Imperial College London

Department of Earth Science and Engineering

MSc in Applied Computational Science and Engineering

Independent Research Project
Project Plan

Analysis of rapid flash calculations using machine learning

by

Hanlin XIA

Email: dombledore.Xia21@imperial.ac.uk
GitHub username: acse-dx21
Repository:
https://github.com/acse-dx21/IRP-flash-calculation-accelaration.git

Supervisors:

Dr. Salinas Pablo

Professor Jackson Matthew D
Santos Silva, Vinicius

September 2022

Analysis of rapid flash calculations using machine learning

Hanlin XIA

September 2022

Abstract

Phase equilibrium (flash) calculations have been a long-standing hot topic in thermodynamics. Common method such as successive substitution method to correlate and predict vapor-liquid equilibrium takes a lot of time due to long-term iteration. Based on that, several methods that use machine learning have been gaining ground. This paper explores the performance of a range of new born models in AI field available today for flash calculation. It shows that models based on Deep neural network(DNN) can perform well in accelerating phase equilibrium (flash) calculations and reach excellence accuracy as well. In addition, our study shows that "traditional machine learning" such as gradient boosting can be less accurate for one shot prediction but still useful due to its versatility and stability in long term predictions. In terms of time consumption, AI model is faster in predict big size data sets using GPU and some DNN models can also balances prediction speed with excellent accuracy.

1 Introduction

Phase equilibrium (flash) calculations have been a long-standing hot topic in thermodynamics. Over a long period of time, a number of experiments or models have been carried out to collect data on mixtures[17]and in the process, models using Equation of state(EOS) have been proposed and verified, eventually becoming widely accepted by people[24][23]. However, common method based on EOS correlating and predicting vapour-liquid equilibrium such as the continuous substitution method(SSM) takes a lot of time due to long iterations. Due to the high time consumption of traditional methods, attention is focused on other modes and attempts to find better solutions. Yu Li et al [16] introduced a new model via a deep neural network (DNN) approach in an attempt to replace this process. Deep neural networks are an important structure in machine learning methods and have had a huge impact in different areas, including self-driving cars, image recognition and more. Due to its black-box nature and flexibility, it has been used in different areas in an attempt to solve long-standing problems, with great success in many areas. Yu et al [16], using variations of DNN and one can obtain very good results for flash calculations. It even outperforms the Newtonian and sparse grid methods in some respects, such as implementation efficiency, and in the same period, physics-informed neural networks (PINNs) were introduced, some of which proved to have good performance in certain situations[11][5]. Despite of deep learning, another AI method based on tree ensemble model have also been used in this region. In 2016, T. Chen et al[6] made a study showed that traditional machine learning methods (like Gradient boosting) had dominant status in processing tabulate data. Also in 2022, Ravid Shwartz-Ziv [20] give a wide range performance test between deep learning model and tree ensemble model (such as XGBoost) in processing tabulate data. They concluded that XGBoost outperformed these deep models in the vast majority of data-sets. Furthermore, because the inputs of neural networks need to be well defined when they are encountered with real-world problems, they are often accompanied by a range of tricky problems such as lack of locality, data sparsity and mixed feature types[20]. At the same time, tree-set based methods tend to require less time to adjust and less computational cost to fit the data, where they will perform better[20]. In their paper, Ravid et al[20] shows that combination of deep models and XGBoost can perform better on these datasets than single model alone.

Due to the success of DNN for flash calculations, but simultaneously the known problems during this process, in this project we are going to compare the different performance of variety, emerging deep learning model and traditional machine learning model for flash calculations. Moreover, we will also study the effect of physical constrain on prediction and the compare long-term effect of performing several consecutive predictions with different models.

2 Background

Building models in flash calculation is an enduringly popular topic, and over the past few decades, efforts have been made to discover better and faster models to accurately predict this process. Models using Equation of state(EOS) has been widely discussed and accepted by people [25]. Generally, flash calculation can be divided into smaller problems include the NPT flash and the NVT flash and NPT is more popular compare to NVT flash. In NPT flash, models using EOS take parameters include critical temperature(Tc),critical pressure(Pc),acentric factor(Ac) and mole fraction of materials(Z_i) can provide exact result in the end. The most common method based on the EOS that people started using was SSM, a method that determines the physical information of the next step such as mass density and molar densities. In the end of the iteration, the phase information of flash equilibrium can be calculated. However, in real case, some root of SSM are not physically meaningful and are quite time consuming to compute, which often limits the practicability in real scenarios. In order to fix this problem, a series of studies have been carried out including Newton's method and Sparse Grids method[27]. Newton's method is implemented by using Newton's iterative equation $x_{n+1} = x_n + \frac{f(x)}{f'(x)}$ and Sparse Grids method using the separation of online and offline model. they can accelerate the original SSM process for around two times and five hundred times separately[16]

