Energy-Efficient Neural Network Training for Scientific Datasets with Advanced Similarity Analytics and Orchestration

Kin Wai NG[1 [0000−0001−9784−8427]], Orcun Yildiz[2 [0009−0006−5910−9221]], Tom Peterka[2 [0000−0002−0525−3205]], Florence Tama[3,4 [0000−0003−2021−5618]], Osamu Miyashita[3 [0000−0002−2530−5674]], Catherine Schuman[1 [0000−0002−4264−8097]], and Michela Taufer[1 [0000−0002−0031−6377]]

[1] University of Tennessee, Knoxville, TN 37996, USA
[2] Argonne National Laboratory (ANL), IL 60439, USA
[3] Center for Computational Science, RIKEN, Kobe, Hyõgo, JP
[4] Nagoya University, Nagoya, Aichi, JP

Abstract. Scientific computing increasingly depends on neural architecture search (NAS) to identify accurate neural networks (NNs) that facilitate breakthroughs in various fields, from protein classification to material discovery. However, conventional NAS workflows face challenges due to excessive training times and inefficient energy consumption resulting from redundant computations and inflexible orchestration. In this paper, we present A4NN.2, the next generation of the Analytics for Neural Network (A4NN) workflow, which overcomes these challenges by introducing a structural similarity engine and advanced orchestration using the Wilkins framework. These enhancements eliminate redundant training and enable modular high-performance workflow executions. A4NN.2 accelerates NN training, reduces energy consumption, and demonstrates broad applicability across benchmark datasets and scientific domains. When used to train NNs to classify protein configurations from X-ray images, A4NN.2 achieves significant efficiency gains by reducing computational costs while maintaining high accuracy, thus accelerating scientific discovery in structural biology.

Keywords: Workflows · Flow control · Energy efficiency · CIFAR10 · CIFAR100 · Protein XFEL diffraction dataset

1 Introduction

Scientific computing increasingly depends on neural architecture search (NAS) to identify accurate neural network (NN) configurations, driving advancements in fields such as protein classification and material discovery. However, long training times and inefficient energy use often hinder traditional NAS workflows due to redundant computations and inflexible orchestration frameworks. This paper presents A4NN.2, the latest version of the Analytics for Neural Network (A4NN) workflow, designed to overcome these challenges. A4NN.2 integrates

a structural similarity engine and advanced orchestration capabilities, using the Wilkins framework to improve efficiency. Building on its predecessor, A4NN.1 [4], which featured a prediction engine to estimate NN accuracy and terminate underperforming networks early, A4NN.2 introduces two key enhancements. First, it automatically identifies and eliminates structural similarities and redundancies among NNs, reducing unnecessary computations. Second, it optimizes workflow flexibility through advanced orchestration using the Wilkins framework [19], enhancing the modularity of tasks and overall efficiency. By reducing redundant computations, A4NN.2 significantly lowers the total FLOPs during the NAS search process—a key proxy for energy consumption—thereby enhancing the sustainability of NAS. We present our findings on the efficiency of A4NN.2, particularly in terms of its reduced energy consumption, while maintaining the accuracy level of NNs. This is demonstrated across two widely recognized benchmarks, CIFAR10 and CIFAR100, in addition to a dataset involving protein diffraction generated by X-ray Free Electron Laser (XFEL).

Identifying structural similarity and redundancy. A4NN.1 relied on a prediction engine that used parametric modeling methods to determine when to stop training early, but it did not address potential structural redundancies among candidate NNs. This oversight resulted in unnecessary training of similar networks, thereby increasing both computational demands and energy consumption. A4NN.2 introduces a new similarity engine that uses graph edit distance techniques to assess and identify structural similarities among NNs. By omitting networks that are structurally similar, we reduce redundant computations, optimize resource usage and improve energy efficiency.

Optimizing workflow flexibility with advanced orchestration. A4NN.1 used a tightly coupled workflow, which limited the flexibility to integrate new tasks or modify existing components without substantial rework. This rigidity hindered the ability to adapt the workflow to different NAS algorithms or evolving research needs. In A4NN.2, we integrate the Wilkins framework [19], a modular workflow orchestrator that decouples NAS algorithms from the task engines used for prediction and similarity. This modularization enhances flexibility, allowing for easy swapping of components, such as replacing the prediction engine with a more advanced model or adding new analytics tasks. The result is a flexible workflow specification, now configured through a high-level specification file, making it easier to customize and adapt to various research needs.

Demonstrate efficiency in energy consumption Although A4NN.1 reduced training time by early stopping, it did not fully optimize energy consumption, as redundant models were still being trained. Furthermore, it lacked a comprehensive evaluation of energy proxies beyond FLOPs. We transform A4NN.2 to achieve significant efficiency gains through the early termination of converging models combined with the exclusion of redundant models, minimizing both training time and energy usage, and comprehensive evaluation of energy proxies, including training time per epoch, total training time per model, and FLOPs per model. We validate the gains of A4NN.2 by applying the workflow to two benchmark datasets (CIFAR10 and CIFAR100) to evaluate its effectiveness across

varying complexity levels. The experimental results show up to 64.3% reduction in FLOPs and 59% reduction in training time in CIFAR10 and 53.4% reduction in FLOPs and 45% reduction in training time in CIFAR100. The well-known benchmarks are used to generate trust in the outcome of A4NN.2. More importantly, we demonstrate the impact of A4NN.2 in searching for accurate NNs for spectroscopy databases and in identifying protein types from X-ray diffraction images, reducing FLOPs by as much as 47.4% to 53.5% across varying beam intensities while preserving accurate solutions.

