Eurographics Symposium on Parallel Graphics and Visualization (2021)
M. Hadwiger, M. Larsen, F. Sadlo (Editors)

Machine Learning-Based Autotuning for Parallel Particle Advection

Samuel D. Schwartz[1], Hank Childs[1] and David Pugmire[2]

[1]Department of Computer and Information Science, University of Oregon, Eugene, Oregon, United States
[2]Oak Ridge National Laboratory, Oak Ridge, Tennessee, United States

Abstract

Data-parallel particle advection algorithms contain multiple controls that affect their execution characteristics and performance, in particular how often to communicate and how much work to perform between communications. Unfortunately, the optimal settings for these controls vary based on workload, and, further, it is not easy to devise straight-forward heuristics that automate calculation of these settings. To solve this problem, we investigate a machine learning-based autotuning approach for optimizing data-parallel particle advection. During a pre-processing step, we train multiple machine learning techniques using a corpus of performance data that includes results across a variety of workloads and control settings. The best performing of these techniques is then used to form an oracle, i.e., a module that can determine good algorithm control settings for a given workload immediately before execution begins. To evaluate this approach, we assessed the ability of seven machine learning models to capture particle advection performance behavior and then ran experiments for 108 particle advection workloads on 64 GPUs of a supercomputer. Our findings show that our machine learning-based oracle achieves good speedups relative to the available gains.

CCS Concepts
• ***Human-centered computing*** $\rightarrow$ ***Scientific visualization; Visualization techniques;***

1. Introduction

Particle advection is used to calculate the trajectory a massless particle follows when placed at some seed location within a vector field. These trajectories are built through a series of displacements called "advection steps," each of which involves solving an ordinary differential equation. Particle advection is a foundational operation for many flow visualization techniques, which use these trajectories to form their own visual output: Poincaré analysis, Finite-Time Lyapunov Exponents (FTLE), Line Integral Convolution (LIC), or just plotting the trajectories directly (by animating particles or via pathlines and streamlines). The workload of particle advection tasks can vary highly based across flow visualization techniques, ranging from one particle to billions, advecting for few steps or many, whether seeds are placed close together or far apart, etc. This variation can significantly affect the performance of the particle advection algorithm.

Particle advection algorithms become more complicated in the context of supercomputers. In this setting, the vector field is typically so large that it cannot fit into the memory of an individual compute node, and so it must be decomposed into blocks. Further, this makes parallel processing more difficult — particle and data must be brought together at the same time to perform an advection step. Many techniques have been proposed for efficient parallelism (see 2.1). With this work, we consider the "parallelize-over-data" (POD) approach, where blocks are partitioned over compute nodes, and particles are sent between compute nodes as they move from block to block.

Within POD, there are still many decisions to make, and the best choices for these decisions are not clear. This is the central challenge motivating this research work. In particular, we consider two tunable characteristics ("knobs") of the POD algorithm and how to select their values. First, when particle advection work is sent to the GPU, the execution happens in an atomic fashion. If there is a lot of work to perform and if all of that work is sent to the GPU as one atomic operation, then that compute node may slow down overall processing — the compute node is not available to take on new work and also it is not sending out particles that exit its blocks to the other compute nodes. At the other extreme, if only a small amount of work is sent to the GPU at a time, then the GPU is forced to do many kernel executions, which increases overhead. In all, the first control is how much work to send to the GPU in one execution. The second control is how often to communicate between compute nodes. If particles are sent on an individual basis, then the compute network can be flooded with many, small messages. However, having a compute node wait and collect these particles into large messages can also cause problems, as it introduces a delay before the next compute node can start advecting.

With this work, we investigate machine learning (ML) based au-

DOI: 10.2312/pgv.20211039

totuning for determining good control settings for POD particle advection. We follow a two-phase process, first training a variety of machine learning models on a corpus of performance data and second generating an oracle that uses effective ML models to choose control settings. We then run a series of experiments on supercomputers using this oracle. Our central research questions are:

- **RQ1:** Which machine learning techniques can sufficiently model the performance of the POD particle advection algorithm?
- **RQ2:** How much speedup does POD particle advection receive from this autotuning approach? Further, how does this speedup compare to optimal settings?

While our primary contribution is on improving particle advection performance, secondary contributions include (1) showing how machine learning-based autotuning can be applied to the scientific visualization space and (2) showing which machine learning models are most useful for this specific problem.

2. Related Work

2.1. Particle Advection on Supercomputers

There are two fundamental ways of parallelizing the problem, POD and "parallelize over particles" (POP), as well as hybrid methods that use elements of both approaches. In the POD method [PPG12], data blocks are distributed over processes. Each process advects the particles located in its data block until each terminates or exits the spatial bounds of the block. When a particle exits a block, it is communicated to the that processor working on its new block. This process continues until all particles terminate.

A number of extensions to POD have been considered to improve the performance of the algorithm. Sisneros et al. [SP16] studied the impact of communication granularity in the POD algorithm. Optimizations for spatial decomposition have been studied by Peterka et al. [PRN*11] to improve load balance. Hybrid parallelism (i.e., both shared and distributed memory) has also been considered for particle advection. Camp et al. [CGC*11] first looked at these methods for streamlines, followed by work looking at the performance on different types of workloads and hardware types [CKP*13, CBP*14]. Finally, Pugmire et al. [PYK*18] provided a hardware-portable method for shared memory particle advection using VTK-m [MSU*16].

2.2. Optimizing and Tuning Algorithm Performance

Optimization and autotuning has long been an active area of research. Balaprakash et al. [BDG*18] provides a comprehensive overview of autotuning oriented optimization techniques for high performance computing applications. These approaches cover a wide spectrum of use cases across the lifecycle of applications, and include the use of search- and machine learning-based techniques. Machine learning-based techniques typically construct a surrogate model over the parameter space of the application and then perform a search over this model to find an optimal configuration. Herodotou et al. [HCL20] provide a survey for autotuning big data software systems using a number of strategies, including heuristic rule, cost model, simulation and machine learning-based methods. There have been many autotuning works in HPC, including the following notable efforts. Zhang et al. [ZDH*] described RLScheduler, an automated system using reinforcement learning to provide optimal scheduling of batch jobs on high performance computing systems. Yigitbasi et al. [YWLE13] assessed several machine learning models' efficacy in autotuning MapReduce parameters and found that a support vector regression model had good accuracy and computational efficiency. Autotuning for performance and energy usage of stencil-based applications running on multicore hardware was addressed using statistical machine learning by Ganapathi et al. [GDFP09].

There have been fewer works on optimizing visualization performance using autotuning and/or machine learning. Bethel et al. [BH12] used a parameter sweep across the input data, output size, hardware and algorithm parameters to identify trends and best practice recomendations. Bruder et al. [BFE17] used a machine-learning based method to predict the performance of interacitve ray casting. This prediction model was used to modify the algorithm parameters in real-time to maintain high frame rates and image quality. Frey et al. [FE16] used a search-based autotuning technique to find optimal paramemters used for in situ generation of data extracts for post processing. The goal of the autotuning was to identify parameters that minimized data size, while at the same time maximising the time required and quality of post processing results.

3. Our Approach

Our approach has two phases. The first phase creates machine learning models that inform particle advection algorithm performance (§3.1). The second phase constructs an "oracle" that visualization practitioners can then use to autotune a POD particle advection algorithm (§3.2).

3.1. Definitions

3.1.1. Machine Learning Model

A trained machine learning model is developed from two components: a data corpus and a particular machine learning architecture (e.g., a neural network, support vector machine).