AI models actually first tried a long time ago[12].But it was only with emergence of Alexnet[15] in 2012 that the role of neural network models with different structures started to be noticed and became popular, after that various models were discovered and shown their unique capabilities. In the process, many deep models were created, such as generative adversarial networks (GAN)[9] and yolo [18] and have great success in processing images. However, deep learning still didn't handle tabular data well at that time before the emergence of attention mechanisms[26].Transformer has demonstrated its disruptive power since its inception[26] in Natural Language Processing. After that several attention-like models have been unearthed in processing tablue data[21] and have shown good performance even better to that of XGBoost in several region[1]. We attempt to use and compare these new born deep model in flash calculation in our paper. As far as we know, we are the first one implement transformer model in flash calculation. traditional machine learning models are likewise evolving, from decision trees to gradient boosting decision tree(GDBT)[6], the accuracy has grown considerably.They can predicts the error of the previous models and then combines all the models at the end to make the final prediction, during the process the model will perform better and better. However, the long training time has become a big problem for this model. To solve this, LightGBM [13] was proposed to solve the problem of long training times. Tree ensemble models have shown advanced performance in processing tabular data compare to deep model in many data set[20]. So we include several tree models in the comparison of the performance in VLE calculation.

In this paper we deploy several models mainly based on traditional machine learning and new born deep learning based on transformer. We will briefly describe the model used in our experience at below

3 Method

3.1 Machine learning models

AI model, one of the most popular models in recent years, has a wide range of variants. Traditionally, models based on decision tree algorithms and Gradient boosting method such as CatBoost [8], LightGBM [13] and XGBoost [6], have shown to have excellent performance when dealing with tabular data. On the other side, several new born AI model based on transformer has emerged and some are considered in our paper.

We deploy tabnet and FT-transformer[10] in our paper and both of them show excellent result in flash calculation. We will also compare a range of other model with them others include: CatBoost Regressor[8],XGBoost Regressor[6],LightGBM Regressor[13]. Details and range of the hyper-parametric search for these models are shown in Appendix B.

3.1.1 transformer-based model

Tabnet is a neural network for tabular data designed by [1]. It implements instance-wise feature selection through a sequential attention mechanism similar to the additive model, and self-supervised learning through an encoder-decoder framework. As a result, decision manifolds can be viewed as hyperplane boundaries, which is similar to tree ensemble method and works well for tabular data. Because of this, tabnet is widely used and have successed in many places.

FT-transformer This is a new and powerful model that can be adapted to tabular data by making simple changes to the original transformer model[26]. This model converts all features (categorical and numerical) into embeddings and applies a bunch of transformer layers to the embeddings.This method have show excellent result in many dataset in the paper[10]. We standardize input data before implement this model in our experience

3.1.2 Tree-ensemble models

XGBoost(Regressor) [6] is a modern model based on GBDT. XGBoost model extend GBDT by a number of improvements, such as second derivative fitting, achieve very good performance in many datasets [29, 19]. Until now, XGBoost has been one of the most popular models in tabular data and has demonstrated its high generality and accuracy[20]. We use python XGBoost library to deploy XGBoost Regressor model.

XGBoost(Regressor-post) Physical informed neural network have been shown to have positive result in neural net work [11].A reason is that a model may predict any result located any place in real line, But some of them might be physically impossible, such as mole fraction large than one or less the zero. We can fix that by perform several constraint similar to PINN to post process the prediction of result of model

LightGBM Regressor [13] Light Gradient Boosting Machine (LGBM) is an improved algorithm based on gradient boosting trees and XGBoost, which is based on a histogram-like decision tree algorithm that greatly improves training efficiency, allowing it to obtain hundreds of times higher training efficiency compared to XGBoost without much loss of accuracy. We use python lightgbm library to deploy LightGBM Regressor model

3.1.3 Neural network

Simple-ANN, A neural network consisting of only fully linked and linear layers composited with activation function, with 2-7 hidden layers, each of which can be represented by the following mathematical formula

$y = f_n \circ f_{n-1} \cdots \circ f_0(x)$

where

$$f_i(x) = \sigma(W_i \cdot x + b)$$

the σ represent activation function and W-i represent corresponding fully connected layer. We use pytorch library in python to deploy our model, in the process we use batch-normalization and dropout[22] to optimize this neural network. They are effective in avoiding falling into local optima and over-fitting. We standardize input data before implement this model in our experience

3.2 Hardware

central processing unit (CPU) is an electronic unit that dominates the entire program running. Even AI model trained based on other hardware often needs the assistance of the CPU. The CPU focuses its relatively small number of cores on a single task and gets it done quickly. This makes it particularly suitable for handling types of work ranging from serial computing to database runs. we will use intel i9 9900K as our CPU processor in this experience.

