PRIONN: Predicting Runtime and IO using Neural Networks

Michael R. Wyatt II
University of Delaware
Newark, Delaware
mwyatt@udel.edu

Stephen Herbein
University of Delaware
Newark, Delaware
sherbein@udel.edu

Todd Gamblin
Lawrence Livermore National Laboratory
Livermore, California
gamblin2@llnl.gov

Adam Moody
Lawrence Livermore National Laboratory
Livermore, California
moody20@llnl.gov

Dong H. Ahn
Lawrence Livermore National Laboratory
Livermore, California
ahn1@llnl.gov

Michela Taufer
University of Delaware
Newark, Delaware
taufer@udel.edu

ABSTRACT

For job allocation decision, current batch schedulers have access to and use only information on the number of nodes and runtime because it is readily available at submission time from user job scripts. User-provided runtimes are typically inaccurate because users overestimate or lack understanding of job resource requirements. Beyond the number of nodes and runtime, other system resources, including IO and network, are not available but play a key role in system performance. There is the need for automatic, general, and scalable tools that provide accurate resource usage information to schedulers so that, by becoming resource-aware, they can better manage system resources.

We tackle this need by presenting a tool for Predicting Runtime and IO using Neural Networks (PRIONN). PRIONN automates prediction of per-job runtime and IO resource usage, enabling IO-aware scheduling on HPC systems. The novelty of our tool is the input of whole job scripts into deep learning models that allows complete automation of runtime and IO resource predictions. We demonstrate the power of PRIONN with runtime and IO resource predictions applied to IO-aware scheduling for real HPC data. Specifically, we achieve over 75% mean and 98% median accuracy for runtime and IO predictions across 300,000 jobs from a real HPC machine. We combine our per-job runtime and IO predictions with queue and system simulations to predict future system IO usage accurately. We predict over 50% of IO bursts in advance on a real HPC system.

KEYWORDS

Convolutional Neural Network, IO-Aware Scheduler, IO Prediction

ACM Reference Format:
Michael R. Wyatt II, Stephen Herbein, Todd Gamblin, Adam Moody, Dong H. Ahn, and Michela Taufer. 2018. PRIONN: Predicting Runtime and IO using Neural Networks. In *Proceedings of 47th International Conference on Parallel Processing (ICPP 2018).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3225058.3225091

ICPP 2018, August 13–16, 2018, Eugene, OR, USA

ACM ISBN 978-1-4503-6510-9/18/08…$15.00
https://doi.org/10.1145/3225058.3225091

1 INTRODUCTION

This paper tackles the problem of current batch schedulers on HPC systems being unaware of resources, such as IO and network bandwidth, when allocating resources to jobs. When submitting HPC jobs, users submit job scripts to a batch system with requests for compute resources (i.e., number of nodes and cores) for a period of time. Consequently, current batch schedulers have access to and use only information on the number of nodes and time for their job allocation decision, omitting the fact that jobs still contend for other resources. For example, co-scheduling many IO-intensive jobs can cause IO contention and underutilization of other resources, ultimately degrading performance (i.e., execution slowdown) as the jobs compete for access to the parallel filesystem. In general, including resource-awareness in schedulers, by considering a broader range of resources, such as IO and network bandwidth, can solve contention and underutilization problems: jobs can be scheduled so that resource usage is known and balanced [5, 10].

Clearly, current resource usage requests (i.e., number of nodes and time) are insufficient to achieve resource-aware scheduling. Delegating the specification of resource usage to users is not a feasible solution either. Analysis of job traces in HPC centers shows how user-requested runtimes are often overestimated. For example, user-requested runtimes for nearly 300,000 jobs on the Cab cluster at Lawrence Livermore National Laboratory in 2016 had a mean error of 172 minutes (24% relative accuracy). Current batch scheduler policies that terminate jobs when they exceed the user-requested time are often the cause for runtime overestimation. Furthermore, when extending the type of resources considered in scheduling decisions to include IO and other resources, users in scientific computing do not have an accurate understanding of jobs' requirements for a given HPC system, and cannot be expected to learn how to accurately estimate the associated resource usage. If schedulers had access to accurate estimates of jobs' resource usage, they would better deal with system resource contention and increase scientific throughput. This need can be addressed by predictive methods that can accurately obtain the per-job resource usage information necessary for resource-aware scheduling. In this paper, we present PRIONN (Predicting Runtime and IO using Neural Networks), a tool to accurately predict per-job runtime and IO bandwidth. We use the predictions to enable IO-awareness in a real HPC scheduler when it allocates resources to jobs.

The need for accurate resource usage estimates was addressed in previous efforts with traditional machine learning models (e.g., Decision Tree and k-Nearest Neighbors) that were mainly used to predict runtime [3, 7, 19–21] and, in some cases, more general resource usage [11, 12, 16]. This previous work has shown that machine learning models can provide more accurate runtime estimates than users [20]. However, previous work also required development and maintenance of parsers to extract features from diverse sets of jobs scripts for input into machine learning models. Therefore, when relying on simple parsers, the previous efforts have failed to generalize (i.e., different types of scripts require different parsers). Moreover, any effort to write general parsers incurs the cost of truncating and removing unique information present only in subsets of job scripts. In other words, not all information from job scripts can be captured by a parser.

Contrary to the traditional ML techniques listed above, deep learning models can automatically detect important patterns in data without truncating and removing text specific to a given script. Therefore, deep learning models remove the need for any manual parsing and feature extraction. This capability has been demonstrated thoroughly in the past decade with the application of Convolutional Neural Networks (CNNs) where traditional models have failed, such as in image recognition and text classification [6, 8, 9, 17, 18]. The challenge of using deep learning models, such as CNNs, when dealing with predicting resource usage from job scripts is how to transform non-image data (i.e., the job scripts) into image-like data suitable for CNNs. We tackle and solve this challenge with PRIONN, our automatic, general, and scalable tool for Predicting Runtime and IO using Neural Networks. PRIONN relies on a novel, direct, and quick data mapping method for job scripts that allows us to exploit the power of deep learning models to provide accurate per-job runtime and IO resource (i.e., per-job IO bandwidth usage) predictions. PRIONN is unique in that it concurrently transforms entire job scripts into image-like data, and by doing so it removes the need for script-specific parsing and feature extraction. We feed the image-like representation of job scripts into a 2D CNN, and by doing so we unleash the power of deep learning models to accurately estimate job resources from whole job scripts. The PRIONN automatic workflow (i.e., mapping and feeding into CNN), when integrated into HPC schedulers, provides us with a general and scalable solution for runtime and IO resource predictions.

Figure 1 shows the integration of PRIONN into a real HPC scheduler. The integration has two main phases: (1) the per-job runtime and IO resource predictions with PRIONN and (2) the use of the predictions in IO-aware scheduling on real HPC systems. In the first phase, PRIONN reads job scripts from recently completed jobs and concurrently maps the text of each job script into an image-like data representation. Then, PRIONN trains a deep learning model (i.e., a 2D CNN) by feeding these representations into the model. PRIONN uses the trained CNN model to predict per-job runtime and IO resource of newly submitted jobs in the system queue. In the second phase, we apply the per-job runtime and IO resource predictions from PRIONN to a scenario with an IO-aware scheduler and data from a real HPC system. We use a system simulator with the per-job runtime and IO resource predictions to forecast future system IO and IO bursts.

The contributions of this paper are as follows:

- The PRIONN components used to leverage a neural network to interpret whole job scripts and predict per-job runtime and IO resource in HPC systems;
- The evaluation of PRIONN's ability to outperform traditional machine learning techniques in accurately predicting per-job runtime and IO resources; and
- The application of PRIONN's predictions (i.e., per-job runtime and IO resources) to capture system IO and IO bursts for an IO-aware scheduler.