A data corpus is a set of samples, where each sample is an ordered pair: (abscissa, ordinate). In this study, the abscissa and ordinate refer respectively, to the inputs and outputs for the model. Here, the absicissa (input) contains information about both the workload characteristics (WC), and the algorithm characteristics (AC). The ordinate (output) defines the execution characteristics (EC). Explicitly, these characteristics consist of the following:

- WC: Particle advection workload characteristics (abscissa/input): Flow field, seeding strategy, number of particles, number of advection steps.
- AC: Particle advection algorithm characteristics (abscissa/input). These are the "knobs" used for the amount of work given to the GPU (Batch size), and frequency of communication (Delay send).
- EC: Execution characteristics (ordinate/output): Speedup over the default algorithm characteristics.

Each sample in the corpus are of the form: $\{(WC, AC), \quad (EC)\}$

A machine learning (ML) technique trains on a data corpus, and infers relationships between the input (abscissa) and output (ordinate) of the data corpus. After training, the ML technique can predict an output for a given input. Thus, after a training phase, an ML model can take workload and algorithm characteristics, (WC, AC) and predict execution characteristics, (EC). That is, given the number of seeds, steps, batch size, etc., an ML model can predict the speedup. Looking ahead to our experiments (§4.1.3), we considered seven machine learning techniques. We also considered two different data corpora for training: workloads that included flow field as an explicit characteristic ("flow field sensitive") and those that excluded flow field ("flow field agnostic"). In all, we considered 14 types of machine learning models.

3.1.2. Oracle

An oracle can be thought of as a function that takes in workload characteristics as input, and outputs a prediction of whether speedup of a particle advection algorithm is possible compared to a baseline and, if so, provides the particle advection algorithm characteristics that it believes will achieve maximum speedup. Note that this is subtly different than our machine learning model.

- Given WC and AC, the machine learning model (ML) gives EC: $ML(WC, AC) = EC$
- Given WC, the oracle gives AC that maximizes speedup: $Oracle(WC) = AC$

In our approach, an oracle uses the predictions from a machine learning model to decide the best algorithm characteristics. Explicitly, for a given WC (i.e., number of particles, steps, and flow field), an oracle returns the AC (i.e., batch size and delay) that it believes will provide the optimal performance. There is some flexibility in how an oracle utilizes its machine learning model, and we consider two variants (see §3.3.2). These two variants, combined with 14 machine learning models yields a total of 28 possible oracles.

3.2. Workflow

Our worfklow, reflected in Figure 1, consists of two phases, model generation and oracle generation, which are described in §3.2.1 and §3.2.2.

3.2.1. Phase 1: Assembling a Machine Learning Model

The purpose of the first phase is to generate a good ML model that can be used in the oracle. It contains two steps.

Step 1: Generate a diverse set of candidate ML models which draw from different architectures, hyperparameters, and data preparation techniques. Many authors have described techniques for developing and applying individual machine learning models to a specific problem. We find the nine-step technique by researchers from Microsoft in [ABB*19] to be particularly informative as a practical framework for constructing individual models.

There are two areas of critical importance for the successful modeling of phenomena. These are: (1) data collection and preparation into appropriate input for a machine learning technique, (2) selection of the hyperparameters and other settings of the machine learning technique. Frequent points of failure in the development of an ML model hinge on inappropriate choices made in (1) or (2).

Step 2: Once each machine learning model in our set has been trained, it will then be assessed for efficacy in modeling phenomena. Models should be assessed on their own merits (e.g., does model X have high enough accuracy to be useful), as well as evaluated through direct comparison with the other models under assessment (e.g., is model X more accurate than model Y). The end outcome of assessment determines whether there is a viable model that sufficiently predicts the targeted execution characteristic. Viable models can be used in the creation of an oracle.

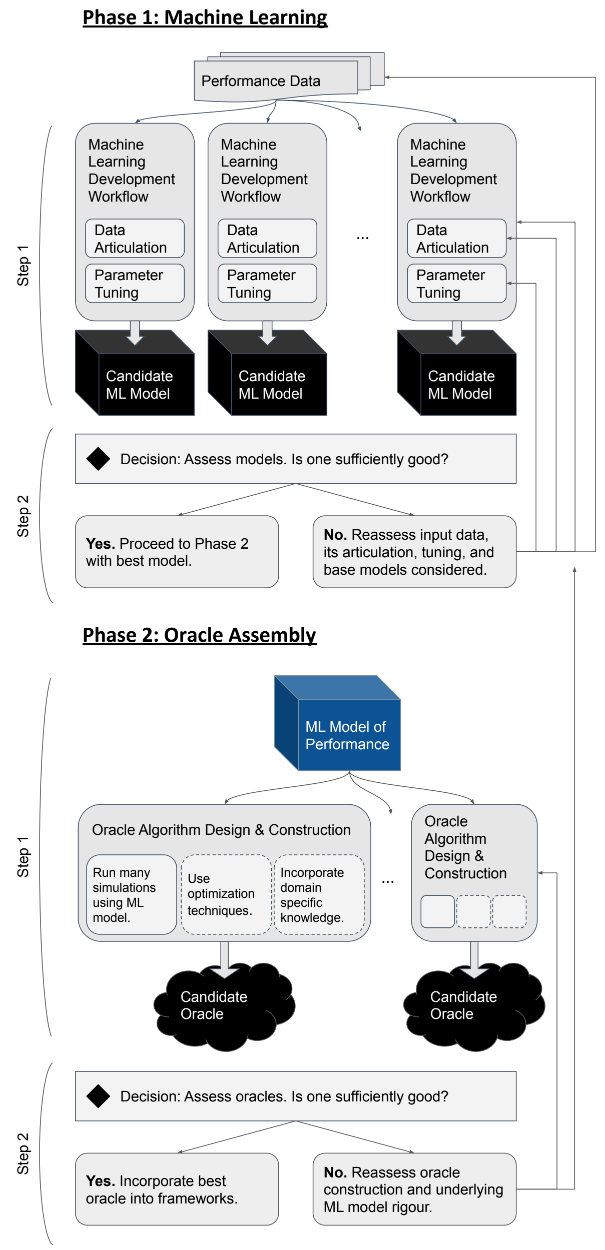
Phase 1: Machine Learning
Performance Data
Step 1
Machine Learning Development Workflow
Data Articulation
Parameter Tuning
Candidate ML Model
...
Step 2
Decision: Assess models. Is one sufficiently good?
Yes. Proceed to Phase 2 with best model.
No. Reassess input data, its articulation, tuning, and base models considered.
Phase 2: Oracle Assembly
ML Model of Performance
Oracle Algorithm Design & Construction
Run many simulations using ML model.
Use optimization techniques.
Incorporate domain specific knowledge.
Oracle Algorithm Design & Construction
Candidate Oracle
Decision: Assess oracles. Is one sufficiently good?
Yes. Incorporate best oracle into frameworks.
No. Reassess oracle construction and underlying ML model rigour.

Figure 1: *Flowchart describing our approach for generating an oracle to optimize particle advection performance.*

Finally, one possibility from this phase is that no ML model adequately captures particle advection performance. In this case, we iterate through the model creation process again. Two common points for machine learning modeling faiulre are (1) a lack of adequate data integrity and preparation, and (2) insufficient tuning of settings and parameters of the machine learning architecture.

3.2.2. Phase 2: Assembling an Oracle

Oracle development consists of two steps: generation and assessment.

