

**OPTIMIZING AND SCALING MACHINE LEARNING MODELS FOR
SCIENTIFIC APPLICATIONS ON EXASCALE SUPERCOMPUTERS**

by

Vineeth Gutta

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer and Information Sciences

Winter 2025

© 2025 Vineeth Gutta
All Rights Reserved

**OPTIMIZING AND SCALING MACHINE LEARNING MODELS FOR
SCIENTIFIC APPLICATIONS ON EXASCALE SUPERCOMPUTERS**

by

Vineeth Gutta

Approved: _____

Dr. Weisong Shi, Ph.D.

Chair of the Department of Computer and Information Sciences

Approved: _____

Dr. Jamie D. Phillips, Ph.D.

Interim Dean of the College of Engineering

Approved: _____

Louis F. Rossi, Ph.D.

Vice Provost for Graduate and Professional Education and
Dean of the Graduate College

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Dr. Sunita Chandrasekaran, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Dr. Ulf Schiller, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Dr. Matthew Louis Mauriello, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Dr. Eric Stahlberg, Ph.D.
Member of dissertation committee

ACKNOWLEDGEMENTS

I would like to acknowledge my family for being immensely supportive, patient, and understanding throughout my PhD. And this work would not have been possible without the guidance, unwavering support, and mentorship of my advisor Dr. Sunita Chandrasekaran. I'm thankful to my committee members Dr. Ulf Shiller, Dr. Matthew Mauriello, and Dr. Eric Stahlberg for their time, thoughtful feedback, advice, and constructive criticism. I'm Thankful to all CRPL members, current and former, who made the journey more rewarding. I would also like to acknowledge the Frederick National lab for Cancer Research and personally thank Dr. Eric Stahlberg for providing the financial support for my PhD over the last 4 years.

TABLE OF CONTENTS

LIST OF TABLES	ix
LIST OF FIGURES	xi
ABSTRACT	xiv
 Chapter	
1 INTRODUCTION	1
1.1 Research Questions	3
1.2 Motivation	3
1.3 Thesis Organization	4
1.4 Thesis Contributions	5
2 EXPLORING PORTABILITY OF DL MODELS	6
2.1 Introduction	6
2.2 Motivation	8
2.3 Background of DL frameworks' portability and previous work	10
2.3.1 Machine learning frameworks	10
2.3.2 Open Neural Network Exchange (ONNX)	13
2.3.3 ONNX Runtime	13
2.3.4 TVM	14
2.3.5 Containerization	14
2.4 Experiments and results of frameworks ported	16
2.4.1 Experimental Setup	18
2.4.2 Results	23
2.5 Chapter Summary	25
2.6 Lessons Learned	26

3	CANCER DRUG RESPONSE MODELS	27
3.1	Porting cancer applications to UDel’s DARWIN and non-DOE systems	27
3.1.1	ATOM Modeling Pipeline (AMPL) overview	27
3.1.2	Cancer Distributed Learning Environment (CANDLE) Benchmarks	28
3.1.3	ARM-based Accelerators	29
3.1.4	Accelerate NCI-DOE resources on next-gen NVIDIA A100s	29
3.2	Adapt models to tissue data	30
3.2.1	Combination drug response predictor (Combo)	32
3.2.2	Methodology	33
3.2.3	Data gathering	33
3.2.4	Testing CANDLE Pilot 1 models on tumor-normal tissue data	34
3.3	Improving CANDLE Pilot 1 models	36
3.3.1	Single drug response models	36
3.3.1.1	Cancer Cell Line Encyclopedia (CCLE) data preprocessing	38
3.3.1.2	CCLE results	42
3.3.2	Tree-based models	43
3.3.2.1	Data preprocessing	43
3.4	Initial experiment	44
3.4.1	Train on CCLE data and test on NCI60 data	45
3.4.1.1	Change from growth to concentration as predictor variable	46
3.4.2	Experimenting role of drug descriptors as features	46
3.4.3	New gene expression data	47
3.4.4	Multi-output regression models	47

3.4.5	Computational metrics	50
3.4.5.1	Convolutional Neural Networks (CNN) results	51
3.4.6	FDA approved drug only models	54
3.4.6.1	Results across several performance metrics	55
3.4.7	Focus on NCI60 data	57
3.4.8	Autoencoders for encoding features	57
3.5	Lessons Learned	58
4	A NOVEL UTILITY FOR COMPARING NEURAL NET AND TREE-BASED MODELS	60
4.1	Introduction	61
4.1.1	Background	61
4.2	Design and Implementation	64
4.2.1	The sources, type, and format of data sets and the changes made for this study	66
4.3	Results	68
4.3.1	Experimental Setup	68
4.3.2	XGBoost	68
4.3.3	CNN	69
4.3.4	Training times of CNN and XGBoost	70
4.3.4.1	Training of full NCI60 drugs	70
4.3.4.2	Training on FDA approved drugs	72
4.4	Lessons Learned	74
5	SCALING ML SURROGATE MODELS FOR PLASMA PHYSICS SIMULATIONS ON FRONTIER EXASCALE SYSTEM	75
5.1	Challenges of creating a software stack for new hardware accelerators	76
5.2	Frontier exascale supercomputer	77
5.3	Background	77

5.4	Particle-In-Cell on GPU (PConGPU) as an application test case . . .	78
5.4.1	Workflow setup	83
5.5	Scaling ML on AMD GPUs	83
5.5.1	Use of PyTorch DistributedDataParallel (DDP) for scaling . . .	85
5.5.2	cINN model with static data	86
5.6	Scaling model training	88
5.6.1	Challenges faced while scaling on Frontier	89
5.7	Main takeaways of PConGPU in-transit ML	90
5.8	Lessons Learned	91
6	IMPROVING SURROGATE MODELS IN DRUG DISCOVERY	93
6.1	Background of IMPECCABLE	93
6.1.1	IMPECCABLE: AI-enabled drug discovery campaign	94
6.1.2	Ligand Pose Optimizer Model (LPOM)	97
6.1.3	Simplified Lennard-Jones Potential Descriptors	99
6.1.4	Geometric Descriptors	99
6.1.5	Molecular Descriptors	100
6.1.6	Description of training data	101
6.2	Improving Ligand Pose Optimization Model	102
6.2.1	Hyperparameter optimization (HPO) and its implications . . .	103
6.2.2	Combining geometric and molecular descriptors	105
6.2.3	Testing separate validation compounds	107
6.2.4	Data reduction	107
6.2.5	A discussion on parallel HPO scaling	110
6.3	Lessons Learned	111
7	CONCLUSION	112
7.1	Research Outcomes and Key Takeaways	112
7.2	Thesis Contributions	116
	BIBLIOGRAPHY	117

LIST OF TABLES

2.1	Systems used to conduct experiments in this study	20
2.2	Specifications of GPUs used during this study	21
2.3	ML models and their inference latencies across CPUs and GPUs . .	23
3.1	K-fold cross validation	44
3.2	Hyperparameter Optimization Values Tested	44
3.3	Results: Mean test scores for combinations of hyperparameters . . .	44
3.4	Initial Data Format	48
3.5	New Data Format	48
3.6	K-fold cross validation	54
3.7	Results using XGBoost on CPUs in Seconds	56
3.8	Autoencoder Results	58
4.1	XGBoost Errors for Model Trained on NCI60 Data	69
4.2	XGBoost Errors for Model Trained on NCI60 Data with Only FDA Approved Drugs List	69
4.3	CNN errors for NCI60 after training after performing HPO	70
4.4	Results using XGBoost. Times for threads represents model runs on CPU with the corresponding threads used for speedup. Last row corresponds to running the same model on single NVIDIA V100 GPU	71
4.5	CNN model trained using all 30,000 drugs as features on an NVIDIA V100 GPU	71

4.6	CNN model trained with all features on a CPU	72
4.7	Fastest training times for CNN and XGBoost on CPU and GPU (all features)	72
4.8	Results using XGBoost. Times for threads represents model runs on CPU with the corresponding threads used for speedup. Last row corresponds to running the same model on single NVIDIA V100 GPU	73
4.9	CNN model FDA drugs trained on a CPU	73
4.10	CNN model FDA drugs trained on a V100 GPU	73
4.11	Fastest training times for CNN and XGBoost on CPU and GPU (FDA model)	73
5.1	Runtime of scaling_benchmark.py across various GPUs in seconds (lower is better)	83
5.2	Million particles/s for scaling_benchmark.py across various GPUs (higher is better)	84
5.3	Baseline training runtimes in seconds on Frontier (1 GCD)	85
5.4	cINN model training MI250X vs. A100	87
6.1	Ligand pose optimizer model training results using Molecular pose descriptors only.	103
6.2	Effect of layer sizes and dropouts on test R^2	105
6.3	Ligand pose optimizer model training results based on descriptor set used. MP: Molecular Pose descriptors (Simplified Lennard-Jones + geometrics descriptors)	107
6.4	Validation results show that the model's performance was much lower than on the validation split of the training data	108
6.5	Train, Validation, and Test R^2 values for different data splits. Data split percent indicates % for Valid and Test	108

LIST OF FIGURES

2.1	Visualization of the ONNX & ONNX Runtime Stack with respect to high-level frameworks and low-level accelerators [35].	14
2.2	Figure showing the status of HiT. Green colored goals were demonstrated in this preliminary research and the blue colored goals will be completed in the next steps of this research.	25
3.1	Figure shows various TCGA cancer types and the data available for each disease	34
3.2	Example predictions for various data sets for model trained with ALMANAC data	35
3.3	Plot of predictions for several data sets when model trained with ALMANAC data	36
3.4	Plot of predictions for several data sets when model trained with ALMANAC data (smaller scale)	36
3.5	Example predictions for various data sets for model trained with GDSC data	37
3.6	Plot of predictions for several data sets when model trained with GDSC data	37
3.7	Plot of predictions for several data sets when model trained with GDSC data (smaller scale)	38
3.8	Example predictions for various data sets for model trained with NCI data	38
3.9	Plot of predictions for several data sets when model trained with GDSC data	39

3.10	Plot of predictions for several data sets when model trained with GDSC data (smaller scale)	39
3.11	Example predictions for various data sets for model trained with RTS data	40
3.12	Plot of predictions for several data sets when model trained with RTS data	40
3.13	Plot of predictions for several data sets when model trained with RTS data (smaller scale)	41
3.14	Plot indicates that most of the cell lines had very few data points. So the data was skewed by the few cell lines that had many data points	49
3.15	Table of predictions for CCLE cell lines (rows) for each of CCLE drugs (columns)	50
3.16	Table derived from Table 3.15 showing predicted, true, and margin between the two for each CCLE cell line and drug combination . .	51
3.17	Several hyperparameters tested for CCLE	51
3.18	XGBoost model predictions for CCLE data	53
3.19	CNN model predictions for CCLE data	53
3.20	Autoencoder model loss plotted vs. training epochs	59
4.1	A flow diagram shows various parts that make up the UNNT software	65
4.2	Figure from [30] shows how AUC is calculated for each drug's viability based on its concentration	67
5.1	The figure depicts a real-time vector field visualization test of PIconGPU. Credit: Rene Widera, HZDR.	78
5.2	The figure shows how streaming eliminates the need for data storage on disk by streaming the data from the producer (simulation) to the consumer (ML). Credit: Jeffrey Kelling, HZDR.	79

5.3	A figure representing the real-time data reduction that's necessary even with streaming because the volume of data produced by the simulation is so large. Credit: Jeffrey Kelling, HZDR.	79
5.4	A visualization of continuous learning method used by PIConGPU to learn and extract insight in real-time from simulation data streaming from the producer to the ML model. Credit: Sunita Chandrasekaran and Richard Pausch, HZDR.	80
5.5	An example figure showing select environmental variables that were necessary for running distributed training.	86
5.6	Comparison of training time in minutes between AMD and NVIDIA GPUs across various number of ranks.	87
5.7	Image of a profile taken while training a model across 8 GCDs in a single node.	88
6.1	Figure adapted from IMPECCABLE [79] to visualize the description provided in text	94
6.2	Figure shows that as batch size increases training time decreases . .	104
6.3	Figure shows that as batch size increases R^2 decreases	104
6.4	Plot of results showing the effect of reducing training data size on R^2 of Train, Validation, and Test	109

ABSTRACT

As software and hardware concurrently advance, it's important to design and build ML frameworks that are hardware architecture agnostic. More specifically as accelerators for ML workflows become more prevalent, the ability to have high level code that can run across such accelerators would be highly beneficial by reducing the need to rewrite code and libraries for each hardware. At the same time, advancement in machine learning (ML) methods has enabled extraction of meaningful information from large and complex datasets that have assisted in better understanding, diagnosing, and treating illnesses such as cancer. This applies to applications beyond oncological drug response and drug discovery including understanding complex plasma physics phenomena.

This thesis focuses on designing and building scalable and portable machine learning-based workflows while adapting them to new hardware architectures. This thesis also includes scaling and improving performance of surrogate models that reduce scientific simulations necessary for extracting insights. A phenomenon increasingly necessary as scientific challenges increase in computational complexity. We demonstrate the ideas using two case studies.

An improved drug discovery pipeline is designed for shorter development timelines through model enhancement and scaling on new hardware capabilities. The thesis investigates gradient boosted tree-based methods as viable alternatives to CNNs in demonstrating the limitations of existing neural network-based drug response models. These gaps are resolved by designing and building software that helps assess the variation in performance of each class of models and includes improvements made to the accessibility of these models for domain experts. The current approaches rely on

RNA sequence based gene expression values of cell lines, 2D molecular drug descriptors, and drug response data to predict cell growth. To overcome the challenges faced with the existing 2D molecular datasets, the next aspect of the thesis focuses on improving the performance of ML techniques that synthesize molecular docking of the 3D molecular drug descriptors for pose estimation of protein-ligand binding to reduce the subsequent molecular dynamics simulations needed in drug-discovery workflows. In addition to hyperparameter optimization (HPO) and model tuning, scaling the training of such models will greatly improve the throughput of lead compound discovery.

Scaling such ML workflows on new hardware architectures like AMD GPUs is challenging. The thesis further explores the scaling aspect by using another case study that involves in-transit ML of plasma physics simulations to uncover correlations between emitted radiation and particle dynamics within the simulation. The ML surrogate model employs online learning using data streamed from the simulation and scales up to 400 GPUs.

To summarize, this thesis introduces novel software frameworks and workflows to advance the state-of-the art for case studies involving drug discovery models for cancer research as well as plasma physics simulations through model enhancement and distributed scaling on large supercomputers.

Chapter 1

INTRODUCTION

The nature of Machine Learning (ML) workflows' increasing compute requirements and problem sizes necessitated a change in not only hardware accelerators but also the software necessary to support these workflows. AlexNet[41] introduced Deep Learning (DL) to GPU acceleration, and the use of GPU acceleration for DL has only increased since then. At the same time, supercomputers went through a change in hardware architecture from homogeneous to heterogeneous systems. The Jaguar supercomputer at Oak Ridge National Laboratory (ORNL) had an x86-based instruction set architecture (ISA), and then it was upgraded to the hybrid system Titan adding the NVIDIA K20X GPUs. The next system at ORNL, Summit, was an upgrade but featured a similar heterogeneous architecture with IBM Power ISA and NVIDIA V100 GPUs. Another big change occurred when ORNL went with the AMD MI250X GPUs for the latest Frontier exascale supercomputer. In the meantime, other hardware accelerators such as Tensor Processing Units (TPUs), Cerebras wafer-scale chips, and more recently Groq's LPU (LLM processing unit) became available as viable alternatives to GPUs. Having such a wide range of hardware architectures brings its own set of challenges and requires standardization and agreement among the industry for software and models to work across architectures.

Each of these architectures often requires the supporting software stack for existing workflows to run seamlessly. Portability is a problem faced not just by ML codes. Many HPC codes exist in legacy form that require backward compatibility or code updates for each of the system architectures as they advance. Similarly, ML workflows that work in one system may not be as conducive to working in other systems with a different system architecture. There are compatibility gaps that develop during this

evolution as software stacks evolve with hardware advancements. This evolution often results in increased performance and subsequently algorithmic improvements. As hardware evolves, new software often fills in gaps and advances the state of the art to make hardware, software, and algorithms deliver performance gains. This thesis focuses on designing and building scalable and portable machine learning-based frameworks to new hardware architectures with a focus on cancer drug discovery as a case study.

This section introduces the case study and the class of problems we chose for evaluating and building such frameworks. Cancer drug discovery models contain a subset that focus on drug response problems that in most cases rely on structured training data. Improving the predictive capabilities of such models requires changes, such as improved data quality and model architecture. Drug response models are generally trained on cell line (cell culture) datasets due to the availability of drug response values for this type of data. Models trained with tumor normal (healthy) and tumor disease tissue data would be ideal except for the fact that tissue data has very little corresponding drug response data, making it challenging to improve drug response models' predictive capabilities. Existing ML techniques such as CNNs have been the primary in silico methods for drug discovery to predict effectiveness of drugs based on their molecular features. To advance the state of these models, this work explores tree-based ML models as viable alternatives to neural network-based models. In many cases, changing the model architecture to test the existing data and problem type can lead to improvements in predictive capabilities, but this process can be taxing for researchers and ML practitioners.

Using ML techniques to synthesize physics-based simulations such as docking for pose estimation and molecular dynamics simulations to estimate the binding free energies of the protein-ligand binding process is another technique to improve cancer drug discovery models. Such ML techniques have a throughput problem that can be addressed using various modeling and training techniques. This case study also addresses these problems using the Frontier exascale system, which brings several challenges that must be overcome during this process.

1.1 Research Questions

1) How do we apply model enhancements and performance optimizations to design drug discovery pipelines for shorter development timelines?

2) How do other ML model architectures such as tree-based models compare to CNNs?

3) How do we improve the accessibility of these models for domain experts, giving them the ability to analyze and compare multiple types of models simultaneously?

4) How well do ML models scale on new hardware architectures and state-of-the-art GPUs? And what challenges will this bring?

5) What insights or research questions will hyperparameter optimization (HPO) and scaling of molecular dynamics ML models lead to?

1.2 Motivation

As software and hardware advance concurrently, it is important to design and build ML frameworks that are hardware architecture agnostic. More specifically, as accelerators for ML workflows become more prevalent, the ability to have high-level code that can run across such accelerators would be highly beneficial by reducing the need to rewrite code and libraries for each hardware. This has many downstream effects on hardware accelerator design as vendors will be forced to not rely on proprietary software to drive hardware sales. That ultimately makes compute cheaper, allowing the costs of development and deployment of ML models to go down. Cancer drug discovery models are a domain in which this development would bring great benefits to the advancement of science. As hardware accelerators, such as GPUs, support a wider range of ML models, it is important to scale the training of ML workloads for cancer drug discovery on systems like Frontier with AMD accelerators. SOTA (state-of-the-art) 175 billion and 1 trillion parameter Large Language Models (LLMs) can scale up to 1024 and 3072 AMD GPUs respectively [23]. Even as hardware support for algorithms improves, they can still face performance challenges during scaling. In addition, as the

software needed for the algorithms to work on evolving software catches up, it can be a point of weakness in areas such as compatibility, efficiency, and performance. The overarching goal of this thesis is to design and build software frameworks that allow such portability of ML models due to the great benefits that it brings.

1.3 Thesis Organization

This section provides the outline of this thesis covering each of the chapters.

Chapter 2 covers a framework that can be used to make ML models portable across hardware architectures using graph-level intermediate representation (IR) to port from one software framework to another. This work also demonstrates the ability to convert ML workflows between hardware architectures. It was done using general models such as Transformer-based models such as BERT[27]. Making it easier to take a model built with one ML framework, like TensorFlow, and have it work seamlessly on another framework, like PyTorch, greatly improves the flexibility of individual frameworks and ML workflows.

Chapter 3 discusses my work on a class of problems in ML-driven drug discovery workflows called drug response problems. We start by porting these ML workflows to systems outside the NIH (National Institutes of Health) and DOE (Department of Energy) environments. Then explore extending these resources to newer hardware architectures. This makes NCI-DOE resources more accessible to researchers outside these organizations on a wider range of systems. The remainder of the chapter discusses the work to make these models more robust and improve their predictive capabilities. These models are often upstream of the long and expensive drug discovery process that can take more than a decade. Improving these models can make the entire process more efficient by better selecting drugs that go into the next phases of the process.

Chapter 4 covers the UNNT (Utility for comparing Neural Nets and Tree-based models) software abstraction we build based on our work done while improving the models in the previous chapter. This chapter also describes the advantage of tree-based models over neural networks when the data is structured. Therefore, this helps

researchers and practitioners compare and analyze such model architectures on their own data.

Chapter 5 covers the work done to make ML workflows scale on the Frontier Exascale system. This includes challenges and setbacks faced with new hardware architecture, software compatibility, and system setup. The work resulted in the scaling of the ML workflows to 100 nodes. This is one of the first demonstrations of such large-scale ML training using Frontier and AMD GPUs and an important endeavor as AMD GPUs become prevalent on compute clusters and on the cloud.

Chapter 6 brings all previous chapters together by focusing on ML drug discovery models that rely on physics-based applications such as docking for pose estimation and molecular dynamics simulations for predicting bind-free energies during the protein-ligand binding process. The objective is to scale these workflows on Frontier using the expertise gained from the work in Chapter 5. Increasing the throughput of these techniques improves the quantity of potential lead compounds that can be considered to advance to later stages of the drug discovery process.

And finally, chapter 7 will provide a general overview of the thesis and the major parts and reiterate the main takeaways.

1.4 Thesis Contributions

1) Established that gradient boosted decision trees (GBDTs) outperform CNNs in predicting drug responses, with potential applicability to other structured datasets.

2) Designed and created a software utility that researchers and domain scientists can use on their custom structured datasets to compare and analyze CNNs and XGBoost models.

3) Proposed and developed novel approaches to address the scalability of ML training on large-scale supercomputers such as Frontier equipped with the state-of-the-art AMD GPUs

Chapter 2

EXPLORING PORTABILITY OF DL MODELS

This chapter consists of a study performed to explore and enhance portability of ML models across various ML frameworks and hardware architectures using a library that uses graph-level intermediate representation (IR) of the ML models to achieve this.

2.1 Introduction

The proliferation of Deep Learning (DL) models in recent years fueled a growth in the number of open source DL frameworks. Once a user chooses a framework there can often be constraints on the hardware accelerators that can be used to train or run models with. Even worse, the choice of their framework can hamper their ability to choose the best hardware accelerators for training and inference.

In this research project we introduce **Hardware independent Translator (HiT)**, a toolchain that aims to increase portability of DL based applications across different kinds of systems. Eventually the goal for HiT is to provide this portability across various hardware architectures. This includes support for application specific integrated circuits (ASICs) such as Tensor Processing Units (TPUs) that can give users the ability to use the vast compute capability some of them have to offer.

The toolchain, HiT, will achieve its goal of increasing application portability by incorporating a variety of tools and techniques. The toolchain will include a high-level abstraction for creating portable models, an Intermediate Representation (IR) layer that translates model graphs from one framework to another and containerization across systems. Other features of the tool chain could also include support for a growing

list of DL model operators such as dropout. The toolchain will be evaluated using an MLPerf like benchmark suite among other case studies.

This work demonstrates the potential high-level abstraction, followed by evaluating a workflow that entails an IR targeting new hardware runtimes and explores the benefit of containerization of models across systems that we have access to.

The number of frameworks available for deep learning is extensive and there is no one framework that dominates in usage [37]. Some examples of these frameworks include PyTorch, TensorFlow, and Scikit-learn. Each of them has its strengths and weaknesses, but for the most part the differences are minimal, where most algorithms can be written in any of them, and the choice of framework comes down to individual or team preferences. Some of these frameworks have back-ends that are optimized for certain hardware accelerators as training a model can be a very compute intensive task. This often means that users are locked into not only the software but also the hardware accelerators they can use to train their models.

Most users do not own the hardware accelerators they use for training. That means they rely on high performance computing (HPC) systems like UD’s DARWIN [78] cluster to share computational resources with other users of these large systems. DARWIN, like other similar systems, has many hardware accelerators available for its users.

Hardware vendors such as NVIDIA and AMD have slightly different GPU architectures. In addition, there are architectures that are being designed and built specifically for deep learning type workflows and are broadly known as application specific integrated circuits (ASICs). Google’s Tensor Processing Unit (TPU) is an example of an ASIC and it is designed to work with TensorFlow. When accessing shared compute resources, users may not get access to the systems they request leading to long wait times. So there is a need for a tool that can allow users to seamlessly transition between different types of systems without the need to manually change the model or the code to run on a new target architecture.

HiT solves compatibility issues between the supported architectures and high-level DL frameworks to allow applications to run on a target architecture. ONNX [25] translates models at the IR level of a compiler, which translates the high level code to the machine level to represent these models, which are typically DL, and consist of a set of operators which are translated into an intermediary format also known as ONNX. ONNX format integrates well with ONNX Runtime[25] because the latter supports various types of hardware accelerators in its backend. ONNX Runtime allows HiT to maintain performance, which is critical during training and inference of deep learning models, and in some cases manages to improve the performance on a target architecture compared to the original one.

The main contributions of this research are:

- A software prototype, HiT, that demonstrates portability of DL applications using a combination of container environments and graph/IR based DL model translators to demonstrate the ability to target these applications on various hardware runtimes and show the performance impacts of doing so.
- A new framework to create a more advanced IR level conversion wrapper based on what we learned through the research done for the previous contribution

In the rest of this chapter, we go into more details regarding various aspects of this research such as the motivation and provide the background details about some crucial parts of this research before presenting the framework and the results of our experiments.

2.2 Motivation

There are many HPC+AI workflows that are designed to run on specific systems based primarily on the hardware available at the time of development. But hardware architectures evolve and organizations acquire different kinds of systems based on factors such as cost, efficiency and performance. And often these decisions are made for the benefit of the wider scientific community. As the Office of Science reports [1], there are substantial needs for extreme-scale computing and the needs vary widely across a growing range of workloads. The report also states the growing challenges of making

scientific applications more portable as the complexity and diversity of HPC grows. As cutting edge HPC transitions into exascale level computing, hardware vendors that can maximize performance of these systems are chosen.

At the Department of Energy’s (DOE) Oakridge Leadership Computing Facility (OLCF), applications running on the current Summit supercomputer [63] need to eventually be transitioned to the upcoming Frontier supercomputer [64]. In this case, the transition will be from NVIDIA GPUs to AMD GPUs. This has several implications including the necessity to port applications and models from NVIDIA’s CUDA runtime to AMD’s runtimes such as MiGraphX. For developing software to run on Frontier, many traditional low level programming languages like C, C++, and Fortran are supported high level options like python will have limited GPU support according to OLCF [77]. Most DL framework users write in python so it will be challenging for these users to transition their DL workflow from NVIDIA and CUDA environment to an environment with a different hardware architecture such as AMD’s GPUs.

Another example is an Exascale Compute Program (ECP) project called CANDLE (CANcer Distributed Learning Environment) [94, 92]. The project’s goal is to address three separate challenges related to cancer surveillance with pilot applications. CANDLE mainly uses python across a few various DL frameworks. Here again there is need for increased portability of CANDLE. As mentioned earlier in the Frontier example, applications written in high level languages like python with frameworks make it challenging to port to a system with a different hardware architecture.

This problem is not limited to large scientific applications that require extreme-scale computing. HPC clusters such as DARWIN offer a combination of accelerators including NVIDIA and AMD GPUs. If researchers wanted to run their applications on a different system, including due to availability of resources, the process is not simple even for small workloads. There is a need for a high level abstraction that can allow users to port their applications between systems that offer different hardware accelerator architectures much more seamlessly compared to the process it currently requires to make than transition.

2.3 Background of DL frameworks’ portability and previous work

To our knowledge, our research is the first to address incompatibilities between DL frameworks, models, and hardware architectures. In this section, we describe the work the HiT builds on. Numba [44] is the most related in terms of its objective. It is a just-in-time compiler for numerical functions in Python that has support for certain types of hardware acceleration including CPU multi-threading and GPUs from NVIDIA and AMD. The libraries that Numba supports, such as NumPy [89], are widely used attracting many users. But NumPy, which is used mainly for numerical computing, does not offer the functionality to build complex ML models unlike DL frameworks. In addition, the main goal of Numba is to offer parallelism whereas HiT’s goal is to remove the hardware constraints when accelerating models during training and inference.

HiT leverages several software libraries to provide its functionalities. For example, ONNX provides IR (Intermediate representation) level translation of models between various popular frameworks such as TensorFlow and PyTorch. In addition, ONNX Runtime allows models in ONNX format access to hardware level acceleration from various providers such as CUDA, TensorRT, and MiGraphX. In addition to these libraries, we also use MLPerf [73], a benchmark suite, as a reference since it provides results for training and inference of leading DL models on various hardware accelerators such as NVIDIA’s V100’s, Google’s TPUs, and the A64FX which is an ARM based accelerator currently being used in Oookami which is the world’s faster supercomputer.

The rest of this section will go into details on the background of each of the major libraries used and how they fit into the overall context of this research.

