Composable Workflow for Accelerating Neural Architecture Search Using In Situ Analytics for Protein Classification

Georgia Channing
University of Tennessee
Knoxville, Tennessee, USA

Ria Patel
University of Tennessee
Knoxville, Tennessee, USA

Paula Olaya
University of Tennessee
Knoxville, Tennessee, USA

Ariel Keller Rorabaugh
University of Tennessee
Knoxville, Tennessee, USA

Osamu Miyashita
RIKEN
Kobe, Hyōgo, JP

Silvina Caino-Lores
University of Tennessee
Knoxville, Tennessee, USA

Catherine Schuman
University of Tennessee
Knoxville, Tennessee, USA

Florence Tama
Nagoya University and RIKEN
Nagoya, Aichi, JP

Michela Taufer
University of Tennessee
Knoxville, Tennessee, USA

ABSTRACT

Neural architecture search (NAS), which automates the design of neural network (NN) architectures for scientific datasets, requires significant computational resources and time — often on the order of days or weeks of GPU hours and training time. We design the Analytics for Neural Network (A4NN) workflow, a composable workflow that significantly reduces the time and resources required to design accurate and efficient NN architectures. We introduce a parametric fitness prediction strategy and distribute training across multiple accelerators to decrease the aggregated NN training time. A4NN rigorously record neural architecture histories, model states, and metadata to reproduce the search for near-optimal NNs. We demonstrate A4NN's ability to reduce training time and resource consumption on a dataset generated by an X-ray Free Electron Laser (XFEL) experiment simulation. When deploying A4NN, we decrease training time by up to 37% and epochs required by up to 38%.

CCS CONCEPTS

• **Computing methodologies** → *Genetic algorithms*; **Neural networks**; *Supervised learning by classification*; • **Applied computing** → **Physics**; **Chemistry**.

KEYWORDS

Neural architecture search, Neural networks, Protein diffraction, Predictive modeling, Deep learning, Early termination

ACM Reference Format:
Georgia Channing, Ria Patel, Paula Olaya, Ariel Keller Rorabaugh, Osamu Miyashita, Silvina Caino-Lores, Catherine Schuman, Florence Tama, and Michela Taufer. 2023. Composable Workflow for Accelerating Neural Architecture Search Using In Situ Analytics for Protein Classification. In *52nd International Conference on Parallel Processing (ICPP 2023), August 7–10, 2023, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3605573.3605636

1 INTRODUCTION

Neural networks (NN) are powerful models that are increasingly used in high-performance computing (HPC) simulations and new research areas such as high-performance artificial intelligence and high-throughput data analytics to solve problems in physics [6, 28, 32, 39], materials science [7, 38, 40, 46], neuroscience [5, 43], and medical imaging [11, 33], among other domains [12, 15, 30]. Finding suitable NNs for specific fitness measurements (e.g., accuracy or loss), datasets (e.g., data from different scientific domains), and problems (e.g., classification, segmentation, and regression) is a time-consuming process involving several rounds of hyperparameter and architecture selection, training, validation, and manual inspection. Given a fitness measurement, a dataset, and a problem, Neural Architecture Search (NAS) automates finding near-optimal models. Still, it comes at a high training cost involving thousands of NNs on many HPC resources. For instance, conventional NAS algorithms exhibit prohibitive computational demand where training each NN to convergence is the main bottleneck [3, 34]. A single NAS run on a small dataset of one or two petabytes can require HPC systems with thousands of accelerators [32]. A training run of a large language model such as BERT takes more than 80 hours on 16 TPUv3 AI accelerator hardware [44], and training a visual transformer requires decades of compute time on a TPUv3 [9]. For even larger datasets, scientists must allocate significant time to the largest compute resources available (in tens of thousands of GPU hours [32]) to conduct a single search of NN models. Furthermore, in the early stages of NAS, up to 88% of NNs fail to learn [14], further wasting compute resources.

Current attempts to optimize NAS remains inefficient. Built-in truncated training, a fixed termination criterion where each NN is trained for a set number of epochs, still wastes expensive HPC resources [10, 25, 26, 41]. Advanced NAS implementations use a fitness prediction strategy to terminate training dynamically. However, by embedding their prediction strategy in their search process, they produce a tightly-coupled solution, so any attempt to optimize the NAS approach interferes with its implementation [1, 8, 16, 20, 23, 24, 27, 36]. Furthermore, complete record trails of NNs through generation, training, and validation are rare; it is thus almost impossible to advance explainable and reproducible machine learning. Existing NN repositories store trained NNs and sometimes their final fully-trained fitness values (e.g., DLHub [4] and Model Zoo [35]) but do not capture the training lifespan in a shareable record trail. It is still an open problem to find general, effective solutions for high-throughput searches of near-optimal neural networks (NN) to capture embedded knowledge and predict hidden trends in scientific data. This project addresses the urgent need for solutions that reduce high-performance computing (HPC) resources for NNs' training while assuring more explainable, reproducible, and nearly optimal NNs.

To address this problem, we introduce the A4NN workflow to reduce training costs in NAS on HPC platforms and produce more explainable and reproducible NN results. Specifically, we define a methodology to (i) increase NN throughput in NAS via the predictions of NNs' fitness (e.g., accuracy or loss) early in the training process, thus enabling early, flexible training termination; (ii) design a workflow that decouples the search from the prediction of fitness for general NAS implementations across fitness measurements for real datasets in protein diffraction; and (iii) generate an NN data commons that share full provenance of the wide variety of NNs with record trails of their structure, metadata, and performance throughout training. We show that our workflow can decrease training time by up to 37% and the training epochs required by up to 38%. We demonstrate that using our workflow does not diminish the performance of the NAS. Finally, we compare the A4NN workflow's performance on the protein crystallography use case to the state-of-the-art for that use case.

The contributions of this paper are as follows:

- We design A4NN, a highly efficient, composable workflow that leverages existing NAS and in situ parametric predictions to classify protein conformations from protein diffraction image datasets.
- We rigorously record neural architecture histories, model states, and metadata to reproduce the search for near-optimal NNs.
- We demonstrate A4NN's ability to reduce training time and resource consumption on a dataset generated by simulating X-ray Free Electron Laser (XFEL) experiments.
- We discuss how the workflow can be deployed across a broader spectrum of NAS and datasets.

2 METHODOLOGY

We design A4NN as a modular and composable workflow that works in concert with existing NAS implementations. The core components of our workflow are a self-contained, externally-controllable parametric prediction engine for predicting NN fitness [18]; a workflow orchestrator handling the movement of data and metadata between the NAS implementation and the prediction engine; a lineage tracker collecting the evolution of NN architectures and their metadata; an analyzer for extracting knowledge from the collected data; a resource manager distributing computation across GPUs; and user and communication interfaces that shield the training process from the user. The path to training and validation datasets, the NAS, and the prediction engine settings are entirely customizable; they can be easily swapped out for other datasets, NAS implementations, and settings. Figure 1 presents the components of our A4NN workflow and their interactions with NAS and users.