Step 1: This step targets the construction of variant algorithms. The key principle of Phase 2 is that all oracle generation techniques which use our approach will utilize the predictive power of the ML models generated in Phase 1 to simulate many runs of our particle advection algorithms; far more runs than were provided to train our ML models in the first place. The design of oracles may also include optimization processes and the use of domain specific knowledge which is explicitly known by human developers and may have implicitily been learned by the ML model. As described in §3.3.2, we construct two oracle variants. Both variants use a naive optimization technique, wherein we search over the ML model to determine the algorithm characteristics which maximizes speedup across a wide swath of parameters. One variant also incorporates domain specific knowledge about the underlying behavior of the particle advection algorithm.

Step 2: Once an oracle is constructed it will then be assessed for efficacy in predicting performance characteristics. Oracles that predict execution and algorithm characteristics when given workload characteristics should be assessed against the true execution characteristics of runs defined by those workload and algorithm characteristics. As with machine learning models, oracles should also be evaluated through direct comparison with the other oracles under assessment (e.g., is oracle X more accurate than oracle Y).

3.3. Oracle Generation

We construct two oracles which use information gleaned from the best machine learning model to predict optimal algorithm characteristics based on a given workload. Both of these oracles make use of a lookup table (LUT).

3.3.1. Table Based Optimization Approach

In an offline process (described in §4.2.1), we calculate the optimal algorithm characteristics for a set of workload characteristics by using the best machine learning model from Phase 1. This information is then stored in a lookup table that can be used subsequently by our oracles. Given a specific workload characteristic, the oracles provide algorithm characteristics that give the maximum predicted speedup. Since lookup table creation is done as an offline process, the calculation time does not affect algorithm performance. More importantly, lookup times are O(1). This speed is the motivating reason for using a lookup table over another optimization algorithm applied to the machine learning model which simulates the particle advection algorithm. Future work could investigate using online optimization strategies which have higher lookup time complexities, yet don't require *a priori* construction of a lookup table.

3.3.2. The Two Oracles Considered

We consider two variants of oracles as defined below. Let WC be workload characteristics, LUT be the lookup table from §3.3.1, and AC be the answer returned by the lookup table, i.e., $AC = LUT(WC)$. Further, assume the attributes of characteristics are accessible. Specifically, let $AC.BATCH\_SIZE$ be the recommended batch size for AC, $AC.SPEEDUP$ be the predicted speedup for AC, and $WC.ADV\_STEPS$ be the number of advection steps for WC. Finally, denote the algorithm characteristics for default execution as $DEFAULT$. Both of our oracles rely heavily on the lookup table from §3.3.1 and Oracle 1 modifies it only slightly.

Algorithm 1: Oracle Variant I
```
A=LUT(W)
if $A.SPEEDUP > 1$ then
    return $A$
else
    return $DEFAULT$
```

One important modification for the variant given by Oracle 2 involves when to detour from the default configuration. The default configuration of our parallel particle advection algorithm ("batch size == all") is actually a good configuration in many cases, and so setting the batch size to a finite value will typically lead to slowdowns. In fact, we suspect that when `batch size > number of advection steps` the default settings will almost always be a better choice. In this case, we want the oracles to recommend the default algorithm configuration, and is an example of using decision logic predicated on domain specific knowledge.

Algorithm 2: Oracle Variant II
```
A=LUT(W)
if $A.SPEEDUP > 1$ and $A.BATCH\_SIZE <$
  $W.ADV\_STEPS$ then
    return $A$
else
    return $DEFAULT$
```

Summarizing, both variants capture the desired behavior of using the default mode if no speedups are possible. The distinguishing component to Oracle Variant II is that it also checks to see if the batch size is less than the number of advection steps. This is because the batch size limits how many steps can be taken, If that batch size is larger than the number of advection steps, then imposing this limit just causes an overhead with no benefit.

4. Experimental Overview

This section provides an overview of two sets of experiments that were performed. The experiments for Phase 1 (§4.1) considered the accuracy of 14 machine learning models. The most accurate models were used as input for the experiments for Phase 2 (§4.2).

4.1. Experimental Overview for Phase 1

These experiments are described in three parts: description of the input data (§4.1.1), machine learning techniques (§4.1.3), model validation and assessment strategy (§4.1.4).

4.1.1. Input Data

The input to Phase 1 was performance data collected from several thousand runs with varied workloads of our particle advection algorithm on the supercomputer Summit [VdB*18]. The selected runs and workloads represented a broad range of use cases for particle advection with algorithm settings selected by a visualization expert. A statistician later augmented these with additional experiments to provide better coverage of the overall space.

Each workload was run with the default algorithm parameters, AC_D, and with a number of non-default parameter settings, $AC_1, AC_2, \ldots AC_N$, where N is the number of times it was run. If the runtime for a workload using the default settings is T_D, and T_i when using settings AC_i, the speedup, S_i, is defined as $S_i = T_D/T_i$.

We then form a sample for our input data as follows: (WC, AC_D, AC_i, S_i).

To ensure equal coverage of the search space of non-default settings across a range of workloads, the following method was employed to select training data:

1. A visualization practitioner selected control settings thought to be good.
2. A statistician assessed each control setting and standardized it. Namely, control features which had an exponential distribution as selected by the visualization practitioner had the natural log applied to it, and then was normalized by subtracting the mean of each non-Boolean feature and dividing by the standard deviation.
3. The statistician applied a principal component analysis (PCA) to the standardized data. This projection, when visualized, had gaps in where the data lay.
4. Gaps in the PCA projection were filled by superimposing a rectilinear grid and selecting the coordinate in the grid which filled the largest hole. This was done by a naive approach, where the grid coordinate which was furthest away from all its neighbors in the PCA projection's Euclidean space was the one selected.
5. The information represented by the PCA selected coordinate was then inverted back into a raw set of control features which could be run on Summit and treated as a new sample.
6. The constructed sample was then added to the set of existing samples. We then repeated this process from point 3. until the PCA projection had no visible gaps in the training data.

In the end, we had a representative corpus of more than 2500 samples to use as our training data.

Specifically, a sample (WC, AC_D, AC_i, S_i) is drawn from one of the following combinations:

- WC: Particle Advection Workload Characteristics
  - Flow Field: Three flow fields were selected by a visualization practitioner. The three flow fields (Fishtank, Fusion and Astro) are vector fields from the NEK5000 thermal hydraulics code [FLK08], NIMROD plasma fusion code [SGG*04] and GenASiS [ECBM10] core-collapse supernova code, respectively.
  - Seeding Strategy: Two seeding strategies were selected, "sparse seeding" and "whole seeding." The sparse seeding strategy placed the seeds at random locations in a confined region of interest in the dataset. The whole seeding strategy placed seeds at random locations inside the total extents of the dataset.
  - Number of Particles: Varied between 1,000 and 500,000,000.
  - Number of Advection Steps: Varied between 100 and 10,000.
- AC_D and AC_i: Particle Advection Algorithm Characteristics
  - Batch Size: Varied between 2 and 100,000,000.
  - Delay Send: A Boolean value (i.e., true or false).
- S_i: Speedup when using Non-Default Algorithm Characteristics compared to Defaults.
  - Speedup: Varied between 0.003139 and 1.604249.

4.1.2. Data Articulation to Machine Learning Techniques

Data cleaning was done to standardize the samples of the form (WC, AC_D, AC_i, S_i) before being fed to our machine learning techniques as a vector $\vec{i}$. This included taking the natural log of features with an exponential distribution, normalizing data by subtracting the mean of each non-Boolean feature and dividing by the standard deviation – as was done in the data sampling process – and removing the default settings for a workload, AC_D, from consideration. Additionally, we created two variants of our corpus, "flow field sensitive" and "flow field agnostic." In the "flow field sensitive" variant, the flow field is explicitly stated in $\vec{i}$. In the "flow field agnostic" variant, $\vec{i}$ does *not* contain the flow field.

