An efficient approach based on graph neural networks for predicting wait time in job schedulers

Tomoe Kishimoto and Tomoaki Nakamura

Computing Research Center, High Energy Accelerator Research Organization, KEK, Oho 1-1, Tsukuba, Japan
tomoe.kishimoto@kek.jp

Abstract. The objective of this study is to predict the wait time in job schedulers with high accuracy. Job executions in supercomputers or data centers are typically managed by job schedulers to efficiently utilize computing resources. A possible disadvantage is that, depending on resource availability and scheduling policy, the job waits for a long time before being executed. Therefore, providing the predicted wait time for individual jobs can contribute to the users' research planning. Additionally, the job wait time potentially becomes an important input for the scheduling policy. However, the prediction of the job wait time is a challenging task because the state of the scheduling system changes dynamically by many uncertainty factors. To address this problem, a graph neural network architecture of deep learning, which is a novel approach for processing job information in the scheduler, was employed in this study. Our experiments using real historical logs confirmed that the proposed deep learning model achieved 0.3–7.9% higher prediction accuracy compared to the boosted decision tree and multi-layer perceptron models. An extensive analysis of the proposed deep learning model was performed to improve the explainability of the experimental results. In particular, the visualization of attention weights in the graph neural network expanded our understanding of the behavior of the proposed deep learning model.

Keywords: Graph neural network · Job scheduler · Explainability.

1 Introduction

The scheduling of jobs is an essential task for utilizing computing resources in supercomputers and data centers, where the term "job" indicates a unit of program execution. This type of task is typically managed by a computer application, the so-called job scheduler or batch scheduler, such as Simple Linux Utility for Resource Management (SLURM) [27] or HTCondor [22]. The job scheduler controls the execution of jobs based on resource availability and scheduling policies, such as fair share. Jobs may possibly wait for a long time before being executed when the computer cluster is congested. If users are aware of the expected wait time before the job submission to the scheduler, they can optimize the job configuration, such as reducing the requested processor cores to increase the chance

of being executed. Additionally, the job wait time becomes an important factor when data centers are distributed like a grid computing system [7]. A scheduling policy to decide where to submit a job is required in such the distributed computing. However, the job wait time cannot be obtained straightforwardly because the state of the job scheduler changes dynamically, and there are many uncertainty factors. For example, users can provide an expected execution time (wall time) for the job when its submission as a configuration; this user-provided job wall time will become a piece of important information for estimating the job wait time because the wall time would affect a scheduling policy. However, the user-provided job wall time is known to be inaccurate because users tend to overestimate it to avoid job termination by the time limit [16]. To mitigate such uncertainty factors, deep learning (DL) technology is a promising approach that can learn the complex correlations required for prediction automatically by processing a large amount of historical data. We can assume that the DL model is capable of prediction if there is a common feature between past and current prediction data.

The objective of this study is to predict the job wait time with high accuracy using DL technology. DL has been successfully adapted to many scientific fields, particularly computer vision [20, 13]. The convolutional neural network [17] is one of the breakthroughs in computer vision, which introduces several key functions based on the domain knowledge of images, such as the local receptive field and the translation invariance, to efficiently process image data. Thus, we need to design a DL model to efficiently process our data, that is, the job information in a scheduler. To predict the job wait time, the relationship between other already running and waiting jobs in the computer cluster is important because we can expect the job wait time to be long if many high-priority jobs are already waiting in the scheduler. There are two main difficulties in including such information in the prediction. First, the DL model needs to handle variable length data because the number of waiting and running jobs differs depending on the situation, Second, there is a problem with the order of jobs when preparing input data from multiple waiting and running jobs.

In this study, we employed a gaph neural network (GNN) [6] architecture to address these problems, which is a novel approach for processing job information in job schedulers. We constructed a DL model based on the GNN architecture and performed experiments using open data [5], which contain historical workload data provided by many data centers, to examine its performance. The obtained results were compared with existing boosted decision tree (BDT) and simple multi-layer perceptron (MLP) models. An extensive analysis of the proposed DL model was conducted to improve the explainability of the experimental results. The details of the datasets and the proposed model structure are discussed in the following sections.

The remainder of this paper is organized as follows. Section 2 describes the related work, including our novelty. Section 3 presents the datasets used in this study. Section 4 provides the details of the proposed model. Section 5 presents

the experimental results. Finally, Section 6 summarizes the conclusion of the study.

2 Related work

The estimation of the job wait time is a long-standing concern, and many approaches have been studied and discussed. Approaches that exploit historical workload data, such as our study, are ordinary methods. For instance, the study in [18] reported that similar jobs and resources were discovered from historical data to predict the job wait time with the assumption that similar jobs under similar resources would most likely have a similar job wait time. To process historical workload data more efficiently, machine learning (ML) approaches, such as K-nearest neighbor and BDT algorithms, have also become common and have been confirmed to work for prediction [12, 9]. However, these ML approaches are limited to processing a fixed length of data, which results in loss of accurate information about job scheduler conditions. To the best of our knowledge, DL techniques for predicting the job wait time have not been adequately discussed, although there is a report for predicting the remaining run time of jobs using a recurrent neural network [26].

In contrast to the aforementioned studies, the first contribution of our study is that the GNN architecture of DL was employed to efficiently process our data. The GNN architecture enabled the processing of data with variable lengths; it was insensitive to the order of jobs, which enabled the handling of multiple waiting and running jobs as input data. Our experiments confirmed that our model outperformed existing BDT and simple MLP models. Second, experiments were performed to improve the explainability of the DL model. The permutation feature importance (PFI) technique was used to determine the importance of the input variables, and the attention mechanism was employed to visualize the importance of the relation between jobs.

3 Datasets

Our experiments were performed using datasets from the Parallel Workloads Archive [5], which ensured reproducibility and allowed us to examine several types of environments. In the Parallel Workloads Archive, the standard workload format [11] was defined, where information for each job, such as the wall-clock time, number of allocated processors, and exit status, was provided based on real workload logs. Table 1 summarizes the datasets used in the experiments. The datasets were selected to examine the different types of job schedulers.

