to provide substantial speed-ups compared to Carnot, especially in the context of flow simulations where parallel execution of flash calculations for up to billions of grid cells is needed.

The rest of this chapter is structured as follows. In Sec. 4.2, we review related work on speeding up flash calculation through algorithmic improvements and the application of machine learning. In Sec. 4.3, we introduce a novel constrained sampling method for efficiently sampling feed composition. In Sec. 4.4, we describe three case studies used to evaluate our proposed PTFlash. In Sec. 4.5, we describe how to efficiently vectorize flash calculation using PyTorch. In Sec. 4.6, we present two neural networks to further speed up flash calculation. In Sec. 4.7, we compare PTFlash with Carnot to demonstrate the attractive speedups that PTFlash can achieve due to the vectorization of algorithms and the use of neural networks. Finally, we summarize the work presented in this chapter and suggest future directions for improving PTFlash in Sec. 4.8.

4.2 RELATED WORK

Some efforts have been made to accelerate flash calculation. [76, 77, 131] proposed a reduction method aiming to reduce the number of independent variables by leveraging the sparsity of the binary interaction parameter matrix, resulting in a limited speed-up [10]. [166] introduced the shadow region method using the results of previous time steps to initiate the current one, which assumes that the changes in pressure, temperature, and composition of a given block are small between two adjacent time steps in typical compositional reservoir simulation. [211] presented tie-line based methods, which approximate the results of flash calculations through linear interpolation between existing tie-lines and can be seen as a kind of look-up table. In [52–54, 91, 217, 232], the authors focused on the use of machine learning, which provides a collection of techniques that can effectively discover patterns and regularities in data. They used support vector machine [30], relevance vector machine [197] and neural networks [66] to directly predict equilibrium phases and provide more accurate initial estimates for flash calculations. In [27, 40], researchers focused on developing faster parallel linear solvers, with [40] mentioning specifically that the vectorization of partial equations of state-related operations would lead to faster execution.

4.3 DATA GENERATION

4.3.1 Design of experiments (DoE)

In computer simulations, data generation is a well-established field and forms a part of the design of experiments (DoE) that is a systematic approach to planning, conducting, and analyzing experiments in order to identify and understand the relationships between variables and their effects on a particular system or process. A key goal of DoE is to make experiments as informative as possible by efficiently sampling the input space, especially when simulations are time-consuming and computationally expensive, and we are therefore unable to conduct extensive experiments due to limited computational resources. A commonly used technique for this purpose is to generate space-filling DoE, in which samples are evenly distributed over the input space. If input variables can be considered

independent of each other, we can use Latin Hypercube Sampling (LHS) [171], which is a statistical method used to sample input data quasi-randomly from a multidimensional distribution. LHS divides each variable dimension into N intervals, where N is the number of sampling points, and places only one point in each interval so that these data points are evenly spread over the input space. Fig. 4.1 compares random sampling and LHS, showing that LHS is a more effective sampling method in terms of "space-filling".

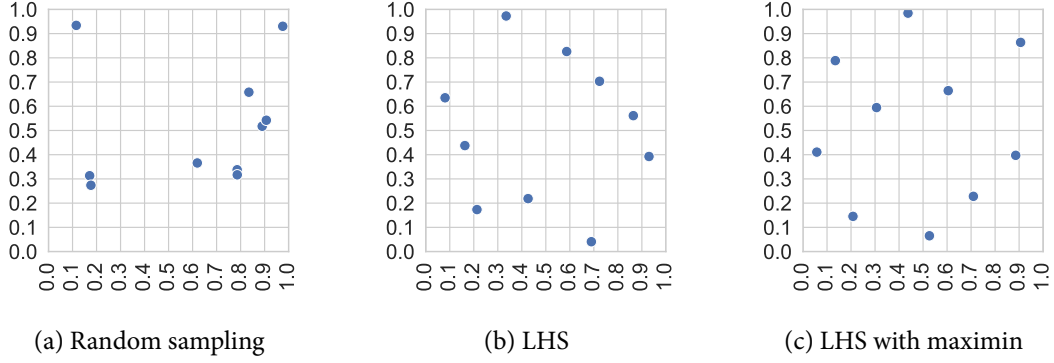


Figure 4.1: We compare random sampling and LHS by generating 10 points in a square using each method. Figure (a): The random sampling leads to some points being so close together that they provide little additional information, resulting in a waste of computational resources. Figure (b): LHS is more effective, with each point located separately in a small square. Figure (c): We use LHS with the maximin criterion aiming to maximize the minimum distance between pairs of points. The maximin criterion leads to more space-filling data at the cost of additional computational overhead, which, however, is not negligible for large numbers of points.

4.3.2 A new method for sampling multiple variables adding up to 1

For our problem, the input space consists of pressure P , temperature T and composition $\mathbf{z} = (z_1, \dots, z_{N_c})$, where N_c is the number of components. In order to sample \mathbf{z} under the constraint $\sum z_i = 1$, a simple approach is to first draw some variables from the uniform distribution $\mathcal{U}(0, 1)$ using LHS and then divide each of them by their sum, as follows:

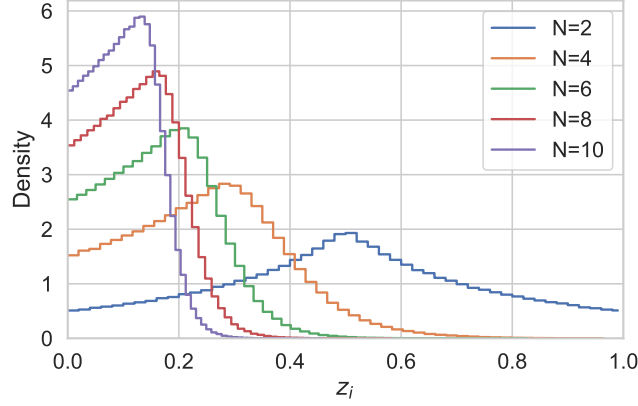
$$x_i \sim U(0, 1) \quad \text{using LHS} \quad (4.1a)$$

$$z_i = \frac{x_i}{\sum_{i=1}^{N_c} x_i} \quad (4.1b)$$

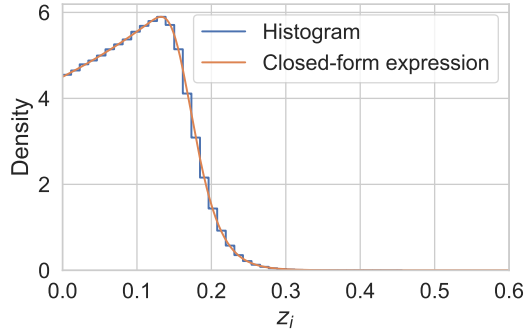
Unfortunately, this simple approach has the problem that it becomes increasingly difficult to obtain large values of z_i as N_c increases, as shown in Fig. 4.2a. We can further derive the closed-form expression of the probability density function (*pdf*) of z_i :

$$\begin{aligned} pdf(z) = & \frac{\sigma}{\sqrt{2\pi}(1-z)^2} \left(\exp\left(-\frac{\mu^2}{2\sigma^2}\right) - \exp\left(-\frac{(\frac{1-z}{z} - \mu)^2}{2\sigma^2}\right) \right) \\ & + \frac{\mu}{(1-z)^2} \left(\Phi\left(\frac{\frac{1-z}{z} - \mu}{\sigma}\right) - \Phi\left(-\frac{\mu}{\sigma}\right) \right) \end{aligned} \quad (4.2)$$

where $\mu = (N_c - 1)/2$, $\sigma^2 = (N_c - 1)/12$, and $\Phi(\cdot)$ denotes the cumulative distribution function of the standard normal distribution. We detail the derivation of $pdf(z)$ in Appendix A. Fig. 4.2b compares the histogram of z_i and $pdf(z)$ for $N_c = 10$. They coincide with each other, which validates $pdf(z)$.



(a) Histogram of z_i with respect to N_c when using Eq. (4.1)



(b) Comparison between the histogram of z_i and $pdf(z)$ for $N_c = 10$

Figure 4.2: Figure (a): The greater N_c is, the lower the probability of large values of z_i . Figure (b): The histogram of z_i and $pdf(z)$ coincide with each other.

To address the aforementioned problem of sampling z with Eq. (4.1), we propose a novel sampling method. Specifically, we transform a set of variables drawn from $\mathcal{U}(0, 1)$ into the Dirichlet distribution $Dir(\alpha)$ whose support is a simplex, as follows:

$$x_i \sim \mathcal{U}(0, 1) \text{ using LHS} \quad (4.3a)$$

$$y_i = \Gamma(\alpha_i, 1).ppf(x_i) \quad (4.3b)$$

$$z_i = \frac{y_i}{\sum_{i=1}^{N_c} y_i} \quad (4.3c)$$

where $\alpha = (\alpha_1, \dots, \alpha_{N_c})$ is the concentration parameter of the Dirichlet distribution and controls its mode, $\Gamma(\alpha_i, 1)$ is the Gamma distribution, and ppf represents the percent-point function, also known as the quantile function. Compared to Eq. (4.1), we introduce an intermediate step — Eq. (4.3b) that transforms $x_i \sim \mathcal{U}(0, 1)$ into $y_i \sim \Gamma(\alpha_i, 1)$, which allows us to get $z \sim Dir(\alpha)$ that satisfies $\sum z_i = 1$. The advantage of this novel sampling method is that we can adjust α to control the mode of $Dir(\alpha)$ to draw more data in the area of interest, as shown in Fig. 4.3.

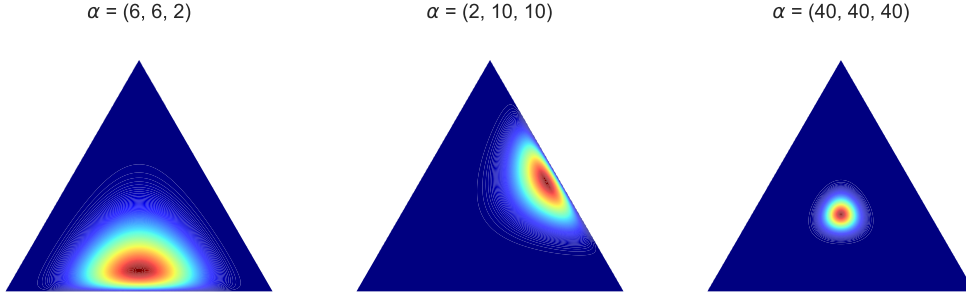


Figure 4.3: This figure visualizes the probability density of the Dirichlet distribution $Dir(\alpha)$ with respect to the concentration parameters α when z is located in a 2-dimensional simplex with $N_c = 3$. In practice, we can sample more data in a region by increasing the probability density of this region through α .

4.4 CASE STUDIES

We introduce three case studies involving hydrocarbons, CO_2 and N_2 , whose properties are shown in Tab. 4.1. Here, we only consider the binary interaction parameter (BIP) between CH_4 and CO_2 , which is 0.0882. The BIPs between the others are 0.

	P_c (MPa)	T_c (K)	ω
CH_4	4.6	190.55	0.0111
C_2H_6	4.875	305.43	0.097
C_3H_8	4.268	369.82	0.1536
$n-C_4H_{10}$	3.796	425.16	0.2008
$n-C_5H_{12}$	3.3332	467.15	0.2635
C_6H_{14}	2.9688	507.4	0.296
$C_7H_{16}^+$	2.622	604.5	0.3565
CO_2	7.382	304.19	0.225
N_2	3.3944	126.25	0.039

Table 4.1: Properties of the components involved in three case studies

The first case study focuses on a system of two components including CH_4 and C_6H_{14} , and the second one involves four components including CH_4 , C_2H_6 , C_3H_8 , and C_4H_{10} . For these two case studies, the ranges of pressure and temperature are 0.1MPa - 10MPa and 200K - 500K, respectively, and we consider the entire compositional space, i.e., $0 < z_i < 1$ for $i = 1, \dots, N_c$. The third case study includes all 9 components in Tab. 4.1. The bounds of pressure and temperature are 5MPa - 25MPa and 200K - 600K, respectively. In addition, from a practical perspective, given that some mixtures do not exist in nature, rather than considering the entire compositional space, we specify four different compositional ranges, as shown in Tab. 4.2. Each compositional range represents one of the common reservoir fluid types, namely wet gas, gas condensate, volatile oil, and black oil. Fig. 4.4 shows the phase diagrams of these four reservoir fluids at the fixed compositions defined in Tab. 4.3. We can see that the more heavy hydrocarbons there are, the lower the pressure range of the phase envelope and the less volatile the fluid is.

	Wet gas	Gas condensate	Volatile oil	Black oil
CH_4	80% - 100%	60% - 80%	50% - 70%	20% - 40%
C_2H_6	2% - 7%	5% - 10 %	6% - 10%	3% - 6 %
C_3H_8	$\leq 3\%$	$\leq 4\%$	$\leq 4.5\%$	$\leq 1.5\%$
$n-C_4H_{10}$	$\leq 2\%$	$\leq 3\%$	$\leq 3\%$	$\leq 1.5\%$
$n-C_5H_{12}$	$\leq 2\%$	$\leq 2\%$	$\leq 2\%$	$\leq 1\%$
C_6H_{14}	$\leq 2\%$	$\leq 2\%$	$\leq 2\%$	$\leq 2\%$
$C_7H_{16}^+$	$\leq 1\%$	5% - 10 %	10% - 30%	45% - 65%
CO_2	$\leq 2\%$	$\leq 3.5\%$	$\leq 2\%$	$\leq 0.1\%$
N_2	$\leq 0.5\%$	$\leq 0.5\%$	$\leq 0.5\%$	$\leq 0.5\%$

Table 4.2: Four fluid types characterized by different compositional ranges

	Wet gas	Gas condensate	Volatile oil	Black oil
CH_4	92.46%	73.19%	57.6%	33.6%
C_2H_6	3.18%	7.8%	7.35%	4.01%
C_3H_8	1.01%	3.55%	4.21%	1.01%
$n-C_4H_{10}$	0.52%	2.16%	2.81%	1.15%
$n-C_5H_{12}$	0.21%	1.32%	1.48%	0.65%
C_6H_{14}	0.14%	1.09%	1.92%	1.8%
$C_7H_{16}^+$	0.82%	8.21%	22.57%	57.4%
CO_2	1.41%	2.37%	1.82%	0.07%
N_2	0.25%	0.31%	0.24%	0.31%

Table 4.3: Some typical reservoir fluid compositions

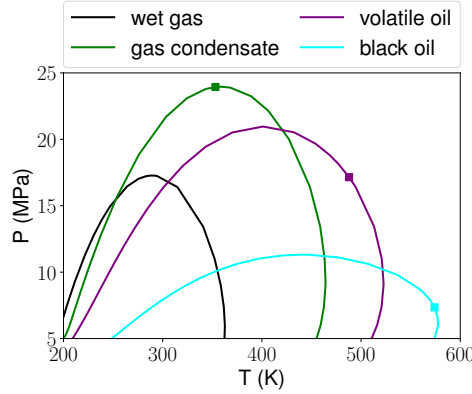


Figure 4.4: This figure shows the phase envelopes of four typical reservoir fluids at fixed compositions. The squares on the phase envelopes represent critical points.

We use the novel sampling method presented in Sec. 4.3.2 to sample z . For the first two case studies, the concentration parameters are $\alpha = 1$, i.e., all-ones vector. For the third case study, we adjust α for different fluid types to make the probability of each compositional range as large as possible, as shown in Tab. 4.4. Fig. 4.5 presents the marginal distribution of z_i for black oil. In summary, we sample z using Eq. (4.3) with different α specified in Tab. 4.4, and then we single out the acceptable samples located in the compositional ranges defined in Tab. 4.2. In the following, unless otherwise specified, four fluid types are always equally represented.

	α_1 for CH_4	α_2 for C_2H_6	α_7 for $C_7H_{16}^+$	α_i for others
Wet gas	100	5	1	1
Gas condensate	40	5	5	1
Volatile oil	55	8	20	1
Black oil	25	4	40	1

Table 4.4: Concentration parameters α for different fluid types in Tab. 4.2

For P and T , we simply use LHS to draw space-filling samples from $\mathcal{U}(0, 1)$ and then linearly transform the samples to the expected ranges. Eventually, the samples of P , T , and z are concatenated together to form the complete input data.

4.5 VECTORIZATION OF TWO-PHASE FLASH CALCULATION

We vectorize two-phase flash calculation so that it takes a batch of samples as input. In this case, pressure \mathbf{P} and temperature \mathbf{T} are vectors, i.e., $\mathbf{P} = (P_1, \dots, P_n)$ and $\mathbf{T} = (T_1, \dots, T_n)$, and composition z is a matrix, i.e., $z = (z_1, \dots, z_n)$, where n denotes the number of samples processed concurrently and is often referred to as the batch dimension.

In recent years, Automatic Vectorization (AV) has emerged and developed. A remarkable example is the high-performance computing library JAX [19] developed by Google. JAX defines a set of primitive operations with preset batching rules. At compile time, JAX traces a function and decomposes it into primitive operations. Then, JAX can vectorize the compiled function by directly applying batching transformations to the primitive operations. In this

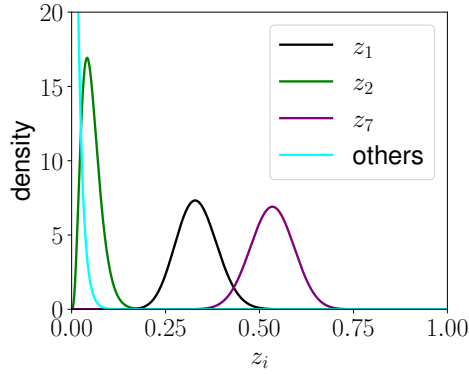


Figure 4.5: This figure shows the marginal distribution of z_i for black oil, and z_1 , z_2 , and z_7 are the molar fractions of CH_4 , C_2H_6 and $C_7H_{16}^+$, respectively.

way, the vectorized function can process a batch of inputs simultaneously rather than processing them one at a time in a loop.

However, AV is limited to predefined primitive operations and is slower than well-designed manual vectorization, which vectorizes a function by carefully revamping its internal operations to accommodate to a batch of inputs. For example, matrix-vector products for a batch of vectors can be directly replaced with a matrix-matrix product. Additionally, flash calculation has an iterative nature and complicated control flow, which may cause the failure of AV. Therefore, for finer-grained control, more flexibility, and better performance, we manually vectorize all algorithms involved in flash calculation, including the solution of the SRK equation of state and the Rachford-Rice equation, stability analysis, and phase split calculations.

One challenge in achieving efficient vectorization is asynchronous convergence, that is, for each algorithm, the number of iterations required to reach convergence generally varies for different samples, which hinders vectorization and parallelism. To alleviate this problem, we designed a general-purpose paradigm, synchronizer, that saves converged results at the end of each iteration and removes the corresponding samples to avoid wasting computational resources on them in subsequent iterations. This is achieved by leveraging a one-dimensional Boolean mask that encapsulates convergence information to efficiently access data in vectors and matrices, as shown in Algorithm 2. The number of unconverged samples decreases over time as a result of incremental convergence. We can use the synchronizer to wrap and vectorize any iterative algorithm. For instance, we illustrate how to perform vectorized stability analysis in Algorithm 3.

The efficiency of the synchronizer may be questioned because previously converged samples must wait for unconverged ones before moving on to the next step. However, this is not a significant issue, as we strive to minimize the waiting time as much as possible. For instance, if the successive substitution fails to converge quickly, we immediately switch to the trust-region method. In any case, the delay caused by waiting is insignificant compared to the acceleration due to vectorization. In addition, we leverage neural networks to provide more accurate initial estimates for iterative algorithms so that all samples converge as simultaneously as possible, thereby reducing asynchrony. We will discuss this further in Sec. 4.6.

Once all algorithms are well vectorized, another challenge is how to globally coordinate the different subroutines of flash calculation. To this end, we add barrier synchronization to

Algorithm 2: PyTorch pseudo-code of synchronizer to save converged results after iteration and remove the corresponding samples

Input: Vectorized iterated function $f(\mathbf{X}, \mathbb{O})$, initial estimate $\mathbf{X}^{(0)}$, other f -related inputs \mathbb{O} , convergence criterion C , maximum number of iterations K

```

1 Initialization
2   Set the number of iterations  $k \leftarrow 1$ 
3   Generate a vector  $\mathbf{i}$  containing indices from 0 to  $n - 1$ 
   /*  $n$  is the number of samples and indexing starts from 0. */
4   Create a placeholder matrix  $\widetilde{\mathbf{X}}$  of the same shape as  $\mathbf{X}^{(0)}$ 
5 while  $k \leq K$  do
6    $\mathbf{X}^{(k+1)} \leftarrow f(\mathbf{X}^{(k)}, \mathbb{O})$ 
7    $\text{mask} \leftarrow C(\dots)$ 
   /*  $C$  returns a Boolean vector and True means convergence. */
8   Saving
9   |    $\text{indices} \leftarrow \mathbf{i}[\text{mask}]$ 
10  |    $\widetilde{\mathbf{X}}[\text{indices}] \leftarrow \mathbf{X}^{(k+1)}[\text{mask}]$ 
11  Removing
12  |    $\mathbf{i} \leftarrow \mathbf{i}[\sim \text{mask}]$ 
13  |    $\mathbb{O} \leftarrow \mathbb{O}[\sim \text{mask}]$ 
   /* Apply this operation to every element in  $\mathbb{O}$  and  $\sim$  denotes the
   logical NOT operator. */
14  |    $\mathbf{X}^{(k+1)} \leftarrow \mathbf{X}^{(k+1)}[\sim \text{mask}]$ 
15  |    $k \leftarrow k + 1$ 
16 if  $\text{len}(\mathbf{i}) \neq 0$  then
17  |    $\widetilde{\mathbf{X}}[\mathbf{i}] \leftarrow \mathbf{X}$ 
   /* Also save unconverged results for further utilization. */

```

Output: Converged results $\widetilde{\mathbf{X}}$ and unconverged indices \mathbf{i}

the entry points of stability analysis and phase split calculations in Fig. 2.3, which prevents any subroutine connected to it from proceeding further until all other subroutines terminate and arrive at this barrier.

We also optimized the code using TorchScript [150], allowing for more efficient execution through algebraic peephole optimizations and fusion of some operations, as well as more practical asynchronous parallelism without the Python global interpreter lock [201], whereby vapor-like and liquid-like estimates are dealt with in parallel in stability analysis.

4.6 ACCELERATION OF FLASH CALCULATION USING NEURAL NETWORKS

To further accelerate flash calculation, we create and train two task-specific neural networks, classifier and initializer. The classifier is used to predict the probability p that a mixture is stable, i.e., $p = \text{classifier}(P, T, \mathbf{z})$, which involves a binary classification problem. It helps bypass stability analysis and thus save time. The initializer is able to initialize K_i more accurately than the Wilson approximation, i.e., $\ln K_i = \text{initializer}(P, T, \mathbf{z})$, which relates to a regression problem. It can reduce the number of iterations required to reach convergence and alleviate the asynchronous convergence we introduced before. Note that the neural

Algorithm 3: PyTorch pseudo-code of vectorized stability analysis

Input: Pressure \mathbf{P} , temperature \mathbf{T} , feed composition \mathbf{z} , component properties ($\mathbf{P}_c, \mathbf{T}_c, \boldsymbol{\omega}$, BIPs), initial estimate $\mathbf{W}^{(0)}$, convergence criteria C_{ss} and C_{tr} , maximum numbers of iterations $K_{ss} = 9$ and $K_{tr} = 20$

- 1 Initialization
- 2 | Instantiate $pteos = PTEOS(\mathbf{P}_c, \mathbf{T}_c, \boldsymbol{\omega}, \text{BIPs})$
 | */* PTEOS is a PyTorch-based class to efficiently calculate the fugacity coefficients and their partial derivatives. */*
- 3 Successive substitution
- 4 | Iterated function f_{ss} specified by Eq. (2.39)
- 5 | Other inputs $\mathbb{O}_{ss} \leftarrow \{\mathbf{P}, \mathbf{T}, \mathbf{z}\}$
- 6 | $\mathbf{W}, \mathbf{i}_{ss} \leftarrow \text{synchronizer}(f_{ss}, \mathbf{W}^{(0)}, \mathbb{O}_{ss}, \mathbb{C}_{ss}, K_{ss})$
- 7 Trust-region method
- 8 | Iterated function f_{tr} specified by Eq. (2.41)
- 9 | $\mathbf{W}_{tr}^{(0)} \leftarrow \mathbf{W}[\mathbf{i}_{ss}]$
- 10 | Other inputs $\mathbb{O}_{tr} \leftarrow \{\mathbf{P}[\mathbf{i}_{ss}], \mathbf{T}[\mathbf{i}_{ss}], \mathbf{z}[\mathbf{i}_{ss}]\}$
- 11 | $\mathbf{W}_{tr}, \mathbf{i}_{tr} \leftarrow \text{synchronizer}(f_{tr}, \mathbf{W}_{tr}^{(0)}, \mathbb{O}_{tr}, \mathbb{C}_{tr}, K_{tr})$
- 12 $\mathbf{W}[\mathbf{i}_{ss}] \leftarrow \mathbf{W}_{tr}$ and $\mathbf{i} \leftarrow \mathbf{i}_{ss}[\mathbf{i}_{tr}]$

Output: Converged results \mathbf{W} and unconverged indices \mathbf{i}

networks presented in this section are dedicated to the case study containing 9 components. However, the basic architecture and training method of neural networks can be generalized to any case.

4.6.1 Classifier

4.6.1.1 Architecture

As shown in Fig. 4.6, the classifier has 3 hidden layers with 32 neurons and uses the SiLU activation function [45, 78, 164]. The output layer has only one neuron and uses the sigmoid activation function compressing a real number to the range (0, 1). The input \mathbf{x} consists of P, T , and \mathbf{z} , and the output is the probability p that a mixture is stable. The scaling layer standardizes the input as $(\mathbf{x} - \mathbf{u})/s$, where \mathbf{u} and s are the mean and standard deviation of \mathbf{x} over the training set. To train the classifier, we use the binary cross-entropy loss (bce), which is the de-facto loss function for binary classification problems and is defined as:

$$\text{bce}(y, p) = y \ln p + (1 - y) \ln(1 - p) \quad (4.4)$$

where y is either 0 for unstable mixtures or 1 for stable ones.

The architecture of the classifier is obtained by tuning hyper-parameters using Tree-Structured Parzen Estimator optimization algorithm [12] with Asynchronous Successive Halving algorithm [114] as an auxiliary tool to early stop less promising trials. We create a dataset containing 100,000 samples (80% for training and 20% for validation), and then tune the hyper-parameters of the classifier with 150 trials to minimize the loss on the validation set (we use Adam [95] as optimizer and the batch size is 512), as shown in Fig. 4.7. We can see that SiLU largely outperforms other activation functions.

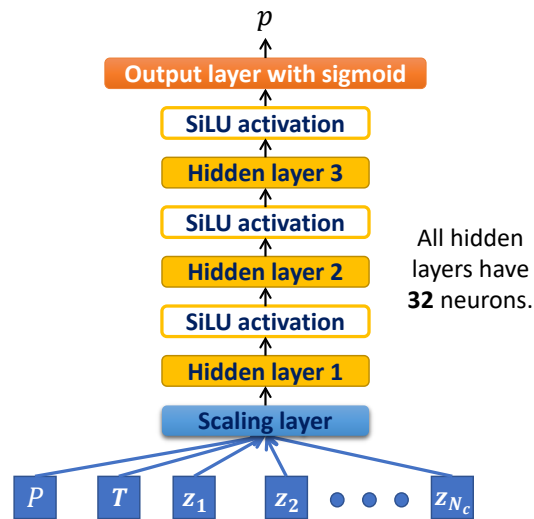


Figure 4.6: Architecture of the classifier for the case study containing 9 components

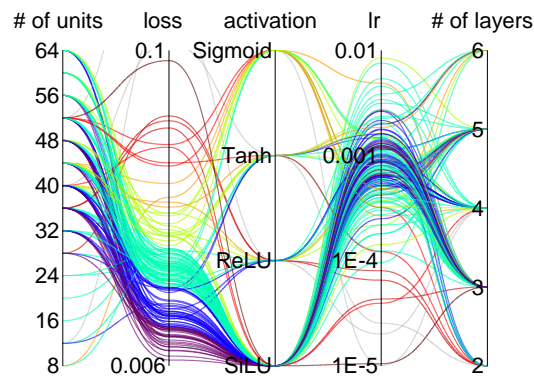


Figure 4.7: This figure is a parallel coordinates plot used to visualize the results of tuning hyperparameters of the classifier, where lr stands for the learning rate. The colors of lines are mapped to the value of the loss on the validation set.

4.6.1.2 Training

We first generate one million samples and then feed them to PTFlash to determine stability (no need for phase split calculations), which takes about 2 seconds. Subsequently, these samples are divided into the training (70%), validation (15%) and test (15%) sets. To train the classifier, we set the batch size to 512 and use Adam with Triangular Cyclic Learning Rate (CLR) [183, 184], which periodically increases and decreases the learning rate during training, as shown in Fig. 4.8. [185] claimed that CLR helps escape local minima and has the opportunity to achieve superb performance using fewer epochs and less time. We found that Adam with and without CLR achieve similar performance, but the former converges five times faster than the latter. Early stopping is also used to avoid overfitting [157]. The total training time is about 5 minutes using Nvidia RTX 3080.