We define the vector $\vec{i} = \begin{bmatrix} inputs = \vec{x} \\ target = \vec{y} \end{bmatrix}$, with the following two forms:

$$\underline{\text{Flow field sensitive}}$$

$$input = \vec{x} = \begin{bmatrix} \text{Flow Field Is “Astro”} \in \{0,1\} \\ \text{Flow Field Is “Fishtank”} \in \{0,1\} \\ \text{Flow Field Is “Fusion”} \in \{0,1\} \\ \text{Seeding Strategy Is “Sparse”} \in \{0,1\} \\ \text{Seeding Strategy Is “Whole”} \in \{0,1\} \\ \text{Delay Send} \in \{0,1\} \\ \text{Normalize(Ln(Number of Particles))} \in \mathbb{R} \\ \text{Normalize(Ln(Number of Advection Steps))} \in \mathbb{R} \\ \text{Normalize(Ln(Batch Size))} \in \mathbb{R} \end{bmatrix}$$

$$target = \vec{y} = \begin{bmatrix} \text{Speedup} \in \mathbb{R} \end{bmatrix}$$

$$\underline{\text{Flow field agnostic}}$$

$$input = \vec{x} = \begin{bmatrix} \text{Seeding Strategy Is “Sparse”} \in \{0,1\} \\ \text{Seeding Strategy Is “Whole”} \in \{0,1\} \\ \text{Delay Send} \in \{0,1\} \\ \text{Normalize(Ln(Number of Particles))} \in \mathbb{R} \\ \text{Normalize(Ln(Number of Advection Steps))} \in \mathbb{R} \\ \text{Normalize(Ln(Batch Size))} \in \mathbb{R} \end{bmatrix}$$

$$target = \vec{y} = \begin{bmatrix} \text{Speedup} \in \mathbb{R} \end{bmatrix}$$

We use 10-fold cross validation (see §4.1.4) as our machine learning assessment strategy, which preempts the need to set aside a single testing dataset. Parameters used for external validation are discussed in §4.2.2 and were preprocessed the same as the training data (e.g., natural log taken and features normalized).

4.1.3. Machine Learning Techniques

The seven machine learning techniques we considered included three neural network and four classical machine learning models: linear regression, random forest, support vector machine and k nearest neighbors regression. These seven models and two data corpora ("flow field sensitive" and "flow field agnostic") make a total of fourteen machine learning models that were considered.

In the description of the seven models, we use the following notation: Let $\vec{x} \in \mathbb{R}^n$ denote an individual sample that is input to a machine learning model and let $\vec{y} \in \mathbb{R}^m$ denote the output of this model for an individual sample. Further, let $X = \{\vec{x}, \cdots\}$ denote the set of inputs for all the samples used in training.We note that $n = 9$ for "flow field sensitive," $n = 6$ for "flow field agnostic," and $m = 1$ for both corpora.

Neural Network Models: Each of our neural network models shared the following characteristics:

- The model is a fully connected, feed-forward neural network.
- The identify activation function is used for the output layer.
- Dropout is not incorporated at any point.
- The optimization algorithm used for training is Hinton's RMS Propagation [HSS12].
- The loss function is mean squared error.
- Neural network "training batch size" (not to be confused with the "batch size" in the data corpus) was chosen to be $\lfloor \sqrt{|X|} \rfloor$.
- 250 and 500 training epochs were used respectively for the "flow field agnostic" and "flow field sensitive" corpora. This selection was due to observed convergence of the loss function.

The neural network models only differed in the composition of their hidden layers, all of which used Rectified Linear Units (ReLUs). We describe the structural configurations for each using the following notation:

$$\underbrace{\textit{\# Inputs}}_{\text{Input}} \rightarrow \underbrace{\textit{\# Nodes in Layer}}_{\text{Layer Type}} \rightarrow \cdots \rightarrow \underbrace{\textit{\# Outputs}}_{\text{Output}}$$

- **Neural Network A** (Model A):

$$\underbrace{n}_{\text{Input}} \rightarrow \underbrace{\left\lceil \frac{1}{2} \cdot (n+m) \right\rceil}_{\text{ReLU Layer}} \rightarrow \underbrace{m}_{\text{Output}}$$

- **Neural Network B** (Model B):

$$\underbrace{n}_{\text{Input}} \rightarrow \underbrace{\left\lceil \frac{2}{3} \cdot (n+m) \right\rceil}_{\text{ReLU Layer}} \rightarrow \underbrace{\left\lceil \frac{1}{3} \cdot (n+m) \right\rceil}_{\text{ReLU Layer}} \rightarrow \underbrace{m}_{\text{Output}}$$

- **Neural Network C** (Model C):

$$\underbrace{n}_{\text{Input}} \rightarrow \underbrace{\left\lceil \frac{1}{2} \cdot (n+m) \right\rceil^2}_{\text{ReLU Layer}} \rightarrow \underbrace{m}_{\text{Output}}$$

Classic Machine Learning Models: The four models used were:

- **Linear Regression** (Model D): We fit a linear model using ordinary least squares fitting. We expect poor performance, but use it a point of comparision given its simplicty, speed and popularity.
- **Random Forest** (Model E): A forest of 100 regression trees, each using Gini impurity as its discrimination function. The forest included bootstrap sampling to build its trees, and used mean squared error as its discriminating criterion. All trees were fully expanded until each leaf node was pure.
- **Support Vector Machine** (Model F): Support Vector Regression using a radial basis kernel and gamma of $1/(n \cdot \mathrm{Var}(X))$.
- **K Nearest Neighbors** (Model G): A k Nearest Neighbors model was used, where the number of neighbors = $\lfloor \sqrt{|X|} \rfloor$.

The neural networks, models A, B, and C, were implemented in Keras [C*15] with a Tensorflow [AAB*15] backend. Models D, E, F, and G were implemented with Scikit-Learn [PVG*11]. Unless otherwise specified, the default parameters for the machine learning algorithms implemented in these libraries were utilized.

4.1.4. Machine Learning Assessment Strategy

Each machine learning model was evaluated by 10-fold cross validation. The model with the lowest Mean Absolute Error and least Unexplained Variance was considered the "best", and subsequently used for all of the oracles in Phase 2.

4.2. Experimental Overview for Phase 2

The overview for phase 2 is divided into two parts: construction details of a lookup table for optimal performance (§4.2.1), and information on our verification runs (§4.2.2). Phase 2 was executed twice, once for each of the best performing models that were trained on the flow-field sensitive and flow-field agnostic corpora.

4.2.1. Lookup Table Construction Details

The lookup table was constructed by running an ML model for a set of commonly encountered workload characteristics with $\approx$ 25,000 algorithm characteristics. For batch size, the values were selected in a logarithmically-spaced manner from 2 to 100,000,000. The value with the highest speedup was the recommend value for a given workload and was stored in the lookup table. The samples used for our experiments were contained in the $\approx$25,000 samples. In practice, for workloads not contained in the lookup table, there are several options. First, a distance metric could be used to determine the closest workload in the table and then use that entry. Second, interpolation between adjacent table entries could be used to determine the optimal settings. However, we have not studied which type of interpolation should be used to ensure accuracy. Third, a new type of oracle that does not use lookup tables could be embedded directly into the particle advection source code. That said, such an oracle would need to do significant searching over algorithm characteristics, and this search process would need to be optimized in order to have a negligible impact on performance.