A4NN.2 addresses the reproducibility concerns of the AI community by open-source workflow configuration files, task codes, and orchestration scripts for full transparency and reproducibility, and integration with Data Commons to store and share training metadata, fostering community collaboration and validation. The code is available at: `https://github.com/TauferLab/A4NN_workflow`

2 The A4NN Workflow

We design and implement an ecosystem for accelerating NAS by integrating modular analytic engines that enhance search efficiency and reduce computational overhead. This modularization enables A4NN.2 to support a suite of engines, allowing for extensibility and incorporating additional analytical tasks beyond the core workflow. Figure 1 shows the A4NN.2 workflow orchestrated by the Wilkins workflow system. This workflow consists of three main tasks, the NAS algorithm (red), the parametric prediction engine (blue), and the similarity engine (green).

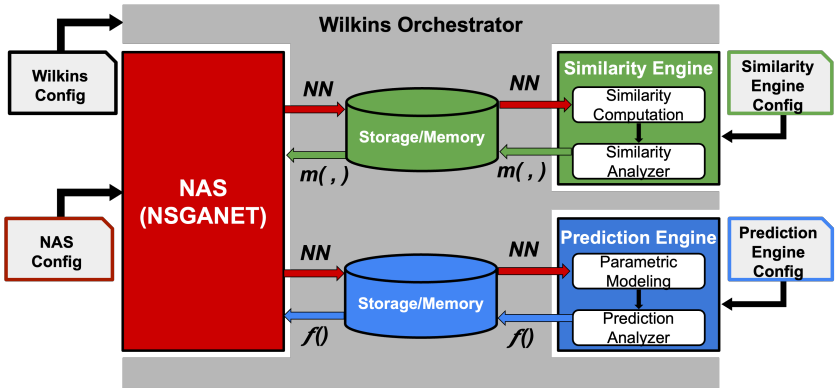


Fig. 1: Main components of the A4NN.2 workflow. Wilkins orchestrates the workflow, managing communication between three main tasks: NSGANET NAS (red), the similarity engine (green), and the prediction engine (blue). The similarity engine identifies and removes redundant NNs based on structural similarity, while the prediction engine estimates NN fitness to enable early termination. The entire workflow is configurable through high-level specification files.

A4NN.2 is compatible with various NAS frameworks, allowing flexibility in selecting search algorithms. In this study, we use NSGANET [12], a multi-objective approach that optimizes NNs for accuracy and computational efficiency. NSGANET evolves NN architectures iteratively, generating an initial population and refining it across generations through mutation and crossover operations. It prioritizes FLOPs minimization to encourage energy-efficient models. FLOPs are estimated by the number of floating-point operations needed to perform a single forward pass through the NN architecture. A4NN.2 accelerates NSGANET by introducing two key areas. The prediction engine estimates the fitness trajectory of NNs, enabling early termination when performance stabilizes. The similarity engine detects and eliminates structurally similar NNs, preventing redundant training and improving search efficiency. The Wilkins orchestrator facilitates seamless communication between these components, ensuring efficient data exchange and workflow execution.

To support diverse configurations with different NAS strategies, A4NN.2 provides configurable settings for each component. Users can specify parameters that control the NAS settings, the A4NN engines (i.e., prediction and similarity), and the Wilkins orchestrator settings. NAS and A4NN engines configurations are specified in JSON format, and Wilkins in YAML.

NAS Conf. File Users set the NSGANET parameters via a JSON file, which includes paths for input datasets, the initial population size, nodes per phase, offspring per generation, total generations, and training epochs.

A4NN Engine Files Users set up the A4NN engines via JSON files, defining parameters for the prediction and similarity engines. For the prediction engine, the configuration outlines the parametric function selection, the number of required data points, the epoch for predicting fitness, the count of predictions for early stopping, and the permitted prediction variance. The similarity engine settings specify the similarity metric and threshold for structural comparisons.

Wilkins Conf. File Users define data and resource needs in a YAML configuration file. Input and output file needs for each task are outlined, and data transfers occur via files. Resource allocation includes one process per task.

2.1 Identification of Structural Similarity and Redundancy

A4NN.2 augments the workflow with a similarity engine that identifies and removes structurally similar architectures before training begins. This prevents redundant computation and improves the overall efficiency of the search. The similarity engine is designed for comparison across NNs. It operates alongside NAS, analyzing architectures and signaling whether a given NN should proceed to training or be ignored. The similarity engine consists of two steps: the structure similarity computation and the similarity analyzer.

