# An Automation Framework for Comparison of Cancer Response Models Across Configurations

Justin M. Wozniak,* Rajeev Jain,* Andreas Wilke,* Rylie Weaver,*
Alexander Partin,* Thomas Brettin,* and Rick Stevens*
* Argonne National Laboratory, Lemont, IL, USA

***Abstract*—Machine learning has made significant advancements in precision medicine, resulting in the development of various deep learning applications. For instance, in cancer drug response prediction, numerous deep learning models have been created. However, comparing these models across vast configurations of hyperparameters and data sets can be challenging. In this paper, we introduce a new scalable workflow suite that aims to answer questions that arise when comparing different models developed by different teams on similar or the same problems. We explain the problem in more detail and discuss our approach using near-exascale or exascale computers.**

## I. Introduction

The intersection of precision medicine and machine learning (ML) offers a wide range of problems and possible approaches. The prediction of tumor response to single and combination drug agents is an active area of ML application development, as tens of deep learning (DL) models are currently available and under active development. Comparing the behavior of these models is very difficult, and is not a well-studied area, as different projects differ wildly in their problem assumptions and approach to the problem.

Both cancer studies and DL models have a range of configuration parameters in which researchers can operate. Some of these include:

1) Data sources- varying studies and experiment types. Different laboratory studies could have subtle differences in data collection.
2) Data representation- varying data formats. For example, a drug could be represented as a SMILES string or a series of features.
3) Algorithms and architectures- varying models could have very different algorithmic structure or DL architectures. While inputs and outputs may be comparable, interpreting variations could be very difficult.

The IMPROVE project at Argonne National Laboratory is collecting and curating AI models for cancer and similar precision medicine problems. IMPROVE has collected tens of models in drug response prediction (DRP). Of these, 28 have been successfully run by the computer science contingent of the team, making for a valuable resource both for precision medicine and computer science investigations in DL. They vary in DL runtime, architecture, and DL approach.

The IMPROVE approach consists of two aspects: a packaging framework and workflow framework. The packaging framework consists of conventions for bundling each model inside a Singularity container and exposing a common shell script interface to perform tasks such as data preprocessing, training, and inference. The second aspect is the subject of this paper. This consists of extensions to the CANDLE/Supervisor system to run multiple models from the IMPROVE packages and compare them across a range of configurations. In this paper, we consider an initial problem of interest to the model comparison topic. Consider a case in which we want to perform cross-model comparison for two models over some range of model parameters, such as underlying training data error or noise, computation constraints, or other such range. It is up to the user to specify the range of configuration values for across which the comparison will take place. It will be up to the system to compare the models across the range of settings and underlying data conditions. We will want to compare error values at coarse-grained level, without investigating root causes of error deviations. The system will be expected to automatically find regions of interest, that is, conditions under which the models vary, so that user investigations may proceed on those subproblems.

The space of possible conditions is very large. It could be addressed by simply running a large run flat sweep over the problem space, but this produces hundreds to thousands of training runs and is intractable, as will be described. To allow for automation and scalability of this problem, we recast the model comparison run as a new workflow-compatible "model" that may be probed by a general-purpose hyperparameter search workflow. This new model will be configured to accept two actual deep learning models, train them nearly identically, except for user-specified variations and report differences in errors. Thus, the approach demonstrates the generality of a scalable automated search framework for deep learning, and provides a viable solution for a range of deep learning model comparison studies.

The remainder of this paper is organized as follows. In §II, we describe the problem in more detail and consider alternative approaches. In §III, we describe cancer drug response prediction with the Uno model for this problem. In §IV, we describe the notion of model comparison in more detail and narrow the scope of this paper. In §V, we describe the workflow framework in more detail. In §VI, we describe the use of containers by our system and how they enable model comparison. In §VII, we present a model comparison case and its experimental results. In §VIII, we recapitulate and conclude and in §IX we describe directions for future work.

## II. Background

In this section, we provide background on the concept of deep learning model comparison.

Machine learning (ML) has the capability to transform many scientific problems. In response to the growing power of ML techniques and the increasing available computing power at large scale computing facilities, the U.S. Department of Energy Exascale Computing Project (ECP) launched the Cancer Distributed Learning Environment (CANDLE). CANDLE developed a suite of software to support scalable deep learning on DOE supercomputing resources. While the CANDLE project is explicitly aimed at supporting deep learning in the three cancer pilot projects in the near-term, its longer-term goal is to support a wide variety of deep learning applications across DOE science domains.

### A. Frameworks for machine learning

Deep learning frameworks are under active development by diverse research communities in both industry (Google, Facebook, Microsoft, etc.) and academia (Berkeley, Oxford, Toronto, etc.). These include Caffe [1], Keras [2], Theano [3], Torch [4], Poseidon [5], Neon [6], TensorFlow [7], CNTK [8], and the Livermore Big Artificial Neural Net (LBANN) [9]. Each of these frameworks differ with respect to the machine learning tasks they target, their ease of use, data preprocessing, and target problems. Most frameworks were architected for a single node implementation and a few distributed memory multi-node implementations have recently emerged; but these implementations are primarily targeted at smaller core counts and for commodity cluster environments. Moreover, these implementations rely on avoiding communication by storing data on local disks. Implementations targeting high-performance computing systems will need novel techniques to fully exploit the system interconnect bandwidth and topologies, as well as the deep memory hierarchies.

### B. Hyperparameter search

Before a model can be trained, a set of hyperparameters must be selected to initialize the model with. The choice of hyperparameters is not only important to performance, but very computationally expensive to check, as each choice of hyperparameters requires the training of a model. As an influential but costly factor in model comparison, hyperparameter optimization (HPO) is a key part the IMPROVE project.

Moreover, since many hyperparameter spaces are non-differentiable with multiple local minima [10]some common optimization approaches are weakened or not viable with HPO. In this section, we introduce some common methods and their pros/cons. Some of the most common algorithms/categories of algorithms for HPO are [11]:

- Grid Search
- Random Search
- Gradient-Based Optimization
- Bayesian Algorithms
- Population Based Algorithms (Genetic and Swarm)

Grid search checks a lattice of hyperparameters and random search randomly checks hyperparameters throughout the hyperparameter space. Both are simple and have some success in smaller hyperparameter spaces, but perform poorly compared to other algorithms in larger and/or higher-dimensional hyperparameter spaces, like many used in the CANDLE workflows. Moreover, grid search tends to be less effective with continuous parameters, because of its discrete checking. Random search tends to be less effective for conditional hyperparameters, because of its inability to take into account previous hyperparameter outputs [11]. Grid search and Random search can easily be implemented with the Python library scikit-learn [12].

