Which Deep Learning Framework Should I Use: A Comparative Study For Deep Regression Modeling

Vishu Gupta
ECE Department
Northwestern University
Evanston, USA
vishugupta2020@
u.northwestern.edu

Wei-keng Liao
ECE Department
Northwestern University
Evanston, USA
wkliao@
eecs.northwestern.edu

Alok Choudhary
ECE Department
Northwestern University
Evanston, USA
choudhar@
eecs.northwestern.edu

Ankit Agrawal
ECE Department
Northwestern University
Evanston, USA
ankitag@
eecs.northwestern.edu

***Abstract*—The combined impact of deep learning techniques and computing resources with an increase in the availability of databases is transforming many research fields leading to technological advances to help solve real-life and research-related problems. In the recent years, various deep learning frameworks and libraries have been developed to implement these algorithms that can operate efficiently at large scale and heterogeneous environments. However, these implementations can vary depending on the framework leading to unexpected inconsistency in the results even for the same algorithm. This irregularity is seen more often in deep learning models trained using advanced parallel computing resources such as GPUs. In this study, we perform an investigation with three of the well-known deep learning frameworks: Tensorflow 1, Tensorflow 2 with Keras, and Pytorch for regression-based problems in physical sciences to analyze how the results vary depending on the framework. We implement different deep neural networks with varying complexity and perform accuracy, time, and computational based analysis to study the effect of the framework on model accuracy, training/testing times, reproducibility, and memory usage.**

***Index Terms*—deep learning, predictive modeling, regression, framework analysis**

I. Introduction

A STEM-based project usually involves multiple stages of thinking, reasoning, investigation, and execution where a group of people conduct background research, develop multiple ideas for solutions, develop and create a prototype, and then test, evaluate, and redesign them over a period of time. For a project that involves big data and various computing resources, one of the most critical component is the computational framework to produce the desired model that can be used for the advancement of the field and future analysis. Among the publicly available frameworks that facilitate the development of various types of models, those that deploy Deep Learning (DL) algorithms have experienced a surge in interest over the past decade due to many factors, including their successful application to various problems. DL methods comprised of multi-layer artificial neural networks have outperformed many state-of-the-art approaches in various fields for classification and regression problems. Such open-source implementations of DL algorithms have been adopted and shared in various frameworks that have been employed and improved upon by people in their respective fields of research for various applications including natural language processing, speech recognition, computer vision, and various domain-specific scenarios [1]–[9]. Several DL frameworks exist, such as Theano, Caffe, TensorFlow [10], Keras [11], PyTorch [12], etc., which use different techniques to optimize, parallelize and deploy models leading to differences in results. Hence, it is not straightforward to decide for a user which particular framework to use due to various options for implementation and limited studies to compare these implementations [13]–[15]. Moreover, the existing studies mainly focus on image classification problems and not on regression problems which are more common in physical sciences [8], [16]–[22]. In general, very few existing works have been devoted to studying the effectiveness of the different frameworks with respect to different datasets and different DL architectures for domain-specific fields. We believe that given a computational environment and dataset, effective and in-depth benchmarking of deep learning frameworks is required to understand the impact of the framework on the overall performance to better guide the domain scientist when applying them.

In this study, we aim to provide a comparison between popular open-source DL frameworks on regression-based problems in one of the fields of physical sciences – materials science – where various machine learning (ML) based techniques has greatly enhanced property prediction and materials discovery [17], [18], [23]–[32]. We compare three well-known DL frameworks by performing model training of deep neural network architectures on different materials datasets for the regression task of materials property prediction and compare them on the basis of performance and computational aspects, which includes the accuracy of the model, training/testing time, resource/memory consumption, and size of the saved model. We use Graphics Processing Units (GPU) for our computational environment as GPUs have been increasingly used for general-purpose computation that requires a highly data parallel architecture, such as deep learning computation.

II. Background

A. Model Input

Traditional approaches for training models incorporated domain knowledge based manually designed materials rep-

979-8-3503-2028-2/22/$31.00 ©2022 IEEE
DOI 10.1109/CSCI58124.2022.00069

TABLE I
DETAILED CONFIGURATIONS FOR DIFFERENT NETWORK ARCHITECTURES. THE NOTATION [...] REPRESENTS A STACK OF MODEL COMPONENTS, COMPRISING A SEQUENCE (WHERE, FC: FULLY CONNECTED LAYER, RE: RELU ACTIVATION FUNCTION). EACH SUCH STACK IS FOLLOWED BY A SHORTCUT CONNECTION IN THE CASE OF FCRN.