Following the encoding scheme of NSGANET, NNs are represented as directed acyclic graphs. Each NN consists of a sequence of phases, where computational operations (e.g., convolution, pooling, or batch normalizations) within a phase are encoded as a binary string. We decode the binary string into a graph, where vertices are computational operations, and edges are connections between

them. To quantify structural similarity, we use the approximated graph edit distance (GED) metric [1]. GED measures the minimum number of edit operations (e.g., vertex or edge insertions or deletions) required to transform one graph in to the other. A lower GED indicates a high degree of similarity between two NN graphs, while a higher GED suggests less similarity. GED has been widely used in many applications requiring graph comparisons [6, 3]. The similarity engine is designed for extensibility, and in future work, we can explore more advanced metrics, including kernel-based [9], or embedding-based approaches [13].

The similarity analyzer step evaluates whether a NN scheduled for training is structurally similar to an already trained network. If the similarity measure indicates redundancy, the analyzer signals the NAS to discard the architecture, thus avoiding redundant computation. If no similar architecture is found, training proceeds as usual, with the prediction engine determining whether early termination is possible. By filtering similar NNs, the similarity engine accelerates the NAS process, reducing computational cost of model search without compromising diversity in candidate NNs.

2.2 Decoupling and Modularization of Workflows for Flexibility

A4NN.1 used a plug-in that was not optimized for scalable communication, which led to rigid communication patterns. This limitation constrained the efficient execution of decoupled tasks in distributed environments. To address this, we empower A4NN.2 with the Wilkins orchestrator, introducing three key features: (i) High-performance HDF5-based data transport for scalable communication between NAS and decoupled tasks; (ii) Adaptive flow control mechanisms to manage data dependencies dynamically, minimizing idle time and optimizing resource utilization; and (iii) Automatic communication channel creation by matching data requirements between NAS and workflow tasks, streamlining execution. These features significantly improve scalability and reduce execution overhead, enabling the workflow to leverage larger HPC systems efficiently.

As shown in Figure 1, Wilkins orchestrates A4NN.2 by launching its tasks concurrently, and managing their communication and dependencies transparently to the user. In this workflow, the NSGANET task sends NN architecture information to the similarity engine, and receives similarity data, $m(,)$, from previously trained NNs to determine whether the NN should be eliminated from training. Next, the NSGANET task sends NN architecture information to the prediction engine, and receives fitness predictions, $f()$, to assess the potential for early termination. Data exchanges occur over HDF5 files, ensuring interoperability and structured storage of A4NN's generated data. Wilkins also allows tasks to communicate in situ using MPI message passing; however, we chose file-based communication to ensure reproducibility and facilitate the integration of the results with Data Commons. Wilkins leverages the LowFive data model [16], an HDF5-VOL plugin, which enables seamless integration with existing A4NN task codes with minimal modification. By decoupling workflow tasks and managing their execution through a modular framework, Wilkins provides the flexibility needed to adapt A4NN to diverse NAS implementations and research objectives.

3 Evaluating A4NN on Benchmark Datasets

We evaluate A4NN on benchmark datasets to assess its efficiency, accuracy, and overall performance. Our evaluation is structured around four key questions: (i) Runtime performance: How much does A4NN reduce the time required for NAS compared to NSGANET; (ii) Energy efficiency proxies: How effectively does A4NN reduce training epochs, training time, and FLOPs compared to NSGANET?; (iii) Impact of model complexity: How does model complexity influence energy efficiency metrics such as FLOPs per epoch?; and (iv) Balancing accuracy and efficiency: How does A4NN balance model accuracy and computational efficiency, and what trade-offs exist between these metrics? We present our results on CIFAR10 and CIFAR100 datasets.

3.1 Benchmark Datasets and Evaluation Settings

We evaluate A4NN using the CIFAR10 and CIFAR100 datasets, two widely used computer vision benchmarks that differ in complexity, allowing for a thorough assessment across varying task difficulties. Both datasets were introduced in 2009 [10] as a subset of the 80 Million Tiny Images dataset [18].

CIFAR10: It contains 60,000 color images (32x32 pixels) divided into 10 categories, each with 6,000 images. It is divided into 50,000 training images and 10,000 test images, providing a balanced and straightforward classification task.

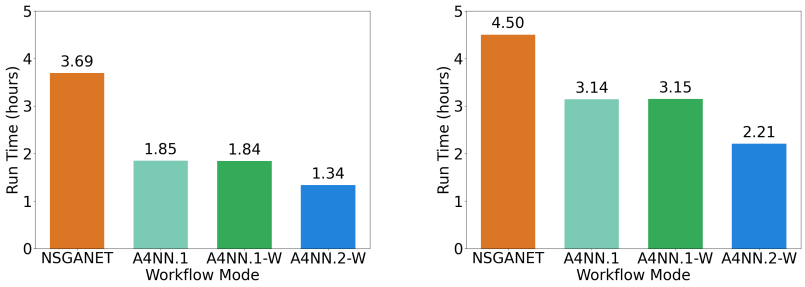
CIFAR100: This dataset is a more intricate version, consisting of 60,000 images divided into 100 detailed classes, each with 600 images. It is partitioned into 50,000 for training and 10,000 for testing. The increased class count and variability within classes present added classification difficulties.

We configure our experiments based on the NSGANET setup from our previous work [4], initializing a population of 10 NNs, generating 10 offspring per generation, and evolving over 10 generations, resulting in 100 trained models. Each model is trained for 25 epochs.