2.3.1 Machine learning frameworks

In recent years, machine learning became mainstream and gained popularity because of the advances made using deep learning that enabled various tasks such as speech recognition, object detection, and many other fundamental problems to be tackled with much greater accuracy. This was mainly driven by an increase in the

availability of data and compute resources. Increased usage of ML and DL models necessitated frameworks to help users build these machine learning models for not just research but also for software development.

ML frameworks are high-level and productivity-focused with the low-level performance-oriented interface with hardware mostly abstracted from the users. These frameworks are also rapidly evolving based on the needs of their users. And because most of them are open-sourced their most ardent users often add features to these frameworks. One such example framework is Scikit-learn [68], which integrates many state-of-the-art (SOTA) machine learning algorithms for medium-scale supervised and unsupervised problems. It also uses several other libraries such as Numpy [89] and Scipy [54] which allow Scikit-learn to store data efficiently in data structure and provide linear algebra functions essential for machine learning. The remaining frameworks discussed in this section in addition to allowing users to build machine learning models also allow for large-scale deep learning to express a variety of algorithms. Some examples of these DL frameworks include TensorFlow [3], Torch [20], Keras [24], PyTorch [67], and CNTK [80].

Most of the DL frameworks adopt a layered software architecture that is similar and provide APIs that enable users to configure DL models and training methods including optimizers. According to [95], these frameworks are implemented on top of various parallel computing libraries: BLAS (Basic Linear Algebra Subprograms) libraries, such as OpenBLAS, cuBLAS, NCCL (NVIDIA Collective Communications Library), OpenMP, and Eigen.

A. TensorFlow and Keras

TensorFlow [3] was originally developed by the Google Brain team for internal Google use to conduct ML and DL research. It is now open sourced and used for both research and production. It is one of the most popular frameworks according to various metrics including search volume, arXiv articles, and GitHub activity [37]. In

addition to these reasons, the main focus of the research is framework agnostic for the most part so it was important to test with various frameworks that are supported well and have a large community. TensorFlow was also extensively used in the CANDLE benchmarks further reinforcing the need to pick TensorFlow as one of the frameworks tested. TensorFlow has a dataflow model where tensors flow between nodes along the edges [95]. The nodes represent a mathematical operation while the edges represent dataflow giving users the flexibility to represent complex neural networks.

Keras [24] is a deep learning API that runs on top of TensorFlow and is written in Python. It was developed to enable faster experimentation. It provides essential abstractions for prototyping with much higher velocity. Like TensorFlow, Keras also provides researchers and developers the ability to take advantage of TPUs and GPUs for faster computation. In addition, it also allows models to be exported as graphs to other runtimes such as browsers, mobile, and embedded systems.

B. PyTorch

PyTorch [67], like TensorFlow, is another open source deep learning library originally developed Facebook’s AI Research lab. Most of the popular graph-based DL frameworks construct a static dataflow graph to represent the computation which can then be applied repeatedly to batches of data. Although this approach provides visibility into the whole computation ahead of time, it comes at the cost of ease of use, ease of debugging the graphs, and flexibility when it comes to the types of computation that can be represented. PyTorch offered users dynamic eager execution with automatic differentiation mainly without sacrificing performance, where operations are evaluated immediately without building graphs [67]. TensorFlow subsequently adopted eager execution and to allow its users similar functionality as PyTorch’s dynamic eager execution.

In addition to its design principles of putting researchers first and ease of use, PyTorch also offers performance-focused implementation using an efficient C++ core,

having separate control and data flow, multiprocessing, and through a custom caching tensor allocator. The combination of these features allows PyTorch to use GPUs with CUDA APIs to saturate the GPU and reach peak performance even though Python has a high overhead because it is an interpreted language.

2.3.2 Open Neural Network Exchange (ONNX)

ONNX (Open Neural Network Exchange)[25] allows conversion of DL models using graphs generated by frameworks discussed in the previous section. These graphs are also colloquially known as the IR (Intermediate representation) layer. ONNX defines a common set of operators and a common file format for using models with a variety of frameworks, tools, runtimes, and compilers. IR is the level where DL models are broken down into operators which are the building blocks of these models. But this is not always possible because of the rapid pace of innovation in DL models where ONNX may not support the latest operators. But ONNX allows users to create custom operators to support the latest DL models in ONNX format. ONNX also interfaces with ONNX Runtime to allow models in ONNX format to infer (and train in the future) which requires interfacing with the hardware runtimes such as CUDA and TensorRT from NVIDIA and MiGraphX from AMD.

2.3.3 ONNX Runtime

ONNX Runtime [26] supports a wide range of hardware runtimes and by extension many architectures. Some of these include the CPU, CUDA and TensorRT for NVIDIA GPUs, MiGraphX for AMD GPUs. Support for the latter is limited as MiGraphX integration is in beta. The combination of ONNX [25] and ONNX Runtime will allow us to create prototypes of DL model pipelines that demonstrate the increased flexibility in hardware architecture choices this software combination allows. In addition to that, experiments, including our own, have shown that ONNX Runtime can offer performance speed up at inference time in certain DL models that are optimized for ONNX Runtime.

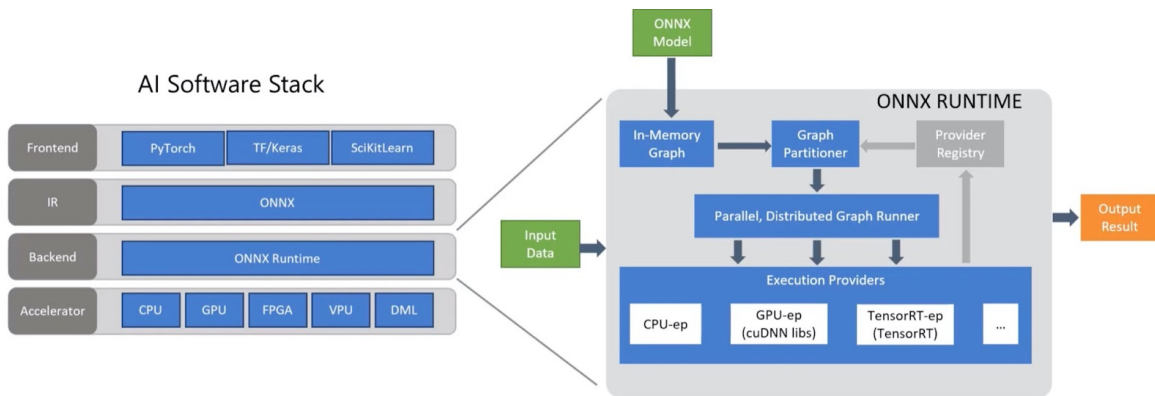


Figure 2.1: Visualization of the ONNX & ONNX Runtime Stack with respect to high-level frameworks and low-level accelerators [35].

2.3.4 TVM

For further performance optimization of DL models, TVM (Tensor Virtual Machine) [17] can help optimize DL models in ONNX format at the IR and compiler level. It’s main objective is to help bridge the gap between high level productivity-focused software frameworks and low level performance and efficiency-oriented hardware back-ends. TVM uses several techniques including operator fusion which combines multiple operators into a single kernel without saving the intermediate results in memory. This optimization can be particularly useful at inference time. TVM also contains an *automated schedule optimizer* that can find optimal operator implementations for each layer of a DL model. But often the search space to choose schedule optimizations is large for each hardware back-end. To solve that it uses ML-based cost model and design choices to optimize finding best schedules. The combination of ONNX, ONNX Runtime, and TVM can potentially help improve performance of DL models.

2.3.5 Containerization

One of the benefits of containerization is being able to install the benchmark, dependencies and its corresponding environment variables in a container image and using the image to run in a number of different clusters rather than installing each of

these components independently on every system of interest. Containerization would also avoid any conflict with libraries that may be installed by the system admin or other developers. It will also allow a seamless usage of different DL frameworks that may typically come with conflicting software dependencies while running on the same system. We will also be able to allocate relevant compute resources in a way that we could run multiple instances of a DL framework concurrently.

Containerization also comes handy even if the underlying platform is from the same vendor, such as NVIDIA’s GPUs. Migrating the code to a new runtime, for example from CUDA to TensorRT, which can run on the same NVIDIA hardware, will still require a different set of software dependencies. The problem is even more pronounced when switching hardware such as from NVIDIA GPUs to AMD GPUs.

Containerization makes this process more streamlined because the containers can be prebuilt with the prerequisite software dependencies to run on a different runtime environment. This is especially true for compute clusters such as UD’s DARWIN because installation of niche software can be restricted and the process to do so can often be cumbersome if it is not officially supported by a HPC compute cluster. Containers alleviate this bottleneck because they provide secure and isolated virtual environments that can come with the required dependencies.

Because of all these advantages, migrating the research workflow to work in containerized environments will be an important aspect of this work. There will be some prerequisites for containerizing including support from the compute clusters or systems for container software such as Docker [52], Singularity [43], or Kubernetes.

A. Docker

Docker is an industry standard container software that promises the ability to package applications and their dependencies into lightweight containers that can start up quickly, create an isolated environment, and can be moved easily between different systems [52]. But Docker creates security issues and causes other problems in HPC

environments. In terms of security, Docker containers have privileged access to the host systems network file system making it unsuitable. In addition, Docker uses cgroups to isolate containers and that conflicts with the slurm scheduler used by many HPC systems for resource allocation [10].

B. Singularity

Even though Docker creates security concerns and is incompatible with HPC systems, its most important feature of isolated environments and lightweight nature can still be useful in HPC context. Singularity [43] is an alternative to Docker for containerization in HPC environments. It provides reproducible, portable, shareable, and distributable containers just like Docker but at the same time it assumes a "no trust" security model where untrusted users run untrusted containers. In addition, Singularity supports HPC hardware and scientific applications because of its support for MPI. Singularity also allows users to bring external driver modules from the underlying system into the container environment and because of this it has built it options to make GPUs, from both NVIDIA and AMD, available to the containers.

Studies in [10] show that it is possible to deploy DL frameworks at scale on HPC systems using containers. But the major distinction between their work and ours is that they did not use GPUs while we use them in addition to CPUs. Furthermore, they use Charliecloud [72] for their containerization software whereas we use both Docker and Singularity depending on the system we run our experiments on. This is because of the drawbacks of Docker in HPC environments mentioned previously requiring us to use Singularity as the alternate container software on HPC clusters.

2.4 Experiments and results of frameworks ported

As mentioned previously in the background section, portability of DL based scientific applications like CANDLE is an important challenge. We create a workflow, named HiT, using a combination of containerization, ONNX, and ONNX Runtime to

show that portability is possible. In addition to achieving portability, our experiments show that this workflow does not have a large overhead in terms of latency of DL models at inference time in ONNX Runtime using ONNX format compared to inference in the native framework. And this was tested across both CPU and various GPUs.

ONNX provides a suite of tools to assist users when converting their models from various DL frameworks to ONNX format. Some frameworks such as PyTorch have that functionality built into the framework natively while ONNX also provides external (non-native) libraries that can be used to convert from other frameworks that do not have ONNX conversion tools natively. Some of the frameworks where this is true include Scikit-learn, TensorFlow and Keras. For each of them, ONNX has built separate libraries such as `sklearn-onnx`, `tensorflow-onnx`, and `keras-onnx` respectively to convert to ONNX format. Even with all these libraries conversion of models must be done properly for inference to work in ONNX Runtime. In addition to that, the helper conversion libraries available to use from the DL frameworks convert a single NN graph and are not reliable when a user converts an ensemble of models.

To solve this, we build an ONNX conversion wrapper that allows users to convert an ensemble of DL models to the ONNX format for inference in ONNX Runtime. Based on our experience from this research so far, a program that offers this functionality needs to have a few crucial components to ensure that an ensemble of models are converted accurately into ONNX format.

The following is high-level abstraction created based on the lessons learned during this research

1. Take the following from user as input
 - DL framework of origin
 - The trained models that are part of their ensemble
 - Each model's dummy input where each of the models can ingest this data to execute each of the models once to construct their graph

- Some training/testing data for each of the models to test equality after conversion
 - Hardware runtime/architecture user will using for inference
2. Import all necessary libraries for conversion (based on DL framework of origin)
 3. Convert each of the models into ONNX format
 - ONNX helper methods convert with Tracing methodology using user provided dummy input
 4. More robust validation. User must have option to override this for models that behave stochastically
 - After conversion is successful for each model, where ONNX validates the model, we execute the converted model in ONNX format (run inference) once with the additional train/testing data user provided.
 - Then we run inference on the original model
 - Complete if outputs of both models are same
 5. Optimize operators of the converted models with TVM [17]
 - Use TVM’s operators fusion feature that transform the computational graph into a fused version that can give improve performance by reducing memory accesses.
 - Incorporate TVM’s automated schedule optimizer.
 6. Return the ensemble of models in ONNX format.

This abstraction not only streamlines the process of converting a complex ensemble of models into ONNX format but also incorporates a method of improving the performance of DL models at inference time.

2.4.1 Experimental Setup

Thanks to my advisor, Dr. Sunita Chandrasekaran, I was able to obtain access to several compute systems with a broad set of hardware accelerators to run my experiments on. This section will give details about each of those systems.

A. Computational Research and Programming Lab (CRPL) Systems

The first, and probably the most important, system used during the course of this research was CRPL’s Leia system at the University of Delaware. It consists of an AMD ThreadRipper 1950X CPU with 16 cores, 32 threads, and 48GB of RAM. In addition, Leia also has 1 NVIDIA GTX 1080 with 8GB GDDR5X RAM with NVIDIA’s Pascal architecture and 2 NVIDIA Tesla P40 with 24GB of GDDR5X RAM and also on the Pascal architecture. It was initially running the Ubuntu 18.04 operating system, but was upgraded to Ubuntu 20.04 during the course of this research. As stated in the next section, all experiments that involved using a GPU on Leia used a single NVIDIA P40 GPU.

The main softwares required were Docker and the NVIDIA Container Toolkit [62] which was necessary for the Docker containers to have the ability to access the GPU on the host system (Leia). Leia was often the cornerstone of this research because the initial setup and trial runs for many experiments carried out on the HPC systems that follow in this section were done on Leia. For example, compute clusters required elevated privileges to build containers using the manifest file such as a Dockerfile. Often this is solved by building the container on Leia and then moving the container to the target system.

The second CRPL lab system used to conduct experiments is Skywalker which consists of the AMD Threadripper 1950X CPU with the same configurations as Leia but it has different GPUs. It consists of 1 NVIDIA Titan V with 12GB of HBM2 (high bandwidth memory) and 2 NVIDIA Tesla V100s with 16GB of HBM2. Both have NVIDIA’s Volta architecture compared to the older Pascal architecture on Leia’s P40s. Like Leia, Skywalker was also updated to Ubuntu 20.04 OS from Ubuntu 18.04. And it also required Docker and NVIDIA Container toolkit to give the containers access to the GPUs.

B. UDEL’s Caviness

Caviness is University of Delaware’s third HPC cluster. It consists of 126 compute nodes with a total of 4536 cores and 24.6 TB of memory [88]. Each of the nodes consists of Intel ”Broadwell” 18-core processors in a dual-socket configuration for a total of 36 cores per node. Caviness also consists of nodes with NVIDIA P100 GPUs that have NVIDIA’s Pascal architecture. Caviness was mainly used to run experiments on this GPU.

C. UDEL’s DARWIN

DARWIN (Delaware Advanced Research Workforce and Innovation Network) is the most recent HPC cluster commissioned at UD. It consists of 105 compute nodes with a total of 6672 cores, 22 GPUs, 100 TB of memory, and 1.2PB of disk storage. Unlike Caviness, DARWIN has a combination of GPUs from both NVIDIA and AMD. It has 9 nodes, each with an NVIDIA T4 GPU consisting of 2,560 CUDA and 320 Tensor cores per node. DARWIN also has 3 nodes each with 4 NVIDIA V100 GPUs consisting of 20,480 CUDA cores and 2,560 Tensor cores per node [78]. In addition to the NVIDIA GPUs, DARWIN also has an AMD MI50 GPU. The AMD GPU is of particular interest for this research. Most of the work pertaining to this research done on DARWIN was done during its phase 1 early access because it is still in the process of entering full production.

Systems	DARWIN	Caviness	CRPL
CPU	Intel + AMD	Intel E5-2695	AMD Threadripper
GPU	NVIDIA V100 + NVIDIA T4 + AMD MI50	NVIDIA P100	NVIDIA P40 + NVIDIA V100

Table 2.1: Systems used to conduct experiments in this study

The experimental setup for this research consists of variables such as DL frameworks, models, hardware accelerators, and hardware runtimes. Some of these have been

GPU	NVIDIA P40	NVIDIA P100	NVIDIA V100	AMD MI50
Peak Single Precision (FP32) Performance	11.7 TFLOPs	9.5 TFLOPs	14.1 TFLOPs	13.3 TFLOPs
Memory Size	24 GB	16 GB	16 GB	32 GB
Memory Type	GDDR5x	HBM2	HBM2	HBM2
Memory Bandwidth	480 GB/s	732 GB/s	900 GB/s	1024 GB/s

Table 2.2: Specifications of GPUs used during this study

discussed in detail in the previous sections. The ones that have not will be explained in this section. All the models outline below are pre-trained in one of the frameworks outlined in the frameworks section above. This section will include details about the various models chosen for experimentation in this research and some of the reasoning behind their choosing them.

A. AlexNet

AlexNet is a convolutional neural network (CNN) that was originally introduced in [41] and won the ImageNet Large Scale Visual Recognition Challenge in 2012. We chose this model because of the extensive documentation available for it in various DL frameworks. This was important to eliminate problems in framework of origin as the main tests would be performed in a target framework such as ONNX.

B. SqueezeNet

For the second model, we chose Squeeze [40] because of its fundamental similarity to AlexNet in terms of the general problem. Because of that the data at inference time would also be similar. But the most important reason for choosing SqueezeNet was to see the impact of model size on performance after conversion to ONNX. SqueezeNet has 50x fewer parameters compared to AlexNet and the model size is thus smaller in

a similar proportion. SqueezeNet manages to achieve this while maintaining accuracy AlexNet achieves [40].

C. BERT

In the past few years, pre-training of language models has significantly increased the size of models. These models deliver state-of-the-art performance across a wide range of NLP tasks. In our experiments, we wanted to incorporate a large language model to compare performance between the native framework and the combination of ONNX and ONNX Runtime across hardware runtimes. We experimented with BERT because of its versatility and performance across eleven different natural language processing (NLP) tasks [27]. BERT has many variations that are fine-tuned for various tasks. For the purpose of this research, we chose the BERT-base-uncased model. Base refers to the original model architecture outlined in [27] and uncased refers to the language model’s capability in terms of the case of text it can take as input. This means all input sequences are lowercase.

D. CANDLE

CANDLE (CANcer Distributed Learning Environment) [94, 92], a DOE and National Cancer Institute (NCI) partnership project, consists of three key challenges. The first one, called the "drug response problem," aims to develop predictive models for drug response to optimize pre-clinical drug screening and drive precision medicine-based treatments for cancer patients. The goal of the second challenge, called the "RAS pathway problem," is to understand the molecular basis of key protein interaction in the RAS/RAF pathway that is present in 30% of cancers. The third challenge, called the "treatment strategy problem," aims to automate the analysis and extraction of information from millions of cancer patient records to determine optimal cancer treatment strategies across a range of patient lifestyles, environmental exposures, cancer types, and healthcare systems. The goal of the CANDLE challenge problem is to solve large-scale ML problems for the three applications mentioned above [2].

2.4.2 Results

This section highlights some of our preliminary results using the HiT framework.

Models	Frameworks	CPU ¹	GPU (P40)	GPU (V100)
AlexNet	PyTorch	2.3s	0.12s	0.07s
	ONNX Runtime	3.14s	0.17s	0.1s
SqueezeNet	PyTorch	5.58s	0.21s	0.22s
	ONNX Runtime	1.29s	0.36s	0.11s
BERT	PyTorch	3.1s	0.825s	0.9s
	ONNX Runtime	2.4s	0.3s	0.178s

Table 2.3: ML models and their inference latencies across CPUs and GPUs

The main takeaway from this table is the increased portability achieved by HiT. All the results in table 3 above were obtained inside a container where the container had access to each of the hardware accelerators specified in the table in the corresponding system each of the containers was operating in. See table 1 for more information on what system each of the accelerators were located in and table 2 for their specifications.

Initially, we used Docker on the CRPL lab systems to circumvent the need for elevated privileges to install various software libraries. In order to achieve this, we still required a system admin with elevated privileges to install Docker and Docker Container toolkit [62] on the systems we were using containers with GPUs on. The benefits of containerization soon became apparent as we tried to request access to HPC systems and were preparing to run experiments on them once we gained access. That said, deploying containers in HPC environment brings its own set of challenges. It was important to allocate compute resources, especially the GPUs, in interactive mode on HPC systems for debugging purposes. Submitting jobs in batch mode is less efficient from a user perspective when ensuring proper functionality.

For all the models converted from a DL framework to ONNX, we insured accuracy of the final output during inference done on ONNX Runtime by comparing the

¹ The CPU in this table is an AMD Threadripper 1950X. For more details see Section 3.5A.

output predictions to the predictions made by the DL framework where the model was converted from. In addition, the process of converting models properly with the input and output names set to ensure proper inference after conversion.

ONNX also has many opsets because it is a library that is evolving and thus requires the developers of ONNX to update and add support for new mathematical operators necessary for DL algorithms. ONNX provides functions that help users verify the validity of the ONNX graph generated after conversion. ONNX's built-in validation code does this but only with the dummy input the user passes in before conversion to ONNX. It's important to note that this only helps confirm whether the output model graph has a valid schema. That is why it is necessary for users of ONNX conversion tools to device ways to verify the output model. We did this by passing same input data to the model in ONNX format and the model in DL framework at inference time. During our research using the DL models listed, we did not find any discrepancies with the validity of the model in ONNX format.

When converting a model from PyTorch to ONNX, the conversion tools that are natively provided by PyTorch use a technique known as *tracing*. Tracing uses the dummy input, which the user is required to provide to convert a model, to capture the structure of PyTorch model by executing once and recording the flow of inputs through the model to construct the Torch IR graph to convert. This technique is stable and well supported. But it has some downsides including lack of broad support, especially since it only supports models that use limited control-flow (conditionals or loops). That means *tracing* does not capture models that use control-flow. It only captures sequences of operations that occur on a single execution as the dummy input flows through the graph on the PyTorch model.

There is an alternative to *tracing* called *scripting* which supports all models because it converts syntax directly by first generating Python AST (Abstract Syntax Tree) and then using the JIT (Just In Time) compiler to do semantic analysis for conversion. This technique has several drawbacks including the need for code changes, and it only supports a subset of Python. The latter is a huge constraint because it

means that users might have to rewrite their Python programs to comply with the conversion requirements that use *scripting*.

When using models with operators that may not be supported well, it is important to use test data at inference time to make sure the outputs between the DL framework and ONNX format are similar. This is similar to what the converters does with the dummy input, which may not always be reliable, to ensure model consistency.

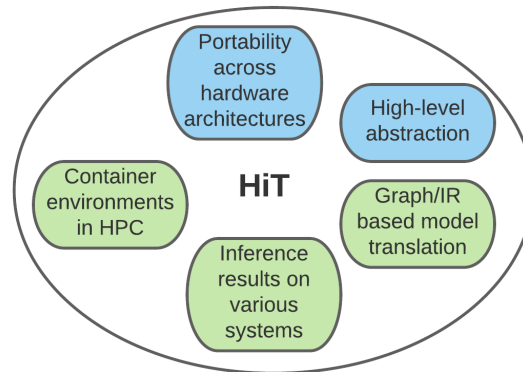


Figure 2.2: Figure showing the status of HiT. Green colored goals were demonstrated in this preliminary research and the blue colored goals will be completed in the next steps of this research.

2.5 Chapter Summary

This preliminary research started with the goal of improving DL application workflows using high-level abstractions. Experimenting across many frameworks, systems, GPUs, and container softwares gave us the opportunity to learn about the advantages and disadvantages of various techniques that exist. All that experience helped us formulate a high-level abstraction that we are confident will improve the portability of scientific applications in HPC. We came across a number of potential techniques, some we incorporated into our abstraction, that will not only help DL applications in terms of portability but also in terms of performance.

2.6 Lessons Learned

The first takeaway is that this is a very challenging problem to solve since it requires agreement on standardization from the major players in the field. That is a tall order mainly due to economic reasons but can also restrict the innovation making it very unlikely. Even with ONNX there are still many challenges that remain open questions because the software stacks that hardware vendors provide for developers has not caught up to the hardware they release. I anticipate this to always be a problem going forward as hardware architectures continue to evolve. The best solutions going forward will likely involve model graph-level intermediate representations to make ML workflows portable across software and hardware stacks.

Chapter 3

CANCER DRUG RESPONSE MODELS

This chapter focuses on the work done to improve drug discovery workflows by making them more portable and improving the predictive capabilities of the ML models.

3.1 Porting cancer applications to UDel’s DARWIN and non-DOE systems

Resources and codes such as AMPL (ATOM Modeling Pipeline)[55] and CANDLE (Cancer Distributed Learning Environment) [94] benchmarks were previously not tested in NIH compute clusters such as Biowulf. There is a need to port these resources to systems such as UD’s DARWIN system to test and document the process of running them. DARWIN was the first system to which NCI’s resources had to be ported to.

3.1.1 ATOM Modeling Pipeline (AMPL) overview

AMPL was the first to be ported to DARWIN, where its dependencies such as DeepChem had to be installed and verified. One of the key challenges faced when porting such libraries is the novel software ecosystems made available by each system. The processes and documentation of one system often do not work when switching to other high performance computing (HPC) clusters. Porting AMPL to DARWIN was not technically challenging when using the Anaconda package manager. Some challenges faced included problems such as DARWIN being limited to Python3.8 while the AMPL dependency DeepChem required Python3.8. Some problems are discovered only at run-time rather than during installation. So its important to thoroughly test all the models that are important. Sometimes HPC systems and clusters themselves are the problem. In such instances, there isn’t much that can be done except to wait for

the system or software issues to be resolved by the systems administrators. DARWIN was transitioning into the production phase when this work was initially attempted. That meant issues such as system downtime and software issues.

In addition to running AMPL directly on the system, the work also showed the AMPL library can be containerized using Singularity on DARWIN. This included making the underlying NVIDIA V100 GPUs accessible to the container running AMPL. In the next phase, this work was repeated on Georgia Tech’s Phoenix cluster. As noted previously, changing systems usually involves working with the available software and libraries to install and run the code and models. Initially, there were CUDA errors that were previously not seen on DARWIN. Eventually, a repeatable solution to make it work was established and documented.

3.1.2 Cancer Distributed Learning Environment (CANDLE) Benchmarks

After AMPL, CANDLE benchmarks were ported to DARWIN with the same goals as with AMPL where the model training would be tested on all the compute resources available and all the issues and challenges fixed and documented. For CANDLE, each of the Pilots (Pilot 1, 2, & 3) and every model in each of the Pilots had to be verified and tested on each system. This includes ten benchmarks in Pilot 1, one in Pilot 2, and 10 in Pilot 3. In many cases, there is a lot of overlap in the package dependencies between the benchmarks but in many cases each benchmark can have unique requirements. After successfully porting and validating CANDLE Benchmarks on DARWIN the process was repeated for GA Tech’s Phoenix cluster. The experience of going through this process for AMPL helped speed up the process for CANDLE.

In addition, the Phoenix cluster had one major advantage compared to UD’s DARWIN cluster. Phoenix cluster had a program that made it easier to run AMPL and access those notebooks on the user’s computer using the web browser. We replicated that functionality on DARWIN, which did not have a built-in program that does that, by creating a reverse SSH pipeline from the cluster to the user’s local machine. But the process is complicated and requires initial setup not only every time a user works on the

cluster but also every time the ssh connection disconnects. On the other hand, there were some things we could do on DARWIN that were not possible on Phoenix such as containerizing the code. Phoenix cluster prevented any use of container software preventing us from deploying the code using them.

3.1.3 ARM-based Accelerators

There were various efforts to run codes such as AMPL and CANDLE on ARM-based accelerators such as the A64FX processor found in the Fugaku supercomputer and also the Ookami test bed at Stony Brook and the Ampere Altra processor. We had access to the Ampere Altra through Oracle cloud but ultimately failed to get AMPL and CANDLE working on it. During preliminary setup of systems on Oracle Cloud's that use the Arm based Ampere Altra processors there were issues during installation of the Anaconda environment. Anaconda is necessary for applications such as AMPL and CANDLE. The conda environment used to run CANDLE Pilots such as P3B1 relies on GPU versions of some packages like Tensorflow. In addition to fixing the environment the code may have to be reconfigure to run on an Arm aarch64 operating system. One advantage the Ookami and the A64FX processor had was the option to use NVIDIA GPUs for acceleration which meant the existing packages and modules would work natively on this system. This work proved that resources such as AMPL and CANDLE can be ported to non-traditional accelerators such as the A64FX.