Graphics processing unit(GPU) is a specialized electronic unit that was originally designed for use in images processing acceleration. But due to the high efficiency of AI models to GPU [7][28], with hundreds of millions of parameters to compute[2] , GPU are one of the most dominant devices for training deep models today. In this paper, we use Nvidia RTX 2080 super as our GPU device in our experience training models

3.3 Comparing models

In this paper, we only investigate how AI model perform on accelerating flash calculation. apparently, these model should show good performance in (1)accuracy (2)make inference(prediction) efficiently. Therefore, we first evaluate how different model's capabilities on the calculation. Then we demonstrate that by 1) increasing training data size, 2) by using post-processing such as physical constrained to the model and compare the effect. On the other side, We will compare how well the GPU accelerates on different data sets. Finally, we have a view of how accumulate error effect on model accuracy as a simulation of real continues time flash calculation.

3.4 Experiment setup

Table 1: Data shape: model input and target

Pcs_1	Pcs_2	Tcs_1	Tcs_2	Ac_1	Ac_2	T	P	Z_1	Z_2	v_1	v_2	l_1	l_2	V	L
4599000.0	3796000.0	190.564	425.12	0.008	0.193	276.929	827698.7	0.384	0.616	0.834	0.166	0.045	0.955	0.43	0.57
3796000.0	3025000.0	425.12	507.6	0.193	0.298	456.026	978924.06	0.392	0.608	0.392	0.608	0.0	0.0	1.0	0.0
4599000.0	2740000.0	190.564	540.2	0.008	0.346	272.649	422675.22	0.167	0.833	0.996	0.004	0.026	0.974	0.145	0.855
3796000.0	3025000.0	425.12	507.6	0.193	0.298	724.627	254372.39	0.999	0.001	0.999	0.001	0.0	0.0	1.0	0.0
4599000.0	4248000.0	190.564	369.83	0.008	0.152	112.353	784557.44	0.02	0.98	0.0	0.0	0.02	0.98	0.0	1.0
4599000.0	2740000.0	190.564	540.2	0.008	0.346	716.948	851632.2	0.658	0.342	0.658	0.342	0.0	0.0	1.0	0.0

On the left of the vertical line is the model input data and on the right is the model target data which generated by Thermo library

3.4.1 data set prepare

In this paper, in order to fit and optimize the data using AI model, substantial data need to be generated for Vapor-Liquid equilibrium (VLE) of several materials. We focus on hydrocarbon system which containing binary combination of Methane, Ethane, Propane, N-Butane,N-Pentane, N-Hexane, Heptane from C1 to C7. Temperature(T), pressure(P) will also be used as input in flash calculation, that is, let model to predict NPT kind flash. Besides that two, three physical property are also been used as input which is critical pressure(Pc), critical temperature(Tc) and acentric factor(Ac)

Table 2: Physical properties of the seven substances in thermo

material	Tc	Pc	Ac
Methane	190.564	4599000.0	0.008
Ethane	305.32	4872000.0	0.098
Propane	369.83	4248000.0	0.152
N-Butane	425.12	3796000.0	0.193
N-Pentane	469.7	3370000.0	0.251
N-Hexane	507.6	3025000.0	0.2975
Heptane	540.2	2740000.0	0.3457

alongside with material fraction(Z)(table 1). The physical property and output data,which is material fractions in vapor(v) and then in liquid(l) and then Molar ratio of gases to liquids(z), is generated from a open source flash equilibrium data library named thermo[3]. It is an easily used library while remains strong functionality and the physical property generate by this table is shown in this table 2. In this experience, all of data will be generated through this library. In this experience we create data with different size. We create four training data set with size of 8400,84000,168000,252000 and four corresponding test data set with size 2100,21000,42000,63000,We will further split a validation set from each training data set with same size of corresponding test set in training process, that is, the ratio is constant 3:1:1 between training, validation and test data set. These data is Stratified created that each material combination accounts for $1/21$ of the certain data set with randomly chose temperature, Pressure and material fraction.This number comes from the combination number C_7^2 . Since some input is not realistic(such as -300°C) and will cause the system to collapse. Instantaneous data generation is also time consuming and can make evaluation process time too long. As a result, we choose to generate meaningful data and then store them into local file and so that we can get to clean data quickly and whenever we want.

3.4.2 The evaluation system

hyper-parameters. Each model must have their corresponding hyper-parameters to regulating its fiting behaviour and different hyper-parameters will result in different performace. In order to find the best hyper-parameters, we choose to use Bayes optimization [4] as a selector. These parameters not only influence the training path of the model, but also determine the final outcome of the model. For each model, we will use bayes optimization for around 30 times and the mean and standard error of result of model from these hyper-parameters will be report into plot.

Metric and evaluation. We used mean squared error (MSE) as an indicator of the accuracy of the model output and the target, and used seconds as the unit of measurement for prediction time. each model will be run at approximately 30 times on each data set with different hyper-parameters chosen by Bayes optimization. The mean and variance of the results for each model under these hyper-parameters will be plotted as lines and corresponding shadows to judge the performance of the model.