For the prediction engine, we use a concave function of the form $\mathcal{F}(x) = a - b^{c-x}$ to extrapolate a candidate fitness prediction at future epochs. We require three predictions within a variance threshold of 0.5 to determine convergence. For the similarity engine, we use GED with a threshold of 2 to determine structural similarity. We maintain consistent experimental parameters across all runs and repeat each workflow five times to assess variability. Experiments were run on the DARWIN HPC cluster at the University of Delaware, using one NVIDIA Tesla V100 GPU and four CPU cores from a 32-core AMD EPYC 7002 Series processor. The cluster features a high-performance Lustre filesystem.

3.2 Runtime Performance

To answer the question *"How much does A4NN accelerate NAS compared to NSGANET?"*, we evaluate the total runtime required to explore and identify best-performing NNs. Our analysis focuses on two key aspects: overall wall time and detailed runtime breakdowns to pinpoint efficiency gains.



(a) CIFAR10 (b) CIFAR100

Fig. 2: Wall times for NNs trained with standalone NAS and A4NN (with and without Wilkins orchestrator) on CIFAR10 and CIFAR100 datasets.

Figure 2 shows the total wall times for the CIFAR10 and CIFAR100 datasets, comparing NSGANET (independently) with the three A4NN configurations (i.e., A4NN.1 using only the prediction engine, A4NN.1-W using Wilkins and the prediction engine, and A4NN.2-W using Wilkins, the prediction engine, and the similarity engine). Our results demonstrate a substantially reduced runtime for all A4NN configurations compared to NSGANET. On CIFAR10 (Figure 2a), A4NN.1-W achieves a 2x speedup, while A4NN.2-W, which incorporates both the prediction and similarity engines, improves further with a 2.7x speedup. Similarly, for CIFAR100 (Figure 2b), A4NN.1-W provides a 1.4x speedup, and A4NN.2-W achieves a 2x improvement over NSGANET.

Table 1 provides a detailed decomposition of the runtime, highlighting the impact of the prediction and similarity engines of A4NN and the integration of Wilkins on overall efficiency. The results highlight two key points. First, Wilkins does not add overhead to the A4NN workflow, as shown by the minimal difference in execution time between A4NN.1 and A4NN.1-W. Second, as expected, NN training accounts for most of the total runtime, while the time spent on A4NN's task engines is relatively minimal. The results also confirm that A4NN can effectively accelerate NAS by reducing redundant computations, with the similarity engine providing additional runtime savings. The integration of Wilkins also ensures efficient and scalable orchestration without penalizing performance.

3.3 Energy-Efficiency Proxies

To answer the question *How effectively does A4NN reduce training epochs, training time, and FLOPs compared to NSGANET?*, we evaluate energy efficiency using three key proxies: the number of training epochs required per model, the total training time per model, and the computational cost in FLOPs per model. Our analysis focuses on comparing the distributions of these metrics to quantify reductions in training effort and computational cost.

Figures 3a and 3d show the distribution of epoch counts per model for CIFAR10 and CIFAR100, where NSGANET trains models to the maximum epoch

Table 1: Runtime breakdown per workflow component (in hours) for each method, comparing standalone NSGANET and A4NN variants across the CIFAR10 and CIFAR100 benchmarks.

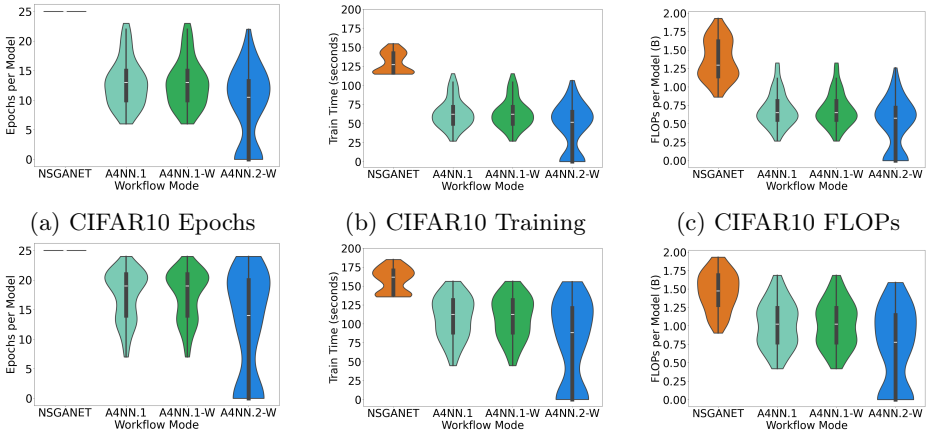
Dataset	Method	NAS Time	A4NN Task Time	Other Time	Total Time
CIFAR10	NSGANET	3.6 ± 0.01	N/A	0.09 ± 0.0	3.69 ± 0.01
	A4NN.1	1.78 ± 0.01	0.004 ± 0.0	0.06 ± 0.0	1.85 ± 0.01
	A4NN.1-W	1.77 ± 0.02	0.004 ± 0.0	0.06 ± 0.0	1.84 ± 0.02
	A4NN.2-W	1.25 ± 0.01	0.03 ± 0.0	0.06 ± 0.0	1.34 ± 0.01
CIFAR100	NSGANET	4.4 ± 0.02	N/A	0.1 ± 0.0	4.5 ± 0.01
	A4NN.1	3.05 ± 0.01	0.007 ± 0.0	0.09 ± 0.0	3.14 ± 0.01
	A4NN.1-W	3.05 ± 0.04	0.006 ± 0.0	0.09 ± 0.0	3.15 ± 0.04
	A4NN.2-W	2.1 ± 0.03	0.03 ± 0.0	0.09 ± 0.0	2.21 ± 0.04