Our results show how PRIONN achieves over 75% mean and 100% median accuracy for predictions of per-job runtime and IO resources across nearly 300,000 jobs from a real HPC machine. We inject PRIONN's predictions into an IO-aware scheduler that manages a real HPC system. Because of PRIONN, the system's scheduler becomes aware of system IO and predicts over 50% of IO bursts (i.e., times of IO contention) in advance.

The remainder of this paper is organized as follows: In Section 2 we describe the components and novelty of the PRIONN workflow for predicting per-job runtime and IO resources, and in Section 3 we evaluate PRIONN using real HPC data to predict per-job runtime and IO resources. Section 4 describes how we use per-job runtime and IO resource predictions to achieve IO-aware scheduling with data from a real HPC system. Section 5 presents related work, and Section 6 summarizes our main results and future work.

2 PREDICTING RESOURCE USAGE

In this section, we describe PRIONN and how PRIONN components (e.g., the data mapping technique and deep learning model) were chosen. We also demonstrate how our tool creates accurate job runtime and IO resource predictions without manually extracting features from job scripts. Figure 2 shows the steps of predicting jobs' resource usage with PRIONN (top) and, for the sake of comparison, with traditional machine learning methods (bottom). The PRIONN tool comprises three steps: (1) data processing, (2) machine learning model, and (3) training and prediction. PRIONN executes the three steps on a single dedicated node of the Surface cluster at the Lawrence Livermore National Laboratory. Each node on Surface has 16 computational cores and two NVIDIA Tesla K40 GPUs. The data processing is performed on a single core; the training and prediction are performed on the node's two K40 GPUs. These overall steps are performed asynchronously to the scheduling of jobs on the production cluster.

2.1 Job Script Data Processing

Data processing is needed to transform raw text from job scripts into data that can be input into machine learning models. A novel component of our tool is the mapping of job scripts to an image-like data representation (i.e., one image-like representation per job script). Each image-like representation of a job script is composed of pixels which are mapped from text characters in the job script. Because of our mapping, whole job scripts can be input into deep learning models. Job scripts must be cropped and padded to a fixed size because deep learning models require a constant size input. We describe this process in Section 2.4. We consider two types of mapping: we map each job script into either a 1D sequence or a

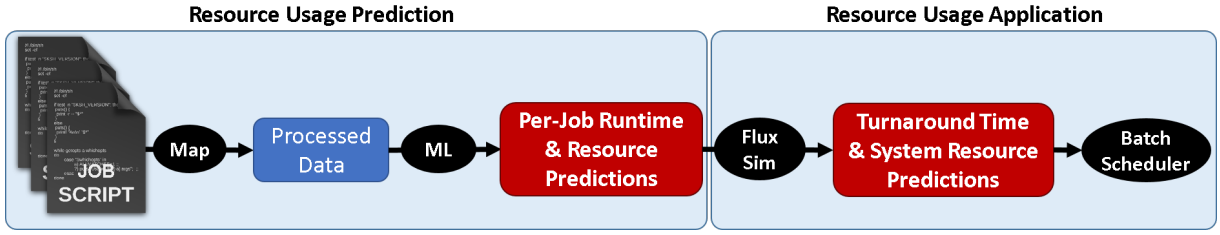


Figure 1: Overview of the PRIONN components and its application to next-generation HPC schedulers.

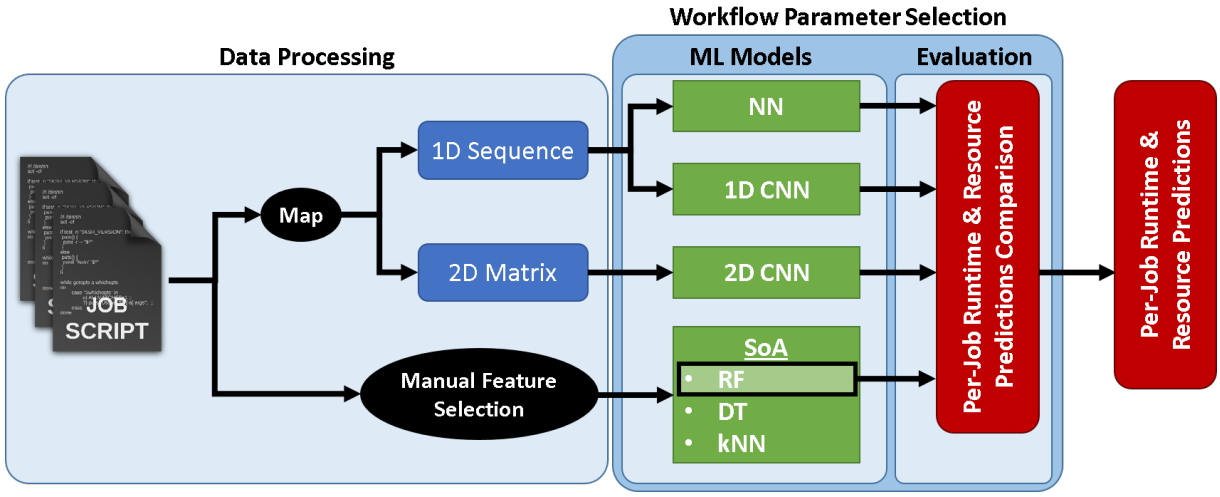


Figure 2: Overview of the PRIONN tool for obtaining jobs' resource usage predictions from HPC job scripts.

2D matrix of pixels (i.e., numerical values). Each pixel is mapped from a character in the job script. When mapping job scripts to a 1D sequence, the job script is flattened such that all lines of the text are concatenated into a single line before mapping the characters to pixels. Mapping to a 2D matrix, on the other hand, preserves the structure of the job script text (i.e., the location of characters in the job script are not altered).

The mapping of characters to pixels is performed on a separate node from the batch scheduler to avoid interference with the critical path of the resource scheduler. The mapping can be performed with four different transformations: binary, simple, one-hot, and word2vec. The *binary character transformation* method is a lossy transformation that converts each character to a binary value. All space characters (i.e., space, tab) are assigned the value "0" and all non-space characters are assigned the value "1". The *simple character transformation* method is a lossless transformation that converts each unique character to a unique value. We convert a standard ASCII character from the job script text file to a unique integer value. The *one-hot character transformation* method is a widely used lossless transformation that converts each unique character to a unique 128 value vector. Each vector has exactly 1 scalar with value "1" and the remaining are "0". The *word2vec transformation* method is a lossless transformation that converts each unique character to a unique 8 value vector. We use Google's word2vec method to obtain this transformation [13]. This method examines the context of a character (i.e., surrounding characters) to embed information about that character in a multidimensional vector.

Independent from the method to map characters into pixels, our data mapping gives PRIONN three advantages over manual feature extraction found in previous resource prediction work [2, 3, 7, 12, 16, 19, 20]. First, PRIONN is automatic and can be deployed as-is on any HPC system. Second, our tool is scalable for the increasing number of HPC jobs and HPC systems in that it does not require ongoing maintenance as job scripts and HPC environments change. Third, our tool is general to any HPC system. Any type of job script can be processed with our mapping to produce data which can be used with a deep learning model.

Contrary to our approach, traditional machine learning models for resource predictions such as Random Forest (RF), Decision Tree (DT), or k-Nearest Neighbor (kNN) rely on manual feature extraction from job scripts: specific features (lines) in job scripts must be identified, parsed, and transformed into data usable with machine learning models [12, 16, 20]. Consequently, a priori knowledge of the job scripts' structure (e.g., number and type of lines) and which features are indeed useful for resource usage prediction are necessary for traditional machine learning models. Job script features (e.g., user, requested time, and submission directory) are identified and parsed from the job scripts using custom parsing scripts. Features containing string data (e.g., user and application name) must be further processed into numerical data using methods such as

Feature Name	Feature Description
Requested Time	User-requested runtime in hours
Requested Nodes	User-requested number of computation nodes
Requested Tasks	User-requested number of tasks
User	User login ID
Group	User login group
Account	User account (i.e., bank)
Job Name	User specified job name
Working Directory	User specified directory for job execution
Submission Directory	Directory from which user submitted job

Table 1: List of manually extracted features from job scripts used with traditional machine learning models for comparison to our tool, PRIONN. These features are based on those described in the paper from the Smith and co-authors in [19, 20].

bag-of-words or label encoding. Features containing numerical data may also require some sort of processing (e.g., date values must be converted to epoch-seconds).

