PADDLE: Performance Analysis using a Data-driven Learning Environment

Jayaraman J. Thiagarajan[†], Rushil Anirudh[†], Bhavya Kailkhura[†], Nikhil Jain[†], Tanzima Islam[*], Abhinav Bhatele[†], Jae-Seung Yeom[†], Todd Gamblin[†]

[†]*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, California 94551 USA*
[*]*Computer Science Department, Western Washington University, Bellingham, Washington 98225 USA*
E-mail: [†]*{jjayaram, anirudh1, kailkhura1, nikhil, bhatele, yeom2, tgamblin}@llnl.gov,* [*]*Tanzima.Islam@wwu.edu*

***Abstract*—The use of machine learning techniques to model execution time and power consumption, and, more generally, to characterize performance data is gaining traction in the HPC community. Although this signifies huge potential for automating complex inference tasks, a typical analytics pipeline requires selecting and extensively tuning multiple components ranging from feature learning to statistical inferencing to visualization. Further, the algorithmic solutions often do not generalize between problems, thereby making it cumbersome to design and validate machine learning techniques in practice. In order to address these challenges, we propose a unified machine learning framework, PADDLE, which is specifically designed for problems encountered during analysis of HPC data. The proposed framework uses an information-theoretic approach for hierarchical feature learning and can produce highly robust and interpretable models. We present user-centric workflows for using PADDLE and demonstrate its effectiveness in different scenarios: (a) identifying causes of network congestion; (b) determining the best performing linear solver for sparse matrices; and (c) comparing performance characteristics of parent and proxy application pairs.**

***Keywords*-machine learning, performance data, feature learning, interpretability**

I. Motivation

There has been an unprecedented growth in the scale of supercomputers built over the last decade. The first Petaflop/s machine, Roadrunner, appeared on the top500 list in June 2008 with more than 100,000 cores. The fastest machine on the June 2017 list is Sunway TaihuLight with 100 times the computational power and cores as Roadrunner. There has been an explosion in the amount of performance data that can be gathered, often exacerbated by the increasing number of components identified as potential sources that can impact performance. Common data sources range from on-node hardware counters to CPU and DRAM power meters, and from hardware counters on network switches to temperature and humidity sensors in the machine room. Hence, performance analysis of high performance computing (HPC) codes at this scale requires analyzing enormous amounts of data, generated at the rate of tens to hundreds of terabytes a day.

The large volume and diversity of data being gathered makes it tedious in many cases, and impossible in others, for a human to analyze and gain insights. As a result, automated performance analysis methodologies are being actively pursued in HPC [1]. In particular, it is becoming increasingly common to adopt tools from machine learning (ML) for exploratory analysis and pattern discovery. In several recent efforts, this has enabled researchers to build predictive models, to infer complex correlation structures, to identify regions of interest for performance optimization, and to detect anomalous patterns [2], [3], [4], [5], [6].

The existing practice for applying ML to HPC problems involves designing an analysis pipeline using standardized frameworks such as `scikit-learn` [7], `Caffe` [8], and `Tensorflow` [9]. Although previous successes with ML signify huge potential for automation, a typical pipeline requires analysts to select and extensively tune multiple components ranging from feature learning to model selection to visualization. Further, the algorithmic solutions rarely generalize between problems, thereby making it cumbersome to design and validate the pipeline for every scenario.

The enormous size of the design space of algorithms, and varying analysis needs and constraints often necessitate interaction with ML experts to design targeted solutions. While this bottom-up approach has been effective for individual cases, it has high overheads in terms of person time, and can result in repeated rediscovery of steps that are common to several analyses. Moreover, HPC and ML researchers run into stumbling blocks as they develop familiarity of each other's domains, including generalizability of models, connection between data size and model uncertainties, validity of assumptions about the data made by ML techniques, etc. Finally, the black-box nature of complex ML algorithms, e.g. deep neural networks, makes it challenging for the analyst to interact and obtain insights into the functioning of the model.

In order to address these challenges of cross-domain interactions, lack of solution reusability, and the abundance of design choices, we propose a unified machine learning framework, *PADDLE* (Performance Analysis using a Data-driven Learning Environment), which bridges the gaps in the existing *modus operandi*, to better facilitate the use of ML in performance analysis. PADDLE provides an end-to-end inferencing pipeline for predictive analysis tasks encountered in HPC, while being robust to data uncertainties and interpretable for subsequent analysis. PADDLE uses deep information decomposition to extract important features from raw data and supports automated and model selection, even when there are multiple data sources. The HPC analyst does not have to explicitly tune any of the components in the pipeline.

The modular structure of PADDLE allows easy adaptation to any analysis task by extending its modules to use desired algorithms from existing ML libraries. The *Visualization* module in PADDLE enables the analyst to understand the characteristics of the inferred model, directly in terms of the input variables in raw data, and to identify inherent correlation structure. The proposed framework can answer questions that commonly arise in HPC performance analysis, such as: (i) which ML algorithm is suitable to my data? (ii) how to visualize and understand the black-box model? (iii) how to make models robust to data uncertainties? and (iv) how to reuse a ML solution for a different problem?

We demonstrate the utility of PADDLE by applying it to HPC datasets from three research studies that were published as one-off uses of ML and data analytics [4], [5], [6]. The first study aims to identify factors that impact network congestion and performance on HPC supercomputers. The second one identifies the best preconditioner-linear solver combination based on the properties of the input matrix. The third study studies the relationships between parent and proxy application pairs in terms of their performance behavior. For all these studies, we show that PADDLE automatically infers optimal models, and achieves state-of-the-art performance in prediction tasks, outperforming task-specific ML solutions. Interestingly, with its robust model design strategies, PADDLE produces much simpler models compared to existing practice, thereby making them interpretable for additional analysis.

In summary, the main contributions of this work are:

- A unified machine learning framework to tackle analytics problems related to HPC performance data.
- A parameter-tuning free, deep feature learning approach that automatically identifies correlations in data.
- Strategies to perform robust model design, even with limited training data.
- Visualization of the learned feature space for understanding black-box models.

II. HPC Performance Analysis and Data

The performance of parallel applications on large supercomputers is often dependent on several factors: the input problem being solved, the algorithm being used and its parameter choices, hardware/system dependent parameters etc. A common goal of performance analysis of an application is to study the dependence of performance, defined as execution time for example, on these parameters. This is useful in many different scenarios, such as identifying scaling bottlenecks at higher core counts, identifying input parameters that result in the best performance, comparing characteristics of different applications, and tuning and redesigning applications for future systems.