Gradient-based optimization extends the idea of gradient descent to hyperparameters. However, calculating the gradient is not possible because the hyperparameter space is generally non-differentiable. Instead, this method explores options around a given hyperparameter selection and goes in the direction of most improvement. Gradient-based optimization can often get trapped in a local minimum instead of finding the global minimum.

Bayesian optimization is a probabilistic model that iteratively picks hyperparameter values and updates itself. Firstly, a probabilistic model of the hyperparameter space is built. Then, a new set of hyperparameter values in the space is chosen with the incentive tradeoff of exploration and optimization. Finally, the evaluation of that selection of hyperparameter values is used to update the probabilistic model, and the process of is repeated until a stopping point has been reached. Bayesian optimization can use varied probabilistic models, including gaussian processes, random forests, and the tree Parzen estimator [11], which come with different libraries in python that can be used for implementation. HyperOpt [13] is one such framework that is designed to perform searches using distributed hardware and accomodate Bayesian optimization which has a scikit-learn variant [14]. Depending on the specific model used, there are different weaknesses to Bayesian hyperparameter optimization, but the class of methods is generally effective [11].

Genetic algorithms (GAs) model the process of natural selection, usually through some combination of mutation, crossover (mating), and selection for many simulated generations. Genetic algorithms tend to be efficient at handling all types of hyperparameter spaces [11]. The NeuroEvolution of Augmenting Topologies (NEAT) algorithm [15] is one example of a GA. Like other GAs, NEAT spawns a genome and 'evolves' with mutation, crossover, and selection, but unlike other GAs, NEAT evolves both the nodes and the topology of the network. NEAT (or at least, its specific NEAT-Python [16] implementation) has added variability through its alteration of intra-node weights and parameters that can prove beneficial by reducing loss at the "starting point" of training, but also has the drawback that this serves as a topology specific feature that somewhat precludes comparison of pure topological strengths and weaknesses.

Particle swarm algorithms (PSOs) simulate the collective behavior of decentralized systems, like an ant colony or beehive. In this algorithm, the individual 'particles' have

communication with each other such that each particle in the population makes decisions on their location (hyperparameters) based on not just their best performing location, but the best performing location of the group or the other particles around them. Swarm algorithms are effective for all types of hyperparameter spaces and are easily parallelized, but need proper initialization [11].

Lastly, several techniques and libraries exist to accelerate model exploration. Some algorithms implement "multi-fidelity optimization" [11], where if a model is performing poorly, it stops being trained. This reduces computation time because less time is spent accurately evaluating models, but decreases accuracy in those evaluations. One example of multi-fidelity optimization is Bayesian Optimization Hyperband (BOHB) [17].

### C. Approaches used by CANDLE

*1) mlrMBO:* In prior work, we focused on mlrMBO [18] which is an R package that uses bayesian optimization for its HPO algorithm. It is designed for optimization problems with mixed continuous, categorical and conditional parameters. The specific iterative approach of mlrMBO is as follows. In the initialization phase, $n_s$ configurations are sampled at random, evaluated, and a surrogate model $M$ is fitted with the input-output pairs. In the iterative phase, at each iteration, $n_b$ promising input configurations are sampled using the model $M$. These configurations are obtained using infill criterion that tries to trade-off exploitation and exploration. The algorithm terminates when user-defined maximum number of evaluations and/or wall-clock time is exhausted.

We focused on mlrMBO as it was shown to obtain state-of-the-art performance on a wide range of test problems, where it was benchmarked against other approaches such as DiceOptim, rBayesianOptimization, SPOT, SMAC, Spearmint, and Hyperopt. Crucial to the effectiveness of mlrMBO is the choice of the algorithm used to fit $M$ and the infill criterion. Given the mixed integer parameters in the hyperparameter search, we used random forest [19] because it can handle such parameters directly, without the need to encode the categorical parameters as numeric. For the infill criterion, we used the qLCB [20], which proposes multiple points with varying degrees of exploration and exploitation

*2) DEAP:* We also use Distributed Evolutionary Algorithms in Python (DEAP) [21], which is a Python framework to easily use and customize evolutionary algorithms that can be implemented with Optunity [22], a hyperparameter tuning engine. DEAP allows users significant flexibility and various options with the mutation, crossover, and selection operations with minimal 'under the hood' understanding/work [23], allowing easy implementation of complex algorithms. Moreover, DEAP does not alter intra-node weights and parameters as NEAT does. Specifically, we use DEAP to implement a genetic algorithm.

In our project, DEAP's customization allows us to manage varied types of hyperparameters and explore the hyperparameter space rigorously. For example, the crossover (mating) operation can be customized to swap hyperparameters with their mating partners, instead of averaging, which prevents integer-type hyperparameters from becoming floats. The crossover operation provides a way for individuals in the population to jump out of local minima, even if that local minima is very steep. Additionally, DEAP allows the natural selection pressure to be tuned higher or lower with different selection methods and parameters like 'tournsize' such that more of the parameter space is explored before the population converges to the best local minimum found. Effective implementation of genetic algorithms usually includes a population size and number of generations in the hundreds, which could lead to non-optimal computation time in smaller parameter spaces. However, in large parameter spaces, DEAP presents a robust framework for hyperparameter optimization, with much available tuning in its algorithm creation. Optunity acts as an interface between DEAP and the network to be optimized, allowing for easy deployment of these various algorithms for the purpose of hyperparameter optimization. Optunity is an excellent implementation of evolutionary algorithms for the purpose of hyperparameter tuning.

### D. Comparison with prior CANDLE work

As in prior work, we use the CANDLE/Supervisor system which internally uses the EMEWS [24] framework to directly incorporate parameter exploration methods for efficient exploration of order > $10^9$ spaces. This framework uses the Argonne-developed Swift/T [25], [26] language to distribute the model exploration workload efficiently across a large-scale multi-node system using MPI.