For the sake of a fair comparison, we manually extract features from our dataset of job scripts and use the extracted information with three traditional practice machine learning models (i.e., RF, DT, and kNN), which have been used for resource predictions in other work [12, 16, 20, 21]. To this end, we replicate the manual feature extraction in [19, 20]. We create a custom parsing script for our dataset of job scripts which captures features. This task proved difficult due to inconsistencies in job script format, demonstrating that this method is not ideal for deployment on real HPC systems. The complete list of parsed features is in Table 1. We use a label encoder to transform each parsed feature into a numerical value in which we assign a unique integer to each unique string value.

2.2 Machine Learning Models

Selection of a machine learning model for PRIONN is driven by the trade-off between prediction accuracy and prediction costs (i.e., training time). These properties are orthogonal: if a prediction method is highly accurate but slow to produce predictions, delays will make the prediction useless for the batch scheduler. We define our tool to be as accurate as possible without impacting the performance of the scheduler. We assess a diverse set of machine learning models for our image-like data mapping and for the manually extracted features. For our mapped data, we assess three deep learning models into which we can ingest our image-like representation of the job scripts: fully connected Neural Network (NN), 1D Convolutional Neural Network (1D-CNN), and 2D Convolutional Neural Network (2D-CNN). For the manually extracted features, we test three low cost, high-impact machine learning models: Random Forest (RF), Decision Tree (DT), and k-Nearest Neighbors (kNN).

With the image-like representation of our job scripts, we train deep learning models for predicting runtime and resource usage. With deep learning, the many hidden layers of neurons are able to automatically detect important features and patterns from sets of characters in job scripts. Simpler machine learning algorithms, such as kNN, DT, and RF, are not able to build features from sets of characters and are not suited to analyze our mapped data. The NN uses a 1D sequence of the mapped data as input and contains many fully connected hidden layers. The 1D-CNN also uses a 1D sequence of the mapped data job scripts as input and contains several 1D convolutional layers followed by several fully connected hidden layers. The 2D-CNN uses a 2D matrix of the mapped data job scripts as input and contains several 2D convolutional layers followed by several fully connected hidden layers. The deep learning models are classifiers, and each node in the final output layer is associated with a value or range of values predicted for resource usage (e.g., for runtime predictions, the output layer is 960 nodes in size where each node is associated with a runtime in minutes between 0 and 960 minutes).

In general, the manually extracted features in Table1 are suitable for the many traditional machine learning models from previous resource prediction work (a detailed description of the related work is provided in Section 5). From an implementation point of view, in our work we use regression versions of kNN, DT, and RF available from the *scikit-learn* Python library [15]. Each of these models uses the manually extracted features as inputs, and the outputs are predictions for runtime and resource usage of individual jobs.

2.3 Training and Evaluation

To substantiate the selection of the best deep learning model for PRIONN and quantify the prediction accuracy to cost tradeoff, we mimic a scheduling system in which jobs are submitted to a batch scheduler's queue with the same frequency found on a high-end cluster. Specifically, in this work, we simulate the Cab cluster at Lawrence Livermore National Laboratory. We perform training and prediction at the time of job submissions. The submission, start, and end times of real HPC jobs are used to replicate the queuing and execution of jobs on the HPC cluster with its SLURM batch scheduler. Historical jobs' data (i.e., jobs that have already executed) are used to train each machine learning model used in this paper. The trained model is then used to predict the resource usage of jobs as they are submitted to the batch scheduler. In our emulation of the real scheduling system, prediction of job runtime and IO resource occurs at the same time as the job submission to the batch scheduler. After every 100 job submissions, models are retrained with the newest historical job data (i.e., jobs that have recently completed). We train each model on the 500 most recently completed jobs. Our empirical evaluation of training with data from 50 up to 5,000 jobs indicated that there is minor improvement of prediction accuracy and higher cost to train beyond 500 jobs for PRIONN. The low training size is unusual for most deep learning tasks, but PRIONN's models are retrained rather than re-initialized after each 100 submitted jobs. Learned parameters in the model pass to subsequent models; thus knowledge is retained across several training events. This characteristic of deep learning models is not present in traditional machine learning models.

When evaluating prediction models, we first consider the accuracy of the predicted resource usage. Then, we consider the time

needed to obtain the resource usage prediction. We compare predicted resource usage to the actual resource usage for each job with relative accuracy. Relative accuracy is preferred over absolute error because it mitigates the negative impact of small prediction error for jobs with high resource usage. For example, a runtime prediction error of 30 minutes is far worse for a one-hour job than for a twelve-hour job. Equation 1 shows how we calculate relative accuracy for each resource usage prediction, where *true* is the actual value and *pred* is the predicted value. The ϵ value in the denominator (i.e., machine epsilon) prevents division by zero when both *true* and *pred* are 0. We use the maximum value between *true* and *pred* in the denominator of Equation 1 for two reasons: (1) to maintain a range of $[0, 1]$ for the metric and (2) to penalize underprediction more than overprediction. We do this because underpredicted resource usage (e.g., we predict IO of 10 MB/s for a job that uses 25 MB/s) fed to an IO-aware scheduler results in resource contention.

$$relativeAccuracy = 1 - \frac{|true - pred|}{max(true, pred) + \epsilon} \quad (1)$$

We use a dataset of real HPC jobs to train and evaluate resource usage predictions. The dataset contains information for 295,077 jobs from Lawrence Livermore National Laboratory. The jobs were executed on the Cab supercomputer, which has 1,296 nodes and a maximum runtime of 16 hours; it is connected to a Lustre parallel file system. The dataset consists of job scripts, execution data, and resource usage data for each job. From the nearly 300,000 jobs in our dataset, 111,596 jobs are unique (i.e., the job script is unique). The jobs were submitted by 492 users running a large range of different scientific applications. The dataset exhibits a wide range of resource usage for both runtime and IO of jobs. A total of 29,291 jobs were either canceled by the user or removed from the system before executing. We exclude these jobs from our analysis, giving a total of 265,786 jobs and 97,361 unique job scripts.

2.4 Optimal Prediction Parameters

We define an optimal set of components (i.e., data mapping method and deep learning model) for our tool based on the tradeoff between accuracy and performance described in Section 2.2. We also determine the best traditional machine learning model for comparison to our tool. For each machine learning model used in this paper, we use the techniques described in Section 2.3 to train and evaluate runtime predictions. We use the data described in Section 2.3 for the evaluation of our resource usage predictions. We predict runtime down to one minute (i.e., real and predicted runtime are rounded to the nearest minute)

We find the optimal settings for PRIONN with a comparison of the four transformations of our data mapping method (i.e., binary, simple, one-hot, and word2vec) and a comparison of the three deep learning models (i.e., NN, 1D-CNN, and 2D-CNN) described in Sections 2.1 and 2.2. As discussed in Section 2.1, deep learning models require a constant size input. Motivated by the need to keep the data processing and training time low, we fix the job scripts to a standard size of 64 rows and 64 columns of characters before mapping the data to our image-like representation. Job scripts with less than 64 rows or columns of characters are extended to this size with space characters. Jobs larger than 64 rows or columns of characters are cropped to the correct size. Standardizing the size of job scripts incurs a small amount of data loss at the very end of the impacted scripts. Note that only 9.9% of jobs scripts contain more than 64 lines of text and 13.8% of text lines contain more than 64 characters.