With the increasing complexity of applications and machines, the number of parameters on which application performance depends has also increased. Multi-physics and multi-module applications are parameterized on tens of input parameters for the application itself. Use of resources such as accelerators, network, and filesystem adds many other parameters that can be tuned in some cases. Even for scenarios when certain parameters cannot be selected for certain resources, their impact needs to be modeled.

Below, we describe three different performance analysis studies and the corresponding data that will be used in Section V to evaluate the proposed framework.

A. Performance of Different Task Mappings

Performance of communication-heavy applications is often limited by contention for resources on the network. A complex interplay between factors such as message injection, routing strategies, and sharing of network resources makes it tricky to study their respective impacts on performance. In the first performance analysis study, we address this challenging problem by modeling the impact of network hardware components on execution time of applications.

Data description: In order to study network behavior and its impact on performance, we run the same application with different task mappings on Blue Gene/Q. This changes the layout of MPI processes on the 5D torus network and as a result the flow of traffic on the network. We run three benchmarks with common communication patterns (2D and 3D Halo, and all-to-all over sub-communicators) with two message sizes (16 KB and 4 MB) on two node counts (1024 and 4096). Different task-to-node mappings are generated using Rubik [10] for each pattern, which are then executed individually to collect network counters data and observed performance for each task mapping.

The network counters data includes parameters that are expected to have an impact on performance, such as, number of bytes transferred on each link, occupancy of router buffers, and delays observed by packets when queued in the router buffers. Other features such as number of hops traversed by each packet and length of injection FIFOs are computed analytically. Significant properties, such as average and maximum, are then computed for each of these parameters to create the input data set for this case study. We refer the reader to [4] for more details on how networking experts created the data set.

B. Performance of Different Linear Solvers

Many computational science and engineering (CSE) codes have compute kernels that solve sparse linear systems of equations repeatedly. Several numerical software packages exist that provide a suite of preconditioners and linear solvers (PC-LS). Typically, the domain expert is expected to choose the right PC-LS combination that performs best for the input problem/matrix at hand. Intuitively, there is a strong correlation between the properties of a matrix and the performance of a specific PC-LS combination. However, this relationship is not always obvious to a casual user, especially for matrices not seen before. Hence, we model the performance of different PC-LS choices with respect to properties of the input matrix.

Data description: We gather a training set of sparse matrices and compute features that we expect to impact performance [5]. This includes features that describe the structure of the matrix, its numerical properties, and the domain-specific information on how the matrix is generated, if available. We obtain the execution times to solve the linear equation represented by each matrix for various PC-LS combinations (13 preconditioners and 9 solvers in Trilinos [11]) on an Intel Xeon node.

For our performance study, we use matrices generated by MFEM [12], a finite element discretization library, when different discretization orders and refinement levels are used for four arbitrarily chosen problems. This generates a set of 879 symmetric positive-definite matrices that exhibit the typical challenges with matrices in PDE-based CSE applications. In our experiments, we consider randomly chosen 75% of matrices for training and the rest for validating our prediction model.

C. Performance of Proxy and Parent Applications

Hardware-software co-design often involves the use of proxy applications – simpler codes that capture the essence of specific performance-critical modules in a production application. Consequently, it is critical to understand which salient performance characteristics of a parent application are covered by a proxy and how well. One method for this analysis is to collect hardware performance counters for the parent and proxy application, and to compare the counters data and its impact on performance.

Data description: In this performance study, we use OpenMC [13], a production Monte Carlo (MC) particle transport simulation code and one of its proxy applications called XSBench. We compare the strong scaling behavior of XSBench and OpenMC on an Intel Xeon node using six different workloads. For each code, we collect hundreds of hardware performance counters available via PAPI that are grouped into different resource groups such as floating point unit, Branch unit, L1 cache, L2 cache, memory and others. These resources represent different architectural components that can impact application performance.

III. A Unified Analysis Framework for HPC Data

Automating performance analysis in HPC requires the design of targeted machine learning solutions that allow the incorporation of task-specific constraints and produce high-fidelity results as evaluated by user-defined quality metrics. A typical workflow for such analysis consists of the following steps: (i) identify data sources, and collect necessary and sufficient data, (ii) formulate the data and analysis into a machine learning task, (iii) find a suitable technique from the plethora of available machine learning solutions, (iv) fine-tune and select the model parameters, (v) build a qualitative understanding of the chosen black-box model (e.g. feature importances, correlations), and (vi) transform results from the analysis into domain knowledge.

The open-ended nature of the steps above makes it extremely challenging for a performance analyst to arrive at a satisfactory solution in a single iteration. Further, it is often not possible to transfer the insights and experience from one problem to another without making grossly inaccurate approximations to the analysis. This makes the process of data analysis difficult and time-consuming. In addition, inherent assumptions made by commonly adopted machine learning techniques are often neglected, thereby making it challenging to reduce uncertainties in the inferred model. Finally, several successful machine learning paradigms, such as deep neural networks, are not interpretable, thereby making the models highly opaque for the analyst. These obvious shortcomings in the HPC performance data analysis workflow strongly motivate the need for a unified and reusable approach that will facilitate robust model inferencing, while being compliant with the workflow needs. We present PADDLE, a unified machine learning framework to fulfill this need.

A. Overview of the PADDLE Framework

PADDLE is comprised of the following modules (Figure 1): (i) *Deep Feature Extraction*: feature extraction from raw data using deep information decomposition, (ii) *Model Design*: automated parameter tuning and robust model selection to trade-off between model complexity and generalizability, (iii) *Multi-source Analysis*: enables joint inferences when there are multiple data sources, e.g. correlation studies, and (iv) *Visualization*: allows exploration of the learned feature space and feature influences on the designed model.

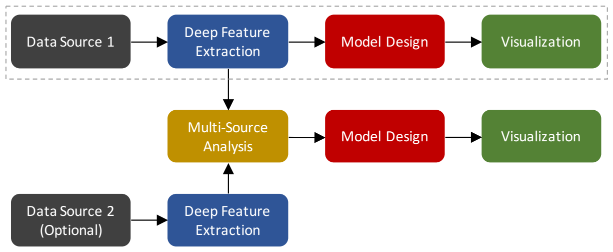
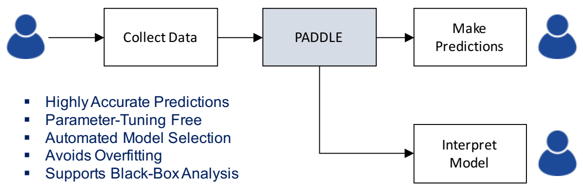


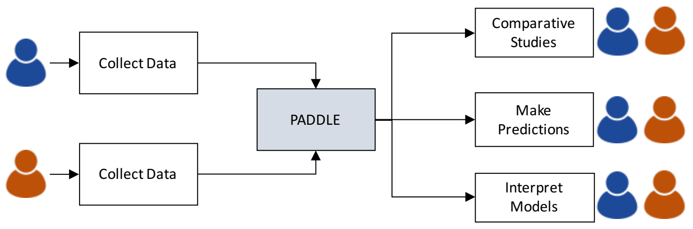
Figure 1. PADDLE – a generic machine learning framework for performance data analysis. The proposed framework is well-suited for a wide-range of problems that require the design of models for predictive or comparative analysis using one or more data sources. In addition to being robust and accurate, the resulting models are also easily interpretable.