Output	FCN-10	FCRN-10	FCN-17	FCRN-17	FCN-24	FCRN-24
1024	[FC1024-Re x 2]	[FC1024-Re] x 2	[FC1024-Re x 4]	[FC1024-Re] x 4	[FC1024-Re x 4]	[FC1024-Re] x 4
512	[FC512-Re x 2]	[FC512-Re] x 2	[FC512-Re x 3]	[FC512-Re] x 3	[FC512-Re x 4]	[FC512-Re] x 4
256	[FC256-Re x 2]	[FC256-Re] x 2	[FC256-Re x 3]	[FC256-Re] x 3	[FC256-Re x 4]	[FC256-Re] x 4
128	[FC128-Re]	[FC128-Re]	[FC128-Re x 3]	[FC128-Re] x 3	[FC128-Re x 4]	[FC128-Re] x 4
64	[FC64-Re]	[FC64-Re]	[FC64-Re x 2]	[FC64-Re] x 2	[FC64-Re x 3]	[FC64-Re] x 3
32	[FC32-Re]	[FC32-Re]	[FC32-Re]	[FC32-Re]	[FC32-Re x 2]	[FC32-Re] x 2
16	-	-	-	-	[FC16-Re x 2]	[FC16-Re] x 2
1	FC1					

resentation for inputs [33]–[36]. However, these approaches tend to be more effective for a specific materials property and may not serve well as the general materials representation that can be used for model input for wide variety of materials problem dealing with classification and regression. Hence, In this work, we use composition based-attributes comprised of raw elemental fractions (EF) as input which are simple to calculate and have consistently shown to excel when used with powerful DL techniques [37], [38]. EF representation used in this work is composed of a 86-dimensional 1D vector where each attribute represents an element in the periodic table. For example, KCl is represented as a 1D vector of 86 numbers, with the K and Cl columns containing 0.5 and everything else as 0. From this, we can say that elemental fraction, when used as model input, can help demonstrate better generalization and facilitate the development of machine learning-based models on other datasets with different materials properties.

B. Materials Property

Materials discovery can be greatly accelerated by leveraging the continuously evolving data-driven approaches based on advanced ML/DL techniques where inputs such as composition-based attributes can be used to predict various materials properties by capturing the underlying knowledge effectively [37], [39], [40]. Materials property used for predictive modeling in this study is formation enthalpy (also known as formation energy) which is defined as the change in energy when one mole of a chemical compound in the standard state (1 atm of pressure and 298.15 K) is formed from its pure elements under the same conditions [41]. We can determine the stability of the material along with whether it is experimentally synthesizable in a laboratory using the value of the formation enthalpy. When more energy is produced in bond formation than needed for bond breaking then we have negative formation enthalpy and more negative formation enthalpy leads to more stable material. It is also shown in [38] that the knowledge learnt from training a model on formation enthalpy can help improve the predictive ability of the model trained on other materials properties. Materials databases also record various other properties such as total energy, magnetic moment, band gap, etc. [42]–[44].

III. METHOD

We next describe the architecture of the deep regression neural network models used in this work and the different deep learning frameworks used to build them.

A. Architecture

We create two types of neural networks FCN-n and FCRN-n ($n = 10, 17, 24$) to study the impact of frameworks.

1) FCN-n: Here, we create an n-layer neural network comprising of stacks of fully connected layers and ReLU as the activation function (except for the last layer which has linear activation) that we refer to as a fully connected network.

2) FCRN-n: We use FCN-n as our base network and add skip connections after each stack (comprised of a fully connected layer and ReLU) to introduce a neural network referred to as a fully connected residual network.

The detailed architecture of the networks used in this work is illustrated in Table I.

B. Deep Learning Framework

The frameworks considered in this study are Tensorflow 1 (TF-1), Tensorflow 2.2 integrated with Keras (TF-2), and PyTorch. These frameworks support multi-core CPUs and multiple GPUs. They also allow for performance optimization via the availability of flexible APIs (e.g. RESTful, TF Serving, Fast.ai), visualization tools (e.g. TensorBoard, Visdom), and configuration options (e.g.). TF-1, TF-2 and PyTorch models are saved in data, h5 and ckpt format respectively. Below we provide a brief overview of these frameworks:

1) Tensorflow: Tensorflow is an open-source library for large-scale machine learning and high performance computation across various platforms which includes CPU, GPU, and distributed computing. It is an open-source project released by the Google Brain team in 2015. Due to its parameter sharing and auto differentiation capabilities, TensorFlow can support different types of architectures. There is also a mechanism to support parallelism via parallel execution of the data flow graph model using multiple computational resources that collaborate to update shared parameters. Tensorflow is modeled as a directed acyclic graph for computation where operation is represented by nodes and deals with a multi-dimensional array called Tensors which are the values that flow along the edges of the graph. Given an operation, it can take zero or

TABLE II
SUMMARY OF THE MAIN FEATURES OF THE DEEP LEARNING FRAMEWORKS USED IN THIS STUDY.

	Tensorflow	**Keras**	**Pytorch**
Date of Release	2016	2015	2017
Developers	Google	Google	Facebook
Core Language	C++	Python, R	Python, C++
APIs	Python, C++	Python	Python
Model parallelism	Yes	Yes	Yes
Data parallelism	Yes	Yes	Yes
Multi-threaded CPU	Yes	Yes	Yes
Multi GPU	Yes	Yes	Yes
NVIDIA cuDNN	Yes	Yes	Yes
Fault Tolerance	Checkpoint & Recovery	Checkpoint & Resume	Checkpoint & Resume
Computational Graph Construction	Static Graph	Static/ Dynamic Graph	Dynamic Graph
Debugging	TF Debugger	TF/ Python Debugger	Python/ Pycharm/ Debugger

more tensors as input and in return give zero or more tensors as output. As long as the graph of which the operation is part of is valid, the operation will be valid.