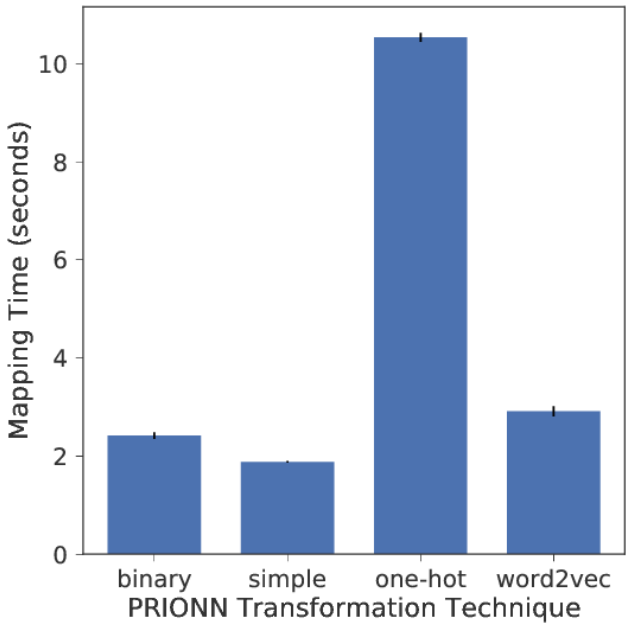


Figure 3: Time in seconds needed to transform 500 job scripts to 500 pixel-like representations by the four different transformations (i.e., binary, simple, one-Hot, and word2vect). The pixel-like representations are used for training the deep learning models.

We evaluate the time and accuracy of prediction for each data mapping method. Figure 3 shows the time necessary to process 500 job scripts (i.e., the number of jobs used each time we train a machine learning model) for each data mapping type. The one-hot transformation requires the most time, and the three other transformations require less than three seconds for 500 jobs. Each transformation maps characters to values that are either scalars or vectors and the vectors can vary in size. As a result, the time to train a deep learning model with data from each transformation method is different. Figure 4 shows the time to train a 2D-CNN for 10 epochs on 500 jobs using data from the four transformation methods. Our results indicate that one-hot requires the most amount of time for training, while the three other transformations take much less time. Figure 5 shows the accuracy for each transformation method and a 2D-CNN model, where word2vec outperforms the other three transformation types. The word2vec transformation provides the best combination of processing and training time with high prediction accuracy.

We evaluate the time and accuracy of prediction for each deep learning model considered for PRIONN. Figure 6 shows the time to train each of the deep learning models with the word2vec data mapping. The 2D-CNN is trained in less time than the NN and more time than the 1D-CNN. Figure 7 shows the runtime prediction accuracy

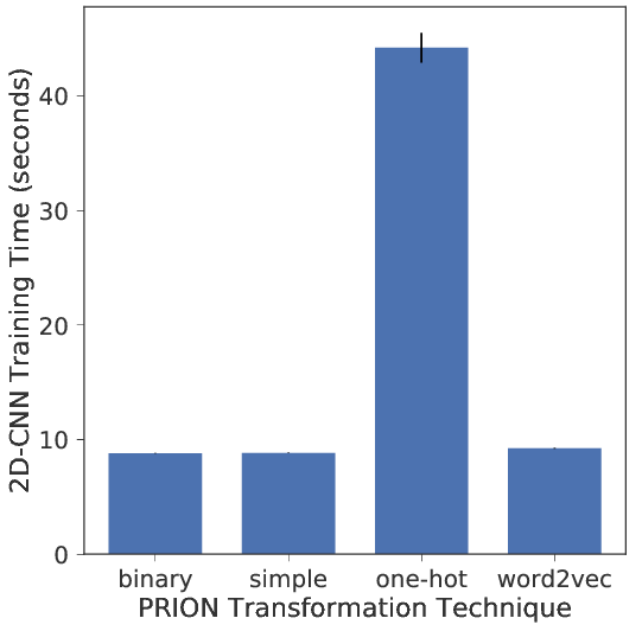


Figure 4: Time in seconds needed to train a 2D-CNN with each type of transformed data from our data mapping method.

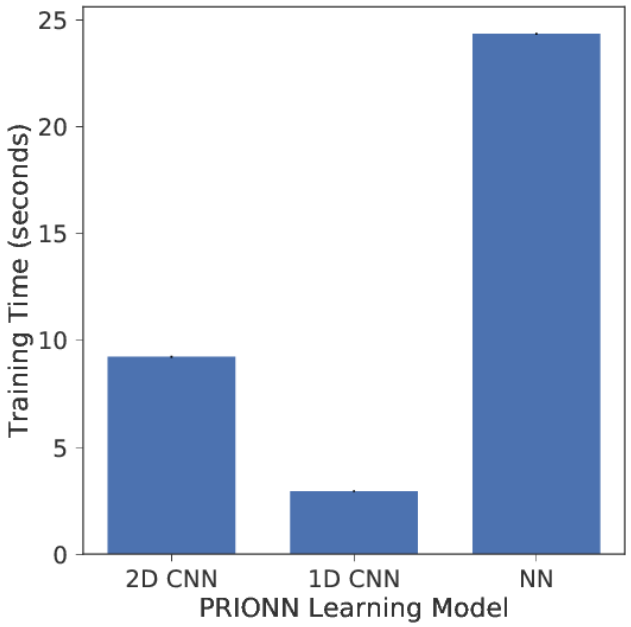


Figure 6: Time in seconds needed to train each of the tested deep learning models using the word2vec data mapping method.

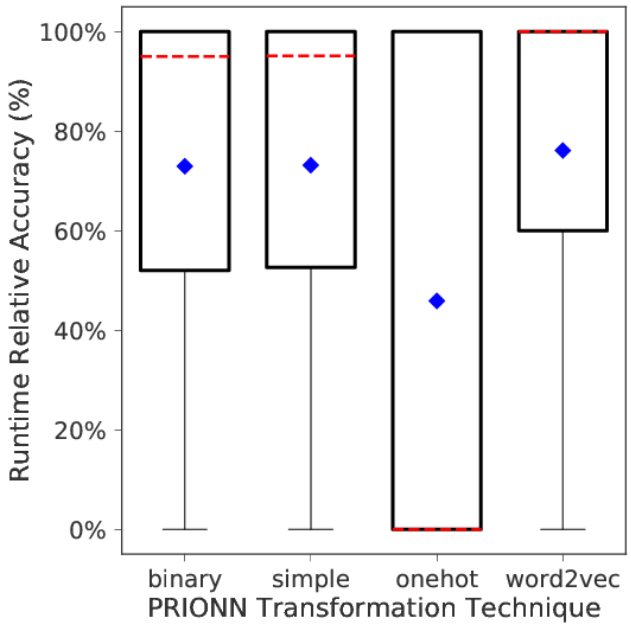


Figure 5: Distributions of relative accuracy for runtime predictions with each type of transformed data and a 2D-CNN.

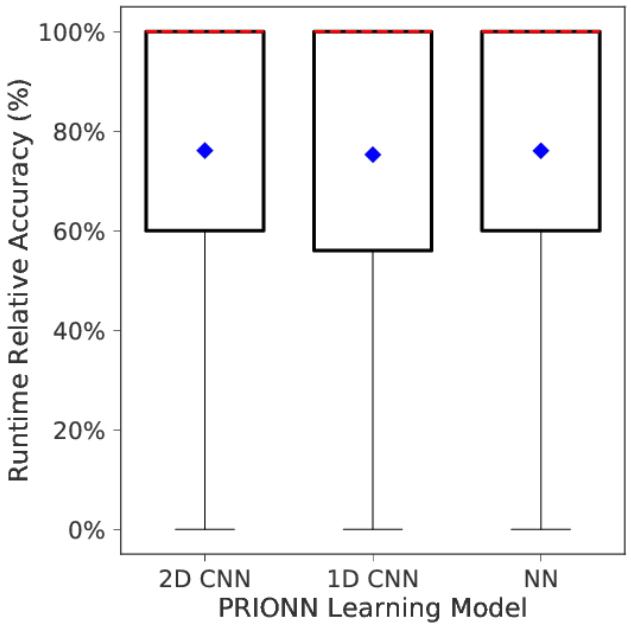


Figure 7: Distributions of relative accuracy for runtime predictions with each type of deep learning model and the word2vec data mapping.

using the word2vec transformation and our three deep learning models. The NN and 2D-CNN produce runtime predictions with higher accuracy than the 1D-CNN. These results indicate that the word2vec transformation and 2D-CNN model give the best results for our resource prediction workflow and are also fast enough to

be used with online training. We can reason that word2vec provided the best accuracy because each character is encoded with information about the context of that character. This information likely decreases the epochs needed to converge on an accurate deep learning model. Similarly, we can hypothesize that the 2D-CNN performs best because the convolutions on the 2D matrix of characters provide an efficient method to build features around patterns of jobs scripts' code (i.e., patterns among subsequent lines of code). Based on our analyses in this section, we use the word2vec transformation and the 2D-CNN deep learning model for our PRIONN tool. Specifically, PRIONN uses the word2vec algorithm with an output vector size of four and a 2D-CNN with four convolutional layers and four fully connected layers.