The jobs in the datasets were ordered by their submission time and split into training, validation, and test data with a ratio of 80%: 10%: 10% from the beginning. The validation data were used to determine the hyperparameters of the DL models. The test data were used to evaluate performance without bias from the training and validation data. For instance, the training and test data in

Table 1: Description of datasets. The number of jobs for each experimental phase is shown. The dates (yyyy-mm-dd) in the brackets indicate the period of time for the corresponding phase. More details of each dataset are provided in [5].

Name	Job scheduler	Training data	Validation data	Test data
SDSC_BLUE	Catalina [1]	186,050 [2000–04–30 to 2002–05–30]	23,256 [2002–05–30 to 2002–08–29]	23,256 [2002–08–29 to 2002–12–30]
HPC2N	Maui [4]	162,297 [2002–08–01 to 2005–04–13]	20,287 [2005–04–13 to 2005–06–13]	20,287 [2005–06–13 to 2006–01–16]
ANL_Intrepid	Cobalt [2]	55,150 [2009–01–05 to 2009–07–08]	6,893 [2009–07–08 to 2009–08–05]	6,893 [2009–08–05 to 2009–09–01]
PIK_IPLEX	LoadLeveler	583.097 [2009–04–09 to 2012–02–06]	72,887 [2012–02–06 to 2012–04–25	72,887 [2012–04–25 to 2012–07–31
RICC	Custom-built	358,236 [2010–04–30 to 2010–09–13]	44,779 [2010–09–13 to 2010–09–18	44,779 [2010–09–18 to 2010–09–30
CEA_CURIE	SLURM	250,262 [2012–02–02 to 2012–09–15]	31,282 [2012–09–15 to 2012–10–02	31,282 [2012–10–02 to 2012–10–13

the SDSC_BLUE dataset correspond to the periods of “2000–04–30 to 2002–05–30" and “2002–08–29 to 2002–12–30," respectively. Therefore, a DL model needs to acquire the capability to predict the job wait time in completely different time ranges.

3.1 Prediction class definition

The aim of this study is to predict the job wait time, which is the difference between a job's submission time and the time at which it actually begins to run (start time). We defined time ranges for the job wait time, as described in Table 2, and the training was performed to predict this class category as a classification problem. The time ranges were determined so that users can recognize a general situation. For example, a job predicted as class 1 may start within 1 min after submission; moreover, a job predicted as class 5 may wait more than 10 h to start. We were interested in conducting a regression analysis to directly predict the job wait time; however, we focused on this classification approach in this study because it simplified the problem.

Figure 1 presents the distribution of the number of jobs in terms of the prediction class for each dataset. The distributions were divided into training, validation, and test phases. We confirmed that the number of jobs in the datasets was unbalanced for the prediction class, with class 1 being the dominant contribution. Thus, the weights for each prediction class (W_i) were calculated by

Table 2: Definition of prediction class.

Prediction class index	Definition
1	(Wait time) $<$ 1 min
2	1 min $\leq$ (wait time) $<$ 10 min
3	10 min $\leq$ (wait time) $<$ 1 h
4	1 h $\leq$ (wait time) $<$ 10 h
5	10 h $\leq$ (wait time)

using following formula:

$$W_i = \frac{\min\limits_{j \in [1,\cdots,5]} N_j}{N_i}, \tag{1}$$

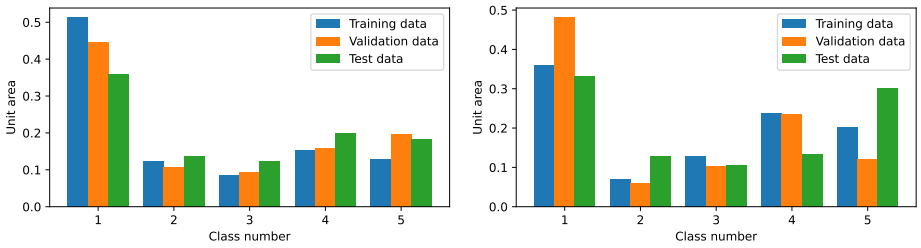
where i and j are the prediction class indeces, and N indicates the number of jobs in the class. The weights were obtained from the training data and are used to calculate the loss values to mitigate the imbalance of classes during DL training.

3.2 Input variables

For predicting the job wait time, the cluster status is important information. For example, we can assume that a job will wait for a long time if the cluster is congested with other jobs. Thus, utilizing information from other jobs, such as already running and waiting jobs, is a key concept of this study, as discussed in the previous sections. A snapshot of when the job was submitted was reconstructed from historical logs in the Parallel Workloads Archive. This snapshot was the unit of input data for the DL models. Snapshots were reconstructed for all jobs in the archive, except for jobs canceled before starting. The canceled jobs were removed because their contribution to the status was unclear. Figure 2 shows an overview of the snapshot. The following four types of jobs were defined in the snapshot:

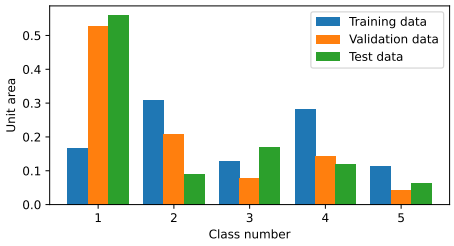
- **Target job**: The job of interest that we aimed to predict the wait time.
- **Running jobs**: Jobs running in the cluster when the target job was submitted. All running jobs in the cluster were used as inputs for the prediction.
- **Waiting jobs**: Jobs waiting in the cluster when the target job was submitted. All waiting jobs in the cluster were used as inputs for the prediction.
- **Finished jobs**: Jobs finished within the last 5 days from the time the target job was submitted. Only 20 finished jobs were used as inputs in order for the latest finished time to reduce the size of the inputs.