This work extends prior work by the CANDLE team in multiple ways. It demonstrates a novel use of the CANDLE/Supervisor workflow framework [27] by applying it to a new problem space, a model difference maximization rather than a hyperparameter search. It extends [28] by focusing on a different model comparison problem, a spatially-oriented search problem, rather than applying cross-study analysis from one study to another.

## III. Drug response prediction

In this section, we describe the Uno model for drug response prediction, and compare it to alternate approaches. We also describe the underlying data.

A highly desirable goal in the application of deep learning is to enable cross-comparison of cancer studies and integrate results into a unified drug response model. The overall idea is to train a neural network (NN) on a corpus of tumor dose responses based on given combinations of cell RNA sequences, drug descriptors, and drug fingerprints. The model can then provide predictions for combinations of RNA sequences and drugs that it was not trained on. The drug response is captured in an AUC (area-under-the-curve) score, a typical measurement in this application area.

The Uno benchmark [28], [29] integrates experimental cancer data from 2.5 million samples across six research centers to examine study biases and to build a unified drug response model. The associated manually designed DNN has four input layers: a cell RNA sequence layer, a dose layer,

a drug descriptor layer, and a drug fingerprints layer. It has three feature-encoding submodels for cell RNA sequence, drug descriptor, and drug fingerprints. Each submodel is composed of three hidden layers. The last layer for each of the submodels is connected to the concatenation layer along with the dose layer. This is connected to three hidden layers. The scalar output layer is used to predict tumor dose.

### A. The Uno data sets

Uno training data consists of a composite data set that can be partitioned in multiple ways. Multiple user-selected data sets can be dynamically assembled before training by selecting from drug response, gene expression, and molecular descriptors of interest; all from separate studies. Uno can process data from user-defined combinations of data from at least 5 studies, including NCI60, CTRP, CCLE, GDSC, and gCSI [28]. In this paper, we focus on CCLE and gCSI because they are small and train quickly, enabling us to study the computational aspects in a reasonable amount of development time. On our test system Lambda7 (described below (§VII-D)), the per-epoch times were gCSI at 13 seconds, CCLE at 17 seconds, CTRP at 317 seconds, and the others much longer.

## IV. Motivation: Model comparison

In this section, we provide a high-level overview of the goals and approaches of model comparison, and specify the smaller scope of the comparisons investigated in this paper.

### A. Overview of model comparison

The overall goal of this effort is to produce a semi-automated framework for doing model comparisons, automating as much as possible. A range of cancer models are now being produced by the community and determining which models are better under varying scenarios is very important. Models may differ greatly on the representation of the underlying data, neural network architecture, and other technical differences. They may also differ in their intended approach and outcome, whether focusing on performance, accuracy, or some combination with respect to a particular range of scenarios.

The ideal model for such a model comparison study should be able to handle pan-cancer and multi-drug scenarios, and provide features that facilitate easy data curation. Additionally, we are interested in models that can predict continuous drug response in cell lines, as measured by AUC or IC50. However, we are not particularly interested in drug-specific models that do not utilize drug features, or models that rely on complex, application-specific feature representations. Our main focus is on finding models that are practical, efficient, and able to deliver accurate results across a range of scenarios.

In this paper, we are prioritizing the following comparisons, in which models are assumed to be built to solve similar problems, such as cross-study generalization analysis. Cross-study generalization analysis demonstrates how well a model that is trained on data from one study predicts the response label for samples from a second study performed at a different location by a different team, for example a model trained on CCLE data and used to predict NCI-60 response labels.

The learning curve of the training pattern is also relevant. In the context of deep learning model comparisons, we treat a learning curve as a plot of model learning performance over experience where experience is the number of samples in the training set. This enables one to visualize model performance as a function of training data. We are also interested in a range of possible error analyses and and error subclass/subtyping. While it would be useful to be able to report the $N$ top models using the framework that IMPROVE is curating for a given problem, this paper focuses on a more broad approach, in which we assume that a user is interested in finding and isolating cases in which the models differ to gain insight into the modeling process.

### B. Scope of this paper

In this paper, we focus on the narrow problem of building an eScience workflow that can perform the numerical and distributed computing operations needed to make a narrow comparison of two models on a narrowly-defined problem. We focus on a case in which a user has two similar models and desires to obtain scenarios in which they perform most differently.

## V. Infrastructure

In this section, we describe the CANDLE distributed deep learning infrastructure used to run automated model comparison workflows.

### A. CANDLE overview

The Cancer Deep Learning Environment (CANDLE) is an open framework for rapid development, prototyping, and scaling deep learning applications on high-performance computing (HPC) systems. CANDLE was initially developed to support a focused set of three pilot applications jointly developed by cancer researchers and deep learning / HPC experts, but is now generalizable to a wide range of use cases. It is designed to ease or automate several aspects of the deep learning applications development process. CANDLE runs on systems from individual laptops to OLCF Frontier, and enables researchers to scale application workflows to the largest possible scale.

Our workflow system, CANDLE/Supervisor [30], is a workflow application framework used to develop multiple deep learning workflows. It is a Supervisor in the sense that it manages the execution of many (thousands) of concurrent, subordinate deep learning training or inference runs. CANDLE/Supervisor was designed to run a wide range of model types, including TensorFlow and PyTorch. It is also generic with respect to the application area. For example, CANDLE runs models in RNA expression data, molecular dynamics data, and clinical text data.

Key components of the CANDLE architecture are shown in Figure 1. A handful of top-level workflows have been developed by the CANDLE team, but these can easily be modified or extended. These include 1) hyperparameter search, in

| | | | |
|---|---|---|---|
| **Network architecture** | | **Training limits** | |
| dense | [1000,1000,1000,1000,1000] ∗ | epochs | 10 |
| dense_feature_layers | [1000,1000,1000] | timeout | 3600.0 |
| scaling | 'std' | **Noise injection** | |
| dropout | 0.1 | noise_add | true |
| out_act | 'softmax' | layer_force | 1000 |
| activation | 'relu' | noise | 20% ∗ |
| **Training settings** | | **Model Specific** | |
| optimizer | 'adamax' | train_sources | ['CCLE', 'gCSI'] ∗ |
| loss | 'mse' | cell_features | ['rnaseq'] |
| metrics | 'accuracy' | drug_features | ['descriptors'] |
| batch_size | 32 | test_sources | ['train'] |
| learning_rate | 0.0001 | agg_dose | 'AUC' |

TABLE I
HYPERPARAMETERS USED FOR UNO BENCHMARK IN THIS PAPER.

which a extendable and replaceable 3rd-party algorithm is used to probe hyperparameter configurations [27], 2) uncertainty quantification workflows [31], in which model sensitivity is probed using noise injection and possible abstention, and 3) training data analysis [32], in which models are trained on various data subsets which allows for insights into the underlying data.