3.1.4 Accelerate NCI-DOE resources on next-gen NVIDIA A100s

The next task was to get access to and test the NCI-DOE resources on the next generation NVIDIA A100 GPUs. The first system we got access to with A100s was the JUWELS Booster system at the Julich computing center. Access to that system was brief and CANDLE was successfully ported to the system and the A100 GPUs. After losing access to that system, we obtained access to NERSC (National Energy Research Scientific Computing Center's) Perlmutter located at the Lawrence Berkeley National Laboratory. In addition to access to more NVIDIA A100 GPUs, NERSC

also has resources that make the system much more user friendly. One such resource is the Jupyterhub where the compute resources can be accessed using a Jupyter GUI (graphical user interface). Perlmutter was the end of the exploration of systems where NCI-DOE are ported to and tested on. At the same time, Perlmutter became the default system for ML workflows due to the software advantages it had compared to other systems previously tested.

3.2 Adapt models to tissue data

The advancement in high-throughput technologies has produced increased amount of data that has enabled design and delivery of precise cancer treatment plan for a specific characteristics of a patient. The data from these methods has also enabled the development of many computational models to predict the efficacy of these precision treatment plans.

As a part of the NCI-DOE collaboration various machine learning algorithms such as Combo, P1B3, Uno, UnoMT from CANDLE Pilot 1 project have been developed and benchmarked to predict drug response in various cancer cell lines. This pilot was built with gene expression and drug response data from the NCI-60 Human Cancer Cell Line Screen (NCI 60), NCI ALMANAC, NCI Sarcoma (SCL), NCI Small Cell Lung Cancer (SCLC), Cancer Cell Line Encyclopedia (CCLE), Genomics of Drug Sensitivity in Cancer (GDSC), Genentech Cell Line Screening Initiative (gCSI), and Cancer Therapeutics Response Portal (CTRP) studies. The genes that were focused for this pilot was obtained from the Library of Integrated Network-Based Cellular Signatures (LINCS) 1000 study. Most of the modeling done using these datasets have been trained using a single source of data and a single dataframe that encompasses drug response data, gene expression data, and drug molecular descriptors to support binary classification or regression machine learning models to predict drug response.

Though the cell lines-based algorithm provides insights into the drug response based on the expression of the genes it may or may not be the same in a tumor-normal tissue scenario. A cell line is a permanently established cell culture that will

proliferate indefinitely given appropriate fresh medium and space. Lines differ from cell strains in that they become immortalized. Previous studies by Ertel et. al[29] indicates that though most of the gene expression values in tumor-normal tissues are like the cell lines but the expression trend shows difference between tumor-normal tissue and cell line due to altered pathways contributed by growth conditions[29]. This research project will provide us with a scope to implement and assess all the machine learning algorithms developed as a part of this pilot on tumor-normal tissue data. Tumor-normal comparisons are crucial for identifying the somatic variants that act as driver mutations in cancer progression. In addition, it will aid us to identify the best performing algorithm for a particular cancer (such as breast, colon, lung, brain, skin etc.)

The long-term goal of the research is to show that the capabilities developed as a part of the NCI-DOE collaboration can be applied in various settings to show its versatility and adaptability. The objective of the current study is to implement and assess the machine learning capabilities developed as part of pilot 1 on a tumor-normal tissue data. The first objective of the study is to provide a comprehensive assessment of machine learning capabilities developed in Pilot 1 on tissue data instead of cell line data. The next objective is to review the performance of each of the machine learning methods on the tissue data. Finally, outline and show case the adoption and implementation of the developed capabilities on a different data to answer the following biological question. Can the machine learning methods developed as part of pilot1 be applied to tumor-normal tissue data? Another goal of this study is to find the best performing algorithm, such as Combo, P1B3, Uno, UnoMT, for the tumor-normal tissue data. And finally, its important to answer what model performs well for a particular type of cancer (such as breast, colon, lung, brain, skin, etc.) based on the tumor-normal data. The challenge is to find the corresponding drug response data for the tissue datasets.

3.2.1 Combination drug response predictor (Combo)

The CANDLE benchmarks include several pilots, including P1B1 (gene expression autoencoder model), P1B2 (cancer type classification), P1B3 (single drug response model), and Combo (Combination of drugs response model). The focus of the work that follows in the following paragraphs will be on the Combo model. The Combo model uses a pair of drugs to predict the growth value of cellines.

The first study focuses on replacing celline-based gene expression data with normal and tumor cell tissue normalized data. The dataset for this came from TCGA (The Cancer Genome Atlas) [87] program. For the initial tests, data was downloaded from existing studies in order to benchmark the models against published work. Using STAR as the aligner for tumor normal and tumor cell datasets for GTEx (Genotype tissue expression) [21] data in FPKM (Fragments per Kilobase of transcript per Million mapped reads) format.

The question here was whether the focus should be on a single type of cancer dataset or whether it should include all types of cancers or combinations of cancer types. We started with a colon cancer dataset, but there were immediate questions if a more balanced dataset should be chosen. That lead us to the thyroid tumor-normal tissue data. We replaced the celline data with tumor-normal data in the Combo drug response model to see how the existing Combo model would perform on new types of data.

To obtain drug response combo scores, real patient data was used as a test set when the model was originally trained on cell line data. The primary steps included

- 1) obtain a few samples of normal and disease data where these samples could be specific to one type of cancer.
- 2) perform normalization of RNA-seq data.
- 3) use the new normalized RNA-seq set to calculate a combo score and compare the pairs of scores with the cell line score.

This required changes in the Combo model inference scripts to accommodate new test data. This gave us information on whether the score changed when we built a new model with real patient data GTEx (control) and GDC (disease) to test with

patient dataset. In other words, we retrain the new model with additional new data and repeat steps 1, 2, and 3. The results did not show promise in this methodology.

3.2.2 Methodology

Obtain tissue data from public databases such as GTEX [21], TCGA [87] and drug response data from PharmacDB[31]. The data is then pre-processed to suit the model to which the data is applied. Split the data in to training, validation, and test sets. Run the model and optimize parameters using CANDLER for input datasets. Finally, evaluate and compare the performance of various combination of models and data.

3.2.3 Data gathering

Initially found a source for some tumor normal data in this work[91] discussing unifying data from different sources and the data is split as expected count, unnormalized, and normalized data. The data is further split by disease/organ/cancer types. While exploring the distribution of the available data[91], we noticed that the data distribution between the tissue-normal and tumor data was skewed for many organs/cancer types. Choosing data for a cancer type that has more of an even distribution seemed like the most prudent approach. So we settled on thyroid (THCA) data. Rather than using the normal GTEX data from this work[91] we are now focusing on using disease (cancer) data from GDC (NCI Genomic Data Commons)[56]. Gathering real patient data from GDC with the study's focus concentrated on TCGA data where some of the diseases (cancer types) of primary interest are included below.

Then the GDC-rnaseq-tool[74] was used to download the training data. This data is then filtered using the LINCS1000[85] gene list consisting of 934 genes. Then we normalize the data using log10 normalization. If necessary, batch normalization[66] will also be applied to this dataset.

Subsequently, the data was narrowed to a single cancer THCA for the initial test because the data from all the diseases are being used. The size of the data increases

GDC TCGA Cancer Types		
TCGA-BRCA	1222	
TCGA-KIRC	611	
TCGA-LUAD	594	
TCGA-UCEC	587	
TCGA-THCA	568	
TCGA-LUSC	551	
TCGA-PRAD	551	
TCGA-HNSC	546	
TCGA-LGG	529	
TCGA-COAD	521	
TCGA-SKCM	472	
TCGA-BLCA	433	
TCGA-LIHC	424	
TCGA-STAD	407	
TCGA-OV	379	
TCGA-KIRP	321	
TCGA-CESC	309	
Total	9025	

BRCA	Breast invasive carcinoma
KIRC	Kidney renal clear cell carcinoma
LUAD	Lung adenocarcinoma
UCEC	Uterine Corpus Endometrial Carcinoma
THCA	Thyroid carcinoma
LUSC	Lung squamous cell carcinoma
PRAD	Prostate adenocarcinoma
HNSC	Head and Neck squamous cell carcinoma
LGG	Brain Lower Grade Glioma
COAD	Colon adenocarcinoma
SKCM	Skin Cutaneous Melanoma
BLCA	Bladder Urothelial Carcinoma
LIHC	Liver hepatocellular carcinoma
STAD	Stomach adenocarcinoma
OV	Ovarian serous cystadenocarcinoma
KIRP	Kidney renal papillary cell carcinoma
CESC	Cervical squamous cell carcinoma and endocervical adenocarcinoma

SARC	Sarcoma
PCPG	Pheochromocytoma and Paraganglioma
PAAD	Pancreatic adenocarcinoma
READ	Rectum adenocarcinoma
GBM	Glioblastoma multiforme
ESCA	Esophageal carcinoma
TGCT	Testicular Germ Cell Tumors
THYM	Thymoma
KICH	Kidney Chromophobe
MESO	Mesothelioma
JVM	Uveal Melanoma
ACC	Adrenocortical carcinoma
JCS	Uterine Carcinosarcoma
DLBC	Lymphoid Neoplasm Diffuse Large B-cell Lymphoma
CHOL	Cholangiocarcinoma

Figure 3.1: Figure shows various TCGA cancer types and the data available for each disease

as expected and this requires us to move the workflow to a system such as DARWIN due to increase in memory usage.

3.2.4 Testing CANDLE Pilot 1 models on tumor-normal tissue data

Changes were required in order to test the existing COMBO model on tumor data. Specifically the existing model, trained on cell line data, must be able to accommodate the new tumor data to make predictions on it. First, the TCGA disease and normal data is filtered through the LINCS1000 gene list. The data was then narrowed to TCGA-THCA (thyroid) data. In addition to the disease samples in TCGA-THCA, there are also about 58 normal samples available to be tested, and this data had to go through the same data filtering and normalization pipeline made for disease samples.

The work resulted in a reliable data preprocessing pipeline to have COMBO model predict on TCGA data that is in FPKM-UQ format. The gene expression datasets for cancer cell lines, generated using RNA-Seq, were collected from the following sources: NCI-60, CCLE, and GDSC. The CTRP and gCSI drug response datasets were generated using the cell lines from the CCLE dataset. Hence, for those, the gene expression data from the matching cell lines were used in the CCLE dataset. The gene expression values were represented as FPKM values. The genes were filtered into

a common set of 17,743 protein coding genes or LINCS1000 genes, and the FPKM values were transformed into TPM values ($\text{FPKM} * 10E6 / \text{Sum of all FPKM values}$), and $\log(2)$ transformed. This allowed the pre-trained COMBO model to accept TCGA-THCA (thyroid) data to get Combo scores for each of the samples passed in for inference. Some changes made to the data during preprocessing include the following. The preprocessing steps were modified to make sure the list of genes, which make up the columns in the dataset used at inference time, are in the same order as the data for cell line expression samples. In addition, the tissue samples contain extra genes compared to the cell line data and the Combo model is only designed to take the number of genes in the cell line data. So after filtering the tissue data with the LINCS1000 genes list we also need to filter it by genes found in cell line data. Initially we chose the number of genes based on how many could fit in the Combo model layer but eventually fixed it to make sure we only use genes found in cell line data. There were more genes in the tissue data compared to the cell line data. The following is a subset of the results from comparing the Combo scores generated across various drug sets and comparing between various cell line sample sets and TCGA-THCA (Thyroid) tissue samples.

These tests were performed on various datasets. The following charts depict the results of the COMBO model trained on various datasets and predicted on each of the datasets where the purple dots represent the tissue data.

ALMANAC Dataframe and Plots

```
df_almanac.head()
```

	Drug1	Drug2	NCI60	NCIPDM	GDSC	RTS	TCGA_THCA
0	NSC.102816	NSC.102816	-6.536485	-5.884707	-6.916785	-5.858955	-6.067032
1	NSC.102816	NSC.105014	-6.758859	-6.172516	-7.251779	-5.880153	-6.050018
2	NSC.102816	NSC.109724	-6.694849	-6.543044	-6.824019	-5.726673	-5.648742
3	NSC.102816	NSC.118218	-6.651502	-7.064325	-7.033169	-6.427966	-5.969188
4	NSC.102816	NSC.119875	-6.557816	-5.946487	-7.234264	-6.836782	-6.295148

Figure 3.2: Example predictions for various data sets for model trained with ALMANAC data

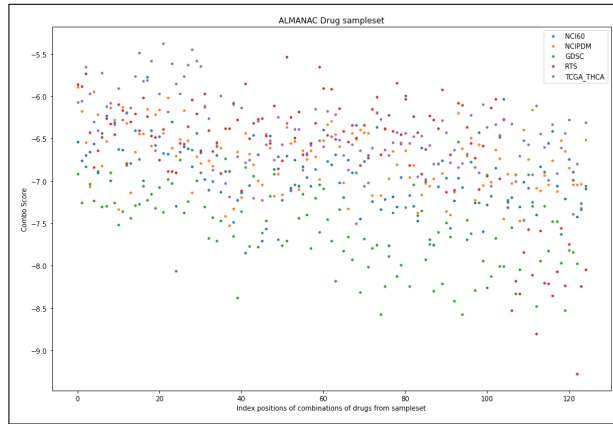


Figure 3.3: Plot of predictions for several data sets when model trained with ALMANAC data

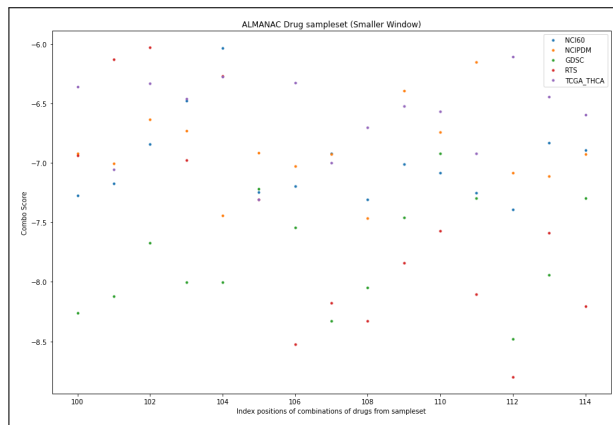


Figure 3.4: Plot of predictions for several data sets when model trained with ALMANAC data (smaller scale)

The results did not show promise for the model ability to train on cell line data and perform well on tumor data. This prompted a pivot back to focusing on improving the cell line data.

3.3 Improving CANDLE Pilot 1 models

3.3.1 Single drug response models

This is when the research backtracked to focus on simpler models due to the complexity of what was being proposed and experimented. The experiments shifted the

GDSC DataFrame and plots

```
df_gdsc.head()
```

	Drug1	Drug2	NCI60	NCIPDM	GDSC	RTS	TCGA_THCA
0	PubChemCID.10027278	PubChemCID.10027278	-7.187004	-7.236749	-5.109547	-6.362080	-6.155531
1	PubChemCID.10027278	PubChemCID.10074640	-6.993516	-7.195045	-5.032453	-6.566966	-7.073278
2	PubChemCID.10027278	PubChemCID.10077147	-6.848020	-6.743789	-5.484946	-6.864372	-6.674413
3	PubChemCID.10027278	PubChemCID.10096043	-6.773817	-6.751509	-5.552955	-7.046565	-6.427799
4	PubChemCID.10027278	PubChemCID.10109823	-6.972153	-6.771286	-5.977024	-7.194940	-7.044696

Figure 3.5: Example predictions for various data sets for model trained with GDSC data

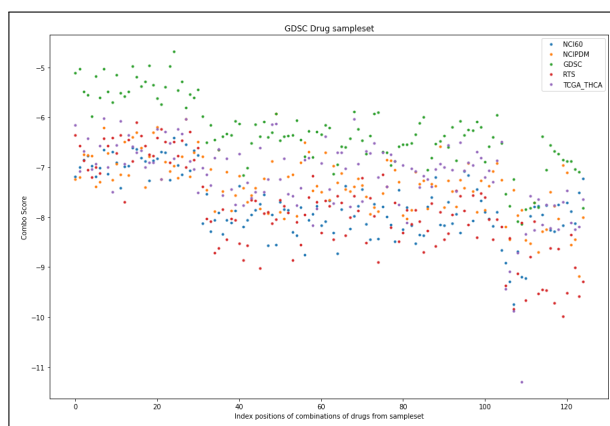


Figure 3.6: Plot of predictions for several data sets when model trained with GDSC data

Combo model to P1B3. The only difference between the two models was that Combo was based on the drug response of two drugs, whereas P1B3 (single drug response model) as the name suggests is based on a single drug response metric. P1B3 code was closely examined to determine the input data format. The focus then shifted to P1B3 for tissue data before applying those techniques in the more complex Combo model. To that end we looked at P1B3 and the baseline test accuracy is 67%. So we set out to find more data to retrain the model to improve its test accuracy. Next, P1B3 was retrained with NCI60's 20,000 protein coding genes dataset. The next task was to bring in more cell line data from other sources such as CCLE, CTRP, gCSI, GDSCm

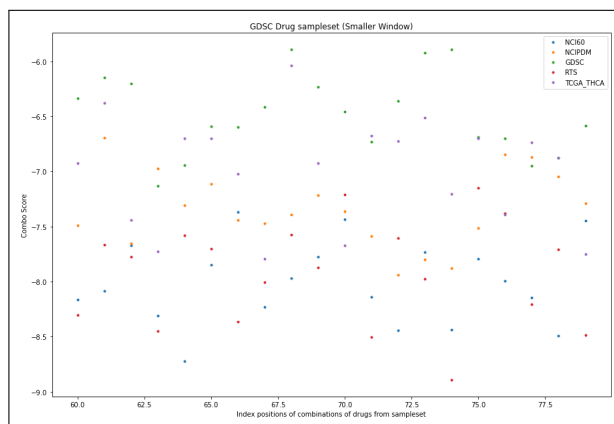


Figure 3.7: Plot of predictions for several data sets when model trained with GDSC data (smaller scale)

NCI_IOA_AOA DataFrame and plots

```
df_nci_ioa_aoa.head()
```

	Drug1	Drug2	NCI60	NCIPDM	GDSC	RTS	TCGA_THCA
0	NSC.102816	NSC.102816	-6.748287	-6.764788	-6.650192	-6.134347	-6.922942
1	NSC.102816	NSC.105014	-6.963635	-6.578068	-7.517027	-6.195334	-6.977890
2	NSC.102816	NSC.109724	-7.028354	-6.741415	-7.280639	-5.863832	-7.445742
3	NSC.102816	NSC.118218	-6.859385	-6.875876	-6.453597	-6.287112	-7.042890
4	NSC.102816	NSC.122758	-6.267356	-6.802461	-7.603879	-5.807427	-6.500591

Figure 3.8: Example predictions for various data sets for model trained with NCI data

and NCI60. Some of these datasets would have to be constructed using datasets from various sources in order to test them. The focus of the data at this time also shifted to CCLE datasets because that was a dataset of interest to our collaborators at NCI.

3.3.1.1 Cancer Cell Line Encyclopedia (CCLE) data preprocessing

Initially, when looking for new data for P1B3, one idea was to take data the Combo models datasets. Although Combo has expression and drug data from various sources, it would not suffice for P1B3. The problem was with drug response data that was available for a pair of drugs and their response instead for only a single drug.

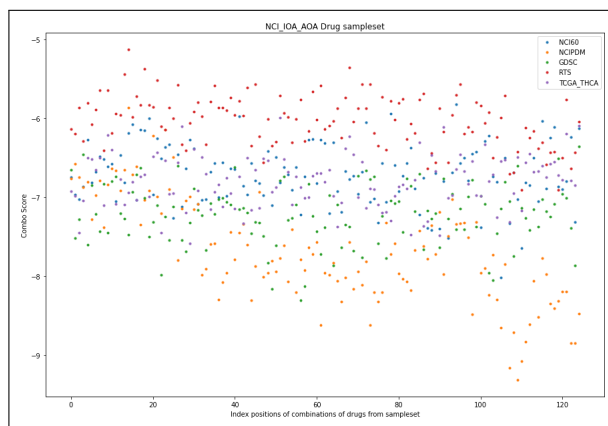


Figure 3.9: Plot of predictions for several data sets when model trained with GDSC data

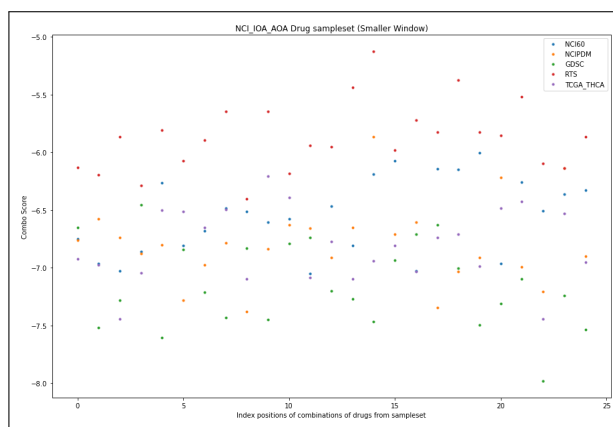


Figure 3.10: Plot of predictions for several data sets when model trained with GDSC data (smaller scale)

The corresponding drug response data for the CCLE expression data is GDSC. The DEPMAP portal[11] contains both the CCLE expression data inside a file named "CCLE.expression.csv" and the dose response data is found under "Sanger GDSC1 and GDSC2" where the corresponding files are "sanger-dose-response.csv and sanger-viability.csv". Another big change we made to the datasets that remained for the rest of the study was filtering of the genes in the rna-seq gene expression data from 20,000 protein-coding genes to 934 Lincs1000 genes.

There was an immediate data gap since the CCLE expression data contains a

RTS DataFrame and plots

```
df_rts.head()
```

	Drug1	Drug2	NCI60	NCIPDM	GDSC	RTS	TCGA_THCA
0	NSC.119875	NSC.119875	-7.194733	-7.089354	-6.081955	-7.171214	-6.711499
1	NSC.119875	NSC.123127	-7.210891	-6.947467	-6.195573	-7.071301	-7.241853
2	NSC.119875	NSC.125973	-6.694912	-7.984023	-6.199084	-7.256630	-6.758933
3	NSC.119875	NSC.256439	-7.008615	-7.558675	-6.040085	-7.535584	-6.606566
4	NSC.119875	NSC.326408	-7.545822	-6.990131	-6.182291	-6.888505	-7.334513

Figure 3.11: Example predictions for various data sets for model trained with RTS data

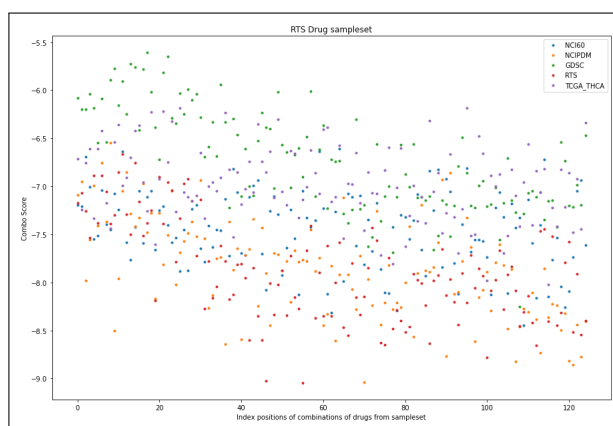


Figure 3.12: Plot of predictions for several data sets when model trained with RTS data

column named 'ARXSPAN_ID' while the GDSC dose response data did not. Instead, it had a 'COSMIC_ID' column. Had to resort to using another dataset called "fitted dose response" which contained both 'COSMIC_ID' and 'ARXSPAN_ID' to map the expression data to the dose response data. For the dose response data, the log of IC50 values is computed because the NCI dose response data is in that format. And filter it to keep only relevant columns such as "DRUG_ID", 'ARXSPAN_ID', "log_ic50", and 'Z_SCORE_PUBLISHED'. Concatenate the dose response of GDSC1 and GDSC2 and make a set of all COSMIC.IDs. All except for one can be found in the sanger-dose-response dataset. So we can find the corresponding ARXSPAN.IDs by creating a new

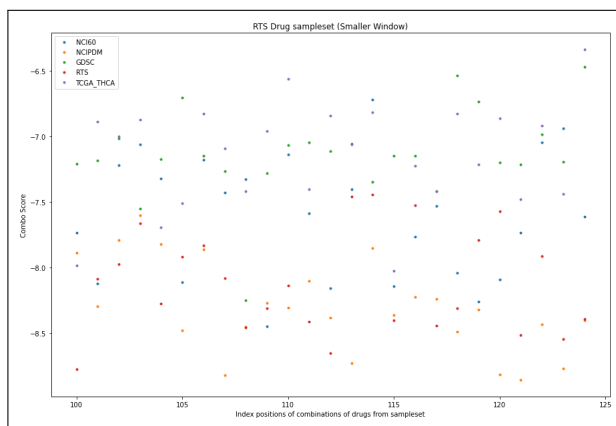


Figure 3.13: Plot of predictions for several data sets when model trained with RTS data (smaller scale)

dataframe that contains two columns of ‘ARXSPAN_ID’ and ‘COSMIC_ID’ and drop all duplicates from the sanger dose-response dataset. Here, a brute force for loop can take days to merge the data since the dataframe retrievals for each row is slow. An inner merge using ‘COSMIC_ID’ is much faster. Then the data had to be cleaned to remove missing values and finally the dose response data was filtered based on the drug descriptor availability and the two dose response datasets were concatenated for more samples.

When preprocessing CCLE expression data we filter it so that each of the samples has the corresponding drug response data in the final combined dose response data above. Finally, we have the drug descriptor dataset ‘pan_drugs_dragon7_descriptors.tsv’ from Modac[57]. Since this includes drug descriptors from all drug datasets, we only take the ones corresponding to GDSC. Filtered the dose response data one more time to only include the drugs for which the corresponding drug descriptor data exists. And finally format the data into the same shape, types, and form as NCI60 data to require minimal changes to the model training code.

One of the problems we ran into already is that calling the merge() function in Pandas dataframe to combine the datasets requires at least 500GB of CPU memory and overwhelmed even the system with the most single node memory we had access to.

P1B3 is designed to not merge all the data at once and instead to complete the merge in batches. We overcame this limitation by performing the final merge in compute jobs on systems, such as DARWIN and NERSC’s Perlmutter, where 500GB of CPU memory can be allocated.

3.3.1.2 CCLE results

P1B3 model trained on CCLE expression data and GDSC drug data performed poorly. Initially, the same parameters as the old model were used to train the model with the new CCLE dataset. It converges to its best accuracy in 10-15 epochs and had a test accuracy of 57%. This performed worse than the original NCI60 data which was 60%. NCI60 has much more drug response data, but on the other hand, it has much less RNA-seq gene expression data. The old NCI expression dataset had only 60 samples whereas the CCLE dataset has 700 samples. The expectation was for the new model to perform better. In terms of drug response data, the NCI dataset is much larger and more comprehensive compared to the sanger dose response dataset we found. The way the codebase preprocesses and merges the data takes the dose response data first and then merges in expression and drug descriptor datasets.

The results for various datasets made us explore alternate machine learning models and modeling techniques to improve drug response performance. Another dataset change we explored was the exclusion of drug descriptor data to test whether those features in the model added value.

At this point, the experiments and results gave us a good idea of what types of datasets are used for the class of research problem that is addressed in this work. All of the data was structured and tabular in nature. So we explored the literature to find ideal candidates for such types of data. Initially, our work was limited to adjacent work in cancer drug response models but then expanded the scope of our search. The search led to many studies that showed that tree-based models such as Random Forests and XGBoost are the ideal model candidates to test initially. We chose XGBoost because it is more robust for our use case.

3.3.2 Tree-based models

Any new XGBoost model trained must be compared with the existing model architecture in P1B3. P1B3 uses an MLP(multilayer perceptron)-based CNN model. Various datasets such as GDSC2 and CCLE were tested on the XGBoost model. The combination of models and data sets was tested to gain insight into the performance of each combination. XGBoost can be either a classification or a regression model. To predict cell growth, we use the regression model. This work was primarily done in the scikitlearn stack where the model, evaluation metrics, and hyperparameter optimization came from the built-in modules.

Various experiments were conducted to check whether the models learned on one type of data translated to accurate predictions on another type of dataset. One such experiment included training on the CCLE data set and a test set of NCI60 data. In order to run this experiment properly, the datasets had to be normalized. This is because various sources of data have various properties that must be normalized. When normalizing, one set of data is set as the reference data. In this case the CCLE dataset became the reference data where the NCI60 data was normalized according to the CCLE data.

Eventually, the focus of the study narrowed down to CCLE data. In addition, the metric the study focused on changed from the growth value of the cell line to the IC50 score. At the same time, the concentration value that represents the dosage of the drug was removed as a feature from the data set.

3.3.2.1 Data preprocessing

After the dataset was initially prepared and the model was trained there was a question regarding the dose response dataset. There seemed to be multiple dosage values for some drugs and thus multiple drug expression combinations for some drugs. This was because when the data set was put together, the GDSC1 and GDSC2 dose response datasets were combined. We finally decided to only use GDSC2 dataset. This change meant that the original data had to go through all the pre-processing steps.