Table 3 summarizes the definitions of the input variables for each job. Many of the input variables used the original values from the standard workload format. Input variables that were not available at the time the target job was submitted in a real situation, such as RUN_TIME of the waiting jobs, were masked

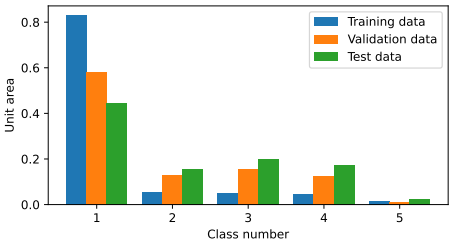


(a) SDSC_BLUE

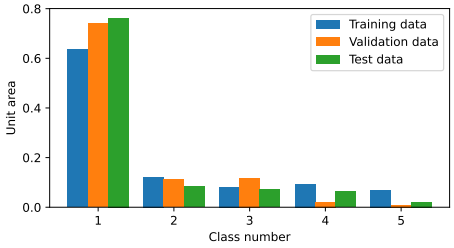
(b) HPC2N



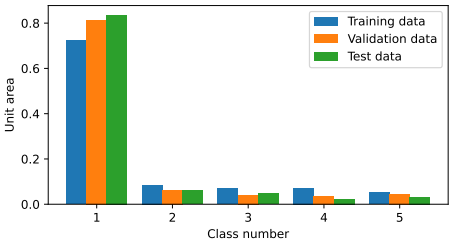
(c) ANL_Intrepid



(d) PIK_IPLEX



(e) RICC



(f) CEA_CURIE

Fig. 1: Distribution of prediction classes for each dataset and phase. The area of each histogram is normalized to 1.

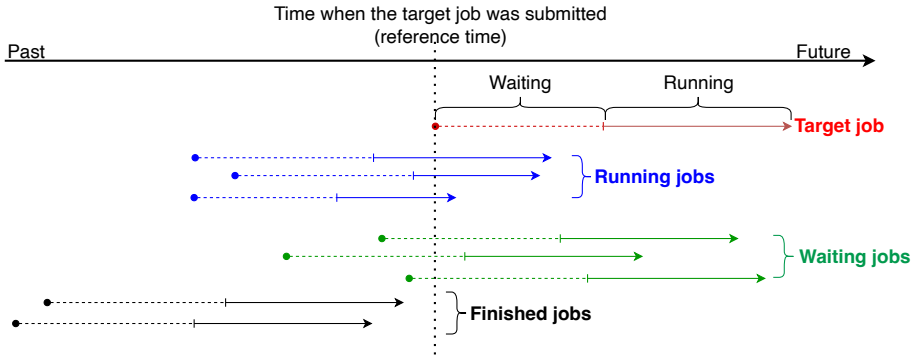


Fig. 2: Overview of the snapshot. Each arrow represents a job in the Parallel Workloads Archive. The dashed and solid lines indicate the wait time and run time, respectively.

with 0 or modified with limited information to avoid leaking future information. Log transformation was applied to all input variables to fit the values within a reasonable range. As an example of the input size, if there were 10 jobs in the snapshot, the total number of input variables was 10 (the number of jobs) $\times$ 21 (the number of input variables) $=$ 210.

4 Proposed DL model

Figure 3 shows an overview of the proposed DL model. In our model, a graph structure composed of nodes and edges was built for each snapshot. Each node in the graph corresponds to a job in the study. Thus, job features (summarized in table 3) were assigned to the node attributes in the initial inputs. Edges were prepared between the target job and other jobs as bidirectional graphs. The self-loop of the target job was also included. Edge attributes were empty in the initial inputs.

The model consisted of two parts, that is, a feature module and a classifier module. The feature module was aimed at extracting global features of the snapshot by exchanging information between the target job and other jobs along the edges. The feature module consisted of a stack of blocks, where each block consisted of a graph attention network (GAT) layer [8], batch normalization (BN) layer [14], and the ReLU activation function. The GAT was a key component in our study that introduced the attention mechanism [24] with the GNN architecture. The GAT learned the importance of node relations as edge attributes (attention weights). For instance, if the relation between the target job and the latest finished job was more important than other relations, the attention weight for information exchange obtained a large value. Thus, the attention weights in the GAT layer were considered to improve the learning efficiency and were also useful for visualizing the relation between the jobs (this will be discussed in Sec-

Table 3: Definition of input variables for each job. The asterisks indicate that the original values from the standard workload format were used. "reference time" is the time that the target job was submitted.

ID	Name	Description
1.	JOB_NUMBER*	A job identifier indicated by an integer.
2.	SUBMIT_TIME	The difference between the job's submission time and the reference time (in seconds).
3.	WAIT_TIME	The running and finished jobs: the difference between the job's submission time and the start time (in seconds). The waiting jobs: the difference between the job's submission time and the reference time (in seconds). The target job: 0 is filled because this is the value of interest.
4.	RUN_TIME	The finished jobs: the wall clock time of the job (in seconds). The running jobs: the difference between the job's start time and the reference time (in seconds). The waiting jobs and the target job: 0 is filled.
5.	ALLOCATE_CORE*	The number of allocated processors.
6.	REQUEST_CORE*	The number of requested processors.
7.	REQUEST_TIME*	The requested time (in seconds).
8.	REQUEST_MEMORY*	The requested memory size (in KB).
9.	STATUS	The target job: 0 is filled. The running jobs: 1 is filled. The waiting jobs: 2 is filled. The finished jobs: the original value from the standard workload format + 3 is filled.
10.	USER_ID*	A user identifier indicated by an integer.
11.	GROUP_ID*	A group identifier indicated by an integer.
12.	APPLICATION_NUMBER*	An application identifier indicated by an integer. This might represent a script file used to run jobs.
13.	QUEUE_NUMBER*	A queue identifier indicated by an integer.
14.	PARTITION_NUMBER*	A partition identifier indicated by an integer.
15.	SUBMIT_WEEKDAY	A weekday identifier $[0,\cdots,6]$ when the job was submitted.
16.	SUBMIT_HOUR	Hour $[0,\cdots,23]$ when the job was submitted.
17.	WAIT_JOB	The number of waiting jobs in the queue at the reference time.
18.	RUN_JOB	The number of running jobs in the queue at the reference time.
19.	WAIT_CORE	The total number of requested cores of the waiting jobs in the queue at the reference time.
20.	RUN_CORE	The total number of requested cores of the running jobs in the queue at the reference time.
21.	USER_TIME	A total CPU time consumed by the user during the last 5 days from the reference time.
22.	USER_WAIT_TIME	The average wait time of two preceding jobs by the user.