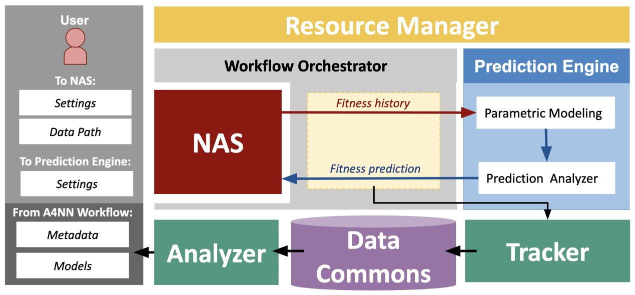


Figure 1: Main components of the A4NN workflow.

2.1 Parametric Prediction Engine

The parametric prediction engine augments the search for NNs in a NAS. It interacts with the NAS, running on dedicated resources and informing the search concurrently. In other words, the prediction engine dynamically predicts the fitness each NN could attain during the NN training phase and informs the NAS of each NN's predicted fitness. As the NAS trains NN from a specified search space, the prediction engine periodically pulls the partially trained NNs and iteratively executes a two-step process depicted in Figure 1. First, the engine constructs a parametric model for an NN's fitness curve. It uses the model to predict the fitness the NN is expected to attain at a given epoch (*Parametric modeling*). Then, it determines whether the prediction has converged (*Prediction analyzer*), deciding whether to terminate or continue the iterative execution of modeling and analysis.

2.1.1 Parametric Modeling. We leverage empirical observations to model an NN's fitness curve and predict the fitness value at a given epoch to terminate training early and reduce the length of the architecture search. We curate an adaptable parametric function that can predict an NN's accuracy. Parametric modeling has been successfully used in various machine learning problems to model curves and extrapolate predictions [42]. However, parametric modeling has not yet been applied to model the fitness learning curves of NNs. We apply parametric modeling to construct models for NN fitness learning curves. Fitness accuracy tends to increase as the NN learns; well-behaved learning curves tend to increase with a concave down shape, with the fitness increasing quickly at the beginning and more slowly as time continues [42]. We use a concave function in the form of ($\mathcal{F}(x) = a - b^{(c-x)}$) to extrapolate a candidate fitness prediction for an NN at a given epoch in the future. We attain the values for the function parameters using the least squares regression of the fitting. Specifically, we use regression to fit a parametric function to the learning curve data of the partially-trained NN results in a function that models NN fitness in terms of training epochs. The iterative nature of the training process in A4NN allows us to add one more fitness data point to the learning curve at each training step and compare the current candidate fitness prediction with previous ones to determine whether the candidate predictions have converged to a stable value. If so, this value is our fitness prediction for the NN.

2.1.2 Prediction Analyzer. The prediction analyzer determines whether fitness predictions converge to a stable value. If so, the analyzer outputs the fitness prediction that the NAS treats as the final fitness of the network in its selection process, and the prediction engine's iterative execution terminates the NN's training. If not, the next iteration begins, and the NN continues to be trained. To this end, the prediction analyzer first checks that the most recent predicted fitnesses are valid fitness values to determine if the fitness predictions have converged to a stable value. Our engine uses validation accuracy as fitness. Thus, the predicted value can neither be greater than 100 nor less than 0. If any of the most recent predicted fitnesses are outside the bounds of possible values, the prediction analyzer returns that it has not converged on a stable value. Figure 2 depicts the predictive engine's behavior on an example NN from our results. Here, we use the parametric function $\mathcal{F}(x) = a - b^{c-x}$ to model the NN's learning curves, where x represents the epoch for which we predict fitness, e_{pred}. The resulting prediction of the fitness at 25 epochs converges at epoch 12, and training should then be terminated.

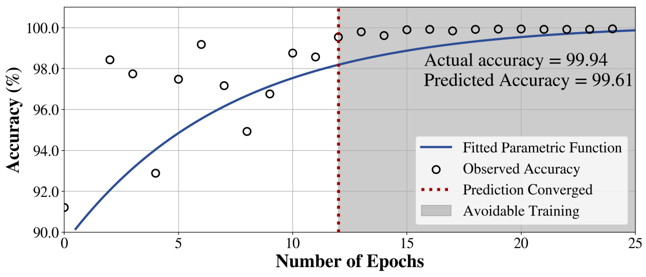


Figure 2: Example of the fitness (accuracy) prediction for an NN trained on our use case using the function $\mathcal{F}(x) = a - b^{(c-x)}$.

2.2 Workflow Orchestrator

A4NN orchestrates the interactions between the prediction engine and NAS as well as the management of the NN's state history (i.e., fitness and predictions) across epochs, automating these processes for the user.

2.2.1 Orchestration Between Prediction Engine and NAS. Rather than re-engineering the NAS and its mechanisms for early termination, A4NN relies on a plug-in to manage communication between the prediction engine and the NAS during the training process and to orchestrate decision-making iteratively. Algorithm 1 describes the training loop. After each training epoch e for an NN M in the NAS, the fitness history ($\mathcal{H}$), which is comprised of all previous fitness values, is fed to the `predictor` (Fitness history in Figure 1). If the prediction engine can predict epoch e (p_e), this prediction is appended the prediction history ($\mathcal{P}$), which is comprised of all previous predictions. If the prediction analyzer has converged on a stable prediction, then the engine communicates to the NAS the final prediction (Fitness prediction in Figure 1), terminating both the engine's iterative execution and the NAS training.

Algorithm 1 Training Loop with Prediction Engine.

```
1:  pred_eng ← pred_eng($e_{pred}$, $\mathcal{F}$, $C_{min}$, $r$)
2:  $converged$ ← $False$
3:  for $e \in num\_epochs$ do
4:      $M.train()$
5:      $h_e$ ← $M.validate()$
6:      $\mathcal{H}$ ← $h_e$
7:      $p_e$ ← pred_eng.predictor($e$, $\mathcal{H}$)
8:      $\mathcal{P}$ ← $p_e$
9:      $converged$ ← pred_eng.analyzer($\mathcal{P}$)
10:     if $converged$ then
11:         break
12:     else
13:         continue
14:     end if
15: end for
16:
17: if $converged$ then
18:     return $\mathcal{P}[-1]$
19: else
20:     return $h_e$
21: end if
```

2.2.2 Orchestration of A4NN History. The NAS and the A4NN engine share the fitness and prediction history, optimizing the memory usage in the training loop. At the end of each training epoch, the workflow orchestrator writes the partially trained NN's state to memory, such that each model can be loaded and re-evaluated from any point in the training phase. When an NN's training is terminated, the workflow orchestrator writes the fitness history, prediction history, training times, FLOPS, and architecture information to storage, such that a model's entire training history can be further analyzed. The write location for model and metadata files is configured as a command-line argument to the NAS.