If a lookup table is used, creation can be done as a one-time offline process done whenever a new supercomputer is deployed and the resulting table embeded into the production software. The actual realization of the oracle in production visualization software is an area requiring further investigation.

4.2.2. Oracle Assessment Strategy

To evaluate the performance of our oracles (and thus our overall approach), we ran 108 experiments. Each experiment was run multiple times: once with the default settings and once with each oracle we considered. The 108 experiments came from considering a cross product of particle advection workload factors:

- Number of particles (6 options): $10^3, 10^4, 10^5, 10^6, 10^7, 10^8$.
- Number of steps (3 options): $10^2, 10^3, 10^4$.
- Seeding strategy (2 options): sparse, whole.
- Flow Field (3 options): Fishtank, Fusion, Astro

4.3. Hardware Used

The particle advection algorithm was implemented in VTK-m [MSU*16, PYK*18] and run using 64 GPUs on the Summit supercomputer at Oak Ridge National Laboratory [VdB*18]. Each Summit node consists of 6 NVIDIA Volta V100 GPUs, 2 POWER9 CPUs, 608 GB of RAM and connected with a Mellanox EDR 100G InfiniBand. The machine learning model training, lookup table construction, and oracle output generation was performed on an Acer Aspire E 15 personal computer running on a 2.6GHz Intel Core i7-6500U CPU and 32 GB or RAM.

5. Results and Analysis

This section is organized by research question, with §5.1 focusing on **RQ1** and §5.2 focusing on **RQ2**.

5.1. RQ1: Using Machine Learning to Model Performance

This section considers **RQ1**: "Which machine learning techniques can sufficiently model the performance of the POD particle advection algorithm?" Further, the analysis is organized by phase, with Phase 1 (§3.2.1, 4.1) in §5.1.1 and Phase 2 (§3.2.2, §4.2) in §5.1.2.

5.1.1. Phase 1

Evaluation of the machine learning technique(s) which best captured POD behavior was performed by identifying trained models with the lowest mean absolute error (defined as |truth - prediction|) and unexplained variance under 10-fold cross validation. Unexplained variance is a measure of the degree to which our model fails to explain variation in the data. We found that R^2 scores, another popular measure of model goodness-of-fit, and explained variance were equal up to three decimal places for all of the models and data corpora we considered. As the unexplained variance is defined as $1-$explained variance, unexplained variance can be thought of as $1-R^2$ for the purposes of this study. These metrics provide evidence that the model has a good understanding of the relationship between a model's inputs and outputs and can therefore provide useful predictions.

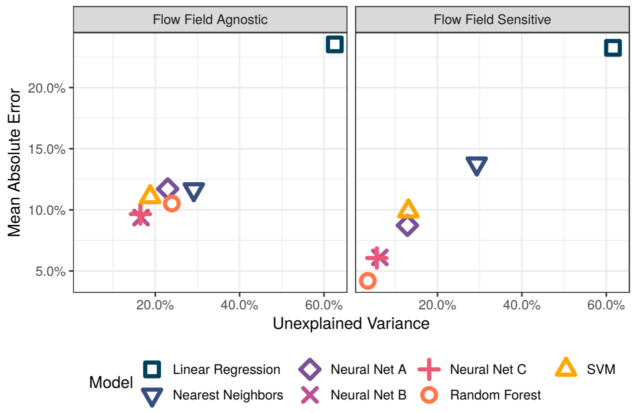

Figure 2: *Plots showing the performance of each model with respect to unexplained variance and mean absolute error. Plots are shown for both flow field agnostic (left) and flow field sensitive (right) corpora. Models closer to (0,0) are better.*

Figure 2 plots the mean absolute error and unexplained variance for all seven models and both data corpora. This plot shows that the most effective model for the flow field agnostic corpus is Neural Network C, as it has nearly the lowest mean absolute error and the most explained variance. For the flow field sensitive corpus, Figure 2 shows that the Random Forest model is the most effective, as it clearly has the lowest error and unexplained variance.

5.1.2. Phase 2

This subsection analyzes the efficacy of the two oracles from §3.3.2: "Oracle I" and "Oracle II." Oracles have a key distinction from the models from Phase 1. Where models take workload and algorithm characteristics as input and output a predicted speedup, oracles take only workload characteristics as input and output algorithm characteristics (ideally the settings that maximize speedup). These oracles achieve their task by using the best machine learning models from Phase 1 as their engines. For brevity, we denote Neural Network C (the best flow field agnostic model) as *NN C* and Random Forest (the best flow field sensitive model) as *RF*. The Cartesian product of the two oracles (I and II) and the two models (*NN C* and *RF*) results in four oracle-model configurations.

We evaluated 108 validation runs (described in §4.2.2) optimized over the two Delay Send options and more than 12,500 potential Batch Sizes. For both NN C and RF, the {Batch Size, Delay Send} combination with the highest predicted speedup was returned. This optimization was a one-time cost and the results were stored in a lookup table. We then executed runs with a predicted speedup greater than one on Summit. We refer to such runs as "acceleratable," since there is an opportunity to have them run faster than the default. The results for all four oracle-models are plotted in Figure 3, which shows the actual and predicted speedups for each configuration that was considered "acceleratable" by that oracle-model.

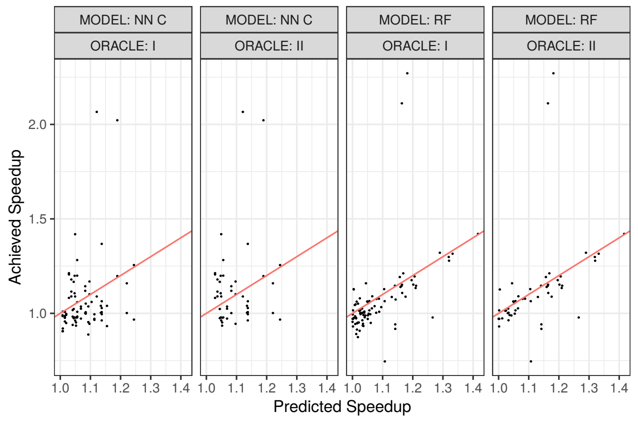

Figure 3: *Scatter plots of predicted speedup versus actual speedup for "acceleratable" runs for all four oracle-models. The red line in each figure represents perfect accuracy, i.e., the predicted speedup and actual speedup are the same.*

Table 1 lists information about the error for each of the four oracle-model configurations. For this analysis, "error" is the difference between the predicted speedup and the actual speedup. Positive values mean the predicted speedup was larger than the actual speedup, while negative values mean that the actual speedup was larger than predicted. Further, the analysis also considers the absolute value of error, which captures the extent to which the prediction was wrong. Finally, while we report outlier behavior (i.e., how much faster or slower a workload performed than the prediction), we feel the most important aspect is mean behavior (i.e., how much compute time is saved in aggregate by using the predicted best settings on a set of workloads).

Oracle Variant:	**I**	**II**	**I**	**II**
ML Model:	**NN C**	**NN C**	**RF**	**RF**
Min |Error|	0.001	0.004	0.001	0.001
Mean |Error|	0.111	0.141	0.079	0.091
Max |Error|	0.945	0.945	1.089	1.089

Table 1: *Summary of absolute errors for each of the four oracle-model configurations' "acceleratable" runs.*

All four oracle-models have mean absolute errors close to their mean absolute errors of the ML models they depend on. Comparing with results from §5.1.1, the mean absolute errors of *NN C - Oracle I* and *NN C - Oracle II* are within 5% of the mean absolute error of the NN C model, and *RF - Oracle I* and *RF - Oracle II* are also within 5% of the RF model. This provides evidence that the oracles are producing algorithm characteristics that provide good speedups with reasonable accuracy, despite the extremal results (i.e., highest speedup) of an oracle's search. Oracles with mean absolute error of significantly different value than the mean absolute error of their underlying ML models would be cause for much greater concern.