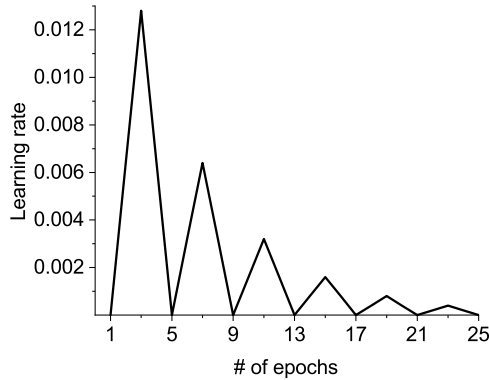


Figure 4.8: This figure shows how the learning rate varies cyclically during the training of classifier.

The final performance of the classifier on the test set is $bce = 0.002$ and accuracy = 99.93%. For a more intuitive understanding of performance, Fig. 4.9 shows the contours of probabilities predicted by the classifier, where the blue contour of $p = 0.5$ basically coincides with the phase envelope. In the zoomed inset, the additional green and yellow contours correspond to $p=0.02$ and 0.98 , respectively.

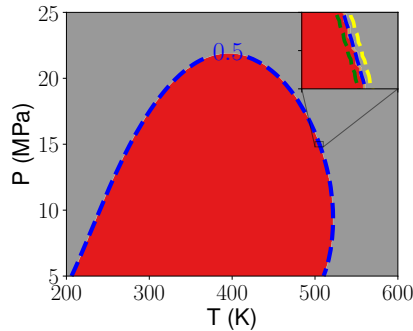


Figure 4.9: This figure illustrates the contours of probabilities predicted by the classifier for volatile oil at the fixed composition defined in Tab. 4.3. The gray and red correspond to the monophasic and two-phase regions, respectively.

4.6.2 Initializer

4.6.2.1 Architecture

As shown in Fig. 4.10, the input of the initializer includes P , T , and z , and its output is logarithmic fugacity coefficients $\ln K_i$. The initializer has 1 hidden layer and 3 residual blocks. Each residual block has 2 hidden layers and a shortcut connection adding the input of the first hidden layer to the output of the second [74]. All hidden layers have 64 neurons and use the SiLU activation function. The output layer has N_c neurons without an activation function. The wide shortcut, proposed in [28], enables neural networks to directly learn simple rules via it besides deep patterns through hidden layers, which is motivated by the fact that the inputs, such as P and T , are directly involved in the calculation of K_i . The concat layer concatenates the input layer and the outputs of the last residual block (the concatenation means putting two matrices $A \in \mathbb{R}^{d_1 \times d_2}$ and $B \in \mathbb{R}^{d_1 \times d_3}$ together to form a new one $C \in \mathbb{R}^{d_1 \times (d_2 + d_3)}$). In addition, the targets of the initializer are $\ln K_i$ instead of K_i , since K_i varies in different orders of magnitude, which hampers the training of the initializer, whereas $\ln K_i$ does not.

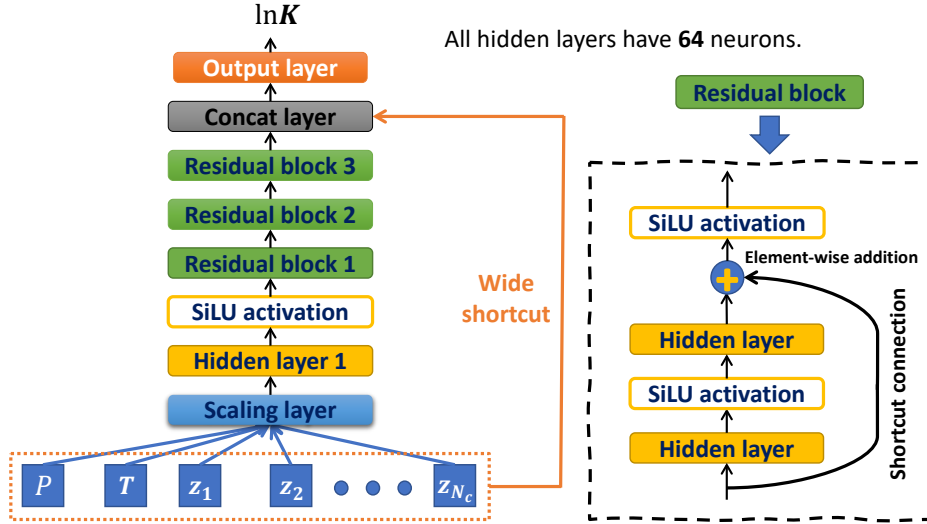


Figure 4.10: Architecture of the initializer for the case study containing 9 components

We found that the convergence of phase split calculations is robust if the distribution coefficients predicted by the initializer can lead to a more accurate vapor fraction, especially around critical points where calculations are sensitive to initial distribution coefficients and prone to degenerate into trivial solutions. Therefore, the loss function used to train the initializer consists of two parts, one is the mean absolute error (mae) in terms of $\ln \mathbf{K}$ and the other is mae in terms of θ_V , as follows:

$$\text{mae}(\ln \mathbf{K}, \ln \hat{\mathbf{K}}) = \sum_{i=1}^{N_c} |\ln K_i - \ln \hat{K}_i| \quad (4.5a)$$

$$\text{mae}(\theta_V, \hat{\theta}_V) = |\theta_V - \hat{\theta}_V| \quad (4.5b)$$

where \mathbf{K} is the ground truth, $\hat{\mathbf{K}}$ is the prediction of the initializer, θ_V is the true vapor fraction at equilibrium, and $\hat{\theta}_V$ is obtained by solving the Rachford-Rice equation f_{RR} given z and $\hat{\mathbf{K}}$.

4.6.2.2 Training

We generate one million samples in the two-phase region (\mathbf{K} is not available in the monophasic region), which are divided into the training (70%), validation (15%) and test (15%) sets. The training of the initializer is carried out in two stages. First, we train it to minimize $\text{mae}(\ln \mathbf{K}, \ln \hat{\mathbf{K}})$, using Adam with CLR and setting the batch size to 512. Second, after the above training, we further train it to minimize $\text{mae}(\ln \mathbf{K}, \ln \hat{\mathbf{K}}) + \text{mae}(\theta_V, \hat{\theta}_V)$, using Adam with a small learning rate of $1e-5$.

Here, $\partial \hat{\theta}_V / \partial \hat{\mathbf{K}}$ is required during backpropagation and can be simply computed via PyTorch's automatic differentiation, which, however, is slow and inefficient. We solve the Rachford-Rice equation in an iterative manner described in Sec. 2.4.3. In this case, automatic differentiation has to differentiate through all the unrolled iterations. Moreover, all intermediate results also need to be retained to evaluate intermediate derivatives, which consumes more memory.

Instead, we can make use of the implicit function theorem [101] to directly obtain $\partial \hat{\theta}_V / \partial \hat{\mathbf{K}}$ by using the derivative information at the solution point of the Rachford-Rice equation. We differentiate the Rachford-Rice equation with respect to \mathbf{K} (note that θ_V is an implicit function of \mathbf{K}) and get:

$$\partial_{\theta_V} f_{RR}(\theta_V, \mathbf{K}) \times \partial \theta_V / \partial \mathbf{K} + \partial_{\mathbf{K}} f_{RR}(\theta_V, \mathbf{K}) = 0 \quad (4.6)$$

We rearrange the above equation and get:

$$\partial \theta_V / \partial \mathbf{K} = -[\partial_{\theta_V} f_{RR}(\theta_V, \mathbf{K})]^{-1} \partial_{\mathbf{K}} f_{RR}(\theta_V, \mathbf{K}) \quad (4.7)$$

Moreover, since $\partial_{\theta_V} f_{RR}(\theta_V, \mathbf{K})$ is a scalar, we can further reduce the above equation to:

$$\partial \theta_V / \partial \mathbf{K} = -\frac{\partial_{\mathbf{K}} f_{RR}(\theta_V, \mathbf{K})}{\partial_{\theta_V} f_{RR}(\theta_V, \mathbf{K})} \quad (4.8)$$

Eq. (4.8) is obviously more efficient than automatic differentiation. For more details and a defense of the above derivation, refer to [101]. After training, the performance of the initializer on the test set is $\text{mae} = 9.66e-4$ in terms of $\ln \mathbf{K}$ and $\text{mae} = 1.86e-3$ in terms of \mathbf{K} .

4.6.3 Strategy for accelerating flash calculation using neural networks

As shown in Fig. 4.11, given P, T and z , we first use the classifier to predict p . Next, based on two predefined thresholds, p_l and p_r ($p_l \leq p_r$), a mixture is thought of as unstable if $p \leq p_l$ or stable if $p \geq p_r$. If $p_l < p < p_r$, we will use stability analysis to avoid unexpected errors. Here, we can adjust p_l and p_r to trade reliability for speed. In general, fewer errors occur with smaller p_l and greater p_r , but it probably takes more time on stability analysis, and vice versa. A special case is $p_l = p_r = p_c$, where p_c could be a well-calibrated probability or simply set to 0.5, which means that we completely trust the classifier (i.e., stable if $p \geq p_c$ or unstable otherwise), and no extra stability analysis is required. For the initializer, it serves both stability analysis if $p_l < p < p_r$ and phase split calculations.

Neural networks can also be used individually. If only the classifier is available, we can initialize \mathbf{K} via the Wilson approximation rather than the initializer in Fig. 4.11. If only the initializer is available, we can use it to initialize K_i in Fig. 2.3.

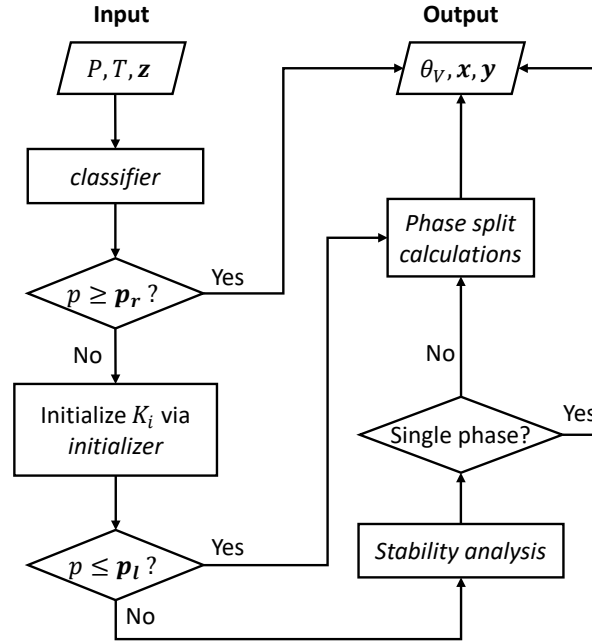


Figure 4.11: Acceleration of flash calculation using neural networks

4.7 RESULTS

In this section, we compare our proposed framework for vectorized flash calculation, PTFlash, with Carnot, an in-house thermodynamic library developed by IFP Energies nouvelles. Carnot is based on C++ and performs flash calculations one at a time on CPU. Regarding the hardware, CPU is Intel 11700F and GPU is NVIDIA RTX 3080 featuring 8704 CUDA cores and 10G memory. We found that using multiple cores of CPU renders the frequency quite unstable due to heat accumulation. Therefore, we use only one core of CPU so that the frequency can be stabilized at 4.5GHz, which allows for a consistent criterion for measuring the execution time.

PTFlash and Carnot give identical results (coincidence to 9 decimal places under double-precision floating-point format) because they use exactly the same convergence criteria for all iterative algorithms. In the following, we will focus on comparing their speeds.

4.7.1 Vectorized flash calculation

We compare the execution time of different methods for flash calculation with respect to the workload quantified by the number of samples n , as shown in Fig. 4.12. Due to GPU memory limitations, the maximum number of samples allowed for the three case studies is 10, 5, and 1 million, respectively. We can see that all three case studies exhibit the same behavior in Fig. 4.12. When the workload is relatively low, e.g., $n < 1000$, Carnot wins by large margins, and CPU is also preferable based on the fact that PTFlash runs much faster on CPU than on GPU. On the one hand, PyTorch has some fixed overhead in the setup of the working environment, e.g., the creation of tensors. On the other hand, when GPU is used, there are some additional costs associated with CPU-GPU communication and synchronization. When n is small, these overheads dominate. As proof, we can see that the

time of PTFlash on GPU hardly changes as n varies from 100 to 10^4 . In contrast, the time of Carnot is almost proportional to n .

As the workload increases, the strength of PTFlash on GPU emerges and becomes increasingly prominent. In three case studies, PTFlash on GPU is 163.4 (2 components), 106.3 (4 components) and 50.5 (9 components) times faster than Carnot at the maximum number of samples. This suggests that PTFlash on GPU is more suitable for large scale computation. Interestingly, we can observe that PTFlash on CPU also outperforms Carnot when the workload is relatively heavy, e.g., $n > 10^3$. In fact, thanks to Advanced Vector Extensions, vectorization enables fuller utilization of the CPU's computational power.

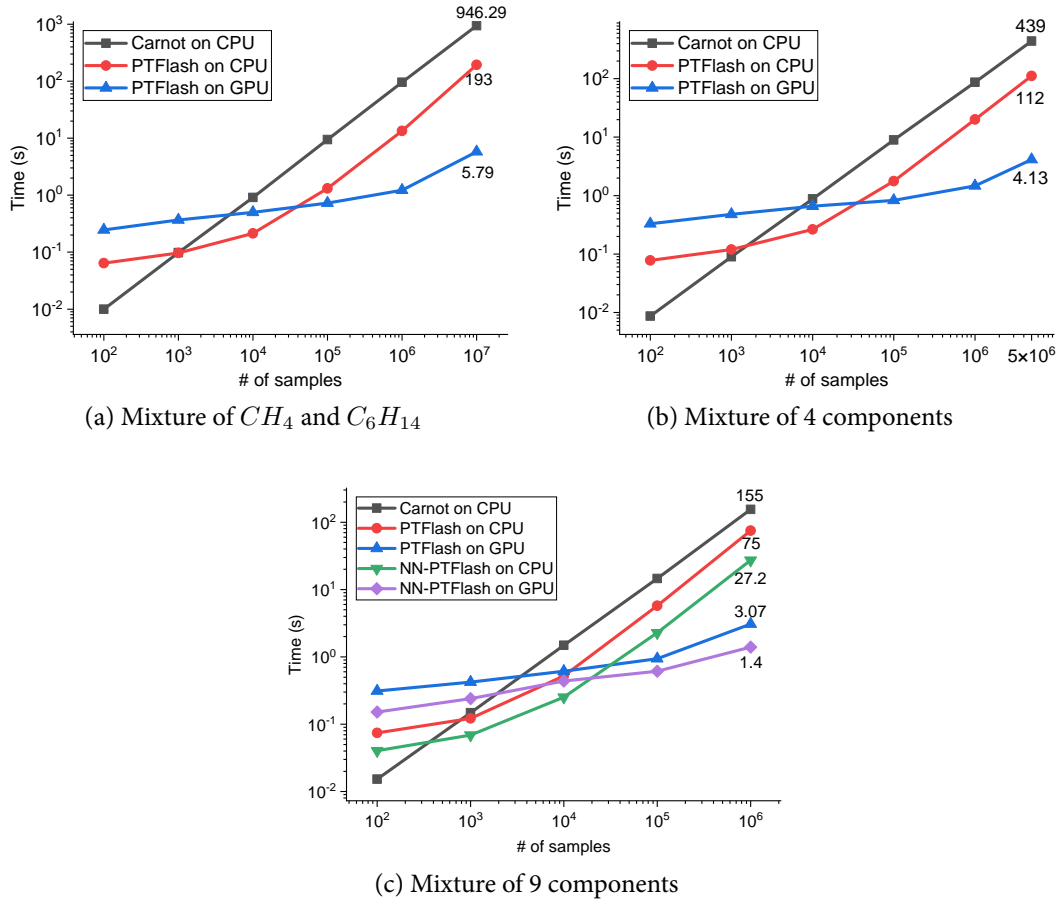


Figure 4.12: Comparison between PTFlash and Carnot in terms of speed. NN-PTFlash is PTFlash accelerated by neural networks, as presented in Sec. 4.6.

We notice that there is a lack of a comprehensive and unified benchmark for the runtime of flash calculation in the literature. Here, we give an article with a case study similar to ours for readers' reference [135], which claimed that the total computation time of flash calculations is 10 seconds for one million samples of a 9-component mixture. However, it is worth pointing out that the sampling method, convergence criteria, and algorithm implementation in this reference article are different from ours. In our work, these aspects are consistent for both Carnot and PTFlash to ensure a fair comparison.

Next, we focus on the mixture of 9 components and analyze the performance of PTFlash for this case study. Tab. 4.5 is a performance profiler of PTFlash on GPU at $n = 10^6$, which records the running time of each subroutine of flash calculations. As a complement to Tab. 4.5, Fig. 4.13 dissects phase split calculations by tracking the total elapsed time and the

convergence percentage up to each iteration, and Fig. 4.14 calculates the mean of critical distances d_c of converged samples at each iteration, where d_c is defined as:

$$d_c = \sqrt{\sum_{i=1}^{N_c} \ln K_i^2} \quad (4.9)$$

The closer to critical points, the smaller d_c . Therefore, d_c indicates the closeness to critical points.

The observations of Fig. 4.13 and Fig. 4.14 are summarized as follows: (1) In Fig. 4.13a, the slope of time with respect to the number of iterations decreases because the workload is gradually reduced due to incremental convergence. (2) In Fig. 4.13b, for the samples that do not converge after the successive substitution, the majority of them (92.67%) converge after 3 iterations of the trust-region method. (3) In Fig. 4.14, d_c decreases during iterations, which means that samples close to critical points converge last and also confirms that convergence is slow in the vicinity of critical points.

Table 4.5: Performance profiler of PTFlash on GPU for the mixture of 9 components at $n = 10^6$ in Fig. 4.12c

	ss of phase split calculations	Stability analysis				Phase split calculations	
		vapor-like estimate		liquid-like estimate		ss	tr
	ss	tr	ss	tr	ss		
# of samples	10^6	625645	130715	625645	90179	413442	223741
Convergence	37.44% ¹	79.11%	100%	85.59%	100%	45.88%	100%
Max number of iterations	3	9	18	9	16	9	13
Total time	0.4565s	0.4136s	0.3417s	0.4044s	0.2706s	0.7412s	0.5132s
		1.3237s ²				1.2544s	
		ss: successive substitution				tr: trust-region method	

¹ 37.44% is the percentage of samples for which any of ΔG , tpd_x and tpd_y is negative after 3 attempts of successive substitution of phase split calculations, as described in 2.4.4.

² The total time of stability analysis is less than the sum of the times of its subroutines because vapor-like and liquid-like estimates are handled concurrently.

The above analysis provides a general understanding of PTFlash, but it is not straightforward to comprehensively analyze PTFlash because each subroutine also contains iterative algorithms, such as the solution of the SRK equation of state and the Rachford-Rice equation. However, given the information already obtained, we know that we need to shorten the time of stability analysis and reduce the number of iterations in order to accelerate PTFlash, which is exactly the role of the classifier and initializer.

4.7.2 Deep-learning-powered vectorized flash calculation

We trained neural networks, classifier and initializer, following Sec. 4.6 for the case study containing 9 components. In this section, we will explore the effects of these neural networks.

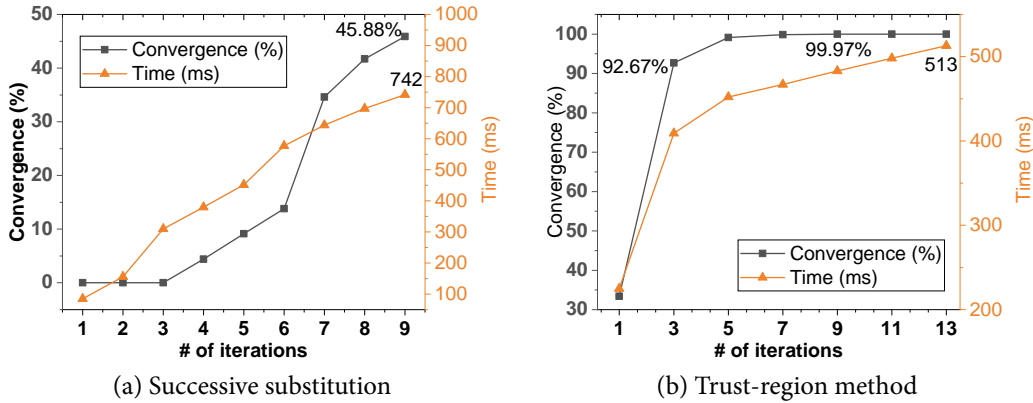


Figure 4.13: Figures (a) and (b) show the convergence percentage and the elapsed time up to each iteration of phase split calculations of PTFlash on GPU.

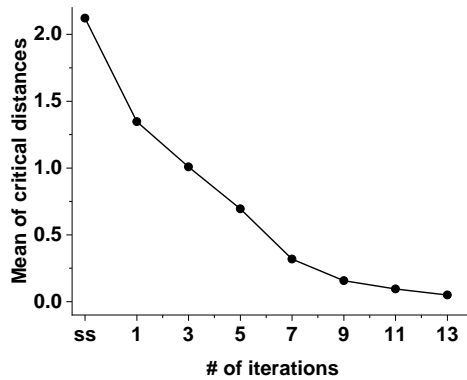


Figure 4.14: This figure shows the average of the critical distances of converged samples at each iteration. On the x-axis, ss corresponds to the end of successive substitution, and other integers are the number of iterations of the trust-region method.

First of all, we set $p_l = 0.02$ and $p_r = 0.98$ as the thresholds of instability and stability, which are carefully chosen so that no misclassification occurs. In Fig. 4.12c, we can see that NN-PTFlash outpaces PTFlash on both CPU (2.7x speed-up) and GPU (2.2x speed-up). In addition, NN-PTFlash on GPU runs almost 110.7 times faster than Carnot at $n = 10^6$.

Tab. 4.6 is the performance profiler of NN-PTFlash on GPU. We can see that the classifier can determine the stability of the vast majority of samples (99.42%), which significantly relieves the burden of stability analysis and saves time. In addition, compared to phase split calculations of PTFlash, the convergence percentage of successive substitution increases from 45.88% to 67.40%, and the overall time is also greatly reduced, which is attributed to better distribution coefficients provided by the initializer.

We also conduct ablation studies to compare the contributions of the classifier and initializer by using them individually. When handling 1 million samples for the case study containing 9 components, NN-PTFlash with only the classifier on GPU takes 1.88s. However, the attempt to use the initializer alone fails because its outputs may reach unreasonably large values, e.g., $1e15$, for stable mixtures far from the boundary between the monophasic and two-phase regions, leading to numerical overflow. In machine learning terminology, this is the out-of-distribution generalization problem, since the initializer is trained on the two-phase region and may suffer from large predictive errors when used within the monophasic region. Nonetheless, there is no problem when the initializer is used in con-

Table 4.6: Performance profiler of NN-PTFlash on GPU (Fig. 4.11) for the mixture of 9 components at $n = 10^6$ in Fig. 4.12c

	classifier	Stability analysis				Phase split	
		vapor-like estimate		liquid-like estimate		calculations	
		ss	tr	ss	tr	ss	tr
# of samples	10^6	5818	1073	5818	1704	413442	134786
Convergence	99.42% ¹	81.56%	100%	70.71%	100%	67.40%	100%
Max number of iterations		9	13	9	12	9	13
Total time	0.0005s	0.1365s	0.128s	0.0514s	0.12s	0.7043s	0.3388s
		0.34s				1.0431s	
		ss: successive substitution				tr: trust-region method	

¹ 99.42% includes 58.38% predicted as stable (i.e., $p > p_r$) and 41.04% predicted as unstable (i.e., $p < p_l$).

junction with the classifier, as the remaining samples located in the monophasic region are fairly close to the boundary after being filtered through the classifier, as shown in Fig. 4.9. Based on the fact that NN-PTFlash using only the classifier consistently performs worse than that using both, we can conclude that both the classifier and initializer play an important role in speeding up flash calculations.

4.7.3 Discussion

The results demonstrate that the systematic and comprehensive vectorization of two-phase flash calculation leads to significant speed-ups when large-scale computation is involved, e.g., the number of samples to process is on the order of millions. Importantly, this speed-up does not compromise accuracy and stability, unlike related work [53, 54, 91, 217] that is subject to the unreliability of machine learning models. In addition, we can see that the classifier and initializer really make a big difference.

Due to GPU memory limitations, the number of samples n is limited in Fig. 4.12. However, we can see that the slopes of time versus n differ significantly between different methods. The time for Carnot is proportional to n , while the time for PTFlash on GPU increases slowly. Therefore, it is reasonable to assume that PTFlash on GPU will have an increasingly pronounced speed advantage as n increases.

Using PyTorch has several benefits in addition to its simplicity and flexibility. First, we can seamlessly incorporate neural networks into PTFlash. Second, any subroutine of PTFlash is fully differentiable through PyTorch’s automatic differentiation, and we can also leverage the implicit function theorem for efficient differentiation, as we did in Sec. 4.6.2.2. Third, PyTorch’s optimized and ready-to-use multi-GPU parallelization largely circumvents the painstaking hand-crafted efforts.

PTFlash also has several limitations. First, PTFlash is based on the SRK equation of state, which is adequate for mixtures containing hydrocarbons and non-polar components, but does not take into account the effect of hydrogen bonding and falls short of adequacy

for cross-associating mixtures containing polar components, such as water and alcohol [98]. In this case, more advanced but also more complicated equations of state should be employed, such as the SAFT equation of state [24, 83, 219–222] or the CPA equation of state [99, 100]. However, the vectorization of these complicated equations of state is much more challenging than that of cubic equations of state. To alleviate this problem, we plan to use neural networks to directly predict fugacity coefficients, allowing us to vectorize the calculation of fugacity coefficients regardless of the equation of state used. Second, PTFlash consumes a significant amount of GPU memory, severely limiting its use on much larger batches of data. We need to optimize PTFlash to reduce GPU memory consumption, e.g., by leveraging the sparsity and symmetry of matrices. Third, PTFlash currently does not support multi-phase equilibrium. Last but not least, neural networks are subject to the out-of-distribution generalization problem. If pressure and temperature are outside the predefined ranges used to train neural networks, predictive performance will deteriorate dramatically. Furthermore, when the mixture components change, we need to create new neural networks and train them from scratch.

4.8 CONCLUSION

In this chapter, we present a fast and parallel framework, PTFlash, for two-phase flash calculation based on PyTorch, which efficiently vectorizes algorithms and gains attractive speed-ups at large-scale calculations. Two neural networks are used to predict the stability of mixtures and to initialize the distribution coefficients more accurately than the Wilson approximation, which greatly accelerates PTFlash. In addition, PTFlash has much broader utility than related work mainly tailored to compositional reservoir simulation. We compare PTFlash with Carnot, an in-house thermodynamic library, and we investigate three case studies containing 2, 4, and 9 components with a maximum number of samples of 10, 5, and 1 million, respectively. The results show that PTFlash on GPU is 163.4, 106.3, and 50.5 times faster than Carnot at the maximum number of samples for the respective case studies.

In future work, we will (1) optimize PTFlash to reduce the consumption of GPU memory and extend our work to vectorize more advanced equations of state and support multi-phase equilibrium, (2) explore the potential of using neural networks to directly predict fugacity coefficients as a substitute for the numerical solution through equations of state, (3) validate PTFlash on more hardware suitable for parallel computing, e.g., TPU, and (4) apply our work to downstream applications, e.g., compositional reservoir simulation.

NNEOS : NEURAL NETWORK-BASED EOS TO CALCULATE FUGACITY COEFFICIENTS

In the previous chapter, we introduced PTFlash, which leverages PyTorch to vectorize the algorithms involved in isothermal two-phase flash calculation using the SRK EoS. This allows for parallel flash calculations on GPUs and thus leads to significant speed improvements during large-scale computation. However, as we noted in the discussion of the previous chapter, we require more complex EoS (such as the SAFT or CPA EoS) to handle mixtures that involve hydrogen bonding. Solving these EoS necessitates numerical algorithms that are both computationally demanding and difficult to effectively vectorize. Consequently, in this chapter, we will explore the possibility of utilizing neural networks to directly predict fugacity coefficients.

5.1 INTRODUCTION

EoS establish the connection between pressure P , temperature T , and volume V , enabling the calculation of various thermodynamic properties for pure compounds or mixtures given known state variables, such as P , T and mole numbers n . EoS are vital for a broad range of applications in chemical, petroleum, environmental, and biological engineering. Throughout this thesis, our focus is on the role of EoS in phase equilibrium calculations, specifically the use of EoS to compute fugacity coefficients. Due to the diversity of compounds present in mixtures (e.g., hydrocarbons, water, alcohols, polymers, electrolytes), researchers have made great efforts in the past decades to develop novel EoS or refine existing ones by introducing new parameters to improve the predictive accuracy of thermodynamic properties. Consequently, a series of increasingly complex EoS have been developed, ranging from the ideal gas law without adjustable parameters ($PV = nRT$) to parametrized EoS, such as the Van der Waals EoS, the cubic EoS, and advanced molecular EoS (such as the SAFT and CPA EoS).

In this chapter, we present a neural network-based EoS (NNEoS) that replaces the CPA EoS to predict logarithmic fugacity coefficients $\ln \varphi$. This enables us to carry out phase equilibrium calculations for mixtures that contain polar and associating compounds using PTFlash, the deep learning framework for two-phase flash calculation we introduced in the previous chapter. Specifically, we train neural networks on data generated by our in-house thermodynamic library, Carnot, which calculates $\ln \varphi$ based on the CPA EoS for a set of input data consisting of pressure P , temperature T , and composition z .