2.3 Lineage Tracker and Data Commons

With the lineage tracker, we record the NN's state history (i.e., architecture, metadata, and performance) for the training span of our curated collection of NNs. Specifically, the tracker captures the arc of an NN architecture's optimization together with its behavior throughout training (i.e., the training parameters such as learning rate and batch size, the criterion used for gain or loss, and the method used for training and the measurement used for fitness). The tracker uploads the record trails in an NN data commons that maintains comprehensive NN record trails, enabling reproducible and explainable machine learning. The data commons provides information about epoch times, training accuracies, validation accuracies, FLOPS, predictions, prediction engine parameters, genomes, and architecture information for each neural architecture. We use the lineage tracking information collected from the tests presented in this paper to create an open-access NN data commons in Harvard Dataverse [19], a public repository for research data. We add complete metadata to leverage the repository's built-in capabilities. Our commons comes with a Python script demonstrating how to load the data into a Pandas DataFrame and calculate and save metrics of interest such as mean accuracy or NN learning rate.

2.4 Analyzer

Users can deploy our fitness models and data commons through a Jupyter Notebook interface that serves as an Analyzer. Specifically, scientists can use the A4NN Jupyter Notebooks to study NN performance and evolution throughout training, the shape of fitness curves, and the relationship between the architecture and performance. The Jupyter Notebook enables potential users to search for NNs with specific attributes. It serves as an interactive visualization tool, allowing users to view learning curve shapes, visualize the structure of NNs, and find NNs of interest. The Jupyter Notebook supports visualization techniques for understanding the structure and function of NNs designed with our workflow. Figure 3 illustrates the type of supported visualization for NN structures.

2.5 Workflow Resource Manager

Our workflow's decoupled nature and its components' modularity assure scalable, high-performance NN training. The decoupling enables optimizations of the resource allocation for each workflow component, the components' orchestration and placement, and the mechanisms to exchange data efficiently among the components. We leverage the scheduling algorithms of Ray [22] and use its first in, first out (FIFO) dynamic scheduling to assign models to GPUs within a generation. When an NN finishes training, another NN within the generation begins training according to GPU availability. As a result of Ray's algorithm, GPU downtime can be accumulated as the number of networks within each generation may not be divisible by the number of available GPUs. Therefore, at the end of each generation's evaluation, some downtime may occur when not all GPUs are used.

2.6 User Interfaces

The interaction between users and the A4NN components is managed by a Jupyter Notebook interface that controls the NAS settings, the path to the training and validation data, and prediction engine settings.

2.6.1 NAS Settings. Configuring the selected NAS implementation is independent of the A4NN workflow. In other words, augmenting a NAS with A4NN does not change the user's interaction with NAS. In this work, we use NSGA-Net. Users submit the NSGA-Net parameters through command-line arguments to the driver script that instantiates the NAS run. These parameters include the size of the starting population, the number of nodes per phase, the number of offspring per generation, the number of generations, and the number of epochs to train.

2.6.2 Data Path. Changing the input dataset used in the A4NN workflow is a straightforward operation and can be done by changing the data path passed to the NAS. Most NAS approaches require the customization of data loaders that structure scientific data for neural network training (e.g., splitting for training and testing sets). The A4NN workflow does not add steps to NAS's data loaders.

2.6.3 Prediction Engine Settings. We configure the prediction engine's parameters by changing the engine's object instantiation. These parameters include the parametric function ($\mathcal{F}$), the minimum number of points to make a prediction (C_{min}), the epoch for which fitness is predicted (e_{pred}), the number of predictions to consider when terminating training (N), and the allowed variance in predictions (r).

3 SCIENTIFIC DATA AND NAS SELECTION

Our workflow allows users to customize the scientific data and the NAS. Here we use a real dataset representing protein diffraction images from the protein crystallography domain as the dataset of interest. We plug NSGA-Net into our workflow.

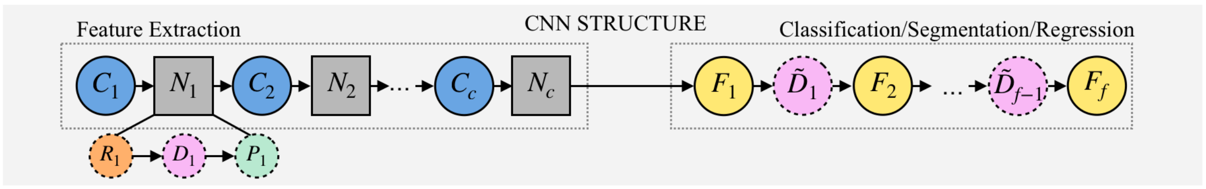
3.1 Protein XFEL Diffraction Dataset

We evaluate A4NN with an X-ray Free Electron Laser (XFEL) protein dataset. We select this dataset for comparison reasons as it has been previously studied with a traditional machine learning approach [29]. In this study, we generate, train, and evaluate NNs to classify protein conformations from protein diffraction patterns generated through simulations of XFEL experiments. A protein conformation refers to the shape adopted by the protein and is caused by the rotation of the protein atoms around one or more single bonds. A4NN generates the NNs that differentiate and classify the different conformations.

In XFEL experiments, proteins are shot with a laser beam. The resulting photon scattering from the protein is recorded as a diffraction pattern. The resulting diffraction patterns are unique images of a given protein, similar to human fingerprints, and encode information about the 3-dimensional structure in 2-dimensional space. Figure 4 shows an overview of an X-ray Free Electron Laser (XFEL) experiment. In this work, we use simulated diffraction patterns. We rely on the *spsim* simulator to generate different diffraction patterns for two conformations of EF2 with PDB ID 1n0u and 1n0v (Conf. A and Conf. B in Figure 4) from the Protein Data Bank. We use *Xmipp* to simulate the different beam orientations. The simulated images match the experimental ones but come with the additional information on the protein's angles needed for our validation. The XFEL beam's intensity directly affects the resultant images' signal-to-noise ratio and low beam intensities are a proxy for noise. The lower the intensity, the higher the noise. The simulator reproduces the outputs of a laser beam pulse measuring different beam intensities. Figures 5a, 5b, and 5 show the resolutions produced by varying intensities of the XFEL beam on the same protein, ranging from low (1×10^{14} photons/μm^2/pulse) to medium (1×10^{15} photons/μm^2/pulse) and high (1×10^{16} photons/μm^2/pulse).

3.2 NAS and NSGA-Net.

We assessed several NAS implementations, including EvoCNN [41], MENNDL [45], NASNet [47], Auto-Keras [13], NSGA-Net [26] to select the NAS we plug into our workflow for our demonstration. The selection criteria were as follows: the NAS must be open-source, easily customizable, and support useful metadata such as model architecture information and performance metrics (i.e., FLOPS). We use NSGA-Net as the NAS in the A4NN evaluation because it allows us to use multi-objective NAS to optimize fitness while minimizing


Feature Extraction
CNN STRUCTURE
Classification/Segmentation/Regression
C_1 N_1 C_2 N_2 … C_c N_c
R_1 D_1 P_1
F_1 $\tilde{D}_1$ F_2 … $\tilde{D}_{f-1}$ F_f

C_i Convolutional layers, $1 \leq i \leq c$ where $c \leq 10$ is the total number of convolutional layers.

N_i Non linear layer blocks. After each convolution, at least 1 non linear layer is chosen from ReLU, dropout, and pooling.