The data source (black boxes in Figure 1) corresponds to the tuple $(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X}$ and $\mathcal{Y}$ denote the set of independent variables and predictive dependent variable respectively in the collected data. Note that, unlike several other machine learning pipelines, PADDLE supports continuous, discrete or categorical variables for both inputs and outputs. Similar to many state-of-the-art machine learning systems, a key component of PADDLE is the unsupervised extraction of latent features from high-dimensional input data (*Deep Feature Extraction* module, blue box). There is anecdotal evidence in recent ML literature that data-driven feature learning can provide much better insights compared to conventional feature selection techniques, e.g. lasso [6], commonly adopted by performance analysts. In addition to providing robust low-dimensional representations, the deep feature extraction step models the dependencies in $\mathcal{X}$ and disentangles the factors of influence for the subsequent model design step.

Given the set of inferred latent features, designing an effective and robust model, $f : \mathcal{X} \rightarrow \mathcal{Y}$, is central to performance analysis (*Model Design* module, red box). In addition to enabling prediction, such a surrogate model allows



(a) Single source data analysis



(b) Multi-source data analysis

Figure 2. User-centric workflows for using PADDLE in typical performance analysis studies in high performance computing.

analysts to gain intuitions about the entire space of $\mathcal{X}$. The large amounts of time typically spent by the analyst to find the right model and to fine-tune it for robust performance is significantly reduced with PADDLE. It achieves this by exploiting the hierarchical nature of the latent space to perform automated model selection and utilizes a noise-enhanced learning strategy to avoid overfitting. PADDLE also supports predictive analysis tasks that require the use of multiple data sources (e.g. data from multiple applications), i.e., to design a model $f : (\mathcal{X}_1, \mathcal{X}_2) \rightarrow \mathcal{Y}$, where $\mathcal{X}_1$ and $\mathcal{X}_2$ correspond to two data sources. This is done by the *Multi-source Analysis* module that uses the latent features from two data sources to jointly infer a model or perform correlation studies.

The final crucial component of PADDLE is the *Visualization* module (green box), that enables analysts to understand the inferred black-box models through a hierarchical visualization of latent features, which are in turn defined in terms of the independent variables in the data. PADDLE presents the composition of latent features using a force-directed graph layout, where the edge thickness indicates the strength of correlation. Further, the feature importances from the inferred model are hierarchically propagated to the independent variables in the data to highlight their influence on the prediction.

B. Using PADDLE: User-centric Workflows

Figure 2 demonstrates how an analyst can use PADDLE for analysis of single source and multi-source data. For example, Figure 2(a) shows a generic workflow for performance prediction studies, where the goal is to predict a performance metric (e.g. execution time) using a set of user-specified attributes from a single source (e.g. network counters). When using PADDLE, the user provides both independent and dependent variables, with the goal of inferring a robust and interpretable model. Subsequently, PADDLE performs deep feature extraction, automatically carries out model selection and produces the predictive model and model visualizations for further analysis. The network prediction case study in Section V-A is a prototypical example of this workflow.

Another common analytics workflow in performance analysis is to use multiple data sources for joint inference or comparative analysis (Figure 2(b)). In this scenario, PADDLE performs feature extraction from the two datasets independently, and then performs joint model inferencing to enable predictive modeling and correlation studies. In addition, the hierarchical visualization is flexible to allow comparative analysis of the features from the two sources. The linear solver prediction and performance coverage analysis case studies in Sections V-B and V-C respectively are different use-cases of this workflow.

IV. Description of Modules in PADDLE

In this section, we describe the different modules in the PADDLE framework, as illustrated in Figure 1.

A. Deep Feature Extraction Module

As the first step, PADDLE performs feature extraction from input data using deep information decomposition [14], [15]. Conceptually, this is similar to modern deep learning techniques that learn a sequence of filters to create concise representations [16], [17]. However, unlike deep neural networks, the proposed approach is effective even for small or moderate-sized data. Further, when compared to shallow feature learning strategies, e.g. independent component analysis, which are typically used for small-sized data, our method is more robust to data uncertainties and can exploit the inherent correlations between input variables. Finally, existing feature learning approaches do not generalize to discrete or categorical data, which is common in HPC. In contrast, our technique is applicable to both discrete and categorical variables.

Formulation: Denoting a set of multivariate random variables by $X = \{X_i\}_{i=1}^N$, dependencies between the variables can be quantified using the multivariate mutual information [18], also known as total correlation (TC), which is defined as follows:

$$TC(X) = \sum_{i=1}^{N} H(X_i) - H(X) \tag{1}$$

where $H(X_i)$ denotes the marginal entropy. This quantity is non-negative and zero only when all the X_i's are independent. Further, denoting the latent source of all dependence in X by Y, we obtain $TC(X|Y) = 0$. Therefore, in the feature extraction step, PADDLE solves the problem of searching for a feature Y that minimizes $TC(X|Y)$. Equivalently, we can define the reduction in TC after conditioning on Y as $TC(X;Y) = TC(X) - TC(X|Y)$.

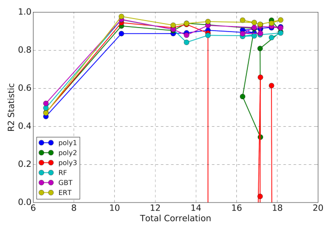
In our context, X corresponds to the set of independent variables in the data. The optimization begins with X, constructs Y_0 to maximize $TC(X;Y_0)$. Subsequently, it computes the remainder information not explained by Y_0, and learns another feature, Y_1, that infers additional dependence structure. This procedure is repeated for k layers until the entirety of multivariate mutual information in the data is explained.