2) Keras: Keras is also an open-source framework developed by the Google AI team members that is used to build neural networks. It can be considered a meta-framework that interacts and runs on top of other existing frameworks, such as Tensorflow and Theano. Keras is implemeted using Python and provides high-level APIs for developing neural networks models to perform deep learning. It relies on specialized libraries that serves as its backend engine instead of handling low level operations such as tensor manipulation and differentiation, and minimizes the number of actions required by a user for any specific action. One of the important feature of Keras is that it enables the users to implement their neural network models in the same way they would implement is on the base framework (such as Tensorflow and Theano) with ease and without sacrificing flexibility.

3) Pytorch: Pytorch is an open-source framework introduced by the Facebook AI research group which facilitates development of neural network based deep learning models through an easy to use API using Python-based framework. Python uses dynamic computation in contrary to static computation graphs (used by other popular deep learning frameworks) which allows greater flexibility when building highly complex architectures.

We summarize the main features of the frameworks used in our study in Table II.

IV. EMPIRICAL EVALUATION

We present a detailed analysis of the design and evaluation of the problem discussed in this work. First, we perform the evaluation of the selected frameworks by comparing the MAEs using elemental fractions as model input to predict formation enthalpy. Next, we perform time evaluation of the frameworks by looking at the training time to perform model training and test time to perform prediction. Finally, we perform computational evaluation for the models by checking the GPU memory usage, model parameters and size of the saved model. Before presenting the results, we discuss the experimental settings and datasets used in this work.

TABLE III
DATASETS USED IN THIS WORK.

Dataset	**Data Size**
OQMD [42]	345220
MP [43]	89181
JARVIS [44]	28171

1) Experimental Settings: We implement deep learning models with Python using Tensorflow 1 (TF-1), Tensorflow 2.2 integrated with Keras (TF-2), and PyTorch framework. In this study, we use six neural networks FCN-n and FCRN-n ($n = 10, 17, 24$), where the number of neurons in each layers are inspired from [37], [45], [46]. We used mean absolute error (MAE) as loss function and the error metric for all the results, Adam as the optimizer with a mini-batch size of 32, learning rate as 0.0001. We used early stopping with patience of 100 which stops the training if the validation loss does not improve for 100 epochs. Note that all model training, evaluation, and benchmarking for FCN-n and FCRN-n ($n = 10, 17, 24$) was conducted on a single NVIDIA GV100GL GPU (Tesla V100-PCIe) with 16GB of memory.

2) Datasets: The three datasets used for evaluation are shown in Table III.

OQMD was downloaded from the database website [42] whereas MP and JARVIS datasets were obtained using Matminer [47]. We perform formation enthalpy prediction from an input vector composed of 86 composition-based elemental fractions. The datasets are randomly split with stratification based on the number of elements in a compound with a fixed random seed of 1234567 into training, validation, and test sets in the ratio of 81:9:10. For each dataset, we perform five runs each using different frameworks by initializing the network weights randomly with a random seed of 1234567 for fixed random initialization and with random seeds of 1, 12, 123, 1234, 1234567 for the different random initializations and report the results.

A. Accuracy Analysis

First, we perform accuracy analysis where we compare the test MAEs of the three frameworks discussed in this work by using formation enthalpy as the materials property when model trained on OQMD, MP and JARVIS dataset using each of the deep neural networks with EF as model input.

Table IV shows the accuracy of the three frameworks using FCN-n and FCRN-n ($n = 10, 17, 24$) across three datasets. We derive the following insights from Table IV.

- **Accuracy** TF-2 performs the best with least MAE on the test set for all the model-dataset combinations making it

TABLE IV

PREDICTION PERFORMANCE BENCHMARKING FOR THE PREDICTION TASK "ACCURACY ANALYSIS" AND "TIME ANALYSIS" FOR DIFFERENT FRAMEWORKS USING MULTIPLE RUNS FOR FIXED RANDOM INITIALIZATION AND DIFFERENT RANDOM INITIALIZATIONS. THE AVERAGE AND STANDARD DEVIATION VALUES ARE REPORTED ACROSS FIVE RUNS. THE TEST TIME IS CALCULATED PER 1000 ENTRIES.