We predict the runtime of each job in our dataset with the manually extracted features in Table 1 and the three traditional machine learning models (i.e., kNN, DT, and RF) representative of previous resource usage predictions. We observe that each one of the three machine learning models has similar prediction accuracy; the RF slightly outperforms the other two with an average relative accuracy of 2% and 3% higher than DT and kNN respectively. We can speculate that the increased complexity of the RF explains why it performs better than the DT. Additionally, the method in which categorical features are encoded to numerical values is a likely explanation for the poor performance of the kNN, which relies on measuring the Euclidean distance between jobs. The time required to extract job script features and train each model is less than one second for 500 jobs (i.e., the training data size for each batch of predictions). Thus, we identify the RF to be the best performing model among traditional methods. We further access RF's accuracy towards previous work in [20] by replicating the work in that paper with our RF implementation and their datasets. Table 2 shows the accuracy from [20] and our RF. We achieve similar or better accuracy than reported in [20] for both datasets. Results in the table confirm that (1) the RF is the best machine learning model for extracted features and (2) the RF achieves similar or better runtime prediction accuracy compared to previous runtime prediction work. We use the RF as a representative of previous methods for comparison to our PRIONN tool in the remainder of this paper.

3 EVALUATION OF RESOURCE PREDICTIONS

We evaluate the data mapping and deep learning model of PRIONN with predictions for runtime and IO resources of real HPC jobs. To this end, we compare our predictions to the best traditional machine learning model predictions as well as user predictions (when applicable). Our results show how PRIONN provides better per-job prediction accuracy and thus is better suited for augmenting schedulers with the information necessary for IO-aware scheduling.

3.1 Per-Job Runtime Predictions

Our dataset of 295,077 jobs comes from real traces whose jobs were executed on the Cab cluster at Lawrence Livermore National Laboratory during 2016. Figure 8a describes our dataset in terms of the distribution of actual runtimes. Nearly half of the jobs have a runtime between 0 and 60 minutes. The mean job runtime is 44 minutes, and a small percentage of jobs have runtimes over three hours. Figure 8b shows the boxplots describing the relative accuracy for predicted job runtimes when using: (1) user requested time, (2) the best traditional machine learning model (i.e., RF), and (3) PRIONN. We observe how our method has a mean accuracy of 76.1%, an increase of 6.0% over RF. The median accuracy for our predictions is 100%, indicating that for over half of the jobs in our dataset, we correctly predict the runtime. High accuracy for runtime prediction is important for accurately predicting which jobs will be running at a future time on an HPC system [4]. Therefore, the increase in mean and median accuracy with PRIONN over the previous methods is substantial for prediction of system IO and IO bursts, which we demonstrate in Section 4. Note how the user predictions are substantially outperformed by PRIONN and the RF.

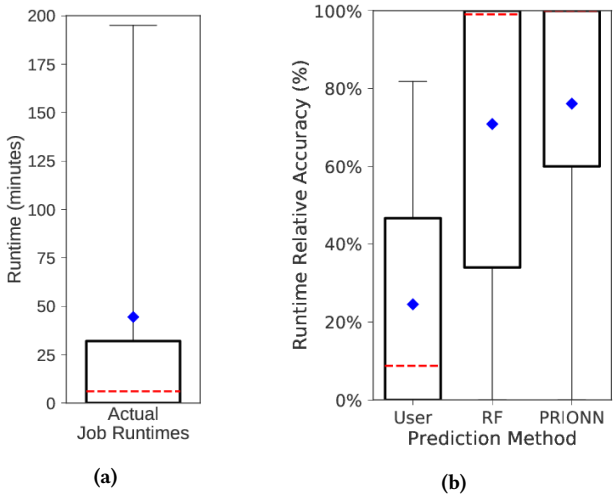


Figure 8: Distribution of actual runtimes for our data (a) and the relative accuracy for predicted job runtimes of user requested time, RF, and our method (b).

3.2 Per-Job IO Predictions

For the IO resource predictions, we use the same dataset as in the section above, and we predict total bytes read and total bytes written for each job. Because bandwidth is what an IO-aware scheduler uses [5], we then compute the read and write bandwidth from the predicted total bytes read and total bytes written. We deal with a diverse dataset of jobs in terms of their actual read and write bandwidth. Figure 9a shows the distribution of actual read and write bandwidth for our dataset. In the figure we observe how the mean bandwidth for read and write is orders of magnitude larger than the median, indicating a handful of jobs in our dataset have extremely large IO bandwidth compared to the majority of the jobs. We first predict the total bytes read and total bytes written for each job in our dataset with PRIONN and the RF. Note that users are not providing this information in their job scripts and thus, a comparison with the user predictions is not possible in this case. We then compute the bandwidth by dividing the total bytes read and written with the predicted runtimes of jobs. Figures 9b and 9c show the boxplots of the relative accuracy for predicted read and write bandwidth with RF and our tool respectively. The

Dataset	Data Size	Runtime MAE (minutes)	
		Smith, et al. [20]	Our Replication
SDSC95	76,840	59.65	35.95
SDSC96	32,100	74.56	76.69

Table 2: Mean Absolute Error (MAE) for runtime from the Smith and co-workers' paper and from the RF model considered in this paper as the best traditional method because of its accuracy.

cross-comparison of these two figures outlines how our method outperforms RF for both bandwidth values (i.e., read and write). Specifically, PRIONN predictions have a mean accuracy of 80.2% and 75.6% for read and write bandwidth, which is 12.1% and 9.6% higher than the RF predictions.

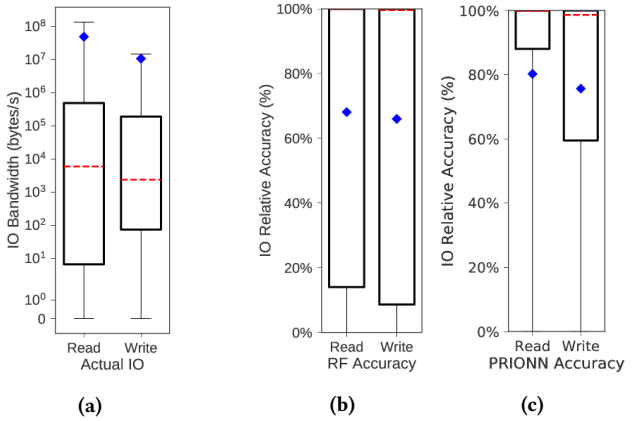


Figure 9: Distribution of read and write bandwidth for our dataset (a) and relative accuracy for predicted read and write bandwidth with RF (b) and PRIONN (c).

4 APPLICATION TO REAL HPC SYSTEMS

We demonstrate the value of per-job runtime and IO predictions from PRIONN to an IO-aware scheduler as part of the second phase of our workflow. Specifically, Figure 10 shows how we use our per-job runtime and IO predictions with the Flux open-source resource management framework simulator and its IO-aware scheduler [1] to predict system IO and IO bursts.