B. *Model Design Module*

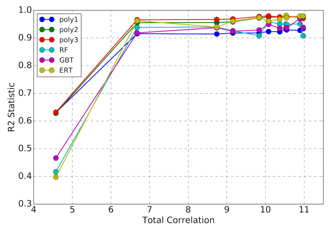
In this module, PADDLE uses the latent features to build predictive models. The primary challenges in this step include choosing a suitable ML technique and tuning its parameters. In existing practice, the prediction error on validation data is used as the guiding metric for these two tasks. Mathematically, prediction error can be decomposed into two competing components: *bias* and *variance*. As we increase the model complexity, the bias term reduces, while the variance becomes significantly larger. Similarly, the uncertainty of a model directly relies on the amount of data used to fit the model.

In addition to choosing parameters through cross-validation, PADDLE automatically analyzes the robustness of popular machine learning algorithms and suggests suitable models based on their bias-variance trade-off. More specifically, PADDLE utilizes the hierarchical representation from the *Deep Feature Extraction* module to evaluate the robustness of the learned models. It fits multiple models with increasing number of latent variables k, and evaluates their consistency with increasing amounts of information about the input data.

For example, Figure 3(a) illustrates the analysis carried out by PADDLE for execution time prediction with network data (Section II-A), wherein the prediction accuracies (R^2 statistic) obtained using different algorithms for increasing number of latent features are shown. The x-axis corresponds to the total correlation described by the set of latent features until that layer. As it can be observed, all methods progressively improve with more latent features (note: each marker corresponds to one additional layer). However, as we go beyond 5 layers, higher-order polynomial regression fails (poly2 and poly3), and only the ensemble methods demonstrate a stable performance, while simpler models such as linear regression (poly1) achieve lower accuracy. This corroborates the observation in [4], wherein the authors found through extensive empirical studies that ensemble methods produced the best performance for this problem.



(a) Analysis with raw data



(b) Robust modeling with noise-enhanced data

Figure 3. *Model Design* module – PADDLE analyzes the robustness of popular machine learning algorithms (polynomial regressors of order 1, 2, 3; Random Forests; Gradient Boosted Trees; and Extremely Randomized Trees) and automatically makes recommendations based on cross-validation.

This sudden performance degradation of complex models indicates the lack of non-robustness of the regression model, i.e. overfitting. PADDLE alleviates this crucial challenge by employing a noise-enhanced learning strategy. By adding uncorrelated noise to the input data, PADDLE regularizes the model design process [19]. Surprisingly, as shown in Figure 3(b), with the proposed robust feature learning, even a simple 2nd-order polynomial regression behaves consistently and matches the performance of sophisticated ensemble methods. Consequently, PADDLE tests the robustness of different algorithms, tunes parameters using cross validation and recommends the model that is highly accurate, robust and simple (for interpretability).

C. *Multi-source Analysis Module*

This is a critical component of PADDLE, designed to deal with multiple sources of data. More specifically, this module supports three functionalities that are applicable to several commonly encountered scenarios: (i) *Joint Feature Learning*: In cases where there is a need to combine features from different data sources, PADDLE enables inferencing of a joint feature space prior to invoking the *Model Design* module; (ii) *Mapping*: Learn a function mapping between two feature spaces extracted from two sources of data; (iii) *Comparison*: Direct comparison of deep features from two data sources using popular statistical metrics, e.g. KL-divergence.

D. *Visualization Module*

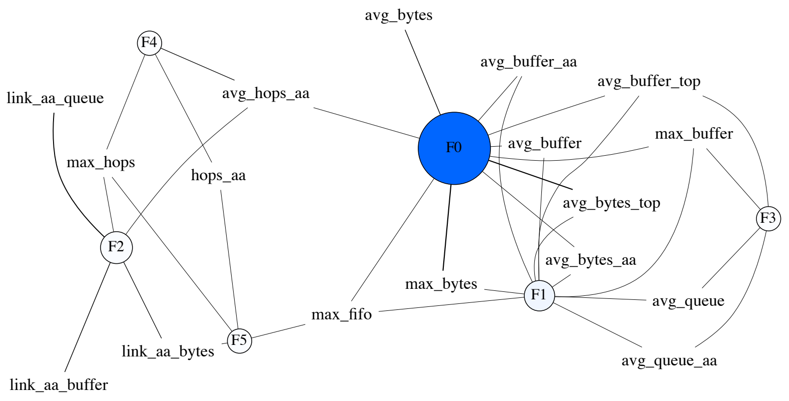
In addition to automating model design, PADDLE allows investigation of the learned models. This is achieved by visualizing the inferred latent feature space and their composition in terms of the input variables. The force-directed graph layout provides a holistic view of the input space. More specifically, the correlations between the variables, and the influence of the latent features on prediction are presented. For example, Figure 4(a) shows the latent space inferred by PADDLE for the network data analysis. The circles correspond to the learned features and their sizes indicate the amount of total correlation represented by each of those features, and they are colored by their importance in the actual prediction (darker shades of blue correspond to higher importance, white indicates not important at all). The edges between input variables and a feature indicate the feature composition, and the edge thickness indicates their contribution to the total correlation explained by that feature. Finally, to arrive at the influence chart in Figure 4(b), PADDLE uses a top-down belief propagation strategy that distributes the importance score of a feature to the variables based on their contribution to the total correlation.

V. Case Studies

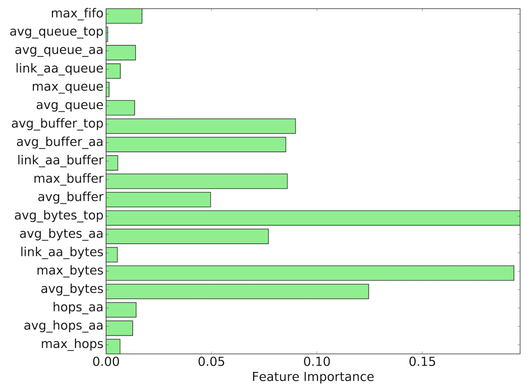
In this section, we present HPC performance analysis studies that use PADDLE, and compare the solutions with previous tailored approaches. In addition to being diverse in their mathematical formulations, these case studies demonstrate the usefulness of different modules in PADDLE. Figure 5 shows the algorithmic pipeline used by PADDLE for the analyses.

A. *Study I: Predicting Execution Time*

In this study, we use PADDLE to analyze the performance of different task mappings using the data described in Section II-A. The dependent variable in this case is the execution time, and the independent variables correspond to statistics from network counters and analytical formulations. As shown in Figure 5,

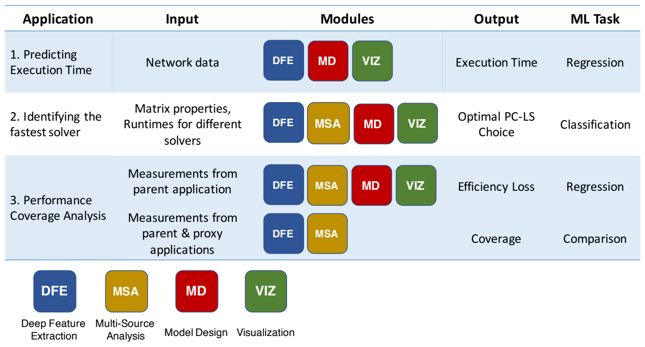


(a) Latent space visualization with the features colored by their influence on the prediction (darker shades of blue indicate more influence).