limit (i.e., 25) while A4NN methods adaptively terminate training earlier. We observe that A4NN methods significantly reduce the number of training epochs compared to NSGANET. A4NN.2-W achieves the lowest median epoch count, reducing training epochs by 58% on CIFAR10 and 44% on CIFAR100. Similarly, A4NN achieves substantial reductions in training time (Figures 3b and 3e), with A4NN.2-W reducing the median training time per model by 59% on CIFAR10 and 45% on CIFAR100. We observe a similar trend for FLOPs (Figures 3c and 3f), where A4NN.2-W achieves the highest efficiency, reducing FLOPs by 55% on CIFAR10 and 47% on CIFAR100 compared to NSGANET.

Moreover, the shape of the distributions in Figure 3 provides further insights into the variability between workflows. The NSGANET distributions are narrow across all metrics, unsurprisingly, since all models train for the maximum number of epochs, resulting in consistently high computational costs. In contrast, A4NN methods exhibit wider distributions, particularly A4NN.2-W, which shows a concentration of models near zero due to the elimination of redundant training for similar models. Differences in distribution shapes across datasets further highlight the impact of task complexity. On CIFAR10, many models fall in the lower range of the distributions, indicating a tendency to train less complex architectures. Conversely, CIFAR100 distributions show a greater proportion of models in the upper range, suggesting a preference for more complex architectures. Given the similar reduction trends and the consistent distribution shapes across energy proxies, we conclude that FLOPs serve as a reliable proxy for energy consumption and efficiency. Our results demonstrate that A4NN methods, especially A4NN.2-W, achieve substantial efficiency gains by reducing training epochs, training time, and FLOPs compared to NSGANET. These observed trends hold on both CIFAR10 and CIFAR100, despite differences in dataset complexity.

3.4 Impact of Model Complexity on Energy Proxies

To answer the question *How does model complexity influence energy efficiency metrics such as FLOPs per epoch?*, we evaluate the relationship between model

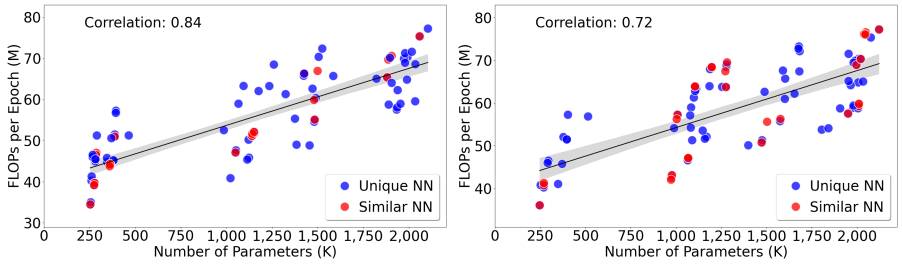


(a) CIFAR10 Epochs (b) CIFAR10 Training (c) CIFAR10 FLOPs

(d) CIFAR100 Epochs (e) CIFAR100 Training (f) CIFAR100 FLOPs

Fig. 3: Distribution of training epochs per model, training time (in seconds) per model, and FLOPs per model for four workflow methods.

complexity, measured by the number of trainable parameters, and FLOPs per epoch, our proxy for energy consumption. Our analysis focuses on identifying correlation between these variables to provide insights into training dynamics and their impact on reducing computational costs in NAS.



(a) CIFAR10 (b) CIFAR100

Fig. 4: Model complexity, measured by parameter numbers, correlates with epoch-wise FLOPs. Models are colored to denote structural uniqueness: blue dots for unique models, and red dots for models similar to past trained ones.

Figure 4 shows the relationship between model complexity and FLOPs per epoch for both CIFAR10 and CIFAR100 datasets. Each point represents a trained NN, color coded to distinguish between unique models (blue) and similar models (red). For CIFAR10, there are 28 similar models out of 100. For CIFAR100, there are 31 similar models out of 100. We observe a strong positive correlation

between model complexity and FLOPs per epoch. For CIFAR10, the correlation coefficient is 0.84, while for CIFAR100 is 0.72. These results indicate that computational cost increases with model complexity, which is expected. However, despite these strong correlations, models with similar parameter counts exhibit considerable variability in FLOPs. This variability suggests that models with the same parameter count can have significantly different architectures (e.g., differing in layer arrangement, types, and connectivity patterns) directly impacting computational cost.