Data Set	**Model**	**Framework**	Fixed Random Initialization			Different Random Initializations		
(Data Size)			**MAE (eV/atom)**	**Training Time (s)**	**Test Time (s)**	**MAE (eV/atom)**	**Training Time (s)**	**Test Time (s)**
OQMD	FCN-10	TF-1	0.0509 ± 0.0007	28016 ± 7276	0.0788 ± 0.0029	0.0512 ± 0.0007	26260 ± 6485	0.0804 ± 0.0054
(345134)		TF-2	**0.0479 ± 0.0003**	24709 ± 5927	0.0769 ± 0.0027	**0.0475 ± 0.0003**	**23852 ± 3547**	0.0776 ± 0.0064
		PyTorch	0.0557 ± 0.0000	**21854 ± 433**	**0.0385 ± 0.0011**	0.0565 ± 0.0012	25116 ± 5890	**0.0391 ± 0.0043**
	FCRN-10	TF-1	0.0450 ± 0.0004	40666 ± 5550	0.1082 ± 0.0016	0.0456 ± 0.0004	37015 ± 6313	0.1087 ± 0.0041
		TF-2	**0.0427 ± 0.0003**	**39912 ± 6340**	0.1016 ± 0.0043	**0.0427 ± 0.0005**	**36205 ± 9015**	0.0963 ± 0.0042
		PyTorch	0.0483 ± 0.0000	48440 ± 1301	**0.0458 ± 0.0055**	0.0491 ± 0.0022	36375 ± 7927	**0.0430 ± 0.0041**
	FCN-17	TF-1	0.0503 ± 0.0002	34935 ± 10301	0.1128 ± 0.0070	0.0503 ± 0.0004	35360 ± 4898	0.1236 ± 0.0113
		TF-2	**0.0469 ± 0.0004**	**30534 ± 3593**	0.0943 ± 0.0066	**0.0463 ± 0.0004**	31768 ± 5002	0.0995 ± 0.0048
		PyTorch	0.0552 ± 0.0000	42968 ± 4982	**0.0601 ± 0.0176**	0.0567 ± 0.0020	**29429 ± 4726**	**0.0522 ± 0.0068**
	FCRN-17	TF-1	0.0447 ± 0.0006	46450 ± 6405	0.1666 ± 0.0098	0.0455 ± 0.0004	**36688 ± 9148**	0.1845 ± 0.0174
		TF-2	**0.0424 ± 0.0002**	**35447 ± 5380**	0.1068 ± 0.0125	**0.0419 ± 0.0006**	50018 ± 13699	0.1214 ± 0.0058
		PyTorch	0.0486 ± 0.0000	47633 ± 6537	**0.0721 ± 0.0162**	0.0487 ± 0.0036	45694 ± 11580	**0.0666 ± 0.0071**
	FCN-24	TF-1	0.0511 ± 0.0005	**33623 ± 8469**	0.1495 ± 0.0097	0.0513 ± 0.0004	**32495 ± 7782**	0.1549 ± 0.0106
		TF-2	**0.0466 ± 0.0005**	37283 ± 4665	0.1200 ± 0.0059	**0.0470 ± 0.0002**	36647 ± 6692	0.1198 ± 0.0066
		PyTorch	0.0548 ± 0.0000	80016 ± 1741	**0.0665 ± 0.0026**	0.0553 ± 0.0017	58906 ± 25381	**0.0583 ± 0.0093**
	FCRN-24	TF-1	0.0447 ± 0.0006	53741 ± 16692	0.2158 ± 0.0017	0.0447 ± 0.0003	52988 ± 11556	0.2165 ± 0.0031
		TF-2	**0.0423 ± 0.0004**	54688 ± 11133	0.1475 ± 0.0089	**0.0424 ± 0.0005**	55896 ± 14351	0.1437 ± 0.0068
		PyTorch	0.0484 ± 0.0000	**51904 ± 770**	**0.0870 ± 0.0015**	0.0485 ± 0.0037	**50065 ± 6987**	**0.0715 ± 0.0101**
MP	FCN-10	TF-1	0.1199 ± 0.0020	**2109 ± 353**	0.0950 ± 0.0033	0.1209 ± 0.0028	**2536 ± 568**	0.0926 ± 0.0062
(89181)		TF-2	**0.1176 ± 0.0029**	2850 ± 470	0.0806 ± 0.0047	**0.1156 ± 0.0005**	2997 ± 543	0.0834 ± 0.0030
		PyTorch	0.1230 ± 0.0000	2571 ± 57	**0.0589 ± 0.0009**	0.1241 ± 0.0028	2704 ± 444	**0.0594 ± 0.0007**
	FCRN-10	TF-1	0.1182 ± 0.0020	**3846 ± 1010**	0.1191 ± 0.0011	0.1194 ± 0.0021	**3235 ± 323**	0.1194 ± 0.0026
		TF-2	**0.1132 ± 0.0012**	5147 ± 637	0.1031 ± 0.0037	**0.1130 ± 0.0015**	4595 ± 1082	0.1020 ± 0.0087
		PyTorch	0.1185 ± 0.0000	5356 ± 53	**0.0841 ± 0.0051**	0.1203 ± 0.0031	5314 ± 1173	**0.0517 ± 0.0149**
	FCN-17	TF-1	0.1228 ± 0.0007	**3210 ± 624**	**0.1262 ± 0.0055**	0.1228 ± 0.0015	**3110 ± 766**	0.1406 ± 0.0100