### B. CANDLE compliance

CANDLE has developed the notion of CANDLE compliance, a set of guidelines and interfaces to make running deep learning models easier. It also makes it possible to invoke them from a common workflow framework. CANDLE compliance includes the notion of "CANDLE hyperparameters" and a few Python methods. CANDLE hyperparameters are generalized to include system settings and other configurations that are not always called hyperparameters in the strict machine learning sense. The required Python methods include a `run()` method that accepts a simple Python dict of hyperparameters, and a small number of initialization methods. CANDLE is a lightweight framework that specifies a JSON format for hyperparameters and logging format for model results, but delegates the storage and use of hyperparameters and results to external algorithms and the deep learning model. Typical results used by workflows are performance or error metrics such as validation loss.

CANDLE has released a library to help with CANDLE compliance. The `candle_lib` package is a `pip`-installable library designed to standardize and streamline machine learning code development and deployment. Originally developed as part of the CANDLE Benchmarks suite, it is now a independently-installable library that provides various utilities including integration with the CANDLE/Supervisor framework to automate running complex workflows on exascale machines. CANDLE-compliant models can use this simple API and its underlying functionality via the hyperparameter controls which control both network architecture *and* system-level functionality including checkpoints [33].

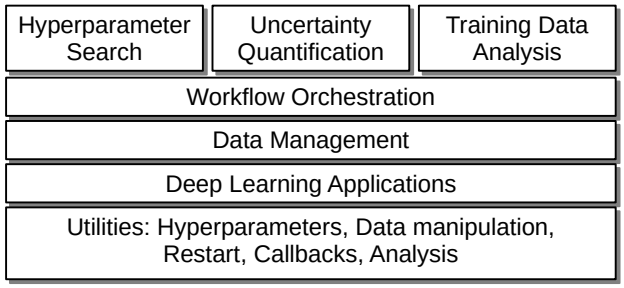



Fig. 1. CANDLE component architecture. Users modify and extend the top-level workflows and the deep learning applications that plug into the system, while benefitting from reusable system components.

The benchmark runs in this paper used the CANDLE hyperparameters shown in Table I. Hyperparameters shown with ∗ indicate typical values but in this work may be automatically probed by the workflow.

### C. Software architecture for model comparison

Key components of the CMP (CoMParator) workflow are depicted in Figure 2. The workflow is driven by the DEAP algorithm for optimization (1), and is configured similarly to its use in hyperparameter search by CANDLE. The search workflow itself (2) is an unmodified reuse of the `GA` workflow in the CANDLE/Supervisor workflow suite. The CMP Model (3), described in the following, is invoked on the hyperparameters selected by the algorithm and conveyed by the workflow system to an available worker for execution. The CMP Model then translates hyperparameters as needed for each of the two underlying models (4), here shown as "Uno Model [0]" and "Uno Model [1]." The results produced by the two submodels are then differenced (5) and written back to the system by the CMP model as though produced by a single model. The algorithm registers the results and proceeds to additional iterations.

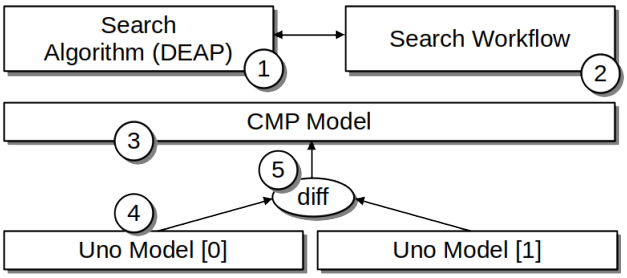



Fig. 2. The CMP workflow. The CMP Model satisfies the interfaces needed to be run inside a CANDLE workflow, but simply executes other models and returns the difference.

The novel part of the workflow for this effort is in the CMP Model and below. The CMP Model is a CANDLE-compliant (§V-B) software module. CMP, however, is not a deep learning model. It is a plain Python module that runs two other CANDLE-compliant models. It currently runs them sequentially, although it would be straightforward to run them concurrently given appropriate handling of the GPUs. It then returns the difference of the two models as specified by the CANDLE `MODEL_RETURN` setting, which is typically validation loss `val_loss`, although it could be any other error metric or value such as a performance metric.

The behavior of the CMP Model is controlled by CANDLE hyperparameters. For example, the hyperparameter set for the CMP run with Uno would contain some common

```
[Global_Params]

# The CMP Model:
model_name = 'cmp'
# The Uno submodels as containers:
cmp_model_name = '/tmp/woz/Uno.sif'
# Additional hyperparameter file for submodels:
cmp_config_file = '/candle_data_dir/uno_auc_model.txt'

# Per-model hyperparameters:
cmp_0_train_sources = 'CCLE'
cmp_1_train_sources = 'gCSI'
cmp_0_run_id = 'M0'
cmp_1_run_id = 'M1'

# Common hyperparameters:
epochs = 10
```

Fig. 3. CANDLE hyperparameter file for CMP Model run. Hyperparameters starting with `cmp_` are treated specially by CMP and passed down to the submodels.

hyperparameters and some specific hyperparameters for each submodel, as shown in Figure 3. CMP passes hyperparameters directly to the submodels, except for those prefixed with `cmp_` or `cmp_[01]_`. These prefixes are stripped. The `cmp_` hyperparameters are only visible in the submodels, and the numbered hyperparameters `cmp_[01]_` are only visible in the corresponding submodel 0 or 1.