Training For the deep learning model, we train it until its validation loss cannot fall any further 100 steps(patient = 100) with maximum epoch 1000. we use the adam optimizer[14] for descent with fixed learning rate. We will normalize the training input data to have zero mean and unit variance and use regularization factor from training set to normalize test data set. As for the tree-ensemble model, we use the corresponding hyper-parameters chosen by Bayes optimization to modify and stop it at the right time, with validation set and patient=100 as well. When iteration stops, they will choose the parameter of model with best performance in validation set and use that model to get test score in test set. For tree-ensemble model, we do not normalize the input data.

Continuous-prediction (rollout) we may have to face a situation where a model repeatedly predicts

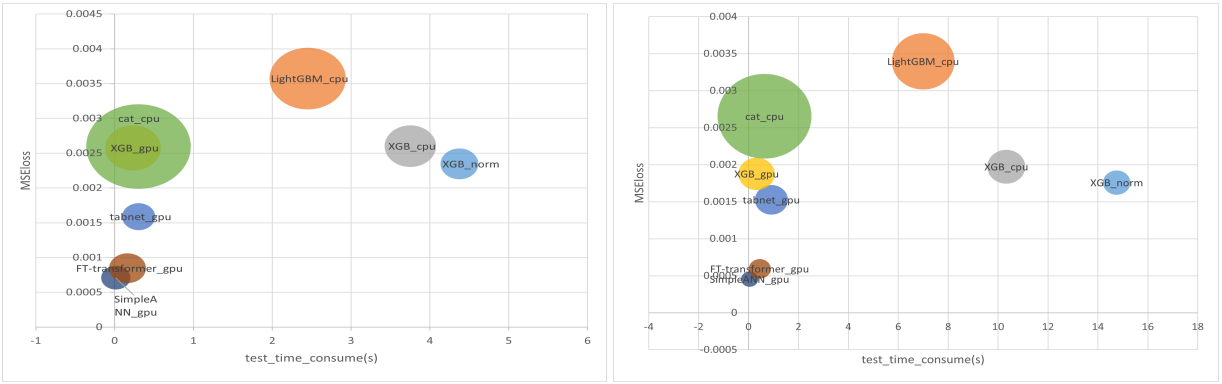

Figure 1: model performance of 21000(test)/84000(train) data set(left) and 63000(test)/256000(train) data set(right). the circle size reflects accuracy standard deviation. The X-axis represents the time taken to predict the corresponding test data set and the Y-axis represents the MSE between the target and the prediction(both lower means better) in test data set.

the VLE several times. In this situation, error may accumulate and model predictions will become increasingly unreliable. We simulated this process by replacing the molar ratio of the substance with the molar ratio calculated by the model. The ratio will be calculated as follows:

$$Z_{1new} = v_{1model}V_{model} + l_{1model}L_{model}$$
$$Z_{2new} = v_{2model}V_{model} + l_{2model}L_{model}$$

We will substitute it into the input of next iteration. We deploy this experience on the data set of size 42000(test)/168000(train),

4 Discussion and Result

4.1 Discussion

4.1.1 accuracy performance

- As we expected, the larger the training data set (figure2), the better the performance across the different data sets. This is because a larger data set gives the model more information and allows it to get a more accurate picture of the situation. When we focus on models(figure1), simple-ANN performs the best, followed by transformer-based depth models such as tabnet and FT-Transformer, due to the fact that the transformer module can focus on the more important information and ignore the less important information. Meanwhile, very good results are also obtained with XGBoost, which performs similarly to tabnet in terms of accuracy, achieving MSE below 0.01 and decreasing steadily as the size of data set grows.figure 6 illustrates the ANN and XGB models in their six outputs compared to that of thermo library(ground truth), in which we can see that the ANN matches the true results better in most cases. Although we still can not say which AI model is the best at present stage. It is important to note that it is difficult to fully compare the merits of the two models, as the different models have different hyper-parameters and different processing of the input data can result in significantly different. The accuracy of Simple-ANN and F-T transformer rely heavily on data normalization, When we remove the regularization the accuracy will be significantly reduced and become the worst model. We need to notice that sometimes it's hard to decide what the variance and mean of mixture condition in reality and since XGBoost and tabnet do not relies on that, it will make it more convenient to implement. It is therefore not able to say that deep model is better

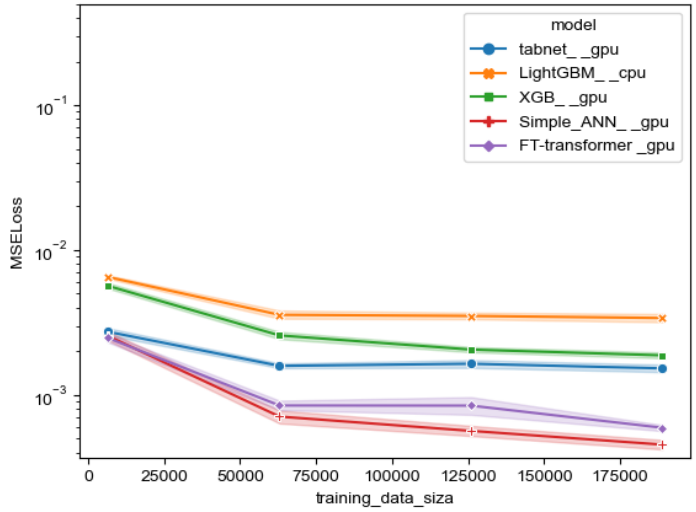

Figure 2: We compare the accuracy of some of the models, with mean and standard error of the average MSE loss for models generated by different hyper-parameters in Bayesian optimization. Each model was trained separately on a different data set