4.1 IO-aware Scheduler

An IO-aware scheduler relies on knowledge of IO behavior and potential IO contention to schedule jobs such that IO bandwidth contention is avoided. We leverage per-job runtime and IO usage predictions from PRIONN to build this knowledge and drive the IO-aware scheduler in its decision. To this end, we use the simulator of the open-source, next-generation job scheduler Flux to mimic the evolution of a high-end HPC system [1, 5]. We modify the simulator to use job runtimes that are predicted either by PRIONN, defined in Section 2, or by the user. Specifically, the scheduler uses our runtime predictions to estimate when a job starts and completes; it also combines the estimated job's start time with our IO predictions (made for the same job) to estimate the job's impact on the system IO. By incorporating the sum of all jobs' IO impacts, we can ultimately estimate future system IO (e.g., patterns including IO bursts).

4.2 Turnaround Time Prediction

The first step to an effective IO-aware scheduler is accurate turnaround time prediction (i.e., the amount of time between when a job is first submitted to the scheduler and when the job completes), as shown in Figure 10. Turnaround time is necessary because it provides insight to which jobs will be executing on the HPC system at future times. The turnaround time prediction for a given job depends on the predicted runtime of the currently queued or in execution jobs. Therefore, inaccuracies of runtime predictions for individual jobs can accumulate into inaccurate turnaround time predictions. Relying on inaccurate runtime predictions, such as those based on user estimates, can result in very poor turnaround time predictions that are detrimental to an IO-aware scheduler.

To predict turnaround time, we submit jobs to our simulated HPC system and record both the simulated turnaround time of each job and the job's execution schedule. When a job is submitted to the system, a snapshot of the system is created. The snapshot creation is followed by four steps. First, we copy the system state (i.e., allocated nodes, free nodes, simulated time, executing jobs, and queued jobs) in memory. Second, we replace the runtime of each job in execution and in the queue with the predicted job runtime. Third, we simulate the evolution of the system state from the snapshot until the submitted job has completed. Last, we record the difference between completion time and submission time of the job as our turnaround time prediction.

To quantify the turnaround time prediction accuracy, we sample five 10,000 job subsets from our original data; the subsets were randomly selected across the span of the job traces. We run five simulations, one for each job subset, and predict turnaround time as described above. Figure 11a shows the distribution of the simulations' turnaround times (i.e., the turnaround time observed in the simulation). Figure 11b compares the resulting relative accuracy of turnaround time predictions when the simulated system uses user-requested runtime (left) and our framework's runtime (right). We observe that our framework improves the mean accuracy by 14.0% and the median accuracy by 14.1% over user-requested runtime. Our mean and median turnaround time accuracy are 42.1% and 40.8%. Additionally, we note that the 75th and 95th percentile accuracies are over 20% greater with our predictions compared to user-requested runtimes. This indicates that using per-job runtime predictions from PRIONN achieves greater than 70% accuracy for turnaround time predictions for the upper-quartile jobs and better turnaround time predictions than users for all jobs.

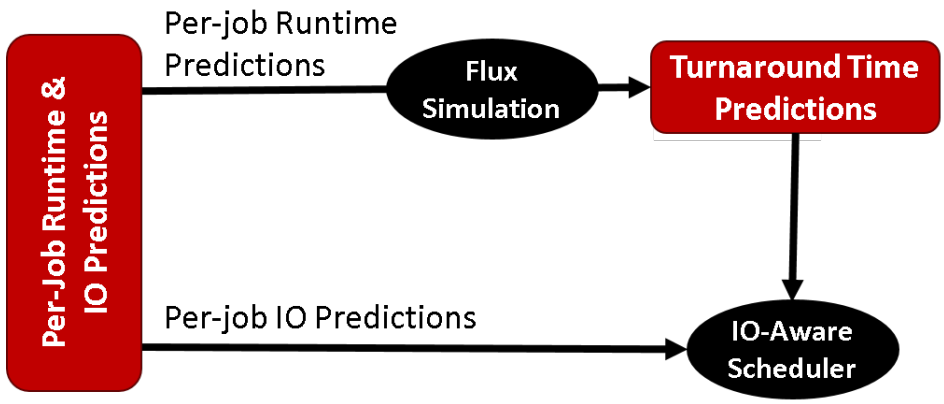
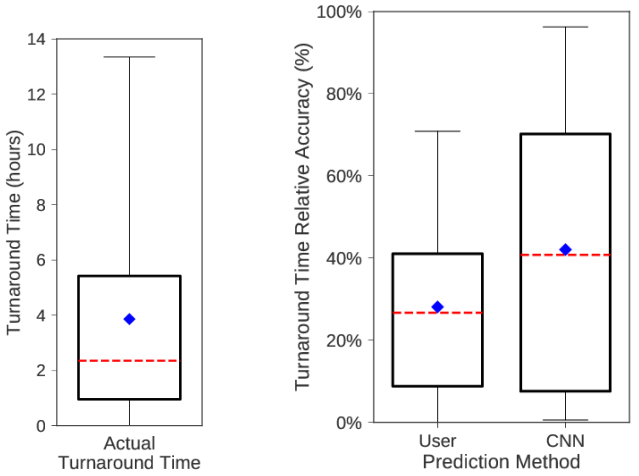


Figure 10: Overview showing the application of our runtime and IO predictions to an IO-aware scheduler to predict system IO and IO bursts.



(a) (b)

Figure 11: Distribution of our simulations' turnaround times (a) and relative accuracy of turnaround time predictions with user requested runtime and our method's runtime (b).

4.3 System IO Prediction

The next step to an IO-aware scheduler is to obtain system IO predictions. To this end, we combine turnaround time predictions from our simulated system with per-job IO usage predictions from PRIONN (i.e., predicted read and write bandwidth), as shown in Figure 10. Specifically, to predict the total system IO in use at a given time, we first use the job turnaround time predictions to determine which jobs are running on the system at that time. Then, for the running jobs, we sum their predicted IO usage, producing the estimated total system IO.

To quantify the accuracy of the estimated total system IO, we perform two types of evaluations, each one using different sources for runtime and turnaround time. In the first evaluation, we use perfect knowledge of runtime and turnaround time (i.e., from real job trace) and our predictions for per-job IO usage. This evaluation isolates the accuracy of our IO predictions from the accuracy of our runtime and turnaround time predictions to demonstrates the strength of PRIONN's per-job IO predictions alone. In the second evaluation, we use our runtime, turnaround time, and IO usage predictions (i.e., from Sections 3 and 4.2). This evaluation reflects how an IO-aware scheduler can rely on runtime and IO usage predictions from PRIONN in a real-world or production scenario.

For each one of the two evaluations, we report two metrics. First, we report the relative accuracy of our predicted IO behavior (i.e., system bandwidth over time). Second, we measure the precision and sensitivity (i.e., recall) for predicting IO bursts, or unusually high levels of IO bandwidth, that occur in the system IO behavior. IO bursts are of particular importance to an IO-aware scheduler because they are the most likely time for IO contention to occur. We define IO bursts based on the actual system IO bandwidth distribution shown in Figure 12a. We calculate the mean and standard deviation of this distribution. One standard deviation above the mean is marked with a green horizontal line at 1.35×10^9 bytes/s in Figure 12a. We define an IO burst as any bandwidth measurement above this value. For each real IO burst, we determine if an equivalent IO burst is also predicted within a given window of time. For example, with a three-minute window, we look for a predicted burst one minute before the actual IO burst, at the time of the real IO burst, and one minute after the actual IO burst. If a burst is predicted in this window, we record a True Positive (TP). We record False Positives (FP) (i.e., we predict an IO burst when there is not an IO burst) and False Negatives (FN) (i.e., there is an IO burst but we do not predict an IO burst) using this same window technique. We use these values (i.e., TP, FP, and FN) to calculate sensitivity and precision for our IO burst predictions. Sensitivity and precision have a range from 0% to 100%; larger values indicate better performance. Sensitivity is the ratio of correctly predicted IO bursts to actual IO bursts (i.e., $\frac{TP}{TP+FN}$). Precision is the ratio of correctly predicted IO bursts to total predicted IO bursts (i.e., $\frac{TP}{TP+FP}$).