Finally, several of our validation runs were excluded, for one of two reasons. First, some of our runs had workload characteristics which prevented the baseline algorithm from completing within a two-hour timeout window. These workload characteristics always involved many particles and many advection steps. Second, one of large workload runs had communication degradation. This degradation was by an order of magnitude between compute nodes, and appeared to be due to severe network congestion unrelated to the POD advection algorithm. These cases constituted fewer than 5% of our validation runs and are excluded from further analysis. These $<$5% of cases are excluded since analysis of cases where time-to-completion is unrelated to the particle advection algorithm itself (e.g., by network noise or default timeout windows) would form an apples-to-oranges comparison when analyzing cases which completed naturally.

5.2. Answer to RQ2: Achieved Speedups & Contextualization

This section considers **RQ2**: "How much speedup does POD particle advection receive from this autotuning approach? Further, how does this speedup compare to optimal settings?" §5.2.1 discusses the speedups achieved and §5.2.2 contextualizes these results.

For these results, we consider three types of speedup. If $T_R(AC)$ is the runtime for some run R using some algorithm characteristics AC, if AC_D is the default algorithm characteristics, and if AC_O is the oracle-selected algorithm characteristics, then:

- **Speedup**, denoted S_R, compares the oracle-enabled algorithm compared to the default algorithm:
$$S_R = \frac{T_R(AC_D)}{T_R(AC_O)}$$
- **Mean Speedup** considers the average speedup over a set of N runs $\{R_1, R_2, \cdots, R_N\}$:
$$\frac{S_{R_1} + S_{R_2} + \cdots + S_{R_N}}{N}$$
- **Aggregated Speedup** considers the speedup achieved if every member of a set of N runs $\{R_1, R_2, \cdots, R_N\}$ is run one time (i.e., the result is weighted by run-time):

$$\frac{T_{R_1}(AC_D) + T_{R_2}(AC_D) + \cdots + T_{R_N}(AC_D)}{T_{R_1}(AC_O) + T_{R_2}(AC_O) + \cdots + T_{R_N}(AC_O)}$$

5.2.1. Achieved Speedup

To assess how much speedup is obtainable with our approach, we ran 108 validation runs with varying workload characteristics (discussed previously in §4.2.2 and §5.1.2) for each of the four oracle-model configurations. For a given workload, each oracle-model configuration provided both algorithm characteristics and a predicted speedup when using those algorithm characteristics. Table 2 reports the mean speedup for these validation runs.

Oracle Variant:	**I**	**II**	**I**	**II**
Machine Learning Model:	**NN C**	**NN C**	**RF**	**RF**
Mean Speedup Achieved	1.0516	1.0569	1.0586	1.0629

Table 2: *Overall mean speedup for each of the four oracle-model configurations.*

Next, we once again considered "acceleratable" configurations, i.e., the runs R where $S_R > 1$. Table 3 shows the rates at which each oracle-model predicts an "acceleratable" workload, and the rate at which acceleration is actually achieved. With respect to the latter rate, we determined this by running all "acceleratable" runs on Summit. Oracle Variant I is more aggressive in predicting speedup for both NN C and RF models. However, Variant II is more accurate than Variant I in predicting which "acceleratable" runs will actually achieve a speedup. This is true for both NN C and RF models.

Oracle Variant:	**I**	**II**	**I**	**II**
Machine Learning Model:	**NN C**	**NN C**	**RF**	**RF**
Rate of "Acceleratable" Runs	77.78%	47.22%	86.11%	51.85%
Rate of "Acceleratable" Runs Actually Achieving Speedup	56.96%	73.91%	61.80%	76.92%

Table 3: *"Acceleratable" rates for each of the four oracle-model configurations.*

We also used our Summit runs to determine the magnitude of speedup. Tables 4 and 5 report these results for each of the four oracle-model configurations for mean speedup and aggregate speedup, respectively. These tables also consider subsets of runs that are often most impactful on actual savings: execution times greater than 5, 10 and 60 seconds (with default settings, i.e., AC_D).

These results show that oracle choice varies based on setting. For runs of only a few seconds in duration, Oracle II obtains higher speedup for both NN C and RF. As execution time increases, the difference between Oracle I and Oracle II disappears for both mean speedup and aggregated speedup. Further, for nearly every analysis

Oracle Variant: **Machine Learning Model:**	**I** **NN C**	**II** **NN C**	**I** **RF**	**II** **RF**
Mean Speedup Achieved	1.067	1.127	1.068	1.126
Mean Speedup Achieved; Default Time > 5 seconds	1.131	1.150	1.161	1.161
Mean Speedup Achieved; Default Time > 10 seconds	1.157	1.171	1.192	1.192
Mean Speedup Achieved; Default Time > 60 seconds	1.203	1.203	1.239	1.239

Table 4: *Mean speedup for workloads considered "acceleratable." The top row considers the mean speedup over all acceleratable workloads, while the remaining rows consider only acceleratable workloads above certain time thresholds.*

Oracle Variant: **Machine Learning Model:**	**I** **NN C**	**II** **NN C**	**I** **RF**	**II** **RF**
Aggregated Speedup Achieved	1.182	1.185	1.204	1.206
Aggregated Speedup Achieved; Default Time > 5 seconds	1.184	1.185	1.207	1.207
Aggregated Speedup Achieved; Default Time > 10 seconds	1.185	1.186	1.208	1.208
Aggregated Speedup Achieved; Default Time > 60 seconds	1.189	1.189	1.213	1.213

Table 5: *Aggregated speedup for workloads considered "acceleratable." The top row considers the aggregated speedup over all acceleratable workloads, while the remaining rows consider only acceleratable workloads above certain time thresholds.*

focusing on longer execution times, RF has a higher mean or aggregated speedup than the same NN C oracle, by as much as 3%. This difference is very likely due to RF's lower unexplained variance and lower error (see §5.1.1). That said, this difference is small overall, i.e., a $0\% - 3\%$ better speedup is likely not meaningful enough to favor an algorithm that is flow field sensitive (RF) over one that is flow field agnostic (NN C).

In all, the main finding of this analysis is that a visualization practitioner should likely utilize Oracle II with NN C — in nearly all cases, the benefit of being flow field agnostic will outweigh the modest speedups from being flow field sensitive.

5.2.2. Assessing Speedups Achieved

This section analyzes how effective we are at achieving speedups. Such an analysis is difficult in nature, as the maximum speedup can only be knowable by running all possible control settings, which is either impossible (if the control settings have an infinite number of values) or merely prohibitively expensive (if the control settings have a finite number of values).

In our case, we decided to inform the effectiveness of our approach by comparing with the runs we executed to form our training corpus. This corpus has good properties and bad properties for this comparative analysis, which limits the conclusions that we can draw from our analysis overall. In terms of good properties, there were a large number of runs (2513) and the experiments represented "good" choices in the eyes of both a visualization practitioner and statistician (see §4.1.1). In terms of bad properties, the experiments for computationally expensive workloads were not as useful for direct comparison. The training corpus actually did contain computationally expensive workloads, but these workloads were picked in the interest of good sampling and not in the interest of comparison. For example, for an ML-autotuning experiment with 10M particles and 10K steps, the closest comparator in the corpus may have had 9M particles and 11K steps. In all, we have many direct comparators for small workloads, and fewer for big ones.