		TF-2	**0.1187 ± 0.0012**	4645 ± 545	0.1403 ± 0.0232	**0.1182 ± 0.0022**	4157 ± 380	0.1069 ± 0.0151
		PyTorch	0.1257 ± 0.0000	6202 ± 775	0.1363 ± 0.0699	0.1242 ± 0.0023	5183 ± 1086	**0.1038 ± 0.0112**
	FCRN-17	TF-1	0.1202 ± 0.0017	**5257 ± 1659**	0.1830 ± 0.0057	0.1184 ± 0.0019	**5299 ± 1507**	0.1921 ± 0.0106
		TF-2	**0.1129 ± 0.0016**	7517 ± 1952	0.1368 ± 0.0099	**0.1130 ± 0.0024**	6801 ± 2032	0.1231 ± 0.0082
		PyTorch	0.1130 ± 0.0000	12092 ± 1059	**0.1362 ± 0.0302**	0.1180 ± 0.0031	7037 ± 2524	**0.1225 ± 0.0259**
	FCN-24	TF-1	0.1223 ± 0.0017	**2910 ± 333**	0.1305 ± 0.0658	0.1213 ± 0.0014	**3402 ± 682**	0.1360 ± 0.0002
		TF-2	**0.1166 ± 0.0014**	5797 ± 1429	0.1210 ± 0.0100	**0.1174 ± 0.0016**	5554 ± 657	0.1177 ± 0.0018
		PyTorch	0.1220 ± 0.0000	5938 ± 125	**0.1142 ± 0.0021**	0.1243 ± 0.0034	7317 ± 2275	**0.0696 ± 0.0246**
	FCRN-24	TF-1	0.1197 ± 0.0014	**5018 ± 719**	0.1537 ± 0.0899	0.1190 ± 0.0010	**6412 ± 1889**	0.2224 ± 0.0056
		TF-2	**0.1138 ± 0.0022**	8979 ± 1531	0.1528 ± 0.0114	**0.1140 ± 0.0022**	7543 ± 2058	0.1561 ± 0.0073
		PyTorch	0.1166 ± 0.0000	9691 ± 134	**0.1424 ± 0.0041**	0.1203 ± 0.0023	9068 ± 3167	**0.0888 ± 0.0337**
JARVIS	FCN-10	TF-1	0.0817 ± 0.0012	**620 ± 138**	0.1308 ± 0.0044	0.0813 ± 0.0014	**649 ± 141**	0.1287 ± 0.0028
(19994)		TF-2	**0.0790 ± 0.0009**	888 ± 135	**0.1022 ± 0.0082**	**0.0790 ± 0.0017**	1214 ± 268	**0.0975 ± 0.0064**
		PyTorch	0.1015 ± 0.0000	1127 ± 14	0.1669 ± 0.0036	0.0998 ± 0.0104	1078 ± 314	0.1668 ± 0.0015
	FCRN-10	TF-1	0.0758 ± 0.0014	**1261 ± 323**	0.1603 ± 0.0083	0.0752 ± 0.0013	**1183 ± 126**	0.1603 ± 0.0038
		TF-2	**0.0714 ± 0.0011**	1891 ± 280	**0.1162 ± 0.0048**	**0.0726 ± 0.0006**	1679 ± 270	0.1086 ± 0.0096
		PyTorch	0.0731 ± 0.0000	1949 ± 38	0.2199 ± 0.0014	0.0759 ± 0.0044	1801 ± 190	**0.0976 ± 0.0685**
	FCN-17	TF-1	0.0839 ± 0.0019	930 ± 244	0.1969 ± 0.0266	0.0846 ± 0.0013	**947 ± 364**	0.1933 ± 0.0095
		TF-2	**0.0797 ± 0.0015**	1711 ± 305	**0.1849 ± 0.0585**	**0.0809 ± 0.0007**	1250 ± 391	**0.1277 ± 0.0321**
		PyTorch	0.1652 ± 0.0000	**807 ± 173**	0.2381 ± 0.0921	0.1234 ± 0.0305	1096 ± 286	0.2791 ± 0.0748
	FCRN-17	TF-1	0.0753 ± 0.0012	**1391 ± 267**	0.2350 ± 0.0115	0.0757 ± 0.0016	**1409 ± 237**	**0.0741 ± 0.0805**
		TF-2	**0.0706 ± 0.0013**	2625 ± 466	**0.2285 ± 0.0378**	**0.0710 ± 0.0012**	2386 ± 675	0.1571 ± 0.0318
		PyTorch	0.0792 ± 0.0000	1676 ± 91	0.3983 ± 0.0199	0.0767 ± 0.0043	2098 ± 503	0.1766 ± 0.0155
	FCN-24	TF-1	0.0825 ± 0.0012	**890 ± 332**	0.2143 ± 0.0079	0.0853 ± 0.0011	**650 ± 83**	0.2115 ± 0.0077
		TF-2	**0.0805 ± 0.0015**	1695 ± 588	**0.1318 ± 0.0074**	**0.0799 ± 0.0020**	1735 ± 585	**0.1328 ± 0.0053**
		PyTorch	0.1025 ± 0.0000	1567 ± 51	0.3447 ± 0.0075	0.1141 ± 0.0232	1473 ± 245	0.1417 ± 0.1130
	FCRN-24	TF-1	0.0750 ± 0.0025	**1790 ± 431**	0.2746 ± 0.0081	0.0752 ± 0.0013	**1820 ± 267**	0.2783 ± 0.0024
		TF-2	**0.0718 ± 0.0008**	3384 ± 866	**0.1695 ± 0.0039**	**0.0721 ± 0.0008**	2820 ± 1145	**0.1729 ± 0.0029**
		PyTorch	0.0770 ± 0.0000	2612 ± 28	0.4041 ± 0.0049	0.0802 ± 0.0078	2302 ± 314	0.1743 ± 0.1269