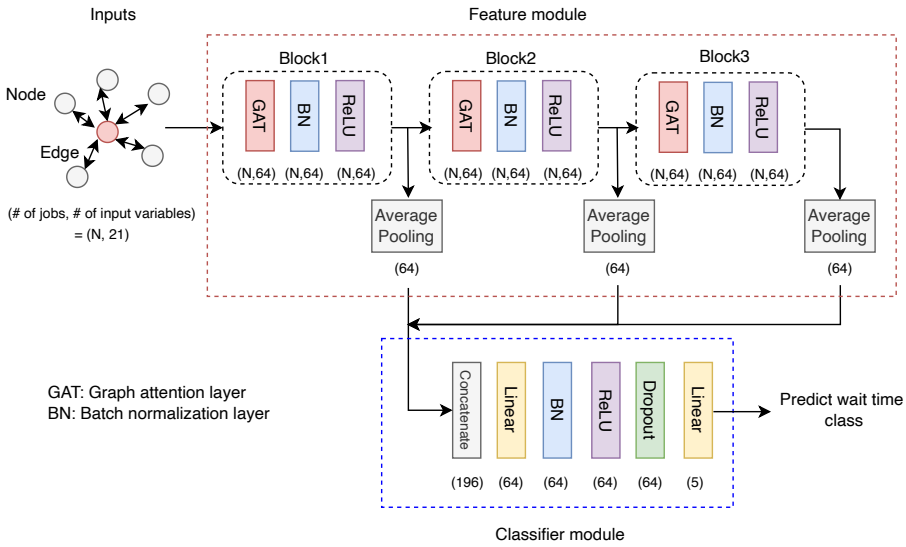


Fig. 3: Overview of proposed DL model. The numbers in the brackets indicate output shapes.

tion 5.2). Another important aspect of the GAT is that it worked with a variable number of jobs and was permutation-invariant because trainable parameters existed only in node-wise and edge-wise computations. This feature overcame the problems of the existing ML approaches discussed in Section 2.

The outputs of each block in the feature module were processed by average pooling, which averaged the node attributes, and was then fed into the classifier module. The classifier module was aimed to predict the wait time class, as described in Section 3.1. The global feature was obtained by concatenating the outputs of the feature module; then, they were processed by a fully connected (linear) layer, BN, ReLU, and linear layers, as shown in Figure 3. The dropout function [21] was also introduced before the last layer to mitigate overlearning.

5 Results and discussion

The proposed model for this experiment was implemented using the DL libraries: PyTorch [19] and DGL [25], and is available in [3]. All executions used a local cluster of NVIDIA Tesla A100 graphics cards.

The cross-entropy loss between the true and predicted class with sample weights (as discussed in Section 3.1) was used as the loss function for training. The best epoch for the validation data was used as the final model parameters after training for up to 30 epochs. We observed that an epoch less than 10 tended to show the best performance owing to overlearning. This indicates that obtaining the generalization between different periods of data is difficult. The

Adam [15] algorithm was used as an optimizer with a learning rate of 0.001. The batch size was fixed at 128. Other hyperparameters, such as the number of nodes in the GAT layer and the number of blocks in the feature module, were optimized by a grid search using the SDSC_BLUE dataset.

Figure 4 shows the accuracy of the test data represented in a confusion matrix format. As a global trend, the prediction classes 1 and 5, which are for job wait times of less than 1 min and greater than 10 h, show better accuracies than the other classes. This observation is consistent with a previous study using the BDT [12] and our initial assumption that the intermediate situations are more difficult to predict than clusters that are extremely empty or congested. Another possible reason for the better accuracy in class 1 is that the prediction of class 1 is more robust than that of other classes because the class 1 has more data, as shown in Figure 1.

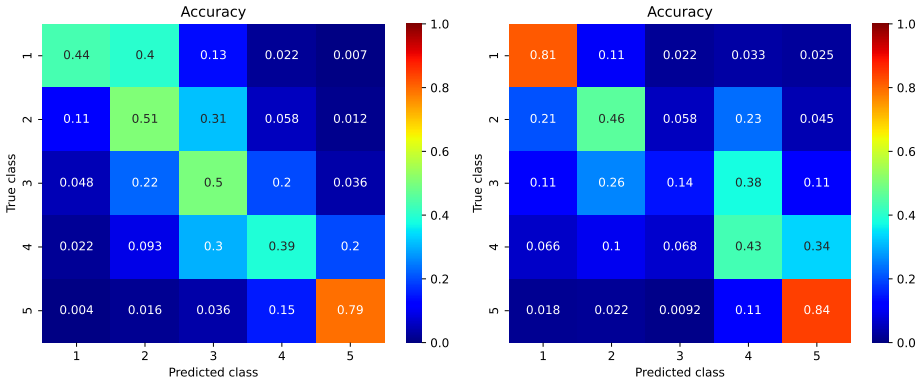
In the following subsections, the proposed model is compared with existing BDT and simple MLP models. The limitations of our experiments are also discussed. Additionally, in-depth analyses were conducted to understand the behavior of the proposed model. In the following subsections, the accuracy results were obtained by averaging the five prediction classes.

5.1 Comparison with other methods

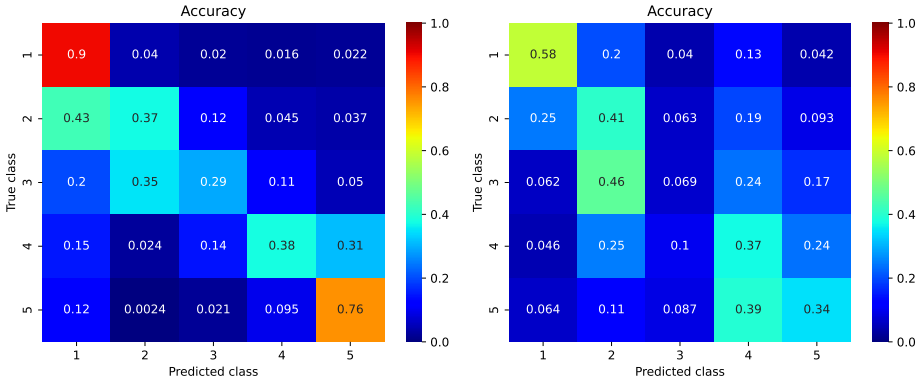
Firstly, our model was compared with BDT and simple MLP models. BDT was executed using XGBoost [10]. The maximum depth in the BDT was 7 in the comparison, which was determined by a parameter scan to maximize the accuracy. The MLP model consisted of a stack of linear, BN, ReLU, and dropout layers. The numbers of layer blocks and nodes in the MLP were optimized by a grid search and were 8 and 256, respectively.