Approach	Aggregated Speedup	Mean Speedup
NN C / Oracle I	1.095	1.051
RF / Oracle I	1.090	1.048
Best of Corpus	1.124	1.095
All of Corpus	0.489	0.754

Table 6: *This table informs the available speedup for our particle advection workloads. It considers only a subset of the runs executed in our overall study. In particular, it only considers the 55 particle advection workloads that have at least ten runs in the training corpus (i.e., rows from Figure 4 where there are at least ten gray glyphs). The corpus is considered in two ways: best and all. For the "best" variant, the top performer of the ten-plus runs is used as the comparator. For the "all" variant, all of the ten-plus experiments are used as comparators.*

Figure 4 plots our comparison with the training corpus. One important conclusion from this plot is that the default settings are quite effective — most of the training experiments (gray glyphs) are slower than the default (blue glyphs). Further, while our ML-autotuning sometimes performs worse than default settings, it often does better, and increasingly better as execution time goes up. And while some of the training corpus experiments beat our ML-autotuning approach, the large majority are worse and also are worse than the default. In all, this provides some evidence that beating default settings is hard, and our ML-autotuning approach is doing at least a somewhat effective job.

Finally, can we quantify how effectively our approach performed? The results for this question can be found in Table 6. From this table, we conclude that our approach claims much of the possible speedup, but that there was more available. Of course, quantifying this further is hard — our training corpus does not represent all configurations and it remains possible that available speedups are higher than what this corpus indicates. That said, more runs in our training data would potentially allow our ML models to become more accurate.

6. Conclusion and Future Work

This research work considers applying machine learning-based autotuning to POD particle advection. We enacted a two-phase workflow that ingested training data, produced models, and evaluated their efficacy. We then ran a series of experiments using the best models as oracles to set values for POD particle advection controls.

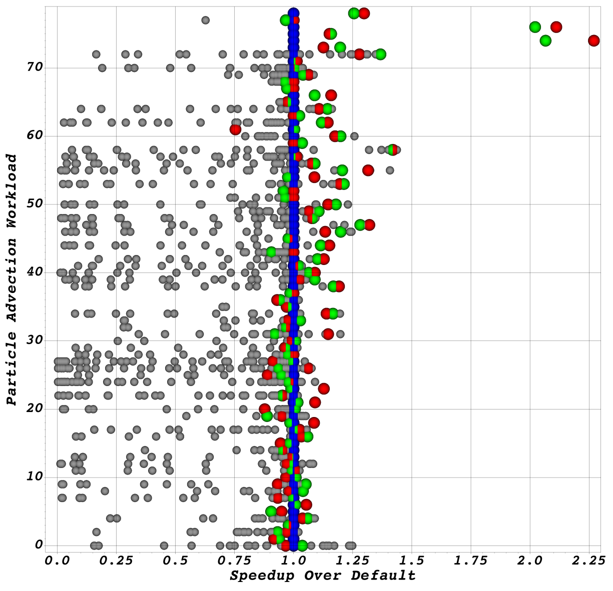

Figure 4: *Scatter plot of speedups achieved over default performance by our ML-autotuning approach and by experiments in our training corpus. Each glyph represents a run we executed, and is colored based on type: gray for runs with non-default algorithm characteristics in our training corpus, blue representing a run with default algorithm characteristics, green for a run with algorithm characteristics selected using NN C Oracle I ("NNCI"), and red for those that used RF Oracle I ("RFI"). Not all experiments are represented in this plot — the two ML-autotuning approaches of NNCI and RFI both predicted possible speedup for only 78 of our 108 particle advection workloads and only runs which are parameterized by these 78 workloads are included. These 78 workloads were ordered from shortest to longest by the execution time using default settings, and the index of this sorting became the Y-value for each glyph. As a result, each row corresponds to a single particle advection workload, and the runtime for that workload increases as the Y-value increases. Finally, the X-value of a glyph is its speedup compared to default settings. This is why all the blue (default) glyphs have X-values of 1.0. Putting it all together, if a given particle advection workload had the* 37^{th} *slowest run time and the algorithm characteristics given by NNCI led to a speedup 1.2X, then a glyph would be placed at (1.2, 37) and colored green.*

The results of these experiments varied — for small workloads the payoff was smaller, and it is doubtful that end-user visualization tools would reproduce our efforts. For larger workloads, the payoff was significant enough that the effort behind the methodology and the compute time could be useful (in the context of delivery within a tool like ParaView or VisIt on a supercomputer that would have repeated use). In all, however, we feel the largest utility in our study is in answering our research questions.

First, **RQ1** asked whether machine learning techniques could sufficiently model POD particle advection performance. We found that several of our models produced good results, with low unexplained variance and mean absolute error. We also looked at how this knowledge translated in practice and found that our predicted speedups had good correlation with real-world results. That said, this correlation could have been even stronger. We also considered whether the vector field mattered in our ability to predict performance. While we found that knowledge of vector field does allow for more accurate prediction, the difference in performance is likely not enough to pursue this direction. In particular, the vector field-agnostic approach makes the approach more broadly applicable. Further, the three vector fields we trained on are sufficiently diverse that we feel these results would hold up when a new vector field is introduced; confirming this belief would be useful future work. Finally, another direction of future work would be investigating improved models. In particular, we are interested in whether expanding our neural network's architecture (e.g., more nodes per layer, additional layers) improves oracle efficacy.

Second, **RQ2** asked what extent speedup we could achieve with the approach and whether the ML-autotuning approach successfully claimed most of the availale speedup. Our speedup analyses led to a variety of answers, but the most speedup we would expect is 20% (which is the number for large workloads). Further, while the ceiling of speedup is not fully illuminated, we speculate that we achieved about 75% of what is possible (loosely reasoned as 9% from our techniques versus 12% speedup from the best of the corpus). While this is a reasonable result, future work could pursue even better outcomes. In particular, our training data consisted of "speedup" as our only execution characteristic. More information may have allowed ML models to infer even deeper understanding of performance. As a final direction of future work, this approach could be applied to richer settings — more visualization algorithms, more control settings, more hardware architectures, etc.

Finally, an important addendum to **RQ2**, the question of potential speedup, is whether our proposed workflow should be applied in practice. We consider this question from the perspective of compute-node hours — developer time was significant in the context of this research project, but this cost could become negligible going forward. Therefore, this workflow is beneficial if the savings in production outweighs the training time. In our case, we trained for about 400 compute-node hours on Summit. If we were able to achieve a 10% speedup for 4000 compute-node hours of production usage over the lifetime of Summit, then the costs would balance, and more usage would lead to savings. It is unclear whether this many compute-node hours will actually be used for advection. If all jobs consisted of streamline generation running on eight nodes for 10 seconds, then there would have to be 180,000 such jobs to offset the training data. In the context of a four-year life span (1460 days), this would mean over one hundred streamlines per day on Summit, which we believe is more than current usage. That said, if the jobs take more nodes or more time, then the benefit is much more likely to be realized. In an in situ setting, running a 30-second FTLE computation for 100 cycles of a single simulation that uses the whole machine (4608 nodes) would offset training costs. In all, the answer depends on the use of advection workloads. Finally, as the training time is reduced, it becomes increasingly easy to offset its time.