the most useful framework for the materials property prediction in terms of accuracy in this analysis. In general, PyTorch was found to be least accurate among the three frameworks.

- **Reproducibility** Interestingly, we found that PyTorch is fully reproducible (zero standard deviation in MAE), whereas TF tends to show some variation, although it is quite small. We believe that random seed in PyTorch (seed_everything) is able to make the training deterministic by eliminating stochasticity altogether, which helps produce the same result across multiple runs on a given input data. On the other hand, random seed used in TF only performs weight initialization, and it is possible that some internal routines (especially those run on GPUs) still have some stochasticity. There are ongoing efforts and libraries [48] that eventually aim to fix the reproducibility related issues in TF.

B. Time Analysis

Next, we perform time analysis where we compare the training time to perform model training and test time to perform prediction by the three frameworks discussed in this work by using formation enthalpy as the materials property when model trained on OQMD, MP and JARVIS dataset using

TABLE V
PREDICTION PERFORMANCE BENCHMARKING FOR THE PREDICTION TASK "COMPUTATIONAL ANALYSIS" FOR DIFFERENT FRAMEWORKS.

Model	Framework	# Model Size (MiB)	GPU Memory Usage (MiB)	# Model Params
FCN-10	TF-1	24.8	15265	2166529
	TF-2	8.3	15265	2166529
	PyTorch	24.8	1124	2166529
FCRN-10	TF-1	33.8	15265	2954977
	TF-2	11.3	15265	2954977
	PyTorch	33.8	1128	2954977
FCN-17	TF-1	53.0	15265	4631361
	TF-2	17.7	15265	4631361
	PyTorch	53.0	1157	4631361
FCRN-17	TF-1	62.0	15265	5419809
	TF-2	20.8	15265	5419809
	PyTorch	62.0	1161	5419809
FCN-24	TF-1	57.0	15265	4982321
	TF-2	19.1	15265	4982321
	PyTorch	57.0	1176	4982321
FCRN-24	TF-1	66.1	15265	5771297
	TF-2	22.2	15265	5771297
	PyTorch	66.1	1180	5771297

each of the the deep neural networks, FCN-n and FCRN-n ($n = 10, 17, 24$) with EF as model input.

Table IV shows the training time to perform model training and test time to perform prediction per 1000 entries of the three frameworks using FCN-n and FCRN-n ($n = 10, 17, 24$) across three datasets. We derive the following insights from Table IV.

- **Training time** We observe that in general the training time for TF is faster as compared to Pytorch. However for large datasets, for some of the cases, we observe that the model training with PyTorch is faster than with TF.
- **Testing time** We observe that in general the testing time with Pytorch is faster as compared to TF. However, when performing model testing with complex architecture and small datasets, we observe that the model testing with TF is faster than with Pytorch.

C. Computational Analysis

Finally, we perform computational analysis by checking the GPU memory usage, model parameters and size of the saved model by the three frameworks discussed in this work by using formation enthalpy as the materials property when the model trained using each of the the deep neural networks, FCN-n and FCRN-n ($n = 10, 17, 24$) with EF as model input.

Table V shows the GPU memory usage, model parameters and size of the saved model of the three frameworks using FCN-n and FCRN-n ($n = 10, 17, 24$). We derive the following insights from Table V.

- **Model Size** The saved model from TF-2 is less than one-third in size as compared to TF-1 and PyTorch.
- **GPU Usage** TF1 and TF2 tend to use all the memory available in a GPU to perform the model training whereas PyTorch uses only the required amount of GPU memory for the same.
- **Model Parameters** For a given architecture, the number of parameters is constant across all the frameworks, which is expected.

After comparing the accuracy, training time and model parameters from Table IV and V respectively, we also observe that large number of model parameters does not always lead to higher accuracy but it tends to increase the training time.

V. CONCLUSION

The main objective of this work was to provide a comparison between three of the most well-known DL frameworks: TF-1, TF-2, and PyTorch on six different deep neural network architectures for regression problems in materials science using three datasets with varying data sizes. Several important insights were obtained w.r.t. the model accuracy, training/testing times, reproducibility, and memory usage. We observe that in general TF-2 gave the best test MAE but Pytorch is most promising when the aim is reproducibility. The training and testing time for the frameworks varies with the dataset size and the size of the saved model. GPU usage and model parameters for a given architecture are independent of the dataset used for model training. We believe that these insights would be interesting to deep learning practitioners to help make more informed decisions about which deep learning framework to use for a given problem, data, and available computational resources.

ACKNOWLEDGMENT

This work was performed under the following financial assistance award 70NANB19H005 from U.S. Department of Commerce, National Institute of Standards and Technology as part of the Center for Hierarchical Materials Design (CHiMaD). Partial support is also acknowledged from NSF award CMMI-2053929, and DOE awards DE-SC0019358, DE-SC0021399, and Northwestern Center for Nanocombinatorics.

REFERENCES

[1] A. Abugabah, A. A. AlZubi, F. Al-Obeidat, A. Alarifi, and A. Alwadain, "Data mining techniques for analyzing healthcare conditions of urban space-person lung using meta-heuristic optimized neural networks," *Cluster Computing*, vol. 23, no. 3, pp. 1781–1794, 2020.

[2] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, no. ARTICLE, pp. 2493–2537, 2011.