However, a significant challenge is that fugacity coefficients φ are discontinuous functions of P , T , and z . More specifically, there is a single discontinuity that splits φ into two continuous pieces. This discontinuity is unknown a priori and depends on the type of components being considered. According to the universal approximation theorem (Sec. 3.2), vanilla feedforward neural networks can only approximate continuous functions, indicating that the discontinuity will lead to irreducible errors if we utilize vanilla feedforward neural networks. To address this issue, we propose a Clustered Regression Network (CRNet), which combines unsupervised clustering and supervised regression. Specifically, CRNet consists of two expert networks and a gate network. During training, the gate network

can discover the discontinuity boundary in an unsupervised manner and divide the entire space into two continuous subspaces along the boundary, with each subspace assigned to an expert network. In this manner, CRNet can approximate piecewise continuous functions.

NNEoS adopts a two-stage approach to predict $\ln \varphi$ by using the compressibility factor Z as an intermediate target. This approach involves two neural networks: the first is a CRNet that predicts Z based on (P, T, z) , and the second is a simple feedforward neural network that predicts $\ln \varphi$ based on (Z, P, T, z) , where Z is provided by Carnot during training and by the first network during inference. Compared to directly predicting $\ln \varphi$ given (P, T, z) , this two-stage approach can reduce the learning difficulty of NNEoS and improve its performance. In fact, Z is also a discontinuous function, leading to the discontinuity of $\ln \varphi$. The calculation of $\ln \varphi$ depends on Z , implying that Z is a lower-level representation and is expected to be easier to learn. Therefore, using CRNet to learn Z can facilitate the discovery of discontinuity.

We also propose HybridEoS, which combines NNEoS and Carnot to achieve a balance between speed and precision. While NNEoS has a speed advantage due to parallel computing on GPUs, its prediction errors may impede the convergence of flash calculations. Carnot provides accurate fugacity coefficients but can only run on CPUs. HybridEoS combines the strengths of both NNEoS and Carnot to enhance overall efficiency and precision. NNEoS is utilized in most cases, while Carnot is specifically employed to provide $\ln \varphi$ and their derivatives for the trust-region methods of stability analysis and phase split calculations.

The rest of this chapter is organized as follows. In Sec. 5.2, we review prior work on substituting machine learning models for numerical EoS and then present Regression Clustering and Mixtures of Experts. In Sec. 5.3, we describe the case study used in this chapter. In Sec. 5.4, we analyze the discontinuity of $\ln \varphi$ and explain its cause. In Sec. 5.5, we introduce CRNet and demonstrate its application to a regression toy problem. In Sec. 5.6, we present the two-stage approach for NNEoS. In Sec. 5.7, we compare the performance of PTFlash using NNEoS and HybridEoS with Carnot. Finally, we conclude our work in this chapter and suggest future research directions in Sec. 5.8.

5.2 RELATED WORK

5.2.1 Replace numerical EoS with machine learning models

In [121], the authors utilized support vector regression (SVR) [6] to predict the pressure P of three pure fluids (H_2O , H_2 , and CO_2) based on temperature (T) and volume (V), as well as predict P of the mixture of these fluids based on T , V , and mole numbers \mathbf{n} . More broadly, their goal is to replace $P_{EoS}(T, V, \mathbf{n})$ with $P_{ML}(T, V, \mathbf{n})$, where ML denotes a machine learning model and SVR was chosen. The authors trained SVR on high-quality data generated by molecular dynamics simulations using the LAMMPS package [156]. Their results demonstrate that SVR can effectively predict the pressure of these pure fluids and their mixtures with very low errors.

[239] proposed the use of neural networks and Gaussian Process Regression (GPR) [175] to predict the properties of pure fluids, such as critical pressure and temperature, vapor pressure, and density. The authors trained neural networks and GPR on a well-curated database of properties for 540 Mie fluids, using as input five molecular descriptors of the SAFT-VR Mie EoS [103]. The results show that both neural networks and GPR achieve low prediction errors on the test set, suggesting their potential to serve as surrogates for analytical EoS in predicting the properties of pure fluids.

[121] used neural networks to predict the Helmholtz free energy A of pure fluids based on T , V and the total number of moles n . The authors claim that their method is able to yield more thermodynamically consistent machine-learned EoS because other thermodynamic properties are derived by computing the partial derivatives of A with respect to T , V and n , as shown in Tab. 2.1, which can be easily accomplished through the automatic differentiation of deep learning frameworks. An innovative aspect of this work is the use of the partial derivatives of A to define the cost function for the training of neural networks, as follows:

$$\mathcal{L} = \left[- \left(\frac{\partial \hat{A}}{\partial V} \right)_{n,T} - P \right]^2 + \left[\left(\frac{\partial \hat{A}}{\partial n} \right)_{V,T} - \mu \right]^2 + \left[\hat{A} - T \left(\frac{\partial \hat{A}}{\partial T} \right)_{n,V} - U \right]^2 \quad (5.1)$$

where \hat{A} is the prediction of neural networks. The authors validate their method on the Van der Waals EoS and the Lennard-Jones EoS [196] for Lennard-Jones fluids.

The works mentioned above demonstrate the potential of machine learning models in predicting properties of pure liquids and mixtures, however, they are not applicable to our work focusing on flash calculation for multi-component mixtures. If we apply the first work to flash calculation, we need to compute $\ln \varphi$ based on $P_{ML}(T, V, \mathbf{n})$, which faces two intractable difficulties. The first is to solve $P = P_{ML}(T, V, \mathbf{n})$ and the second is to obtain the residual Helmholtz energy A^r , which involves integrating $\int_V^\infty P_{ML}(T, V, \mathbf{n})$. In our work, we use neural networks to directly predict $\ln \varphi$, which effectively circumvents these difficulties.

If we use the approach of the third work, we need to first learn the residual Gibbs energy G^r using neural networks and then calculate $\ln \varphi_i = \partial G^r / \partial n_i$ (see Eq. (2.12)) via backpropagation. This is computationally expensive and time-consuming, especially when the second-order derivatives of G^r are required via backpropagation twice, leading to significant memory consumption. Despite these limitations, the third work's concept of designing thermodynamically consistent machine-learned EoS merits consideration and could be a promising direction for future research.

5.2.2 Regression Clustering and Mixture of Experts

Regression Clustering (RC) [228] is a technique that combines clustering and regression, which applies K base regression learners to data simultaneously and guides the clustering of the data into K subsets by minimizing the global regression error, with each cluster assigned to a regression learner. A common application of RC is to combine a set of linear regression models to address non-linear regression problems, known as clustered or cluster-wise linear regression [4, 36, 149, 189]. Our proposed CRNet can be viewed as an enhanced RC model, for which the base regression learner is a neural network that enables the modeling of more complex patterns.

Mixture of Experts (MoE) [227] aggregates a number of experts by computing the weighted sum of their predictions as the final output. The aggregation weights are determined by a gate network, which divides the input space into subspaces and assigns each subspace to an expert. This allows MoE to effectively handle data with heterogeneous patterns. Our proposed CRNet can be seen as a simplified MoE with only two experts, which is tailored to our problem of identifying two subspaces.

5.3 CASE STUDY

Throughout this chapter, we use a mixture of water and methane as the case study to facilitate our analysis and visualization. The CPA EoS is used to account for the effect of hydrogen bonding in the presence of water. The ranges of P and T are 1 - 40MPa and 200 - 600K, respectively, and the entire compositional space is considered, i.e., $0 < z_i < 1$. To sample input data, we utilize the LHS technique to draw P , T , and the molar fraction of water z_1 within the expected ranges, and then we calculate the molar fraction of methane as $z_2 = 1 - z_1$. We depict the phase diagram of this mixture to gain more insight into its characteristics, as shown in Fig. 5.1. It can be observed that when z_1 is close to 1, i.e., a very small amount of methane is present, the mixture is aqueous. When T exceeds 300K and z_2 is relatively high, the mixture is more likely to be vapor. In most other cases, two phases coexist.

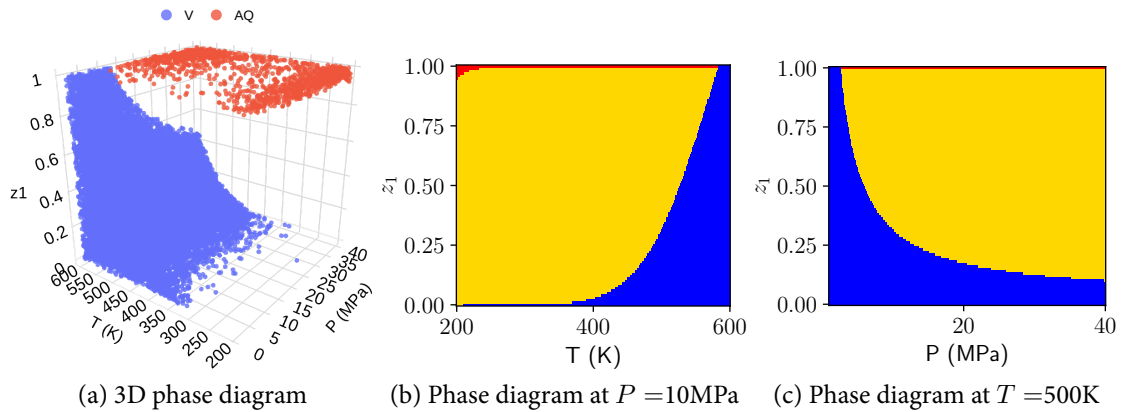


Figure 5.1: The red, blue and yellow represent the aqueous phase, the vapor phase and the coexistence of two phases, respectively. In Figure (a), we hide the predominant coexistence of two phases for a better view of the aqueous and vapor phases. Figures (b) and (c) show the phase diagrams at $P = 10\text{MPa}$ and $T = 500\text{K}$, respectively.

5.4 ANALYSIS OF THE DISCONTINUITY OF FUGACITY COEFFICIENTS

Fig. 5.2 provides the visualization of $\ln \varphi$ at different pressures of 1, 10.5 and 20.5MPa by showing the slice planes of $\ln \varphi$ with respect to T and the molar fraction of water z_1 . At $P = 1\text{MPa}$, a distinct boundary separating two groups of colors can be observed, indicating the presence of discontinuity. At $P = 10.5\text{MPa}$, the discontinuity is incomplete and terminates midway, as illustrated in both Fig. 5.2b and Fig. 5.2d. At $P = 20.5\text{MPa}$, the colors change continuously, indicating the disappearance of discontinuity.

Next, we explain the cause of the discontinuity of $\ln \varphi$. We know that the calculation of $\ln \varphi$ requires Z by solving the following equation:

$$f(Z) = P - P_{\text{EoS}}(T, Z, \mathbf{n}) = 0 \quad (5.2)$$

The above equation may have one root or multiple roots. In the latter case, the smallest root is assigned to the liquid phase and the largest one is assigned to the vapor phase. The root associated with the smallest Gibbs energy G is then used to calculate $\ln \varphi$, since the smallest G corresponds to the most stable system according to the principle of minimum energy. The transition from one root to multiple roots will cause discontinuities in the solution of $f(Z)$, which in turn results in the discontinuity of $\ln \varphi$.

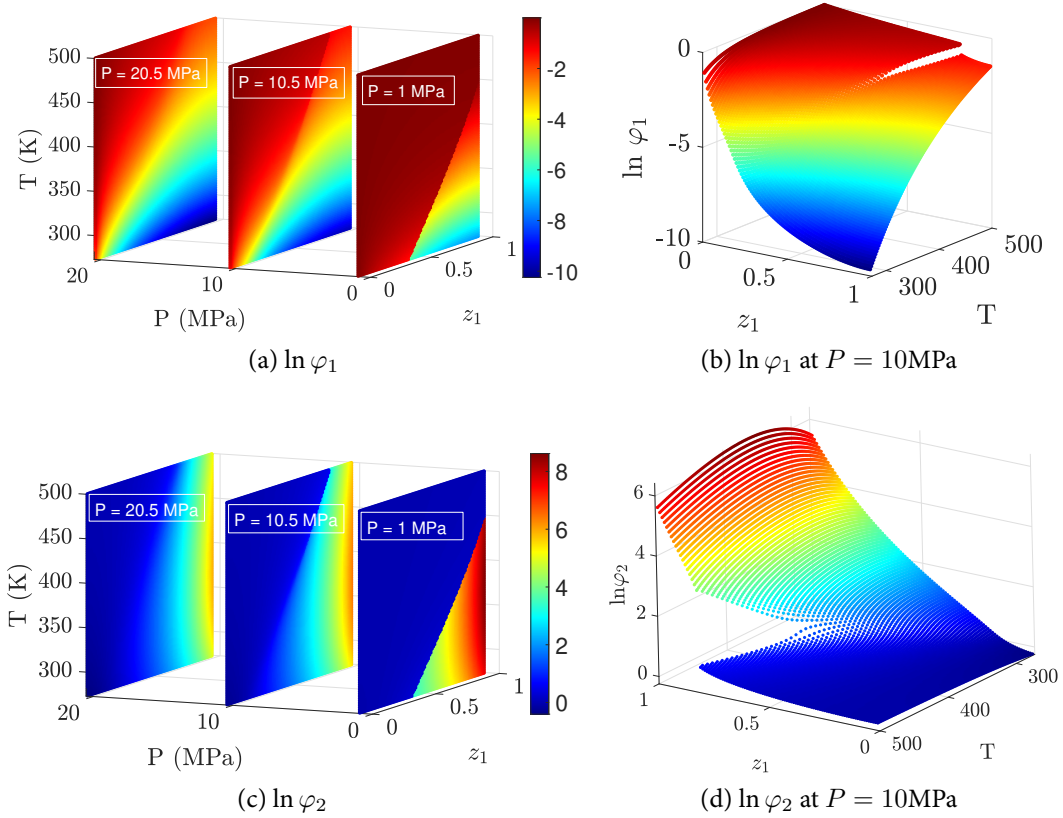


Figure 5.2: Fugacity coefficients for the mixture of water and methane at different pressures, where z_1 is the molar fraction of water.

5.5 CLUSTERED REGRESSION NETWORK

To learn discontinuous functions, we propose a Clustered Regression Network (CRNet), as illustrated in Fig. 5.3. CRNet consists of two expert networks (E_1 and E_2) and a gate network G_{NN} , which share the same trunk network T_{NN} to learn high-level representations t from the input x . The gate network G_{NN} provides the probability of selecting the first expert, denoted by p . The final output of CRNet is a weighted sum of the two experts:

$$y = p e_1 + (1 - p) e_2 = G_{NN}(t) E_1(t) + (1 - G_{NN}(t)) E_2(t) \text{ with } t = T_{NN}(x) \quad (5.3)$$

In CRNet, the output layer of the gate network employs the sigmoid activation function $\sigma(\cdot)$, that is, $p = \sigma(g) = 1/(1 + \exp(-g))$, where g is the pre-activation value. CRNet is designed to partition the entire space into two distinct parts, with each expert specializing in one part exclusively, that is, $p = 0$ or 1 . To achieve this, we introduce a sparse loss function as follows:

$$\begin{aligned} \mathcal{L}_{sp} &= \mathbb{E}_{(x,y) \sim \mathcal{D}_{tr}} \left[\frac{d\sigma(g)}{dg} \right] \\ &= \mathbb{E}_{(x,y) \sim \mathcal{D}_{tr}} [\sigma(g) (1 - \sigma(g))] \\ &= \mathbb{E}_{(x,y) \sim \mathcal{D}_{tr}} [p (1 - p)] \end{aligned} \quad (5.4)$$

The above sparse loss is actually the gradient of $\sigma(g)$, as shown in Fig. 5.4. Minimizing \mathcal{L}_{sp} drives g towards negative or positive infinity, causing p to converge to 0 or 1. In practice,

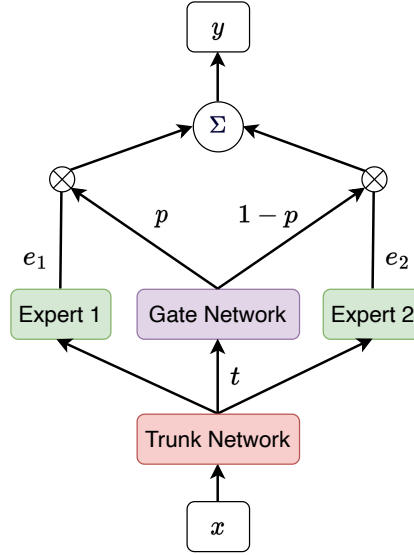


Figure 5.3: Architecture of clustered regression networks

\mathcal{L}_{sp} is multiplied by a relaxation coefficient γ , which linearly increases from 0 to 1 in the first half of training and remains at 1 in the second half. This enables experts to be fully trained and gradually become specialized in the early stages of training.

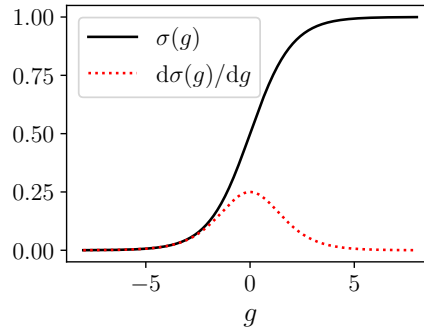


Figure 5.4: Sigmoid activation function and its gradient

CRNet may suffer from an unbalanced expert load, for which only one expert is used and the other is disregarded, resulting in p being constant at 0 or 1. To address this issue, we first introduce a measure of expert load for a training mini-batch X of size N_X . Specifically, the load of the first expert L_1 is defined as the sum of the weights allocated to it, i.e., $L_1 = \sum_{i=1}^{N_X} p_i$, and that of the second expert L_2 is defined as $L_2 = N_X - L_1$. To balance the expert load, we minimize the Kullback-Leibler (KL) divergence between the uniform distribution $\mathcal{U} = (0.5, 0.5)$ and $L = (L_1, L_2)/N_X = (l, 1-l)$ with $l = L_1/N_X$, as follows:

$$\mathcal{L}_{kl} = D_{KL}(\mathcal{U}||L) = \frac{1}{2} \log \frac{1}{2l} + \frac{1}{2} \log \frac{1}{2(1-l)} \quad (5.5)$$

$$= -\frac{1}{2} \log [4l(1-l)] \quad (5.6)$$

This KL loss \mathcal{L}_{kl} ensures that the expert load will be approximately evenly distributed between the two experts for each mini-batch. This is a strong inductive bias, as it assumes that the data is approximately evenly split on each side of the discontinuity. This assumption

holds for our test case, but further parameterization of \mathcal{L}_{kl} may be necessary for different test cases.

In addition to \mathcal{L}_{sp} and \mathcal{L}_{kl} , the supervised regression loss on targets is:

$$\mathcal{L}_y = \mathbb{E}_{(x,y) \sim \mathcal{D}_{tr}} [\text{MAE}(y, \hat{y})] \quad (5.7)$$

where MAE denotes the mean absolute error and \hat{y} is the prediction of CRNet, as calculated by Eq. (5.3). The final cost function to train CRNet is:

$$\mathcal{L} = \lambda_{sp} \mathcal{L}_{sp} + \lambda_{kl} \mathcal{L}_{kl} + \lambda_y \mathcal{L}_y \quad (5.8)$$

where λ are the trade-off parameters to balance different losses.

To showcase the effectiveness of CRNet, we apply it to a toy regression problem as depicted in Fig. 5.5. In this toy problem, T_{NN} , G_{NN} , E_1 , and E_2 of CRNet all have two hidden layers with 32 units and use the SiLU activation function. The output layers of G_{NN} , E_1 , and E_2 have one unit. We set $\lambda_{sp} = \lambda_{kl} = 0.1$ and $\lambda_y = 1$. It can be seen that CRNet successfully identifies the middle break point and divides the data into two parts, with each expert specializing in one part.

For comparison, we also apply four clustering algorithms, including K-means clustering [118], spectral clustering [209], Gaussian mixture [167], and agglomerative clustering [141], to the same data. However, as shown in Fig. 5.6, these clustering algorithms fail to divide the data in the same way as CRNet. In fact, CRNet implicitly leverages an inductive bias that x and y are not independent of each other and lie on a low-dimensional manifold, which is not taken into account by vanilla clustering algorithms.

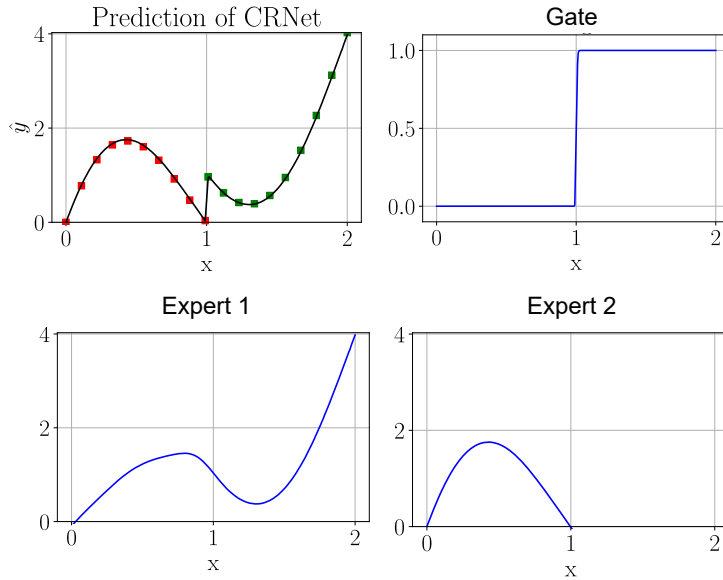


Figure 5.5: Application of CRNet to a toy regression problem. The data points in the upper-left subplot are marked in red if the output of the gate network $p < 0.5$ and green if $p > 0.5$.

5.6 NEURAL NETWORK-BASED EQUATION OF STATE

We propose NNEoS, which replaces numerical EoS with neural networks to compute $\ln \varphi$ for flash calculations. NNEoS uses the compressibility factor Z as an intermediate target

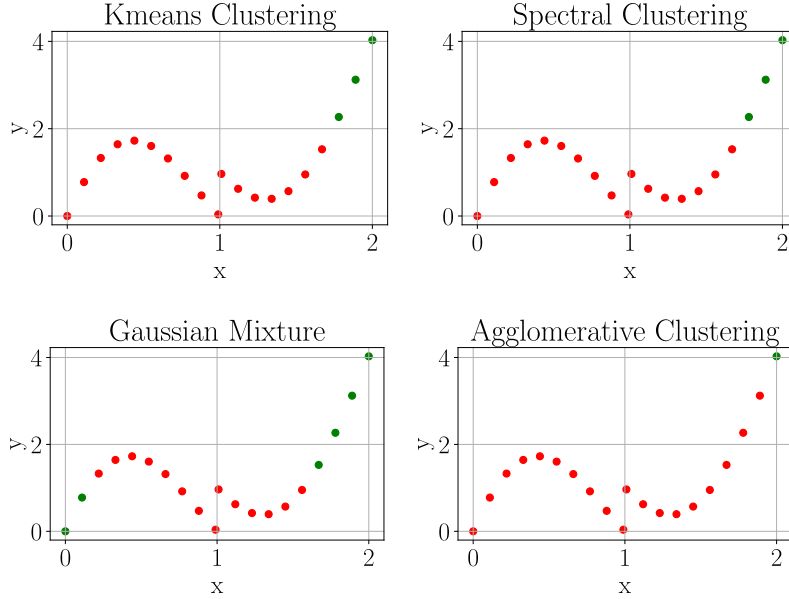


Figure 5.6: Four clustering algorithms for the toy regression problem

and comprises two neural networks: Net_Z that is a CRNet to predict Z based on (P, T, z) and Net_φ that is a simple feedforward neural network to predict $\ln \varphi$ based on (Z, P, T, z) , where Z is provided by Net_Z during inference.

The architecture of NNEoS is illustrated in Fig. 5.7. We have already discussed the scaling layer, wide shortcut, and concat layer in Sec. 4.6.2. Additionally, we introduce two other improvements to NNEoS:

- We add $\ln P$ and $1/T$ to the inputs of Net_Z and Net_φ , which is shown to improve their performance. Although neural networks are capable of extracting and learning useful features from raw data, hand-crafted feature engineering is still beneficial in some cases.
- In Fig. 5.7b, we introduce two intermediate variables, namely $\tilde{z}_1 = z_1/(z_1 + z_2)$ and $\tilde{z}_2 = z_2/(z_1 + z_2)$. While \tilde{z}_1 and \tilde{z}_2 may appear redundant, they ensure that the predicted $\ln \hat{\varphi}$ are intensive properties that are independent of the amount of substances in the system. It is worth noting that the composition $z = (z_1, z_2)$ should be interpreted as the mole numbers that happen to sum up to 1. The variables \tilde{z}_1 and \tilde{z}_2 make $\ln \hat{\varphi}$ scale-invariant to z . In other words, multiplying z by a factor will not change the value of $\ln \hat{\varphi}$. This allows $\partial \hat{\varphi} / \partial z$ to represent the partial derivatives of $\ln \hat{\varphi}$ with respect to the mole numbers, which are necessary for the trust-region method of stability analysis (Sec. 2.4.2) and phase split calculations (Sec. 2.4.3).

A dataset of one million samples is processed by Carnot to calculate Z and $\ln \varphi$ using the CPA EoS. The resulting data is then split into the training (70%), validation (15%), and test (15%) sets. We use PyTorch to implement and train neural networks under the double-precision floating-point format.

Net_Z is a CRNet and its training cost function is given by Eq. (5.8), where we set $\lambda_{sp} = 0.1$, $\lambda_{kl} = 0.15$ and $\lambda_y = 1$. We train Net_Z using Adam with a batch size of 512 for 200 epochs. A cyclic learning schedule is employed, as depicted in Fig. 5.8b, with the maximum learning rate determined through a learning rate range test, as shown in Fig. 5.8a. The training

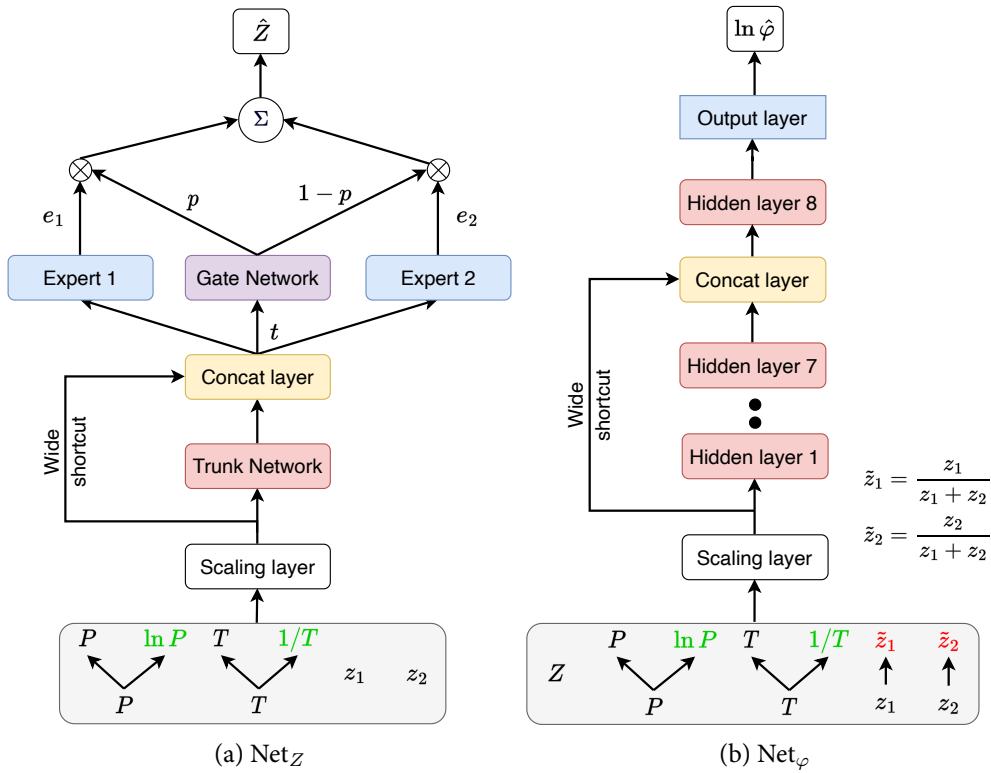


Figure 5.7: This is the architecture of NNEoS consisting of Net_Z and Net_φ . In Figure (a), the trunk network has 5 hidden layers, the gate network has 2 hidden layers, and two experts have 2 hidden layers. All hidden layers of Net_Z have 32 units and use the SiLU activation function, and the output layers of the gate network and experts have one unit. In Figure (b), Net_φ has 8 hidden layers with 64 units and the SiLU activation function, and its output layer has 2 units. We introduce \hat{z}_1 and \hat{z}_2 to ensure that $\ln \hat{\varphi}$ are intensive properties and $\partial \ln \hat{\varphi} / \partial z$ represent the partial derivatives of $\ln \hat{\varphi}$ with respect to the mole numbers instead of the composition. In addition, the inclusion of $\ln P$ and $1/T$ can improve the performance of both Net_Z and Net_φ .