To execute the BDT and MLP models, we had to prepare a fixed length of input data; however, the number of running and waiting jobs was dynamically changed in the snapshots. Therefore, the running and waiting jobs were ordered by the submission time; then, N jobs were selected from the recently submitted jobs. The recently submitted jobs were selected because the attention weights indicated that they were more important than the old jobs (which will be discussed in Section 5.3). For instance, if N was 10, the length of the input data was (1 (the number of target jobs) + 10 (the number of running jobs) + 10 (the number of waiting jobs) + 20 (the number of finished jobs)) $\times$ 21 (the number of input variables) = 861. Although there were input data that did not include all running and waiting jobs because of the limitation of the fixed length in the BDT and MLP, the global information of clusters was still available via the input variables of WAIT_JOB, RUN_JOB, WAIT_CORE, RUN_CORE, and USER_TIME. If the number of running or waiting jobs was less than N, 0 was filled to obtain a fixed length.

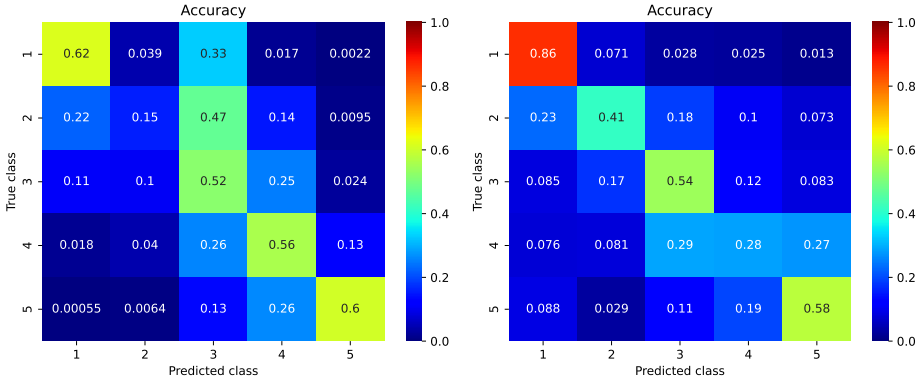
Table 4 summarizes the accuracies obtained for each experimental condition and dataset, where the cases of N=10 and 50 were executed for the BDT and MLP models. The results of N=50 tend to show worse accuracy than that of N=10 due to overlearning. Table 4 also includes the results of a simple approach



(a) SDSC_BLUE (b) HPC2N



(c) ANL_Intrepid (d) PIK_IPLEX



(e) RICC (f) CEA_CURIE

Fig. 4: Confusion matrix of test data for each dataset. The accuracies were obtained by averaging 20 runs with different initial values.

without the ML techniques (average method). The average method, which was proposed in [23], just averages the wait times of two previous jobs by the same user, and uses this averaged value as the prediction. Table 4 confirms that our model shows higher average prediction accuracies for all datasets, which are ranging from 0.3 to 7.9% improvements on top of other approaches. Only the PIK_IPLEX dataset shows the consistent accuracy with the MLP model (N=10) within the error. Therefore, we consider that the proposed model can extract features more effectively based on the GNN architecture than the BDT and simple MLP models.

Table 4: Accuracy results for each dataset and approach. The accuracies of DL and ML approaches are the average values of 20 runs with different initial values, and the errors are one standard error.

Dataset	Proposed model	MLP		BDT		Average method
		N=10	N=50	N=10	N=50	
SDSC_BLUE	**0.524 ± 0.004**	0.468 ± 0.003	0.459 ± 0.005	0.461 ± 0.002	0.446 ± 0.002	0.409
HPC2N	**0.537 ± 0.006**	0.495 ± 0.005	0.489 ± 0.004	0.465 ± 0.004	0.448 ± 0.003	0.380
ANL_Intrepid	**0.540 ± 0.010**	0.475 ± 0.005	0.439 ± 0.007	0.429 ± 0.005	0.455 ± 0.003	0.527
PIK_IPLEX	**0.355 ± 0.006**	0.352 ± 0.004	0.331 ± 0.005	0.270 ± 0.005	0.273 ± 0.003	0.337
RICC	**0.490 ± 0.010**	0.411 ± 0.007	0.406 ± 0.009	0.391 ± 0.006	0.397 ± 0.008	0.361
CEA_CURIE	**0.535 ± 0.006**	0.506 ± 0.004	0.461 ± 0.005	0.471 ± 0.004	0.468 ± 0.003	0.381

Our study has the following limitations. The training of our model was significantly slower than that of the MLP model, which was approximately 60 batch/s (our model) and 320 batch/s (MLP model) using SDSC_BLUE dataset. Moreover, the BDT and average method require much lower computational costs without the graphics cards. Owing to this computational constraint, we were unable to increase the number of dataset types and evaluate a larger dataset. Additionally, we were interested in measuring the prediction latency, which is an important factor for deploying a prediction mechanism, such as a command line tool, to real clusters. Therefore, improving the training speed and measuring the latency are topics for future research. The following subsections focus on the explainability of the proposed model.

5.2 Time dependency

The prediction accuracies of our model were calculated for different periods of the validation and test data. We expected that better performance would be

observed when the evaluation period was close to the training period because the condition of clusters may be similar. Figure 5 shows the time dependency of the observed accuracies for each dataset. As expected, a decreasing trend in accuracy was observed if we focused on the SDSC_BLUE dataset. However, the accuracies were still better than the random prediction even 10 weeks after the training period. This result indicates that our model successfully obtained generalization ability for completely different data periods.