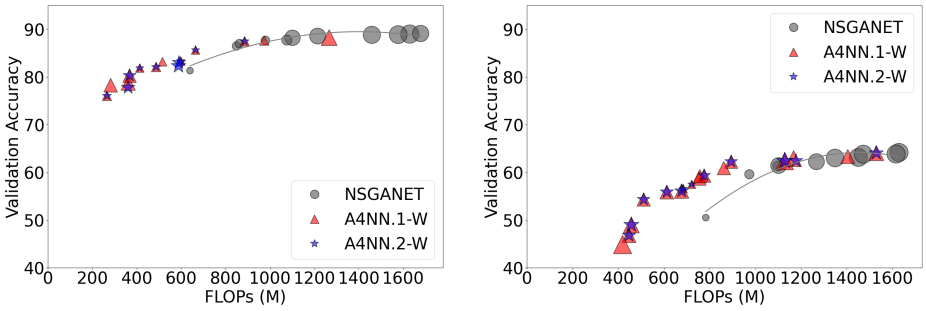
Furthermore, we observe that similar models are not concentrated in a specific region but are instead dispersed across the parameter space. This distribution indicates that structurally similar networks can have different computational costs despite having a similar number of parameters. These findings highlight the importance of our similarity engine in identifying redundant models, and reducing unnecessary computational costs associated with training them.

3.5 Balancing Accuracy and Efficiency

To answer the question "How does A4NN balance model accuracy and computational efficiency, and what trade-offs exist between these metrics?", we analyze the relationship between validation accuracy and FLOPs across trained NNs. Our analysis focuses on two key aspects: constructing Pareto frontiers to capture optimal trade-offs and quantifying FLOPs reductions achieved by A4NN to pinpoint efficiency gains.

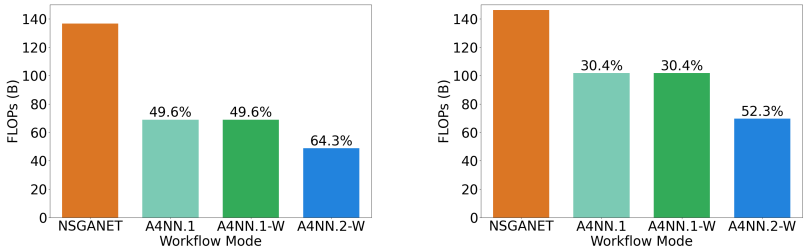
Figure 5 presents the Pareto optimal solutions identified by A4NN variants and NSGANET, where each point represents a model, with symbol shape indicating the workflow method and symbol size corresponding to the number of parameters. By analyzing these frontiers, we evaluate A4NN's ability to balance accuracy and efficiency compared to NSGANET. For CIFAR10, A4NN models (blue and red) achieve validation accuracy comparable to NSGANET (gray) while requiring fewer FLOPs. It is important to note that A4NN.2 preserves most of the solutions identified by A4NN.1, even when similar models are dropped from training. Furthermore, A4NN models often reach high accuracy models with fewer parameters than NSGANET. We observe a similar pattern for CIFAR100 (Figure 5b), where A4NN models maintain efficiency gains over NSGANET, achieving similar accuracy at a lower computational cost. This is evident from the leftward shift of Pareto optimal solutions generated by A4NN compared to those of NSGANET. We did not tune hyper-parameters or apply data augmentation, keeping settings aligned with A4NN.1 for fair comparison. Despite this, A4NN.2 reliably selects competitive architectures.

We further quantify these improvements in Figure 6 by aggregating total FLOPs across all 100 evaluated architectures. A4NN.2-W achieves a 64.3% reduction in FLOPs in CIFAR10 and a 53.4% reduction on CIFAR100 compared to NSGANET. Overall, these findings demonstrate that A4NN.2 effectively balances accuracy and efficiency, reducing computational costs significantly while retaining high-performing solutions.



(a) CIFAR10: Pareto Frontiers (b) CIFAR100: Pareto Frontiers

Fig. 5: Pareto-optimal frontiers for NNs illustrating the trade-off between validation accuracy and total FLOPs. Models produced by NSGANET (gray circles), A4NN.1-W (red triangles), and A4NN.2-W (blue stars) are shown as markers. Each marker is a Pareto-optimal model where enhancing one metric (accuracy or FLOPs) reduces the other. Marker size denotes the model's parameter count.

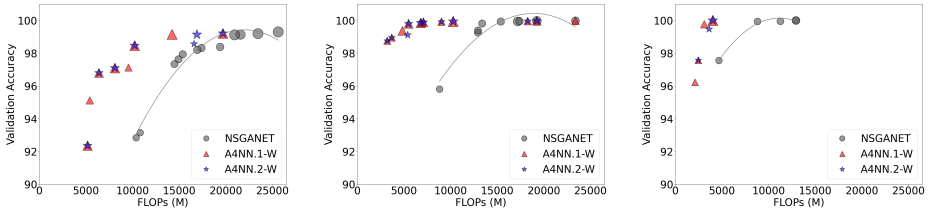


(a) CIFAR10 (b) CIFAR100

Fig. 6: Total FLOPs for training 100 NN architectures and FLOPs percentages saved by A4NN vs. standalone NSGANET for CIFAR10 and CIFAR100.