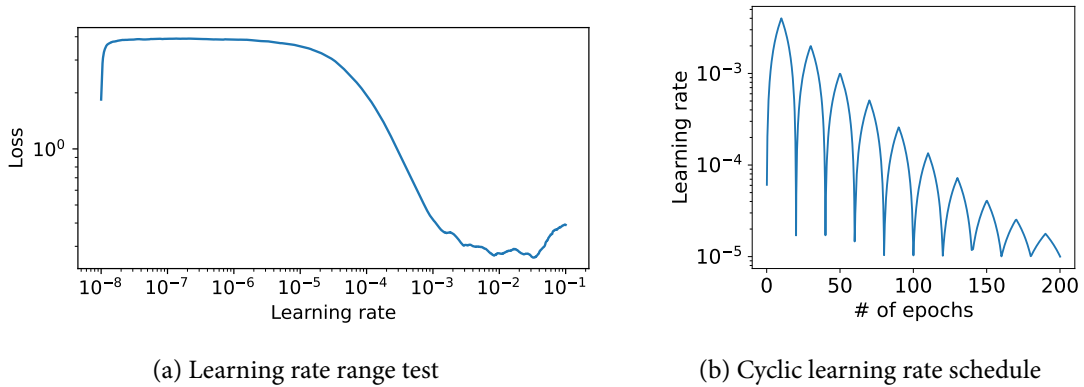


Figure 5.8: Figure (a) is a learning rate range test used to determine an appropriate learning rate by observing the change in the training loss as the learning rate increases from small to large values. Figure (b) shows how the learning rate varies for the cyclic schedule.

losses are illustrated in Fig. 5.9, and Fig. 5.10 shows the outputs of key elements of Net_Z at $P = 1\text{MPa}$. We observe that Net_Z accurately identifies the discontinuity boundary.

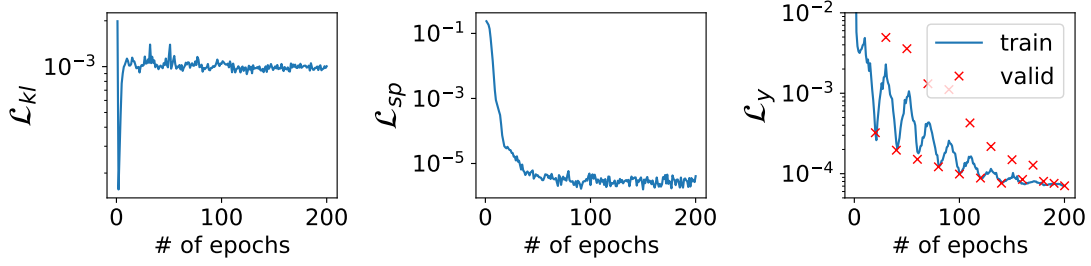


Figure 5.9: Training losses of Net_Z

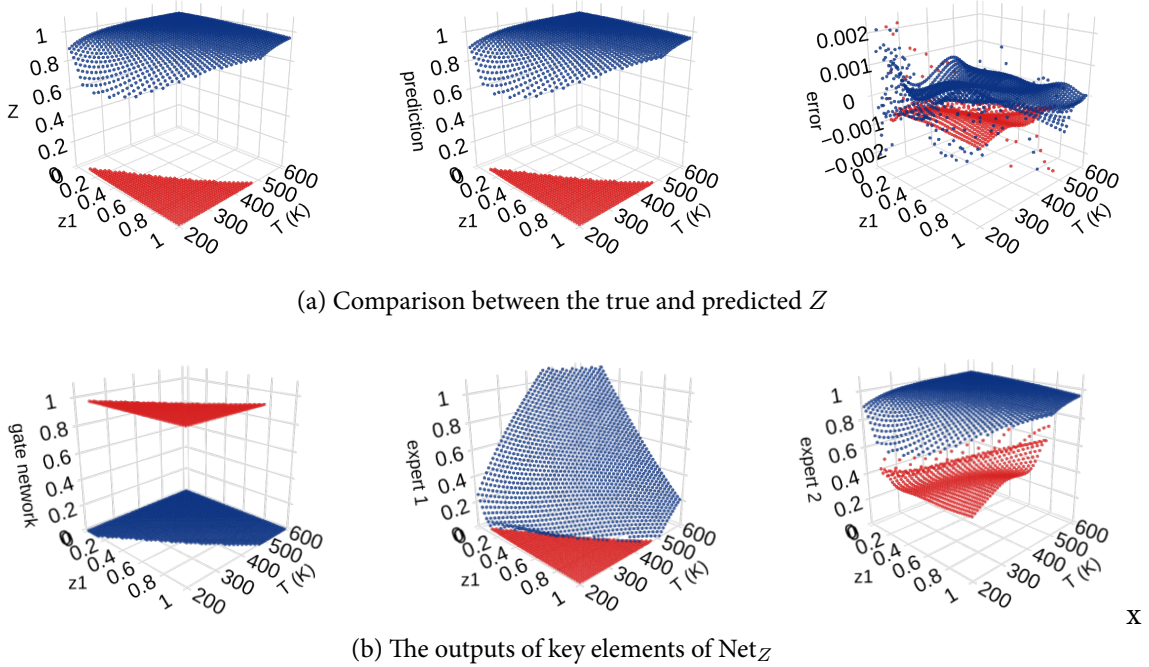


Figure 5.10: The prediction of Net_Z and the outputs of its key parts at $P = 1\text{MPa}$

For the training of Net_φ , we minimize the following loss:

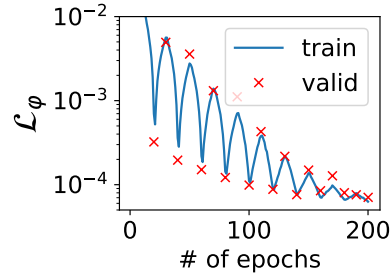
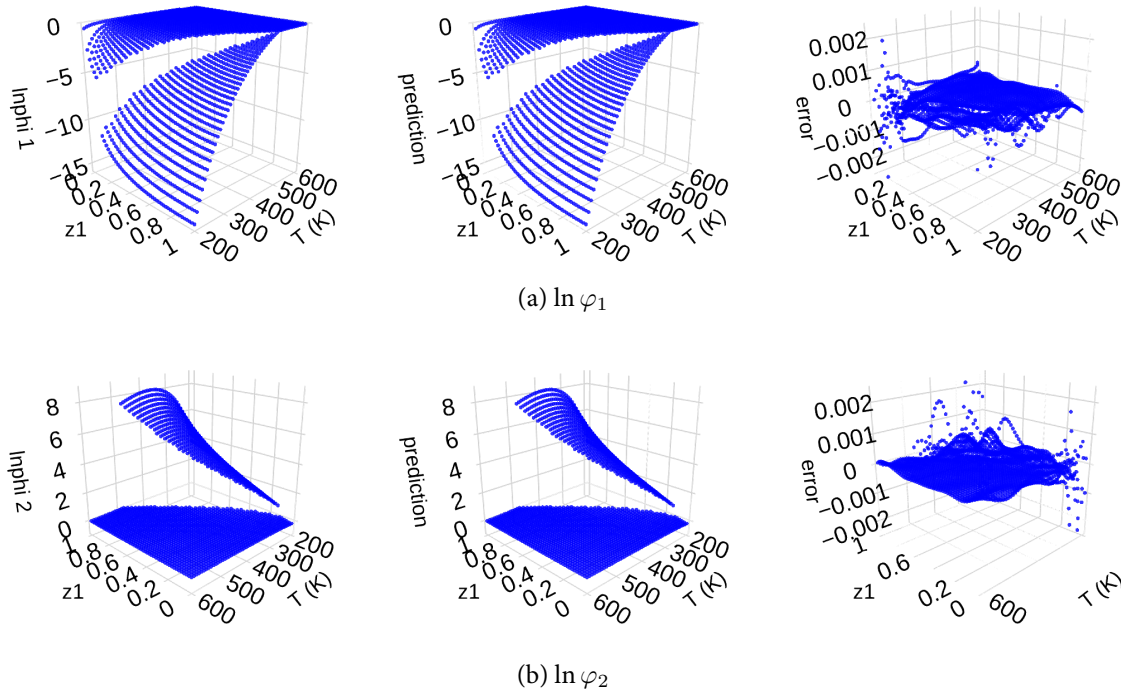
$$\mathcal{L}_\varphi = \mathbb{E}_{(x,\varphi) \sim \mathcal{D}_{tr}} [\text{MAE}(\ln \varphi, \ln \hat{\varphi})] \quad (5.9)$$

We train Net_φ using Adam with a batch size of 512 for 200 epochs, and we employ the cyclic learning schedule. Fig. 5.11 shows the training loss \mathcal{L}_φ . Fig. 5.12 compares the true and predicted $\ln \varphi$ at $P = 1\text{MPa}$.

The performance of NNEoS on the test sets are $\text{MAE} = 6.65\text{e-}5$ in terms of Z for Net_Z , $\text{MAE} = 7.53\text{e-}5$ in terms of $\ln \varphi$ for Net_φ using the true Z as input, and $\text{MAE} = 9.96\text{e-}5$ in terms of $\ln \varphi$ for Net_φ using the predicted \hat{Z} of Net_Z as input.

5.7 RESULTS

In this section, we will integrate NNEoS into PTFlash, the fast and parallel deep learning framework for two-phase flash calculation introduced in the previous chapter. Additionally,

Figure 5.11: Training loss of Net_φ Figure 5.12: Comparison between the true and predicted $\ln \varphi$ at $P = 1\text{MPa}$

we propose HybridEoS to combine Carnot and NNEoS. We then compare Carnot with PTFash using NNEoS and HybridEoS. The hardware is Intel 11700F CPU and NVIDIA RTX 3080 GPU featuring 8704 CUDA cores and 10G memory. Only one core of CPU is used, and its frequency can be stabilized at 4.5GHz.

5.7.1 Comparison between Carnot and NNEoS

We compare the execution time of Carnot and NNEoS for the calculation of $\ln \varphi$ and their derivatives $\partial \ln \varphi$, as illustrated in Fig. 5.13. Additionally computing $\partial \ln \varphi$ takes approximately twice as much time as computing $\ln \varphi$ alone. For Carnot, computing $\partial \ln \varphi$ is nearly as computationally intensive as computing $\ln \varphi$. For NNEoS, calculating $\partial \ln \varphi$ requires an additional backpropagation (Sec. 3.3.2). At the maximum number of samples $n = 5 \times 10^6$, NNEoS on CPU and GPU are roughly 6 and 80 times faster than Carnot, respectively, demonstrating the significant acceleration of NNEoS in the calculation of $\ln \varphi$. The time of Carnot and NNEoS on CPU is roughly proportional to n , while the time of

NNEoS on GPU hardly changes for $n < 1000$ and then also becomes proportional to n for $n > 1000$, suggesting that the throughput of GPU has been reached.

It is worth noting that the speedup of NNEoS over Carnot shown in Fig. 5.13 only demonstrates the theoretical acceleration, which may not necessarily translate to practical acceleration when NNEoS is integrated into PTFflash for complete flash calculation, because the prediction errors of NNEoS may lead to an increase in the number of iterations or even cause flash calculation to diverge, resulting in a less favorable acceleration.

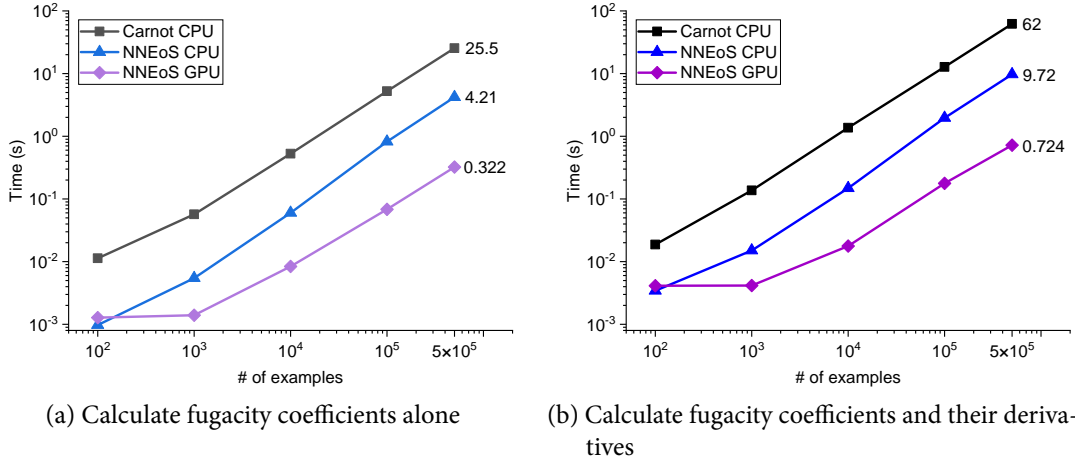


Figure 5.13: Comparison between the execution time of Carnot and NNEoS for calculating fugacity coefficients

5.7.2 Comparison between Carnot and PTFflash using NNEoS

To integrate NNEoS into PTFflash, we have to disable TorchScript that was previously used to optimize the code of PTFflash in Sec. 4.5. This is because TorchScript is currently incompatible with functorch [80], a JAX-like PyTorch-based library, which is needed for efficient per-sample-gradient computation to obtain the derivatives of NNEoS for a batch of input data simultaneously.

It is of great importance to investigate the potential buildup of prediction errors of NNEoS during the iterative process of flash calculation, as these errors could lead to poor results or even cause flash calculation to diverge. To verify this, we compare Carnot with PTFflash using NNEoS on five representative samples, as shown in Tab. 5.1. We recorded the number of iterations required by each subroutine as well. For the first two relatively simple examples, PTFflash and Carnot yield roughly identical results. The third and fourth examples are more challenging, as they are on the verge of transitioning into pure aqueous and vapor phases, respectively. PTFflash performs well on the third example, but wrongly classifies the fourth as the aqueous phase. The fifth example is near the critical point and therefore demands greater precision. Unfortunately, the vapor fraction obtained by PTFflash largely differs from that of Carnot. Furthermore, we observe that the number of iterations required by PTFflash is greater than that of Carnot, indicating that while NNEoS can provide satisfactory results in most cases, its prediction errors lead to an increase in the number of iterations and thereby diminish the speed advantage of NNEoS in computing $\ln \varphi$.

In order to fully evaluate PTFflash using NNEoS, we apply it to one million samples. We set the maximum number of iterations of the successive substitution to 18. The perfor-

Table 5.1: Comparison between the results of Carnot and PTFlash using NNEoS

P (Pa)			T (K)			z_1					
21410710			455.65			0.3127993					
13825639			339.62			0.8656848					
29614171			403.09			0.0130436					
33138567			223.31			0.9777717					
8643204			572.21			0.9932372					
Carnot			PTFlash using NNEoS								
x_1^1		y_1^1	θ_V^1			\hat{x}_1		\hat{y}_1	$\hat{\theta}_V$		
0.9964627		0.0650217	0.7339847			0.9964629		0.0650324	0.7339932		
0.9982982		0.0025541	0.1331803			0.9982980		0.0025537	0.1331800		
0.9968380		0.0130401	0.9999965			0.9968382		0.0130421	0.9999985		
0.9777756		0.0000039	0.0000040			0.9777717			0		
0.9999311		0.9923353	0.8812626			0.9999389		0.9932190	0.9973019		
Number of iterations of each subroutine ²											
stability analysis			phase split			stability analysis			phase split		
vapor		liquid	calculations			vapor		liquid	calculations		
			7						10		
2		6	5			2		18	6		
4		4	4			3		6	4		
			3			2		4			
			2			4		2	3		

¹ x_1 and y_1 represent the molar fraction of water in the aqueous phase and that in the vapor phase, respectively, and θ_V denotes the vapor fraction.

² Only the successive substitution method is used for each subroutine.

mance profiler is shown in Tab. 5.2. It can be seen that the trust-region method of stability analysis only achieves convergence percentages of 25.87% and 34.76% for the vapor-like and liquid-like estimates after 40 iterations, respectively. This may be due to the proximity of unconverged samples to critical points, which necessitates a higher level of predictive performance from NNEoS.

5.7.3 HybridEoS to combine Carnot and NNEoS

We present HybridEoS to balance speed and precision, which combines Carnot and NNEoS in such a way that we use NNEoS to compute $\ln \varphi$ for the successive substitution and Carnot to compute $\ln \varphi$ and $\partial \ln \varphi$ for the trust-region method. To efficiently call C++-based Carnot within Python-based PTFlash, we utilize the Simplified Wrapper and Interface Generator (SWIG) [8] to develop a Python interface for Carnot. Although Carnot can only process samples one at a time on CPU, it may not be a major issue since we set the maximum number of iterations of the successive substitution to 18, allowing the majority of samples (over 90%) to converge and hence reducing the strain on the trust-region method. The performance profiler of PTFlash using HybridEoS on GPU is shown in Tab. 5.3. We can see

	ss of phase split calculations	Stability analysis				Phase split calculations	
		vapor-like estimate		liquid-like estimate		ss	tr
	ss	tr	ss	tr	ss		
# of samples	10^6	470954	1380	470954	30383	837077	1518
Convergence	52.9%	99.71%	25.87%	93.55%	34.76%	99.82%	99.87%
Max number of iterations	3	18	40	18	40	18	40
Total time	3.2515s	1.7548s	0.3903s	2.8651s	2.0352s	10.8040s	1.4127s
		7.0454s				12.2167s	
ss: successive substitution				tr: trust-region method			

Table 5.2: Performance profiler of PTFlash using NNEoS on GPU for the mixture of water and methane

that all subroutines converge successfully. However, the execution time of the trust-region method remains relatively large due to the intensive communication between CPU and GPU, because Carnot can only process data on CPU memory while PTFlash manipulates data on GPU memory.

	ss of phase split calculations	Stability analysis				Phase split calculations	
		vapor-like estimate		liquid-like estimate		ss	tr
	ss	tr	ss	tr	ss		
# of samples	10^6	470954	1380	470954	30383	837075	1518
Convergence	52.9%	99.71%	100%	93.55%	100%	99.82%	100%
Max number of iterations	3	18	7	18	5	18	17
Total time	3.2516s	1.752s	0.4009s	2.8552s	7.484s	10.6887s	1.4319s
		12.4921s				12.1217s	
ss: successive substitution				tr: trust-region method			

Table 5.3: Performance profiler of PTFlash using HybridEoS on GPU for the mixture of water and methane

We compare the execution time of Carnot and PTFlash using HybridEoS for flash calculation with respect to the number of samples n , as shown in Fig. 5.14. At $n = 10^6$, PTFlash on CPU and GPU are 1.7 and 16.9 times faster than Carnot on CPU, respectively.

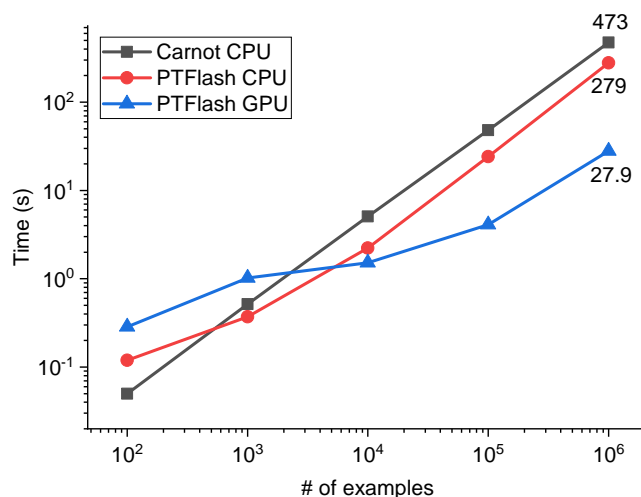


Figure 5.14: Comparison between the execution time of Carnot and PTFlash using HybridEoS

5.7.4 Discussion

Our findings demonstrate the potential of NNEoS as a substitute for the CPA EoS to quickly and accurately provide fugacity coefficients for flash calculation. NNEoS enables us to perform parallel flash calculations on GPUs using PTFlash without vectorizing the CPA EoS, resulting in substantial speedups. Currently, the application of machine learning in the field of EoS is in its infancy, with related work focusing on relatively simple applications such as predicting the properties of pure fluids. Our work involves multi-component mixtures and is more challenging.

Clearly, NNEoS has its limitations. Despite our efforts to address the discontinuity problem with CRNet and the two-stage approach, and to improve the performance of NNEoS through feature engineering and practical training techniques, PTFlash using NNEoS still needs more iterations for convergence and even yields erroneous results compared to Carnot, especially near critical points. While our proposed HybridEoS partially alleviates this problem, it is still an expedient and suboptimal solution as it disrupts hardware consistency during calculations and thus increases the overhead of CPU-GPU communication and data copy. In addition, NNEoS also faces some common challenges associated with neural networks, such as the out-of-distribution generalization problem, i.e., the performance of NNEoS deteriorates outside the predefined ranges of pressure and temperature, and the transfer learning problem, i.e., how to apply NNEoS to different components without retraining it from scratch.

The slow convergence of PTFlash using NNEoS compared to Carnot is not essentially due to the prediction errors of NNEoS but rather to the fact that NNEoS does not satisfy the thermodynamic consistency presented in Sec. 2.3. NNEoS merely imitates the CPA EoS through supervised learning, but is not an eligible EoS due to the violation of thermodynamic consistency. Therefore, we should prioritize satisfying thermodynamic consistency over simply reducing prediction errors. One approach to achieving this is to employ physics-informed neural networks [161, 162], which encode Eqs. (2.15), (2.16) and (2.17b) involved in the thermodynamic consistency into regularization terms that are added to the cost function. However, this approach is still a form of “supervised learning” in that the thermodynamic consistency is learned rather than inherent. It remains

an open question in deep learning how to efficiently incorporate prior knowledge (e.g., thermodynamic consistency) into neural networks to improve generalization performance.

5.8 CONCLUSION

In this chapter, we present CRNet to address the discontinuity problem of fugacity coefficients φ , and then we develop NNEoS to calculate $\ln \varphi$ using the two-stage approach with the compressibility factor Z as an intermediate target. NNEoS enables us to perform flash calculation for the mixture of water and methane without vectorizing the CPA EoS, and NNEoS achieves a roughly 80x speedup compared to Carnot on large-scale computation of $\ln \varphi$ due to parallel computing on GPUs while maintaining a high level of precision. PTFlash using NNEoS proves to be efficient and accurate in regions far from critical points and phase transition boundaries. To balance speed and precision, we also propose HybridEoS to combine the strengths of NNEoS and Carnot. PTFlash using HybridEoS on GPU is roughly 17 times faster than Carnot when processing one million samples. Our results demonstrate the potential of neural networks to replace numerical EoS in computing $\ln \varphi$.

In the future, we will focus on developing thermodynamically consistent NNEoS and improving the generalization of NNEoS to different components so that we do not need to retrain NNEoS from scratch when the components of mixtures change.

Part III

DOMAIN GENERALIZATION

HMOE: HYPERNETWORK-BASED MIXTURE OF EXPERTS FOR DOMAIN GENERALIZATION

In the previous chapter, we proposed CRNet to solve the regression problem of learning discontinuous functions by partitioning the overall space into two continuous subspaces and assigning them to two experts. CRNet can be seen as a simplified version of Mixture of Experts (MoE). In this chapter, we will extend the concept of CRNet to the "discontinuity" problem in the field of image classification, i.e., domain generalization. Due to domain shift, machine learning systems typically fail to generalize well to domains different from those of training data, which is what domain generalization (DG) aims to address. Although various DG methods have been developed, most of them lack interpretability and require domain labels that are not available in many real-world scenarios. This chapter presents a novel DG method, called HMOE: Hypernetwork-based Mixture of Experts (MoE), which does not rely on domain labels and is more interpretable. MoE proves effective in identifying heterogeneous patterns in data. For the DG problem, heterogeneity arises exactly from domain shift. HMOE uses hypernetworks taking vectors as input to generate experts' weights, which allows experts to share useful meta-knowledge and enables exploring experts' similarities in a low-dimensional vector space. We compare HMOE with other DG algorithms under a fair and unified benchmark – DomainBed. Our extensive experiments show that HMOE can divide mixed-domain data into distinct clusters that are surprisingly more consistent with human intuition than original domain labels. Compared to other DG methods, HMOE shows competitive performance and achieves SOTA results in some cases.

6.1 INTRODUCTION

Domain generalization (DG) aims to train models on known domains that can generalize well to unseen domains, which is of crucial importance for deploying models in safety-critical real-world applications. Over the past decade, a variety of DG algorithms have been proposed [68, 213, 236], focusing primarily on developing DG-specific data augmentation techniques and learning domain-invariant representations to build generalizable predictors. However, many high-performing DG algorithms rely on domain labels to explicitly reduce domain discrepancy, severely limiting their applicability in real-world scenarios where domain annotation may be prohibitively expensive. Additionally, current DG algorithms lack interpretability and fail to provide insight into the causes of success or failure in generalizing to new domains. Therefore, the goal of this work is to develop a novel DG algorithm which does not require domain labels and is more interpretable.

We follow the nomenclature of [25], which refers to DG with domain labels as vanilla DG and the more challenging DG without domain labels as compound DG. Our work focuses on addressing compound DG through inferring latent domains from mixed-domain data and using them efficiently. [16, 39, 140] demonstrated that using domain-wise datasets can theoretically achieve lower generalization error bounds and better DG performance than using mixed data directly, indicating the importance of domain information. Furthermore, latent domain discovery helps us understand the workings of models and enhances inter-

pretability. To make the problem tractable, we assume that latent domains are distinct and separable.

In this chapter, we propose HMOE: Hypernetwork-based Mixture of Experts (MoE). MoE is a well-established learning paradigm that aggregates a number of experts by calculating the weighted sum of their predictions [88, 89], where the aggregation weights, also known as gate values, are determined by a routing mechanism and add up to 1. To discover latent domains, HMOE leverages MoE’s divide and conquer property, that is, the routing mechanism can learn to softly partition the input space into subspaces in an unsupervised manner during training [227], with each assigned to an expert. We further expect that each subspace is associated with a latent domain, enabling latent domain discovery. During inference, we can compare the similarities between an unseen test domain and the inferred domains based on gate values, hence improving interpretability. [69, 235] have validated MoE in domain adaptation [214] and mentioned that MoE can leverage the specialty of individual domain and alleviate negative knowledge transfer [191] compared to using a single model to learn multiple different domains.

HMOE innovatively uses a neural network, called a hypernetwork [70], which takes vectors as input to generate the weights of experts of MoE. By mapping vectors into experts, hypernetworks enable the exploration of experts’ similarities in a low-dimensional vector space, facilitating latent domain discovery. Hypernetworks also serve as a link between experts, providing a platform for them to exchange information and promoting knowledge sharing.

MoE’s intrinsic soft partitioning is not always effective and sometimes fails to maintain a consistent division of the input space, especially when the distinction between latent domains is not significant. To address this issue, we propose a differentiable dense-to-sparse Top-1 routing algorithm, which forces gate values to become one-hot and converges to hard partitioning. This leads to sparse-gated MoE, which improves and stabilizes latent domain discovery. In addition, to better incorporate hypernetworks into MoE, we introduce an embedding space that contains a set of learnable embedding vectors corresponding one-to-one with experts. This embedding space is fed to hypernetworks to generate the weights of experts and is also part of the routing mechanism to compute gate values, thus enhancing the interaction between hypernetworks and the routing mechanism.

We summarize our contributions as follows: (1) We present a novel DG method, HMOE, within the framework of MoE. HMOE does not require domain labels, enables latent domain discovery, and offers excellent interpretability. (2) HMOE innovates the use of hypernetworks to generate expert weights and achieves sparse-gated MoE. (3) As far as we know, HMOE is the first work that can jointly learn and utilize latent domains in an end-to-end way. (4) We conduct comprehensive experiments to compare HMOE with other DG methods under a fair and unified evaluation framework – DomainBed [68]. HMOE achieves competitive performance and even state-of-the-art results in some cases.

The rest of this chapter is organized as follows. In Sec. 6.2, we present previous work on DG and compare our proposed HMOE with some existing methods for compound DG. In Sec. 6.3, we introduce the architecture of HMOE and define various losses to train HMOE. In Secs. 6.4 and 6.5, we evaluate HMOE using a toy regression problem and DomainBed. Last, we summarize our work and propose some research directions to improve HMOE in Sec. 6.6.

6.2 RELATED WORK

6.2.1 Domain generalization (DG)

The goal of DG is to train a predictor on known domains that can generalize well to unseen domains.

6.2.1.1 Vanilla DG

The first line of work is to design DG-specific data augmentation techniques to increase the diversity and quantity of training data to improve DG performance [120, 158, 178, 208, 226, 230, 237, 238]. Previous work learned domain-invariant representations through invariant risk minimization [2, 5, 102], kernel methods [16, 57, 60, 140], feature alignment [61, 113, 127, 139, 147, 154, 194, 199, 212], and domain-adversarial training [58, 59, 65, 113, 116]. Another approach is to disentangle latent features into class-specific and domain-specific representations [86, 93, 143, 155, 229]. General machine learning paradigms were also applied to vanilla DG, such as meta-learning [7, 42, 110, 111], self-supervised learning [21, 94], gradient manipulation [84, 165, 180], and distributionally robust optimization [102, 169].

6.2.1.2 Compound DG

There are some DG algorithms that do not require domain labels by design [25, 84, 115, 127, 143, 229]. Besides improving DG performance, latent domain discovery is also an important task for compound DG and contributes to better interpretability. [25, 127] can do this but have two main limitations: (1) Their methods proceed in two phases: first infer latent domains from mixed data and then deal with DG using the inferred domains, which is similar to vanilla DG. The problem is that the second phase depends on the first and cannot provide some feedback to correct possible errors in domain discovery. (2) Their methods assume that domain shift arises from stylistic differences to identify latent domains, which does not always hold.

On the contrary, all components of HMOE are jointly optimized in an end-to-end fashion, and HMOE leverages MoE to discover latent domains without an explicit induced bias on the cause of domain shift.