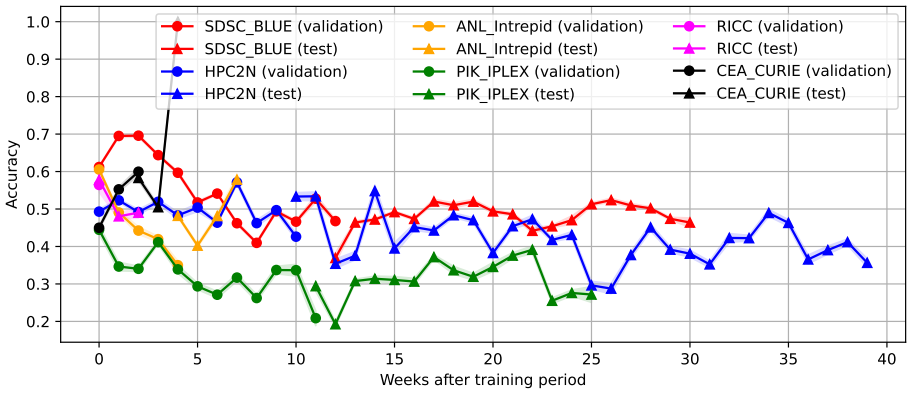


Fig. 5: Time dependencies of observed accuracies for each dataset. The proposed model was used. The x-axis represents weeks elapsed since the period of training data. Each point represents the accuracy of the validation or test data within a week. The accuracy is the average value of 20 runs with different initial values, and the errors are one standard error.

5.3 Importance of input variables

The PFI technique was used to understand the importance of input variables. In the PFI technique, each input variable was randomly shuffled within the dataset during the prediction phase; then, the difference in accuracy from the nominal value (PFI_i) was obtained as follows:

$$\mathrm{PFI}_i = \mathrm{ACC}_{\mathrm{nominal}} - \mathrm{ACC}_i, \tag{2}$$

where $\mathrm{ACC}_{\mathrm{nominal}}$ and ACC_i are the accuracies when all input variables are available and variable i is shuffled, respectively. Thus, we can consider that the contribution of variable i to the prediction is large if the PFI_i has a larger value. Figure 6 summarizes the PFI values for each dataset. As global observations, SUBMIT_TIME, WAIT_TIME, and RUN_TIME show larger contributions. We can confirm that our model utilizes the information from the running, waiting, and finished jobs in the snapshot because these input variables were masked

in the target job. In Table 4, the PIK_IPLEX dataset shows the lowest accuracy of our proposed model, and a limited improvement compared to the other approaches. One of the reasons is that REQEST_CORE and REQUEST_TIME were not available in the Parallel Workloads Archive of this dataset, which show large contributions in the other datasets.

5.4 Visualization of attention weights

Figure 7 shows the observed attention weights for the five selected snapshots in the SDSC_BLUE dataset. Each point in the figure indicates the target job (red), running job (blue), waiting job (green), and finished job (black). The arrows with a color gradient represent the attention weight values between the target job and other jobs, where only the direction to the target job is shown. The distance (D) between the target job and the other jobs in the figures was calculated from the difference between the reference time and the submission time for each job, that is, SUBMIT_TIME in Table 3. Log transformation was applied to D to fit the distance within a reasonable range. Thus, a job close to the target job in the figure indicates that it was recently submitted. In addition, the jobs submitted by the same user were clustered so that they were positioned near each other.

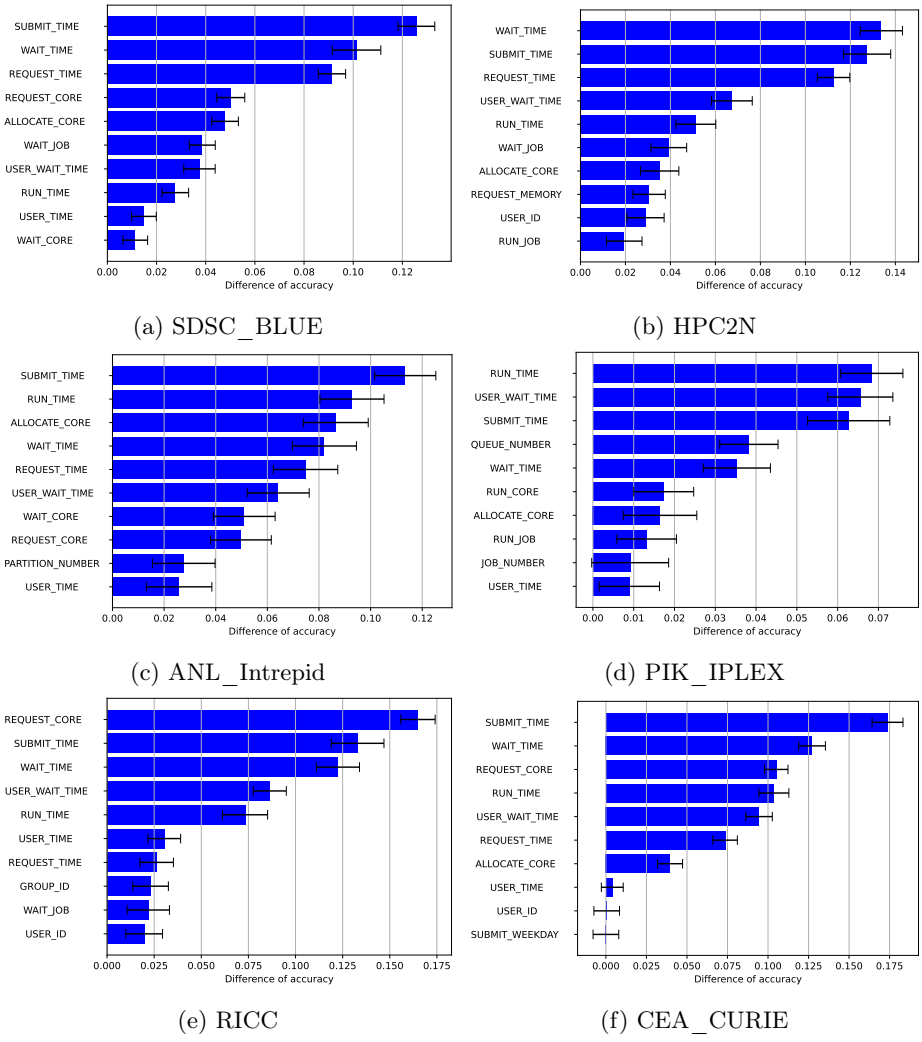
The attention weights between the target job and closer jobs show larger values as a trend in the figure, which indicates that recently submitted jobs are more important than old jobs for prediction. To understand this more clearly, the distance ratio was defined as follows:

$$\text{Distance ratio} = \frac{D^{\text{attention}} - D^{\min}}{D^{\max} - D^{\min}}, \tag{3}$$

where $D^{\text{attention}}$ is the distance of the job that shows the maximum attention weight in the snapshot; $D^{\max}$ and $D^{\min}$ are the maximum and minimum distances for all jobs in the snapshot, respectively. For example, if the distance ratio is zero, the job closest to the target job shows the largest attention weight in the snapshot. Figure 8 shows the distribution of the distance ratio for the test data of the SDSC_BLUE dataset. The distribution was divided by job types, that is, running, waiting, and finished jobs. It can be confirmed that there are large entries in the first bins of the distance ratio = 0.