R_i ReLU layer. At most 1 after convolution C_i

D_i Dropout layer. At most 1 after convolution C_i

P_i Pooling layer. At most 1 after convolution C_i

F_j Fully connected layers, $1 \leq j \leq f$ where $f \leq 5$ is the total number of fully connected layers.

$\tilde{D}_j$ Dropout layer. At most 1 after fully connected layer F_j for $j < f$

Figure 3: Structural representation of generic NN provided by the Analyzer in A4NN.

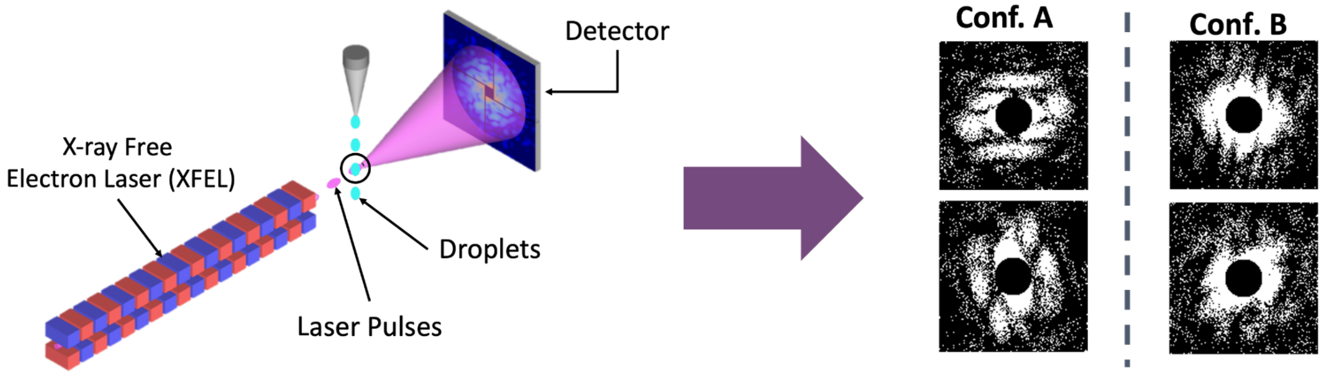
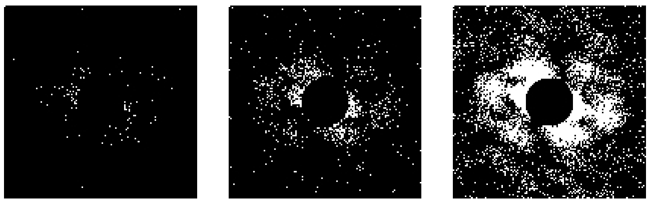


Figure 4: Overview of an X-ray Free Electron Laser (XFEL) experiment generating two different sets of patterns for two conformations of the same eEF2 protein.



(a) Low **(b) Medium** **(c) High**

Figure 5: Simulated beam intensities for EF2 protein: (a) Low, 1×10^{14} **photons/**μm^2**/pulse; (b) Medium,** 1×10^{15} **photons/**μm^2**/pulse; and (c) High:** 1×10^{16} **photons/**μm^2**/pulse**

FLOPS usage. By minimizing FLOPS usage, we encourage the development of models that can be executed on CPUs with relatively low energy consumption.

The NSGA-Net evolutionary algorithm is as follows. It generates genomes that encode architecture information like layer types within the specified macro search space; initializes the first generation's population of architectures; begins training and validating the first generation; and uses mutation and crossover techniques to create offspring from the top two NNs from the first generation, determined by tournament selection. In our comparison of NSGA-Net alone versus NSGA-Net integrated into A4NN, we train and test models with balanced conformation classes for each beam intensity using an 80/20 train-test split of 63,508 images and 15,876 images, respectively.

4 EVALUATION

We evaluate A4NN across metrics such as validation accuracy, FLOPS, the number of epochs saved, and wall time for training. Validation accuracy measures the fitness of a neural architecture; thus, the higher, the better. FLOPS is a proxy for energy consumed by a neural architecture; thus, the lower, the better. The number of completed training epochs is a proxy for the number of GPU hours and amount of energy consumed by the NAS; thus, the lower, the better. Last, wall time represents the time a scientist waits for a successful model; thus, the lower, the better. Furthermore, we compare A4NN to state-of-the-art methods for classifying protein conformations from XFEL datasets. In our evaluation process, we answer four critical questions: *"What is the performance and accuracy impact* *of our augmented search compared to using the standalone NAS?"*, *"How does our search scale?"*, *"How does the A4NN workflow compare to the state-of-the-art for the protein diffraction datasets?"*, and *"How do we maintain data provenance in the A4NN workflow?"*

4.1 Evaluation Parameters

To compare A4NN to the standalone NSGA-Net, we hold constant both A4NN and NSGA-Net's parameters across our tests. We configure the prediction engine in A4NN with user input as listed in Table 1. We use the function $\mathcal{F}(x) = a - b^{c-x}$ to model each NN's fitness and configure the prediction engine to require three predictions ($N = 3$) within a variance threshold, r, of 0.5 to converge.

Table 1: Prediction Engine Configuration

Variable	Setting	Description
$\mathcal{F}$	$\mathcal{F}(x) = a - b^{c-x}$	parametric function for fitness modeling
C_{min}	3	minimum number of epochs before making a prediction
e_{pred}	25	epoch for which to predict final fitness
N	3	number of predictions to consider when converging
r	0.5	variance of prediction to tolerate in convergence

We configure NSGA-Net as listed in Table 2. We direct NSGA-Net to begin with a population of 10 neural networks, produce 10 offspring per generation, evolve for 10 generations, and train each network for 25 epochs. Each test produces 100 networks in total.

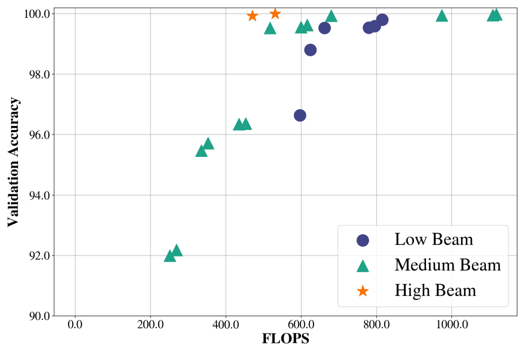
Table 2: NSGA-Net Configuration

Setting	Value
size of starting population	10
number of nodes per phase	4
number of offspring per generation	10
number of generations	10
number of epochs to train	25

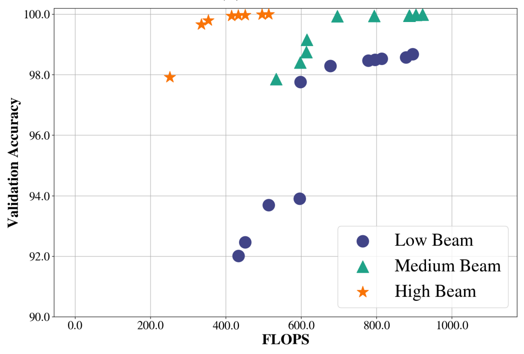
4.2 A4NN vs Standalone NAS

To quantify the impact of augmenting NSGA-Net with A4NN, we compare the performance of A4NN with the performance of standalone NSGA-Net for each of the three beam intensities in terms of accuracy versus FLOPS, epochs saved, and wall time.