3.4 Initial experiment

First, the XGBoost model was trained using recommended parameters such as `n_estimators=1000`, `max_depth=10`, `eta=0.1`, `subsample=0.7`, `colsample_bytree=0.8`. `N_estimators` is the number of trees the model constructs. `Max depth` is the distance from the leaf nodes to the head of the tree. `Eta` is the learning rate. `Subsample` is the percentage of data it randomly samples prior to growing trees to reduce overfitting. And `colsample_bytree` subsamples by tree rather than other factors such as level or node.

K Fold Cross Val.	R² Score	RMSE Score
1	0.936	0.24
2	0.935	0.235
3	0.942	0.224
4	0.945	0.222

Table 3.1: K-fold cross validation

Parameter	Val 1	Val 2	Val 3
eta	0.1	0.3	
max_depth	3	6	10
subsample	0.5	0.7	0.9
n_estimators	500	1000	2000

Table 3.2: Hyperparameter Optimization Values Tested

Parameters	Mean Test Score
'subsample': 0.9, 'n_estimators': 2000, 'max_depth': 6, 'eta': 0.1	0.96
'subsample': 0.9, 'n_estimators': 2000, 'max_depth': 3, 'eta': 0.3	0.93
'subsample': 0.5, 'n_estimators': 2000, 'max_depth': 10, 'eta': 0.3	0.84
'subsample': 0.9, 'n_estimators': 1000, 'max_depth': 10, 'eta': 0.3	0.90

Table 3.3: Results: Mean test scores for combinations of hyperparameters

Hyperparameter optimization was performed using randomized search rather than the grid search technique, and thus we may not have seen the best set of parameters. The best parameter values were: Subsample: 0.9, n_estimators: 2000, max_depth: 6, eta:0.1. Out of these only eta value from the standard model remained.

3.4.1 Train on CCLE data and test on NCI60 data

The next experiment focuses on testing whether an XGBoost model on CCLE data can perform well during inference on NCI60 data. Even though this seems straightforward to test we cannot just use datasets from two varied sources and expect it to perform properly in a reliable method. This is because each dataset has something known as the batch effect that is a distinct bias between datasets and the removal of such batch effects are crucial for any analysis done that involves comparing or combining the two disparate datasets. This effect is not just present in this scenario. The benefit of combining batches of data to enhance model capabilities through data augmentation is often inhibited by batch effects. Batch effects can be broadly defined as variations in technical factors across batches of data. Its important to have a method that removes batch effects while preserving biological signals in the data. Therefore, we use the batch correction method Combat-seq [97] to normalize batches and remove batch effects. Combat-seq uses a negative binomial regression model that retains the integer nature of count data in RNA-seq data. We utilize a Python library called pyComBat[5] that implements the technique described previously. We ensure that both the CCLE and NCI60 expression data are filtered using common genes since they become the features for the model. In other words, we only keep genes that are common in both datasets.

The results of this normalization process were not as expected because we originally normalized the CCLE/GDSC IC50 dose response values with log2. We could not do the same with NCI60 dose response values since many of them were negative and log2 normalization would make them NaN.

There was a question that came up after the experiment above asking whether retraining the XGBoost model is necessary after Combat normalization where the main variable being whether Combat changes the original dataset? If it does not and only normalizes the new data then model training would not be required, and otherwise it would. Initially, we were retraining the model because Combat normalization changed the values of both the CCLE and NCI60 expression datasets. The assumption was that the model had to be retrained on the Combat normalized CCLE dataset to predict on Combat normalized NCI60 expression data. That's not ideal for an end user who would just want to predict on their expression data rather than retrain a model for their particular dataset. The solution to avoid retraining the model is to specify a reference batch when normalizing the new dataset so that the CCLE data will remain the same, thus making it unnecessary to retrain the model for every new dataset an end user wants to predict with our trained model.

3.4.1.1 Change from growth to concentration as predictor variable

In terms of the features the model uses during training, one of the major changes we identified was to use the concentration values from the drug response data as an output variable along with the growth value since the user will not have that data to input to the model when making predictions. One of the difficulties of tree-based models is that they are generally designed to have a single output variable. So there will be challenges to adapting a tree-based model such as XGBoost to a multi-output regression model.

3.4.2 Experimenting role of drug descriptors as features

A few experiments included training models without drug descriptors as features. After a few iterations the conclusion was that XGBoost models trained with drug descriptors included as features always improved model performance compared to models trained without drug descriptors as features.

3.4.3 New gene expression data

After the results of the previous experiments conducted on XGBoost models with CCLE and NCI60 datasets either individually or in combination a new dataset was compiled. It was important to make sure the baseline model works as intended. The new gene expression data were already combat normalized, so the only major preprocessing needed was filtering with lincs1000 genes. One major difference between this dataset and previous datasets is that this includes both the IC50 and AUC values in the drug dataset.

When the model is trained with features such as gene expression values and drug data such as drug identifier and IC50 values with the AUC being the output variable, the model has an RMSE is 0.03 and R2 is 0.94. But the model seems to fail to converge when we make IC50 the output variable. The same happens when we make both AUC and IC50 the output variables. To test the importance of IC50 as a feature variable we took the IC50 out as a feature from the original model that performed well and the new RMSE is 0.07 and R2 is 0.73. So it seems like the IC50 value is an important feature for the model.

An experiment that was conducted included training on NCI60 data and testing on CCLE data. The CCLE expression and GDSE drug response data did not perform well on NCI60 data.

3.4.4 Multi-output regression models

One major goal for this experiment was to make the XGBoost model a multi-output regression model where it will predict more than one variable as outputs. This proved to be more difficult than expected since the XGBoost and other tree-based models are designed to have only one output variable. One solution is to use a wrapper class called MultiOutputRegressor that builds a separate tree for each of the output variables. As mentioned previously, our efforts to make both IC50 and AUC values output variables did not yield good results. It seemed like the model was failing to

converge as the error rates were extremely large for RMSE and large and negative for R2.

In another experiment, we tested how each of the drugs in the dataset performs in each of the cell lines. This meant that we get an output for every single drug. In order to achieve this, a change had to be made to how the data is fed to the model. The main changes included a change to the shape of the data and a model that can take the shape of the new data. Since the model with IC50 and AUC as output variable did not work, we wanted to experiment with something new. Inspired by work done in TUDGA[69], we wanted to have the model output a value for each of the drugs in our dataset. So when given a gene expression value the model will output a value for each of the drugs. We built this manually where we constructed a separate XGBoost model for each of the drugs in our dataset. Before we could set out to model this we had to change the format of the dataset for this new problem. The original training dataset had one row for each of the celline and drug combinations. We had to change this so that each drug would now have its own column in the dataset just as each gene in the gene expression has its own column. See data format example below.

	gene1	gene2	gene3	gene4	gene5	DrugID	IC50	AUC
celline1								
celline2								
celline3								
celline4								

Table 3.4: Initial Data Format

	gene1	gene2	gene3	drug1	drug2	drug3
celline1				AUC	AUC	AUC
celline2				AUC	AUC	AUC
celline3				AUC	AUC	AUC
celline4				AUC	AUC	AUC

Table 3.5: New Data Format

The implication here is that we cannot use all the drugs in our dataset since the matrix of all drug and celline combinations is sparse. In other words, we do not have data for every drug in our dataset for each celline. This means we have more data cleanup to do. We need to get rid of drugs that have a large number of null values and only keep those with minimal null values. Because we need a full matrix without null values, we still have to remove any cellines that do not have data for any of the remaining drugs we keep in our dataset. This will only be a few cellines and thus should not affect the data too much.

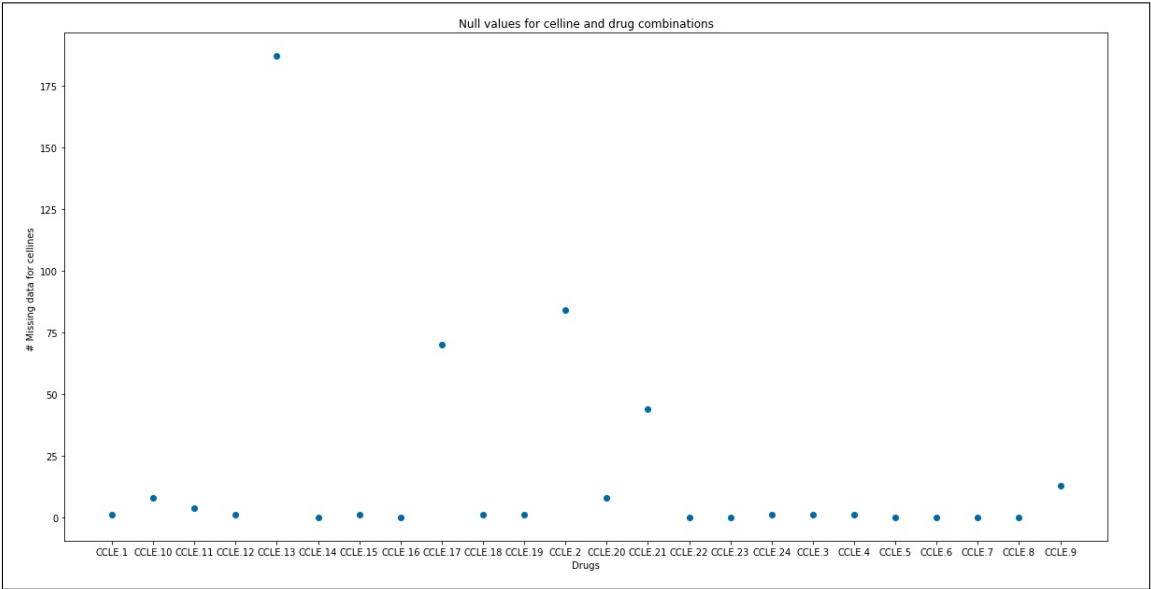


Figure 3.14: Plot indicates that most of the cell lines had very few data points. So the data was skewed by the few cell lines that had many data points

This ultimately proved to be an ineffective methodology. Building a separate XGBoost model for each drug made each of the models perform poorly. Each model had the same training data. Only the output variable, or the y values, was different. The data was missing features that differentiate between each of the drugs and was the conclusion of this experiment. The R2 error was always negative, which meant that the model had a hard time finding a relationship in the dataset. One flaw in this technique was that if there were 20 drugs in the dataset there would be 20 separate models, but

each of those models would train on identical datasets. We needed features that the model could learn to differentiate between each of the drugs. Drug descriptor data can be perfect for this. At this point, it seemed like we exhausted all ideas about training a model that works well without using drug descriptors. However, it is still important to show predictions from the model for each drug where a user can input cellines into the model and get predictions for each drug.

3.4.5 Computational metrics

The next step involved training the XGBoost model with CCLE drug descriptor data in addition to the existing CCLE expression and drug response data. This also meant that we no longer needed the training data restructuring we did in order to train numerous models. We could revert to training a single model that should learn features for all the drugs in the training data. After training on this new dataset the model performed much better compared to the training dataset without drug descriptors. We saw test accuracy in around 0.75 using the R2 error and 0.07 using RMSE error. In addition, we create a separate validation set that we put aside before training and included 10% of the cellines. The results were similar on this validation set. The predicted results can also be shown in a format that displays the predicted AUC value of every celline and drug combination. Fig. 3.15 is table where rows are celline names and columns are drug names.

DRUG	CCLE.1	CCLE.10	CCLE.11	CCLE.12	CCLE.13	CCLE.14	CCLE.15	CCLE.16	CCLE.17	CCLE.18
celline										
CCLE.697	0.816897	0.815736	0.458766	0.838903	0.524170	0.503839	0.883535	0.761090	0.830739	0.331380
CCLE.A549	0.813823	0.831542	0.487750	0.855860	0.607623	0.635966	0.891217	0.779165	0.873731	0.519351
CCLE.GCIY	0.727843	0.825050	0.493273	0.835842	NaN	0.610703	0.787389	0.741021	0.792341	0.574883
CCLE.HARA	0.804909	0.826365	0.500451	0.832993	NaN	0.612438	0.890629	0.778914	NaN	0.411794
CCLE.HCC1806	0.795179	0.759296	0.509218	0.774433	NaN	0.659702	0.766397	0.672768	0.850499	0.420345
CCLE.HCC2935	0.800697	0.789257	0.488551	0.858355	0.573109	0.630874	0.907704	0.629418	0.912466	0.448176
CCLE.HEP3B217	0.771761	0.803841	0.469095	0.850280	0.587152	0.551296	0.863317	0.651744	0.889185	0.452220
CCLE.HS746T	0.851679	0.865389	0.512534	0.851741	NaN	0.626984	0.861128	0.809554	0.852762	0.456422

Figure 3.15: Table of predictions for CCLE cell lines (rows) for each of CCLE drugs (columns)

	celline	DRUG	AUC_pred	AUC_true	AUC_diff
92	CCL.E.697	CCL.E.1	0.816897	0.7692	0.047697
93	CCL.E.697	CCL.E.10	0.815736	0.7777	0.038036
94	CCL.E.697	CCL.E.11	0.458766	0.3723	0.086466
95	CCL.E.697	CCL.E.12	0.838903	0.7869	0.052003
96	CCL.E.697	CCL.E.14	0.503839	0.4337	0.070139
...
10532	CCL.E.U118MG	CCL.E.8	0.820346	0.8317	0.011354
10533	CCL.E.U118MG	CCL.E.9	0.898613	0.9665	0.067887
10534	CCL.E.U118MG	CCL.E.13	0.610430	0.5646	0.045830
10535	CCL.E.U118MG	CCL.E.17	0.882105	0.9213	0.039195
10536	CCL.E.U118MG	CCL.E.2	0.868536	0.8486	0.019936

Figure 3.16: Table derived from Table 3.15 showing predicted, true, and margin between the two for each CCLE cell line and drug combination

- Parameters tested
 - ETA: [0.1, 0.3]
 - Max_depth: [3, 6, 10]
 - Subsample: [0.5, 0.7, 0.9]
 - N_estimators: [500, 1000, 2000]
- Best parameters: ETA: 0.1, max_depth: 10, subsample: 0.5, n_estimators: 500
- Mean test score for best parameters: 0.7
- Validation scores: RMSE: 0.07, R2: 0.75

Figure 3.17: Several hyperparameters tested for CCLE

When we used the best hyperparameters to train a new model as validation the R2 score was 0.75 on the test set and 0.74 on validation set while the RMSE error was 0.076 and 0.083 on the test and validations sets respectively.

3.4.5.1 Convolutional Neural Networks (CNN) results

Next, we wanted to compare the XGBoost model to the original CNN model from P1B3. So we trained the CNN model using the CCLE expression, dose response, and drug descriptor data. This involved making changes to the code that does preprocessing before the CNN model is trained to account for the new column names. Even though we are now using the AUC value as the predictor (output) variable and the CCLE expression and drug response data is new, the model performs similar to the original NCI60 data CNN model using mean squared error test accuracy. But when

we use the evaluation metric functionality and using the metric “root mean squared” error it converges to an RMSE of around 0.06 within 50 epochs. This is similar to what we saw with the XGBoost model.

Initially, we were unable to compare XGBoost and CNN head-to-head due to difficulty obtaining an R^2 score error metric from the CNN model built with Keras. The TensorFlow version would not allow us to use the R^2 score as a metric. To solve this, we had to use other libraries to obtain the R^2 score, but even then we can only do that at inference time. So we only have an R^2 score for the test error, not the training error.

The CNN model is doing poorly on the test set. R^2 score error is negative, which probably means that the model fails to converge. And RMSE is 5. For both these metrics, the value should be between 0 and 1 where lower R-squared is better while higher RMSE is ideal. The solution should be to perform hyperparameter optimization to improve model performance. There could also be issues with data preprocessing that cause this issue that must be identified and fixed. Improving the R-squared value based on the predictions on the CNN model trained with the CCLE dataset is important. To illustrate the difference between the CNN and CCLE model predictions, here is a sample comparison of the predictive capability of XGBoost on the left and CNN on the right. CNN model trained on the CCLE dataset does not seem to be reliable in predicting the output value. In some cases, as illustrated below, it predicts negative values.

The table below shows the results of various hyperparameter optimization tests performed and the subsequent R^2 score and the RMSE error values seen. As you can see below, the Tanh activation function appears to significantly improve performance, but the R^2 score is still negative. This may still mean that the CNN model is failing to converge and model the relationship in the dataset to reliably make predictions. The RMSE errors are much better and in the range where we expect them to be.

The results from CNN hyperparameter optimization of the model trained with CCLE data prompted us to switch back to NCI60 for our primary dataset that we will

	celline	DRUG	AUC_pred	AUC_true
92	CCL.E.697	CCL.E.1	0.816897	0.7692
93	CCL.E.697	CCL.E.10	0.815736	0.7777
94	CCL.E.697	CCL.E.11	0.458766	0.3723
95	CCL.E.697	CCL.E.12	0.838903	0.7869
96	CCL.E.697	CCL.E.14	0.503839	0.4337
...
10532	CCL.E.U118MG	CCL.E.8	0.820346	0.8317
10533	CCL.E.U118MG	CCL.E.9	0.898613	0.9665
10534	CCL.E.U118MG	CCL.E.13	0.610430	0.5646
10535	CCL.E.U118MG	CCL.E.17	0.882105	0.9213
10536	CCL.E.U118MG	CCL.E.2	0.868536	0.8486

Figure 3.18: XGBoost model predictions for CCLE data

	celline	DRUG	AUC_true	AUC_pred
0	CCL.E.697	CCL.E.1	0.7692	3.717297
2	CCL.E.697	CCL.E.11	0.3723	33.181919
3	CCL.E.697	CCL.E.12	0.7869	40.043743
4	CCL.E.697	CCL.E.14	0.4337	-45.146439
5	CCL.E.697	CCL.E.15	0.8721	40.931801
...
1082	CCL.E.U118MG	CCL.E.7	0.9969	38.088825
1083	CCL.E.U118MG	CCL.E.8	0.8317	33.303879
1084	CCL.E.U118MG	CCL.E.9	0.9665	145.045990
1086	CCL.E.U118MG	CCL.E.17	0.9213	39.331879
1087	CCL.E.U118MG	CCL.E.2	0.8486	62.234158

Figure 3.19: CNN model predictions for CCLE data

use to evaluate XGBoost and compare with the CNN model.

Once I switched to training and evaluating the XGBoost model with NCI60 dataset, I ran into an old problem that we left unaddressed from a few months ago. Due to the nature of XGBoost model training, it does not allow training in batch mode. CNN does, which means only the current batch of data must be merged to

Activation	Learning Rate	R ² Score	RMSE Score
Relu	0.001	-5282775	310
Relu	0.01	-20723	19.46
Relu	0.1	NaN	NaN
Tanh	0.001	-52.8	0.99
Tanh	0.01	-37.39	0.83
Tanh	0.1	-36.56	0.82

Table 3.6: K-fold cross validation

input into the model instead of the entire dataset. This saves us memory during the merge performed before feeding the data to the model for training. XGBoost takes all the data at the beginning before we start training thus all the NCI60 datasets must be merged prior to training. This regularly caused us memory issues on NERSC’s Perlmutter system. One solution would be to implement distributed pandas dataframe and distributed XGBoost training workflow to try and solve this issue. But that would take a while to reconfigure the codebase and is slightly complex but achievable. After speaking with our collaborators and testing various chunks of the dataset, we decided to train the XGBoost model on a limited number of drugs in the NCI60 dataset. Where the model performance did not differ no matter which chunk of the drugs we chose to train the model with. But this seemed arbitrary, so we wanted to come up with a more reasonable approach to filtering the data. An idea from our collaborator was to use FDA approved drugs to train the model.

3.4.6 FDA approved drug only models

First step was to gather the list of FDA approved drugs and then cross reference them with NCI60 drugs. I was able to find 459 FDA approved drugs in the NCI60 drugs list. But after cross-referencing with the drug descriptor data we have for all NCI60 drugs I discovered that only 159 FDA approved drugs in the NCI60 drug list had a corresponding drug descriptor row of data. This meant that our data set would have a maximum of 159 NCI60 drugs compared to the previous 10,000 randomly selected

NCI60 drugs. That also meant that I had to try a better drug descriptor dataset for NCI60 to increase the number of drugs in the training set to as close to 459 as possible. However, the effort was unsuccessful as I had to cross-reference multiple datasets because of the IDs being used by the drug descriptor dataset we found. In the end I found drug descriptor values for only 74 additional drugs and the dataset was in Mordred format where the drug descriptor columns of the new data found and the existing data did not line up, forcing us to filter columns to include only the common descriptor columns between Mordred and Dragon 7.

This meant that we were probably better off proceeding with the 159 FDA-approved NCI60 drugs for which we have drug descriptors. Therefore, testing with the drug descriptor dataset was ultimately unsuccessful. Now the plan was to train XGBoost with only 159 NCI60 drugs and then try to fix the memory issues of using the entire NCI60 drug set.

The XGBoost model built with 159 FDA approved NCI60 anticancer drugs had a R^2 error of 0.85 and an RMSE error of 0.06. The validation set with six cellines (out of 60) set aside had a R^2 error: 0.72 and a RMSE error of 0.09.

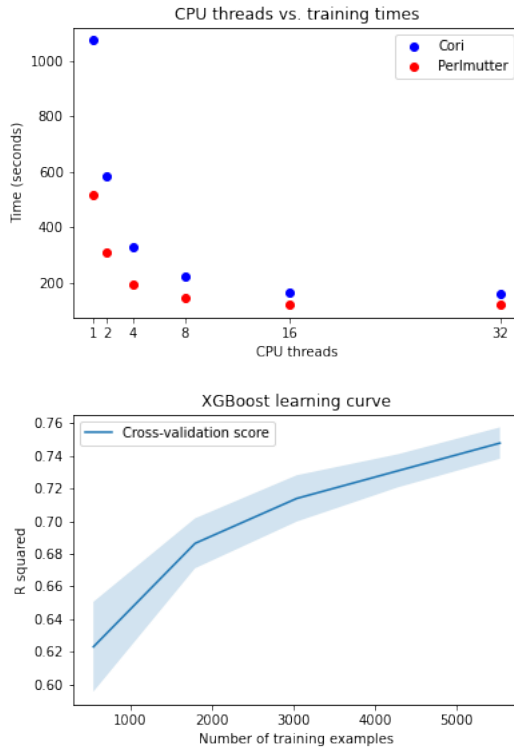
3.4.6.1 Results across several performance metrics

Cori and Perlmutter are systems located at NERSC at Lawrence Berkeley Lab. Cori is an all-CPU system while Perlmutter has the latest NVIDIA A100 GPUs. The original CNN model is capable of running on both GPU and CPU by design since it is built with the TensorFlow framework. However, the XGBoost model runs on CPU by default. To train XGBoost on a GPU we use the parameter ‘tree_method=’gpu_hist’ in the XGBRegressor function. As the results from the XGBoost model training show, training times are lower on Perlmutter on both CPU and GPU compared to training runs on Cori. Training XGBoost on Perlmutter’s A100 GPU had the lowest training time of 57.69 seconds. Since Cori does not have GPUs, we can only compare the training times across CPUs available on Cori and Perlmutter.

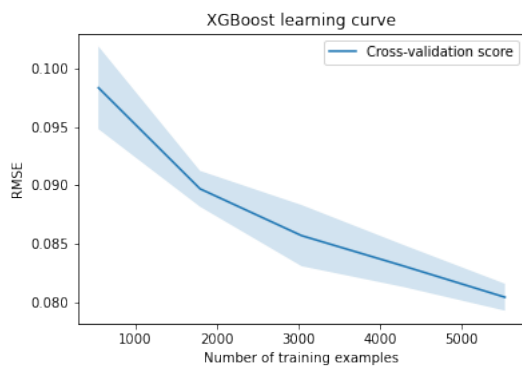
The table below shows the effect of strong scaling on the training time of XGBoost across Cori and Perlmutter. Across all thread counts, Perlmutter trains XGBoost faster than Cori, and both seem to train fastest when the thread count is around 16.

Threads	Cori - Intel Xeon	Perlmutter - AMD EPYC
1	1074.41	518.74
2	584.05	307.25
4	328.51	195.45
8	223.07	142.80
16	164.22	120.23
32	161.76	123.08

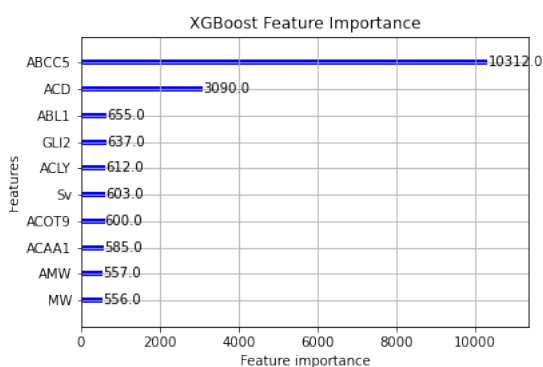
Table 3.7: Results using XGBoost on CPUs in Seconds



The chart above shows the accuracy when using the R2 error metric as the number of training samples increases.



The chart above shows the accuracy when using the RMSE error metric as the number of training samples increases.



This chart shows the most significant features in the XGBoost model. It seems like ABCC5 is known to have anti-cancer properties in multi-drug settings.

3.4.7 Focus on NCI60 data

The custom CCLE + GDSC drug dataset we put together was difficult to evaluate on the original CNN model, making a performance comparison with the new XGBoost model challenging. In order to have publishable results and a baseline to compare with the focus data set, we returned to the NCI60 data. The next chapter details the final results of that work.

3.4.8 Autoencoders for encoding features

There was an idea to use Autoencoders to encode features such as gene expression data or drug descriptors to see if the models we are testing would improve when

paired with part of the data encoded using Autoencoders. We managed to train the Autoencoder found in P1B1 with the CCLE dataset to have the model predict the type of cancer disease given the gene expression data. There was a lot of data preprocessing that had to be done to have the data ready for the model. The cell line names in drug response data and gene expression data did not match where one had ARXSPAN_ID while another had COSMIC_ID. We had to use another dataset that contained both ARXSPAN_ID and COSMIC_ID to map the datasets with each other.

The original data consisted of the GDSC2 TCGA classification (cancer type) and COSMIC_ID. CCLE gene expression data with FPKM values and ARXSPAN_ID. And finally COSMIC_ARXSPAN which matches COSMIC_IDs to ARXSPAN_ID. The goal here is to have data with cancer type and corresponding gene activation levels that match the format of the data. The process includes dropping irrelevant columns from GDSC2 (all we need is TCGA and Cosmic ID). Merge COSMIC_ARXSPAN into GDSC2 by COSMIC_ID so that gdsc2 now has arxspan. Rename ccle_exp “cellname” label with ARXSPAN_ID to match GDSC2’s label. And finally merge GDSC2 and ccle_exp on ARXSPAN_ID to have cancer type paired with gene activation levels.

Training Data	MSE	R² Score	Correlation
GDC	0.033	0.15	0.81
CCLE cell line	0.0085	0.62	0.94

Table 3.8: Autoencoder Results

Ultimately, the experiments run on Autoencoder did not show enough promise to replace or outperform existing methods.

3.5 Lessons Learned

The work and experiments described in this chapter lay the foundations of the work that follows in the next couple of chapters. The key takeaway from this chapter pertaining to making the cancer drug discovery workflows covered more portable is that portability can be challenging even when using hardware accelerators with the same

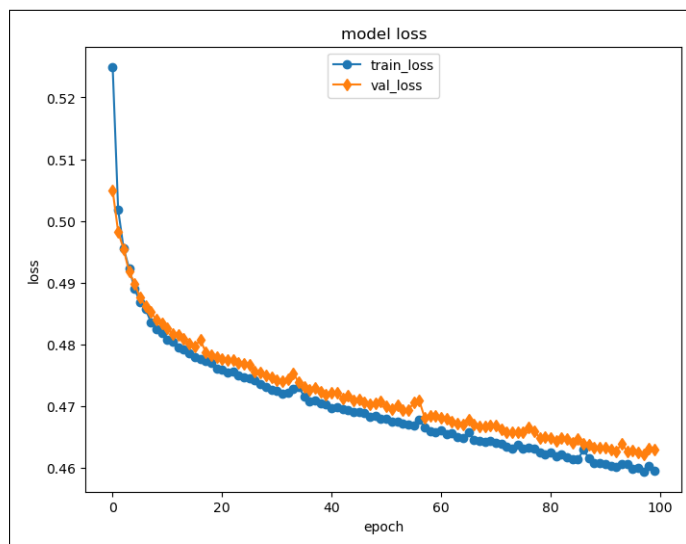


Figure 3.20: Autoencoder model loss plotted vs. training epochs

architecture. These challenges only compound as we switch hardware architectures, as we see in future chapters.

On the other hand, improving drug response models' predictive capabilities brought its own set of challenges. The endeavor to make cell line models work on tissue data yielded unsatisfactory results due to the lack of ground truth for tissue datasets. This was known when starting the work, but it was still worth exploring the viability of this idea. To make these models more robust for such approaches to work, we constructed a new CCLE dataset for training. When changing the data did not suffice, changing the model to XGBoost yielded better accuracy scores compared to the existing CNN model. The process of comparing multiple model types proved the need for a software abstraction to reduce friction for other ML practitioners to try such experiments with their own datasets.