than tree ensemble model in all cases, but the future performance of this type of model is still expected due to the advantages shown by our study in phase calculation .The results of the post-processing of the XGBoost output are also shown in the figure 1 named XGB-norm. In that plot, We can see some improvement with this process in accuracy, but it also requires additional time for post-processing. It shows that the model can perform better with post-processing methods to remove data that do not conform to physical rules. As for the newborn models, the transformer's excellent attention mechanism allows it to outperform other models and given that transformer-based models are an emerging field and have shown strong performance, we can expect this kind of model to show even stronger performance in future studies.

4.1.2 GPU/CPU performance

- We test three different model in figure 3. In that plot, the use of GPU provides roughly hundreds of times improvements in time-efficient to the models when data set is big enough (figure 3). Each model was tested with GPU and CPU separately. The plot shows the mean and variance of test in time from model with different hyper-parameters. The only two variables in our experiments is the equipment and size of data set, with including single instance prediction, in which we can see that the gap between CPU and GPU will become increasingly apparent thanks to powerful parallelism of the GPU and their high adaptability to AI models. At the very beginning the effect of GPU acceleration was not significant and even less efficient than GPU in some cases. However, in the end we can find that the GPU can accelerate the model by roughly one or two orders of magnitude when the data set is large enough. Both tree ensemble model and deep model need to process large amounts of data to train parameters, so they can both benefit from the powerful parallelism of the GPU. However, we still need to be aware that the CPU is generally more efficient in sparse prediction, thus in reality we will need to choose the device by when prediction request is low

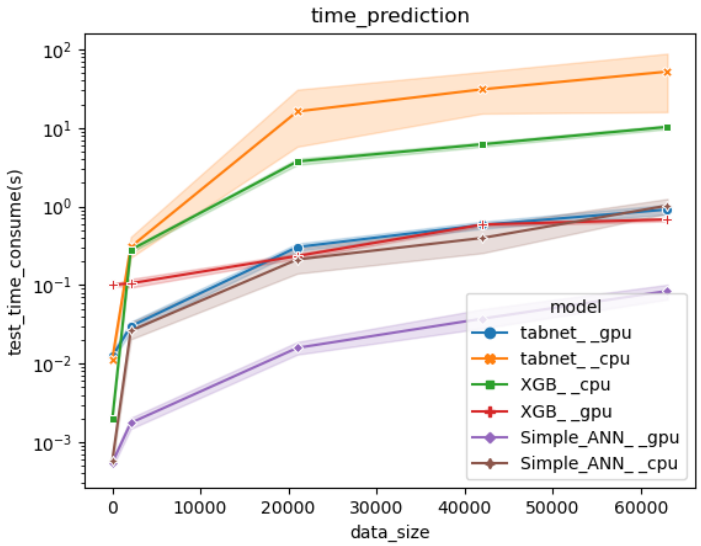

Figure 3: This graph shows the model acceleration levels of the GPU and CPU for different models in four test data sets of sizes 1 to 63,000, with mean and standard error of the average test time for models generated by different hyper-parameters in Bayesian optimization

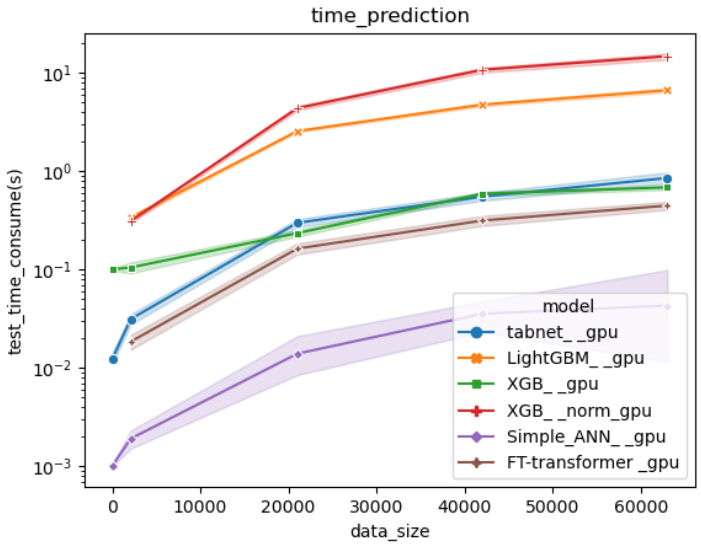

Figure 4: We compare the time consuming between models in four test data sets of sizes 1 to 63,000, with mean and standard error of the average test time for models generated by different hyper-parameters in Bayesian optimization