(b) Influence chart showing feature importance, obtained through hierarchical belief propagation.

Figure 4. Study I: Predicting the execution time of different task mappings in the network data using PADDLE.



Application	Input	Modules	Output	ML Task
1. Predicting Execution Time	Network data	DFE MD VIZ	Execution Time	Regression
2. Identifying the fastest solver	Matrix properties, Runtimes for different solvers	DFE MSA MD VIZ	Optimal PC-LS Choice	Classification
3. Performance Coverage Analysis	Measurements from parent application	DFE MSA MD VIZ	Efficiency Loss	Regression
	Measurements from parent & proxy applications	DFE MSA	Coverage	Comparison

Figure 5. Overview of analyzing different performance data using PADDLE.

this can be formulated as learning a regression function $f(.)$ to predict execution time. The workflow adopted by PADDLE for this problem is the basic single data source workflow from Figure 1. In [4], the authors extensively analyzed the performance of popular machine learning techniques for this problem and reported that ensemble methods, such as gradient boosting machines, produced the most accurate predictions. Despite their prediction capability, ensemble models are very complex for the analyst to understand and infer relationships between the different variables.

Approach: We follow the experiment setup used in [4], wherein 33% of the samples in the dataset are used for training the model, and the rest for evaluation. During the training stage, PADDLE automatically designs the optimal model for the task at hand. For an unseen test sample, PADDLE computes its latent features and makes a prediction with the designed regression model. Following common practice, we use the R^2 statistic and the mean absolute error (MAE) metric to measure the quality of prediction on the testing set.

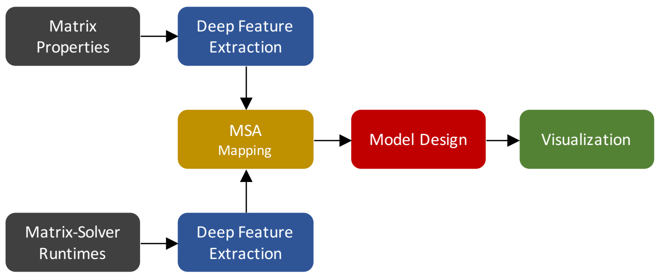
Results: The performance of different regression models, obtained using extensive parameter tuning, is listed in Table I and compared with PADDLE. As can be observed, polynomial regression models, directly operating on the raw data, overfit and completely fail, while ensemble models produce satisfactory performance. Interestingly, PADDLE outputs a 2nd-order polynomial regressor, by automatically trading off between model complexity and prediction, and achieves improved prediction. Note that, in contrast to a 2nd-order polynomial regressor operating on the raw data, the success of the same model in PADDLE can be directly attributed to the deep feature extraction and robust learning strategies. Figure 4(a) shows the visualization of the latent space inferred by PADDLE. It reveals that the feature F0 is the most influential when predicting execution time, and also captures the most total correlation in the input data. An analyst can quickly infer insights about the data from this graph layout. For example, the variable *max_fifo* is correlated to both *avg_queue* (F1) and *avg_bytes* (F0), and six features were sufficient to explain all mutual information in this 19-dimensional dataset. Finally, the hierarchical structure of the latent space enables propagation of influences to the individual input variables (Figure 4(b)).

TABLE I
PREDICTION ACCURACY COMPARISON FOR NETWORK DATA.

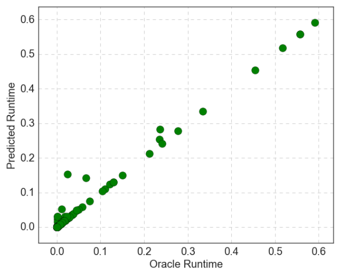
Method	R^2	MAE
Poly-2	0	0.697
Gradient Boosted Trees	0.96	0.012
Extremely Randomized Trees	0.955	0.013
PADDLE	**0.972**	**0.009**

B. *Study II: Identifying the Fastest Linear Solver*

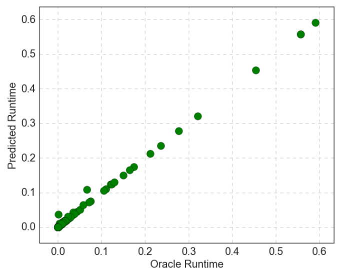
In this study, we want to predict the optimal preconditioner and linear solver (PC-LS) combination for solving a linear system (dataset described in Section II-B). As shown in Figure 5, this problem is formulated as a classification task, wherein each PC-LS combination is viewed as a class. In this case, the training data has two sources of measurements, (i) $\mathcal{X}_1$: properties of the input sparse matrix, and (ii) $\mathcal{X}_2$: execution times of different solvers for each matrix in the training set. However, for a test sample, we have access only to the matrix properties and not the latter. This necessitates multi-source analysis, wherein we learn a mapping between the two feature spaces, prior to model design for PC-LS prediction.



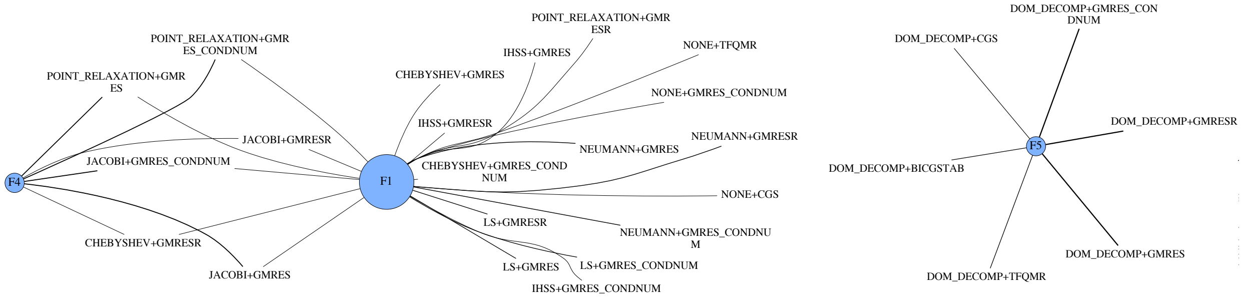
(a) Illustration of PADDLE workflow for linear solver prediction.