Chapter 4

A NOVEL UTILITY FOR COMPARING NEURAL NET AND TREE-BASED MODELS

The use of deep learning (DL) is steadily gaining traction in scientific challenges such as cancer research. Advances in enhanced data generation, machine learning algorithms, and compute infrastructure have led to an acceleration in the use of deep learning in various domains of cancer research such as drug response problems. In our study, we explored tree-based models to improve the accuracy of a single drug response model and demonstrate that tree-based models such as XGBoost (eXtreme Gradient Boosting) have advantages over deep learning models, such as a convolutional neural network (CNN) for single drug response problems. However, comparing models is not a trivial task. To make training and comparing CNNs and XGBoost more accessible to users, we developed an open-source library called UNNT (A novel Utility for comparing Neural Net and Tree-based models). The case studies in this chapter focus on cancer drug response datasets; however, the application can be used on datasets from other domains, such as chemistry.

Advancement in data science, machine learning (ML), and artificial intelligence (AI) methods has enabled the extraction of meaningful information from large and complex datasets that have helped to better understand, diagnose, and treat cancer. The understanding of the drug response domain in cancer research has been accelerated with the development of ML models to aid in the prediction of the effectiveness of drugs based on a specific genomic molecular feature. In this study we developed a novel robust framework called UNNT (A novel Utility for comparing Neural Net and Tree-based models) that trains and compares deep learning method such as CNN and tree-based method such as XGBoost on the user input dataset. We applied this software to the

single drug response problem in cancer to identify the best performing ML method based on the National Cancer Institute 60 (NCI60) dataset. In addition, we studied the computational aspects of training each of these models where our results show that neither is evidently superior on both CPUs and GPUs while training. This shows that when both models have similar error rates for a dataset, the available hardware determines the model choice for training.

4.1 Introduction

To leverage machine learning (ML) for cancer applications, the National Cancer Institute (NCI) at the National Institutes of Health (NIH) in collaboration with the Department of Energy (DOE) established the Joint Design of Advanced Computing Solutions for Cancer (JDACS4C) program[7]. This program developed three pilot projects focused on cancer research: Pilot 1-cellular level; Pilot 2-molecular level; Pilot 3-population level[?]. Along with the pilots, NCI-DOE also developed the CANDLE (Cancer Distributed Learning Environment) [93] project for hyperparameter optimization (HPO) on the models.

Our work in this paper is related to a subset of drug response problems addressed in Pilot 1-cellular level. Specifically, our work builds on existing single drug response predictor models officially known as P1B3, which uses a deep neural network to model tumor growth based on gene expression, drug concentrations, and drug descriptors data. We compared the performance of the existing CNN-based P1B3, built within the CANDLE framework, with the new tree-based methods. We show that a tree-based method, like XGBoost, is a better model than the CNN neural network when the training data, such as drug response data, is tabular.

4.1.1 Background

Unlike computer vision models, which rely on unstructured data such as images, the CANDLE framework drug response, and other models, rely on structured data in tabular format. The big breakthrough in Deep Learning came because of the ability

of neural networks to perform well on unstructured data such as images. Deep neural networks have been successfully adapted to various domains outside of computer vision such as natural language processing (NLP)[4] and through the CANDLER framework to various problems in cancer research[93]. We tested the existing CNN model architecture used in CANDLER’s single drug response model with the NCI60 dataset, and it peaked at an accuracy around 70%. Further improving such models will require data augmentation or changing to a new model architecture.

Recent studies[82] have shown that tabular data may not require complex black-box models such as CNNs to perform well. Gradient boosted decision tree (GBDT) models such as XGBoost[16] can match or exceed the performance of deep learning models[82]. State-of-the-art deep learning models for tabular data perform worse than XGBoost when tested on new data not in their respective original studies[82].

In addition, more recent work investigates the differences between the models to help researchers understand the inductive biases of each type of model[36]. Another recent work conducts an in-depth survey comparing machine learning methods with deep learning approaches [9]. It also concludes that GBDT ensembles tend to outperform state-of-the-art deep learning models for tabular datasets [9]. On the other hand,[49] takes a slightly different perspective compared to the two previous works cited[9][36]. The authors of this work explore the properties of datasets that make them better suited for either Neural Networks or GBDTs. They find that GBDT models are better at handling skewed feature distributions compared to neural networks. This is yet another study conducted with the explicit goal of helping practitioners choose the best model for their work.

All of these studies [9][36][49] help researchers and practitioners understand the strategies that they should use for their own tabular datasets. In spite of all the analysis from previous work, there still remains a need to help researchers and practitioners, with their own tabular datasets, seamlessly compare the two model architectures rather than only rely on the analysis and comparisons using fixed datasets provided by recent studies. One of the major contributions of our work is to address that existing gap.

The following paragraphs summarize the main characteristics of the two types of models on which the analysis is based in the rest of the chapter. Tree-based models are supervised learning methods that create decision trees based on the training data provided. Decision trees are nodes in the tree-based model that create a 'split' at a particular point in the data range for a particular feature in the data, inferring rules during this process. This model uses the Exact Greedy Algorithm[16] for split finding. They can be used for both classification and regression problems.

The main difference occurs when the split occurs due to the different metrics used to minimize loss. And unlike decision trees, regression trees contain a score on each leaf that is used to calculate the final score by summing the leafs[16]. They are formally known as Gradient Boosted Regression Trees (GBRT) and are part of a larger class of methods known as Classification and Regression Trees (CART). During training, XGBoost builds decision trees sequentially and then uses a technique known as boosting, where each successive tree gives more weight to examples that were previously misclassified. After each iteration, XGBoost computes the gradients of loss functions based on predictions and then creates a new decision tree to reduce errors made by previous trees[16].

The CNN model consists of a multilayer perceptron (MLP) and is a feed forward neural network. It generally consists of an input layer, hidden layers, and an output layer. The input layer receives the data and the hidden layers learn a continuous function based on the training data. These consist of convolutional layers called filters (kernels) that slide over the input data and compute the dot product between their weights and the input data by computing the summation of the product of corresponding elements in two matrices[45]. Following this operation, known as a convolution, non-linearity is introduced into the network using an activation function to learn complex relationships between features. Then a pooling layer, also using a kernel, slides across the data to reduce spatial dimensions and overfitting[42].

A series of convolutional and pooling layers are typically followed by at least one fully connected layer to learn high-level features extracted by previous layers and

the relationships between those features. After the forward pass through the network, a loss function is used to compare CNN’s output to the ground truth and is used to update the network’s weights and biases using backpropagation and gradient descent. Backpropagation computes the gradient of the loss functions for each of the weights and biases in the network going back to the input layer and these gradients are used to update the model parameters to minimize loss using optimization techniques such as gradient descent[45]. Finally, the output layers perform predictions which can be classification or a numerical output in a regression problem. For classification, a softmax layer converts the raw output from the network into class probabilities. Many other regularization techniques such as dropout are applied to prevent overfitting and improve model performance[42]. This network architecture achieved state-of-the-art results in domains with grid-like unstructured data such as images but is not ideal for structured data.

4.2 Design and Implementation

Recent works such as [9], [36], and [49], which are complementary to our work, explain how various models and datasets they explore perform with tabular datasets. The extensive analysis in these papers[9][36][49] still leaves a gap where domain scientists and researchers need to validate and test with their own data the findings of the recent works cited. Their work further reinforces the need for an abstraction that allows users to quickly test their own data and see if the conclusions of the studies can be replicated. We address a major gap that remains because a user must eventually apply the findings in these studies to their own datasets.

To address this gap and provide users with structured (tabular) data with the option to compare both the CNN and XGBoost models, we developed an open source comparative library called UNNT that allows users to bring their data to train both models. See Fig. 4.1 for a flow chart that represents the library in S1 Fig. The XGBoost model relies on the open source libraries from Distributed (Deep) Machine

Learning Community (DMLC) which UNNT uses to provide users with the ability to train XGBoost models [16].

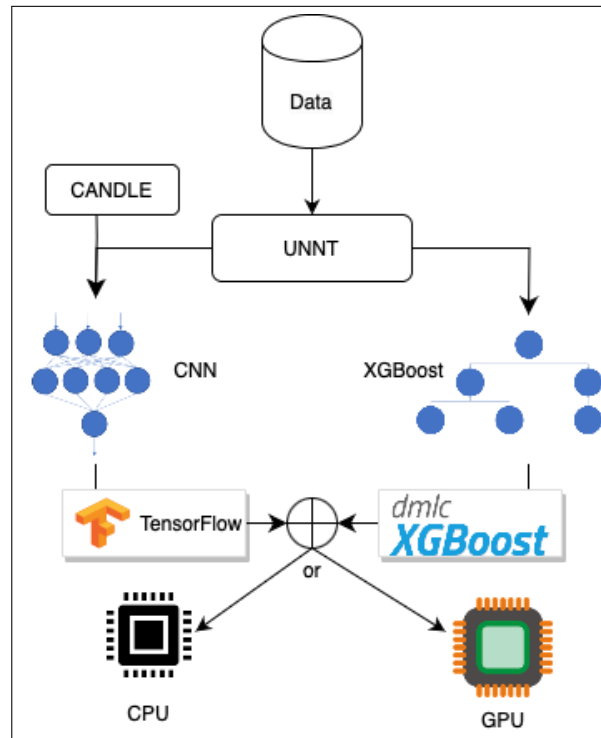


Figure 4.1: A flow diagram shows various parts that make up the UNNT software

For data preprocessing, calculating metrics after training, and displaying results, we rely on other packages such as Pandas, Scikit-Learn, Numpy, and Matplotlib. To build CNN models, we employ some of the functionality provided by the CANDLE library in Pilot 1 Benchmark 3 to do data preprocessing, model definition and instantiation, and model training. This has dependencies such as TensorFlow 1.0. In addition, we use scikit-learn to import metrics to quantify the performance of the model.

The preprocessing steps are important for the predictive accuracy of our pre-trained models to work on any data users bring to train CNN and XGBoost models. Because the exact format of user-provided datasets is unknown, users are responsible for any preprocessing and data cleaning steps that would be necessary prior to model training. Users need to decide what features they want to keep in the dataset being

used to train and test the models, and our models should be able to accommodate data of any shape as long as it fits into the device memory. If the data do not fit into the device memory of the system being used, it will require the user to distribute the data and compute across more than one CPU/GPU. Libraries such as cuDF [61] can be used to distribute data across NVIDIA GPUs and Dask [76] can be used to distribute compute across CPUs/GPUs.

Once users provide their data, UNNT splits that data into training, validation, and testing sets, and then the user can specify the percentage of data used for testing and validation. The defaults are 70% training data and 30% test data for XGBoost, while half of the test data is used for validation of the CNN model. We use scikit-learn's `train_test_split()` method to randomly sample since we are comparing it to a CNN model that randomly sampled. The library converts each of the data splits into numpy arrays to train using both CNN and XGBoost. There are several hyperparameters that can be set, for both CNN and XGBoost models, and we will have recommended default parameters which can be customized by the user. It is important to note that compared to XGBoost CNN models are more complex and have more hyperparameters. In many cases, there is little overlap between them. Users have full control of the parameters tested to find the best combination of hyperparameters for their dataset for each of the models.

Finally, UNNT provides users error metrics to evaluate both trained models using their data, including R^2 and root mean square error (RMSE).

4.2.1 The sources, type, and format of data sets and the changes made for this study

The data for the study was obtained from The Predictive Oncology Model and Data Clearinghouse (MoDaC)[57] data warehouse that was released as part of JDACS4C[7]. The dataset includes RNA-Seq expression profiles, drug response, and molecular drug descriptors for National Cancer Institute 60 (NCI60)[81] cell lines.

RNA-Seq expression data from these various cell lines are normalized using ComBat-seq [97]. The drug dose response data obtained from these cell lines are normalized using a Hill slope model with three bounded parameters [83] and the drug molecular descriptors were generated using the Dragon software package (version 7.0) [15].

The NCI60 data includes a combination of gene expression values[75], drug response values[59], and drug descriptors[60] found in MoDaC [57].

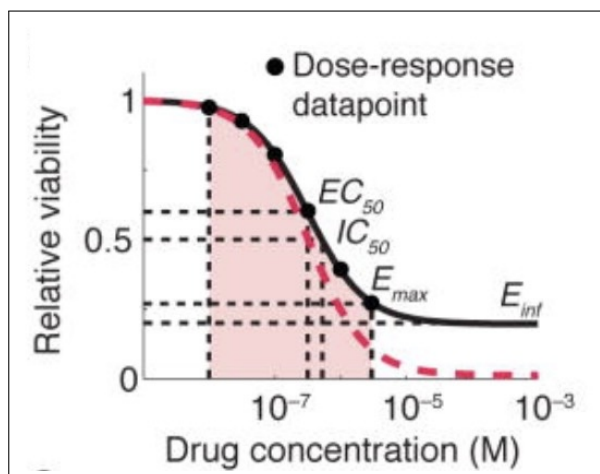


Figure 4.2: Figure from [30] shows how AUC is calculated for each drug’s viability based on its concentration

Our NCI60 data consists of data similar to the NCI60 data with which the original single drug response model was trained using the CANDLE benchmarks, with two main differences. Firstly, our data set uses only the lincs1000 genes[85] in the RNA sequence gene expression data instead of all the protein coding genes, due to the importance of those genes in the dataset. Secondly, the original NCI60 dataset in drug response data uses ‘Growth’ and ‘Concentration’ values where ‘Growth’ is the target variable and ‘Concentration’ is a feature we no longer use. The ‘Growth’ value was derived from the IC₅₀ data point depicted in Fig. 4.2 representing the concentration of a drug where the response is half its theoretical maximum[30].

The cell line drug response data is generated using the drug response curve,

see Fig. 4.2, and various points along this curve are part of the original dataset we obtained.

For our study, ‘Growth’ is replaced with ‘AUC’ data in the drug response dataset because the area under the curve (AUC) [30] is a more definitive parameter that combines both the potency (concentration) and the efficacy of a drug and is more robust when comparing a single drug across multiple cell lines for similar dose levels. We see this in Fig. 4.2 since IC50 is a single point of data along the drug response curve, whereas AUC includes the area under the curve.

4.3 Results

4.3.1 Experimental Setup

The compute resources for this work used NSF sponsored cluster, DARWIN, [78] at UDEL and Perlmutter[58], at LBNL. DARWIN and Perlmutter have GPU and CPU nodes. DARWIN uses NVIDIA V100 GPUs, while Perlmutter has the latest A100 GPUs. DARWIN has AMD EPYC 32 core processors which are similar to Perlmutter, which has an AMD EPYC 7713 64-core CPU.

4.3.2 XGBoost

We trained an XGBoost model using NCI60 expression, dose response, and drug descriptor data with AUC as the target variable (predictor). Our observed test accuracy yielded 0.83 for the R^2 score and 0.05 RMSE score. In addition, we created a separate dataset set aside before training and included 10% of the cell lines. Table 5.3 shows the difference in test errors between the two.

To find the best set of hyperparameters, we performed hyperparameter optimization using grid search technique and cross-validation. Grid search trains a new model for every combination of hyperparameters while cross-validation uses a different subset as test data to get an average across five subsets. The best set of hyperparameters found were ETA:0.1, Max depth: 10, Subsample: 0.5, N estimators:500. We used these hyperparameters to train a new model, and the results are shown in Table

5.3. Hyperparameter optimization led to a slight improvement, but was less than our initial expectations. This was a result of well-documented ranges for the various parameters and thus we happened to choose values that were close to optimal for each of the hyperparameters.

	R² Score	RMSE Score
Test error	0.82	0.051
Test error with new cell lines	0.76	0.065

Table 4.1: XGBoost Errors for Model Trained on NCI60 Data

XGBoost model training requires the training data to be fully merged before training commences and this resulted in memory issues. To solve memory problems, we experimented with smaller datasets and reduced the number of drugs from 30,000 to 159, based on a list of approved FDA drugs[59]. Results found in Table 4.2.

	R² Score	RMSE Score
Test error	0.84	0.069
Test error with new cell lines	0.66	0.094

Table 4.2: XGBoost Errors for Model Trained on NCI60 Data with Only FDA Approved Drugs List

4.3.3 CNN

We trained the original CNN model using the new NCI60 data as described in the data section above. The results shown in Table 4.3 were after applying hyperparameter optimization (HPO) and determining the best parameters where the learning rate is 0.01 and tanh as activation function. The model did not converge with other activation functions, such as ReLU, for the NCI60 data.

As shown in Table 4.3, the CNN model performed much worse than XGBoost when trained on NCI60 data. The best possible value of the RMSE metric is 0 and can go to infinity, but a value such as 0.81 does not give us good insight into the

Table 4.3: CNN errors for NCI60 after training after performing HPO

	R² score	RMSE score
Test error	-30.32	0.81

quality of the regression model. On the other hand, the negative R² value for the test score indicates that the performance of the model is poor, but its magnitude does not indicate how poorly it performed. The best value for R² is 1, meaning the model completely explains predicted data variability [18]. These results indicate that both R² and RMSE scores show that the XGBoost model outperforms the CNN model trained on this NCI60 dataset. It is important to emphasize that the choice of R² was highly dependent on the nature of the dataset used in this study and may not be widely applicable. For more analysis of the advantages of the R² score for datasets similar to what we used for this study, see [18].

4.3.4 Training times of CNN and XGBoost

The original CNN model is capable of running on both GPUs and CPUs by design, since it is built with the TensorFlow framework. While the XGBoost model runs on the CPU by default, it can also be trained on the GPU where we use the parameter *tree_method="gpu_hist"* in the XGBRegressor function. This means that both models in UNNT can be accelerated using GPUs. In the following section, we show comparisons of model training times with CPUs and GPUs. Comparison of CPU threads vs. single GPU performance shown in Table 4.4.

4.3.4.1 Training of full NCI60 drugs

In addition to the model built using only FDA approved drugs, we also built an XGBoost model using all the available drug data we have access to; however, the use of 30,000 drugs presented many challenges due to the volume. The main challenge to using the entire drug list was to find a system with at least 500GB of memory for the merged data before we train the XGBoost model.

Threads	Time (hours)
1	5.6
2	6
4	5.94
8	6
16	5.9
32	6.7
64	7.2
V100 GPU	0.3

Table 4.4: Results using XGBoost. Times for threads represents model runs on CPU with the corresponding threads used for speedup. Last row corresponds to running the same model on single NVIDIA V100 GPU

CNN models can have varying training times based on parameters used for training such as subsampling with fewer features, specifying fewer training, validation, test steps, and by reducing the number of training epochs. We discuss some of those results. Table 4.5 shows that the CNN model does not improve as the number of epochs increases, eliminating the benefit of higher epochs. And the difference between CNN and XGBoost training times is large. Table 4.6 shows that the CNN model converges to its optimal learning capacity in 1 epoch, hence it would still take three times longer to train than an XGBoost model trained with a V100 GPU in the best case scenario of 1 epoch.

Epochs	R ² score	RMSE score	Time (hours)
1	-30.46	0.821	2.3
5	-30.32	0.819	11.4
10	-30.32	0.819	22.96
15	-30.32	0.819	34.36

Table 4.5: CNN model trained using all 30,000 drugs as features on an NVIDIA V100 GPU

When comparing training times from Tables 4.5 and 4.6 we can see that the CNN model takes half as long to train on the CPU compared to training on the GPU. This is most likely a result of the size of the dataset, where data transfer from CPU to

GPU becomes a bottleneck and increases training time.

Epochs	R ² score	RMSE score	Time (hours)
1	-30.35	0.81	1.1
5	-30.31	0.81	5
10	-30.32	0.81	9.97
15	-30.32	0.81	15

Table 4.6: CNN model trained with all features on a CPU

	CPU (hours)	GPU (hours)
CNN	1.1	2.3
XGBoost	5.2	0.3

Table 4.7: Fastest training times for CNN and XGBoost on CPU and GPU (all features)

4.3.4.2 Training on FDA approved drugs

Table 4.7 shows us that an XGBoost model trains much faster on a GPU when training on a dataset that only contains FDA drug subset. We observed that the training times on the CPU increased as more threads were added. This occurs when the communication overhead is greater than the computational benefit of distributing a model across cores.

Table 4.8 shows the one instance in which the CNN model trains faster than an XGBoost model when training on a similar dataset. Comparing Tables 4.8 and 4.9, we see that the CNN model trains faster on a CPU even with less training data. The CPU results for CNN are not broken down by the number of threads because TensorFlow 1.0, the framework used to build the CNN model, does not support threading on CPUs.

Tables 4.7 and 4.10 show the advantage of training XGBoost on a V100 GPU that is consistently the fastest for the same data. Finally, Table 4.11 validates that using a GPU for XGBoost is optimal for training even with a smaller dataset.

Threads	Time (seconds)
1	1495.44
2	1527.33
4	1534.78
8	1549.24
16	1568.54
32	1599.01
64	1639.83
V100 GPU	160

Table 4.8: Results using XGBoost. Times for threads represents model runs on CPU with the corresponding threads used for speedup. Last row corresponds to running the same model on single NVIDIA V100 GPU

Epochs	R ² score	RMSE score	Time (seconds)
1	-29.76	0.81	130
5	-30.18	0.81	592
10	-30.45	0.81	1186
15	-31.02	0.82	2355

Table 4.9: CNN model FDA drugs trained on a CPU

Epochs	R ² score	RMSE score	Time (seconds)
1	-30.07	0.81	300
5	-30.96	0.81	1356
10	-30.33	0.81	2672
15	-31.35	0.81	5348

Table 4.10: CNN model FDA drugs trained on a V100 GPU

	CPU (seconds)	GPU (seconds)
CNN	592s	1356s
XGBoost	1495s	160s

Table 4.11: Fastest training times for CNN and XGBoost on CPU and GPU (FDA model)

4.4 Lessons Learned

Exploring a niche domain such as drug response modeling for cancer cell lines, we show that using a neural network (CNN) does not yield the best results. Instead, we show the impact of the tree-based XGBoost model over a CNN model, especially when the datasets trained on are tabular and run on a GPU. Our results demonstrate that, using the same dataset, an XBoost model is faster than a CNN model while running on an NVIDIA GPU. An observable downside to using XGBoost is the larger memory requirement for training, as documented in this work, and this varies depending on the size of the dataset.

As part of this work, we have developed a software, UNNT, that allows users to bring their own data and build models such as CNNs and XGBoost as well as compare how the models perform on their dataset. Domain scientists prefer to have such software utilities to streamline the process of using their proprietary datasets for training, analyzing, and more streamlined. Doing this manually is possible but time consuming and adds friction to the modeling work that domain scientists and other practitioners need to do in order to make such analyses possible without UNNT. As the related works show, it is possible to get general insights about what type of model should be used, but the models must still be trained, tested, and compared. Thus, UNNT makes a useful software for domain scientists to experiment with two unique model architectures for tabular data.

Chapter 5

SCALING ML SURROGATE MODELS FOR PLASMA PHYSICS SIMULATIONS ON FRONTIER EXASCALE SYSTEM

As machine learning models improved along with the data available to train them, GPUs became necessary to accelerate the matrix multiplication-based workloads that general-purpose hardware like CPUs have difficulty performing efficiently.

As the size of the data sets grew and the machine learning models increased in size and complexity, the matrix multiplication operations they perform needed to be parallelized. This is an area where GPU architecture is more useful for performing large-scale tasks in parallel. Modern GPU architectures include hardware components such as Tensor cores designed and optimized for the data structures used to store ML training data. Further advancements such as support for lower precision compute operations are also designed specifically for the type of datasets used widely in ML training workloads.

NVIDIA was at the forefront of the latest AI renaissance because NVIDIA GPUs had a history of performing well in areas such as gaming and scientific computing. Eventually NVIDIA GPUs became the default mode of computation for training and eventually inference of machine learning models. As the models evolved and became larger, there was a need for an increasingly large number of NVIDIA GPUs. At the same time NVIDIA not only improved the hardware capabilities, but also built a rich ecosystem of software to make the GPU hardware and software more widely and easily accessible to programmers. The Summit supercomputer at Oak Ridge National Laboratory (ORNL) contained NVIDIA V100 GPUs. Although the system was established mainly for scientific computing, the rise of machine learning workloads and their increasing scale made systems like Summit more important.

At the same time surrogate models became increasingly useful in reducing or completely replacing computationally intensive tasks in scientific applications. A surrogate model is a simplified approximation of more complex, higher-order models. Like any model, they map input data to an output, but the relationship between the input and output is computationally expensive to analyze and evaluate. This chapter of the thesis focuses on the work done to scale a surrogate model that extracts insights from large-scale plasma physics simulations. In the absence of this surrogate model, analysis of the simulation data generated would be too time consuming to conduct. Specifically, scaling this model was necessary because the surrogate model is part of a large workflow that extracts knowledge from large-scale simulation data, as it is produced, streamed to an online learning ML model using continual learning. This workflow was the first of its kind at the exascale level.

5.1 Challenges of creating a software stack for new hardware accelerators

The Department of Energy (DOE) for its next generation Exascale system decided to move away from NVIDIA to AMD GPUs. Even though AMD made GPUs for many years, mainly for gaming in the form of ATI's GPU business, which they acquired, GPUs for machine learning is a new domain. Most importantly, AMD did not have the software ecosystem that machine learning developers, frameworks, and existing code bases relied on in terms of CUDA and all the supporting software stack NVIDIA built and maintained on top of CUDA. This presented a challenge for anyone who needed to run machine learning workflows on the AMD MI250X GPUs in the Frontier Exascale system at ORNL. Frontier is a top-1 system making it imperative that machine learning workloads were supported and comparable in performance to the current generation alternatives from NVIDIA.

AMD's GPUs, even after Frontier went into production and was widely used for scientific computing applications, lagged in terms of software support for popular modern ML frameworks such as PyTorch and TensorFlow. There was a need to test machine learning models on the latest AMD MI250X GPUs. This chapter details the

work done and results of the efforts made to not only run ML models on AMD GPUs, but also run them at scale.

5.2 Frontier exascale supercomputer

Frontier, one of the fastest supercomputers in the world, is equipped with 37,632 AMD GPUs and consists of over 9,408 nodes along with 9,408 AMD EPYC CPUs and represents the computing power of 1 quintillion, or 10^{18} calculations per second. Several real-world applications such as ExaSMR, EXAALT, PELE, PIconGPU, WarpX, E3SM and CANDLE need abundance of memory and compute cores to speed up problems that might otherwise take months or years on a petascale system, or such computations may not even be attainable on a non-exascale system. Not only in the HPC space, but in the large-scale Generative AI space as well, training Large Language Models with many parameters (in the range of billion to trillion parameter model) would require exascale scale computing resources so that we can reap the full potential of what AI can offer towards addressing complex scientific challenges.

5.3 Background

In preparation for the Frontier exascale supercomputer to be deployed into production, OLCF (Oak Ridge Leadership Computing Facility) at ORNL selected eight research projects to participate in a program called Frontier Center for Accelerated Application Readiness (CAAR). The main application in this chapter is one of the eight selected projects.

The reason testing and scaling high level ML frameworks such as PyTorch on Frontier and AMD MI250X GPUs is necessary is due to the increasing use of ML frameworks not only for highly compute intensive tasks like training foundation models but also for ML applications in various domains where ML applications will be a part of a broader application pipeline that includes HPC applications. In such instances, it is imperative that both the HPC code and ML code run and scale as expected on the

new hardware architecture of Frontier. The focus of the work presented in this chapter is on the ML applications that run using the PyTorch framework.

Most ML applications train models using a technique known as "offline" training where the training data is stored on the file systems of HPC systems until the ML application training starts when it is moved into the memory hierarchy of a modern HPC system. There are various techniques employed by various model training strategies where the data is loaded in batches, whereas for some types of models, like XGBoost, all at once. Ultimately, the data moves from storage to memory and is then removed from memory. The workflow described in this chapter involves a workflow where such a setup is not feasible, making the online training methodology important to implement. The following section describes the application and setup that requires an online training methodology.

5.4 Particle-In-Cell on GPU (PIConGPU) as an application test case

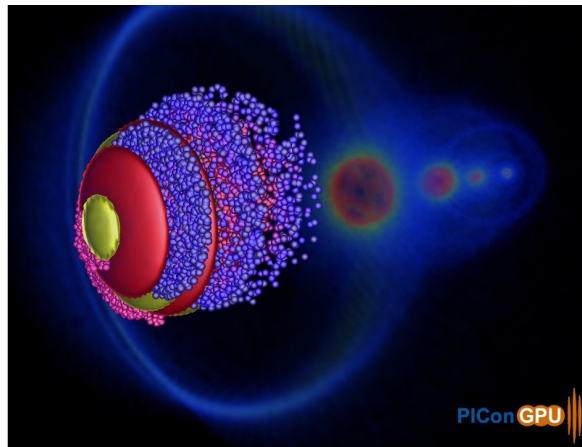


Figure 5.1: The figure depicts a real-time vector field visualization test of PIConGPU. Credit: Rene Widera, HZDR.