The CMP Model currently returns a relative difference scaled by the inverse of the larger value:

$$V_{\text{CMP}} = \frac{-|V_1 - V_0|}{\max(V_1, V_0)} \tag{1}$$

where $V_{\text{CMP}}$ is the CMP Model result and $V_0$ and $V_1$ are the performance results for models $M_0$ and $M_1$ respectively. $V_{CMP}$ is always negative so minimizing this value maximizes the relative difference. This can be easily reconfigured for other studies. This is to handle cases such as the noise analysis done herein (§VII) in which there is large variation from very small errors to very large errors, and returning an absolute difference would strongly bias the search toward the large error magnitudes, even if the relative difference between them is trivial. We assume that future studies will require more complex differencing functions to handle other cases, possibly with user feedback and steering to orient the search toward or away from known areas of interest or disinterest.

## VI. Model sharing via Linux containers

In this section, we describe how community models can be shared via Linux containers.

### A. Scientific software distribution via containers

Containers allow the packaging and distribution of tools together with their execution environment. While multiple solutions exist to package software environments, e.g., Conda or Spack, container technology allows the simultaneous execution of programs without loading and unloading conflicting packages. This is especially handy for workflow systems. We have various model codes depending on different libraries and software versions. To run complex model workflows, we enforce standardized interfaces for model training and inference. All models are packaged inside a container for distribution and easy execution. Also, Docker provides a convenient way of packing software and distributing container images, we are focusing on Singularity [34] or Apptainer [35] since most HPC systems support it nowadays. Apptainer is rootless by default and allows unprivileged users (non-root) to make use of containers on HPC systems. In addition, it interacts seamlessly with GPUs and network file systems.

### B. Container use for community model comparison

To create workflows that can scale model execution and comparisons, we need to make sure that all supported model codes follow the same interface specification and are packaged in a container. Furthermore, containers need to expose a common interface to the workflow execution engine. There is little to no overhead for the workflow system other than that needed to ensure data is properly mounted inside the container and results can be written out.

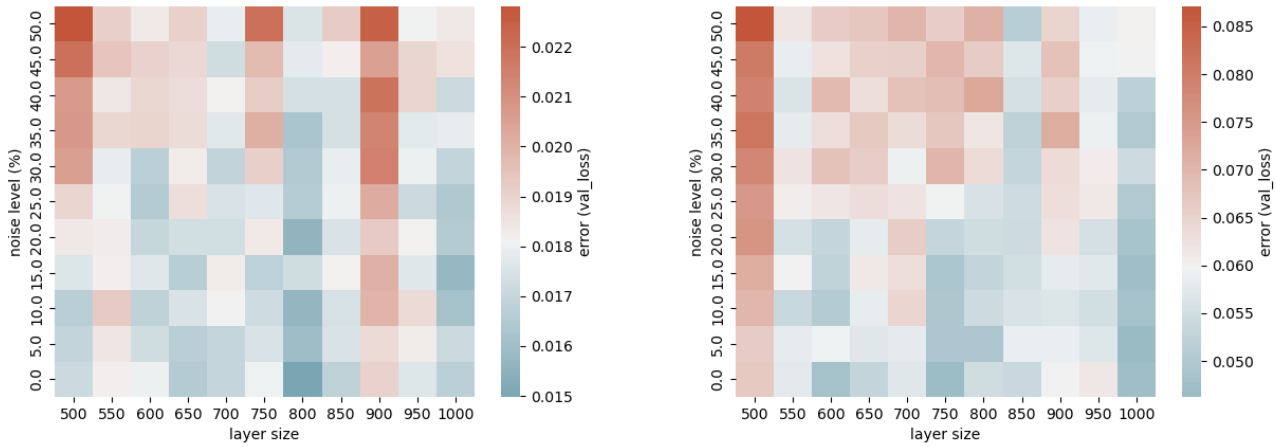
In a manner similar to CANDLE-compliance (§V-B) the IMPROVE framework establishes a set of rules for model codes and their containers to comply with. Every model code provides shell scripts for data preprocessing, training, and inference with identical interfaces. Within the IMPROVE project, the computer science team ran and validated 28 models. These models have been developed using different model engines and their versions (8 models with TensorFlow v1, 1 model with TensorFlow v2, and 19 with PyTorch). Ten of these models have been made compliant and are exposing identical interfaces. By encapsulating the diverse model codes and their deployment requirements into a container, we can now execute the model codes without managing software dependencies for the different models.

The combination of using containers for the distribution and execution of model codes together with the standardized interfaces is enabling plug-and-play of these models into our workflows. The shell script framework includes:

1) `preprocess.sh`: Data preprocessing. For example, this could transform raw drug response prediction data into a data format more amenable to machine learning.
2) `train.sh`: Model training. Ingests machine learning-formatted data to train the model and saves the model along with relevant statistics and metadata. This is the script invoked by the CMP workflow described here.
3) `infer.sh`: Model inferencing. Uses the trained model and machine learning data to perform inferences and saves raw predictions or other relevant statistics.

A CANDLE convention is to use the environment variable `CANDLE_DATA_DIR` to specify where training data and other configuration information may be found. In IMPROVE containers, this is always mapped to `/candle_data_dir` inside the container via the Singularity `--bind` flag.

A large collection of containerized machine learning models built for similar problems is a valuable resource and we expect a range of future studies to be made possible; model comparison is just one aspect of many viable research paths.




(a) Uno model performance with dataset CCLE.

(b) Uno model performance with dataset gCSI.

Fig. 4. Uno model performance as trained by two different datasets. Layer size is the number of neurons in each of the dense layers in the Uno model. Noise is the scale of the Gaussian noise applied to the drug response.

## VII. Case studies in model comparison

In this section, we describe how model comparison can be automatically performed. As a demonstration, we use the Uno model as trained on two different underlying datasets.

### A. Model performance in Uno

In this experiment we use the single-drug response prediction from two public datasets that are widely used resources in the field of cancer drug response prediction and training: 1. the gCSI (Genentech Cell Line Screening Initiative [36]) and 2. CCLE (Cancer Cell Line Encyclopedia [37]). It must be noted that the gCSI dataset contains data on a diverse set of compounds, including both FDA-approved drugs and investigational agents, allowing for a more comprehensive analysis of drug response. The CCLE dataset, while providing some drug response data, primarily focuses on genomic and transcriptomic characterization of cell lines rather than extensive drug profiling. There might be variations in the specific cancer cell lines included in each dataset. Some cell lines may be present in both datasets, but the overall composition and coverage of cell lines may differ. Prior work [28] showed that the gCSI is the most predictable among the several cell line data sets considered in their cross-validation study.