6.2.2 Hypernetworks

A hypernetwork is a neural network that generates the weights of another target network. Hypernetworks were initially proposed by [70] and have since been applied to optimization problems [124, 144], meta-learning [233], continuous learning [20, 210], multi-task learning [119, 126, 195], few-shot learning [176], and federated learning [177].

6.2.3 Mixture of Experts (MoE)

MoE was originally proposed by [88, 89] and consists of two main components: experts and a gate network, as shown in Fig. 6.1. The output of MoE is the weighted sum of experts, with gate values calculated by the gate network on a per-example basis. In recent years, MoE has regained attention as a way to scale up deep learning models and more efficiently

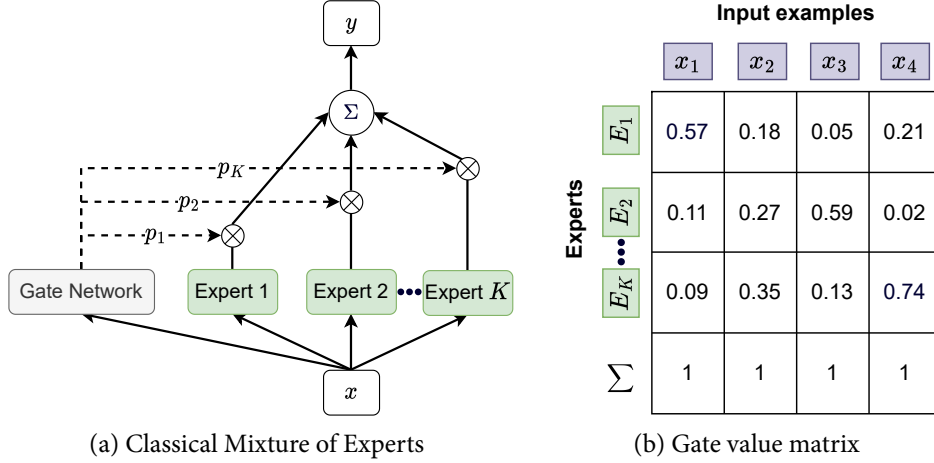


Figure 6.1: In Figure (a), Mixture of Experts calculates the weighted sum of experts’ outputs. In Figure (b), the aggregation weights, also known as gate values, are calculated by the gate network on a per-example basis.

harness modern hardware [44, 48, 49, 107, 179, 241]. In this case, sparse MoE is preferred, which routes each example only to the experts with Top-1 or Top-K gate values, instead of all of them.

6.2.4 Application of hypernetworks and MoE in DG

To the best of our knowledge, no work has applied hypernetworks to solve DG in computer vision. Recently, [207] applied hypernetworks to DG in natural language processing (NLP) and achieved SOTA results on two NLP-related DG tasks.

As for MoE, [108] proposed replacing feed-forward network layer (FFN) of Vision Transformer (ViT) [41] with a sparse mixture of FFN experts to improve DG performance. [69, 235] applied MoE to a task similar to DG, namely domain adaptation [214], but they require domain labels to train an expert for each domain separately. [235] aggregates the outputs of experts via a transformer-based aggregator, but its aggregator is trained with fixed experts and cannot provide probabilities of experts, while HMOE can do this

6.3 METHOD

6.3.1 Problem setting

Let \mathcal{X} denote an input space and \mathcal{Y} a target space. A domain S is characterized by a joint distribution P_{XY}^s on $\mathcal{X} \times \mathcal{Y}$. In vanilla DG setting, we have a training set containing M known domains, i.e., $\mathcal{D}_{tr}^V = \{\mathcal{D}^s\}_{s=1}^M$ with $\mathcal{D}^s = \{(x_i^s, y_i^s, d_i^s)\}_{i=1}^{N_s}$ where $(x_i^s, y_i^s) \sim P_{XY}^s$ and d_i^s is the domain index or label. Also consider a test dataset \mathcal{D}_{te} composed of unknown domains different from those of \mathcal{D}_{tr}^V . Vanilla DG aims to train a robust predictor $f : \mathcal{X} \rightarrow \mathcal{Y}$ on \mathcal{D}_{tr}^V to achieve a minimum predictive error on \mathcal{D}_{te} , i.e., $\min_f \mathbb{E}_{(x,y) \sim \mathcal{D}_{te}} [\ell(f(x), y)]$, where ℓ is the loss function.

Our work focuses on the more difficult compound DG, for which the training set $\mathcal{D}_{tr} = \{(x_i, y_i)\}_{i=1}^N$ contains mixed domains and has no domain annotation. However, as demonstrated in [68, 213, 236], intrinsic inter-domain relationships play a key role in

obtaining better generalization performance. Therefore, our proposed HMOE is designed to discover latent domains by dividing \mathcal{D}_{tr} into clusters that match human intuition about visual relationships between different domains, and to fully leverage the learned domain information to perform well on unknown domains.

6.3.2 Overall architecture

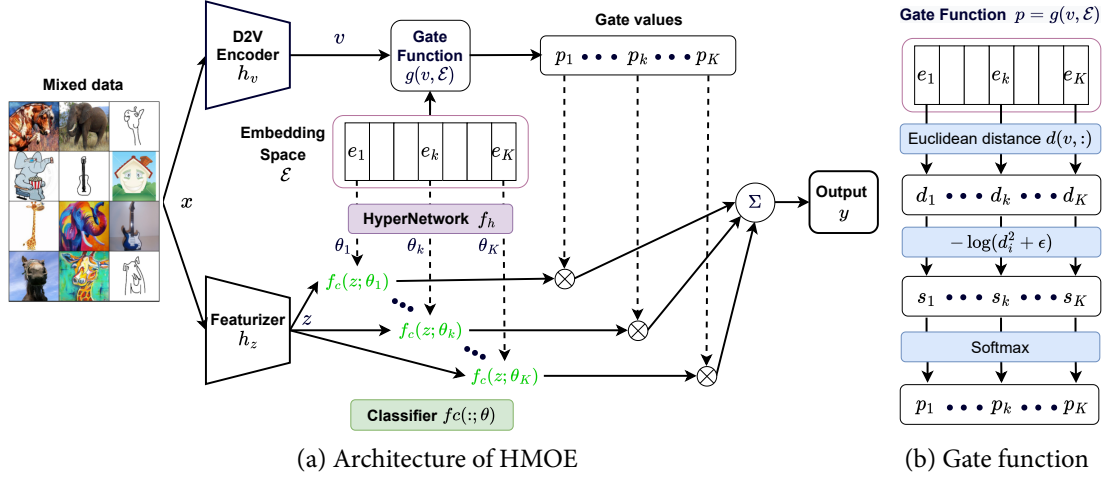


Figure 6.2: (a) In the upper branch (i.e., the domain path), the input is mapped to the embedding space through the D2V encoder, and gate values are calculated by a predefined gate function. In the lower branch (i.e., the classifier path), the hypernetwork takes embedding vectors as input to create a set of classifiers. The final output is the weighted sum of classifiers' outputs. (b) The gate function determines gate values based on the distances between the output of the D2V encoder and the embedding vectors. The smaller the distance, the greater the gate value.

An overview of the HMOE architecture is depicted in Fig. 6.2a. HMOE processes each input x through two paths: the domain path, which is intended to discover latent domains, and the classifier path, which aims to train a classifier expert for each latent domain.

The classifier path begins with a featurizer h_z to extract high-level features from x , which can be a pretrained network, such as VGG [181], ResNet [74], or ViT [41]. We define a discrete learnable embedding space \mathcal{E} consisting of K embedding vectors $\{e_k \in \mathbb{R}^D\}_{k=1}^K$ (D represents the embedding dimension), each corresponding to a classifier expert. These vectors are fed into a hypernetwork f_h to generate a set of weights $\{\theta_k\}_{k=1}^K$, which further form a set of experts $\{f_c(\cdot; \theta_k)\}_{k=1}^K$. The output of the featurizer z is passed to these experts to compute their corresponding outputs, that is, $y_k = f_c(z; \theta_k)$.

The domain path begins with a Domain2Vec (D2V) encoder h_v , which transforms x into the embedding space \mathcal{E} and outputs $v \in \mathbb{R}^D$. The output v is then compared with the embedding vectors through a predefined gate function $g(v, \mathcal{E})$, as shown in Fig. 6.2b, to produce a set of probabilities $\mathbf{p} = \{p_k\}_{k=1}^K$. The final output of HMOE is the weighted sum of the outputs of experts as follows:

$$y = \sum_{k=1}^K p_k y_k = \langle g(h_v(x), \mathcal{E}), [f_c(h_z(x); f_h(e_k))]_{k=1}^K \rangle \quad (6.1)$$

In the classical MoE, the gate network and experts share the same input. In contrast, the D2V encoder of HMOE takes images as input rather than the featurizer’s extracted features, which mainly contain class-specific information for classification. If we connect the D2V encoder to the featurizer, HMOE risks separating the input space based on semantic categories rather than domain-wise distinction.

6.3.3 Hypernetworks

We use the hypernetwork f_h taking a vector e as input to generate the weights of the classifier f_c . In our work, both f_h and f_c are MLPs. In a sense, f_c acts as a placeholder computational graph, e can be seen as a conditioning signal, and f_h maps e to a function. The role of f_h is multifaceted: (1) f_h eases latent domain discovery. (2) f_h allows for the use of many experts without significantly increasing the number of parameters. (3) Compared to the classical MoE, f_h offers another way of interaction between experts and the routing mechanism besides the aggregation of experts. (4) As we will see later, by directly taking the D2V encoder as input, f_h enables the generalization of experts beyond aggregation.

6.3.4 Routing mechanism

6.3.4.1 Gate function

To quantify the responsibilities of experts for each input example and to aggregate experts’ outputs, we need to calculate gate values \mathbf{p} . As shown in Fig. 6.2b, based on the output of the D2V encoder v and the embedding space \mathcal{E} , we define a gate function $g(v, \mathcal{E})$ to calculate \mathbf{p} as follows:

$$d_k = \|v - e_k\|_2 \quad (6.2a)$$

$$s_k = -\log(d_k^2 + \epsilon) \quad (6.2b)$$

$$p_k = \frac{\exp(s_k)}{\sum_{j=1}^K \exp(s_j)} \quad (6.2c)$$

where ϵ is a small value. The negative logarithm in Eq. (6.2b) is used to establish a negative correlation between d_k and p_k (i.e., the smaller d_k , the larger p_k) and to nonlinearly rescale the distance d (i.e., stretch small d and squeeze great d), which makes \mathbf{p} less sensitive to large d .

6.3.4.2 Differentiable dense-to-sparse Top-1 routing

Based on gate values \mathbf{p} , the routing mechanism determines where and how to route input examples. A consistent and cohesive routing is crucial to the training stability and convergence of MoE [32]. In order to stabilize the routing and enhance latent domain discovery to capture less obvious domain differences, sparse-gated MoE is preferable. However, the commonly used Top-1 or Top-K functions are not differentiable and cause oscillatory behavior of gate values during training [72]. To overcome this limitation, we propose a differentiable dense-to-sparse Top-1 routing algorithm by introducing an entropy loss on \mathbf{p} as follows:

$$\mathcal{L}_{en} = \mathbb{E}_{(x,y) \sim \mathcal{D}_{tr}} [\mathbb{H}(g(h_v(x), \mathcal{E}))] \quad (6.3)$$

where $\mathbb{H}(\cdot)$ denotes the entropy of a distribution. In practice, we multiply \mathcal{L}_{en} by γ_{en} that linearly increases from 0 to 1 in the first half of training and remains at 1 in the second. Early on, γ_{en} is small, and the distances between v and the embedding vectors are almost the same, leading to a uniform \mathbf{p} . Therefore, all experts can be fully trained and gradually become specialized. In the later stages, \mathcal{L}_{en} forces \mathbf{p} to become one-hot based on specialized experts.

Due to the negative logarithm in Eq. (6.2b), the D2V encoder has to move towards one of the embedding vectors rather than away from the others in order to minimize \mathcal{L}_{en} . As a result, the output of the D2V encoder will converge to \mathcal{E} and become quantized during training.

6.3.4.3 Expert load balancing

Sparse MoE may suffer from an unbalanced expert load, which is problematic if only a small subset of experts are used while the others are left idle. To alleviate this problem, a widely used approach is to introduce an auxiliary importance loss $CV(I(X))^2$ [179], where X represents a single batch, $I(X) = [I_1(X), \dots, I_K(X)]$ denotes the importance of experts, for which $I_k(X)$ is defined as the sum of gate values assigned to the k th expert (i.e., sum the gate value matrix in Fig. 6.1b along the example dimension), and CV is the coefficient of variation. However, [152] showed that this importance loss over-penalizes unbalanced expert utilization and may be counter-productive, since in most cases the expert load is naturally unbalanced. In this case, [152] defined a distribution $P = I(X) / \sum I(X)$ and used the KL-divergence between P and the uniform distribution \mathcal{U} to balance the expert load, which is also used in our work:

$$\mathcal{L}_{kl} = D_{KL}(P \parallel \mathcal{U}) = D_{KL} \left(\frac{I(X)}{\sum I(X)} \parallel \mathcal{U} \right) \quad (6.4)$$

Compared to the importance loss, \mathcal{L}_{kl} achieves a better trade-off between expert specialization and load balancing.

6.3.5 Embedding space

The embedding space \mathcal{E} plays a key role in HMOE. As we can see, the embedding vectors have an effect on both the generation of expert weights and the routing mechanism, thus serving as a bridge to balance these two parts. In addition, these embedding vectors are learnable like the weights of neural networks and attract the D2V encoder during training under the influence of \mathcal{L}_{en} .

6.3.6 Class-adversarial training on D2V

We expect the D2V encoder h_v to contain as little class-specific information as possible, which ensures that HMOE partitions the input space based on domain-wise distinction rather than semantic categories. Inspired by Domain-Adversarial Neural Networks [59], we define an adversarial classifier f_c^{ad} taking v as input and add the following loss to perform class-adversarial training on h_v :

$$\mathcal{L}_{ad} = \mathbb{E}_{(x,y) \sim \mathcal{D}_{tr}} [\ell_{ce}(f_c^{ad}(GRL(v, \lambda_{grl})), y)] \quad (6.5)$$

where ℓ_{ce} denotes the cross-entropy loss and GRL represents the gradient reversal layer, which acts as an identity function in the forward pass and multiplies the gradient by $-\lambda_{grl}$ in the backward pass. As suggested in [59], we define λ_{grl} as follows:

$$\lambda_{grl} = 2/(1 + \exp(-10 \times pct_{tr})) - 1 \quad (6.6)$$

where pct_{tr} varies linearly from 0 to 1 during training.

6.3.7 Semi-/supervised learning on domains

Due to the probabilistic nature of MoE, given an input x and the corresponding gate values $\mathbf{p} = \{p_k\}_{k=1}^K$, we can interpret p_k as the probability of selecting the k th expert E_k given x , i.e., $p(E_k|x)$. In addition, E_k is thought to be associated with a specific domain \mathcal{S}_m . Therefore, we get $p_k = p(E_k|x) = p(\mathcal{S}_m|x)$. Consider a dataset with domain labels $\mathcal{D}_d = \{(x_i, d_i)\}_{i=1}^{N_d}$ (class labels are not necessary) with $d_i \in \{1, \dots, M_d\}$, we can make use of \mathcal{D}_d as follows:

$$\mathcal{L}_d = \mathbb{E}_{(x,d) \sim \mathcal{D}_d} [\ell_{ce}(\mathbf{p}, d)] \quad (6.7)$$

Note that M_d may be smaller than K , but this has no bearing on the calculation of \mathcal{L}_d . In this case, we assume that the first M_d experts are assigned to M_d domains, and the other experts do not have domain information and learn from the data by themselves. If all domain labels are available in the training data, \mathcal{L}_d becomes supervised learning on domains.

6.3.8 Training and inference

In addition to the losses mentioned above, the supervised loss on targets is as follows:

$$\mathcal{L}_y = \mathbb{E}_{(x,y) \sim \mathcal{D}_{tr}} [\ell_{ce}(\hat{y}, y)] \quad (6.8)$$

where \hat{y} is the prediction of HMOE, as calculated by Eq. (6.1). The final training loss is:

$$\mathcal{L} = \lambda_y \mathcal{L}_y + \lambda_{en} \mathcal{L}_{en} + \lambda_{kl} \mathcal{L}_{kl} + \lambda_{ad} \mathcal{L}_{ad} + \lambda_d \mathcal{L}_d \quad (6.9)$$

where λ are trade-off hyper-parameters to balance different losses. Generally, λ_y is set to 1 and \mathcal{L}_d is not used for compound DG without domain labels.

For inference, we provide three modes: MIX, MAX, and OOD. MIX refers to the mixture of experts that is calculated by Eq. (6.1), MAX employs the output of the expert with the highest gate value, and OOD¹ (Out of Domain) uses the output of a classifier whose weights are generated by the hypernetwork directly taking the D2V encoder as input. OOD enables the generalization of experts beyond aggregation.

6.4 TOY REGRESSION PROBLEM

Although this work focuses on image classification, we start with a toy regression problem to gain some insight into the learning dynamics of HMOE, such as how gate values evolve and how experts become specialized gradually. We use the function $y = \sin(4\pi x)$ to

¹ The OOD inference can be efficiently implemented using PyTorch-based JAX-like library, functorch.

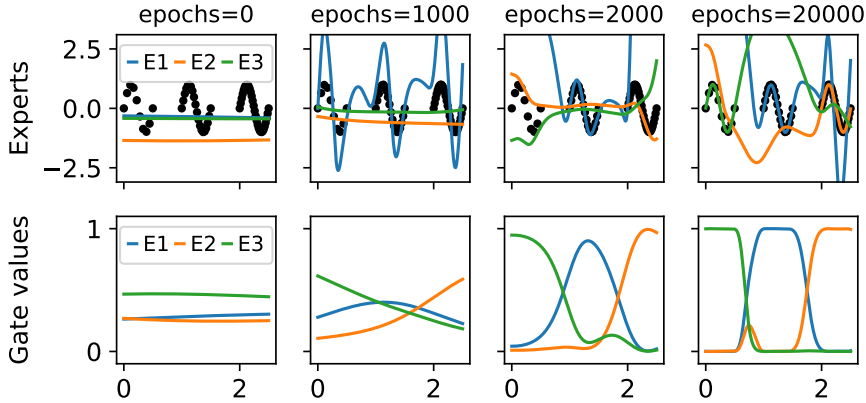


Figure 6.3: We generate some data points using the function $y = \sin(4\pi x)$ in three intervals and fit HMOE with three embedding vectors to these points. This figure shows the experts' outputs and gate values during training. HMOE well identifies three intervals and experts also become specialized.

generate 10, 20, and 30 data points uniformly in three intervals: $[0, 0.5]$, $[1, 1.5]$ and $[2, 2.5]$, respectively. Unequal data points are used to simulate a naturally unbalanced expert load.

HMOE uses three embedding vectors of dimension $D = 8$, which are initialized using the standard normal distribution. All networks of HMOE are MLPs with 32 hidden units. The featurizer is a three-layer MLP whose input size is 1 and output size is 32. The encoder is a three-layer whose input size is 1 and output size is D . The classifier is a two-layer MLP whose input size is 32 and output size is 1. The hypernetwork is a four-layer MLP whose input size is D and output size is the total number of learnable parameters (i.e., weights and biases) of the classifier. In addition, all MLPs use the SiLU activation function [78] except the output layers.

We employ \mathcal{L}_y (use MSE as the loss function), \mathcal{L}_{en} , and \mathcal{L}_{kl} with $\lambda_y = \lambda_{en} = \lambda_{kl} = 1$, and train HMOE using Adam [95] with learning rate 0.001 over 20,000 epochs. The evolution of the experts' outputs and gates values w.r.t. training epochs is shown in Fig. 6.3. We can see that three experts compete with each other and gradually locate their positions, and HMOE manages to identify three intervals even with imbalanced data.

After training, Fig. 6.4 compares three modes of inference, which all coincide well with the training points. This toy regression problem gives us an intuitive understanding of HMOE's ability to detect heterogeneous patterns in data.

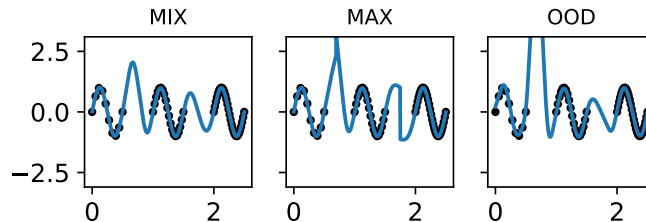


Figure 6.4: This figure shows three modes of inference of HMOE for the toy regression problem.

6.5 DOMAINBED

6.5.1 Datasets and model evaluation

DomainBed [68] provides a unified codebase to implement and train DG algorithms and integrates commonly used DG-related datasets. In this work, we conduct experiments on Colored MNIST with 3 domains [5], Rotated MNIST with 6 domains [61], PACS with 4 domains [109], VLCS with 4 domains [46], OfficeHome with 4 domains [205], and TerraIncognita with 4 domains [9]. In Tab. 6.1, we give detailed statistics and visualize some samples for each domain of each dataset.


























Dataset	Domains				# of classes	# of samples
ColoredMNIST [5]	+90%	+80%	-90%		2	70,000
						
(degree of correlation between color and label)						
RotatedMNIST [61]	0°	15°	30°	45°	60°	75°
						
VLCS [46]	Caltech101	LabelMe	SUN09	VOC2007	5	10,729
						
PACS [109]	Art	Cartoon	Photo	Sketch	7	9,991
						
OfficeHome [205]	Art	Clipart	Product	Real	65	15,588
						
TerraIncognita [9]	L100	L38	L43	L46	10	24,788
						
(camera trap location)						

Table 6.1: Description and visualization of datasets used in our experiments (Adapted from [68])

To select models and tune hyper-parameters, DomainBed provides three options, of which we select the training-domain validation that randomly draws 80% from the data of each training domain to form the training set and uses the remaining as the validation set. This option best matches the setting of compound DG without domain labels and access to test domains.

6.5.2 Implementation details

For Colored and Rotated MNIST, following [68], we use as the featurizer a four-layer ConvNet (refer to Appendix D.1 of [68]). The D2V encoder h_v consists of two conv layers (32 units, 3×3 kernels, ReLU), followed by global average pooling and a fully-connected (fc) layer to map to the embedding dimension D .

For other datasets, we use ResNet-50² pretrained on ImageNet [38] as the featurizer and freeze all batch normalization layers. The D2V encoder h_v cascades 3 conv layers (64-128-256 units, stride 2, 4×4 kernels, ReLU), two residual blocks (each has 2 conv layers with 256 units, 3×3 kernels, ReLU), and a 3×3 conv layer with D units followed by global average pooling. We use Instance Normalization [200] with learnable affine parameters before all ReLU of h_v .

For all datasets, the classifier f_c is a fc layer whose input size is the featurizer’s output size (128 for ConvNet and 2048 for ResNet-50) and output size is the number of classes. The hypernetwork f_h is a five-layer MLP with 256-128-64-32 hidden units and SiLU [78] except the output layer, and its input size is D and output size is the total number of learnable parameters (i.e., weights and biases) of f_c . In addition, we use the hyperfan method proposed by [23] to initialize f_h . If \mathcal{L}_{ad} is used, the adversarial classifier is a three-layer MLP with 256 hidden units and ReLU except the output layer, and its input size is D and output size is the number of classes. We set $D = 32$ and initialize embedding vectors using the standard normal distribution.

We define three HMOE variants based on the number of embedding vectors K and whether domain labels are used:

- (1) HMOE-DL: Domain labels of \mathcal{D}_{tr} are provided. In this case, we only use \mathcal{L}_y and \mathcal{L}_d with $\lambda_y = \lambda_d = 1$ and discard other losses, and K is the number of training domains per dataset.
- (2) HMOE-DN: Domain numbers are known but domain labels. In this case, K is the number of training domains per dataset. We use \mathcal{L}_y , \mathcal{L}_{en} , \mathcal{L}_{kl} , and \mathcal{L}_{ad} with $\lambda_y = \lambda_{en} = \lambda_{kl} = 1$ and $\lambda_{ad} = 0.01$.
- (3) HMOE-ND: No domain information is available and we use a fixed $K = 5$. The setting of losses is the same as in HMOE-DN.

DomainBed trains all DG algorithms with Adam for 5,000 iterations. For Colored and Rotated MNIST / other datasets, the learning rate is 0.001 / 5e-5, the batch size is 64 / 32 \times number of training domains, and models are evaluated on the validation set every 100 / 300 iterations. Each experiment uses one domain of a dataset as the test domain and trains algorithms on the others, which is repeated 3 times with different random seeds. The average accuracy over 3 replicates is reported. In addition, we do not tune HMOE’s hyper-parameters and use the settings mentioned above consistently. Other DG algorithms also use the default settings predefined in DomainBed. All experiments are performed on PyTorch using a A5000 GPU.

6.5.3 Results

We use the up-to-date domain generalization benchmark on DomainBed, and the comparison of HMOE (3 variants and 3 inference modes) with other DG algorithms is shown in Tab. 6.2, where the best results are underlined. ERM means the vanilla supervised learning that just fine-tunes ResNet-50 on mixed data, also called DeepAll in some papers and serving as a performance baseline. We report the average accuracy of all test domains for each dataset. Refer to Appendix B for detailed domain generalization results of each test domain for each dataset.

² For a fair comparison with other DG algorithms, we use the pretrained ResNet-50 of IMAGENET1K-V1 in PyTorch, although V2 is better.

Algorithm		Colored MNIST	Rotated MNIST	VLCS	PACS	OfficeHome	Terra Incognita
w/ Domain Labels							
IRM [5]		52.0	97.7	78.5	83.5	64.3	47.6
GroupDRO [169]		52.1	98.0	76.7	84.4	66.0	43.2
Mixup [225]		52.1	98.0	77.4	84.6	68.1	47.9
MLDG [110]		51.5	97.9	77.2	84.9	66.8	47.7
CORAL [194]		51.5	98.0	<u>78.8</u>	86.2	<u>68.7</u>	47.6
MMD [113]		51.5	97.9	77.5	84.6	66.3	42.2
DANN [59]		51.5	97.8	78.6	83.6	65.9	46.7
CDANN [116]		51.7	97.9	77.5	82.6	65.8	45.8
MTL [16]		51.4	97.9	77.2	84.6	66.4	45.6
ARM [231]		<u>56.2</u>	<u>98.2</u>	77.6	85.1	64.8	45.5
VREx [102]		51.8	97.9	78.3	84.9	66.4	46.4
HMOE-DL	MIX	51.6	97.3	76.7	83.5	64.7	45.0
	MAX	51.7	97.0	77.6	83.9	63.2	43.2
	OOD	51.7	97.4	76.8	84.5	63.7	44.0
w/o Domain Labels							
ERM [204]		51.5	98.0	77.5	85.5	66.5	46.1
RSC [84]		51.7	97.6	77.1	85.2	65.5	46.6
SagNet [143]		51.7	98.0	77.8	86.3	68.1	48.6
HMOE-DN	MIX	51.9	97.5	76.8	84.8	65.4	48.7
	MAX	51.9	97.4	76.6	85.1	65.4	<u>49.5</u>
	OOD	51.9	97.5	75.8	84.9	65.3	48.4
HMOE-ND	MIX	51.6	97.5	76.6	84.5	65.5	48.4
	MAX	51.7	97.4	76.8	86.6	65.5	45.0
	OOD	51.7	97.5	76.7	<u>87.0</u>	65.6	47.1

Table 6.2: Domain generalization results on DomainBed

For Colored and Rotated MNIST, all algorithms exhibit similar performance, except the impressive results of ARM. HMOE achieves SOTA results on PACS and TerraIncognita, which well demonstrates its effectiveness. However, ERM outperforms HMOE and most DG algorithms for VLCS and OfficeHome. VLCS contains real camera photos, and its domain shift is mainly caused by changes in scene and perspective. We find that the visual differences between different domains of VLCS are subtle. In this case, forcing to reduce or model domain discrepancy may cause or aggravate overfitting. For OfficeHome, this is also the case.

Interestingly, HMOE-DL is inferior to HMOE-DN/ND in most cases, which implies that HMOE works better using its own learned domain information than using given domain labels. We find that latent domains discovered by HMOE are more human-intuitive than original domain labels (see Sec. 6.5.4).

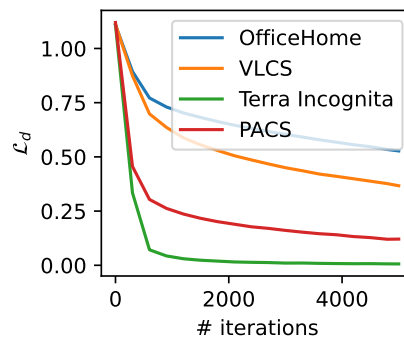


Figure 6.5: Average of \mathcal{L}_d over all test domains per dataset

HMOE-DN / ND are basically tied in terms of performance. For three inference modes, MAX and OOD are competitive with MIX in most cases and can be used to sacrifice a little accuracy for efficiency in practice because MAX and OOD are more computationally efficient without computing all experts like MIX.