HPC simulations such as digital twins of earth produce extremely large quantities of data using the computational power of Exascale systems such as Frontier. The use of pre-trained ML models for in situ inference is becoming an increasingly popular

technique in such scientific applications where the analysis is done in place as the data is generated.

PIConGPU [12, 13](Particle-In-Cell on GPU) is a plasma simulation code that has been at the forefront of developing central technologies for GPU-accelerated plasma simulations. It was the first plasma simulation code to run performant simulations across various hardware architectures using a single code base, using the alpaka library [96], frequently demonstrating excellent scaling and performance on the top ten high-performance compute systems in the world over many years [53].

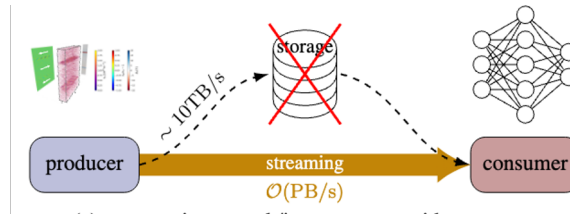


Figure 5.2: The figure shows how streaming eliminates the need for data storage on disk by streaming the data from the producer (simulation) to the consumer (ML). Credit: Jeffrey Kelling, HZDR.

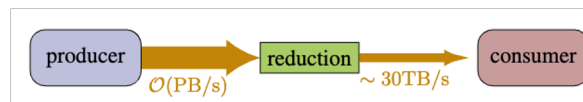


Figure 5.3: A figure representing the real-time data reduction that's necessary even with streaming because the volume of data produced by the simulation is so large. Credit: Jeffrey Kelling, HZDR.

Applications such as PIconGPU create situations where the stream of data produced by the simulation is at such a large scale that data reduction demands in-transit analysis. In such cases, even if the volume of data is reduced to the minimum extent suitable for analysis, storage of such data for further offline analysis is technically infeasible due to the physical constraints of the file system at hand. It also leads to a mismatch in throughput between the memory hierarchies of modern HPC systems necessitates solutions for online analysis of data generated at high rates and volumes

making offline analysis impossible. This work focuses on such a use case with in-transit training of machine learning models at extreme data rates where training the model requires the majority of the data produced by the source.

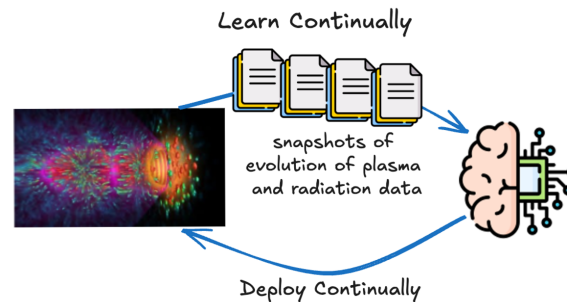


Figure 5.4: A visualization of continuous learning method used by PIConGPU to learn and extract insight in real-time from simulation data streaming from the producer to the ML model. Credit: Sunita Chandrasekaran and Richard Pausch, HZDR.

The model being trained itself is of such complexity that it can in principle incorporate the complexity of the total volume of data produced by the source, while at no time a total view of the data is available. Unlike most ML training techniques that rely on iterative training, [34] which requires permanent storage of datasets as the models learn over multiple epochs, online learning trains the model from streams of data over time. This work uses continual learning because the data is produced and discarded immediately after it is used for model training. By streaming data into an ML framework, it becomes possible to learn a reduced representation of the data and uncover correlations, providing a solution to the challenge of conducting exploratory studies without predetermined analyses. The main challenge of this work is to integrate particle-in-cell simulations via data streaming with an ML framework such as PyTorch.

The goal is to showcase a workflow that integrates a particle-in-cell simulation of the Kelvin-Helmholtz instability (KHI) in PIConGPU [13], supported by openPMD streaming [71], with a PyTorch [22]-based ML training code. This coupling aims to uncover correlations between emitted radiation and particle dynamics within the simulation in an unsupervised manner. This study demonstrates, for the first time, an

in-transit learning workflow at scale.

A key goal in many areas of plasma physics is to probe plasma dynamics at the fundamental particle level at high spatial and temporal resolution, e. g. using powerful radiation sources such as free electron lasers at X-ray wavelengths. In many situations, such powerful probing diagnostics are not available. In these cases, the radiation emitted from the plasma, especially in the case of electrons moving at relativistic velocities, provides a complementary diagnostic because the motion and acceleration of charged particles are directly linked to the radiation emitted by them. Collecting this radiation in all its details would in principle allow for reconstructing the complete phase space dynamics of the plasma at high spatial and temporal resolution without the need of a probe. A complete phase-space reconstruction from detailed radiation spectra is an ill-posed inverse problem. This work focuses on the well-known relativistic Kelvin-Helmholtz instability (KHI) [13] to highlight the challenges in solving an extreme-scale inverse problem via a complex HPC workflow coupling a GPU-accelerated plasma simulation to a large-scale in situ neural network for in situ learning. Already in 2013 [13], it was shown that the computation of high quality, high resolution observational data such as radiation spectra can only be performed in situ, as storing the ground truth data, the phase space trajectory data of each particle in the simulation at each time step, is impossible due to the aforementioned throughput hierarchy.

Training an ML model for this task requires connecting the phase-space data to the radiation data in order to allow for inversion, but it was clear that these phase-space data could not be stored, which is a major bottleneck. As an example, scaling to a moderate 25% of the Frontier system, one would be faced with 1 PByte of data for every time step. With total time steps per simulation of the order of a few 1,000 time steps and time step durations of the order of 0.1 to 1 s, we frequently observe data rates of 1 to 10 PByte/s for particle trajectory data and theoretically require total volumes on disk of the order of 10 EByte. It is clear that the only way to train a data-driven model to reconstruct the phase space from the radiation is by streaming the particle and radiation data from the simulation directly into the model for training.

The following steps detail the complex in-memory workflow to reconstruct local phase space dynamics from radiation.

- Simulate a sufficiently large domain at high spatial and temporal resolution to capture both the local plasma dynamics as well as the collective contributions of different plasma regions to the total radiation spectrum.
- Calculate in situ the radiation spectrum directly as the sum over contributions from all radiating particles in the simulation at each time step, at high angular and spectral resolution and bandwidth.
- Collect for each time step the particle phase space data, positions, and momenta, and the complex amplitude and phase for each wavelength and direction of the radiation.
- Prepare the data for an ML model by finding a suitable representation of spectral and phase space data.
- Stream that data for each time step to an ML model and asynchronously train the model with that data.
- Create new simulation data and train the model simultaneously with every time step.
- Stop the simulation after observing the complete physical process under investigation.

The following steps detail the technical setup employed for the exascale data streaming and continual learning.

- Start a scalable, GPU-accelerated PIconGPU simulation of sufficient size and resolution, utilizing a large subset of nodes on Frontier.
- Schedule PyTorch and N/RCCL along with the PIconGPU simulation. (N/RCCL - NVIDIA and ROCm collective communication library) Optimize scheduling for data transport and locality, physical system size and resolution, simulation and training time, total time of solution and use of resources.
- Stream for each time step particle momenta and positions and spectral data to a large-scale ML model.
- Transform at each time step the phase-space and spectral data from its simulation format to an optimum representation for the ML model.
- Train the model with data from each time step.
- Repeat the workflow for a sufficient number of time steps to cover all relevant stages of the plasma instability simulation with respect to the dynamical evolution and the amount of training data needed.

5.4.1 Workflow setup

5.5 Scaling ML on AMD GPUs

The application workflow described in the previous section contains two distinct workflows that work in conjunction. The PIConGPU simulation and openPMD streaming applications are HPC applications consisting of the first part. The ML training code is the second part of the workflow. The focus of this section will be on work that was eventually incorporated into the ML training and scaling aspect of the workflow from the previous section.

The starting point for running PyTorch models on AMD GPUs is a proxy application known as the "toy" model that was designed to run in offline training. The code `scaling_benchmark.py` built a toy model using randomly generated data where the model and the data were proxies of the actual model, which was still under development at this point. The team was using systems with NVIDIA V100 GPUs to test the model during its development. The only benchmark of the proxy code and data running on an AMD GPU was work done on the testbed system Crusher. Crusher is an OLCF system at ORNL that was supposed to replicate the hardware on the then upcoming Frontier system and contained 192 nodes each with 4 AMD MI250X GPUs. And each MI250X contains 8 GCDs with the `scaling_benchmark.py` code validated on 1 GCD on Crusher. The following are the results of that work.

Coupling blocks (complexity)	V100	A100	MI250X (1 GCD)
2	70.29	38.65	209.25
2	70.06	38.67	217.76
2	69.95	38.66	209.57
4	139.45	75.25	372.74
4	139.38	75.26	366.06
4	139.18	75.25	369.84
6	208.89	111.93	522.86
6	208.44	111.88	513.73
6	208.53	111.94	513.94

Table 5.1: Runtime of `scaling_benchmark.py` across various GPUs in seconds (lower is better)

Coupling blocks (complexity)	V100	A100	MI250X (1 GCD)
2	7.11	12.93	2.38
2	7.13	12.92	2.29
2	7.14	12.93	2.38
4	3.58	6.64	1.34
4	3.58	6.64	1.36
4	3.59	6.64	1.35
6	2.39	4.46	0.95
6	2.39	4.46	0.97
6	2.39	4.46	0.97

Table 5.2: Million particles/s for scaling_benchmark.py across various GPUs (higher is better)

Before I could begin working on this, I had to apply for system access which is a longer process compared to getting access to systems previously mentioned such as UDEL’s DARWIN or NERSC’s Perlmutter. System access requires an interview for identity verification and an RSA 2-factor token.

After gaining access to the system, it was important to familiarize with the system software, job submission, and set up a workflow that makes it seamless to access the system on a daily basis. As mentioned in Chapter 2, switching systems creates a small learning curve because the same task can require different steps on each of the systems. In addition, unlike Perlmutter, Frontier and OLCF do not provide a Jupyter environment-based interface to make it easier to access the system. The workaround is to use VSCode’s Remote window feature to add Frontier as a host. This makes it slightly more convenient to access, manipulate files, and write code on a remote system.

The first step was to replicate the conda environment from Crusher from the previous experiment. In addition, there were a few changes made to the software setup when switching from Crusher to Frontier. It is important to load ROCM software and install a version of PyTorch with the ROCM version compatible with the rocm module loaded on Frontier. Instead of amd/5.2.0 the new method for loading ROCM is module load rocm/5.2.0. And then install rocm specific PyTorch using ”pip3 install torch –no-cache-dir –index-url https://download.pytorch.org/whl/rocm5.2.0”. And this also

upgraded PyTorch to 2.0 specifically for more AMD hardware support included in the 2.0.

The baseline performance was established on Frontier, and the following table shows the results.

Coupling blocks (complexity)	Runtime (s)	Particles/s
2	60.46	8.26
4	113.53	4.04
6	169.69	2.94
6	231.61	2.15

Table 5.3: Baseline training runtimes in seconds on Frontier (1 GCD)

5.5.1 Use of PyTorch DistributedDataParallel (DDP) for scaling

The results presented so far for runs on Crusher and Frontier have a caveat because they are running on 1 Graphics Compute Die (GCD). Since each AMD MI250X GPU has 2 GCDs, it means the code is not utilizing a full GPU’s compute capabilities. This is an idiosyncrasy specific to AMDs latest GPUs where to increase performance of a single GPU dies from what would be two GPUs were combined into 1. From a software perspective, each MI250X GPU is visible as 2 GPUs. The implication here being that it requires distributed code to fully utilize a single GPU. So we have to write distributed training code in order to test the full GPU’s capability and benchmark it. Going forward, unless explicitly stated the use of GPU will refer to GCDs.

There are various techniques to make a machine learning model training code run across multiple GPUs. TensorFlow and Keras have built in modules that allow this feature, but there are also examples like the Horovod framework that works across various popular ML frameworks such as TensorFlow, PyTorch, and MXNet. Similarly, PyTorch also has its own built-in module called DistributedDataParallel (DDP) for making model training multi-gpu and multi-node. Since PICongPU’s existing code

was written in PyTorch and because of more support for AMD accelerators and ROCM software suite, PyTorch DDP was the choice to distribute model training.

After initially implementing PyTorch DDP in the `scaling_benchmark` code, we decided to benchmark the software stack involved on Frontier using known DDP code examples so that we can test it on NVIDIA systems to check if the issues we were seeing were AMD GPU or system-specific. Some third-party benchmarking codes included the following. The first one is an MNIST code example and various versions of this code were used while experimenting including basic multi-gpu, multi-node, and a model parallel example.

After some initial issues, the `scaling_benchmark.py` code worked on 8 GCDs (4 GPUs). The fix was to add some environment variables below.

```
export MIOPEN_USER_DB_PATH="/tmp/my-miopen-cache"  
export MIOPEN_CUSTOM_CACHE_DIR=${MIOPEN_USER_DB_PATH}  
rm -rf ${MIOPEN_USER_DB_PATH}  
mkdir -p ${MIOPEN_USER_DB_PATH}
```

Figure 5.5: An example figure showing select environmental variables that were necessary for running distributed training.

Both data parallel and model parallel solutions were tested and scaled up to 8 GPUs. Data parallel is a model training technique where a copy of the model is distributed to each of the GPUs and each model copy gets a different batch of training data. Once each model computes its gradients, all the instances of the model must do a collective all-reduce communication to average the gradients. On the other hand, model parallelism involves distributing a single instance of the model across GPUs (typically in the same node). Often, model parallelism and data parallelism are combined for training at scale.

5.5.2 cINN model with static data

In the next phase of experiments, we move on to code that uses an invertible CNN model since it is close to the model architecture used to learn from the simulation

data being generated. The following is a result of the first experiment that compared training times on MI250X and A100 GPUs.

MI250X GCDs	Runtime (s)	A100 GPUs	Runtime (s)
1	0.94	-	-
2	0.98	1	13
4	1.01	2	6.5
6	1.09	3	3.3
8	1.24	4	3.3

Table 5.4: cINN model training MI250X vs. A100

The following graph also shows a visualization of the data presented above.

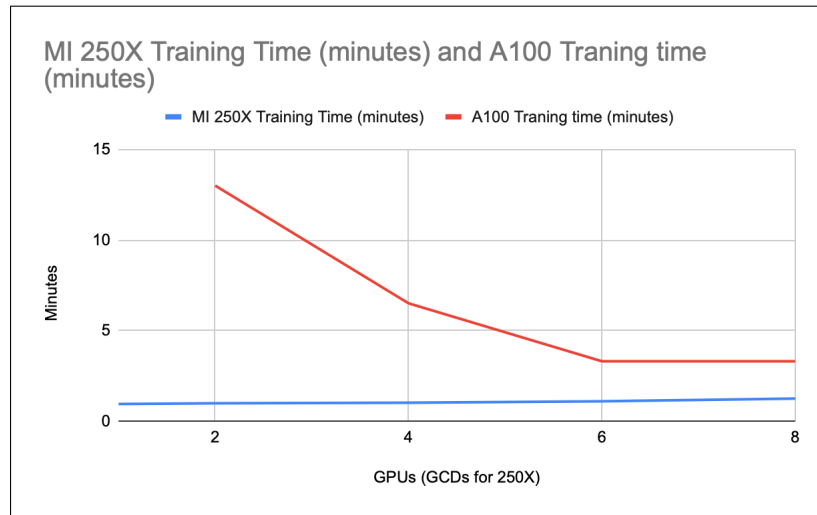


Figure 5.6: Comparison of training time in minutes between AMD and NVIDIA GPUs across various number of ranks.

Even though the MI250X GPU seems to be outperforming the A100 the training times increase as more GPUs are allocated, making it likely that the code is failing to properly train across all 8 GPUs (GCDs). To test this hypothesis, the code was profiled and the resulting image below shows the code allocating memory and distributing data to all 8 GCDs but only one 1 GCD does all the computation explaining the performance results where allocating 1 GCD is the fastest.

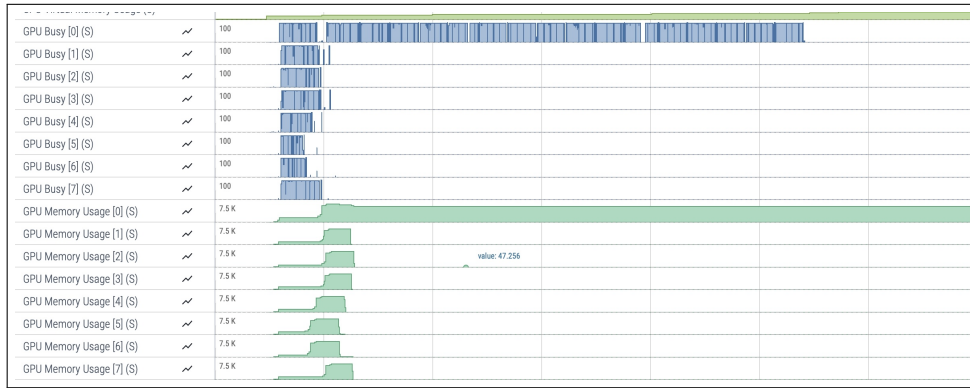


Figure 5.7: Image of a profile taken while training a model across 8 GCDs in a single node.

5.6 Scaling model training

Initially, it was unclear to us whether the technical difficulties we were facing could even be overcome. There were many communication issues we encountered with PyTorch DDP and the underlying software and communication protocols used on Frontier. One of the most common issues we encountered was the 'RendezvousConnectionError'. It was difficult to trace the origin of that error. In other words, it was hard to tell if that error was an issue with our code and how DDP was implemented in it, or if the ROCm version of PyTorch was causing the issues, or if Frontier's hardware or interconnects were the problem.

For a very time even after we started working on scaling the models on Frontier, it was not very clear if we were the first or only team to work on this on Frontier. There was no documentation about the software setup and job submissions for distributed ML training on Frontier. The first indication that there were teams working on similar workflows came from a paper [23] discussing scaling LLM training on Frontier.

When faced with these challenges, it was important to use process of elimination to figure out where the issues were originating. In order to do that, we needed another system where we could replicate the same software setup. We obtained access to AMD's Accelerator cloud (AAC) which had the AMD MI210s, which although are not an exact replica of the GPUs on Frontier, would suffice for benchmarking and

for comparing software rather than performance. Replicating the workflow on AMD's AAC cloud system led to more work since the process to load software and submit jobs is slightly different between every system, but it was still very useful to validate all the experiments and tests we were running on Frontier and validate the issues encountered. Although this helped us through some hurdles, not all challenges were resolved in this process. The next step was to change the codes and try a code base that has been validated on one of the two systems with AMD GPUs. AAC cloud has a guide on how to run Megatron-DeepSpeed on their system, so once we replicated their steps our next goal was to repeat this and adapt the steps and installation process to run on Frontier. Successful completion of this process validated that Frontier can run such workflows in a distributed methodology.

5.6.1 Challenges faced while scaling on Frontier

One big flaw we found on the Frontier system as we ran various codes, models, and data training when experimenting was the delay that is caused when running something in the compute nodes before training starts. This delay is caused by the conda environment being copied and shared to the compute nodes. The solution, which was recommended by OLCF staff, was to use 'conda pack' to package a prebuilt conda environment and export it to NVMe storage using sbcast and when running it use the environment located on the NVMe. Even though sbcast introduces some overhead, ultimately its much faster when running a conda environment at scale.

We eventually figured out the configuration settings to run ML training using PyTorch DDP at scale as we went from single-node to multinode setup. There are small DDP code changes necessary when going from single-node to multinode. This process of scaling and creating a repeatable setup for distributed training on Frontier was a long and arduous but was ultimately very useful to guide the PICongPU team when scaling the PyTorch-based ML training code. This part of a larger application workflow scaled to 100 nodes on Frontier.

A unique and important requirement of the presented workflow is the ability to not only run but also scale on a large number of Frontier’s AMD MI250X GPUs. Software libraries for machine learning on AMD GPUs are maturing as we speak. Our machine learning model is small enough to fit in a single GCD; therefore, parallel training of this model is done using data parallelism where copies of the model are distributed across GCDs with each copy of the model receiving different chunks of data to train on. Once each model computes its gradients, all the instances of the model must do a collective all-reduce communication to average the gradients. We expect the scaling of the workflow to critically depend on the optimization of this all-to-all communication in PyTorch DDP, since communication within the PIconGPU simulation is only between next neighbors.

Running PyTorch workflows on Frontier requires installing a specific version of PyTorch in either an Anaconda environment or a python virtual environment with a corresponding ROCm software module. Configuring the environment, including resolving dependency conflicts between software packages necessary for the custom fork of ADIOS2 [33] and PyTorch with ROCm. PyTorch DDP uses the RCCL backend for communication needed to perform distributed training across GPUs within and between compute nodes.

We experienced a hard limit to scaling in our current setup: The all-to-all communication between PyTorch DDP ranks using the RCCL backend hits system limitations on the possible number of open sockets beyond 100 nodes. Potential options to circumvent this limit include using a libfabric backend for N/RCCL or using PyTorch DDP’s MPI backend. While the RCCL backend was able to support PyTorch up to the scale used in this study, we can already conclude that further extensive evaluation of the communication backends within PyTorch will become mandatory for stable software foundation for our near future scaling studies on Frontier.

5.7 Main takeaways of PIconGPU in-transit ML

- The is the first work to demonstrate at scale that the coupling between simulation, data streaming, and in-transit distributed ML can be used to extract scientific

knowledge and perform analysis.

- This workflow was orchestrated on what was then the top-1 supercomputer (now 2nd) and overcame many challenges that came with porting this workflow to evolving hardware accelerators. The path to go from 1 GCD to multi-node distributed ML training was challenging because of the software libraries required to achieve this.
- The workflow relied on data reduction due to the scale of data produced by the simulation and the ML model achieved partial reconstruction of the plasma distribution, and the model was able to clearly classify the instability regions in the plasma simulations.

5.8 Lessons Learned

The ML models using docking and MD simulations will require scaling up on the Frontier exascale supercomputer, so the experience gained as a result of this work is invaluable for this and other future works.

The initial case study that we focus on to scale ML workflows on Frontier is a plasma physics code named PIconGPU with an HPC component that also scales on Frontier. The coupling between simulation, data streaming, and distributed ML training was the first time such an in-transit learning workflow was demonstrated at scale. This work also proved the viability of running distributed ML training on AMD GPUs such as the MI250X. It is important that Frontier, as a top-1 system with these GPUs, can run ML workflows at scale as. The implication is that ML researchers and practitioners now have more hardware architectures to reliably run ML workflows at scale.

When we first set out to do this, it was highly unclear whether scaling ML models beyond 1 GCD (1/2 GPU) on AMD MI250X would even be technically feasible. The path to go from single GCD to distributed multi-node training was not clear because the software libraries and system environment and support necessary to make this happen were not documented anywhere.

The process involved crafting the combination of libraries that would make it a reliable and repeatable process to install the dependencies. OLCF as an organization

was pivotal for us to pull this off. Much of the documentation that is now available on how to run such workflows on Frontier was nonexistent when we worked on this just a few months ago. The only information that was available came as blog posts from AMD stating that it was now possible to run such workflows on AMD's MI250X and similar GPUs. We were left to figure out everything else.

If this was an NVIDIA powered system with the latest GPUs, such as the H100s or the next-generation Blackwell architecture, the process would have been much more seamless with many previous examples and familiar errors when fixing problems that would come up. Due to the influence of NVIDIA's software stack on AMD's ROCm software, it was often challenging to diagnose certain errors that were occurring. The lack of proper software and documentation certainly played a huge role in making this a more difficult process than it needed to be.

To have a landscape of interchangeable hardware that work across standard ML libraries, it is important for hardware accelerators to provide the software abstractions and translation layers for portability.

Chapter 6

IMPROVING SURROGATE MODELS IN DRUG DISCOVERY

This chapter focuses on protein-ligand binding and pose estimation in drug discovery problems and is a progression of the work described in Chapters 2 & 3. The workflows described in the earlier work solve problems using SMILES strings which are processed to create 2D representations that are analyzed from the drug descriptors. Several types of such representations exist including MORDRED, which we used in previous work, MOE descriptors, and others. The limitation of such descriptors is that most of these descriptors fail to account for the three-dimensional geometry and interaction potential of the molecules. The main reason why many descriptors don't go beyond the analysis of the 2D molecular description to represent the descriptors has to do with the fact that there are many 3D configurations for many molecules for any given 2D molecular description.

6.1 Background of IMPECCABLE

To explain the contributions made in this chapter of the thesis, it is necessary to explain how the contributions made fit and contribute to a larger effort with greater prominence in terms of the goals and contributions its pushing towards. The work and contributions described in this chapter directly impact and improve a surrogate model that aims to reduce computationally intensive molecular dynamics simulations in a larger drug discovery pipeline. In addition, the work described in the previous chapter, focusing on scaling ML workflows on Frontier, and the insights gained there greatly helped expedite and achieve some of the goals of this work.

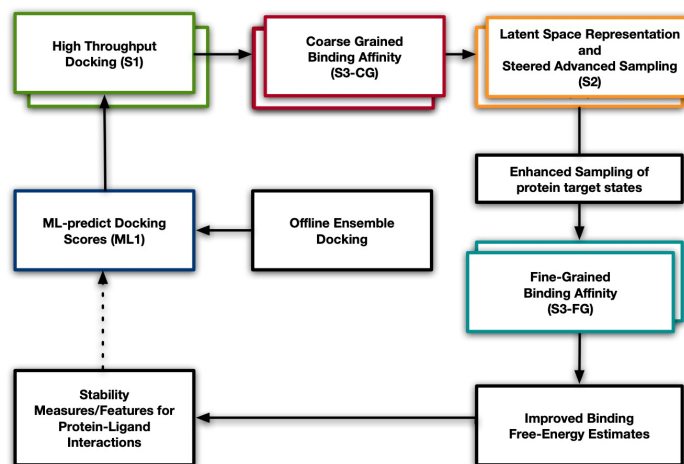


Figure 6.1: Figure adapted from IMPECCABLE [79] to visualize the description provided in text

6.1.1 IMPECCABLE: AI-enabled drug discovery campaign

The drug discovery process typically used in the pharmaceutical industry is a time-consuming and expensive process with development times that can stretch as far as a decade. In chapters 2 & 3 we explored some *in silico* methodologies that have the potential to improve the drug discovery process, and existing methodologies can be further improved to better select lead compounds that proceed to later stages of the drug discovery process, making the whole process more efficient and faster.

Molecular dynamics (MD) plays an important role in the study of biological systems and functions. Increasing computational power and resources available due to software/hardware improvements makes performing MD simulations at a large scale now feasible. The sizes of the systems and timescales of the processes being studied using MD are increasing day by day, and so is the importance of MD in this domain. One of the prominent categories of MD-based methods, most relevant for protein-ligand complexes, is free energy prediction approaches. Given the importance of MD simulations in scientific research, it is important to ensure that the predictions based on them are reliable and reproducible.

Methods such as ESMACS[90] and TIES[8] employ ensemble simulations in

order to capture the uncertainty associated with variations in starting conditions of MD simulations leading to accurate, precise and reproducible binding affinities. Given this sensitivity in MD, the binding pose of a ligand molecule within the binding pocket of its target protein plays an important role in the accurate prediction of its binding affinity - the key quantity of interest in early-stage drug discovery used for filtering out potential drug candidates. Therefore, incorrect binding pose could lead to false negatives or false positives excluding ideal molecules during the initial stages of the drug discovery process. This makes it quite important to ensure that the best ligand binding pose is known before performing simulations on the protein-ligand models under consideration.

IMPECCABLE[79] lists three measures of performance to evaluate the methodologies to overcome the fundamental limitation that no single approach achieves the required efficiency. The performance measures include (i) throughput or the number of ligands per unit of time; (ii) number of effective ligands sampled per unit of time; (iii) peak performance in flop/s.

IMPECCABLE brings together multiple algorithms into a single unified pipeline allowing upstream and downstream feedback to overcome the fundamental limitations of classical *in silico* drug design. According to this work[79], most ML/AI solutions have focused efforts on building effective means to analyze large volumes of data generated through either ligand docking simulations – for the filtering favorable vs. unfavorable ligand binding poses for a protein – or molecular dynamics (MD) simulations of select protein-ligand complexes[79].

In protein-ligand binding, the ligand is usually a molecule that produces a signal by binding to a site on a target protein. Docking programs are generally good at pose predictions but less effective in predicting binding free energy of protein-ligand complexes. On the other hand, MD simulations are effective at predicting binding-free energies, their intrinsic limitations in sampling protein-ligand complex formation processes imply that the approach may be computationally infeasible to translate on large compound libraries.

IMPECCABLE, depicted in Fig. 6.1, consists of an ML prediction stage (ML1) followed by three stages of data processing. S1 is high-throughput docking. S2 is Latent space representation and steered advanced sampling. S3 is the coarse-grained binding affinity. IMPECCABLE’s focus is on the use of ML methods combined with physics-based computational methods to estimate docking poses of compounds that are promising leads for a given protein target (S1) and binding free-energy computations (S3). The limitations of S1 and S3 are overcome by predicting the likelihood of binding between small molecules and a protein target (ML1), and accelerating the sampling of landscapes to bound the binding free-energy values for a given protein-ligand complex (S2).