(b) Runtime of the predicted vs. oracle PC-LS (using *basic* features only).



(c) Runtime of the predicted PC-LS (using *basic* + *domain-specific* features).



(d) A subset of factors from the latent space inferred by PADDLE for the MFEM matrices dataset.

Figure 6. Study II: Using PADDLE to predict the best performing PC-LS choice for the MFEM dataset. In addition to producing highly accurate predictions compared to state-of-the-art algorithms, the latent space visualization reveals clusters of solvers in terms of their effectiveness to this data.

This workflow is described in Figure 6(a), which first infers the latent spaces for the two data sources, and subsequently uses the *Multi-source Analysis* module to learn the mapping between the two spaces.

The prediction problem essentially boils down to observing a row from $\mathcal{X}_1$ and predicting the "best" column from $\mathcal{X}_2$. Existing out-of-the-box classifiers do not associate similarly good PC-LS choices, and, consequently, fail to distinguish reasonably good choices from extremely poor ones. This provides no protection against very expensive mistakes (e.g., identifying the slowest PC-LS as the fastest), because a classification model is not sensitive to the actual runtime values. In contrast, PADDLE enables us to encode the entire performance profile in the latent space obtained using $\mathcal{X}_2$, thereby making more reasonable choices even when the optimal solution is not picked.

Approach: PADDLE first identifies latent features, $\widehat{\mathcal{X}}_1$ and $\widehat{\mathcal{X}}_2$ respectively. It is expected that matrices with similar properties, should group together in $\widehat{\mathcal{X}}_1$ and matrices that are close to each other in $\widehat{\mathcal{X}}_2$ are expected to exhibit similar execution times with regard to different PC-LS combinations. Since both $\widehat{\mathcal{X}}_1$, and $\widehat{\mathcal{X}}_2$ are representations of the same matrices in different spaces, PADDLE automatically learns a mapping between these spaces to predict the solver performance. Formally, PADDLE learns the transformation $g : \widehat{\mathcal{X}}_1 \mapsto \widehat{\mathcal{X}}_2$.

Let $\mathcal{X}_1^t$ represent the matrix properties for the set of test samples. PADDLE computes its corresponding latent space representation resulting in $\widehat{\mathcal{X}}_1^t$. Subsequently, the learned mapping g is used to transform these latent features into $\widehat{\mathcal{X}}_2^t$. Finally, PADDLE uses a simple K-Nearest Neighbor classifier to make the prediction. Note that, for a neighbor it considers only the PC-LS choices whose performance is within 10% of the best execution time for that matrix.

Results: For evaluating PC-LS prediction quality, we use the *Absolute Relative Error* (ARE) metric, defined as follows:

$$ARE = \frac{|t_p - t_o|}{t_o} \qquad (2)$$

where t_p is the execution time for the predicted PC-LS choice, and t_o is the *oracle* execution time, i.e. the execution time of the best PC-LS, for an unseen test matrix. Compared to conventional metrics such as classification accuracy, this metric can reveal the worst-case behavior of the prediction methods. Table II shows the prediction performance obtained using state-of-the-art approaches compared to PADDLE, for the MFEM dataset (Section II-B). As it can be observed from the results, PADDLE outperforms existing hand-tuned solutions in the literature. Figures 6(b), (c) show that with robust model learning, the performance improvements obtained by adding domain-specific features is not significant.

Investigating the latent spaces of the two data sources can provide interesting insights into the data. For example, visualizing the latent space of $\mathcal{X}_2$ in Figure 6(d) reveals groups of PC-LS choices that exhibit similar performance characteristics. The constituent methods in feature F1 are characterized by the solver (GMRES and variants), with a wide-range of preconditioners. Whereas, the feature F5 reveals when the domain decomposition preconditioner is used, the actual linear solver has a lesser influence on the observed

TABLE II
COMPARISON OF THE DIFFERENT PREDICTION ALGORITHMS IN DETERMINING THE PC-LS CHOICE. IN EACH CASE, WE SHOW THE MEAN ABSOLUTE RELATIVE ERROR COMPUTED USING THE ORACLE EXECUTION TIMES FOR THE UNSEEN TEST MATRICES.

Method	Basic	Basic + Domain-Specific
SVM	1.58	0.92
k-NN	3.66	0.39
Yeom et al. [5]	0.58	0.31
PADDLE	**0.13**	**0.09**

performance. Note that, the performance characteristics of these choices are dependent on the implementation as well as the data and the numerical algorithms, thereby making insights from PADDLE highly valuable.

C. Study III: Performance Coverage of Proxy Applications

In this study, we use PADDLE to perform comparative studies between two applications based on their performance characteristics (dataset described in Section II-C). Figure 8(a) shows the typical representation of how proxy applications are compared against their parents. The X-axis shows how well a resource alone can predict the parallel efficiency loss of an application on a node. The Y-axis shows the coverage score of each resource for the proxy application.

Approach: This is an interesting use-case for PADDLE where the *Multi-source Analysis* module is utilized to deal with two data sources (corresponding to the parent and proxy applications), as well as multiple resource groups within an application. While the computation of resource importance is performed using the hardware counter data from the parent application, the deep features from both the applications are compared to obtain the coverage score. This is illustrated in Figure 7. Here, PADDLE extracts the deep features using the counter data corresponding to each of the resource groups, and combines these features to build a joint feature space by exploiting correlations across different resources. Subsequently, the *Model Design* module automatically builds a regression model to predict efficiency loss of the parent application. The feature influences returned by PADDLE are used as the resource importance scores.

PADDLE uses the comparison functionality in the *Multi-source Analysis* module to compute the coverage score for each resource group. As described earlier, this is done by using statistical metrics on the deep features from the two applications. PADDLE finally provides the following outputs – (1) the overall resource importance vs coverage chart; (2) automatically tuned model for runtime prediction; (3) hierarchical feature representations, i.e. joint features $\rightarrow$ resource-specific features $\rightarrow$ counter data. Looking at both the importance and coverage scores for the resource groups can effectively summarize the performance characteristics of the two applications.