For our first evaluation of system IO bandwidth prediction, we use perfect turnaround time knowledge for all jobs in our dataset

and per-job IO usage predictions from PRIONN. Figure 12b shows our first metric: the relative accuracy of each system IO prediction. We achieve the mean and median accuracy of 63.6% and 55.3% respectively. Figure 13 shows our second metric: the sensitivity and precision of our IO burst prediction across windows ranging from 5 minutes to 60 minutes. We observe in the figure that we predict 47.5% of real IO bursts within two minutes of their occurrence (i.e., sensitivity is 47.5% at a window size of 5 minutes). It also shows that 73.9% of predicted IO bursts correctly predict a real IO burst within two minutes of it occurring (i.e., precision is 73.9% at a window size of 5 minutes). Finally, we note that as the window size for predicting IO bursts increases, the sensitivity and precision also increase. From these results, we can see that IO predictions from PRIONN are sufficiently accurate to predict nearly 75% of IO bursts.

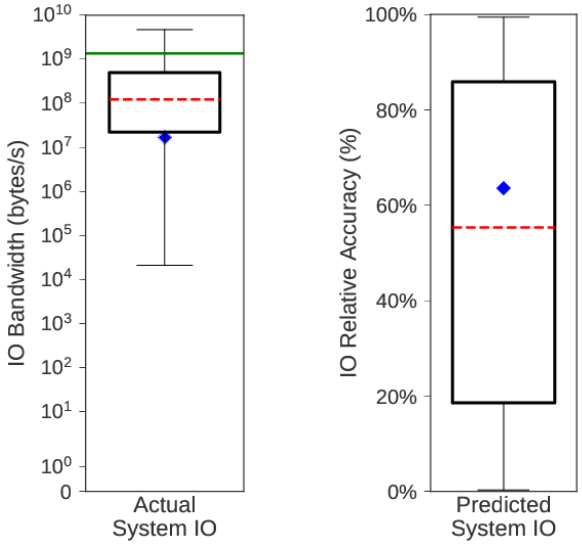


Figure 12: Actual aggregate IO (a) and relative accuracy of the system's accumulate IO predictions (b) using perfect turnaround time knowledge.

For our second evaluation of system IO bandwidth prediction, we use predicted turnaround time and IO usage for our five samples of 10,000 jobs. Figure 14a shows the distribution of simulated system IO. We compare the distributions of system IO for jobs used in our first evaluation (i.e., all jobs), shown in Figure 12a, and jobs used in our second evaluation (i.e., sampled jobs), shown in Figure 14a. We note that the distributions of system IO are not identical, but have similar maximum, mean, and median values. This indicates that the randomly chosen sample of jobs is a good representation of all jobs. Figure 14b shows our first metric: the relative accuracy of each system IO prediction using predicted turnaround time and predicted IO usage. Comparing Figure 12b with Figure 14b shows that the prediction accuracy for system IO decreases when our turnaround time predictions are used in place of perfect turnaround time knowledge. This is an expected result of using less accurate

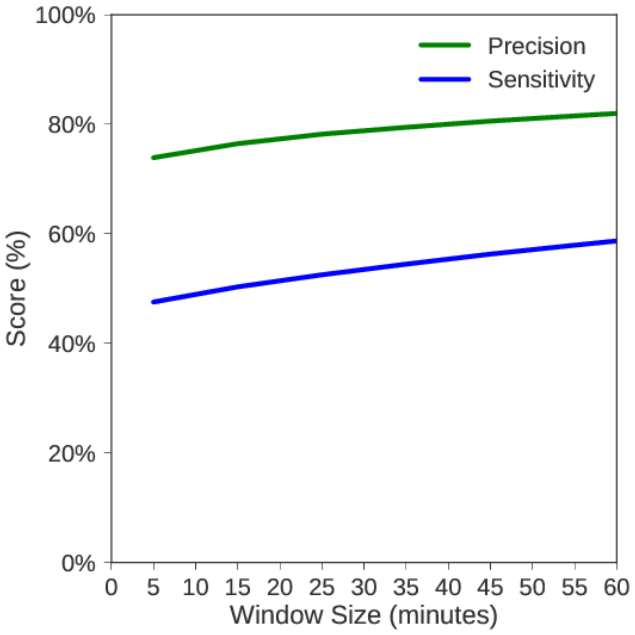


Figure 13: Sensitivity and precision of our IO burst predictions across windows ranging from 5 minutes to 60 minutes using perfect turnaround time knowledge.

turnaround time information. Despite the decreased average accuracy, we are still able to accurately predict several patterns in the IO behavior in the simulation, as indicated by the top whisker of the boxplot in Figure 14b.

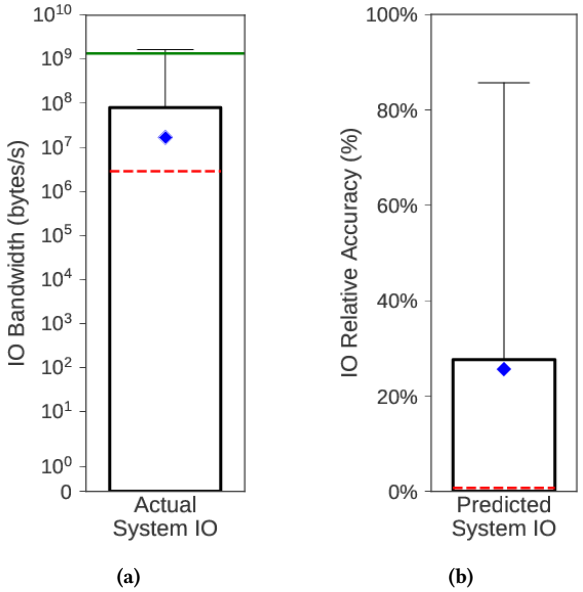


Figure 14: Actual aggregate IO (a) and relative accuracy of the system's accumulate IO predictions (b) using our predicted turnaround time from our simulated system.

Figure 15 shows our second metric: the sensitivity and precision of our IO burst prediction across windows ranging from 5 minutes to 60 minutes. We observe in Figure 15 that we predict 55.3% of real IO bursts within two minutes of them occurring (i.e., sensitivity is 55.3% at a window size of 5 minutes). It also shows that 70.0% of predicted IO bursts predict a real IO burst within two minutes of it occurring (i.e., precision is 70.0% at a window size of 5 minutes). We compare sensitivity and precision of IO burst prediction in our first evaluation, shown in Figure 13, and our second evaluation, shown in Figure 15. We note that despite switching from perfect to predicted turnaround time, we achieve similar sensitivity and precision. Like Figure 13, we also observe that sensitivity and precision increase as window size increases in Figure 15. These results indicate that per-job runtime and IO usage predictions from PRIONN provide the IO-aware scheduler with enough information to predict over 50% of IO bursts correctly. This is a huge improvement over being able to predict 0% of IO bursts without PRIONN.

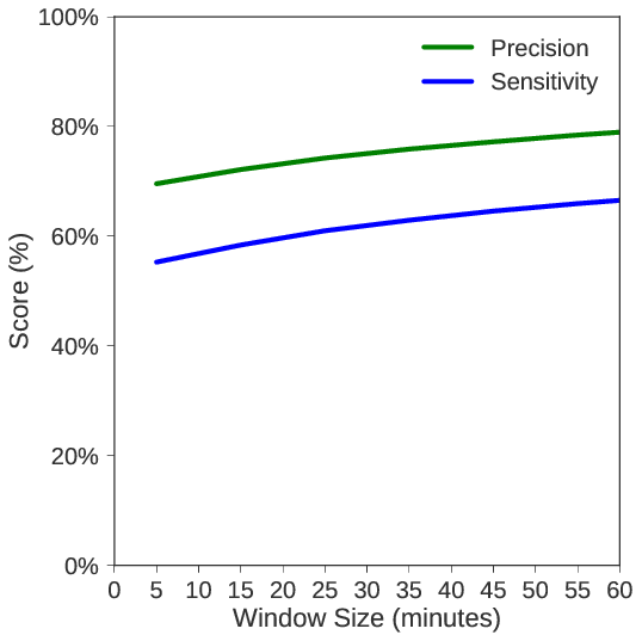


Figure 15: Sensitivity and precision of our IO burst prediction across windows ranging from 5 minutes to 60 minutes using our predicted turnaround time from our simulated system.