ML models for docking score prediction (ML1) consist of scoring functions used to score poses in order to determine the most likely pose of the molecule, the magnitude of which is used to provide an indication of active versus inactive ligands, and to rank order sets of libraries. IMPECCABLE creates an ML surrogate model to replace the use of docking as a means of locating regions of chemical space likely to include strong-binding drug leads. Molecular information, in the form of 2D SMILES string, represented as 2D image depictions are used as featurization method.

In high-throughput docking (S1), the protein-ligand docking consists of a computational pipeline of ligand 3D structure enumeration, docking and scoring, and final pose scoring where the input is a protein structure with a designed binding region and a database of molecules in SMILES format to dock in.

In ML driven molecular dynamics (S2), the learning approaches learn lower-level features from the input data and aggregate them such that they can be used in a variety of supervised, semi-supervised, and unsupervised ML tasks. The learning approaches consist of variational autoencoders to automatically reduce high dimensionality of MD trajectories and cluster conformations into a small number of conformational states that share similar structure. This process is used to quantify statistical insights into the time-dependent structural changes a biomolecule undergoes in simulations,

identify events that characterize large-scale changes at multiple time scales, build low-dimensional representations of the data, and infer substates from these representations.

Binding free-energy calculations (S3) is a step where promising lead compounds are identified from initial hit generated at preceding stages. Initial hits are evaluated and potentially good compounds optimized to achieve nanomolar affinities. The change in free energy between free and bound states of protein and ligand, also known as binding affinity, is a measure of binding potency of a molecule making it a good parameter for evaluating and optimizing hits.

The implication of this work is that by combining ML approaches with physics-based models such as docking and molecular dynamics simulations, a three-order of magnitude improvement is achieved in improving the size of compound libraries that can screen with traditional approaches while simultaneously providing access to binding free energy calculations that can impose better confidence intervals in the ligands selected for further optimization.

6.1.2 Ligand Pose Optimizer Model (LPOM)

The latest iteration of the IMPECCABLE framework improves the original by using generative AI, ML-based docking surrogate models, and ligand pose optimization model to enhance binding affinity predictions, streamlining the identification of high-affinity compounds for drug discovery. In this framework, the model we construct and enhance is used to evaluate the quality of given docking poses as starting points for further MD simulations. The Ligand Pose Optimizer Model (LPOM) is the machine learning model trained to predict the MM-PBSA energy of the relaxed state for each individual docking pose used in the training space.

The LPOM is used to predict the relaxed Molecular Mechanics Poisson-Boltzmann Surface Area (MM-PBSA)[84] of docking poses that are generated for each ligand. The candidate poses of each ligand are then ranked based on the predicted MM-PBSA value, allowing for the selection of a high probability subset of novel poses used as starting points for relaxation through refined MD simulations. This provides an efficient and

effective method to reduce the number of MD simulations to be run for a given ligand, substantially diminishing the aggregate number of simulations required for the overall workflow.

The LPOM consists of features such as a novel set of descriptors that incorporate information about the protein-ligand interaction, creating a protein-ligand atom-pair interaction profile. These features capture the interaction space between each protein and ligand atom pair through interatomic spacing since it affects the stability and final energy of the pose.

Although molecular docking is faster and more efficient, results are not as accurate or as reliable as MD simulations, frequently yielding poses with close contacts between interacting atoms in the protein-ligand complex, resulting in large forces and overall pose instability. Consequently, the standard approach of simply training to predict the interaction energies of the docking poses, as commonly used [46, 51, 28], will not be suitable for the LPOM approach. To resolve this for training the LPOM, IMPECCABLE runs relaxation on the training poses to allow each pose to further optimize within the protein pocket. This results in a more realistic pose and the corresponding energy, from which we subsequently compute the MM-PBSA energy as a measure of the binding-free energy.

The relaxed pose energies, together with the interaction descriptors, were subsequently used as training data for the LPOM to predict the relaxed protein-ligand pose energy. LPOM is the machine learning model trained to predict the MM-PBSA energy of the relaxed state for each individual docking pose used in the training space. The ATOM Machine Learning Pipeline (AMPL) [55], an end-to-end machine learning pipeline used for molecular property prediction, is used to create the LPOM used in the IMPECCABLE workflow.

The next sections detail the various descriptors that are used as features to train LPOM. Descriptors are methods to characterize molecular features and can encode information about structures or fingerprints and chemical properties.

6.1.3 Simplified Lennard-Jones Potential Descriptors

Our initial descriptors were a subset of descriptors from the TinyIFD[39]. In order to keep computational overhead minimal, contact-based descriptors were used with a simplified version of the Lennard-Jones potential. We considered each contact between the protein atoms H, C, N, O, S and the ligand atoms H, C, N, O, S, halogen atoms totaling to 30 descriptors. For each type of protein-ligand element pair, we extracted the distance between each set of protein and ligand atoms. The descriptor for that atom pair is then computed by summing over a distance-based function:

$$C(E_1, E_2) = \sum_{j \in E_1}^{ligand} \sum_{k \in E_2}^{protein} f(d_{jk}) \quad (6.1)$$

$$f(d_{jk}) = \left(\frac{1}{d_{jk}}\right)^{12} - \left(\frac{1}{d_{jk}}\right)^6 \quad (6.2)$$

Where C is the contact score for the pair of elements E_1 and E_2 . j is the ligand atoms of type E_1 , k is the protein atoms of type E_2 , and d_{jk} is the distance between j and k .

The descriptor set is then a vector v where each entry v_{ij} corresponds to the contact score $C(E_1^i, E_2^j)$ and E_1^i and E_2^j are elements from protein and ligand atoms respectively.

6.1.4 Geometric Descriptors

With the realization that significant information about the protein-ligand interaction was lost in the summation process for each protein-ligand element pair type, a new expanded descriptor was defined to preserve details about the interaction. Similarly to the simplified Lennard-Jones potential descriptors, we considered each contact between the protein atoms H, C, N, O, S and the ligand atoms H, C, N, O, S. However, instead of performing a summation involving each contribution, a histogram was created for each element pair type to provide a distance profile in lieu of a total sum. Using a range from zero Å to 12 Å, 24 intervals (buckets) of 0.5 Å were defined. The distance was subsequently computed for each protein-ligand interacting atom pair. With

the corresponding bucket being incremented, a protein-ligand interaction profile was created for each pose. For distances greater than 12 Å, the contribution was ignored. The resulting size of the protein-ligand interaction space increased to 600 features with 25 total combinations of elements and 24 buckets per combination.

This differs from the simplified Lennard-Jones potential descriptor approach in that information about the interactions for all magnitudes is preserved for each protein-ligand interacting atom pair. The simplified Lennard-Jones potential descriptors, using a sum over all of the atom pairs, can become dominated by atom pairs in close proximity to one another. In addition, the summation approach does not provide differentiation on the magnitude of each contribution to the sum. This expanded protein-ligand interaction descriptor provides the model key information regarding the distribution of the atom pairs on a more detailed level.

6.1.5 Molecular Descriptors

Both descriptors above, simplified Lennard-Jones potential and geometric descriptors, are classified as structure-based descriptors that encode knowledge about the structure of the target. On the other hand, molecular descriptors generated with ECFP (Extended Connectivity Fingerprints), RDKit, MOE (Molecular Operating Environment), and Mordred are ligand-based descriptors that encode chemical properties and fingerprints.

Molecular descriptors are defined as "final result of a logical and mathematical procedure, which transforms chemical information encoded within a symbolic representation of a molecule into a useful number or the result of some standardized experiment" by Todeschini et al. [86]. These types of descriptors aim to provide information about a given molecule with varying complexity ranging from molecular weight to quantum mechanical properties. These descriptors are often applied in quantitative structure-activity relationship (QSAR) models as they have been found to be effective in predictive models. Different molecular descriptor sets tend to cover different attributes of molecules.

The **limitation** of molecular descriptors is that they describe only the ligand itself based on the associated input SMILES string. These descriptors do not take into account the protein, the relative orientation of the ligand, and the interaction space of the pose itself. In the context of the IMPECCABLE framework, we are interested in selecting the best poses for each ligand when interacting with a given protein. Although molecular descriptors can inform the expected performance of a specific ligand for a given or average orientation in the pocket of a protein, they do not provide information on the relative quality across multiple poses.

6.1.6 Description of training data

This section describes the source of the data and the preprocessing steps performed using various parts of the IMPECCABLE pipeline before it was used to train LPOM. In this study, 3C-like protease (3CLpro) of SARS-CoV-2 was used. The initial model for the training dataset is based on 10,000 compounds identified through a surrogate docking model[6, 19]. The compounds were first processed with FixpKa[65] 2.1.3.0 to obtain the correct protonation states at pH 7.4. Subsequently, up to 200 conformers per compound were generated using OMEGA[38] 4.1.2.0. The prepared structural library of the compounds was then docked to the protein (PDB ID: 6W63) using FRED[50] 4.1.1.0. From these, the top 100 docking poses with the best Chemgauss4[50] docking scores were selected, and each protein-compound complex was minimised for 200 steps with NAMD[70] package using a sophisticated conjugate gradient and line search algorithm. Finally, the binding free energy for each minimised structure was calculated using the MM-PBSA (molecular mechanics Poisson-Boltzmann surface area) approach using AmberTools[14].

The compounds for the separate validation dataset were selected from a previous study, where a generative active learning (GAL) approach was employed to produce high-quality small molecules[48]. This approach combined REINVENT[47], a generative AI, with ESMACS[90], a physics-based method for accurate ranking of binding free energies. In addition, these binding free energies were used iteratively to refine

the compounds generated by REINVENT. To validate the current LPOM model, approximately 3,000 compounds were chosen from the first three GAL iterations, each iteration generating a batch of 1,000 compounds.

6.2 Improving Ligand Pose Optimization Model

Given the novelty of predicting the MM-PBSA energies for *relaxed* protein-ligand interaction geometries from an initial state pose, a traditional comparison to state-of-the-art prediction approaches for relaxed protein-ligand interaction geometries was not possible. The primary goal of this effort was to create the best performing model while minimizing computational time, thus the Ligand Pose Optimizer Model (LPOM) development progressed in phases. The first phase focused on maximizing performance using a minimal descriptor protein-ligand interaction descriptor with minimum computing requirements to differentiate among poses. The next phase focused on incorporating additional information about protein-ligand interaction to further differentiate among poses. The combination of these two phases and the descriptors that are generated to train the model will be called Molecular Pose (MP) descriptors in the remainder of the manuscript. The baseline results for the LPOM training with these descriptors are in [Table 6.1](#)

And finally, the third phase consisted of combining the descriptors in the previous phases with molecular descriptors to enhance the information available to the model to learn and better predict the best poses. Some of the molecular feature generators that compute these features using the molecular descriptors in the form of SMILES strings include Mordred and RDKit. Any result from LPOM training with molecular descriptors as features will include the name of the generator used. This includes baseline results for LPOM trained using only the molecular descriptors. Although the limitations of this technique were discussed previously, it is important to measure the standalone performance of these features.

Results from Model Training (R^2) - MP Only	
Metric	MP
Train	0.65
Valid	0.63
Test	0.64

Table 6.1: Ligand pose optimizer model training results using Molecular pose descriptors only.

6.2.1 Hyperparameter optimization (HPO) and its implications

Initially, our goal was to improve the performance of the Ligand pose estimation model (LPOM) using the coefficient of determination (R^2) as a metric. The primary area of interest was the use of hyperparameter optimization (HPO) to find the set of hyperparameters for LPOM. Why was HPO identified as an area to explore as a potential avenue to improve the existing model? Previously, the work involving the creation of the model in the first two phases described above led to the following results in Table 6.1. The hyperparameters were mostly set using heuristics rather than employing any form of search. This meant that there was a possibility that the most optimal hyperparameters were not set; therefore, HPO could lead to an improvement in model performance.

The hyperparameter optimization work performed is on the combination of model and hyperparameters that produces acceptable results, but it's unclear whether LPOM had the best set or if another set of hyperparameters will improve its R^2 score. HPO during training will help evaluate if the best hyperparameters were chosen or find a new set of parameters that improve model performance.

The first example of a hyperparameter that we found where the default was not the best option in terms of training the model is batch size. We found that as batch size increased, the time it took to train the model decreased. Fig. 6.2 shows that the reduction in training time plateaus around the batch size of 1000. In addition, increasing the batch size from the default value used in AMPL[55] to 1000 only led to a slight drop in error as the data splits. This can be seen in Fig. 6.3. Therefore,

changing the default batch size to 1000 brought benefits in terms of reduced training times with minimal reduction in the model's R^2 .

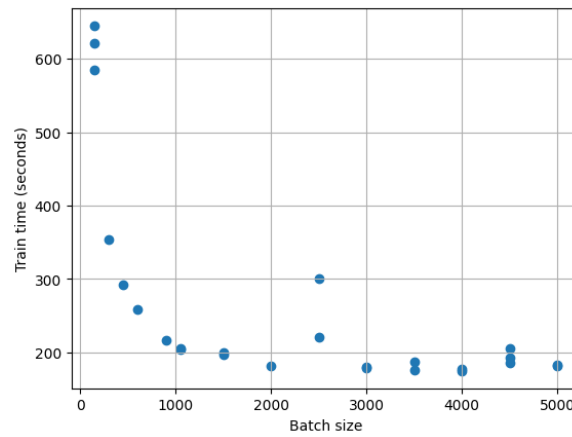


Figure 6.2: Figure shows that as batch size increases training time decreases

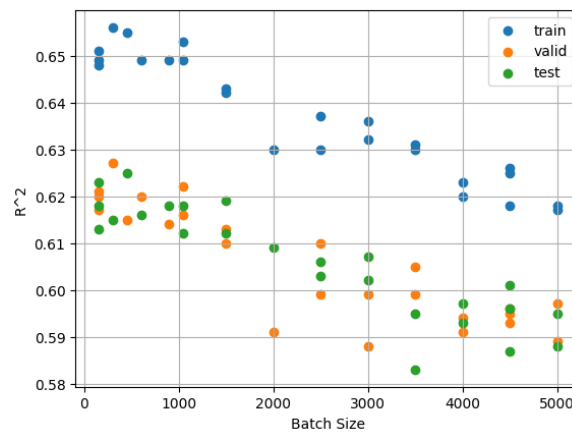


Figure 6.3: Figure shows that as batch size increases R^2 decreases

The next set of hyperparameters that we tested are weight decay and weight decay penalty. These are regularization techniques that make a model simpler. The goal of making a model simpler is to help it generalize better to test data. These two parameters were initially tested during a hackathon by a team I mentored. They used

AMPL’s baseline neural network model with the default dataset provided by AMPL and found that the two hyperparameters did not provide major benefits or drawbacks. We performed some tests using LPOM and our training data and also concluded that weight decay and weight decay penalty parameters do not seem to have any noticeable effect on LPOMs test R^2 .

Revisiting the established goal to increase the LPOM’s R^2 , we wanted to test the effect parameters such as the number of hidden layers, layer sizes, and dropout. Fig. 6.2 shows a table showing various combinations of layers, layer sizes, and dropout values and their effect on test R^2 . The results show that increasing the layer sizes and adjusting the number of layers by adding an extra layer had a negative impact on the results.

R^2	Layer Sizes	Dropouts
0.5696	[200, 100]	[0.4, 0.4]
0.5657	[100, 50]	[0.4, 0.4]
0.5645	[50, 25]	[0.4, 0.4]
0.5551	[200, 50]	[0.5, 0.5]
0.5399	[200, 50]	[0.6, 0.6]
0.5357	[200, 100]	[0.6, 0.6]
0.5398	[200, 100, 50]	[0.5, 0.5, 0.5]
0.5085	[400, 200, 100]	[0.5, 0.5, 0.5]

Table 6.2: Effect of layer sizes and dropouts on test R^2

Implications of HPO: Overall, the results of HPO were mixed. We initially discovered that batch size would make training faster without a trade-off in model performance. But the remaining HPO results either yielded results close to the original parameters or made the model perform worse.

6.2.2 Combining geometric and molecular descriptors

Since the Ligand Pose Optimization model’s performance did not improve with HPO we needed to find other alternatives. The results of the HPO, Fig. 6.2, showed that the number of features of the model was unlikely to be the constraint in terms of

performance. That meant that adding more features could potentially benefit LPOM. Therefore, the next phase of the study explored the effects of combining molecular descriptors with existing geometric descriptors on the performance of the model.

Firstly, we needed to establish a baseline for the molecular descriptors we plan to add. To do this, we replaced the geometric features, such as Simplified Lennard-Jones Potential descriptors and geometric descriptors, with molecular descriptor features. The first two columns of Table 6.3 show the results of using only the descriptors generated through Mordred and RDKit. We compare those results with the original model results found in the MP column of Table 6.3. This process was more challenging than initially anticipated because of the way AMPL handles molecular descriptors. It required a multistep process to generate the molecular descriptors and then retrain LPOM with the descriptor file saved in the previous step.

Then we combine the geometric features of the original training data with the molecular features and the results are found in the columns MP + Mordred and MP + RDKit in Table 6.3. **The results showed that the combination of geometric and molecular features produced the best performing model when evaluated using the coefficient of determination or R^2 .**

Although we use R^2 to evaluate the performance of the model in this study, the coefficient of determination is not the best way to measure the impact this model has on the IMPECCABLE framework. With a perfect model, where $R^2 = 1.0$, we would only need to suggest one pose for each of the n ligands. However, in the real world, we can realistically only recommend some number (m) of poses that ideally contain the best pose for each of the n ligands. With this in mind, we can reframe our model as a classification model.

The primary objective of the LPOM is to minimize the number of m poses that must be recommended, based on the predicted MM-PBSA values for each ligand, to recommend at least one of the top p poses based on the true MMPBSA values. This makes LPOM a surrogate model for molecular dynamics (MD) simulations that are typically performed through

Results from Model Training (R^2)					
	Mordred	RDKit	MP	MP + Mordred	MP + RDKit
Train	0.49	0.47	0.65	0.77	0.77
Valid	0.48	0.47	0.63	0.75	0.73
Test	0.48	0.48	0.64	0.75	0.75

Table 6.3: Ligand pose optimizer model training results based on descriptor set used. MP: Molecular Pose descriptors (Simplified Lennard-Jones + geometrics descriptors)

IMPECCABLE on each of the poses in the ligand-protein binding pocket. Although LPOM does not completely eliminate the need for MD simulations, it greatly reduces the number of poses that MD simulations must run on to confirm the best pose for each ligand-protein complex.

6.2.3 Testing separate validation compounds

In order to test the robustness of the LPOM, we wanted to test how the model trained on 3CLpro, the data used thus far, performs on a new set of chemical compounds. The IMPECCABLE team generated approximately 3,000 compounds in three different batches. The results of the validation on these new compounds with models trained on 3CLpro data are shown in Table 6.4. The most likely cause of this decrease in performance when testing on new compounds is due to a chemical shift between the 3CLpro data and the new compounds. This means the validation data is not the ideal set to test the robustness of LPOM. So for now the best validation of this model comes from the test split created during model training.

6.2.4 Data reduction

Another aspect of this study includes performing experiments to test the minimum amount of data required in order for the model to still perform well. Successfully reducing training data size will have not only computational benefits, but also the ability to integrate new data sources to perform weak scaling.

Validation on new compounds	
	R^2
Mordred	0.48
RDKit	0.48
MP	0.48
MP + Mordred	0.48
MP + RDKit	0.48

Table 6.4: Validation results show that the model’s performance was much lower than on the validation split of the training data

Table 6.5: Train, Validation, and Test R^2 values for different data splits. Data split percent indicates % for Valid and Test

Data Split (%)	Train	Valid	Test
15	0.65	0.63	0.63
20	0.65	0.61	0.61
30	0.64	0.60	0.60
40	0.62	0.57	0.57
45	0.61	0.54	0.54
47	0.58	0.52	0.52
49	0.57	0.47	0.48
49.5	0.55	0.46	0.46

Data reduction was performed by increasing iteratively the percentage of data used for validation and testing. The percentages for both validation and testing remained the same for consistency. For example, setting aside 15% for validation and 15% for testing means that the remaining 70% is used to train the model. The first column of Table 6.5 shows the percentage values assigned for the validation and test sets.

As larger percentages of data were allocated for validation and test sets, we expected R^2 to decrease. However, it was unclear to what extent the impact of reducing the percentage of data decreased. Since we start with a fixed-sized dataset, the percent of data is a proxy for the total amount of data allocated to train, test, and validation sets. Therefore, iteratively reducing the size of the training dataset, by adjusting the

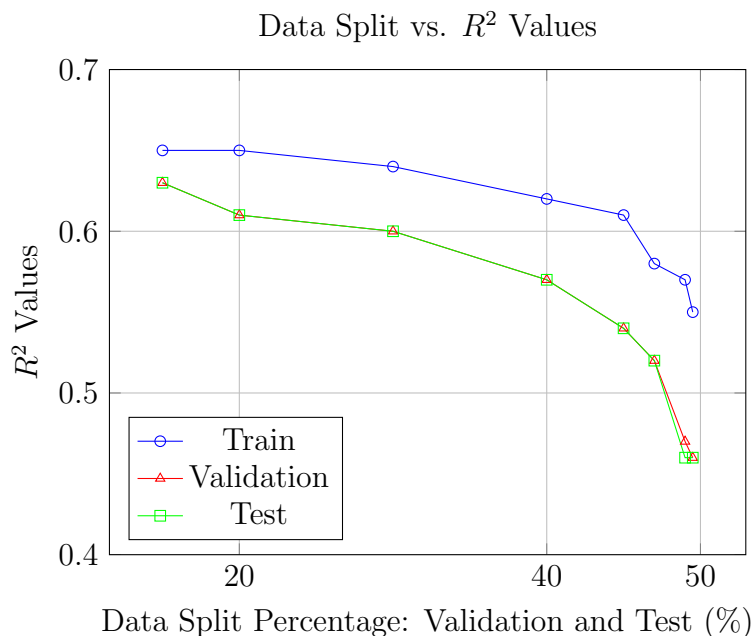


Figure 6.4: Plot of results showing the effect of reducing training data size on R^2 of Train, Validation, and Test

validation and test percentages, should indicate the effect that reducing the training data would have on the model performance.

Fig. 6.4 and Table 6.5 show the results of iteratively reducing the training data for LPOM by increasing the percentage of data allocated to both validation and test sets. Although R^2 was expected to decrease as data allocated to training is reduced, the model performance held up remarkably well even as most of the data was reallocated from training to validation and test splits. An interesting trend depicted in Fig. 6.4 is that even though R^2 is declining for training, test, and validation sets, the decrease in performance of the validation and test sets accelerates as more data is allocated away from training set. Even with that decrease in performance, the model seems to hold up well even as the percentage of training data is reduced to less than 10% of the total available data. The training set was reduced to 1% of the total available while 99% was allocated to the validation and testing sets and the result shows how quickly the model reaches a baseline as its performance increases with increasing training data.

6.2.5 A discussion on parallel HPO scaling

Initially, we expected parallel hyperparameter optimization to drive the improvements to the LPOM. We expected each of the various hyperparameters to have a greater impact on the model performance and its variability depending on the parameters values. Since we expected to have a large combination of hyperparameters it seemed important to perform hyperparameter optimization using parallel methods. This includes work done to train the same model on multiple GPUs concurrently, each with a different set of hyperparameters. Although we were able to establish the ability to perform parallel HPO, the original goal became much less impactful when the HPO results did not produce the ideal outcomes in terms of model performance.

Even though it is possible to run every possible combination of the possible hyperparameters optimization, we must take a more measure approach before allocate compute resources at the Exascale level. When an individual hyperparameter does not have any affect on the model across a wide range of values it does not make sense to test combinations of those hyperparameters with combinations of every other hyperparameter. The scale of the problem grows exponentially and is very computationally intensive. The main flaw of this approach is that this method can waste critical compute resources even when running on systems such as the Frontier Exascale supercomputer[64].

When pursued a more measured approach with HPO where we found some parameters like batch size that optimized our training without sacrificing model performance. We also had some hyperparameters such as weight decay and weight decay penalty which had negligible impact on the model’s ability to generalize. On the other hand we also saw hyperparameters and combinations of hyperparameters such as the number of layers, layer sizes, and dropout that negatively impacted our results. The suboptimal outcomes from this experiment were still fruitful because this showed that the model’s features were probably not the limiting factor for its performance. We tried to increase the capacity of the model through this experiment to see if the number of features exceeded the capacity of the model. When we got the opposite result, it

informed us that there could be room for improvement in model performance by finding additional features the model can rely on for greater predictive capabilities. This directly led to the next experiment, in which we supplemented the existing geometric features with molecular features, which ultimately led to the improved performance of LPOM. Subsequently, the focus of the study on improving model performance shifted more towards features comprising the model than toward HPO search and optimization.

Therefore HPO still benefited this study not directly but more by informing us about the directions we needed to pursue to improve the model.

6.3 Lessons Learned

Surrogate models are increasingly becoming important and central in many scientific applications such as PIconGPU and IMPECCABLE. In many cases, surrogate models replace expensive, time-consuming, and computationally expensive simulations. The previous chapter discussed the importance of scaling surrogate models to extract insights from plasma-physics simulations by reducing the costly analyses that must be done on the data produced by such simulations.

In this chapter, we focus on another surrogate model in a different domain. In this case study, the surrogate model is used to reduce the need for computationally expensive MD simulations in a drug discovery framework. This is aimed at improving traditional drug discovery pipelines that are costly and time-consuming. The main contribution of this chapter is to improve the predictive performance of the Ligand Pose Optimization model. As explained in the beginning of this chapter, the model contributes to the IMPECCABLE drug discovery framework by reducing the expensive MD simulations that would normally be required for each of the poses in the ligand-protein complexes from molecular docking simulations to better select lead compounds for later stages of drug discovery.

Chapter 7

CONCLUSION

The landscape of ML applications, problems, model sizes (capacity), software, and hardware accelerators has evolved over the last few years. Much of that progress occurred simultaneously with the research discussed in this thesis. Some questions that were posed at the beginning of this thesis remain. That has more to do with the nature of hardware and software co-design than anything else. As hardware evolves, the software necessary to run ML workflows tries to catch up. This was the case when I started my Ph.D. and will remain that way for the foreseeable future. Thus, some of the questions posed at the beginning of this thesis will remain because they are less questions to be answered or problems to be solved and are more guidelines to help focus our attention on an ongoing race between hardware and software. Where portability is the goal of reducing the distance between hardware and software evolution; ideally by not slowing down hardware advancements.

As stated in the introduction, this thesis focuses on designing and building scalable and portable machine learning-based frameworks to new hardware architectures with a focus on cancer drug discovery and plasma physics simulations as case studies.

7.1 Research Outcomes and Key Takeaways

At the beginning of this thesis, some research questions were posed to formulate the motivation and bring to light some of the research gaps that this work addresses. We revisit these research questions to highlight the contributions made in this thesis.

1) How do we apply model enhancements and performance optimizations to design drug discovery pipelines for shorter development timelines?

When it comes to the drug response problems that are benchmarked in CANDLE Pilot 1, enhancements for those drug response models came in the form of establishing tree-based models as an improvement over CNNs. This was after extensive work done looking into how cell line-based models translate to tissue data sets. This goal seemed unviable after establishing the lack of truth data in the form of drug response data for tissue cells. In addition, the development of scaling ML workflows on systems like Frontier will be the basis of increasing drug discovery model throughput making them more capable.

In addition, the improvements made to the Ligand Pose Optimization model (LPOM) will increase the throughput of the promising lead compounds filtered to advance them to later stages of drug discovery process. This model will be integrated into the larger IMPECCABLE 2.0 workflow to enable this advancement. The use of ML techniques in drug discovery process is relatively new and the regulatory framework is still in the process of catching up to the use these computational techniques. The guidance to support the use of AI and ML techniques to evaluate and improve the decision making process for drugs is in its early stages[32]. In the future, models such as LPOM and regulatory frameworks[32] could shape each other as some of the techniques become standardized and encoded in the guidance while the guidance sets the safety parameters and boundaries for the models to maintain.

2) How do other ML model architectures such as tree-based models compare to CNNs?

The research established in this thesis shows that, for classes of problems that rely on structured datasets, tree-based models often outperform neural networks like CNNs. The thesis presents two datasets, CCLE and NCI60, where this conclusion holds for drug response problems. The thesis also includes model performance metrics that show that XGBoost not only outperforms in terms of accuracy but also trains faster on GPUs. The conclusion being if both models perform similarly in terms of accuracy it makes sense to go with XGBoost for the hardware acceleration they enjoy over CNNs when the training data is held constant.

3) How do we improve the accessibility of these models for domain experts giving them the ability to analyze and compare multiple types of models simultaneously?