[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[4] T. Ohki, V. Gupta, and M. Nishigaki, "Efficient spoofing attack detection against unknown sample using end-to-end anomaly detection," in *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 2019, pp. 224–230.

[5] Z. Jiang and S. Gao, "An intelligent recommendation approach for online advertising based on hybrid deep neural network and parallel computing," *Cluster Computing*, vol. 23, no. 3, pp. 1987–2000, 2020.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[7] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[8] A. Agrawal and A. Choudhary, "Deep materials informatics: Applications of deep learning in materials science," *MRS Communications*, vol. 9, no. 3, pp. 779–792, 2019.

[9] K. Choudhary, B. DeCost, C. Chen, A. Jain, F. Tavazza, R. Cohn, C. W. Park, A. Choudhary, A. Agrawal, S. J. Billinge *et al.*, "Recent advances and applications of deep learning methods in materials science," *npj Computational Materials*, vol. 8, no. 1, pp. 1–26, 2022.

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[11] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[13] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *2016 7th International Conference on Cloud Computing and Big Data*. IEEE, 2016, pp. 99–104.

[14] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of caffe, neon, theano, and torch for deep learning," 2016.

[15] R. Elshawi, A. Wahab, A. Barnawi, and S. Sakr, "Dlbench: a comprehensive experimental evaluation of deep learning frameworks," *Cluster Computing*, pp. 1–22, 2021.

[16] A. Agrawal and A. Choudhary, "Perspective: Materials informatics and big data: Realization of the "fourth paradigm" of science in materials science," *APL Materials*, vol. 4, no. 5, p. 053208, 2016.

[17] R. Pollice, G. dos Passos Gomes, M. Aldeghi, R. J. Hickman, M. Krenn, C. Lavigne, M. Lindner-D'Addario, A. Nigam, C. T. Ser, Z. Yao *et al.*, "Data-driven strategies for accelerated materials design," *Accounts of Chemical Research*, vol. 54, no. 4, pp. 849–860, 2021.

[18] J. Westermayr, M. Gastegger, K. T. Schütt, and R. J. Maurer, "Perspective on integrating machine learning into computational chemistry and materials science," *The Journal of Chemical Physics*, vol. 154, no. 23, p. 230903, 2021.

[19] K. Choudhary and B. DeCost, "Atomistic line graph neural network for improved materials property predictions," *npj Computational Materials*, vol. 7, no. 1, pp. 1–8, 2021.

[20] D. Jha, V. Gupta, W.-k. Liao, A. Choudhary, and A. Agrawal, "Moving closer to experimental level materials property prediction using ai," *Scientific reports*, vol. 12, 2022.

[21] Y. Mao, H. Lin, C. X. Yu, R. Frye, D. Beckett, K. Anderson, L. Jacquemetton, F. Carter, Z. Gao, W.-k. Liao *et al.*, "A deep learning framework for layer-wise porosity prediction in metal powder bed fusion using thermal signatures," *Journal of Intelligent Manufacturing*, pp. 1–15, 2022.

[22] Y. Mao, Z. Yang, D. Jha, A. Paul, W.-k. Liao, A. Choudhary, and A. Agrawal, "Generative adversarial networks and mixture density networks-based inverse modeling for microstructural materials design," *Integrating Materials and Manufacturing Innovation*, pp. 1–11, 2022.

[23] A. Seko, H. Hayashi, K. Nakayama, A. Takahashi, and I. Tanaka, "Representation of compounds for machine-learning prediction of physical properties," *Physical Review B*, vol. 95, no. 14, p. 144110, 2017.

[24] D. P. Tabor, L. M. Roch, S. K. Saikin, C. Kreisbeck, D. Sheberla, J. H. Montoya, S. Dwaraknath, M. Aykol, C. Ortiz, H. Tribukait *et al.*, "Accelerating the discovery of materials for clean energy in the era of smart automation," *Nature Reviews Materials*, vol. 3, no. 5, pp. 5–20, 2018.

[25] K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev, and A. Walsh, "Machine learning for molecular and materials science," *Nature*, vol. 559, no. 7715, pp. 547–555, 2018.

[26] B. Sanchez-Lengeling and A. Aspuru-Guzik, "Inverse molecular design using machine learning: Generative models for matter engineering," *Science*, vol. 361, no. 6400, pp. 360–365, 2018.

[27] G. Pilania, "Machine learning in materials science: From explainable predictions to autonomous design," *Computational Materials Science*, vol. 193, p. 110360, 2021.

[28] Z. Qin, G. S. Jung, M. J. Kang, and M. J. Buehler, "The mechanics and design of a lightweight three-dimensional graphene assembly," *Science advances*, vol. 3, no. 1, p. e1601536, 2017.

[29] N. Nusran, K. R. Joshi, K. Cho, M. A. Tanatar, W. R. Meier, S. Bud'ko, P. C. Canfield, Y. Liu, T. A. Lograsso, and R. Prozorov, "Spatially-resolved study of the meissner effect in superconductors using nv-centers-in-diamond optical magnetometry," *New Journal of Physics*, vol. 20, no. 4, p. 043010, 2018.

[30] D. Morgan and R. Jacobs, "Opportunities and challenges for machine learning in materials science," *Annual Review of Materials Research*, vol. 50, pp. 71–103, 2020.