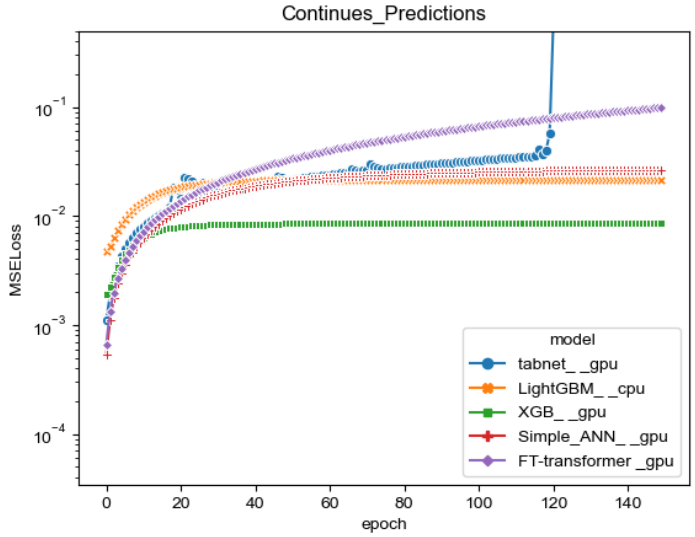

Figure 5: the performance of different models in successive prediction experiments and found that some models that performed very well in one prediction became progressively less accurate as the number of iterations increased.

4.1.3 time performance*

- In the figure4 above, we find that Simple-ANN performs significant better than others. Clearly, the less parameter the model the faster the prediction can be made, whereas other models tend to consume a high amount of computation time due to the huge amount of parameters of the model. Then, As the sample size increases, the model prediction time also gradually increases. In real life, there is always a finite amount of time to forecast a phase statement, so it makes sense to compare the time consumed by different models' forecasts. Although in reality there are many reasons that the prediction speed of a model can be affected. Different equipment and hyper-parameters can cause different models to perform differently, and the degree of optimisation of different models by different organisations can also lead to different results, our experiments are still instructive to some extent. From the figure4 we can see that even with the upper bound of the shading of the line, ANN is still much faster than the other models

4.1.4 rollout performance

- In reality, we often have to deal with models that predict continuous changes in a period of time, which is why the stability of the model in continuous prediction is so important. Despite the excellent performance of these models, they are still not as accurate as they could be in continuous prediction (figure 5), and the accumulation of errors leads to a deteriorating ability to apply the models.The plot in figure5 shows that all models increase in error as the number of iterations increases. Some model like F-T transformer and Simple ANN have a very good performance in terms of one prediction, but they can't handle well in continuous prediction and then gradually become less accuracy than other models(figure5). Some models such as tabnet will even converge to positive infinity after a certain number of iterations. A reason could be that each prediction of sub-model in tree ensemble models is finite, so the final prediction is also finite. However, for the depth model, since the model results are completed by a series of algebraic operations and no bounds are set, the accumulation of errors in successive predictions may convergence to infinity and the model will be no longer able to predict what will happen after that.

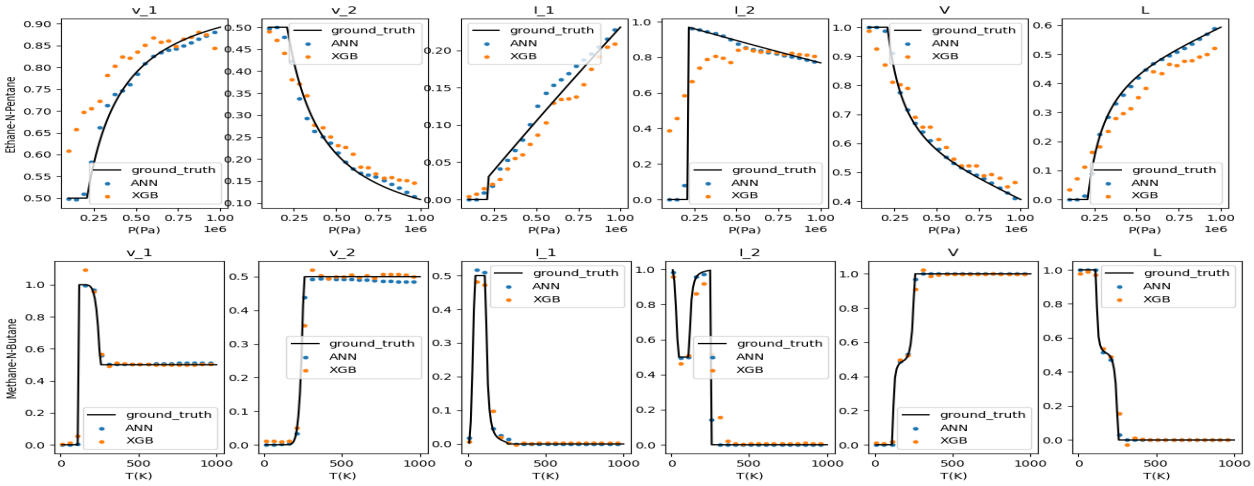

Figure 6: Simple-ANN and XGBoost model prediction vs data generate from thermo library(ground truth). The graph above shows the pressure (in Pa) versus the six outputs of the model at a temperature equal to 500 K. The graph below shows the temperature versus the six outputs of the model for a pressure equal to 1e5 Pa. The materials used is to the left of the images