Results: Figure 8(a) reveals that memory and the cache subsystem (L3, L2, L1) are the most important resources for OpenMC on Intel Xeon. Figure 8(b) presents the joint

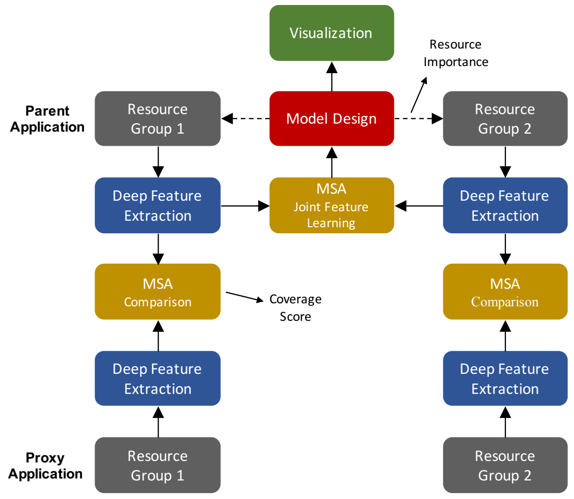


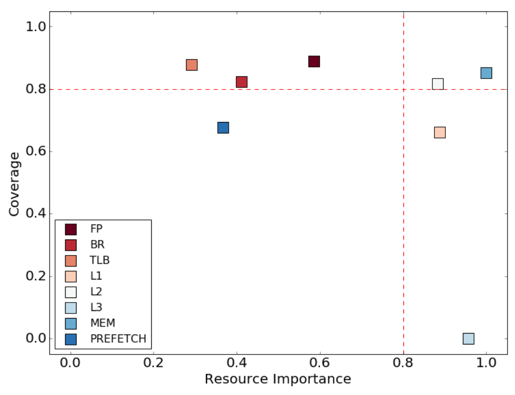
Figure 7. Illustration of the workflow for using PADDLE to perform coverage analysis of a proxy application with respect to a parent application.

feature space from PADDLE showing correlations across different resources and the importance of features in predicting performance (darker shade indicates higher influence). The joint latent space visualization reveals that the feature F3 is the most influential in describing efficiency loss of OpenMC. Further, this latent feature correlates to performance events corresponding to MEM_Y1, L3_Y1, L2_Y2, and L1_Y2. Here, for each of the resources, Yi corresponds to the resource-specific features inferred by PADDLE. In other words, all of these components have similar impact on the efficiency loss of OpenMC as the application scales on a node. This conforms with the observation in Figure 8(a) where MEM, L3, L2, and L1 get similar resource importances. Figure 8(a) also shows that only MEM is covered well, and others are not. This again conforms with the published coverage analysis of XSBench with respect to OpenMC on Intel Xeon in [6].

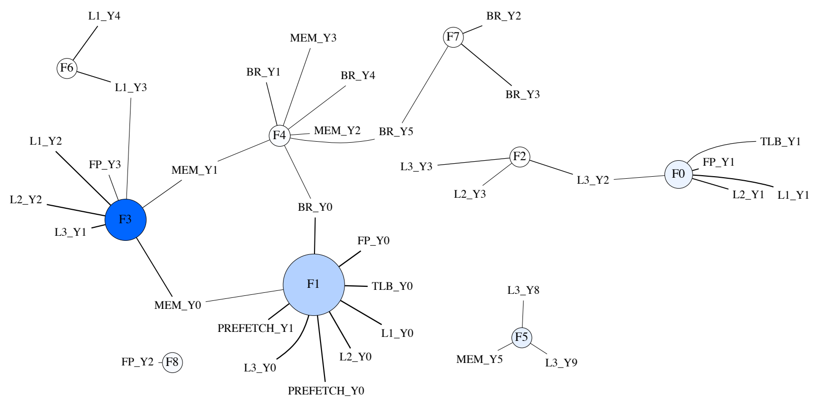
PADDLE also presents the counter-level correlation for each resource. Figure 8(c) shows the counter-level correlations for MEM. The size of the latent feature Y_0 shows that node prefetch, store, and last level cache misses contribute the most to the total correlation in the data. Since on current architectures, multiple components and complex interactions among them make it difficult to study the correlations among different performance events, adopting a principled framework such as PADDLE can provide valuable insights in that regard.

VI. RELATED WORK

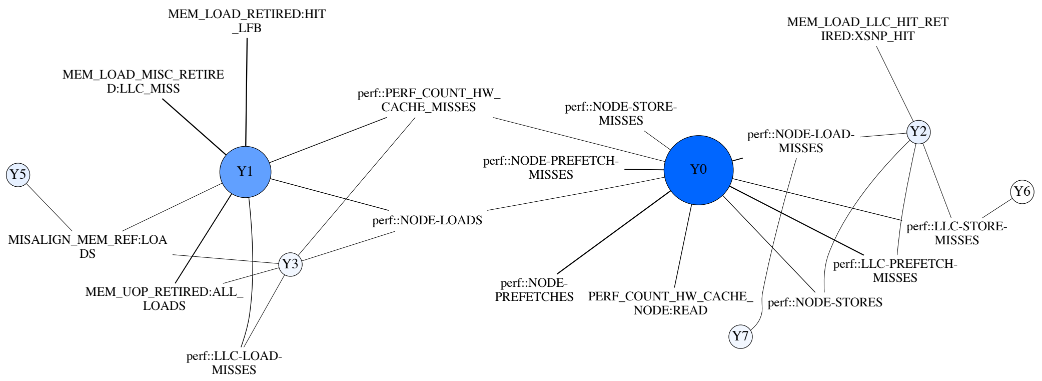
Most uses of machine learning in HPC involve applying supervised learning algorithms to automate challenging design choices, which would otherwise require significant manual effort. For example, Sukhija et al. [3] use supervised learning to select the best dynamic loop scheduling algorithm for parallel loops to improve on-node performance. Others utilize decision tree classifiers to identify the fastest parameter values for different loops in a code [26] and to associate performance metrics with program execution behavior [27]. Bhowmick et al. [2] first formulated the problem of choosing the optimal solver for a linear system as a classification task, and other



(a) Resource importance vs. coverage.



(b) Visualization of the joint latent space that combines the latent spaces from all resource groups (darker shades of blue indicate higher influence in performance prediction).



(c) Latent features for the MEM resource group, the most influential in performance prediction.

Figure 8. Study III: Coverage analysis of XSBench using PADDLE with respect to the OpenMC application.

TABLE III
COMPARISON OF THE PROPOSED FRAMEWORK WITH EXISTING POPULAR ML APPROACHES.

Features	**Dimensionality Reduction**	**Supervised Learning**	**Deep Neural Networks**	**Proposed Approach**
Hierarchical feature learning	×	×	✓	✓
Low dimensional structure inference	✓	×	✓	✓
Supports the use of multi-source data	×	×	✓	✓
Supports the use of categorical data	×	✓	×	✓
Interpretability of the inferred models	✓	✓	×	✓
Robustness to limited training data	N/A	✓	×	✓
Facilitates model selection	N/A	×	×	✓
References	[20], [21], [22]	[23], [24], [25]	[16], [17]	N/A

subsequent efforts, such as the Lighthouse taxonomy system, have developed effective classification pipelines to do the same for sparse linear systems [28]. Each of these HPC problems deploys different machine learning algorithms and analysis pipelines, specifically tuned for the problem at hand, possibly with guidance from machine learning experts. Consequently, the resulting solutions cannot be easily generalized to other problems. The proposed analysis framework, PADDLE, is specifically designed to enable solution generalization and easy interpretability of the inferred models.