Domain scientists prefer to have such software utilities to make the process of using their proprietary datasets for training, analyzing and more streamlined. Doing this manually is possible but time consuming and adds friction to the modeling work domain scientists and other practitioners need to do in order to make such analyses possible without UNNT. To make it easier for other researchers to conduct studies and answer questions similar to those answered by the previous question a novel utility is designed and created to train, compare, and analyze two model architectures. By doing this, the process of testing similar hypotheses for model performance will be much more seamless.

The key takeaway is that a software is necessary as a starting point when practitioners have domain data that is structured and want to extract insights in the data. UNNT is an effort to streamline this process to perform initial tests to test which type of model is most suitable for the datasets users have. This will inform the user about which direction to take depending on the results seen with UNNT.

4) How well do ML models scale on new hardware architectures such as AMD GPUs? And what challenges will this bring?

The PIConGPU case study is used for designing and developing processes to scale ML models on new hardware architectures such as Frontier. As far as we knew at the time of this work, we were some of the first to try this on not only Frontier but also on AMD GPUs in general. The PIConGPU's ML training scaled up to 400 AMD MI250X GPUs and the future results will only improve from that established benchmark. PIConGPU is a plasma-physics code with an HPC component that also scales well on Frontier. The coupling between the simulation, data streaming and distributed ML training was the first time such an in-transit learning workflow was demonstrated at scale. My contribution to scaling the ML training was integral to the larger in-transit ML workflow. To the best of my knowledge, we were the first to

demonstrate such a workflow at the exascale level. This ultimately proved the viability of running distributed ML training on AMD GPUs like the MI250X.

This work opened the door to further work that can be done with scheduling strategies in HPC workloads. The combination of scientific simulations and ML training and the data streamed between them brings new challenges and the work was the first step in establishing the possibility of learning scientific insights using such orchestrations. Other domains or applications may need different scheduling methods depending on the compute and data needs specific to them. This work also informs on the software upgrades that need to be made to the AMD software stack to enable ML training and compatibility of algorithms that exist on platforms with NVIDIA GPUs. This was further validated by prominent technical analysts who concluded that the AMD ROCm software has much further to go before it can provide the features supported by NVIDIA out of the box. This is mainly future work for AMD and the open source community developing the ROCm software stack. It will be a challenge to keep up with the advances NVIDIA is making with CUDA software stack, but we believe these advancements are still necessary for advancement of HPC software and hardware design.

5) What insights or research questions will hyperparameter optimization (HPO) and scaling of molecular dynamics ML models lead to?

PIConGPU’s ML scaling and the experience gained were invaluable as we scaled cancer drug discovery ML workflows that synthesize docking for pose estimation and molecular dynamics for protein-ligand docking.

Performing hyperparameter optimization (HPO) on the surrogate model led to insights that helped guide future experiments, which ultimately yielded an improved model with greater predictive capabilities for selecting the top poses of the ligand-protein docking complexes. The improved model further reduces the need for computationally expensive MD simulations that were previously necessary in the absence of the surrogate model. The implication of this work is that improved model accuracy should lead to higher throughput in terms of lead molecules evaluated to advance to

later stages of the drug discovery pipeline.

7.2 Thesis Contributions

The research questions were the motivation for this thesis, and the following are the outcomes and contributions of this work.

1) Established that gradient boosted decision trees (GBDTs) outperform CNNs in predicting drug responses, with potential applicability to other structured datasets. Designed and created a software utility that researchers and domain scientists can use on their custom structured datasets to compare and analyze CNNs and XGBoost models.

2) Proposed and developed novel approaches to address the scalability of ML training on large-scale supercomputers such as Frontier equipped with the state-of-the-art AMD GPUs

3) Improved scalability and performance of surrogate models that reduce the computational cost of scientific applications in domains such as plasma physics and drug discovery.

BIBLIOGRAPHY

- [1] Crosscut report: Exascale requirements reviews. Technical report, The Office of Science, 2017.
- [2] Early application results on pre-exascale architecture with analysis of performance challenges and projections. Technical report, The Office of Science, 2020.
- [3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016.
- [4] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1), 2021.
- [5] Abdelkader Behdenna, Julien Haziza, Chloé-Agathe Azencott, and Akpéli Nordor. pycombat, a python tool for batch effects correction in high-throughput molecular data using empirical bayes methods. 03 2020.
- [6] Agastya P. Bhati, Shunzhou Wan, Dario Alfè, Austin R. Clyde, Mathis Bode, Li Tan, Mikhail Titov, Andre Merzky, Matteo Turilli, Shantenu Jha, Roger R. Highfield, Walter Rocchia, Nicola Scafuri, Sauro Succi, Dieter Kranzlmüller, Gerald Mathias, David Wifling, Yann Donon, Alberto Di Meglio, Sofia Vallecorsa, Heng Ma, Anda Trifan, Arvind Ramanathan, Tom Brettin, Alexander Partin, Fangfang Xia, Xiaotan Duan, Rick Stevens, and Peter V. Coveney. Pandemic drugs at pandemic speed: infrastructure for accelerating covid-19 drug discovery with hybrid machine learning- and physics-based simulations on high-performance computers. *Interface Focus*, 11(6):20210018, 2021.

- [7] Tanmoy Bhattacharya, Thomas Brettin, James H. Doroshov, Yvonne A. Evrard, Emily J. Greenspan, Amy L. Gryshuk, Thuc T. Hoang, Carolyn B. Veal-Lauzon, Dwight Nissley, Lynne Penberthy, Eric Stahlberg, Rick Stevens, Fred Streitz, Georgia Tourassi, Fangfang Xia, and George Zaki. Ai meets exascale computing: Advancing cancer research with large-scale high performance computing. *Frontiers in Oncology*, 9, 2019.
- [8] Mateusz K. Bieniek, Agastya P. Bhati, Shunzhou Wan, and Peter V. Coveney. Ties 20: Relative binding free energy with a flexible superimposition algorithm and partial ring morphing. *Journal of Chemical Theory and Computation*, 17(2):1250–1265, 2021. PMID: 33486956.
- [9] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2022.
- [10] David Brayford, Sofia Vallecorsa, Atanas Atanasov, Fabio Baruffa, and Walter Riviera. Deploying ai frameworks on secure hpc systems with containers. *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep 2019.
- [11] Broad Institute. Depmap portal, 2023. Accessed: 2023-05-25.
- [12] Heiko Burau, Renée Widera, Wolfgang Hönig, Guido Juckeland, Alexander Debus, Thomas Kluge, Ulrich Schramm, Tomas E. Cowan, Roland Sauerbrey, and Michael Bussmann. PIconGPU: A fully relativistic particle-in-cell code for a GPU cluster. *IEEE Transactions on Plasma Science*, 38(10 PART 2):2831–2839, 2010.
- [13] M. Bussmann, H. Burau, T. E. Cowan, A. Debus, A. Huebl, G. Juckeland, T. Kluge, W. E. Nagel, R. Pausch, F. Schmitt, U. Schramm, J. Schuchart, and R. Widera. Radiative signatures of the relativistic Kelvin-Helmholtz instability. In *SC '13 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 5–1 – 5–12, 2013.
- [14] David A. Case, Hasan Metin Aktulga, Kellon Belfon, David S. Cerutti, G. Andrés Cisneros, Vinícius Wilian D. Cruzeiro, Negin Forouzesh, Timothy J. Giese, Andreas W. Götz, Holger Gohlke, Saeed Izadi, Koushik Kasavajhala, Mehmet C. Kaymak, Edward King, Tom Kurtzman, Tai-Sung Lee, Pengfei Li, Jian Liu, Tyler Luchko, Ray Luo, Madushanka Manathunga, Matias R. Machado, Hai Minh Nguyen, Kurt A. O’Hearn, Alexey V. Onufriev, Feng Pan, Sergio Pantano, Ruxi Qi, Ali Rahnamoun, Ali Rishch, Stephan Schott-Verdugo, Akhil Shajjan, Jason Swails, Junmei Wang, Haixin Wei, Xiongwu Wu, Yongxian Wu, Shi Zhang, Shiji Zhao, Qiang Zhu, Thomas E. III Cheatham, Daniel R. Roe, Adrian Roitberg, Carlos Simmerling, Darrin M. York, Maria C. Nagan, and Kenneth M. Jr. Merz. AmberTools. *Journal of Chemical Information and Modeling*, 63(20):6183–6191, 2023.

- [15] KODE CHEMOINFORMATICS. Dragon (software for molecular descriptor calculation), 2017.
- [16] 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, pages 785–794, New York, NY, USA, 2016. ACM.
- [17] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Tvm: An automated end-to-end optimizing compiler for deep learning, 2018.
- [18] Davide Chicco, Matthijs J Warrens, and Giuseppe Jurman. The coefficient of determination r-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. *PeerJ Comput Sci*, 7:e623, July 2021.
- [19] Austin Clyde, Xuefeng Liu, Thomas Brettin, Hyunseung Yoo, Alexander Partin, Yadu Babuji, Ben Blaiszik, Jamaludin Mohd-Yusof, Andre Merzky, Matteo Turilli, Shantenu Jha, Arvind Ramanathan, and Rick Stevens. Ai-accelerated protein-ligand docking for sars-cov-2 is 100-fold faster with no significant change in detection. *Scientific Reports*, 13(1):2105, Feb 2023.
- [20] Ronan Collobert, Koray Kavukcuoglu, and Clement Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.
- [21] The GTEx Consortium, Kristin G. Ardlie, David S. Deluca, Ayellet V. Segrè, Timothy J. Sullivan, Taylor R. Young, Ellen T. Gelfand, Casandra A. Trowbridge, Julian B. Maller, Taru Tukiainen, Monkol Lek, Lucas D. Ward, Pouya Kheradpour, Benjamin Iriarte, Yan Meng, Cameron D. Palmer, Tõnu Esko, Wendy Winckler, Joel N. Hirschhorn, Manolis Kellis, Daniel G. MacArthur, Gad Getz, Andrey A. Shabalina, Gen Li, Yi-Hui Zhou, Andrew B. Nobel, Ivan Rusyn, Fred A. Wright, Tuuli Lappalainen, Pedro G. Ferreira, Halit Ongen, Manuel A. Rivas, Alexis Battle, Sara Mostafavi, Jean Monlong, Michael Sammeth, Marta Mele, Ferran Reverter, Jakob M. Goldmann, Daphne Koller, Roderic Guigó, Mark I. McCarthy, Emmanouil T. Dermitzakis, Eric R. Gamazon, Hae Kyung Im, Anuar Konkashbaev, Dan L. Nicolae, Nancy J. Cox, Timothée Flutre, Xiaoquan Wen, Matthew Stephens, Jonathan K. Pritchard, Zhidong Tu, Bin Zhang, Tao Huang, Quan Long, Luan Lin, Jialiang Yang, Jun Zhu, Jun Liu, Amanda Brown, Bernadette Mestichelli, Deneé Tidwell, Edmund Lo, Mike Salvatore, Saboor Shad, Jeffrey A. Thomas, John T. Lonsdale, Michael T. Moser, Bryan M. Gillard, Ellen Karasik, Kimberly Ramsey, Christopher Choi, Barbara A. Foster, John Syron, Johnell Fleming, Harold Magazine, Rick Hasz, Gary D. Walters, Jason P. Bridge, Mark Miklos, Susan Sullivan, Laura K. Barker, Heather M. Traino, Maghboeba Mosavel, Laura A. Siminoff, Dana R. Valley, Daniel C. Rohrer, Scott D. Jewell, Philip A.

- Branton, Leslie H. Sobin, Mary Barcus, Liqun Qi, Jeffrey McLean, Pushpa Har-
 iharan, Ki Sung Um, Shenpei Wu, David Tabor, Charles Shive, Anna M. Smith,
 Stephen A. Buia, Anita H. Undale, Karna L. Robinson, Nancy Roche, Kimberly M.
 Valentino, Angela Britton, Robin Burges, Debra Bradbury, Kenneth W. Ham-
 bright, John Seleski, Greg E. Korzeniewski, Kenyon Erickson, Yvonne Marcus,
 Jorge Tejada, Mehran Taherian, Chunrong Lu, Margaret Basile, Deborah C. Mash,
 Simona Volpi, Jeffery P. Struewing, Gary F. Temple, Joy Boyer, Deborah Colan-
 tuoni, Roger Little, Susan Koester, Latarsha J. Carithers, Helen M. Moore, Ping
 Guan, Carolyn Compton, Sherilyn J. Sawyer, Joanne P. Demchok, Jimmie B.
 Vaught, Chana A. Rabiner, Nicole C. Lockhart, Kristin G. Ardlie, Gad Getz,
 Fred A. Wright, Manolis Kellis, Simona Volpi, and Emmanouil T. Dermitzakis.
 The genotype-tissue expression (gtex) pilot analysis: Multitissue gene regulation
 in humans. *Science*, 348(6235):648–660, 2015.
- [22] PyTorch Contributors. Pytorch distributed data parallel documentation. <https://pytorch.org/docs/2.0/notes/ddp.html>, 2024. Accessed: 2024-09-21.
- [23] Sajal Dash, Isaac Lyngaas, Junqi Yin, Xiao Wang, Romain Egele, Guojing Cong,
 Feiyi Wang, and Prasanna Balaprakash. Optimizing distributed training on fron-
 tier for large language models, 2023.
- [24] Keras Developers. Keras: Api built on top of tensorflow. <https://keras.io/>.
 Accessed: 2020-05-07.
- [25] ONNX Developers. Open Neural Network eXchange. <https://onnx.ai>, 2019.
- [26] ONNX Runtime Developers. Onnx runtime. <https://www.onnxruntime.ai>.
- [27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-
 training of deep bidirectional transformers for language understanding, 2019.
- [28] Jacob D. Durrant and J. Andrew McCammon. Binana: A novel algorithm for
 ligand-binding characterization. *Journal of Molecular Graphics and Modelling*,
 29(6):888–893, 2011.
- [29] Adam Ertel, Arun Verghese, Stephen Byers, Michael Ochs, and Aydin Tozeren.
 Pathway-specific differences between tumor cell lines and normal and tumor tissue
 cells. *Molecular cancer*, 5:55, 02 2006.
- [30] Mahya Fallahi-Sichani, Shadi Honarnejad, Laura M. Heiser, Joe W. Gray, and Pe-
 ter K. Sorger. Metrics other than potency reveal systematic variation in responses
 to cancer drugs. *Nature Chemical Biology*, 9(11):708–714, November 2013. Epub
 2013 Sep 8.
- [31] Nikta Feizi, Sisira Kadambat Nair, Petr Smirnov, Gangesh Beri, Christopher Ee-
 les, Parinaz Nasr Esfahani, Minoru Nakano, Denis Tkachuk, Anthony Mammoliti,

- Evgeniya Gorobets, Arvind Singh Mer, Eva Lin, Yihong Yu, Scott Martin, Marc Hafner, and Benjamin Haibe-Kains. Pharmacodb 2.0 : Improving scalability and transparency of in vitro pharmacogenomics analysis, 2021.
- [32] U.S. Food and Drug Administration (FDA). Considerations for the use of artificial intelligence in support of regulatory decision-making for drug and biological products, 2025. Accessed: 2025-01-15.
- [33] William F. Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, Axel Huebl, Mark Kim, James Kress, Tahsin Kurc, Qing Liu, Jeremy Logan, Kshiti Mehta, George Ostrouchov, Manish Parashar, Franz Poeschel, David Pugmire, Eric Suchyta, Keichi Takahashi, Nick Thompson, Seiji Tsutsumi, Lipeng Wan, Matthew Wolf, Kesheng Wu, and Scott Klasky. Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561, 2020.
- [34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [35] Goswami, Manash and Palagiri, Kundana. Deploying your models to gpus with onnx runtime in cloud and azure stack edge, 2020. [Online; accessed May 5th, 2021].
- [36] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 507–520. Curran Associates, Inc., 2022.
- [37] Jeff Hale. Deep learning framework power scores 2018. 2018.
- [38] P. C. D. Hawkins, A. G. Skillman, G. L. Warren, B. A. Ellingson, and M. T. Stahl. Conformer generation with omega: Algorithm and validation using high quality structures from the protein databank and the cambridge structural database. *Journal of Chemical Information and Modeling*, 50(4):572–584, 2010.
- [39] Darren J. Hsu, Russell B. Davidson, Ada Sedova, and Jens Glaser. tinyifd: A high-throughput binding pose refinement workflow through induced-fit ligand docking. *Journal of Chemical Information and Modeling*, 63(11):3438–3447, 2023. PMID: 37204814.
- [40] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 10.5mb model size, 2016.

- [41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.
- [42] 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.
- [43] K. Gregory Kurtzer, Vanessa Sochat, and W. Michael Bauer. Singularity: Scientific container for mobility of compute. *PLOS One*, 2017.
- [44] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: a llvm-based python jit compiler. In *LLVM '15: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, New York, NY, 2015. ACM.
- [45] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [46] Sarita Limbu and Sivanesan Dakshanamurthy. A new hybrid neural network deep learning method for protein–ligand binding affinity prediction and de novo drug design. *International Journal of Molecular Sciences*, 23(22):13912, Nov 2022.
- [47] Hannes H Loeffler, Jiazhen He, Alessandro Tibo, Jon Paul Janet, Alexey Voronov, Lewis H Mervin, and Ola Engkvist. Reinvent 4: Modern ai–driven generative molecule design. *Journal of Cheminformatics*, 16:20, 2024.
- [48] Hannes H. Loeffler, Shunzhou Wan, Marco Klähn, Agastya P. Bhati, and Peter V. Coveney. Optimal molecular design: Generative active learning combining reinvent with precise binding free energy ranking simulations. *Journal of Chemical Theory and Computation*, 20(18):8308–8328, Sep 2024.
- [49] Duncan C. McElfresh, Sujay Khandagale, Jonathan Valverde, C. VishakPrasad, Ben Feuer, Chinmay Hegde, Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When do neural nets outperform boosted trees on tabular data? *ArXiv*, abs/2305.02997, 2023.
- [50] M. McGann. Fred pose prediction and virtual screening accuracy. *Journal of Chemical Information and Modeling*, 51(3):578–596, 2011.
- [51] Alexander T. McNutt, Patrick Francoeur, Rishabh Aggarwal, Takashi Masuda, Regina Meli, Matthew Ragoza, Jocelyn Sunseri, and David R. Koes. Gnina 1.0: molecular docking with deep learning. *Journal of Cheminformatics*, 2021.
- [52] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.

- [53] Felix Meyer, Benjamin Hernandez, Richard Pausch, René Widera, David Groß, Sergei Bastrakov, Axel Huebl, Guido Juckeland, Jeffrey Kelling, Matt Leinhauser, David Rogers, Ulrich Schramm, Klaus Steiniger, Stefan Gumhold, Jeff Young, Michael Bussmann, Sunita Chandrasekaran, and Alexander Debus. Hardware-agnostic interactive exascale in situ visualization of particle-in-cell simulations. In *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC '23*, New York, NY, USA, 2023. Association for Computing Machinery.
- [54] K. Jarrod Millman and Michael Aivazis. Python for scientists and engineers. *Computing in Science Engineering*, 13(2):9–12, 2011.
- [55] Amanda J. Minnich, Kevin McLoughlin, Margaret Tse, Jason Deng, Andrew Weber, Neha Murad, Benjamin D. Madej, Bharath Ramsundar, Tom Rush, Stacie Calad-Thomson, Jim Brase, and Jonathan E. Allen. Ampl: A data-driven modeling pipeline for drug discovery. *Journal of Chemical Information and Modeling*, 60(4):1955–1968, 2020. PMID: 32243153.
- [56] National Cancer Institute. Genomic data commons, 2023. Accessed: 2023-05-10.
- [57] National Cancer Institute (NCI). *Predictive Oncology Model and Data Clearinghouse (MoDaC)*, 2023. Accessed: 2021-06-1.
- [58] National Energy Research Scientific Computing Center (NERSC). *Perlmutter Supercomputer (National Energy Research Scientific Computing Center, NERSC)*, 2022. Accessed: 2021-11-20.
- [59] National Cancer Institute (NCI). Nci60 drug response dataset, May 2021. Accessed: 2022-04-15.
- [60] National Cancer Institute (NCI). Nci60 molecular drug descriptors dataset, May 2021. Accessed: 2022-04-15.
- [61] NVIDIA Corporation. cudf: Gpu dataframe library. <https://github.com/rapidsai/cudf>, 2022. Documentation available at <https://docs.rapids.ai/api/cudf/stable/>.
- [62] NVIDIA Docker Developers. Nvidia container toolkit, 2024. Accessed: 2021-02-25.
- [63] Oak Ridge Leadership Compute Facility (ORNL). *Summit Supercomputer*, 2018. Accessed: 2021-03-11.
- [64] Oak Ridge Leadership Compute Facility (ORNL). *Frontier Supercomputer*, 2019. Accessed: 2023-05-15.
- [65] OpenEye, Cadence Molecular Sciences. Quacpac 2.2.5.0, 2024. Santa Fe, NM. <http://www.eyesopen.com>.

- [66] Alexander Partin. Doe-nci-pilot1 / normalization, 2023. Accessed: 2023-04-21.
- [67] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.
- [68] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python, 2018.
- [69] Rafael Peres da Silva, Chayaporn Suphavitai, and Niranjan Nagarajan. Tugda: Task uncertainty guided domain adaptation for robust generalization of cancer drug response prediction from *in vitro* to *in vivo* settings. *Bioinformatics*, 37(Supplement_1):i76–i83, July 2021.
- [70] James C. Phillips, David J. Hardy, Julio D. Maia, John E. Stone, João V. Ribeiro, Rafael C. Bernardi, Ronak Buch, Giacomo Fiorin, Jérôme Hémin, Wei Jiang, and et al. Scalable molecular dynamics on cpu and gpu architectures with namd. *The Journal of Chemical Physics*, 153(4), Jul 2020.
- [71] Franz Poeschel, Juncheng E, William F. Godoy, Norbert Podhorszki, Scott Klasky, Greg Eisenhauer, Philip E. Davis, Lipeng Wan, Ana Gainaru, Junmin Gu, Fabian Koller, René Widera, Michael Bussmann, and Axel Huebl. Transitioning from file-based hpc workflows to streaming data pipelines with openpmd and adios2. In Jeffrey Nichols, Arthur ‘Barney’ Maccabe, James Nutaro, Swaroop Pophale, Pravallika Devineni, Theresa Ahearn, and Becky Verastegui, editors, *Driving Scientific and Engineering Discoveries Through the Integration of Experiment, Big Data, and Modeling and Simulation*, pages 99–118, Cham, 2022. Springer International Publishing.
- [72] Reid Priedhorsky and Tim Randles. Charliecloud: Unprivileged containers for user-defined software stacks in hpc. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC ’17, New York, NY, USA, 2017. Association for Computing Machinery.
- [73] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Damos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David

- Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. Mlperf inference benchmark. <https://arxiv.org/abs/1911.02549>, 2020.
- [74] Colin Reid. gdc-rnaseq-tool, 2023. Accessed: 2023-04-21.
- [75] William C. Reinhold. Nci60 rna-sequence gene expression value dataset. Accessed: 2023-03-13.
- [76] Matthew Rocklin and contributors. Dask: Library for dynamic task scheduling. <https://dask.org>, 2015.
- [77] Philip C. Roth. Developing software for olcf frontier. <https://ecpannualmeeting.com/assets/overview/sessions/Roth-Path-to-Frontier-20200206.pdf>, 2020.
- [78] Rudolf Eigenmann, Benjamin E. Bagozzi, Arthi Jayaraman, William Totten, and Cathy H. Wu, University of Delaware, DARWIN Team. *Delaware Advanced Research Workforce and Innovation Network (DARWIN)*, 2021. Accessed: 2021-03-11.
- [79] Aymen Al Saadi, Dario Alfe, Yadu Babuji, Agastya Bhati, Ben Blaiszik, Alexander Brace, Thomas Brettin, Kyle Chard, Ryan Chard, Austin Clyde, Peter Coveney, Ian Foster, Tom Gibbs, Shantenu Jha, Kristopher Keipert, Dieter Kranzlmüller, Thorsten Kurth, Hyungro Lee, Zhuozhao Li, Heng Ma, Gerald Mathias, Andre Merzky, Alexander Partin, Arvind Ramanathan, Ashka Shah, Abraham Stern, Rick Stevens, Li Tan, Mikhail Titov, Anda Trifan, Aristeidis Tsaris, Matteo Turilli, Huub Van Dam, Shunzhou Wan, David Wifling, and Junqi Yin. Impeccable: Integrated modeling pipeline for covid cure by assessing better leads. In *Proceedings of the 50th International Conference on Parallel Processing, ICPP '21*, New York, NY, USA, 2021. Association for Computing Machinery.
- [80] Frank Seide and Amit Agarwal. Cntk: Microsoft’s open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*, page 2135, New York, NY, USA, 2016. Association for Computing Machinery.
- [81] Robert H. Shoemaker. The nci60 human tumour cell line anticancer drug screen. *Nature Reviews Cancer*, 6(10):813–823, 2006.
- [82] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *CoRR*, abs/2106.03253, 2021.

- [83] Petr Smirnov, Victor Kofia, Alexander Maru, Mark Freeman, Chantal Ho, Nehme El-Hachem, George-Alexandru Adam, Wail Ba-alawi, Zhaleh Safikhani, and Benjamin Haibe-Kains. Pharmacodb: An integrative database for mining in vitro anticancer drug screening studies. *Nucleic Acids Research*, 46(D1), 2017.
- [84] Jayashree Srinivasan, Thomas E. Cheatham, Piotr Cieplak, Peter A. Kollman, and David A. Case. Continuum solvent studies of the stability of dna, rna, and phosphoramidatedna helices. *Journal of the American Chemical Society*, 120(37):9401–9409, 1998.
- [85] Aravind Subramanian and the LINCS Consortium. Broad institute human 11000 epsilon dataset, June 2015. Accessed: 2022-05-1.
- [86] Roberto Todeschini and Viviana Consonni. *Molecular Descriptors for Chemoinformatics*. Wiley-VCH, Weinheim, Germany, 2nd edition, 2009.
- [87] Katarzyna Tomczak, Patrycja Czerwinska, and Maciej Wiznerowicz. The cancer genome atlas (tcga): An immeasurable source of knowledge. *Contemp Oncol (Pozn)*, 19:A68–A77, 01 2015.
- [88] University of Delaware, IT Research Computing Group. *Caviness High-Performance Computing Cluster*, 2021. Accessed: 2021-03-11.
- [89] Stéfan van der Walt, S Chris Colbert, and Gaël Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, Mar 2011.
- [90] Shunzhou Wan, Agastya P. Bhati, Stefan J. Zasada, and Peter V. Coveney. Rapid, accurate, precise and reproducible ligand–protein binding free energy prediction. *Interface Focus*, 10(6):20200007, 2020.
- [91] Qingguo Wang, Joshua Armenia, Chao Zhang, Alexander Penson, Ed Reznik, Ligu Zhang, Thais Minet, Angelica Ochoa, Benjamin Gross, Christine Iacobuzio-Donahue, Doron Betel, Barry Taylor, Jianjiong Gao, and Nikolaus Schultz. Unifying cancer and normal rna sequencing data from different sources. *Scientific Data*, 5:180061, 04 2018.
- [92] Justin M Wozniak, Philip E Davis, Tong Shu, Jonathan Ozik, Nicholson Collier, Manish Parashar, Ian Foster, Thomas Brettin, and Rick Stevens. Scaling deep learning for cancer with advanced workflow storage integration. In *2018 IEEE/ACM Machine Learning in HPC Environments, MLHPC 2018*, pages 114–123. Institute of Electrical and Electronics Engineers Inc., 2019.
- [93] Justin M Wozniak, Rajeev Jain, Prasanna Balaprakash, Jonathan Ozik, Nicholson T Collier, John Bauer, Fangfang Xia, Thomas Brettin, Rick Stevens, Jamaludin Mohd-Yusof, Cristina Garcia Cardona, Brian Van Essen, and Matthew

- Baughman. CANDLE/Supervisor: a workflow framework for machine learning applied to cancer research. *BMC Bioinformatics*, 19(18):491, December 2018.
- [94] Justin M Wozniak, Rajeev Jain, Prasanna Balaprakash, Jonathan Ozik, Nicholson T Collier, John Bauer, Fangfang Xia, Thomas Brettin, Rick Stevens, Jamaludin Mohd-Yusof, et al. Candle/supervisor: A workflow framework for machine learning applied to cancer research. *BMC bioinformatics*, 19(18):59–69, 2018.
- [95] Yanzhao Wu, Ling Liu, Calton Pu, Wenqi Cao, Semih Sahin, Wenqi Wei, and Qi Zhang. A comparative measurement study of deep learning as a service framework. *IEEE Transactions on Services Computing*, pages 1–1, 2019.
- [96] E. Zenker, B. Worpitz, R. Widera, A. Huebl, G. Juckeland, A. Knüpfer, W. E. Nagel, and M. Bussmann. Alpaka – an abstraction library for parallel kernel acceleration. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 631–640, 5 2016.
- [97] Yuqing Zhang, Giovanni Parmigiani, and W Evan Johnson. ComBat-seq: batch effect adjustment for RNA-seq count data. *NAR Genomics and Bioinformatics*, 2(3):lqaa078, 09 2020.