4.2.1 Accuracy versus FLOPS. We generate Pareto frontiers from 100 architectures designed in each test for maximized validation accuracy and minimized FLOPS usage. Figure 6a shows the validation accuracy and the FLOPS of the Pareto optimal models generated by A4NN. Figure 6b shows the same metrics for the Pareto optimal models generated by the standalone NAS. Each symbol represents a Pareto optimal model, and the shape of each symbol demarcates the beam intensity (i.e., low, medium, or high). Each beam intensity results in a different number of Pareto optimal models. The tests are performed on a single GPU.



(a) A4NN



(b) Standalone NAS

Figure 6: Validation accuracy and FLOPS of the Pareto optimal models for A4NN and NSGA-Net for the three beam intensities.

We search for models with high validation accuracy and low FLOPS for this comparison. For low beam intensity tests, A4NN generates models that achieve accuracy that reach 99.8% using below 650 FLOPS versus the 98.1% accuracy achieved with a similar number of FLOPS by standalone NSGA-Net. In the medium beam intensity tests, A4NN generates models that outperform the models generated by the standalone NAS. Specifically, A4NN achieves near 100% validation accuracy, while standalone NSGA-Net achieves under 99% accuracy for the same FLOPS utilization. For high beam intensity tests, A4NN achieves similar accuracy and FLOPS to NSGA-Net at 99.9% and 450 FLOPS. As high beam intensity reflects low noise in training and validation images, it is unsurprising that both methodologies can achieve high accuracy and low FLOPS. However, A4NN delivers additional time and resource savings as it matches the validation accuracy and FLOPS of NSGA-Net standalone. We demonstrate this benefit in Figures 7 and 9.

4.2.2 Epoch Savings. We measure the number of training epochs required by A4NN and standalone NSGA-Net to evaluate 100 neural architectures. For the tests using A4NN, we measure the number of epochs using one and four GPUs. NSGA-Net does not support multiple GPUs. Thus, the standalone NAS tests are executed on a single GPU. In Figure 7, the bars show the number of training epochs required by each test for each of the three beam intensities. Without A4NN, NSGA-Net trains all models for 25 epochs by default. As we generate 100 models per test, the total number of epochs trained for each beam intensity by standalone NSGA-Net is 2,500. In the figure, we group results by beam intensity and measure the percentage of saved epochs relative to the standalone NSGA-Net baseline of 2,500 epochs. Using A4NN, we reduce the required training epochs by 13.3%, 34.1%, and 30.5% on the low, medium, and high beam intensity data, respectively. As the number of training epochs is a proxy for energy consumption and GPU hours, we significantly increase the energy efficiency of NSGA-Net with the A4NN workflow.

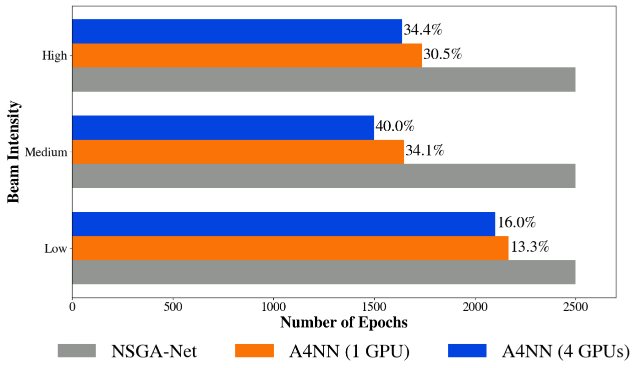


Figure 7: Epochs required for testing 100 architectures and percentages of saved epochs with A4NN over the standalone NAS when using one and four GPUs.

We evaluate the effects of the different beam intensities on the models' convergence and measure the percentage of converged models. The lower the number of epochs and the higher the percentage, the better. Figure 8 shows the distribution of e_t, the epoch at which training is terminated because an NN's fitness predictions have converged to a stable value. The legend describes the percentage of neural architectures in which training was terminated. The figure shows only results for A4NN since all models were generated by a standalone NSGA-Net train for all 25 epochs. For the low beam intensity tests, we observe that the average e_t is greater than 18. Though more than 60% of the models had training terminated early, only a few epochs of training per model were saved. The average e_t is under 12.5, or half of the full training period for the medium beam intensity tests. Figure 8 also shows that the prediction engine can terminate training early for more than 70% of medium beam intensity models. For the high beam intensity tests, we observe an almost inverted bell curve in the distribution of e_t. Figure 8 shows that, for high beam models with training terminated early, training was terminated early in the training phase with an average e_t of 10. However, only 55% of the high beam models have training terminated early, while some high beam models forgo more than half of their training, and almost as many must be trained for the entire training period. We also observe a trend first observed and discussed in Figure 7: the training across multiple GPUs converges faster than on a single GPU.

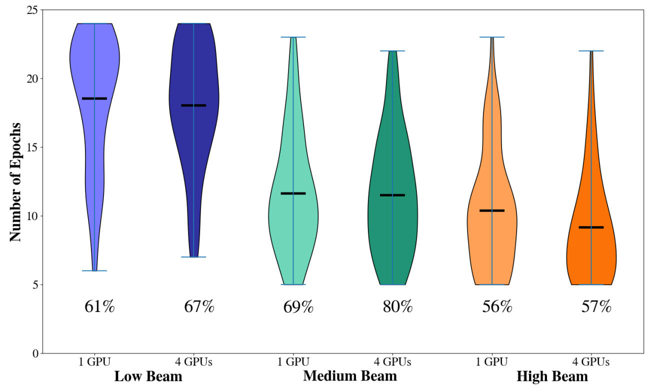


Figure 8: Effect of the different beam intensities on the convergence of the model's predictions in terms of the number of epochs and the percentage of the converged models.

4.3 Compute Performance and Scalability

We study the compute performance in terms of wall time savings and scalability as a ratio of the measured training times. While wall times may be indirectly related to the number of epochs, they do not necessarily decrease linearly with the number of epochs because the length of each epoch may vary from iteration to iteration and from dataset to dataset.

4.3.1 Wall Time Savings. To minimize the time a scientist must wait for accurate NNs to be designed and evaluated, we measure the wall times of tests with A4NN and standalone NSGA-Net. Figure 9 compares the wall times required by A4NN and standalone NAS. We group results by beam intensity and present results of tests using A4NN with one and 4 GPUS. On the low beam intensity data, on which the prediction engine only conserves 16.0% of training epochs, we observe a wall time decrease of 3.5 hours. We observe greater wall time savings of 15.8 and 16.3 hours on the medium and high beam intensity data due to greater epoch savings. Comparing Figures 7, 8, and 9, we also observe the impact of varying distributions of e_t on wall time savings. Despite a greater percentage of medium beam intensity models terminating training early, as shown in Figure 8, we observe greater time savings for the high beam intensity data, as shown in Figure 9. This results from the different distributions of e_t between the medium and high beam intensity data. The overhead cost of the A4NN prediction engine is negligible compared to its savings. Over the course of a test in which 100 models are evaluated, using the A4NN prediction engine adds an average of 52.16 seconds to the total wall time. Each interaction with the prediction engine, as depicted in Algorithm 1, takes an average of 28.07 milliseconds. The variance of the overhead per epoch is 1.12 milliseconds.