[31] A. Mannodi-Kanakkithodi and M. K. Chan, "Computational data-driven materials discovery," *Trends in Chemistry*, vol. 3, pp. 79–82, 2021.

[32] P. Friederich, F. Häse, J. Proppe, and A. Aspuru-Guzik, "Machine-learned potentials for next-generation matter simulations," *Nature Materials*, vol. 20, no. 6, pp. 750–761, 2021.

[33] Y. Saad, D. Gao, T. Ngo, S. Bobbitt, J. R. Chelikowsky, and W. Andreoni, "Data mining for materials: Computational experiments with a b compounds," *Physical Review B*, vol. 85, no. 10, p. 104104, 2012.

[34] L. M. Ghiringhelli, J. Vybiral, S. V. Levchenko, C. Draxl, and M. Scheffler, "Big data of materials science: Critical role of the descriptor," *Physical Review Letters*, vol. 114, no. 10, p. 105503, 2015.

[35] J. Lee, A. Seko, K. Shitara, K. Nakayama, and I. Tanaka, "Prediction model of band gap for inorganic compounds by combination of density functional theory calculations and machine learning techniques," *Physical Review B*, vol. 93, no. 11, p. 115104, 2016.

[36] A. D. Sendek, Q. Yang, E. D. Cubuk, K.-A. N. Duerloo, Y. Cui, and E. J. Reed, "Holistic computational structure screening of more than 12000 candidates for solid lithium-ion conductor materials," *Energy & Environmental Science*, vol. 10, no. 1, pp. 306–320, 2017.

[37] D. Jha, L. Ward, A. Paul, W.-k. Liao, A. Choudhary, C. Wolverton, and A. Agrawal, "ElemNet: Deep learning the chemistry of materials from only elemental composition," *Scientific reports*, vol. 8, no. 1, p. 17593, 2018.

[38] V. Gupta, K. Choudhary, F. Tavazza, C. Campbell, W.-k. Liao, A. Choudhary, and A. Agrawal, "Cross-property deep transfer learning framework for enhanced predictive analytics on small materials data," *Nature communications*, vol. 12, no. 1, pp. 1–10, 2021.

[39] R. E. Goodall and A. A. Lee, "Predicting materials properties without crystal structure: Deep representation learning from stoichiometry," *Nature communications*, vol. 11, no. 1, pp. 1–9, 2020.

[40] A. Y.-T. Wang, S. K. Kauwe, R. J. Murdock, and T. D. Sparks, "Compositionally restricted attention-based network for materials property predictions," *npj Computational Materials*, vol. 7, no. 1, pp. 1–10, 2021.

[41] D. W. Oxtoby, H. P. Gillis, and L. J. Butler, *Principles of modern chemistry*. Cengage Learning, 2015.

[42] S. Kirklin, J. E. Saal, B. Meredig, A. Thompson, J. W. Doak, M. Aykol, S. Rühl, and C. Wolverton, "The open quantum materials database (oqmd): assessing the accuracy of dft formation energies," *npj Computational Materials*, vol. 1, p. 15010, 2015.

[43] A. Jain, S. P. Ong, G. Hautier, W. Chen, W. D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, and K. a. Persson, "The Materials Project: A materials genome approach to accelerating materials innovation," *APL Materials*, vol. 1, no. 1, p. 011002, 2013. [Online]. Available: http://link.aip.org/link/AMPADS/v1/i1/p011002/s1\&Agg=doi

[44] K. Choudhary, K. F. Garrity, A. C. E. Reid, B. DeCost, A. J. Biacchi, A. R. H. Walker, Z. Trautt, J. Hattrick-Simpers, A. G. Kusne, A. Centrone, A. Davydov, J. Jiang, R. Pachter, G. Cheon, E. Reed, A. Agrawal, X. Qian, V. Sharma, H. Zhuang, S. V. Kalinin, B. G. Sumpter, G. Pilania, P. Acar, S. Mandal, K. Haule, D. Vanderbilt, K. Rabe, and F. Tavazza, "JARVIS: An integrated infrastructure for data-driven materials design," 2020.

[45] D. Jha, V. Gupta, L. Ward, Z. Yang, C. Wolverton, I. Foster, W.-k. Liao, A. Choudhary, and A. Agrawal, "Enabling deeper learning on big data for materials informatics applications," *Scientific reports*, vol. 11, no. 1, pp. 1–12, 2021.

[46] V. Gupta, W.-k. Liao, A. Choudhary, and A. Agrawal, "Brnet: Branched residual network for fast and accurate predictive modeling of materials properties," in *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*. SIAM, 2022, pp. 343–351.

[47] L. T. Ward, A. R. Dunn, A. Faghaninia, N. E. R. Zimmermann, S. Bajaj, Q. Wang, J. E. P. Montoya, J. Chen, K. Bystrom, M. T. Dylla, K. Chard, M. Asta, K. A. Persson, G. J. Snyder, I. T. Foster, and A. Jain, "Matminer: An open source toolkit for materials data mining," *Computational Materials Science*, vol. 152, pp. 60–69, 2018.

[48] B. B. Christoph Angerer *et al.*, "Tensorflow determinism," https://github.com/NVIDIA/framework-determinism, 2019.