6 Conclusions

In this study, we proposed an efficient approach for predicting the job wait time using a GNN architecture. We built and trained a DL model based on the GNN to efficiently process the dynamic information of job data. Our experiments confirmed that 0.3–7.9% higher prediction accuracy of the job wait time was achieved compared to the BDT and MLP models. An in-depth analysis was conducted to improve the explainability of the proposed model. In particular, the visualization of the attention weights expanded our understanding of the



(a) SDSC_BLUE (b) HPC2N

(c) ANL_Intrepid (d) PIK_IPLEX

(e) RICC (f) CEA_CURIE

Fig. 6: Top 10 PFI values of test data for each dataset. The proposed model was used. The accuracy is the average value of 20 runs with different initial values, and the errors are one standard error.

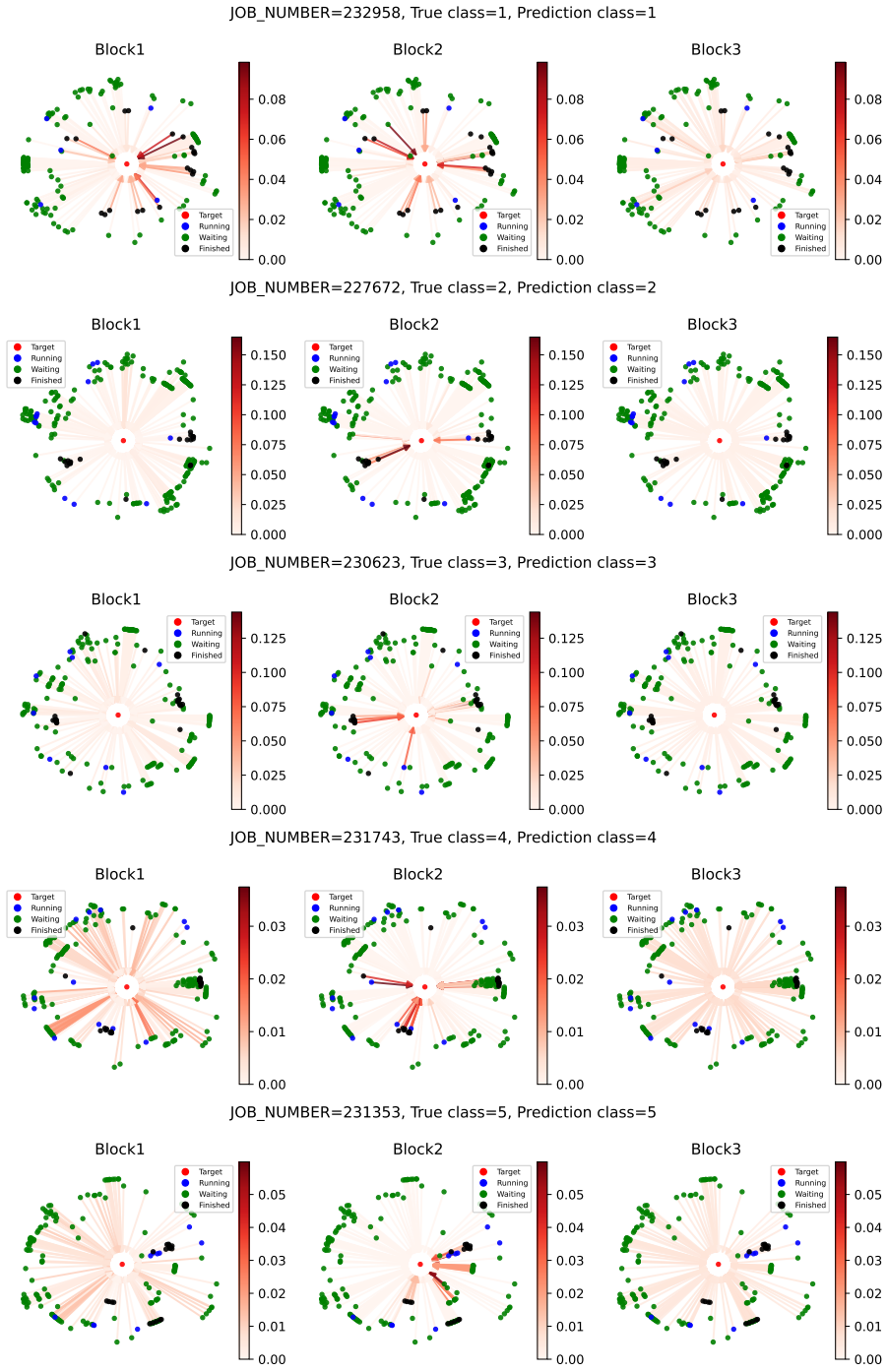


Fig. 7: Attention weights for 5 selected snapshots in the SDSC_BLUE dataset. The 5 snapshots, one from each prediction class, are shown from top to bottom. The attention weights for 3 blocks in the feature module are shown from left to right for each snapshot. The snapshots were selected from the test data.