In the workflows described in the following we use the above two datasets and study the difference in performance of the models trained on the two datasets. We vary the number of dense layers (neurons) and the noise level in the data. We use the same hyperparameters for both the datasets, only the training data is modified.

To illustrate the behavior of the two models we formulate a scenario in which training data noise and the number of neurons are varied, to study hypothetical cases in which data quality is variable and/or computational constraints are imposed.

The Gaussian noise is applied to the labels (drug response AUC values) by adding a Gaussian random variable. This variable has expected value 0, with $\sigma$ set to range of possible percentages of the mean value of the labels found in the unmodified training data. This was applied by using `numpy.random.normal()` as the data is ingested. The noise level was scaled by a given percentage, here called the noise level, from 0%-50%.

We vary the number of neurons in the Uno deep learning model by simply overriding the default neuron count in each dense layer of the model with a value in the range from [500,1000].

We ran these scenarios in a simple "flat" CANDLE/Supervisor workflow called "dense-noise." The CMP Model was not used, we simply called the Uno model in the Supervisor `container` mode, which invokes the Uno container from the Swift/T worker node via the shell command `singularity exec`. Each model ran on 1 GPU.

### B. Individual model performance

We first illustrate individual model performance by sampling each value in the grid of hyperparameter space defined by our scenario. We first desire to see the full hyperparameter space as defined by our grid. Note that this is only tractable for a coarse sampling of a low-dimensional parameter space, herein we use two parameters. For more parameters an automated search would be required.

The grid search was performed on *Polaris* at the Argonne Leadership Computing Facility [38]. Polaris is a 560 node system based on 1 AMD EPYC "Milan" processor per node, with 4 NVIDIA A100 GPUs per node. Each node has 512 GB RAM and a 3 TB local NVRAM filesystem, with access to a Lustre parallel filesystem. Each GPU appears to TensorFlow as 2 GPUs, so we ran 8 TensorFlow models concurrently per node.

Our Singularity container on Polaris for Uno runs Python 3.6.9 and TensorFlow 2.4.2 on CUDA 11.4. The image is based on Ubuntu 18.04.5. It uses a modified version of Uno that simply admits for the noise and layer size behavior described above, the neural network is unchanged.

We ran Uno alone as part of a sweep workflow as depicted in Figure 4. The results are illustrated as a heat map with the noise level percentage on the Y-axis and the layer size on the X-axis. Each data cell represents the validation loss (`val_loss`) reported by TensorFlow. The plot is colored so that red values are higher errors (worse quality results) and the blue values are lower errors (better quality results). The central white values are set to the median of the plotted data.

Each range of models was run for 10 epochs, which is a known value for reasonable Uno results (models typically converge after 7 epochs [32]).

As shown, the CCLE data has significantly lower median and maximum `val_loss` results. For both models, broadly speaking, results are generally better for lower noise settings, and higher layer sizes, as expected.

Noise in the labels is expected to cause the model to be unable to converge to the true patterns in the underlying data and some gradient steps in the training procedure will take the model weights in a wildly inaccurate direction.

Reducing the layer sizes used by Uno from the recommended setting of 1000 per dense layer also harms performance. This is also expected as the loss of neurons per layer reduces the available memory of the neural network and its capacity to pick up on patterns in the underlying data.

Some aspects of the individual models are clear. The CCLE-trained model appears to have unusually poor performance in the 900 column, but this is probably an anomaly. Both models start performing very poorly as the layer size is reduced below 550, and both seem to perform noticeably better as the layers increase in size from 950 to 1000, justifying the recommended value of 1000.

Holding the two model results side by side, it is difficult to visually identify a clear *comparative* pattern in the behavior of the models. It is unclear how to interpret in what scenarios the models differ greatly.

### C. Model performance comparison: Grid search

To be able to identify the difference between the models, we ran them in the CMP workflow over the same grid. The results returned were the CMP result from Equation 1. The result is plotted on the same axes as before in Figure 5 (a).

As shown in the plot, the relative error is higher in the high-noise, high-layer-size region of the top right. While this is somewhat a mathematical expectation that derives from Equation 1, it is interesting to note that this does holds more strongly for high noise than low noise, indicating that the models do differ in their robustness to noise. It is also clear visually that more extreme values are found in this plot than in the individual plots above, as visually noticeable by the abundance of stronger shades of both red and blue. This indicates that there is more to learn about the varying responses of the two models.

### D. Model performance comparison: GA search

To determine the ability of automated search to find interesting differences between the two models, we then ran the full CMP workflow on the problem. We ran it using the DEAP algorithm in the Supervisor `GA` (genetic algorithm) workflow.

To demonstrate that the `GA` approach allows for more limited resources, we ran it on a smaller system at Argonne, *Lambda7*. This is a single node with a 2 CPUs, each a 20-core, 40-thread Intel Xeon Gold 6246. The system has 8 NVIDIA Tesla V100 GPUs. The system has 1 TB RAM and a simple local NVRAM-based filesystem. This also demonstrates the portability of our workflow scripts, which were unmodified between Lambda7 and Polaris.

We configured DEAP to run in the bounding box described above for the same parameter space as the grid search scenarios. For each search variable, the mutation value $\sigma$ was set to 10. We started the run with a population size of 16, and limited the search to 5 generations. DEAP commonly produces fewer samples than the initial population size in generations after the first; in this run, it only produced 56 samples total. This contrasts with the over 100 samples required for the full search. We expect this would be even more pronounced in a study with more dimensions, or if finer-grained results are required.