References

[AAB*15] ABADI M., AGARWAL A., BARHAM P., BREVDO E., CHEN Z., CITRO C., CORRADO G. S., DAVIS A., DEAN J., DEVIN M., GHEMAWAT S., GOODFELLOW I., HARP A., IRVING G., ISARD M., JIA Y., JOZEFOWICZ R., KAISER L., KUDLUR M., LEVENBERG J., MANÉ D., MONGA R., MOORE S., MURRAY D., OLAH C., SCHUSTER M., SHLENS J., STEINER B., SUTSKEVER I., TALWAR K., TUCKER P., VANHOUCKE V., VASUDEVAN V., VIÉGAS F., VINYALS O., WARDEN P., WATTENBERG M., WICKE M., YU Y., ZHENG X.: TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: `http://tensorflow.org/`. 6

[ABB*19] AMERSHI S., BEGEL A., BIRD C., DELINE R., GALL H., KAMAR E., NAGAPPAN N., NUSHI B., ZIMMERMANN T.: Software engineering for machine learning: A case study. In *International Conference on Software Engineering (ICSE 2019) - Software Engineering in Practice track* (May 2019), IEEE Computer Society. 3

[BDG*18] BALAPRAKASH P., DONGARRA J., GAMBLIN T., HALL M., HOLLINGSWORTH J. K., NORRIS B., VUDUC R.: Autotuning in high-performance computing applications. *Proceedings of the IEEE 106*, 11 (2018), 2068–2083. 2

[BFE17] BRUDER V., FREY S., ERTL T.: Prediction-based load balancing and resolution tuning for interactive volume raycasting. *Vis. Informatics 1* (2017), 106–117. 2

[BH12] BETHEL E. W., HOWISON M.: Multi-core and many-core shared-memory parallel raycasting volume rendering optimization and tuning. *The International Journal of High Performance Computing Applications 26*, 4 (2012), 399–412. `doi:10.1177/1094342012440466`. 2

[C*15] CHOLLET F., ET AL.: Keras. `https://keras.io`, 2015. 6

[CBP*14] CHILDS H., BIERSDORFF S., POLIAKOFF D., CAMP D., MALONY A. D.: Particle Advection Performance over Varied Architectures and Workloads. In *IEEE International Conference on High Performance Computing (HiPC)* (Goa, India, Dec. 2014), pp. 1–10. 2

[CGC*11] CAMP D., GARTH C., CHILDS H., PUGMIRE D., JOY K. I.: Streamline Integration Using MPI-Hybrid Parallelism on a Large Multicore Architecture. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 17*, 11 (Nov. 2011), 1702–1713. `doi:10.1109/TVCG.2010.259`. 2

[CKP*13] CAMP D., KRISHNAN H., PUGMIRE D., GARTH C., JOHNSON I., BETHEL E. W., JOY K. I., CHILDS H.: GPU Acceleration of Particle Advection Workloads in a Parallel, Distributed Memory Setting. In *Proceedings of EuroGraphics Symposium on Parallel Graphics and Visualization (EGPGV)* (Girona, Spain, May 2013), pp. 1–8. 2

[ECBM10] ENDEVE E., CARDALL C. Y., BUDIARDJA R. D., MEZZACAPPA A.: Generation of Magnetic Fields By the Stationary Accretion Shock Instability. *The Astrophysical Journal 713*, 2 (2010), 1219–1243. 5

[FE16] FREY S., ERTL T.: Auto-tuning intermediate representations for in situ visualization. In *2016 New York Scientific Data Summit (NYSDS)* (2016), pp. 1–10. `doi:10.1109/NYSDS.2016.7747807`. 2

[FLK08] FISCHER P. F., LOTTES J. W., KERKEMEIER S. G.: nek5000 Web page, 2008. http://nek5000.mcs.anl.gov. 5

[GDFP09] GANAPATHI A., DATTA K., FOX A., PATTERSON D.: A case for machine learning to optimize multicore performance. *HotPar'09 Proceedings of the First USENIX conference on Hot topics in parallelism* (2009). 2

[HCL20] HERODOTOU H., CHEN Y., LU J.: A survey on automatic parameter tuning for big data processing systems. *ACM Computing Surveys 53* (04 2020), 1–37. `doi:10.1145/3381027`. 2

[HSS12] HINTON G., SRIVASTAVA N., SWERSKY K.: Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Neural networks for machine learning* (2012). 6

[MSU*16] MORELAND K., SEWELL C., USHER W., LO L., MEREDITH J., PUGMIRE D., KRESS J., SCHROOTS H., MA K.-L., CHILDS H., LARSEN M., CHEN C.-M., MAYNARD R., GEVECI B.: VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications (CG&A) 36*, 3 (May/June 2016), 48–58. 2, 7

[PPG12] PUGMIRE D., PETERKA T., GARTH C.: Parallel integral curves. In *High Performance Visualization: Enabling Extreme Scale Scientific Insight*, Bethel E. W., Childs H., Hansen C., (Eds.). CRC Press, 2012. 2

[PRN*11] PETERKA T., ROSS R., NOUANESENGSY B., LEE T., SHEN H., KENDALL W., HUANG J.: A study of parallel particle tracing for steady-state and time-varying flow fields. In *2011 IEEE International Parallel Distributed Processing Symposium* (2011), pp. 580–591. 2

[PVG*11] PEDREGOSA F., VAROQUAUX G., GRAMFORT A., MICHEL V., THIRION B., GRISEL O., BLONDEL M., PRETTENHOFER P., WEISS R., DUBOURG V., VANDERPLAS J., PASSOS A., COURNAPEAU D., BRUCHER M., PERROT M., DUCHESNAY E.: Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830. 6

[PYK*18] PUGMIRE D., YENPURE A., KIM M., KRESS J., MAYNARD R., CHILDS H., HENTSCHEL B.: Performance-Portable Particle Advection with VTK-m. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)* (Brno, Czech Republic, June 2018), pp. 45–55. 2, 7

[SGG*04] SOVINEC C., GLASSER A., GIANAKON T., BARNES D., NEBEL R., KRUGER S., PLIMPTON S., TARDITI A., CHU M., THE NIMROD TEAM: Nonlinear Magnetohydrodynamics with High-order Finite Elements. *J. Comp. Phys. 195* (2004), 355. 5

[SP16] SISNEROS R., PUGMIRE D.: Tuned to terrible: A study of parallel particle advection state of the practice. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2016), pp. 1058–1067. 2

[VdB*18] VAZHKUDAI S. S., DE SUPINSKI B. R., BLAND A. S., GEIST A., SEXTON J., KAHLE J., ZIMMER C. J., ATCHLEY S., ORAL S., MAXWELL D. E., LARREA V. G. V., BERTSCH A., GOLDSTONE R., JOUBERT W., CHAMBREAU C., APPELHANS D., BLACKMORE R., CASSES B., CHOCHIA G., DAVISON G., EZELL M. A., GOODING T., GONSIOROWSKI E., GRINBERG L., HANSON B., HARTNER B., KARLIN I., LEININGER M. L., LEVERMAN D., MARROQUIN C., MOODY A., OHMACHT M., PANKAJAKSHAN R., PIZZANO F., ROGERS J. H., ROSENBURG B., SCHMIDT D., SHANKAR M., WANG F., WATSON P., WALKUP B., WEEMS L. D., YIN J.: The design, deployment, and evaluation of the coral pre-exascale systems. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis* (2018), pp. 661–672. 4, 7

[YWLE13] YIGITBASI N., WILLKE T. L., LIAO G., EPEMA D.: Towards machine learning-based auto-tuning of mapreduce. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems* (2013), pp. 11–20. `doi:10.1109/MASCOTS.2013.9`. 2

[ZDH*] ZHANG D., DAI D., HE Y., BAO F. S., XIE B.: Rlscheduler: An automated hpc batch job scheduler using reinforcement learning. *SC'20: The International Conference for High Performance Computing, Networking, Storage, and Analysis 2020.* 2