6.5.4 Latent domain discovery

We use t-SNE [202] to visualize the output of the D2V encoder, as shown in Fig. 6.6. We can see that HMOE effectively partitions the mixed data into a number of clusters, each centered around an embedding vector. As expected, the output of the D2V encoder converges to embedding vectors. For PACS (Fig. 6.6a), the training domains are well separated. Some cartoon images look quite artistic and are classified as art. In addition, test photo samples are projected into the art cluster, suggesting that the D2V encoder should capture some semantics about latent domains since photo is closest to art. If we increase the number of embedding vectors K to 5, cartoon and sketch clusters are split into two sub-clusters, as shown in Fig. 6.6b. For TerraIncognita (Fig. 6.6c), the dots of the same color are largely clustered together. The training domains are to some extent separated, although L38 and L43 are partially mixed. The test domain L46 seems to be more similar to L100. For OfficeHome (Fig. 6.6d), the training domains are mixed within each cluster, indicating a conflict between domain labels and inferred domains. This also explains why \mathcal{L}_d cannot be significantly reduced for OfficeHome in Fig. 6.5.

To intuitively understand how HMOE distinguishes between domains, we visualize some samples to compare domain labels and HMOE’s clusters, as shown in Fig. 6.7. HMOE seems

to partition TerraIncognita based on illumination and OfficeHome based on background complexity, which more matches human intuition than domain labels.

After the above analysis, we conclude that the success of HMOE, e.g., SOTA on PACS and TerraIncognita, can be attributed to its ability to self-learn more reasonable and informative domain knowledge and use it efficiently.

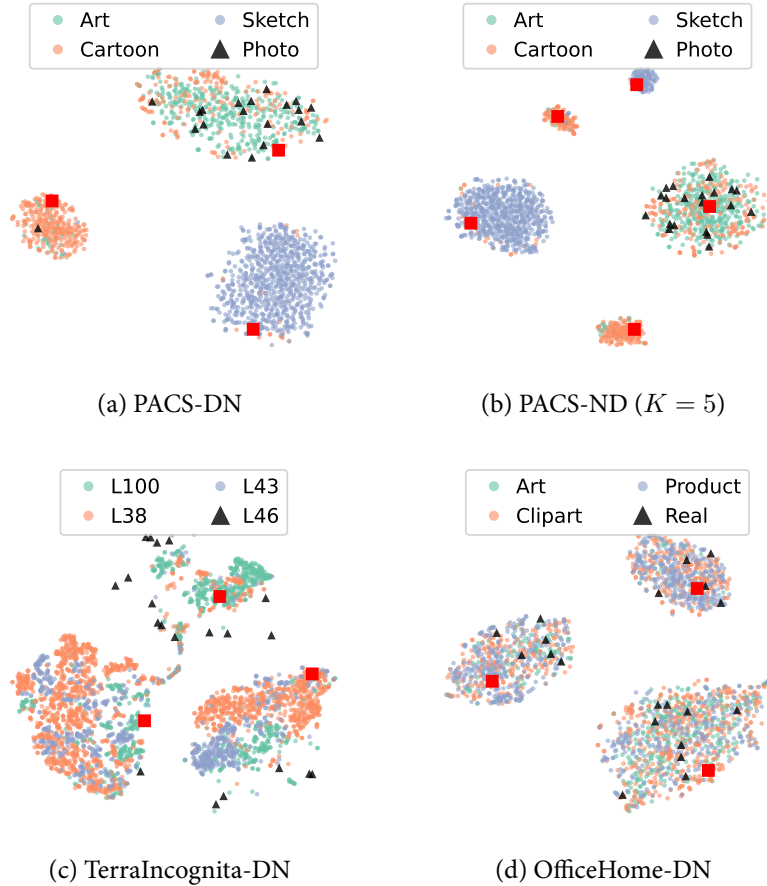


Figure 6.6: The t-SNE visualization of the output of the D2V encoder. The suffixes in captions (DN and ND) represent HMOE-DN / ND, red squares are embedding vectors, black triangles are 20 samples randomly drawn from the test domain, and other dots are training domains. The silhouette coefficients are 0.73, 0.72, 0.54 and 0.63 for Figs. 6.6a to 6.6d, respectively.

6.5.5 Ablation study

In this section, we conduct an ablation study to analyze the contribution of each component of HMOE. The results are shown in Tab. 6.3, which reports the average accuracy of three inference modes. We employ the silhouette coefficient (SC) to quantitatively evaluate the HMOE’s clustering based on both the compactness and separation of clusters. SC ranges from -1 (poor) to 1 (good). We use gate values to identify clusters and the output of the D2V encoder to measure the distance between them.

TOP-1 ROUTING \mathcal{L}_{en} AND EXPERT LOAD BALANCING \mathcal{L}_{kl}

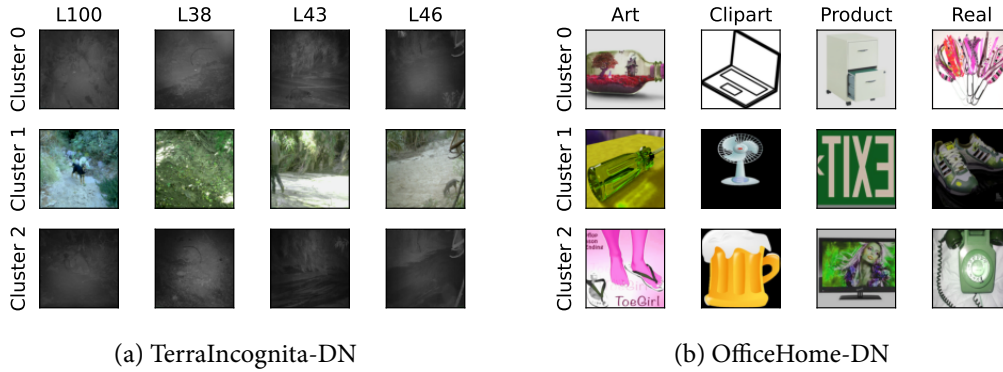
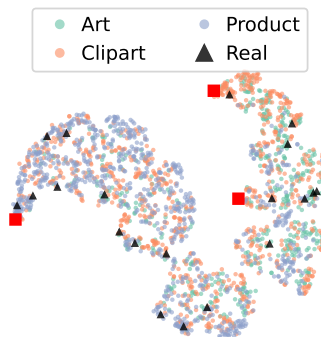


Figure 6.7: Comparison between domain labels and HMOE clusters

Name	\mathcal{L}_{en}	\mathcal{L}_{kl}	\mathcal{L}_{ad}	VLCS	PACS	Office	TerraInc	Avg. SC
H1	-	-	-	76.1	83.2	64.2	46.8	0.37
H2	-	-	✓	76.8	84.2	64.7	47.5	0.27
H3	✓	-	-	76.3	81.8	63.7	43.4	Collapse
H4	✓	-	✓	75.9	82.1	62.2	44.1	Collapse
H5	✓	✓	-	76.0	84.0	64.2	47.0	0.65
H6	✓	✓	✓	76.4	84.9	65.4	48.9	0.60

Table 6.3: Ablation study for HMOE-DN (avg. accuracy of MIX, MAX and OOD is reported, ✓ means the corresponding loss is used, and SC denotes the silhouette coefficient.)

Tab. 6.3 shows that the joint use of \mathcal{L}_{en} and \mathcal{L}_{kl} leads to better clustering with greater SC and promotes latent domain discovery. Without them, HMOE divides the data through MoE’s intrinsic soft partitioning, as shown in Fig. 6.8. H6 outperforms H2 in most cases, which could indicate that better clustering helps improve the DG performance. However, H1 and H5 have comparable performance, probably due to the absence of \mathcal{L}_{ad} . We find that \mathcal{L}_{en} without \mathcal{L}_{kl} suffers from the learning collapse problem, i.e., some embedding vectors collapse together and the D2V encoder outputs similar values. An example is shown Fig. 6.9. This shows the importance of \mathcal{L}_{kl} .

Figure 6.8: Soft partitioning of HMOE for OfficeHome using $K = 3$ and only the target loss \mathcal{L}_y

CLASS-ADVERSARIAL TRAINING \mathcal{L}_{ad} helps improve accuracy in most cases, verifying the necessity of removing class-specific information from the D2V encoder. H2 and H6

have smaller SC than H1 and H5, respectively, which is logical as class information can still be used by H1 and H5 for clustering, but is somewhat removed for H2 and H6 via \mathcal{L}_{ad} .

6.5.6 More empirical analysis

MORE EMBEDDING VECTORS We further increase K to 8 and find that HMOE also suffers from the learning collapse problem, as shown in Fig. 6.9. When embedding vectors are much more than needed, HMOE has difficulties in how to correctly assign the data to different experts.

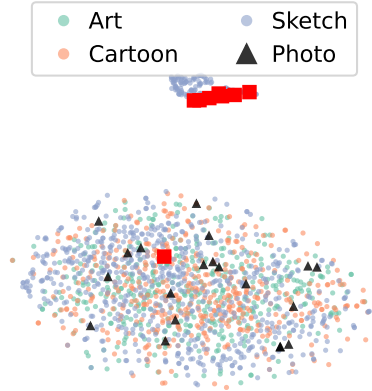


Figure 6.9: Learning collapse for PACS with $K = 8$

TRAIN HMOE USING OOD As mentioned earlier, hypernetworks f_h associate experts with vectors, enabling the exploration of experts' similarities in a low-dimensional vector space and facilitating latent domain discovery. To verify this, we use OOD (i.e., f_h takes the D2V encoder h_v as input) to train HMOE, which means that each input can have its own expert, instead of selecting from a few given experts. This involves minimizing the following loss:

$$\mathcal{L}_{ood} = \mathbb{E}_{(x,y) \sim \mathcal{D}_{tr}} [\ell_{ce}(f_c(h_z(x); f_h(h_v(x))), y)] \quad (6.10)$$

After training, the D2V encoder is also able to distinguish between different domains, as shown in Fig. 6.10. This nicely demonstrates the role of hypernetworks in learning and capturing semantic similarities across domains.

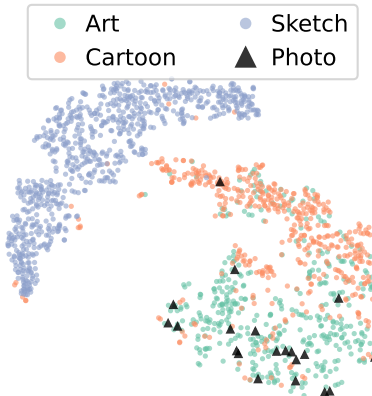


Figure 6.10: Train HMOE using OOD for PACS

USE SWIN TRANSFORMER AS FEATURIZER [108, 182] investigate the impact of the backbone architecture (i.e., the featurizer for HMOE) on DG, and [108] found that the transformer-based backbone outperforms the CNN-based counterpart. Motivated by this, we replace ResNet-50 with Swin Transformer [122] (the pretrained small version in PyTorch and its output size is 768) as the featurizer of HMOE-DN. The results are shown in Tab. 6.4, where ERM just fine-tunes Swin Transformer and we report the MIX mode of HMOE-DN. Tab. 6.4 is clearly superior to Tab. 6.2. HMOE-DN outperforms ERM on PACS and TerraIncognita and they are comparable on the other two datasets.

	VLCS	PACS	OfficeHome	TerraIncognita
ERM [204]	79.8	86.9	76.0	54.1
HMOE-DN	79.6	87.6	76.1	55.3

Table 6.4: Use Swin Transformer as featurizer of HMOE-DN

6.6 CONCLUSION

This chapter presents a novel DG method, HMOE, which does not require domain labels, enables latent domain discovery, and provides excellent interpretability. HMOE uses the framework of Mixture of Experts (MoE) to solve the DG problem and employs hypernetworks to generate the weights of experts. Compared to other DG methods requiring domain labels, HMOE shows competitive performance and achieves SOTA results on the PACS and TerraIncognita datasets. It is worth mentioning that the discovery and utilization of domain information are jointly undertaken for HMOE, rather than in stages like other related work.

However, it also remains unclear how to effectively determine an appropriate number of experts or embedding vectors to fully explore domain information while avoiding the learning collapse. A promising solution that we will explore in future work is to use tree-structured hierarchical MoE to discover hierarchical domain knowledge, where each level contains only a number of experts but the number of multi-level inferred domains grows exponentially.

Finally, HMOE is versatile and scalable, and it should also be applicable to a wide range of problems beyond the scope of DG that are troubled by heterogeneous patterns.

CONCLUSION AND PERSPECTIVES

7.1 CONCLUSION

In this thesis, we have concentrated on accelerating large-scale flash calculations through parallel computing on specialized hardware accelerators, e.g., GPUs. By processing a batch of inputs simultaneously, we have achieved significant speed-up compared to Carnot, our in-house C++-based thermodynamic library, which conducts flash calculations one at a time on CPUs. Our success can be attributed to two key contributions: the vectorization of algorithms using the deep learning framework — PyTorch, which allows us to leverage a wide range of hardware without requiring code modifications, and the use of neural networks to replace time-consuming subroutines and provide more accurate initial values for iterative processes. Specifically, we replaced stability analysis, the Wilson approximation, and equations of state with neural networks to predict the stability of mixtures, provide better initialization of distribution coefficients, and calculate fugacity coefficients, respectively. We prioritized both speed and reliability rather than blindly pursuing acceleration. This is crucial in mitigating the malfunctioning of neural networks caused by incorrect predictions. For instance, when neural networks exhibit low confidence in predicting the stability of mixtures, we continue to employ stability analysis. Additionally, we combined neural networks with Carnot to deliver more accurate fugacity coefficients for flash calculations, which turned out to be much faster than using Carnot alone.

We also delved into the challenge of effectively learning heterogeneous patterns in data, which stems from our need to learn discontinuous fugacity coefficients. To tackle this problem, we introduced a clustered regression network that can automatically divide the entire space into two continuous parts and assign each part to an expert network. We further extended the concept of discontinuity in regression to the field of image classification, namely domain generalization (DG), and proposed a novel DG approach called HMOE, which is capable of unsupervised learning of latent domains from mixed data and efficiently using them via Mixture of Experts (MoE), with each expert being automatically assigned to a domain. HMOE pioneers the use of hypernetworks to generate expert weights and achieves sparse-gated MoE. Notably, HMOE can jointly learn and utilize latent domains in an end-to-end manner.

7.2 PERSPECTIVES

7.2.1 Improve the generalization of neural networks to components

We utilize neural networks to carry out a variety of machine learning tasks for a mixture of fixed components, including both classification tasks such as predicting stability and regression tasks such as learning distribution coefficients and fugacity coefficients. When some components of the mixture are no longer needed, we can reuse original neural networks by simply setting the corresponding molar fractions to 0 in the input of neural networks. However, in the case of replacing or adding components, we have to retrain new neural networks from scratch. Although we employ practical techniques, e.g., the cyclic

learning rate schedule, to enable neural networks to achieve superb performance with a limited number of training epochs, training a neural network for every possible mixture is clearly impractical and inefficient due to the infinite number of mixtures. Therefore, it is essential to equip neural networks with the ability to generalize to different components.

To accomplish this, one potential solution is to incorporate molecular descriptors into the input, enabling neural networks to factor in the properties of individual components and the interactions between different components. Graph neural networks [172] are particularly suitable for this purpose. Fig. 7.1 depicts an example of how to construct graph data and utilize graph neural networks for different machine learning tasks.

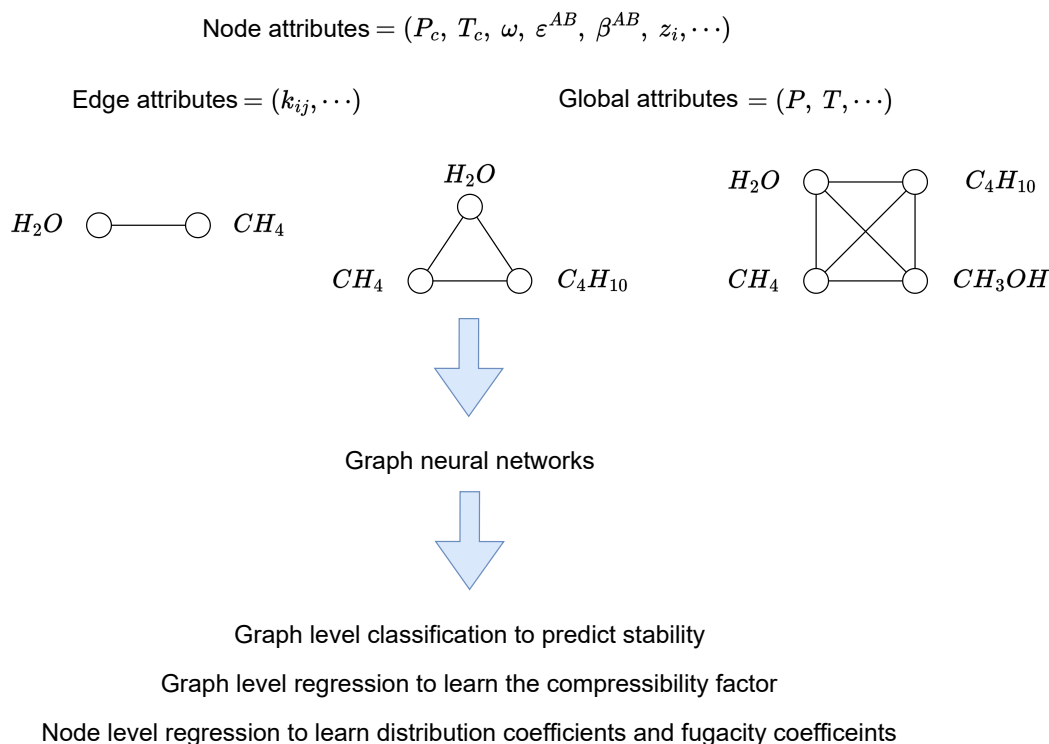


Figure 7.1: This figure demonstrates the construction of graph data to consider the properties of components and the use of graph neural networks for various machine learning tasks. The graph data consists of node, edge and global attributes, where P_c and T_c are the critical pressure and temperature, respectively, ω is the acentric factor, ε^{AB} and β^{AB} denote the association energy and volume used in the CPA EoS, respectively, and k_{ij} refers to the binary interaction parameter. Note that we set z_i as a node attribute and P and T as global attributes. Mixtures of different components are used to train graph neural networks simultaneously.

7.2.2 A more interpretable method for domain generalization

While our proposed HMOE can partition mixed data into multiple clusters, each of which contains images that seem to share certain semantic similarities, such as background complexity, illumination strength, and painting style, it is still uncertain what criteria HMOE employs to conduct clustering. As a result, we aim to design a more interpretable approach for domain generalization in the absence of domain labels.

To achieve this goal, one potential approach is to use the framework of Variational Autoencoder (VAE) [96] for disentangled representation learning [198] in order to identify the explanatory factors of the underlying data-generating process. This approach requires us to make two assumptions: (1) Images are generated from two sets of factors of variation, class-related z_y and domain-dependent z_d , with one set being independent of the other. (2) Each set of factors has a clustering structure controlled by a categorical variable, i.e., class y or domain d . Specifically, the space of factors can be divided into a number of distinct or less overlapping manifolds, each of which is associated with a class or domain category. The first assumption permits the disentanglement of factors of variation into z_y and z_d , and the second one indicates two priors of z_y and z_d conditioned on y and d , respectively. [92] demonstrated that conditionally factorized priors enhance the identifiability of VAE, providing greater opportunities to discover the true latent factors.

To enforce these two assumptions, Fig. 7.2 depicts the prototype of a VAE-based DG method, which consists of class and domain encoders to yield z_y and z_d , respectively, and a decoder to reconstruct images based on the concatenation of z_y and z_d . Moreover, z_y is learned with the guidance of available class labels, which exhibits strong invariance to cross-domain variation and allows us to develop a robust class classifier, while z_d is learned in an unsupervised manner during data reconstruction. Once all components of Fig. 7.2 are well trained, the decoder can act as an image generator, enabling the investigation of the meaning of each factor of z_d by varying its value to observe the changes in reconstructed images.

Our next step is to devise an efficient disentanglement strategy that enhances the identifiability of z_d by minimizing its informativeness regarding classes y and reduces the mutual information between z_y and z_d to promote disentanglement.

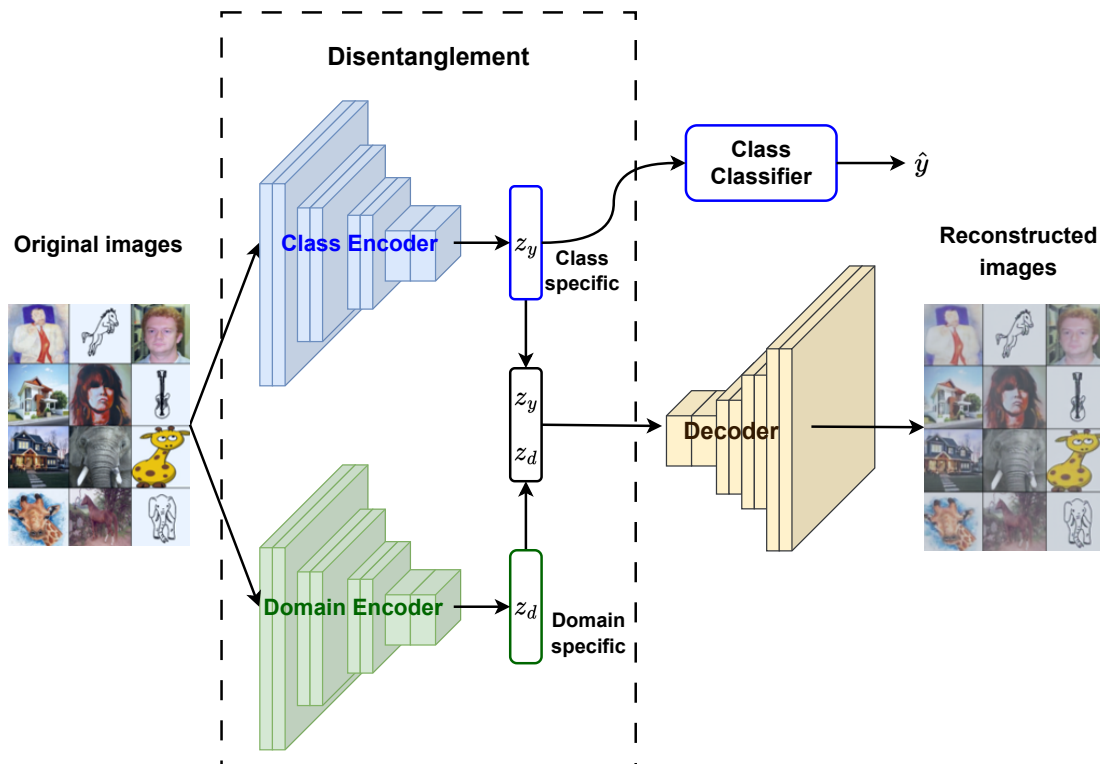


Figure 7.2: This figure shows the prototype of a VAE-based DG method aiming to disentangle latent representation into class/domain-specific factors, i.e., z_y and z_d .

Part IV

APPENDIX

DERIVATION OF THE CLOSED-FORM EXPRESSION OF THE PROBABILITY DENSITY FUNCTION

This appendix is the supplementary material to Sec. 4.3. We will use capital letters to represent random variables instead of lowercase letters used in the text to align with the thermodynamic notation. In this appendix, we aim to derive the closed-form expression for the probability density function of Z_i , which is sampled in the following way:

$$X_i \sim U(0, 1) \quad \text{using LHS}$$

$$Z_i = \frac{X_i}{Y} \quad \text{where} \quad Y = \sum_{i=1}^N X_i$$

The distribution of X_i is a uniform distribution $U(0, 1)$ with mean $\mu_0 = 0.5$ and variance $\sigma_0^2 = 1/12$. Without loss of generality, we use X and Z to refer to any X_i and Z_i , respectively. Based on the central limit theorem, the distribution of Y can be approximated by a normal distribution $\mathcal{N}(\mu_Y, \sigma_Y^2)$, where $\mu_Y = N\mu_0$ and $\sigma_Y^2 = N\sigma_0^2$. By definition, we can calculate the cumulative distribution function of Z as follows:

$$F_Z(z) = P(Z \leq z) = P\left(\frac{X}{Y} \leq z\right) \quad (\text{A.2})$$

Then we express $F_Z(z)$ as an integration of the joint density function of X and Y , denoted by $f_{X,Y}(x, y)$, as follows:

$$F_Z(z) = \int_0^1 \int_{x/z}^{\infty} f_{X,Y}(x, y) dy dx \quad (\text{A.3})$$

Based on Bayes' rule, we have $f_{X,Y}(x, y) = f_{Y|X}(y|x) f_X(x)$. Note that X and Y are not independent of each other, and the conditional distribution of Y given X , denoted by $f_{Y|X}(y|x)$, can be approximated by a normal distribution $\mathcal{N}(\mu + x, \sigma^2)$, where $\mu = (N_c - 1)\mu_0$ and $\sigma^2 = (N_c - 1)\sigma_0^2$. Consequently, we can calculate $f_{X,Y}(x, y)$ as follows:

$$f_{X,Y}(x, y) = f_{Y|X}(y|x) f_X(x) \quad (\text{A.4})$$

$$= f_{Y|X}(y|x) \quad (\text{A.5})$$

$$= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y - (\mu + x)}{2\sigma^2}\right) \quad (\text{A.6})$$

$$\text{for } 0 \leq x \leq 1 \quad \text{and} \quad x \leq y \leq N - 1 + x$$

We then get the the probability density function of Z by differentiating Eq. (A.3):

$$f_Z(z) = \frac{d}{dz} \int_0^1 \left(F_{X,Y}(x, N - 1 + x) - F_{X,Y}\left(x, \frac{x}{z}\right) \right) dx \quad (\text{A.7a})$$

$$= \int_0^1 \frac{d}{dz} \left(F_{X,Y}(x, N - 1 + x) - F_{X,Y}\left(x, \frac{x}{z}\right) \right) dx \quad (\text{A.7b})$$

$$= \int_0^1 \frac{x}{z^2} f_{X,Y}\left(x, \frac{x}{z}\right) dx \quad (\text{A.7c})$$

$$= \int_0^1 \frac{x}{\sqrt{2\pi}\sigma z^2} \exp\left(-\frac{\left(\frac{1-z}{z}x - \mu\right)^2}{2\sigma^2}\right) dx \quad (\text{A.7d})$$

where $F_{X,Y}(x, y)$ denotes the joint cumulative distribution of X and Y . We transform Eq. (A.7d) using $u = (1 - z)x/z$ and $\nu = u - \mu$, and we have:

$$f_Z(z) = \int_{-\mu}^{\frac{1-z}{z}-\mu} \frac{\nu + \mu}{\sqrt{2\pi}\sigma(1-z)^2} \exp\left(-\frac{\nu^2}{2\sigma^2}\right) d\nu \quad (\text{A.8})$$

Fortunately, the above equation can be further simplified to yield the final closed-form expression of $f_Z(z)$ that we used in Sec. 4.3:

$$\begin{aligned} f_Z(z) &= \frac{\sigma}{\sqrt{2\pi}(1-z)^2} \left(\exp\left(-\frac{\mu^2}{2\sigma^2}\right) - \exp\left(-\frac{(\frac{1-z}{z}-\mu)^2}{2\sigma^2}\right) \right) \\ &+ \frac{\mu}{(1-z)^2} \left(\Phi\left(\frac{\frac{1-z}{z}-\mu}{\sigma}\right) - \Phi\left(\frac{-\mu}{\sigma}\right) \right) \end{aligned} \quad (\text{A.9})$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution, $\mu = (N - 1)\mu_0$ and $\sigma^2 = (N - 1)\sigma_0^2$.

DETAILED DOMAIN GENERALIZATION RESULTS

This appendix is the supplementary material to Sec. 6.5.3, and we provide the domain generalization results of each test domain for each dataset. The best results are underlined.