Our results in this section indicate that runtime and IO predictions from PRIONN can be used to generate accurate predictions of HPC system IO bandwidth and IO bursts. We show that with up to 57.9% mean turnaround prediction error, future IO bursts can be predicted with >50% accuracy. Additionally, with up to 36.4% mean system IO prediction error, future IO bursts can be predicted with >50% accuracy.

5 RELATED WORK

Many HPC job resource usage prediction methods have been proposed. Previous methods have involved a variety of machine learning models, familiar and novel. The most common attribute of each resource usage prediction method is the use of manually extracted features. Most of the previous efforts to predict job resource usage have focused on runtime predictions and derivatives of runtime, like turnaround time and wait time.

Smith, Foster, and Taylor used historical HPC job data to predict runtime in [19, 20]. For each job runtime prediction, they train a linear regression model based on historically similar jobs. Job similarity is calculated with several extracted features, including user and job queue. Other runtime prediction papers have utilized similar methods of manually extracted features and machine learning models as Smith, Foster, and Taylor. For example, Krishnaswamy, Loke, and Zaslavsky develop similarity templates for predicting job runtime based on extracted features in [7]. Cunha et al. use a kNN model to predict runtime and turnaround time for HPC jobs in [3]. This work utilizes extracted features from job scripts augmented with data from the scheduler, such as the number of jobs in the queue at job submission time. Chen, Lu, and Pattabiraman present a method for predicting runtimes of jobs being executed using features extracted from log files and a hidden Markov model [2]. Tsafrir, Etsion, and Feitelson predict job runtimes based on a user-centric model in [21]. The average users' previous job runtimes and use this as an estimation for the runtime of the next job submitted by a user. Downey developed a statistical model for predicting the queue time of a job based on jobs already running on an HPC system in [4]. Similarly, in the work of Nurmi, Brevik, and Wolksi a statistical method is developed, QBETS, to predict wait times for jobs [14]. Both of these works make predictions based on wait times of jobs currently running on a system and do not build a prediction model based on extracted features.

A smaller body of work has also focused on other job resources, such as IO. Lofstead et al. outline challenges of dealing with IO contention of parallel file systems and the benefits for preventing contention that come with knowing job IO behavior [10]. Other works present methods for using extracted features to predict IO among other resources to prevent resource contention. McKenna et al. test several machine learning methods for predicting runtime and IO usage of HPC jobs in [12]. They test kNN, DT, and RF models with manually extracted features from job scripts and job logs. Rodrigues et al. predict job runtime, wait time, and memory usage with an ensemble of machine learning algorithms, including kNN and RF, in [16]. Their method extracted features from log files and batch scheduler logs. Matsunaga and Fortes investigate the prediction of many job features, such as CPU, memory, and IO usage with several machine learning algorithms using extracted features in [11].

To the best of our knowledge, no previous work has been published on the prediction of runtime and IO use of jobs based on entire job scripts and used to predict IO activity, such as IO bursts, for IO-aware schedulers.

6 CONCLUSION AND FUTURE WORK

In this paper, we show the benefits of using PRIONN, our tool to Predict Runtime and IO using Neural Networks, for an IO-aware scheduler. Our tool relies on a novel method for mapping the text of whole job scripts into image-like representations. PRIONN utilizes the image-like representations to train a 2D-CNN model for accurate per-job runtime and IO resource prediction. We achieve accuracies for runtime and IO resource predictions that exceed previous work. Specifically, we predict runtime and IO resources with over 75% mean and 98% median accuracy across nearly 300,000 jobs from a large cluster at LLNL. We apply the predictions from PRIONN to an HPC system simulator to accurately predict system IO and IO bursts. We correctly predict over 50% of future IO bursts (i.e., times of IO contention). Our work demonstrates that PRIONN can augment schedulers with the per-job resource usage knowledge necessary to enable IO-aware scheduling. Future work includes incorporating application input decks into PRIONN's workflow and the prediction of other types of resources such as power and network.

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-739253.

REFERENCES

[1] Dong H. Ahn, Jim Garlick, Mark Grondona, Don Lipari, Becky Springmeyer, and Martin Schulz. 2014. Flux: a next-generation resource management framework for large HPC centers. In *10th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems*. 9–17.

[2] Xin Chen, Charng-Da Lu, and Karthik Pattabiraman. 2013. Predicting job completion times using system logs in supercomputing clusters. In *Proceedings of the 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*. 1–8.

[3] Renato LF. Cunha, Eduardo R. Rodrigues, Leonardo P. Tizzei, and Marco AS. Netto. 2017. Job placement advisor based on turnaround predictions for HPC hybrid clouds. *Future Generation Computer Systems* 67 (2017), 35–46.

[4] Allen B. Downey. 1997. Using queue time predictions for processor allocation. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. 35–57.

[5] Stephen Herbein, Dong H. Ahn, Don Lipari, Thomas R.W. Scogland, Marc Stearman, Mark Grondona, Jim Garlick, Becky Springmeyer, and Michela Taufer. 2016. Scalable I/O-aware job scheduling for burst buffer enabled hpc clusters. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC '16)*. ACM, New York, NY, USA, 69–80.

[6] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1746–1751.

[7] Shonali Krishnaswamy, Seng Wai Loke, and Arkady Zaslavsky. 2004. Estimating computation times of data-intensive applications. *IEEE Distributed Systems Online* 5, 4 (2004).

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th Neural Information Processing Systems Conference (NIPS)*. 1097–1105.

[9] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent Convolutional Neural Networks for Text Classification. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 333. 2267–2273.

[10] Jay Lofstead, Fang Zheng, Qing Liu, Scott Klasky, Ron Oldfield, Todd Kordenbrock, Karsten Schwan, and Matthew Wolf. 2010. Managing Variability in the IO Performance of Petascale Storage Systems. In *Proceedings of the 22nd ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 1–12.

[11] Andréa Matsunaga and José AB Fortes. 2010. On the use of machine learning to predict the time and resources consumed by applications. In *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*. 495–504.

[12] Ryan McKenna, Stephen Herbein, Adam Moody, Todd Gamblin, and Michela Taufer. 2016. Machine Learning Predictions of Runtime and IO Traffic on High-End Clusters. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. 255–258.

[13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th Neural Information Processing Systems Conference (NIPS)*. 3111–3119.

[14] Daniel Nurmi, John Brevik, and Rich Wolski. 2007. QBETS: queue bounds estimation from time series. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. 76–101.

[15] 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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[16] Eduardo R. Rodrigues, Renato LF. Cunha, Marco AS. Netto, and Michael Spriggs. 2016. Helping HPC users specify job memory requirements via machine learning. In *Proceedings of the Third International Workshop on HPC User Support Tools*. 6–13.

[17] Patrice Y Simard, David Steinkraus, John C Platt, et al. 2003. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *Proceedings of the 7th International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 3. 958–962.

[18] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[19] Warren Smith, Ian Foster, and Valerie Taylor. 1998. Predicting application run times using historical information. In *Workshop Job Scheduling Strategies for Parallel Processing (JSSPP)*. 122–142.

[20] Warren Smith, Valerie Taylor, and Ian Foster. 1999. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. 202–219.

[21] Dan Tsafrir, Yoav Etsion, and Dror G Feitelson. 2007. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems* 18, 6 (2007).