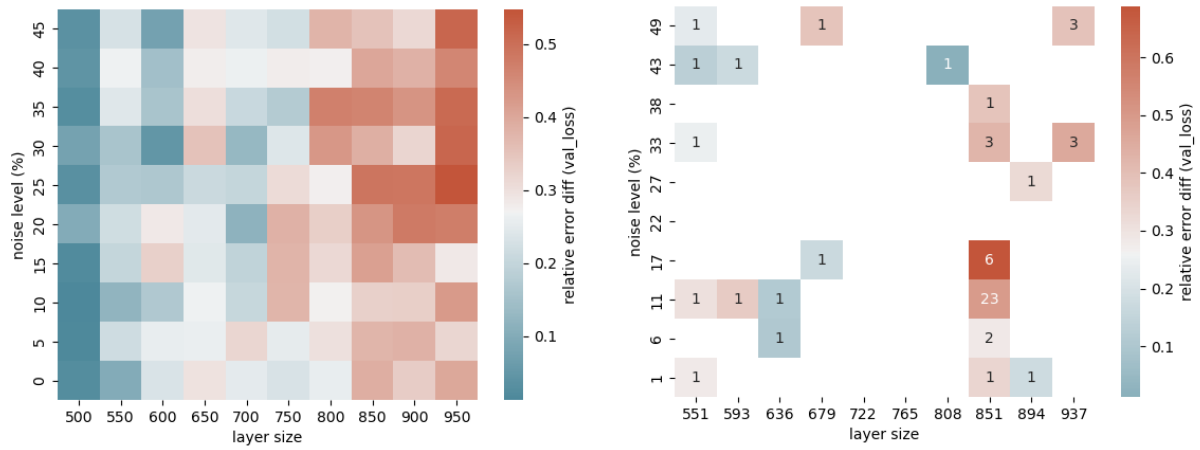
The results are shown in Figure 5 (b). The axes are the same as in previous runs, except that the labels are dynamically chosen based on bucketing the available data into deciles. Thus, the labels do not quite match prior runs, although the space is roughly equivalent. Some cells contain more than 1 sample run, in which case the results are averaged.

As shown, the algorithm clearly picks up on the pattern that the variation is on the right side of the plot. In total, only 9 samples were performed in the leftmost 3 columns, whereas 44 samples were performed in the rightmost 3 columns. In fact, 23 samples were performed in cell (851, 11) and 6 in the sample above it (851, 17), indicating that unusual variation is happening there. As shown in the color bar, the DEAP-based run also found a difference value higher than any value in the grid search, as the scale maxes out at well above 6, as opposed to 5 in the grid search case.

The performance statistics for DEAP are compared to that of grid search in Figure 6. In this plot, the 5 DEAP generations are compared to the fixed maximum relative difference found by grid search. Both the population mean and maximum are shown for DEAP. The maximal value found by grid search was 0.5479 and the maximal value found by DEAP was 0.7314, a 33% improvement. This supports the visual observation that DEAP is focusing on more interesting regions of the hyperparameter space both as a whole population and in the specific fine-grained targeting of the maximal differences.

## VIII. Conclusion

In this paper, we described the overall problem of cancer drug response prediction and the previously reported Uno model that is used to study this problem. We described the new efforts of the IMPROVE project to catalog and standardize a large range of community-developed cancer models for drug response prediction and other uses. There is a need to be able to compare these models for various reasons, including to gain insight into model behavior.




(a) Uno model performance differencing via grid search. For each noise and layer size, the colored data cell indicates the relative difference in validation loss between the Uno on the gCSI and CCLE datasets.

(b) Uno model performance search with DEAP. Colored data cells indicate the relative difference in validation loss between models, averaged across all samples in that cell. Counts indicate the number of samples performed per cell.

Fig. 5. Uno model performance as searched via grid and DEAP methods.

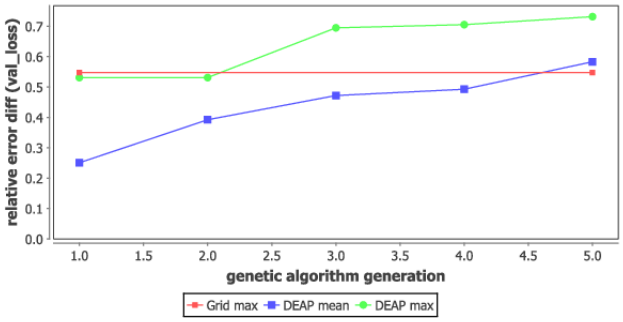



Fig. 6. DEAP population performance by generation. DEAP exceeds the most significant difference found by grid search after only 3 generations.

We focused on the problem of comparing two models under a range of hyperparameters and determing in which regions of the hyperparameter space they differ the most. By investigating scenarios such as noise in the training data or computational constraints such as layer size, models can be compared and subtle differences among them may become apparent. We propose that this can be useful to the developers and end users of deep learning models for cancer and other application areas.

In this paper, we introduced the CMP Model and an associated workflow based on a previously developed framework for hyperparameter optimization. The new model represents an innovative use of software architecture principles and demonstrates how a generalizable, scalable, and portable workflow framework can be adapted to new scientific uses. It also demonstrates the new capabilities of the framework to incorporate containers and run them on supercomputers like the Polaris system.

We presented results from grid search and automated search for two models that represent a plausible scientific investigation into model behavior in a reasonable range of scenarios. In the grid search, we showed that the two models behaved as expected in unfavorable scenarios, such as with noisy data or constraints on neural network size. It was also clear that although some patterns were clear, model comparison is visually difficult, and it would be difficult to obtain actionable results with a purely visual approach. We showed that automated search can quickly pick up on patterns in the model comparison space and find wider variations than grid search with fewer samples.

## IX. Future work

The project and features presented here have the capability to unlock a range of future work. The model curation effort of the IMPROVE project, not properly presented here, has the capability to greatly impact cancer research by cataloging and improving a large range of community cancer models. The project will allow model developers to learn from other models and improve their own models, while attempting to solve problems posed by other models and approaches and other data sets.

The automated search capabilities representing the main thrust of this paper will be further investigated in more realistic, higher-dimensional searches, on bigger data sets. We chose the smallest and fastest-training available datasets for Uno to demonstrate the workflow capabilities here, but other datasets have very long training times. The combination of longer training times, bigger datasets, bigger search spaces, larger genetic algorithm populations, and larger neural networks will pose exciting challenges for exascale computing.

## References


[1] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

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

[3] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[4] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A Matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[5] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," *CoRR*, vol. abs/1706.03292, 2017. [Online]. Available: http://arxiv.org/abs/1706.03292

[6] N. Systems, "Neon," https://github.com/NervanaSystems/neon, 2017, accessed: 2017-09-14.