Algorithm		+90%	+80%	-90%	Avg
w/ Domain Labels					
IRM [5]		72.5	73.3	10.2	52.0
GroupDRO [169]		73.1	73.2	10.0	52.1
Mixup [225]		72.7	73.4	10.1	52.1
MLDG [110]		71.5	73.1	9.8	51.5
CORAL [194]		71.6	73.1	9.9	51.5
MMD [113]		71.4	73.1	9.9	51.5
DANN [59]		71.4	73.1	10.0	51.5
CDANN [116]		72.0	73.0	10.2	51.7
MTL [16]		70.9	72.8	10.5	51.4
ARM [231]		<u>82.0</u>	<u>76.5</u>	10.2	<u>56.2</u>
VREx [102]		72.4	72.9	10.2	51.8
	MIX	71.8	72.9	10.1	51.6
HMOE-DL	MAX	71.9	73.1	10.1	51.7
	OOD	71.9	73.2	10.1	51.7
w/o Domain Labels					
ERM [204]		71.7	72.9	10.0	51.5
RSC [84]		71.9	73.1	10.0	51.7
SagNet [143]		71.8	73.0	10.3	51.7
	MIX	72.0	73.0	<u>10.7</u>	51.9
HMOE-DN	MAX	72.0	73.0	<u>10.7</u>	51.9
	OOD	72.0	73.0	<u>10.7</u>	51.9
	MIX	71.6	73.2	10.1	51.6
HMOE-ND	MAX	71.9	73.2	10.2	51.7
	OOD	71.6	73.5	10.1	51.7

Table B.1: Domain generalization results on ColoredMNIST

Algorithm		0	15	30	45	60	75	Avg
w/ Domain Labels								
IRM [5]		95.5	98.8	98.7	98.6	98.7	95.9	97.7
GroupDRO [169]		95.6	98.9	98.9	99.0	98.9	<u>96.5</u>	98.0
Mixup [225]		95.8	98.9	98.9	98.9	98.8	<u>96.5</u>	98.0
MLDG [110]		95.8	98.9	<u>99.0</u>	98.9	99.0	95.8	97.9
CORAL [194]		95.8	98.8	98.9	99.0	98.9	96.4	98.0
MMD [113]		95.6	98.9	<u>99.0</u>	99.0	98.9	96.0	97.9
DANN [59]		95.0	98.9	<u>99.0</u>	99.0	98.9	96.3	97.8
CDANN [116]		95.7	98.8	98.9	98.9	98.9	96.1	97.9
MTL [16]		95.6	99.0	<u>99.0</u>	98.9	99.0	95.8	97.9
ARM [231]		<u>96.7</u>	<u>99.1</u>	<u>99.0</u>	99.0	<u>99.1</u>	<u>96.5</u>	<u>98.2</u>
VREx [102]		95.9	99.0	98.9	98.9	98.7	96.2	97.9
HMOE-DL	MIX	94.5	97.7	98.8	98.7	99.0	94.9	97.3
	MAX	94.1	97.9	98.5	98.5	98.6	94.7	97.0
	OOD	95.0	97.9	98.6	98.9	<u>99.1</u>	94.7	97.4
w/o Domain Labels								
ERM [204]		95.9	98.9	98.8	98.9	98.9	96.4	98.0
RSC [84]		94.8	98.7	98.8	98.8	98.9	95.9	97.6
SagNet [143]		95.9	98.9	<u>99.0</u>	<u>99.1</u>	99.0	96.3	98.0
HMOE-DN	MIX	94.1	98.6	98.7	98.6	99.0	95.9	97.5
	MAX	94.0	98.6	98.7	98.6	98.8	95.9	97.4
	OOD	94.0	98.6	98.7	98.6	99.0	95.9	97.5
HMOE-ND	MIX	94.1	98.6	98.7	98.6	99.0	95.9	97.5
	MAX	94.0	98.6	98.7	98.6	98.8	95.9	97.4
	OOD	94.0	98.6	98.7	98.6	99.0	95.9	97.5

Table B.2: Domain generalization results on RotatedMNIST

Algorithm		Caltech101	LabelMe	SUN09	VOC2007	Avg
w/ Domain Labels						
IRM [5]		98.6	64.9	73.4	77.3	78.5
GroupDRO [169]		97.3	63.4	69.5	76.7	76.7
Mixup [225]		98.3	64.8	72.1	74.3	77.4
MLDG [110]		97.4	65.2	71.0	75.3	77.2
CORAL [194]		98.3	<u>66.1</u>	73.4	<u>77.5</u>	<u>78.8</u>
MMD [113]		97.7	64.0	72.8	75.3	77.5
DANN [59]		<u>99.0</u>	65.1	73.1	77.2	78.6
CDANN [116]		97.1	65.1	70.7	77.1	77.5
MTL [16]		97.8	64.3	71.5	75.3	77.2
ARM [231]		98.7	63.6	71.3	76.7	77.6
VREx [102]		98.4	64.4	<u>74.1</u>	76.2	78.3
HMOE-DL	MIX	97.7	62.3	72.0	74.8	76.7
	MAX	97.0	63.6	73.2	76.7	77.6
	OOD	97.0	63.4	72.0	74.9	76.8
w/o Domain Labels						
ERM [204]		97.7	64.3	73.4	74.6	77.5
RSC [84]		97.9	62.5	72.3	75.6	77.1
SagNet [143]		97.9	64.5	71.4	<u>77.5</u>	77.8
HMOE-DN	MIX	97.1	63.8	71.2	75.1	76.8
	MAX	97.4	63.4	70.9	74.8	76.6
	OOD	96.4	62.9	69.3	74.7	75.8
HMOE-ND	MIX	95.8	65.7	72.4	72.5	76.6
	MAX	97.3	61.7	72.1	76.1	76.8
	OOD	96.9	61.8	72.0	76.0	76.7

Table B.3: Domain generalization results on VLCS

Algorithm		Art	Cartoon	Photo	Sketch	Avg
w/ Domain Labels						
IRM [5]		84.8	76.4	96.7	76.1	83.5
GroupDRO [169]		83.5	79.1	96.7	78.3	84.4
Mixup [225]		86.1	78.9	<u>97.6</u>	75.8	84.6
MLDG [110]		85.5	80.1	97.4	76.6	84.9
CORAL [194]		88.3	80.0	97.5	78.8	86.2
MMD [113]		86.1	79.4	96.6	76.5	84.6
DANN [59]		86.4	77.4	97.3	73.5	83.6
CDANN [116]		84.6	75.5	96.8	73.5	82.6
MTL [16]		87.5	77.1	96.4	77.3	84.6
ARM [231]		86.8	76.8	97.4	79.3	85.1
VREx [102]		86.0	79.1	96.9	77.7	84.9
HMOE-DL	MIX	84.1	77.3	96.3	76.4	83.5
	MAX	82.9	78.6	95.9	78.1	83.9
	OOD	85.0	78.3	95.3	79.4	84.5
w/o Domain Labels						
ERM [204]		84.7	80.8	97.2	79.3	85.5
RSC [84]		85.4	79.7	<u>97.6</u>	78.2	85.2
SagNet [143]		87.4	80.7	97.1	80.0	86.3
HMOE-DN	MIX	83.9	82.3	95.0	77.9	84.8
	MAX	84.7	82.4	96.4	76.9	85.1
	OOD	83.9	80.2	95.6	<u>80.1</u>	84.9
HMOE-ND	MIX	88.8	78.9	96.6	73.9	84.5
	MAX	88.8	82.7	95.7	79.1	86.6
	OOD	<u>88.9</u>	<u>84.3</u>	95.7	79.0	<u>87.0</u>

Table B.4: Domain generalization results on PACS

Algorithm		Art	Clipart	Product	Real	Avg
w/ Domain Labels						
IRM [5]		58.9	52.2	72.1	74.0	64.3
GroupDRO [169]		60.4	52.7	75.0	76.0	66.0
Mixup [225]		62.4	<u>54.8</u>	<u>76.9</u>	78.3	68.1
MLDG [110]		61.5	53.2	75.0	77.5	66.8
CORAL [194]		<u>65.3</u>	54.4	76.5	<u>78.4</u>	<u>68.7</u>
MMD [113]		60.4	53.3	74.3	77.4	66.3
DANN [59]		59.9	53.0	73.6	76.9	65.9
CDANN [116]		61.5	50.4	74.4	76.6	65.8
MTL [16]		61.5	52.4	74.9	76.8	66.4
ARM [231]		58.9	51.0	74.1	75.2	64.8
VREx [102]		60.7	53.0	75.3	76.6	66.4
HMOE-DL	MIX	59.5	50.5	73.6	75.2	64.7
	MAX	58.5	47.7	72.5	74.1	63.2
	OOD	58.6	49.9	72.8	73.7	63.7
w/o Domain Labels						
ERM [204]		61.3	52.4	75.8	76.6	66.5
RSC [84]		60.7	51.4	74.8	75.1	65.5
SagNet [143]		63.4	<u>54.8</u>	75.8	78.3	68.1
HMOE-DN	MIX	59.4	52.9	74.6	74.7	65.4
	MAX	60.0	52.1	74.6	74.9	65.4
	OOD	60.2	52.5	73.6	74.7	65.3
HMOE-ND	MIX	60.0	52.4	74.3	75.1	65.5
	MAX	60.0	52.4	73.3	76.3	65.5
	OOD	60.0	54.1	72.8	75.6	65.6

Table B.5: Domain generalization results on OfficeHome

Algorithm		L100	L38	L43	L46	Avg
w/ Domain Labels						
IRM [5]		54.6	39.8	56.2	39.6	47.6
GroupDRO [169]		41.2	38.6	56.7	36.4	43.2
Mixup [225]		<u>59.6</u>	42.2	55.9	33.9	47.9
MLDG [110]		54.2	44.3	55.6	36.9	47.7
CORAL [194]		51.6	42.2	57.0	39.8	47.6
MMD [113]		41.9	34.8	57.0	35.2	42.2
DANN [59]		51.1	40.6	57.4	37.7	46.7
CDANN [116]		47.0	41.3	54.9	39.8	45.8
MTL [16]		49.3	39.6	55.6	37.8	45.6
ARM [231]		49.3	38.3	55.8	38.7	45.5
VREx [102]		48.2	41.7	56.8	38.7	46.4
HMOE-DL	MIX	43.1	44.9	55.8	36.1	45.0
	MAX	42.2	38.1	55.0	37.4	43.2
	OOD	43.0	42.2	54.6	36.3	44.0
w/o Domain Labels						
ERM [204]		49.8	42.1	56.9	35.7	46.1
RSC [84]		50.2	39.2	56.3	<u>40.8</u>	46.6
SagNet [143]		53.0	43.0	57.9	40.4	48.6
HMOE-DN	MIX	54.0	43.6	<u>58.3</u>	38.8	48.7
	MAX	54.0	<u>47.0</u>	<u>58.3</u>	38.8	<u>49.5</u>
	OOD	54.1	42.1	<u>58.3</u>	39.0	48.4
HMOE-ND	MIX	58.8	41.8	56.1	36.8	48.4
	MAX	44.7	41.8	56.5	36.8	45.0
	OOD	53.4	41.8	55.6	37.5	47.1

Table B.6: Domain generalization results on TerraIncognita

BIBLIOGRAPHY

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. “Tensorflow: A System for Large-Scale Machine Learning.” In: 12th $\{\$USENIX\}$ Symposium on Operating Systems Design and Implementation ($\{\$OSDI\}$ 16). 2016, pp. 265–283.
- [2] Kartik Ahuja, Ethan Caballero, Dinghuai Zhang, Jean-Christophe Gagnon-Audet, Yoshua Bengio, Ioannis Mitliagkas, and Irina Rish. “Invariance Principle Meets Information Bottleneck for Out-of-Distribution Generalization.” In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 3438–3450.
- [3] Jan-Olof Andersson, Thomas Helander, Lars Höglund, Pingfang Shi, and Bo Sundman. “Thermo-Calc & DICTRA, Computational Tools for Materials Science.” In: *Calphad* 26.2 (2002), pp. 273–312.
- [4] Bertan Ari and H. Altay Güvenir. “Clustered Linear Regression.” In: *Knowledge-Based Systems* 15.3 (2002), pp. 169–175.
- [5] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. “Invariant Risk Minimization.” In: arXiv preprint arXiv:1907.02893 (2019). arXiv: [1907.02893](https://arxiv.org/abs/1907.02893).
- [6] Mariette Awad, Rahul Khanna, Mariette Awad, and Rahul Khanna. “Support Vector Regression.” In: *Efficient learning machines: Theories, concepts, and applications for engineers and system designers* (2015), pp. 67–80.
- [7] Yogesh Balaji, Swami Sankaranarayanan, and Rama Chellappa. “Metareg: Towards Domain Generalization Using Meta-Regularization.” In: *Advances in neural information processing systems* 31 (2018).
- [8] David M. Beazley. “SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++.” In: *Tcl/Tk Workshop*. Vol. 43. 1996, p. 74.
- [9] Sara Beery, Grant Van Horn, and Pietro Perona. “Recognition in Terra Incognita.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 456–473.
- [10] Abdelkrim Belkadi, Wei Yan, Michael L Michelsen, and Erling H Stenby. “Comparison of Two Methods for Speeding up Flash Calculations in Compositional Simulations.” In: *SPE Reservoir Simulation Symposium*. OnePetro, 2011.
- [11] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. “A Theory of Learning from Different Domains.” In: *Machine learning* 79.1 (2010), pp. 151–175.
- [12] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. “Algorithms for Hyper-Parameter Optimization.” In: *Advances in neural information processing systems* 24 (2011).
- [13] James Bergstra and Yoshua Bengio. “Random Search for Hyper-Parameter Optimization.” In: *Journal of machine learning research* 13.2 (2012).

- [14] James Bergstra, Daniel Yamins, and David Cox. “Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures.” In: *International Conference on Machine Learning*. PMLR, 2013, pp. 115–123.
- [15] Deniz A. Bezgin, Aaron B. Buhendwa, and Nikolaus A. Adams. “JAX-Fluids: A Fully-Differentiable High-Order Computational Fluid Dynamics Solver for Compressible Two-Phase Flows.” In: *Computer Physics Communications* 282 (2023), p. 108527.
- [16] Gilles Blanchard, Aniket Anand Deshmukh, Ürun Dogan, Gyemin Lee, and Clayton Scott. “Domain Generalization by Marginal Transfer Learning.” In: *The Journal of Machine Learning Research* 22.1 (2021), pp. 46–100.
- [17] Felipe J. Blas and Lourdes F. Vega. “Prediction of Binary and Ternary Diagrams Using the Statistical Associating Fluid Theory (SAFT) Equation of State.” In: *Industrial & engineering chemistry research* 37.2 (1998), pp. 660–674.
- [18] Avrim L. Blum and Ronald L. Rivest. “Training a 3-Node Neural Network Is NP-complete.” In: *Neural Networks* 5.1 (1992), pp. 117–127.
- [19] James Bradbury et al. *JAX: Composable Transformations of Python+NumPy Programs*. 2018.
- [20] Dhanajit Brahma, Vinay Kumar Verma, and Piyush Rai. “Hypernetworks for Continual Semi-Supervised Learning.” In: *arXiv preprint arXiv:2110.01856* (2021). [arXiv: 2110.01856](https://arxiv.org/abs/2110.01856).
- [21] Fabio M. Carlucci, Antonio D’Innocente, Silvia Bucci, Barbara Caputo, and Tatiana Tommasi. “Domain Generalization by Solving Jigsaw Puzzles.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2229–2238.
- [22] Rich Caruana. “Multitask Learning.” In: *Machine learning* 28.1 (1997), pp. 41–75.
- [23] Oscar Chang, Lampros Flokas, and Hod Lipson. “Principled Weight Initialization for Hypernetworks.” In: *International Conference on Learning Representations*. 2019.
- [24] Walter G Chapman, Keith E Gubbins, George Jackson, and Maciej Radosz. “New Reference Equation of State for Associating Liquids.” In: *Industrial & engineering chemistry research* 29.8 (1990), pp. 1709–1721.
- [25] Chaoqi Chen, Jiongcheng Li, Xiaoguang Han, Xiaoqing Liu, and Yizhou Yu. “Compound Domain Generalization via Meta-Knowledge Encoding.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 7119–7129.
- [26] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. “Neural Ordinary Differential Equations.” In: *Advances in neural information processing systems* 31 (2018).
- [27] Zhangxin Chen, Hui Liu, Song Yu, Ben Hsieh, and Lei Shao. “GPU-based Parallel Reservoir Simulators.” In: *Domain Decomposition Methods in Science and Engineering XXI*. Springer, 2014, pp. 199–206.

- [28] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishikesh Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. "Wide & Deep Learning for Recommender Systems." In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. 2016, pp. 7–10.
- [29] Wen-Hsi Cheng, Ming-Shean Chou, Chih-Hao Perng, and Fu-Sui Chu. "Determining the Equilibrium Partitioning Coefficients of Volatile Organic Compounds at an Air–Water Interface." In: *Chemosphere* 54.7 (2004), pp. 935–942.
- [30] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks." In: *Machine learning* 20.3 (1995), pp. 273–297.
- [31] George Cybenko. "Approximation by Superpositions of a Sigmoidal Function." In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.
- [32] Damai Dai, Li Dong, Shuming Ma, Bo Zheng, Zhifang Sui, Baobao Chang, and Furu Wei. "StableMoE: Stable Routing Strategy for Mixture of Experts." In: *arXiv preprint arXiv:2204.08396* (2022). arXiv: [2204.08396](https://arxiv.org/abs/2204.08396).
- [33] Laurence Patrick Dake. *Fundamentals of Reservoir Engineering*. Elsevier, 1983. ISBN: 0-08-056898-X.
- [34] Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. "Identifying and Attacking the Saddle Point Problem in High-Dimensional Non-Convex Optimization." In: *Advances in neural information processing systems* 27 (2014).
- [35] Jean-Charles De Hemptinne and Jean-Marie Ledanois. *Select Thermodynamic Models for Process Simulation: A Practical Guide Using a Three Steps Methodology*. Editions Technip, 2012. ISBN: 2-7108-0949-4.
- [36] Wayne S. DeSarbo and William L. Cron. "A Maximum Likelihood Methodology for Clusterwise Linear Regression." In: *Journal of classification* 5.2 (1988), pp. 249–282.
- [37] Ulrich K Deiters and Ricardo Macías-Salinas. "Calculation of Densities from Cubic Equations of State: Revisited." In: *Industrial & Engineering Chemistry Research* 53.6 (2014), pp. 2529–2536.
- [38] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A Large-Scale Hierarchical Image Database." In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Ieee, 2009, pp. 248–255. ISBN: 1-4244-3992-2.
- [39] Aniket Anand Deshmukh, Yunwen Lei, Srinagesh Sharma, Urun Dogan, James W. Cutler, and Clayton Scott. "A Generalization Error Bound for Multi-Class Domain Generalization." In: *arXiv preprint arXiv:1905.10392* (2019). arXiv: [1905.10392](https://arxiv.org/abs/1905.10392).
- [40] Ali H Dogru, Larry Siu Kuen Fung, Usuf Middy, Tareq Al-Shaalan, and Jorge Alberto Pita. "A Next-Generation Parallel Reservoir Simulator for Giant Reservoirs." In: *SPE Reservoir Simulation Symposium*. OnePetro, 2009.
- [41] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xi-aohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, and Sylvain Gelly. "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale." In: *arXiv preprint arXiv:2010.11929* (2020). arXiv: [2010.11929](https://arxiv.org/abs/2010.11929).

- [42] Qi Dou, Daniel Coelho de Castro, Konstantinos Kamnitsas, and Ben Glocker. “Domain Generalization via Model-Agnostic Learning of Semantic Features.” In: *Advances in Neural Information Processing Systems* 32 (2019).
- [43] Kevin Dowd and Charles Severance. “High Performance Computing.” In: (2010).
- [44] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, and Orhan Firat. “Glam: Efficient Scaling of Language Models with Mixture-of-Experts.” In: *International Conference on Machine Learning*. PMLR, 2022, pp. 5547–5569. ISBN: 2640-3498.
- [45] Stefan Elfving, Eiji Uchibe, and Kenji Doya. “Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning.” In: *Neural Networks* 107 (2018), pp. 3–11.
- [46] Chen Fang, Ye Xu, and Daniel N. Rockmore. “Unbiased Metric Learning: On the Utilization of Multiple Datasets and Web Images for Softening Bias.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1657–1664.
- [47] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatin, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. “Discovering Faster Matrix Multiplication Algorithms with Reinforcement Learning.” In: *Nature* 610.7930 (2022), pp. 47–53.
- [48] William Fedus, Jeff Dean, and Barret Zoph. “A Review of Sparse Expert Models in Deep Learning.” In: *arXiv preprint arXiv:2209.01667* (2022). arXiv: [2209.01667](https://arxiv.org/abs/2209.01667).
- [49] William Fedus, Barret Zoph, and Noam Shazeer. *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity*. 2021.
- [50] Joel H. Ferziger, Milovan Perić, and Robert L. Street. *Computational Methods for Fluid Dynamics*. Vol. 3. Springer, 2002. ISBN: 3-540-42074-6.
- [51] Yuan-Hao Fu and Stanley I. Sandler. “A Simplified SAFT Equation of State for Associating Compounds and Mixtures.” In: *Industrial & engineering chemistry research* 34.5 (1995), pp. 1897–1909.
- [52] Vassilis Gaganis. “Rapid Phase Stability Calculations in Fluid Flow Simulation Using Simple Discriminating Functions.” In: *Computers & Chemical Engineering* 108 (2018), pp. 112–127.
- [53] Vassilis Gaganis and Nikos Varotsis. “Machine Learning Methods to Speed up Compositional Reservoir Simulation.” In: *SPE Europec/EAGE Annual Conference*. OnePetro, 2012.
- [54] Vassilis Gaganis and Nikos Varotsis. “An Integrated Approach for Rapid Phase Behavior Calculations in Compositional Modeling.” In: *Journal of Petroleum Science and Engineering* 118 (2014), pp. 74–87.
- [55] Yarín Gal and Zoubin Ghahramani. “A Theoretically Grounded Application of Dropout in Recurrent Neural Networks.” In: *Advances in neural information processing systems* 29 (2016), pp. 1019–1027.
- [56] Yarín Gal and Zoubin Ghahramani. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning.” In: *International Conference on Machine Learning*. PMLR, 2016, pp. 1050–1059.

- [57] Chuang Gan, Tianbao Yang, and Boqing Gong. “Learning Attributes Equals Multi-Source Domain Generalization.” In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016, pp. 87–97.
- [58] Yaroslav Ganin and Victor Lempitsky. “Unsupervised Domain Adaptation by Back-propagation.” In: International Conference on Machine Learning. PMLR, 2015, pp. 1180–1189.
- [59] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. “Domain-Adversarial Training of Neural Networks.” In: The journal of machine learning research 17.1 (2016), pp. 2096–2030. ISSN: 1532-4435.
- [60] Muhammad Ghifary, David Balduzzi, W. Bastiaan Kleijn, and Mengjie Zhang. “Scatter Component Analysis: A Unified Framework for Domain Adaptation and Domain Generalization.” In: IEEE transactions on pattern analysis and machine intelligence 39.7 (2016), pp. 1414–1430.
- [61] Muhammad Ghifary, W. Bastiaan Kleijn, Mengjie Zhang, and David Balduzzi. “Domain Generalization for Object Recognition with Multi-Task Autoencoders.” In: Proceedings of the IEEE International Conference on Computer Vision. 2015, pp. 2551–2559.
- [62] Alejandro Gil-Villegas, Amparo Galindo, Paul J. Whitehead, Stuart J. Mills, George Jackson, and Andrew N. Burgess. “Statistical Associating Fluid Theory for Chain Molecules with Attractive Potentials of Variable Range.” In: The Journal of chemical physics 106.10 (1997), pp. 4168–4186.
- [63] Xavier Glorot and Yoshua Bengio. “Understanding the Difficulty of Training Deep Feedforward Neural Networks.” In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
- [64] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks.” In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.
- [65] Rui Gong, Wen Li, Yuhua Chen, and Luc Van Gool. “Dlow: Domain Flow for Adaptation and Generalization.” In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019, pp. 2477–2486.
- [66] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT press, 2016.
- [67] Joachim Gross and Gabriele Sadowski. “Perturbed-Chain SAFT: An Equation of State Based on a Perturbation Theory for Chain Molecules.” In: Industrial & engineering chemistry research 40.4 (2001), pp. 1244–1260.
- [68] Ishaan Gulrajani and David Lopez-Paz. “In Search of Lost Domain Generalization.” In: arXiv preprint arXiv:2007.01434 (2020). arXiv: [2007.01434](https://arxiv.org/abs/2007.01434).
- [69] Jiang Guo, Darsh J. Shah, and Regina Barzilay. “Multi-Source Domain Adaptation with Mixture of Experts.” In: arXiv preprint arXiv:1809.02256 (2018). arXiv: [1809.02256](https://arxiv.org/abs/1809.02256).

- [70] David Ha, Andrew Dai, and Quoc V. Le. “Hypernetworks.” In: arXiv preprint arXiv:1609.09106 (2016). arXiv: [1609.09106](https://arxiv.org/abs/1609.09106).
- [71] Dion Häfner, René Löwe Jacobsen, Carsten Eden, Mads RB Kristensen, Markus Jochum, Roman Nuterman, and Brian Vinter. “Veros v0. 1—A Fast and Versatile Ocean Simulator in Pure Python.” In: *Geoscientific Model Development* 11.8 (2018), pp. 3299–3312.
- [72] Hussein Hazimeh, Zhe Zhao, Aakanksha Chowdhery, Maheswaran Sathiamoorthy, Yihua Chen, Rahul Mazumder, Lichan Hong, and Ed Chi. “Dselect-k: Differentiable Selection in the Mixture of Experts with Applications to Multi-Task Learning.” In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 29335–29347.
- [73] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1026–1034.
- [74] 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*. 2016, pp. 770–778.
- [75] MD Hebden. “An Algorithm for Minimization Using Exact Second Derivatives.” In: (1973).
- [76] Eric M Hendriks. “Reduction Theorem for Phase Equilibrium Problems.” In: *Industrial & engineering chemistry research* 27.9 (1988), pp. 1728–1732.
- [77] Eric M Hendriks and ARD Van Bergen. “Application of a Reduction Method to Phase Equilibria Calculations.” In: *Fluid Phase Equilibria* 74 (1992), pp. 17–34.
- [78] Dan Hendrycks and Kevin Gimpel. “Gaussian Error Linear Units (Gelus).” In: arXiv preprint arXiv:1606.08415 (2016). arXiv: [1606.08415](https://arxiv.org/abs/1606.08415).
- [79] Philipp Holl, Vladlen Koltun, Kiwon Um, and Nils Thuerey. “Phiflow: A Differentiable Pde Solving Framework for Deep Learning via Physical Simulations.” In: *NeurIPS Workshop*. Vol. 2. 2020.
- [80] Richard Zou Horace He. *Functorch: JAX-like Composable Function Transforms for PyTorch*. 2021.
- [81] Kurt Hornik. “Approximation Capabilities of Multilayer Feedforward Networks.” In: *Neural networks* 4.2 (1991), pp. 251–257.
- [82] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. “Meta-Learning in Neural Networks: A Survey.” In: *IEEE transactions on pattern analysis and machine intelligence* 44.9 (2021), pp. 5149–5169.
- [83] Stanley H Huang and Maciej Radosz. “Equation of State for Small, Large, Polydisperse, and Associating Molecules.” In: *Industrial & Engineering Chemistry Research* 29.11 (1990), pp. 2284–2294.
- [84] Zeyi Huang, Haohan Wang, Eric P. Xing, and Dong Huang. “Self-Challenging Improves Cross-Domain Generalization.” In: *European Conference on Computer Vision*. Springer, 2020, pp. 124–140.
- [85] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. “Sequential Model-Based Optimization for General Algorithm Configuration.” In: *International Conference on Learning and Intelligent Optimization*. Springer, 2011, pp. 507–523.

- [86] Maximilian Ilse, Jakub M Tomczak, Christos Louizos, and Max Welling. “Diva: Domain Invariant Variational Autoencoders.” In: *Medical Imaging with Deep Learning*. PMLR, 2020, pp. 322–348. ISBN: 2640-3498.
- [87] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” In: *International Conference on Machine Learning*. PMLR, 2015, pp. 448–456.
- [88] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. “Adaptive Mixtures of Local Experts.” In: *Neural computation* 3.1 (1991), pp. 79–87.
- [89] Michael I. Jordan and Robert A. Jacobs. “Hierarchical Mixtures of Experts and the EM Algorithm.” In: *Neural computation* 6.2 (1994), pp. 181–214.
- [90] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. “Highly Accurate Protein Structure Prediction with AlphaFold.” In: *Nature* 596.7873 (2021), pp. 583–589.
- [91] A. Kashinath, M.L. Szulczewski, and A.H. Dogru. “A Fast Algorithm for Calculating Isothermal Phase Behavior Using Machine Learning.” In: *Fluid Phase Equilibria* 465 (June 2018), pp. 73–82. ISSN: 03783812. DOI: [10.1016/j.fluid.2018.02.004](https://doi.org/10.1016/j.fluid.2018.02.004). (Visited on 11/25/2021).
- [92] Ilyes Khemakhem, Diederik Kingma, Ricardo Monti, and Aapo Hyvarinen. “Variational Autoencoders and Nonlinear Ica: A Unifying Framework.” In: *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2207–2217. ISBN: 2640-3498.
- [93] Aditya Khosla, Tinghui Zhou, Tomasz Malisiewicz, Alexei A. Efros, and Antonio Torralba. “Undoing the Damage of Dataset Bias.” In: *European Conference on Computer Vision*. Springer, 2012, pp. 158–171.
- [94] Daehee Kim, Youngjun Yoo, Seunghyun Park, Jinkyu Kim, and Jaekoo Lee. “Self-reg: Self-supervised Contrastive Regularization for Domain Generalization.” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 9619–9628.
- [95] Diederik P Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *arXiv preprint arXiv:1412.6980* (2014). arXiv: [1412.6980](https://arxiv.org/abs/1412.6980).
- [96] Diederik P Kingma and Max Welling. “Auto-Encoding Variational Bayes.” In: *arXiv preprint arXiv:1312.6114* (2013). arXiv: [1312.6114](https://arxiv.org/abs/1312.6114).
- [97] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. “Machine Learning–Accelerated Computational Fluid Dynamics.” In: *Proceedings of the National Academy of Sciences* 118.21 (2021), e2101784118.
- [98] Georgios M Kontogeorgis and Georgios K Folas. *Thermodynamic Models for Industrial Applications: From Classical and Advanced Mixing Rules to Association Theories*. John Wiley & Sons, 2009.
- [99] Georgios M Kontogeorgis, Epaminondas C Voutsas, Iakovos V Yakoumis, and Dimitrios P Tassios. “An Equation of State for Associating Fluids.” In: *Industrial & engineering chemistry research* 35.11 (1996), pp. 4310–4318.