4.2 observation

First we tested how the amount of data affects the accuracy of models, then we compared the difference in accuracy between the models, and we tested the gain in performance of the model by physical constraint. we also tested the difference in prediction time between the models in terms of test time, and that in difference caused by using GPU and CPU. we have following observation:

- In terms of accuracy, Simple ANN performs the best,but transformer based model can get similar accuracy when training size get large enough.
- In terms of prediction time consumption, Simple-ANN has an absolute advantage.
- GPU allows the model to be trained faster in most cases, except when the data set is particularly small. In addition, GPU can accelerate deep learning better than tree-ensemble model
- Physical constraint will help model perform better
- The error increases when rollout applied in each model, However, the pattern of error accumulation can be different. Generally, tree-ensemble method perform better than other model in this case

The observation is surprising. We thought that more advanced models might perform better, but it turns out that the ANN has the highest accuracy. It may be that more advanced models require more time to optimise and we need longer to tune to approach the best performance of the model. On the other hand, we observe that the depth model is more sensitive than the tree model in terms of accumulate error, it often get worse result after 50 iteration. This may also be a reflection of the chaotic nature of the deep model.

5 Conclusions*

In this paper, we study how different AI model performance in making fast and accurate prediction of equilibrium in phase change. Due to the rapid growth in AI era, we can use various model in flash calculation. In our experience, despite significant progress in AI model, Simple ANN perform best in terms of single prediction accuracy and prediction time consumption as long as the data features are regularly distributed. On the other side, deep model based on transformer can also show close results

in both accurate predict and computation accelerating, which show great potential of deep model based on transformer. Now dozens of models based on transformer are still being created every year since this is still a new field, models capable of greater adaptability and accuracy may be on the horizon. On the other hand, XGBoost Regressor has best results in rollout prediction. In addition, our experience shows that by adding physical constraints, models can do even better.

Beside acceleration based on different model, we also navigate how GPU performance in reducing calculation time. We conclude that generally GPU have positive effect on computation, except for data size is extremely low. In real world, it can happened requests are very sparse and computation by using CPU will become a better choice. Based on that, find way to keep balance between CPU and GPU will become a recommended method to improve model performance in time.

References

[1] Sercan Ö. Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6679–6687, May 2021. https://ojs.aaai.org/index.php/AAAI/article/view/16826.

[2] Sean Baxter. moderngpu 2.0. https://github.com/moderngpu/moderngpu/wiki, 2016.

[3] Caleb Bell and Contributors. Thermo: Chemical properties component of chemical engineering design library (chedl). 2016-2021. https://github.com/CalebBell/thermo.

[4] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science &amp Discovery*, 8(1):014008, july 2015. https://doi.org/10.1088/1749-4699/8/1/014008.

[5] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review, 2021. https://arxiv.org/abs/2105.09506.

[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery. https://doi.org/10.1145/2939672.2939785.

[7] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1337–1345, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. http://proceedings.mlr.press/v28/coates13.pdf.

[8] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support, 2018. https://arxiv.org/abs/1810.11363.

[9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. https://arxiv.org/abs/1406.2661.

[10] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 18932–18943. Curran Associates, Inc., 2021. https://proceedings.neurips.cc/paper/2021/file/9d86d83f925f2149e9edb0ac3b49229c-Paper.pdf.

[11] Thelma Anizia Ihunde and Olufemi Olorode. Application of physics informed neural networks to compositional modeling. *Journal of Petroleum Science and Engineering*, 211:110175, 2022. https://www.sciencedirect.com/science/article/pii/S0920410522000675.

[12] Maria C Iliuta, Ion Iliuta, and Faiçal Larachi. Vapour–liquid equilibrium data analysis for mixed solvent–electrolyte systems using neural network models. *Chemical Engineering Science*, 55(15):2813–2825, 2000. https://www.sciencedirect.com/science/article/pii/S0009250999005291.

[13] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014. https://arxiv.org/abs/1412.6980.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[16] Yu Li, Tao Zhang, Shuyu Sun, and Xin Gao. Accelerating flash calculation through deep learning methods. *Journal of Computational Physics*, 394:153–165, 2019. https://www.sciencedirect.com/science/article/pii/S0021999119303596.

[17] John M Prausnitz, Rudiger N Lichtenthaler, and Edmundo Gomes De Azevedo. *Molecular thermodynamics of fluid-phase equilibria*. Pearson Education, 1998.

[18] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. http://arxiv.org/abs/1506.02640.