4.3.2 Scalability. We study the overall scalability of A4NN in terms of saved epochs and wall clock times on a single GPU against four GPUs. In Figure 9, we do not observe a near-linear epoch speed-up:

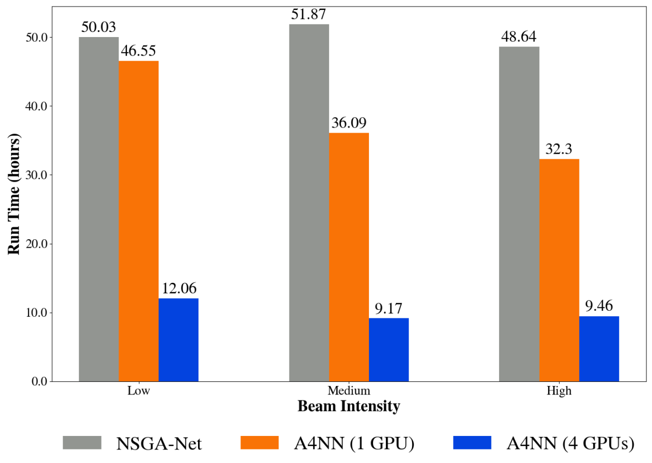


Figure 9: Wall times required by training NNs with A4NN and standalone NAS for the three beam intensities using one GPU (for the A4NN and NSGA-Net) and for four GPUs (for A4NN).

we observe 1.17x, 1.13x, and 1.2x epoch savings when moving from a single GPU to four GPUs. On the other hand, in Figure 9, the training distribution across multiple GPUs results in near-linear wall time speed-ups. Compared to the A4NN workflow's performance on a single GPU, we observe a 3.8x, 3.9x, and 3.4x wall time speed-up on the low, medium, and high beam intensity images distributed across the four GPUs. Furthermore, we observe slight accuracy improvements with models generated by the distributed workflow compared to those generated on a single GPU. We hypothesize that this phenomenon may result from the balance of breadth and depth of testing when distributed. Additional tests are in progress to fully reason about these observations.

4.4 Comparison with State-of-the-Art Methods

For the classification of protein conformations from protein diffraction image datasets, the state-of-the-art solution is the X-ray Free Electron Laser-based Protein Structure Identifier (XPSI) framework [29]. The XPSI framework is trained on a single NVIDIA-V100 GPU in 15 hours and 27 minutes. It achieved validation accuracy of 92%, 99%, and 100% on the low, medium, and high beam intensity images, respectively. As shown in Table 3, models produced

Table 3: Wall time and accuracy of A4NN versus XPSI for the three beam intensity on a single GPU.

Beam	Metric	**A4NN**	**XPSI**
Low	Wall Time	46.55 h	15.45 h
	Accuracy	97.8%	92%
Medium	Wall Time	36.09 h	15.45 h
	Accuracy	99.9%	99%
High	Wall Time	32.3 h	15.45 h
	Accuracy	100.0%	100%

by A4NN can match or improve on validation accuracy offered by XPSI. For example, our experiments delivered models can classify conformations with low, medium, and high beam intensity data at validation accuracy of 97.9%, 99.9% for low and medium beam intensities contrary to the 92%, 99% of XPSI, showing how our workflow is providing a more robust solution for noisy images. Note that the beam intensity is a proxy for embedded noise. When executed on just one GPU, the A4NN workflow cannot compete with the training time offered by XPSI. Where the XPSI framework requires only 15 hours and 27 minutes of training for each beam intensity, A4NN requires 46 hours and 33 minutes, 36 hours and 5.5 minutes, and 32 hours and 18.3 minutes to train models for the low, medium, and high beam intensity data, respectively. However, A4NN offers two significant advantages. First, it is composable and thus allows users to customize the NAS and the datasets deployed. Second, by distributing the workflow's training across four GPUs, the A4NN workflow delivers scalable training that the rigid XPSI framework cannot support. Using multiple GPUs and our prediction engine, the A4NN workflow can deliver models for the low, medium, and high beam intensity datasets in 12 hours and 3.8 minutes, 9 hours and 10 minutes, and 9 hours and 27.6 minutes, respectively.

4.5 Data Lineage

A4NN contributes robust data lineage information to Dataverse. As a result of our tests, we generated 54 GB of accessible and documented data in Dataverse. We save models after every training epoch for each test conducted as retrievable and deployable `torch.packages`. In total, we deliver 25,790 models, on which further research can be conducted. In our data commons, we also include training metadata files (i.e., epoch times, training accuracy, validation accuracy, FLOPS, predictions, engine parameters, genomes, and architecture information) for each neural architecture. Figure 10 shows an example of NN architectures that our analyzer can visualize. The figure presents the architecture of the NN Model 51 using the structural representation from Figure 3. NN Model 51 is one of the near-optimal NNs identified by A4NN for low beam intensive diffraction images.

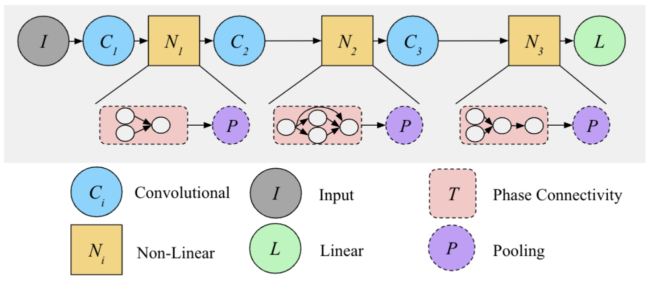


Figure 10: NN Model 51, one of the near-optimal NNs identified by A4NN for low beam intensity images.

5 RELATED WORKS

This work builds upon the previous works of Olaya et al. [29], Patel et al. [31], and Rorabaugh et al. [18]. We also take inspiration from previous works that use NAS for scientific datasets, such as Kandasamy et al. [17] and Balaprakash et al. [2]. From work that introduces other prediction strategies for NAS, such as Li et al. [21], Domhan et al. [8], and Ru et al. [37]. In the previous work of Olaya et al. [29], they created an open-source XPSI framework that utilizes a traditional machine learning approach to identify structural properties from 2D protein diffraction patterns to assist in the 3D reconstruction and understanding of the protein's structure. With the use of k-Nearest Neighbors (kNN) and autoencoders for feature extraction, their framework's workflow predicts different protein types, conformations, and protein orientations quickly and efficiently. However, getting to their final model required hundreds of hours of intervention by machine learning specialists to tune and achieve their most optimal model for this specific dataset. Their workflow also did not seek to minimize FLOPS usage.