In Table III, we provide a comparison of PADDLE with some of the commonly used machine learning approaches in performance analysis and emphasize the gaps bridged by our approach. Note that a typical machine learning pipeline combines and tunes several different components for dimensionality reduction, supervised learning, visualization and interpretation. A unique feature of our approach is that it provides a unified strategy to formulate different performance analysis tasks using a common optimization algorithm. In addition to simplifying the design process for data analysis pipelines, PADDLE produces high-fidelity models that perform similar to or sometimes better than state-of-the-art solutions.

VII. DISCUSSION AND SUMMARY

With the increase in complexity and number of components present in HPC systems, performance analysis and interpretation of data are becoming more challenging. As a result, automated performance analysis using machine learning is becoming more commonplace in HPC. The existing practice of using machine learning techniques for performance analysis includes a number of steps that are common across different types of analyses such as performance prediction and comparative study. However, today's approaches suffer from a number of challenges such as lack of solution reusability, lack of understanding of the inherent assumptions made by commonly adopted machine learning techniques, and hard-to-interpret models that are highly opaque to the analyst. In order to address all these challenges, we presented a generic machine learning framework, PADDLE in this paper.

The presented data-driven learning framework provides a highly adaptable unified environment for conducting a wide-range of performance analysis tasks such as performance comparison, prediction, and extraction of correlation among different features. In addition to standard performance prediction, PADDLE supports performance studies across multiple data sources, leverages a hierarchical feature learning approach to provide parameter-tuning free modeling, and enables robust generation of highly interpretable models. We presented three performance analysis studies in this paper to demonstrate how analysis tasks from widely different domains can be formulated easily to leverage the analysis and visualization pipelines of PADDLE. Given that machine learning techniques are being used heavily for extracting information from data and all such tasks involve certain common steps, the principled approach in PADDLE is timely since it bridges the gap between the HPC performance analysis and data science communities.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-740303).

References

[1] K. A. Huck and A. D. Malony, "Perfexplorer: A performance data mining framework for large-scale parallel computing," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. IEEE Computer Society, 2005.

[2] S. Bhowmick, V. Eijkhouta, Y. Freund, E. Fuentes, and D. Keye, "Application of machine learning to the selection of sparse linear solvers," *Int. J. High Perf. Comput. Appl*, 2006.

[3] N. Sukhija, B. Malone, S. Srivastava, I. Banicescu, and F. M. Ciorba, "A learning-based selection for portfolio scheduling of scientific applications on heterogeneous computing systems," *Parallel and Cloud Computing*, vol. 3, no. 4, pp. 66–81, Oct. 2014.

[4] A. Bhatele, A. R. Titus, J. J. Thiagarajan, N. Jain, T. Gamblin, P.-T. Bremer, M. Schulz, and L. V. Kale, "Identifying the culprits behind network congestion," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*, ser. IPDPS '15. IEEE Computer Society, May 2015, LLNL-CONF-663150.

[5] J.-S. Yeom, J. J. Thiagarajan, A. Bhatele, G. Bronevetsky, and T. Kolev, "Data-dependent performance modeling of linear solvers for sparse matrices," in *Proceedings of the 7th International Workshop in Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, ser. PMBS '16, Nov. 2016, lLNL-CONF-704087.

[6] T. Z. Islam, J. J. Thiagarajan, A. Bhatele, M. Schulz, and T. Gamblin, "A machine learning framework for performance coverage analysis of proxy applications," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2016.

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

[8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[10] A. Bhatele, T. Gamblin, S. H. Langer, P.-T. Bremer, E. W. Draeger, B. Hamann, K. E. Isaacs, A. G. Landge, J. A. Levine, V. Pascucci, M. Schulz, and C. H. Still, "Mapping applications with collectives over sub-communicators on torus networks," in *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2012.

[11] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, "An overview of the Trilinos project," *ACM Trans. Math. Softw.*, vol. 31, no. 3, pp. 397–423, 2005.

[12] "MFEM: Modular finite element methods," mfem.org.

[13] P. K. Romano and B. Forget, "The OpenMC monte carlo particle transport code," *Annals of Nuclear Energy*, vol. 51, pp. 274–281, 2013.

[14] G. Ver Steeg and A. Galstyan, "The information sieve," in *International Conference on Machine Learning*, 2016, pp. 164–172.

[15] G. Ver Steeg, S. Gao, K. Reing, and A. Galstyan, "Sifting common information from many variables," *stat*, vol. 1050, p. 23, 2017.

[16] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 1096–1103. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390294

[17] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, Aug 2013.

[18] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

[19] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural Comput.*, vol. 7, no. 1, pp. 108–116, Jan. 1995. [Online]. Available: http://dx.doi.org/10.1162/neco.1995.7.1.108

[20] J. Shlens, "A tutorial on principal component analysis," in *Systems Neurobiology Laboratory, Salk Institute for Biological Studies*, 2005.

[21] P. Comon, "Independent component analysis, a new concept?" *Signal Processing*, vol. 36, no. 3, pp. 287 – 314, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0165168494900299

[22] A. Hyvrinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural Networks*, vol. 13, no. 45, pp. 411 – 430, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608000000265

[23] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Advances in Neural Information Processing Systems 9*, M. C. Mozer, M. I. Jordan, and T. Petsche, Eds. MIT Press, 1997, pp. 155–161. [Online]. Available: http://papers.nips.cc/paper/1238-support-vector-regression-machines.pdf

[24] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Annals of Statistics*, vol. 28, p. 2000, 1998.

[25] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1010933404324

[26] D. Beckingsale, O. Pearce, I. Laguna, and T. Gamblin, "Apollo: Reusable models for fast, dynamic tuning of input-dependent code," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2017, pp. 307–316.

[27] W. Yoo, K. Larson, S. Kim, W. Ahn, R. Campbell, and L. Baugh, "Automated fingerprinting of performance pathologies using performance monitoring units (pmus)," in *Proc. of USENIX Workshop on Hot topics in parallelism. USENIX Association*, 2011.

[28] B. Norris, S. Bernstein, R. Nair, and E. R. Jessup, "Lighthouse: A user-centered web service for linear algebra software," *CoRR*, vol. abs/1408.1363, 2014. [Online]. Available: http://arxiv.org/abs/1408.1363