[19] Xudie Ren, Haonan Guo, Shenghong Li, Shilin Wang, and Jianhua Li. A novel image classification method with cnn-xgboost model. In Christian Kraetzer, Yun-Qing Shi, Jana Dittmann, and Hyoung Joong Kim, editors, *Digital Forensics and Watermarking*, pages 378–390, Cham, 2017. Springer International Publishing.

[20] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022. https://www.sciencedirect.com/science/article/pii/S1566253521002360.

[21] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C. Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training, 2021. https://arxiv.org/abs/2106.01342.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. http://jmlr.org/papers/v15/srivastava14a.html.

[23] T.C. Tan, C.M. Chai, A.T. Tok, and K.W. Ho. Prediction and experimental verification of the salt effect on the vapour–liquid equilibrium of water–ethanol–2-propanol mixture. *Fluid Phase Equilibria*, 218(1):113–121, 2004. https://www.sciencedirect.com/science/article/pii/S0378381203005168.

[24] T.C. Tan, Rowell Tan, L.H. Soon, and S.H.P. Ong. Prediction and experimental verification of the effect of salt on the vapour–liquid equilibrium of ethanol/1-propanol/water mixture. *Fluid Phase Equilibria*, 234(1):84–93, 2005. https://doi.org/10.1016/j.fluid.2005.05.019.

[25] T.C. Tan, Rowell Tan, L.H. Soon, and S.H.P. Ong. Prediction and experimental verification of the effect of salt on the vapour–liquid equilibrium of ethanol/1-propanol/water mixture. *Fluid Phase Equilibria*, 234(1):84–93, 2005. https://www.sciencedirect.com/science/article/pii/S0378381205001871.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[27] Yuanqing Wu, Christoph Kowitz, Shuyu Sun, and Amgad Salama. Speeding up the flash calculations in two-phase compositional flow simulations – the application of sparse grids. *Journal of Computational Physics*, 285:88–99, 2015. https://www.sciencedirect.com/science/article/pii/S0021999115000169.

[28] Huan Zhang, Si Si, and Cho-Jui Hsieh. Gpu-acceleration for large-scale tree boosting, 2017. https://arxiv.org/abs/1706.08359.

[29] Yun Zhao, Girija Chetty, and Dat Tran. Deep learning with xgboost for real estate appraisal. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1396–1401, 2019. https://doi.org/10.1109/SSCI44817.2019.9002790.

A Training\Test Data set range

- temperature(T):uniformly distribution [10, 1000] unit: K
- pressure(P): uniformly distribution [1e5,10e5] unit: Pa
- material fraction(Zs):uniformly distribution [0,1], with sum = 1

B hyperparameters and optimization

B.1 XGBoost Regressor

library: XGBoost.

list of hyper-parameters range

- tree-method="gpu-hist" if GPU else None
- subsample: uniformly distribution [0.2, 1.0],
- learning-rate: uniformly distribution [0.01, 0.1],
- n-estimators: uniformly distribution [300, 500],
- max-depth: uniformly distribution [3, 10],
- colsample-bytree: uniformly distribution [0.6, 0.99],
- reg-lambda: uniformly distribution [10, 30]

B.2 XGBoost Regressor-post

We modify output of XGBoost Regressor model of this form

1: Set the output greater than one to one

2: Set the output less than zero to zero

3: if predicted V = 0 or L = 1, set v-1,v-2 to 0

4: if predicted V = 1 or L = 0, set l-1,l-2 to 0

B.3 Tabnet

library: pytorch-tabnet.

list of hyper-parameters range

- n-d: uniformly distribution [32, 128],
- n-a: uniformly distribution [32, 512],
- n-steps: uniformly distribution [1, 5],
- gamma: uniformly distribution [0.5, 2],
- lambda-sparse=0,
- n-independent=2,
- n-shared=1,

B.4 LGBMRegressor

library: lightgbm.

list of hyper-parameters range

- subsample: uniformly distribution [0.2, 1.0],
- learning-rate: uniformly distribution [0.01, 0.1],
- n-estimators: uniformly distribution [300, 500],
- max-depth: uniformly distribution [3, 10],
- colsample-bytree: uniformly distribution [0.6,0.99],
- reg-lambda: uniformly distribution [10,30]

B.5 RandomForestRegressor

library: sklearn.

list of hyper-parameters range

- n-estimators: uniformly distribution [5,300],
- min-samples-split: uniformly distribution [10, 100],
- min-impurity-decrease: uniformly distribution [0, 0.1],
- max-depth: uniformly distribution [3, 50]

B.6 CatBoostRegressor

library: catboost.

list of hyper-parameters range

- "iterations": uniformly distribution [20,200],
- "learning-rate": uniformly distribution [0.01,0.5],
- "depth": uniformly distribution [2,16],
- "bagging-temperature" : uniformly distribution [0.1,0.5]

B.7 Simple-ANN

library: pytorch.

list of hyper-parameters range

- node-per-layer: uniformly distribution [100, 1000],
- hidden-layer: uniformly distribution [2, 6],
- dropout-rate: Fixed 0.25