4 Applying A4NN to Scientific Datasets

We compare A4NN's performance to NSGANET to evaluate the trade-off between accuracy and computational cost in a real-world scenario. Specifically, we use A4NN with the Protein XFEL Diffraction dataset [14] to assess its ability to reduce power consumption while training NNs for classifying protein conformations within the dataset. The dataset consists of diffraction patterns generated by XFEL experiments, where proteins are exposed to intense laser beams, producing photo scattering patterns that capture structural information. The *spsim* simulator was used to generate different diffraction patterns for two conformations of EF2 with PDB ID 1n0u and 1n0v from the Protein Data Bank. In this study, we generate, train, and evaluate NNs to classify these protein conformations (i.e., differentiate between 1n0u and 1n0v). The dataset includes three subsets generated by varying intensities of the XFEL beam on the same proteins:



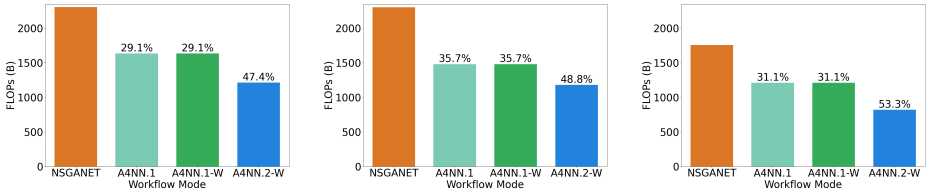
(a) Low (b) Medium (c) High

Fig. 7: Pareto-optimal frontiers for NNs using (a) Low, (b) Medium, and (c) High beam intensity protein diffraction datasets, illustrating the balance between validation accuracy and total FLOPs. Models by NSGANET (gray circles), A4NN.1-W (red triangles), and A4NN.2-W (blue stars) are depicted as markers. Each marker is a Pareto-optimal model, where enhancing one metric affects the other. Marker size indicates the number of model parameters.

Low (1×10^{14} photons/μm^2/pulse), Medium (1×10^{15} photons/μm^2/pulse), and High (1×10^{16} photons/μm^2/pulse). 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.

Figure 7 shows the FLOPs of the Pareto optimal models selected by each workflow method across all intensity levels, where lower values indicate better performance. At Low beam intensity, A4NN.2-W achieves comparable validation accuracy to NSGANET while reducing FLOPs by approximately 47.4%, demonstrating a more efficient search for optimal architectures even in noisy data scenarios. Specifically, A4NN.2-W models consistently require fewer than 10,000 MFLOPs, compared to NSGANET models that exceed 15,000 MFLOPs. This reduction not only accelerates training, but also conserves energy, highlighting the capability of A4NN to handle low signal-to-noise ratios efficiently. At Medium beam intensity, A4NN.2-W continues outperforming NSGANET by achieving the same high validation accuracy with up to 48.8% fewer FLOPs. The Pareto front shows that A4NN models cluster around 10,000 MFLOPs, whereas NSGANET models require up to 20,000 MFLOPs. This efficiency gain accelerates the convergence to high accuracy, highlighting A4NN's adaptability to medium noise levels. Even at High beam intensity, where data quality is improved and training naturally converges faster, A4NN.2-W maintains a FLOPs reduction of 53.3% compared to NSGANET. The Pareto frontiers reveal that A4NN models consistently achieve high accuracy with FLOPs below 5,000 MFLOPs, while NSGANET models demand higher computational costs.

Figure 8 shows the total FLOPs required for training 100 NN architectures and percentages of FLOPs saved by A4NN compared to standalone NSGANET for the protein diffraction datasets. Figure 8 shows the total FLOPs required for training 100 NNs and the corresponding percentages of FLOPs saved by A4NN compared to standalone NSGANET for protein diffraction datasets. Lower FLOPs indicate better energy efficiency and reduced computational cost.



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

Fig. 8: Total FLOPs for training 100 NN architectures and FLOPs percentages saved by A4NN vs. standalone NSGANET for the protein diffraction datasets.

At low beam intensity, A4NN.2-W achieves the highest savings, reducing total FLOPs by 47.4% compared to NSGANET. This significant reduction is due to the effective elimination of redundant models and the early termination of converging architectures. Both A4NN.1 and A4NN.1-W achieve a 29.1% reduction, highlighting the impact of early termination even without the similarity engine. The considerable savings in FLOPs at low beam intensity demonstrate the ability of A4NN to efficiently handle noisy data sets where signal-to-noise ratios are low and training is typically more resource intensive. For medium intensity, A4NN.2-W demonstrates superior performance by reducing total FLOPs by 48. 8%, maintaining high precision while requiring fewer computations. Both A4NN.1 and A4NN.1-W achieve a 35.7% reduction, showcasing the benefits of modular orchestration and early stopping. The medium-intensity efficiency gains highlight A4NN's adaptability to datasets with moderate noise levels, optimizing training cycles without sacrificing model performance. At high beam intensity, where data quality improves and training naturally converges faster, A4NN.2-W achieves a remarkable 53.3% reduction in total FLOPs compared to NSGANET, the highest savings across all intensity levels. In contrast, both A4NN.1 and A4NN.1-W yield a 31.1% reduction. The superior performance of A4NN.2-W at high beam intensity illustrates its capability to maximize resource utilization even under optimal data conditions. This efficiency gain is attributed to the synergy between the structural similarity engine and advanced orchestration, which collectively eliminate redundant computations.