Previous work by Patel et al. [31] further built upon the XPSI framework by introducing the use of a modified version of the Non-dominated Sorting Genetic Algorithm for the Neural Network Architecture Search (NSGA-Net) algorithm as presented by Lu et al. [26] to perform classification on the protein diffraction dataset. Patel el al. [31] demonstrated the promise of NSGA-Net on this dataset by generating a rather large pool of NNs and identifying a couple of models that best achieved the goals of high accuracy and energy efficiency. Not unlike [2]'s work, this methodology did not require manual tuning to be performed on the models individually, making this methodology beneficial to domain scientists who either do not have the expertise to code ML models for their datasets or have access to HPC machines. However, the NAS runtime was highly time-consuming and not distributed. Rorabaugh et al. [18] proposed a fitness prediction engine called *PENGUIN* that informs the NAS search. It decouples the search and prediction strategies of the NAS to allow for flexible and composable workflows that utilize parametric prediction modeling for highly efficient training of each model. Their engine's effects were simulated on MENNDL [45] with CIFAR-10, CIFAR-100, and SHVN datasets. These simulations demonstrated *PENGUIN*'s ability to reduce training time and costs and increase throughput significantly. Other uses of NAS on scientific datasets can be seen in works such as Deep Emulator Network SEarch (DENSE) [17]. The Deep Emulator Network SEarch (DENSE) generates emulators for different scientific simulation applications [17]. NAS frameworks have also been used to perform predictive modeling for cancer applications across large HPC machines with a single Python library module [2]. However, these NAS frameworks consume lots of time and compute resources that some domain scientists cannot access. Some studies are working to improve the performance estimation of NAS to help those with limited computing budgets. They aim to cut down the time it takes for the NAS to complete the generation of models and save on carbon emissions from each run. There are estimator methods that are based on different techniques, such as early-stopping during training [21] and learning curve extrapolation [8]. Other methods utilize training speed estimation measurements to estimate the final test performance of models [37] to prove the connection between training speed and generalization. In this work, we respond to work in this scientific domain with an enhanced NAS workflow to decrease wall times and resource consumption.

6 CONCLUSIONS

This paper demonstrates the superior performance of the A4NN workflow over state-of-the-art machine learning methods and standalone NAS implementations for classifying protein conformations from protein diffraction image datasets. In contrast to traditional machine learning methods, we propose a modular, composable, and distributed workflow that can be generalized to other datasets and NAS implementations than NSGA-Net and the protein diffraction data used in this work. Compared to a standalone NAS such as NSGA-Net, using A4NN reduces training epochs by up to 38% and training times by up to 37%. We deliver high accuracy of 97%, 99%, and 100% across low-resolution, medium-resolution, and high-resolution protein diffraction images. Furthermore, we deliver 54 GB of models and metadata in an open-access Dataverse commons for the community to study the behavior of neural networks in training. Data in the Dataverse commons can also be visualized using the analyzer in A4NN to develop an intuitive understanding of the qualities of successful NNs and support the reproducible explainability of NN evolutions.

A4NN can be used to study the behavior of NN architecture lineage and patterns, answering questions such as "Are there structural similarities between successful architecture produced by NAS?", "How can we visualize diverse neural architectures to identify patterns in successful architectures?", "Which parametric functions are best able to predict neural architecture fitness?" and "Is there a significant correlation between high FLOPS and high validation accuracy?"

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation (NSF) under grant numbers 1841758 and 2223704, and IBM through a Shared University Research Award. This work was additionally supported in part by the FOCUS Establishing Supercomputing Center of Excellence Project.

REFERENCES

[1] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. 2017. Accelerating Neural Architecture Search Using Performance Prediction. In *Proceedings of the NIPS Workshop on Meta-Learning*. arXiv:1705.10823 [cs.LG]

[2] Prasanna Balaprakash, Romain Egele, Misha Salim, Stefan Wild, Venkatram Vishwanath, Fangfang Xia, Tom Brettin, and Rick Stevens. 2019. Scalable reinforcement-learning-based neural architecture search for cancer deep learning research. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[3] Han Cai, Ligeng Zhu, and Song Han. 2018. Proxylessnas: Direct Neural Architecture Search on Target Task and Hardware. *arXiv preprint arXiv:1812.00332* (2018).

[4] Ryan Chard, Zhuozhao Li, Kyle Chard, Logan Ward, Yadu Babuji, Anna Woodard, Steven Tuecke, Ben Blaiszik, Michael Franklin, and Ian Foster. 2019. DLHub: Model and Data Serving for Science. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 283–292.

[5] Shanyu Chen, Zhipeng He, Xinyin Han, Xiaoyu He, Ruilin Li, Haidong Zhu, Dan Zhao, Chuangchuang Dai, Yu Zhang, Zhonghua Lu, Xuebin Chi, and Beifang Niu. 2019. How Big Data and High-performance Computing Drive Brain Science. *Genomics, Proteomics & Bioinformatics* 17, 4 (2019), 381–392. Big Data in Brain Science.

[6] Anshul Choudhary, John F. Lindner, Elliott G. Holliday, Scott T. Miller, Sudeshna Sinha, and William L. Ditto. 2020. Physics-enhanced Neural Networks Learn Order and Chaos. *Phys. Rev. E* 101 (Jun 2020), 062207. Issue 6. https://link.aps.org/doi/10.1103/PhysRevE.101.062207

[7] Juan-Pablo Correa-Baena, Kedar Hippalgaonkar, Jeroen van Duren, Shaffiq Jaffer, Vijay R. Chandrasekhar, Vladan Stevanovic, Cyrus Wadia, Supratik Guha, and Tonio Buonassisi. 2018. Accelerating Materials Development via Automation,

Machine Learning, and High-Performance Computing. *Joule* 2, 8 (2018), 1410–1420.

[8] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. 2015. Speeding up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves. In *Proceedings of the 24th International Conference on Artificial Intelligence* (Buenos Aires, Argentina) *(IJCAI'15)*. AAAI Press, 3460–3468.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, and Sylvain Gelly. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv preprint arXiv:2010.11929* (2020).

[10] Francisco Erivaldo Fernandes Junior and Gary G. Yen. 2019. Particle Swarm Optimization of Deep Neural Networks Architectures for Image Classification. *Swarm and Evolutionary Computation* 49 (2019), 62–74.

[11] A. Gertych, Z. Swiderska-Chadaj, and Z Ma. 2019. Convolutional Neural Networks Can Accurately Distinguish Four Histologic Growth Patterns of Lung Adenocarcinoma in Digital Slides. *Scientific Reports* 9, 1483 (2019).

[12] Jie Hou, Badri Adhikari, and Cheng Jianlin. 2018. DeepSF: Deep Convolutional Neural Network for Mapping Protein Sequences to Folds. *Bioinformatics* 34:8 (2018), 1295–1303.

[13] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-keras: An efficient neural architecture search system. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2019).

[14] Travis Johnston, Steven R. Young, David Hughes, Robert M. Patton, and Devin White. 2017. Optimizing Convolutional Neural Networks for Cloud Detection. In *Proceedings of the Machine Learning on HPC Environments (MLHPC)* (Denver, CO, USA). Article 4, 9 pages.