- [100] Georgios M Kontogeorgis, Iakovos V Yakoumis, Henk Meijer, Eric Hendriks, and Tony Moorwood. “Multicomponent Phase Equilibrium Calculations for Water–Methanol–Alkane Mixtures.” In: *Fluid Phase Equilibria* 158 (1999), pp. 201–209.
- [101] Steven George Krantz and Harold R Parks. *The Implicit Function Theorem: History, Theory, and Applications*. Springer Science & Business Media, 2002.
- [102] David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Remi Le Priol, and Aaron Courville. “Out-of-Distribution Generalization via Risk Extrapolation (Rex).” In: *International Conference on Machine Learning*. PMLR, 2021, pp. 5815–5826. ISBN: 2640-3498.
- [103] Thomas Lafitte, Anastasia Apostolakou, Carlos Avendaño, Amparo Galindo, Claire S. Adjiman, Erich A. Müller, and George Jackson. “Accurate Statistical Associating Fluid Theory for Chain Molecules Formed from Mie Segments.” In: *The Journal of chemical physics* 139.15 (2013), p. 154504.
- [104] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning.” In: *nature* 521.7553 (2015), pp. 436–444.
- [105] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. “Backpropagation Applied to Handwritten Zip Code Recognition.” In: *Neural computation* 1.4 (1989), pp. 541–551.
- [106] ClaudeF Leibovici and Jean Neoschil. “A New Look at the Rachford-Rice Equation.” In: *Fluid Phase Equilibria* 74 (July 1992), pp. 303–308. ISSN: 03783812. DOI: [10.1016/0378-3812\(92\)85069-K](https://doi.org/10.1016/0378-3812(92)85069-K). (Visited on 12/09/2021).
- [107] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. “Gshard: Scaling Giant Models with Conditional Computation and Automatic Sharding.” In: *arXiv preprint arXiv:2006.16668* (2020). arXiv: [2006.16668](https://arxiv.org/abs/2006.16668).
- [108] Bo Li, Jingkang Yang, Jiawei Ren, Yezhen Wang, and Ziwei Liu. “Sparse Fusion Mixture-of-Experts Are Domain Generalizable Learners.” In: *arXiv preprint arXiv:2206.04046* (2022). arXiv: [2206.04046](https://arxiv.org/abs/2206.04046).
- [109] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M. Hospedales. “Deeper, Broader and Artier Domain Generalization.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 5542–5550.
- [110] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. “Learning to Generalize: Meta-learning for Domain Generalization.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 2018. ISBN: 2374-3468.
- [111] Da Li, Jianshu Zhang, Yongxin Yang, Cong Liu, Yi-Zhe Song, and Timothy M. Hospedales. “Episodic Training for Domain Generalization.” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1446–1455.
- [112] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. “Visualizing the Loss Landscape of Neural Nets.” In: *arXiv preprint arXiv:1712.09913* (2017). arXiv: [1712.09913](https://arxiv.org/abs/1712.09913).

- [113] Haoliang Li, Sinno Jialin Pan, Shiqi Wang, and Alex C Kot. “Domain Generalization with Adversarial Feature Learning.” In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, pp. 5400–5409.
- [114] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. “A System for Massively Parallel Hyperparameter Tuning.” In: Proceedings of Machine Learning and Systems 2 (2020), pp. 230–246.
- [115] Pan Li, Da Li, Wei Li, Shaogang Gong, Yanwei Fu, and Timothy M. Hospedales. “A Simple Feature Augmentation for Domain Generalization.” In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021, pp. 8886–8895.
- [116] Ya Li, Xinmei Tian, Mingming Gong, Yajing Liu, Tongliang Liu, Kun Zhang, and Dacheng Tao. “Deep Domain Generalization via Conditional Invariant Adversarial Networks.” In: Proceedings of the European Conference on Computer Vision (ECCV). 2018, pp. 624–639.
- [117] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. “Fourier Neural Operator for Parametric Partial Differential Equations.” In: arXiv preprint arXiv:2010.08895 (2020). arXiv: [2010.08895](https://arxiv.org/abs/2010.08895).
- [118] Aristidis Likas, Nikos Vlassis, and Jakob J. Verbeek. “The Global K-Means Clustering Algorithm.” In: Pattern recognition 36.2 (2003), pp. 451–461.
- [119] Xi Lin, Zhiyuan Yang, Qingfu Zhang, and Sam Kwong. “Controllable Pareto Multi-Task Learning.” In: arXiv preprint arXiv:2010.06313 (2020). arXiv: [2010.06313](https://arxiv.org/abs/2010.06313).
- [120] Alexander H. Liu, Yen-Cheng Liu, Yu-Ying Yeh, and Yu-Chiang Frank Wang. “A Unified Feature Disentangler for Multi-Domain Image Translation and Manipulation.” In: Advances in neural information processing systems 31 (2018).
- [121] Yuanbin Liu, Weixiang Hong, and Bingyang Cao. “Machine Learning for Predicting Thermodynamic Properties of Pure Fluids and Their Mixtures.” In: Energy 188 (2019), p. 116091.
- [122] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. “Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows.” In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021, pp. 10012–10022.
- [123] Chris Lomont. “Introduction to Intel Advanced Vector Extensions.” In: Intel white paper 23 (2011).
- [124] Jonathan Lorraine and David Duvenaud. “Stochastic Hyperparameter Optimization through Hypernetworks.” In: arXiv preprint arXiv:1802.09419 (2018). arXiv: [1802.09419](https://arxiv.org/abs/1802.09419).
- [125] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. “Learning Nonlinear Operators via DeepONet Based on the Universal Approximation Theorem of Operators.” In: Nature Machine Intelligence 3.3 (2021), pp. 218–229.
- [126] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. “Parameter-Efficient Multi-Task Fine-Tuning for Transformers via Shared Hypernetworks.” In: arXiv preprint arXiv:2106.04489 (2021). arXiv: [2106.04489](https://arxiv.org/abs/2106.04489).

- [127] Toshihiko Matsuura and Tatsuya Harada. “Domain Generalization Using a Mixture of Multiple Latent Domains.” In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 34. 2020, pp. 11749–11756. ISBN: 2374-3468.
- [128] Nicolai Meinshausen. “Causality from a Distributional Robustness Point of View.” In: 2018 IEEE Data Science Workshop (DSW). IEEE, 2018, pp. 6–10. ISBN: 1-5386-4410-X.
- [129] Michael L Michelsen. “The Isothermal Flash Problem. Part I. Stability.” In: Fluid phase equilibria 9.1 (1982), pp. 1–19.
- [130] Michael L Michelsen. “The Isothermal Flash Problem. Part II. Phase-split Calculation.” In: Fluid phase equilibria 9.1 (1982), pp. 21–40.
- [131] Michael L Michelsen. “Simplified Flash Calculations for Cubic Equations of State.” In: Industrial & Engineering Chemistry Process Design and Development 25.1 (1986), pp. 184–188.
- [132] Michael L. Michelsen. “Multiphase Isenthalpic and Isentropic Flash Algorithms.” In: Fluid phase equilibria 33.1-2 (1987), pp. 13–27.
- [133] Michael L Michelsen. “Phase Equilibrium Calculations. What Is Easy and What Is Difficult?” In: Computers & chemical engineering 17.5-6 (1993), pp. 431–439.
- [134] Michael L. Michelsen. “Robust and Efficient Solution Procedures for Association Models.” In: Industrial & engineering chemistry research 45.25 (2006), pp. 8449–8453.
- [135] Michael L Michelsen, Wei Yan, and Erling H Stenby. “A Comparative Study of Reduced-Variables-Based Flash and Conventional Flash.” In: SPE Journal 18.05 (2013), pp. 952–959.
- [136] Michael Loch Michelsen and Jørgen Møllerup. Thermodynamic Modelling: Fundamentals and Computational Aspects. Tie-Line Publications, 2004.
- [137] Tom M. Mitchell and Tom M. Mitchell. Machine Learning. Vol. 1. McGraw-hill New York, 1997.
- [138] Alan Morningstar, Markus Hauru, Jackson Beall, Martin Ganahl, Adam GM Lewis, Vedika Khemani, and Guifre Vidal. “Simulation of Quantum Many-Body Dynamics with Tensor Processing Units: Floquet Prethermalization.” In: PRX Quantum 3.2 (2022), p. 020331.
- [139] Saeid Motiian, Marco Piccirilli, Donald A. Adjeroh, and Gianfranco Doretto. “Unified Deep Supervised Domain Adaptation and Generalization.” In: Proceedings of the IEEE International Conference on Computer Vision. 2017, pp. 5715–5725.
- [140] Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. “Domain Generalization via Invariant Feature Representation.” In: International Conference on Machine Learning. PMLR, 2013, pp. 10–18.
- [141] Fionn Murtagh and Pierre Legendre. “Ward’s Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward’s Criterion?” In: Journal of classification 31.3 (2014), pp. 274–295.
- [142] Vinod Nair and Geoffrey E Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines.” In: Icml. 2010.

- [143] Hyeonseob Nam, HyunJae Lee, Jongchan Park, Wonjun Yoon, and Donggeun Yoo. “Reducing Domain Gap by Reducing Style Bias.” In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021, pp. 8690–8699.
- [144] Aviv Navon, Aviv Shamsian, Gal Chechik, and Ethan Fetaya. “Learning the Pareto Front with Hypernetworks.” In: arXiv preprint arXiv:2010.04104 (2020). arXiv: [2010.04104](https://arxiv.org/abs/2010.04104).
- [145] O Orbach and CM Crowe. “Convergence Promotion in the Simulation of Chemical Processes with Recycle-the Dominant Eigenvalue Method.” In: The Canadian Journal of Chemical Engineering 49.4 (1971), pp. 509–513.
- [146] Nathan Otterness and James H. Anderson. “AMD GPUs as an Alternative to NVIDIA for Supporting Real-Time Workloads.” In: 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- [147] Sinno Jialin Pan, Ivor W. Tsang, James T. Kwok, and Qiang Yang. “Domain Adaptation via Transfer Component Analysis.” In: IEEE transactions on neural networks 22.2 (2010), pp. 199–210.
- [148] Sinno Jialin Pan and Qiang Yang. “A Survey on Transfer Learning.” In: IEEE Transactions on knowledge and data engineering 22.10 (2010), pp. 1345–1359.
- [149] Young Woong Park, Yan Jiang, Diego Klabjan, and Loren Williams. “Algorithms for Generalized Clusterwise Linear Regression.” In: INFORMS Journal on Computing 29.2 (2017), pp. 301–317.
- [150] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. “Pytorch: An Imperative Style, High-Performance Deep Learning Library.” In: Advances in neural information processing systems 32 (2019), pp. 8026–8037.
- [151] Vishal M. Patel, Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. “Visual Domain Adaptation: A Survey of Recent Advances.” In: IEEE signal processing magazine 32.3 (2015), pp. 53–69.
- [152] Svetlana Pavlitskaya, Christian Hubschneider, Lukas Struppek, and J. Marius Zöllner. “Balancing Expert Utilization in Mixture-of-Experts Layers Embedded in CNNs.” In: arXiv preprint arXiv:2204.10598 (2022). arXiv: [2204.10598](https://arxiv.org/abs/2204.10598).
- [153] Ding-Yu Peng and Donald B Robinson. “A New Two-Constant Equation of State.” In: Industrial & Engineering Chemistry Fundamentals 15.1 (1976), pp. 59–64.
- [154] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. “Moment Matching for Multi-Source Domain Adaptation.” In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019, pp. 1406–1415.
- [155] Xingchao Peng, Zijun Huang, Ximeng Sun, and Kate Saenko. “Domain Agnostic Learning with Disentangled Representations.” In: International Conference on Machine Learning. PMLR, 2019, pp. 5102–5112. ISBN: 2640-3498.
- [156] Steve Plimpton. “Fast Parallel Algorithms for Short-Range Molecular Dynamics.” In: Journal of computational physics 117.1 (1995), pp. 1–19.
- [157] Lutz Prechelt. “Early Stopping-but When?” In: Neural Networks: Tricks of the Trade. Springer, 1998, pp. 55–69.

- [158] Fengchun Qiao, Long Zhao, and Xi Peng. “Learning to Learn Single Domain Generalization.” In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020, pp. 12556–12565.
- [159] Joaquin Quinonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. *Dataset Shift in Machine Learning*. Mit Press, 2008. ISBN: 0-262-17005-1.
- [160] Henry H Rachford and JD Rice. “Procedure for Use of Electronic Digital Computers in Calculating Flash Vaporization Hydrocarbon Equilibrium.” In: *Journal of Petroleum Technology* 4.10 (1952), pp. 19–3.
- [161] 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.” In: *Journal of Computational physics* 378 (2019), pp. 686–707.
- [162] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Physics Informed Deep Learning (Part i): Data-driven Solutions of Nonlinear Partial Differential Equations.” In: arXiv preprint arXiv:1711.10561 (2017). arXiv: [1711.10561](https://arxiv.org/abs/1711.10561).
- [163] Prajit Ramachandran, Barret Zoph, and Quoc V Le. “Searching for Activation Functions.” In: arXiv preprint arXiv:1710.05941 (2017). arXiv: [1710.05941](https://arxiv.org/abs/1710.05941).
- [164] Prajit Ramachandran, Barret Zoph, and Quoc V Le. “Swish: A Self-Gated Activation Function.” In: arXiv preprint arXiv:1710.05941 7 (2017), p. 1. arXiv: [1710.05941](https://arxiv.org/abs/1710.05941).
- [165] Alexandre Rame, Corentin Dancette, and Matthieu Cord. “Fishr: Invariant Gradient Variances for out-of-Distribution Generalization.” In: *International Conference on Machine Learning*. PMLR, 2022, pp. 18347–18377. ISBN: 2640-3498.
- [166] Claus P Rasmussen, Kristian Krejbjerg, Michael L Michelsen, and Kersti E Bjurstrøm. “Increasing the Computational Speed of Flash Calculations with Applications for Compositional, Transient Simulations.” In: *SPE Reservoir Evaluation & Engineering* 9.01 (2006), pp. 32–38.
- [167] Douglas A. Reynolds. “Gaussian Mixture Models.” In: *Encyclopedia of biometrics* 741.659-663 (2009).
- [168] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning Representations by Back-Propagating Errors.” In: *nature* 323.6088 (1986), pp. 533–536.
- [169] Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. *Distributionally Robust Neural Networks for Group Shifts: On the Importance of Regularization for Worst-Case Generalization*. Apr. 2020. DOI: [10.48550/arXiv.1911.08731](https://doi.org/10.48550/arXiv.1911.08731). arXiv: [arXiv:1911.08731](https://arxiv.org/abs/1911.08731). (Visited on 10/25/2022).
- [170] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010.
- [171] Thomas J. Santner, Brian J. Williams, William I. Notz, and Brian J. Williams. *The Design and Analysis of Computer Experiments*. Vol. 1. Springer, 2003.
- [172] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. “The Graph Neural Network Model.” In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.
- [173] Jürgen Schmidhuber. “Deep Learning in Neural Networks: An Overview.” In: *Neural networks* 61 (2015), pp. 85–117.

- [174] Samuel Schoenholz and Ekin Dogus Cubuk. “Jax Md: A Framework for Differentiable Physics.” In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 11428–11441.
- [175] Matthias Seeger. “Gaussian Processes for Machine Learning.” In: *International journal of neural systems* 14.02 (2004), pp. 69–106.
- [176] Marcin Sendera, Marcin Przewięźlikowski, Konrad Karanowski, Maciej Zięba, Jacek Tabor, and Przemysław Spurek. “Hypershot: Few-shot Learning by Kernel Hypernetworks.” In: *arXiv preprint arXiv:2203.11378* (2022). arXiv: [2203.11378](https://arxiv.org/abs/2203.11378).
- [177] Aviv Shamsian, Aviv Navon, Ethan Fetaya, and Gal Chechik. “Personalized Federated Learning Using Hypernetworks.” In: *International Conference on Machine Learning*. PMLR, 2021, pp. 9489–9502. ISBN: 2640-3498.
- [178] Shiv Shankar, Vihari Piratla, Soumen Chakrabarti, Siddhartha Chaudhuri, Preethi Jyothi, and Sunita Sarawagi. “Generalizing across Domains via Cross-Gradient Training.” In: *arXiv preprint arXiv:1804.10745* (2018). arXiv: [1804.10745](https://arxiv.org/abs/1804.10745).
- [179] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer.” In: *arXiv preprint arXiv:1701.06538* (2017). arXiv: [1701.06538](https://arxiv.org/abs/1701.06538).
- [180] Yuge Shi, Jeffrey Seely, Philip HS Torr, N. Siddharth, Awni Hannun, Nicolas Usunier, and Gabriel Synnaeve. “Gradient Matching for Domain Generalization.” In: *arXiv preprint arXiv:2104.09937* (2021). arXiv: [2104.09937](https://arxiv.org/abs/2104.09937).
- [181] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” In: *arXiv preprint arXiv:1409.1556* (2014). arXiv: [1409.1556](https://arxiv.org/abs/1409.1556).
- [182] Sarath Sivaprasad, Akshay Goindani, Vaibhav Garg, and Vineet Gandhi. “Reappraising Domain Generalization in Neural Networks.” In: *arXiv preprint arXiv:2110.07981* (2021). arXiv: [2110.07981](https://arxiv.org/abs/2110.07981).
- [183] Leslie N Smith. “No More Pesky Learning Rate Guessing Games.” In: *CoRR*, abs/1506.01186 5 (2015). arXiv: [1506.01186](https://arxiv.org/abs/1506.01186).
- [184] Leslie N Smith. “Cyclical Learning Rates for Training Neural Networks.” In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2017, pp. 464–472.
- [185] Leslie N Smith and Nicholay Topin. “Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates.” In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*. Vol. 11006. International Society for Optics and Photonics, 2019, p. 1100612.
- [186] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. “Practical Bayesian Optimization of Machine Learning Algorithms.” In: *Advances in neural information processing systems* 25 (2012).
- [187] Giorgio Soave. “Equilibrium Constants from a Modified Redlich-Kwong Equation of State.” In: *Chemical engineering science* 27.6 (1972), pp. 1197–1203.
- [188] Sho Sonoda and Noboru Murata. “Neural Network with Unbounded Activation Functions Is Universal Approximator.” In: *Applied and Computational Harmonic Analysis* 43.2 (2017), pp. 233–268.

- [189] Helmuth Späth. “Algorithm 39 Clusterwise Linear Regression.” In: *Computing* 22.4 (1979), pp. 367–373.
- [190] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.
- [191] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. “Which Tasks Should Be Learned Together in Multi-Task Learning?” In: *International Conference on Machine Learning*. PMLR, 2020, pp. 9120–9132. ISBN: 2640-3498.
- [192] Rick Stevens, Valerie Taylor, Jeff Nichols, Arthur Barney Maccabe, Katherine Yelick, and David Brown. *AI for Science: Report on the Department of Energy (DOE) Town Halls on Artificial Intelligence (AI) for Science*. Tech. rep. Argonne National Lab.(ANL), Argonne, IL (United States), 2020.
- [193] Adarsh Subbaswamy, Bryant Chen, and Suchi Saria. “A Unifying Causal Framework for Analyzing Dataset Shift-Stable Learning Algorithms.” In: *Journal of Causal Inference* 10.1 (2022), pp. 64–89.
- [194] Baochen Sun and Kate Saenko. “Deep Coral: Correlation Alignment for Deep Domain Adaptation.” In: *European Conference on Computer Vision*. Springer, 2016, pp. 443–450.
- [195] Yi Tay, Zhe Zhao, Dara Bahri, Don Metzler, and Da-Cheng Juan. “Hypergrid Transformers: Towards a Single Model for Multiple Tasks.” In: (2021).
- [196] Monika Thol, Gabor Rutkai, Andreas Köster, Rolf Lustig, Roland Span, and Jadran Vrabec. “Equation of State for the Lennard-Jones Fluid.” In: *Journal of Physical and Chemical Reference Data* 45.2 (2016), p. 023101.
- [197] Michael E Tipping. “Sparse Bayesian Learning and the Relevance Vector Machine.” In: *Journal of machine learning research* 1.Jun (2001), pp. 211–244.
- [198] Michael Tschannen, Olivier Bachem, and Mario Lucic. “Recent Advances in Autoencoder-Based Representation Learning.” In: *arXiv preprint arXiv:1812.05069* (2018). arXiv: [1812.05069](https://arxiv.org/abs/1812.05069).
- [199] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. “Deep Domain Confusion: Maximizing for Domain Invariance.” In: *arXiv preprint arXiv:1412.3474* (2014). arXiv: [1412.3474](https://arxiv.org/abs/1412.3474).
- [200] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. “Instance Normalization: The Missing Ingredient for Fast Stylization.” In: *arXiv preprint arXiv:1607.08022* (2016). arXiv: [1607.08022](https://arxiv.org/abs/1607.08022).
- [201] Guido Van Rossum and Fred L Drake. *The Python Language Reference Manual*. Network Theory Ltd., 2011.
- [202] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing Data Using T-SNE.” In: *Journal of machine learning research* 9.11 (2008).
- [203] Joaquin Vanschoren. “Meta-Learning: A Survey.” In: *arXiv preprint arXiv:1810.03548* (2018). arXiv: [1810.03548](https://arxiv.org/abs/1810.03548).
- [204] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer science & business media, 1999. ISBN: 0-387-98780-0.

- [205] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. “Deep Hashing Network for Unsupervised Domain Adaptation.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5018–5027.
- [206] Ricardo Vilalta and Youssef Drissi. “A Perspective View and Survey of Meta-Learning.” In: *Artificial intelligence review* 18.2 (2002), pp. 77–95.
- [207] Tomer Volk, Eyal Ben-David, Ohad Amosy, Gal Chechik, and Roi Reichart. “Example-Based Hypernetworks for out-of-Distribution Generalization.” In: *arXiv preprint arXiv:2203.14276* (2022). arXiv: [2203.14276](https://arxiv.org/abs/2203.14276).
- [208] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John C. Duchi, Vittorio Murino, and Silvio Savarese. “Generalizing to Unseen Domains via Adversarial Data Augmentation.” In: *Advances in neural information processing systems* 31 (2018).
- [209] Ulrike Von Luxburg. “A Tutorial on Spectral Clustering.” In: *Statistics and computing* 17.4 (2007), pp. 395–416.
- [210] Johannes Von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. “Continual Learning with Hypernetworks.” In: *arXiv preprint arXiv:1906.00695* (2019). arXiv: [1906.00695](https://arxiv.org/abs/1906.00695).
- [211] Denis Voskov and Hamdi A Tchelepi. “Compositional Space Parameterization for Flow Simulation.” In: *SPE Reservoir Simulation Symposium*. OnePetro, 2007.
- [212] Jindong Wang, Wenjie Feng, Yiqiang Chen, Han Yu, Meiyu Huang, and Philip S. Yu. “Visual Domain Adaptation with Manifold Embedded Distribution Alignment.” In: *Proceedings of the 26th ACM International Conference on Multimedia*. 2018, pp. 402–410.
- [213] Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Tao Qin, Wang Lu, Yiqiang Chen, Wenjun Zeng, and Philip Yu. “Generalizing to Unseen Domains: A Survey on Domain Generalization.” In: *IEEE Transactions on Knowledge and Data Engineering* (2022). ISSN: 1041-4347.
- [214] Mei Wang and Weihong Deng. “Deep Visual Domain Adaptation: A Survey.” In: *Neurocomputing* 312 (2018), pp. 135–153.
- [215] Peng Wang and Erling H Stenby. “Non-Iterative Flash Calculation Algorithm in Compositional Reservoir Simulation.” In: *Fluid Phase Equilibria* 95 (1994), pp. 93–108.
- [216] Qing Wang, Matthias Ihme, Yi-Fan Chen, and John Anderson. “A Tensorflow Simulation Framework for Scientific Computing of Fluid Flows on Tensor Processing Units.” In: *Computer Physics Communications* 274 (2022), p. 108292.
- [217] Shihao Wang, Nicolas Sobecki, Didier Ding, Lingchen Zhu, and Yu-Shu Wu. “Accelerating and Stabilizing the Vapor-Liquid Equilibrium (VLE) Calculation in Compositional Simulation of Unconventional Reservoirs Using Deep Learning Based Flash Calculation.” In: *Fuel* 253 (Oct. 2019), pp. 209–219. ISSN: 00162361. DOI: [10.1016/j.fuel.2019.05.023](https://doi.org/10.1016/j.fuel.2019.05.023). (Visited on 11/25/2021).
- [218] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. “A Survey of Transfer Learning.” In: *Journal of Big data* 3.1 (2016), pp. 1–40.
- [219] MS Wertheim. “Fluids with Highly Directional Attractive Forces. I. Statistical Thermodynamics.” In: *Journal of statistical physics* 35.1 (1984), pp. 19–34.

- [220] MS Wertheim. “Fluids with Highly Directional Attractive Forces. III. Multiple Attraction Sites.” In: *Journal of statistical physics* 42.3 (1986), pp. 459–476.
- [221] MS Wertheim. “Fluids with Highly Directional Attractive Forces. IV. Equilibrium Polymerization.” In: *Journal of statistical physics* 42.3 (1986), pp. 477–492.
- [222] Michael S Wertheim. “Fluids with Highly Directional Attractive Forces. II. Thermodynamic Perturbation Theory and Integral Equations.” In: *Journal of statistical physics* 35.1 (1984), pp. 35–47.
- [223] David H. Wolpert and William G. Macready. “No Free Lunch Theorems for Optimization.” In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.
- [224] Tianju Xue, Shuheng Liao, Zhengtao Gan, Chanwook Park, Xiaoyu Xie, Wing Kam Liu, and Jian Cao. “JAX-FEM: A Differentiable GPU-accelerated 3D Finite Element Solver for Automatic Inverse Design and Mechanistic Data Science.” In: arXiv preprint arXiv:2212.00964 (2022). arXiv: [2212.00964](https://arxiv.org/abs/2212.00964).
- [225] Shen Yan, Huan Song, Nanxiang Li, Lincan Zou, and Liu Ren. “Improve Unsupervised Domain Adaptation with Mixup Training.” In: arXiv preprint arXiv:2001.00677 (2020). arXiv: [2001.00677](https://arxiv.org/abs/2001.00677).
- [226] Xiangyu Yue, Yang Zhang, Sicheng Zhao, Alberto Sangiovanni-Vincentelli, Kurt Keutzer, and Boqing Gong. “Domain Randomization and Pyramid Consistency: Simulation-to-real Generalization without Accessing Target Domain Data.” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 2100–2110.
- [227] Seniha Esen Yuksel, Joseph N. Wilson, and Paul D. Gader. “Twenty Years of Mixture of Experts.” In: *IEEE transactions on neural networks and learning systems* 23.8 (2012), pp. 1177–1193.
- [228] Bin Zhang. “Regression Clustering.” In: *Third IEEE International Conference on Data Mining*. IEEE, 2003, pp. 451–458. ISBN: 0-7695-1978-4.
- [229] Hanlin Zhang, Yi-Fan Zhang, Weiyang Liu, Adrian Weller, Bernhard Schölkopf, and Eric P Xing. “Towards Principled Disentanglement for Domain Generalization.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 8024–8034.
- [230] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. “Mixup: Beyond Empirical Risk Minimization.” In: arXiv preprint arXiv:1710.09412 (2017). arXiv: [1710.09412](https://arxiv.org/abs/1710.09412).
- [231] Marvin Zhang, Henrik Marklund, Nikita Dhawan, Abhishek Gupta, Sergey Levine, and Chelsea Finn. “Adaptive Risk Minimization: Learning to Adapt to Domain Shift.” In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 23664–23678.
- [232] Tao Zhang, Yu Li, Yiteng Li, Shuyu Sun, and Xin Gao. “A Self-Adaptive Deep Learning Algorithm for Accelerating Multi-Component Flash Calculation.” In: *Computer Methods in Applied Mechanics and Engineering* 369 (2020), p. 113207.
- [233] Dominic Zhao, Johannes von Oswald, Seijin Kobayashi, João Sacramento, and Benjamin F. Grewe. “Meta-Learning via Hypernetworks.” In: (2020).

- [234] Yun Zhi and Huen Lee. “Fallibility of Analytic Roots of Cubic Equations of State in Low Temperature Region.” In: *Fluid phase equilibria* 201.2 (2002), pp. 287–294.
- [235] Tao Zhong, Zhixiang Chi, Li Gu, Yang Wang, Yuanhao Yu, and Jin Tang. “Meta-DMoE: Adapting to Domain Shift by Meta-Distillation from Mixture-of-Experts.” In: arXiv preprint arXiv:2210.03885 (2022). arXiv: [2210.03885](https://arxiv.org/abs/2210.03885).
- [236] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. “Domain Generalization: A Survey.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022). ISSN: 0162-8828.
- [237] Kaiyang Zhou, Yongxin Yang, Timothy Hospedales, and Tao Xiang. “Learning to Generate Novel Domains for Domain Generalization.” In: *European Conference on Computer Vision*. Springer, 2020, pp. 561–578.
- [238] Kaiyang Zhou, Yongxin Yang, Yu Qiao, and Tao Xiang. “Domain Generalization with Mixstyle.” In: arXiv preprint arXiv:2104.02008 (2021). arXiv: [2104.02008](https://arxiv.org/abs/2104.02008).
- [239] Kezheng Zhu and Erich A. Müller. “Generating a Machine-Learned Equation of State for Fluid Properties.” In: *The Journal of Physical Chemistry B* 124.39 (2020), pp. 8628–8639.
- [240] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. “A Comprehensive Survey on Transfer Learning.” In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.
- [241] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. “Designing Effective Sparse Expert Models.” In: arXiv preprint arXiv:2202.08906 (2022). arXiv: [2202.08906](https://arxiv.org/abs/2202.08906).
- [242] Nicolas von Solms, Michael L. Michelsen, and Georgios M. Kontogeorgis. “Computational and Physical Performance of a Modified PC-SAFT Equation of State for Highly Asymmetric and Associating Mixtures.” In: *Industrial & engineering chemistry research* 42.5 (2003), pp. 1098–1105.