Figures 7 and 8 demonstrate that A4NN.2-W consistently achieves significant energy savings across all intensity levels, reducing total training costs while maintaining high accuracy.

5 Related Work

This study builds upon the previous works of Olaya et al. [14], Patel et al. [15], Rorabaugh et al. [8], and Channing et al. [4]. We also take inspiration from other studies leveraging NAS for scientific datasets, such as Kandasamy et al. [7] and Balaprakash et al. [2]. Olaya et al. [14] introduced the XPSI framework to predict protein types and orientations from 2D diffraction patterns, but required signif-

icant human intervention and did not address computational efficiency. Patel et al. [15] expanded XPSI by employing NSGANET for NAS, reducing manual tuning. However, their approach still faced long runtimes and lacked distribution. Rorabaugh et al. [8] introduced the PENGUIN fitness prediction engine to decouple search and prediction strategies, improving efficiency in NAS workflows. Channing et al. [4] proposed the A4NN workflow which laid the foundation for a more efficient workflow in NN training. These studies, however, do not address potential architectural redundancies among candidate NNs in NAS. Other NAS applications for scientific datasets, like DENSE [7] and cancer modeling on HPC machines [2], face similar challenges of time and resource consumption, limiting accessibility for domain scientists. Efforts to improve NAS efficiency have led to methods like early stopping [11], learning curve extrapolation [5], and training speed estimation [17], which reduce computation time and resource usage. This work addresses these challenges by augmenting the A4NN workflow to further reduce wall times and energy consumption.

6 Conclusion

This paper demonstrates the effectiveness of A4NN.2 in accelerating NAS while significantly reducing energy consumption across diverse datasets. Integrating a structural similarity engine and advanced orchestration, A4NN.2 eliminates redundant training and optimizes resource usage. Our experiments on a Protein XFEL Diffraction dataset show that A4NN.2 achieves up to 47.4% FLOPs savings at Low beam intensity, 48.8% at Medium beam intensity, and 53.3% at High beam intensity compared to NSGANET, while maintaining high accuracy. These gains are especially prominent in noisy datasets, where A4NN.2 accelerates convergence with fewer training epochs. Overall, A4NN.2 balances accuracy and efficiency, paving the way for sustainable scientific computing.

Acknowledgments

This work was supported by the National Science Foundation (NSF) under grant numbers 2331152 and 2223704. This work was supported by Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

References

1. Abu-Aisheh, Z., et al.: An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems. In: 4th International Conference on Pattern Recognition Applications and Methods (2015)
2. Balaprakash, P., et al.: 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 (2019)

3. Bunke, H., Allermann, G.: Inexact Graph Matching for Structural Pattern Recognition. Pattern Recognition Letters (1983)
4. Channing, Georgia and et al.: Composable Workflow for Accelerating Neural Architecture Search Using In Situ Analytics for Protein Classification. In: Proceedings of the 52nd International Conference on Parallel Processing (2023)
5. Domhan, T., et al.: Speeding up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves. In: Proceedings of the 24th International Conference on Artificial Intelligence (2015)
6. Ferrer, M., Bunke, H.: Graph Edit Distance–Theory, Algorithms, and Applications. Image Processing and Analysis with Graphs: Theory and Practice (2012)
7. Kasim, M.F., et al.: Building High Accuracy Emulators for Scientific Simulations with Deep Neural Architecture Search. Machine Learning: Science and Technology (2021)
8. Keller Rorabaugh, A., et al.: Building High-Throughput Neural Architecture Search Workflows via a Decoupled Fitness Prediction Engine. IEEE Transactions on Parallel and Distributed Systems (2022)
9. Kriege, N.M., et al.: A Survey on Graph Kernels. Applied Network Science (2020)
10. Krizhevsky, A., Hinton, G.: Learning Multiple Layers of Features from Tiny Images. Tech. rep., University of Toronto, Toronto, Ontario (2009)
11. Li, L., et al.: Hyperband: A Novel Bandit-based Approach to Hyperparameter Optimization. Journal of Machine Learning Research **18**(185), 1–52 (2018)
12. Lu, Z., et al.: NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 419–427 (2019)
13. Makarov, I., et al.: Survey on Graph Embeddings and their Applications to Machine Learning Problems on Graphs. PeerJ Computer Science **7**, e357 (2021)
14. Olaya, P., et al.: Identifying Structural Properties of Proteins from X-ray Free Electron Laser Diffraction Patterns. IEEE 18th International Conference on e-Science (e-Science) (2022)
15. Patel, R., et al.: A Methodology to Generate Efficient Neural Networks for Classification of Scientific Datasets. IEEE 18th International Conference on e-Science (e-Science) (2022)
16. Peterka, T., et al.: Lowfive: In Situ Data Transport for High-performance Workflows. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS) (2023)
17. Ru, R., et al.: Speedy Performance Estimation for Neural Architecture Search. Advances in Neural Information Processing Systems (2021)
18. Torralba, A., et al.: 80 Million Tiny Images: A Large Dataset for Nonparametric Object and Scene Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (2008)
19. Yildiz, O., et al.: Wilkins: HPC in Situ Workflows Made Easy. Frontiers in High Performance Computing (2024)