[15] A. Kamilaris and F. Prenafeta-Boldú. 2018. A Review of the Use of Convolutional Neural Networks in Agriculture. *Journal of Agricultural Science* 156:3 (2018), 312–322.

[16] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. 2018. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NeurIPS, 2020–2029.

[17] M F Kasim, D Watson-Parris, L Deaconu, S Oliver, P Hatfield, D H Froula, G Gregori, M Jarvis, S Khatiwala, J Korenaga, and et al. 2021. Building high accuracy emulators for scientific simulations with deep neural architecture search. *Machine Learning: Science and Technology* 3, 1 (2021), 015013.

[18] Ariel Keller Rorabaugh, Silvina Caíno-Lores, Travis Johnston, and Michela Taufer. 2022. Building High-Throughput Neural Architecture Search Workflows via a Decoupled Fitness Prediction Engine. *IEEE Transactions on Parallel and Distributed Systems* 33, 11 (2022), 2913–2926.

[19] Ariel Keller Rorabaugh, Silvina Caíno-Lores, Michael R. Wyatt II, Travis Johnston, and Michela Taufer. 2021. Architecture Descriptions and High Frequency Accuracy and Loss Data of Random Neural Networks Trained on Image Datasets.

[20] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. 2017. Learning Curve Prediction with Bayesian Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S11KBYclx

[21] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. arXiv:1603.06560 [cs.LG]

[22] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. 2018. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118* (2018).

[23] Chaoyue Liu and Mikhail Belkin. 2018. Accelerating SGD with Momentum for Over-parameterized Learning. *arXiv preprint arXiv:1810.13395* (2018).

[24] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive Neural Architecture Search. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[25] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2018. Hierarchical Representations for Efficient Architecture Search. In *Proceedings of the International Conference on Learning Representations*.

[26] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. 2019. NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm. arXiv:1810.03522 [cs.CV]

[27] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. 2018. Neural Architecture Optimization. In *Advances in Neural Information Processing Systems 31*. NeurIPS.

[28] Zhiping Mao, Ameya D. Jagtap, and George Em Karniadakis. 2020. Physics-informed Neural Networks for High-speed Flows. *Computer Methods in Applied Mechanics and Engineering* 360 (2020), 112789. https://www.sciencedirect.com/science/article/pii/S0045782519306814

[29] Paula Olaya, Silvina Caino-Lores, Vanessa Lama, Ria Patel, Ariel Keller Rorabaugh, Osamu Miyashita, Florence Tama, and Michela Taufer. 2022. Identifying structural properties of proteins from X-ray free electron laser diffraction patterns. *2022 IEEE 18th International Conference on e-Science (e-Science)* (2022).

[30] Gyunam Park and Minseok Song. 2020. Predicting performances in business processes using deep neural networks. *Decision Support Systems* 129 (2020), 113191.

[31] Ria Patel, Ariel Keller Rorabaugh, Paula Olaya, Silvina Caino-Lores, Georgia Channing, Catherine Schuman, Osamu Miyashita, Florence Tama, and Michela Taufer. 2022. A methodology to generate efficient neural networks for classification of scientific datasets. *2022 IEEE 18th International Conference on e-Science (e-Science)* (2022).

[32] Robert M Patton, J Travis Johnston, Steven R Young, Catherine D Schuman, Don D March, Thomas E Potok, Derek C Rose, Seung-Hwan Lim, Thomas P Karnowski, and Maxim A Ziatdinov. 2018. 167-PFlops Deep Learning for Electron Microscopy: From Learning Physics to Atomic Manipulation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 638–648.

[33] Robert M. Patton, J. Travis Johnston, Steven R. Young, Catherine D. Schuman, Thomas E. Potok, Derek C. Rose, Seung-Hwan Lim, Junghoon Chae, Le Hou, Shahira Abousamra, Dimitris Samaras, and Joel Saltz. 2019. Exascale Deep Learning to Accelerate Cancer Research. In *2019 IEEE International Conference on Big Data (Big Data)*. 1488–1496.

[34] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* (2018).

[35] Rahul Ramesh and Pratik Chaudhari. 2022. Model Zoo: A Growing "Brain" That Learns Continually. arXiv:2106.03027 [cs.LG]

[36] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. 2017. Large-scale Evolution of Image Classifiers. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML, 2902–2911.

[37] Binxin Ru, Clare Lyle, Lisa Schut, Miroslav Fil, Mark van der Wilk, and Yarin Gal. 2021. Speedy Performance Estimation for Neural Architecture Search. https://arxiv.org/abs/2006.04492. arXiv:2006.04492 [stat.ML]

[38] Max Schwarzer, Bryce Rogan, Yadong Ruan, Zhengming Song, Diana Y. Lee, Allon G. Percus, Viet T. Chau, Bryan A. Moore, Esteban Rougier, Hari S. Viswanathan, and Gowri Srinivasan. 2019. Learning to Fail: Predicting Fracture Evolution in Brittle material models using recurrent graph convolutional neural networks. *Computational Materials Science* 162 (2019), 322–332. https://www.sciencedirect.com/science/article/pii/S0927025619301223

[39] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. 2021. Graph Neural Networks in Particle Physics. *Machine Learning: Science and Technology* 2, 2 (Jan 2021), 021001. https://doi.org/10.1088/2632-2153/abbf9a

[40] Tiberiu Stan, Zachary T. Thompson, and Peter W. Voorhees. 2020. Optimizing Convolutional Neural Networks to Perform Semantic Segmentation on Large Materials Imaging Datasets: X-ray Tomography and Serial Sectioning. *Materials Characterization* 160 (2020), 110119. https://www.sciencedirect.com/science/article/pii/S1044580319304930

[41] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G. Yen. 2020. Evolving deep convolutional neural networks for Image Classification. *IEEE Transactions on Evolutionary Computation* 24, 2 (2020), 394–407.

[42] Tom Viering and Marco Loog. 2021. The Shape of Learning Curves: a Review. *arXiv preprint arXiv:2103.10948* (2021).

[43] Guangyu Robert Yang and Xiao-Jing Wang. 2020. Artificial Neural Networks for Neuroscientists: A Primer. *Neuron* 107, 6 (2020), 1048–1070.

[44] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large Batch Optimization for Deep Learning: Training BERT in 76 Minutes. *arXiv preprint arXiv:1904.00962* (2019).

[45] Steven R. Young, Derek C. Rose, Travis Johnston, William T. Heller, Thomas P. Karnowski, Thomas E. Potok, Robert M. Patton, Gabriel Perdue, and Jonathan Miller. 2017. Evolving Deep Networks using HPC. *Proceedings of the Machine Learning on HPC Environments* (2017).

[46] Xiaolong Zheng, Peng Zheng, Liang Zheng, Yang Zhang, and Rui-Zhi Zhang. 2020. Multi-channel Convolutional Neural Networks for Materials Properties Prediction. *Computational Materials Science* 173 (2020), 109436. https://www.sciencedirect.com/science/article/pii/S0927025619307359

[47] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for Scalable Image Recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018).