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[8] F. Seide and A. Agarwal, "CNTK: Microsoft's open-source deep-learning toolkit," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 2135–2135. [Online]. Available: http://doi.acm.org/10.1145/2939672.2945397

[9] B. Van Essen, H. Kim, R. Pearce, K. Boakye, and B. Chen, "LBANN: Livermore Big Artificial Neural Network HPC Toolkit," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, ser. MLHPC '15. New York, NY, USA: ACM, 2015, pp. 5:1–5:6. [Online]. Available: http://doi.acm.org/10.1145/2834892.2834897

[10] G. Luo, "A review of automatic selection methods for machine learning algorithms and hyper-parameter values," *Netw. Model. Anal. Heal. Inf. Bioinf.*, vol. 5, no. 1, pp. 4–5, 2016. [Online]. Available: https://doi.org/10.1007/s13721-016-0125-6

[11] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 302–308, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220311693

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[13] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. of the 30th International Conference on Machine Learning*, 2013.

[14] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: a Python library for model selection and hyperparameter optimization," *Computational Science & Discovery*, vol. 8, no. 1, p. 014008, 2015. [Online]. Available: http://stacks.iop.org/1749-4699/8/i=1/a=014008

[15] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[16] CodeReclaimers, "NEAT-Python," https://github.com/CodeReclaimers/neat-python, 2017.

[17] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1437–1446.

[18] B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang, *mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions*, 2017. [Online]. Available: https://arxiv.org/abs/1703.03373

[19] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[20] F. Hutter, H. Hoos, and K. Leyton-Brown, "Parallel algorithm configuration," *Learning and Intelligent Optimization*, pp. 55–70, 2012.

[21] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[22] M. Claesen, J. Simm, D. Popovic, Y. Moreau, and B. D. Moor, "Easy hyperparameter search using Optunity," *CoRR*, vol. abs/1412.1114, 2014. [Online]. Available: http://arxiv.org/abs/1412.1114

[23] J. Kim and S. Yoo, "Software review: Deap (distributed evolutionary algorithm in python) library," *Genet Program Evolvable Mach*, vol. 20, no. 1, pp. 139–142, 2019. [Online]. Available: https://doi.org/10.1007/s10710-018-9341-4

[24] J. Ozik, N. Collier, J. M. Wozniak, and C. Spagnuolo, "From desktop to large-scale model exploration with Swift/T," in *Proc. Winter Simulation Conference*, 2016.

[25] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. Lusk, and I. T. Foster, "Swift/T: Scalable data flow programming for distributed-memory task-parallel applications," in *Proc. CCGrid*, 2013.

[26] T. G. Armstrong, J. M. Wozniak, M. Wilde, and I. T. Foster, "Compiler techniques for massively scalable implicit task parallelism," in *Proc. SC*, 2014.

[27] J. M. Wozniak, R. Jain, P. Balaprakash, J. Ozik, N. Collier, J. Bauer, F. Xia, T. Brettin, R. Stevens, J. Mohd-Yusof, C. G. Cardona, B. V. Essen, and M. Baughman, "CANDLE/Supervisor: A workflow framework for machine learning applied to cancer research," *BMC Bioinformatics*, vol. 19, no. 18, p. 491, 2018. [Online]. Available: https://doi.org/10.1186/s12859-018-2508-4

[28] F. Xia, J. E. Allen, P. Balaprakash, T. S. Brettin, C. Garcia-Cardona, A. Clyde, J. D. Cohn, J. H. Doroshow, X. Duan, V. Dubinkina, Y. A. Evrard, Y. J. Fan, J. Gans, S. He, P. Lu, S. Maslov, A. Partin, M. Shukla, E. A. Stahlberg, J. M. Wozniak, H. S. Yoo, G. F. Zaki, Y. Zhu, and R. Stevens, "A cross-study analysis of drug response prediction in cancer cell lines," *Briefings Bioinform.*, vol. 23, no. 1, 2022. [Online]. Available: https://doi.org/10.1093/bib/bbab356

[29] P. Balaprakash, R. Egele, M. Salim, S. Wild, V. Vishwanath, F. Xia, T. Brettin, and R. Stevens, "Scalable reinforcement-learning-based neural architecture search for cancer deep learning research," in *Proc. SC*, 2019.

[30] "CANDLE Organization on GitHub." [Online]. Available: https://github.com/ECP-CANDLE

[31] R. Jain, A. Shah, J. Mohd-Yusof, J. M. Wozniak, T. Brettin, F. Xia, and R. Stevens, "Probing decision boundaries in cancer data using noise injection and counterfactual analysis," in *Computational Approaches for Cancer Workshop @ SC*, 2021.

[32] J. M. Wozniak, H. Yoo, J. Mohd-Yusof, B. Nicolae, N. Collier, J. Ozik, T. Brettin, and R. Stevens, "High-bypass learning: Automated detection of tumor cells that significantly impact drug response," in *Proc. Machine Learning in High Performance Computing Environments (MLHPC) @ SC*, 2020.

[33] R. Jain, J. M. Wozniak, and J. Mohd-Yusof, "Supporting a community of cancer models with the CANDLE checkpoint module," in *Cancer Workshop poster @ SC*, 2022.

[34] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PLOS ONE*, vol. 12, no. 5, pp. 1–20, 05 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0177459

[35] S. Developers, "Singularity," 2021, Zenodo listing for Singularity and Apptainer. [Online]. Available: https://doi.org/10.5281/zenodo.1310023

[36] P. M. Haverty, E. Lin, J. Tan, Y. Yu, B. Lam, S. Lianoglou, R. M. Neve, S. Martin, J. Settleman, R. L. Yauch *et al.*, "Reproducible pharmacogenomic profiling of cancer cell line panels," *Nature*, vol. 533, no. 7603, pp. 333–337, 2016.

[37] J. Barretina, G. Caponigro, N. Stransky, K. Venkatesan, A. A. Margolin, S. Kim, C. J. Wilson, J. Lehár, G. V. Kryukov, D. Sonkin *et al.*, "The Cancer Cell Line Encyclopedia enables predictive modelling of anticancer drug sensitivity," *Nature*, vol. 483, no. 7391, pp. 603–607, 2012.

[38] "Polaris web site." [Online]. Available: https://www.alcf.anl.gov/polaris


## X. Acknowledgments


This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under contract number DE-AC02-06CH11357.