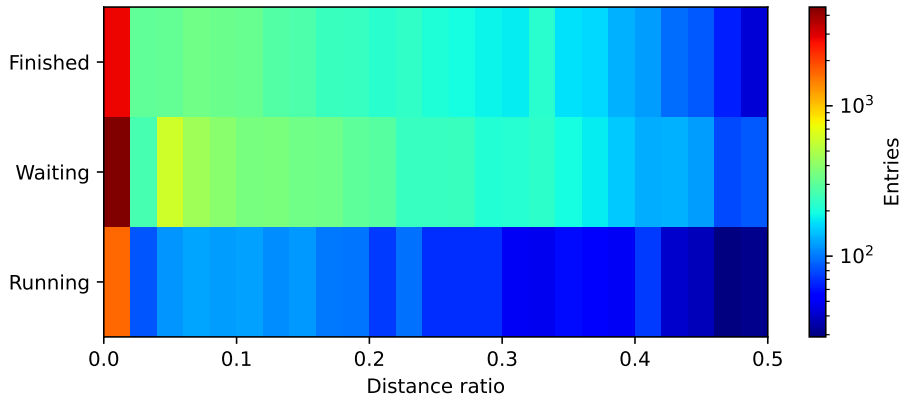


Fig. 8: Two-dimensional distribution of maximum attention weight in the snapshot. The x-axis and y-axis represent the distance ratio and job type, respectively. The definition of the distance ratio is described in the text. The test data of SDSC_BLUE dataset were used.

behavior of the DL model, which indicated that the information of recently submitted jobs is important for prediction.

We also identified the limitations of the proposed approach during our experiments. Overlearning is the main concern for improving accuracy. The transfer learning technique seems to be a feasible approach to obtain robustness for different state of clusters. We can train the model using a dataset; subsequently, it can be used as the initial weight parameters for the training of other datasets. The data augmentation technique for our data is also required to suppress overlearning. The prediction latency was not measured in our experiments, and is a topic for future research.

References

1. Catalina scheduler. https://www.sdsc.edu/catalina/, (Accessed: Nov. 30, 2022)
2. Cobalt scheduler. https://trac.mcs.anl.gov/projects/cobalt/, (Accessed: Nov. 30, 2022)
3. deepbatch. https://github.com/ktomoe/deepbatch/, (Accessed: Dec. 16, 2022)
4. Maui scheduler. http://docs.adaptivecomputing.com/maui/, (Accessed: Nov. 30, 2022)
5. Parallel workloads archive. https://www.cs.huji.ac.il/labs/parallel/workload/index.html, (Accessed: Nov. 30, 2022)
6. Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., Pascanu, R.: Relational inductive biases, deep learn-

ing, and graph networks (2018). https://doi.org/10.48550/ARXIV.1806.01261, https://arxiv.org/abs/1806.01261
7. Bos, K., Brook, N., Duellmann, D., Eck, C., Fisk, I., Foster, D., Gibbard, B., Grandi, C., Grey, F., Harvey, J., Heiss, A., Hemmer, F., Jarp, S., Jones, R., Kelsey, D., Knobloch, J., Lamanna, M., Marten, H., Mato Vila, P., Ould-Saada, F., Panzer-Steindel, B., Perini, L., Robertson, L., Schutz, Y., Schwickerath, U., Shiers, J., Wenaus, T.: LHC computing Grid: Technical Design Report. Version 1.06 (20 Jun 2005). Technical design report. LCG, CERN, Geneva (2005), http://cds.cern.ch/record/840543
8. Brody, S., Alon, U., Yahav, E.: How Attentive are Graph Attention Networks? . In: International Conference on Learning Representations (2022), https://openreview.net/forum?id=F72ximsx7C1
9. Brown, N., Gibb, G., Belikov, E., Nash, R.: Predicting batch queue job wait times for informed scheduling of urgent hpc workloads (2022). https://doi.org/10.48550/ARXIV.2204.13543, https://arxiv.org/abs/2204.13543
10. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785–794. KDD '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939785, http://doi.acm.org/10.1145/2939672.2939785
11. Feitelson, D.G., Tsafrir, D., Krakov, D.: Experience with using the parallel workloads archive. Journal of Parallel and Distributed Computing **74**(10), 2967–2982 (2014). https://doi.org/https://doi.org/10.1016/j.jpdc.2014.06.013, https://www.sciencedirect.com/science/article/pii/S0743731514001154
12. Gombert, L., Suter, F.: Learning-based Approaches to Estimate Job Wait Time in HTC Datacenters. In: Klusáček, D., Cirne, W., Rodrigo, G. (eds.) 24th Workshop on Job Scheduling Strategies for Parallel Processing. Job Scheduling Strategies for Parallel Processing. JSSPP 2021, vol. 12985, pp. 101–125. Portland, United States (May 2021). https://doi.org/10.1007/978-3-030-88224-2_6, https://hal.archives-ouvertes.fr/hal-03357129
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015). https://doi.org/10.48550/ARXIV.1512.03385, https://arxiv.org/abs/1512.03385
14. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37. p. 448–456. ICML'15, JMLR.org (2015)
15. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2017)
16. Klusáček, D., Chlumský, V.: Evaluating the impact of soft walltimes on job scheduling performance. In: Klusáček, D., Cirne, W., Desai, N. (eds.) Job Scheduling Strategies for Parallel Processing. pp. 15–38. Springer International Publishing, Cham (2019)
17. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (may 2017). https://doi.org/10.1145/3065386, https://doi.org/10.1145/3065386
18. Li, H., Groep, D., Wolters, L.: Efficient response time predictions by exploiting application and resource state similarities. In: The 6th IEEE/ACM International Workshop on Grid Computing, 2005. pp. 8 pp.– (2005). https://doi.org/10.1109/GRID.2005.1542747

19. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
20. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). https://doi.org/10.48550/ARXIV.1409.1556, https://arxiv.org/abs/1409.1556
21. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**(56), 1929–1958 (2014), http://jmlr.org/papers/v15/srivastava14a.html
22. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the condor experience. Concurrency - Practice and Experience **17**(2-4), 323–356 (2005)
23. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Transactions on Parallel and Distributed Systems **18**(6), 789–803 (2007). https://doi.org/10.1109/TPDS.2007.70606
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017), https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
25. Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., Zhang, Z.: Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. arXiv preprint arXiv:1909.01315 (2019)
26. Wang, Q., Zhang, H., Qu, C., Shen, Y., Liu, X., Li, J.: Rlschert: An hpc job scheduler using deep reinforcement learning and remaining time prediction. Applied Sciences **11**(20) (2021). https://doi.org/10.3390/app11209448, https://www.mdpi.com/2076-3417/11/20/9448
27. Yoo, A.B., Jette, M.A., Grondona, M.: Slurm: Simple linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) Job Scheduling Strategies for Parallel Processing. pp. 44–60. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)