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

大学共同利用機関法人
高エネルギー加速器研究機構

Tomoe Kishimoto

Computing Research Center, KEK

Introduction

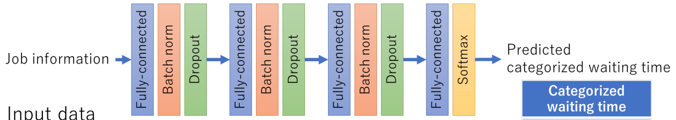
- Estimation of the “job wait time” in job schedulers is a long-standing concern and a challenging task
  - Machine Learning (ML) and Deep Learning (DL) are promising approaches for this task, which learn complex correlations automatically
  - Several activities were already reported in the FJPPL project

→ We introduce a modern DL technique to efficiently address this task

Keywords: Graph neural network and explainability

W.Takase (KEK-CRC)

- Set up Fully-Connected Neural Network model

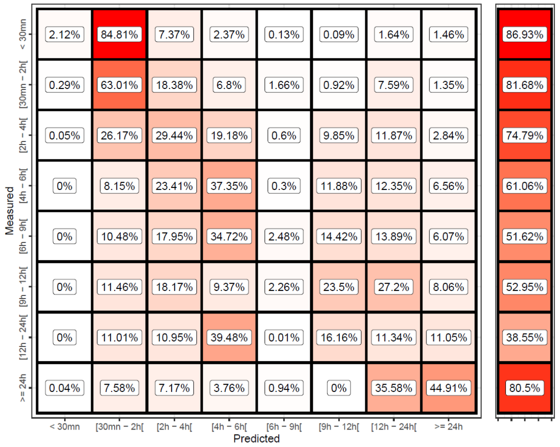


- Input data
  - Finished job logs last 7 months
  - Sampled 400,000 job histories

Input job information	
The last observed user’s waiting time	Run time currently spent
Submission time of the above	Day of submission
Number of cores currently in use	Submission time in 24 hours
Current CPU utilization efficiency	Queue
Number of current waiting jobs	

Categorized waiting time
0 to 10 mins
10 to 30 mins
30 min to 1 hour
1 to 3 hours
3 to 6 hours
More than 6 horus

F.Suter and L.Gombert (CC-IN2P3)

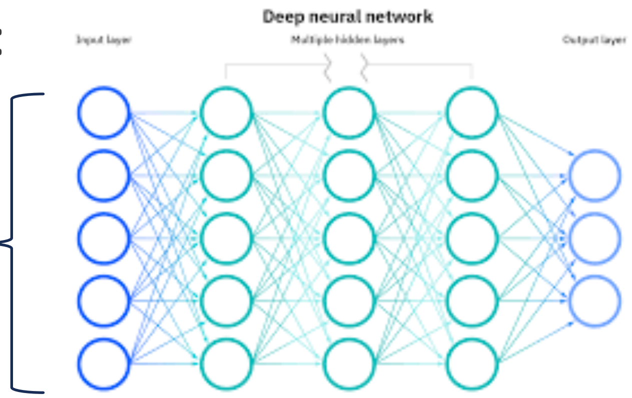


Problem in traditional approaches

- To predict the job wait time with high accuracy, the relation between other already running and waiting jobs is important
  - E.g.) we can expect that the job wait time will be long if many high-priority jobs are already waiting in the scheduler
  - The condition of the scheduler changes dynamically
- However, traditional ML and DL require a “fixed” length of data:
  - E.g.) the length of input data should be 5
  - We lose accurate information in a job scheduler

→ We need to design a DL model to handle the variable length of data
→ Graph Neural Networks (GNN) is employed in this study

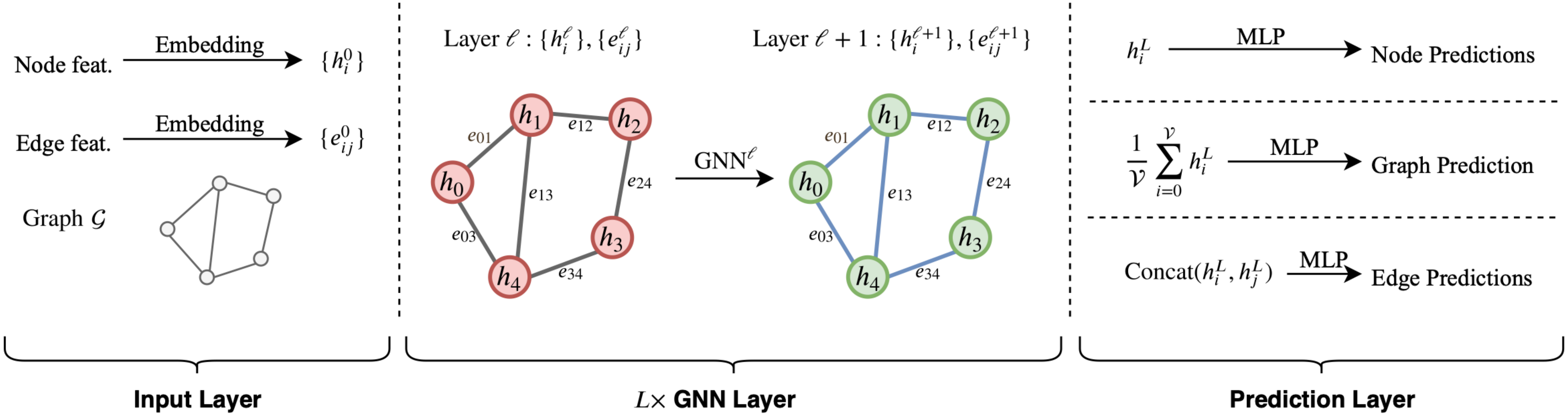
Multi-layer perceptron (MLP) model



Graph neural network

- Data are prepared with a graph structure:

https://graphdeeplearning.github.io/post/benchmarking-gnns/



Point: trainable parameters exist only node-wise and edge-wise embedding
→ Graph Neural Network (GNN) can handle the variable number of nodes and edge very naturally

Datasets

- Experiments are performed using "parallel workloads archive" (open data)
  - Contains historical job accounting information

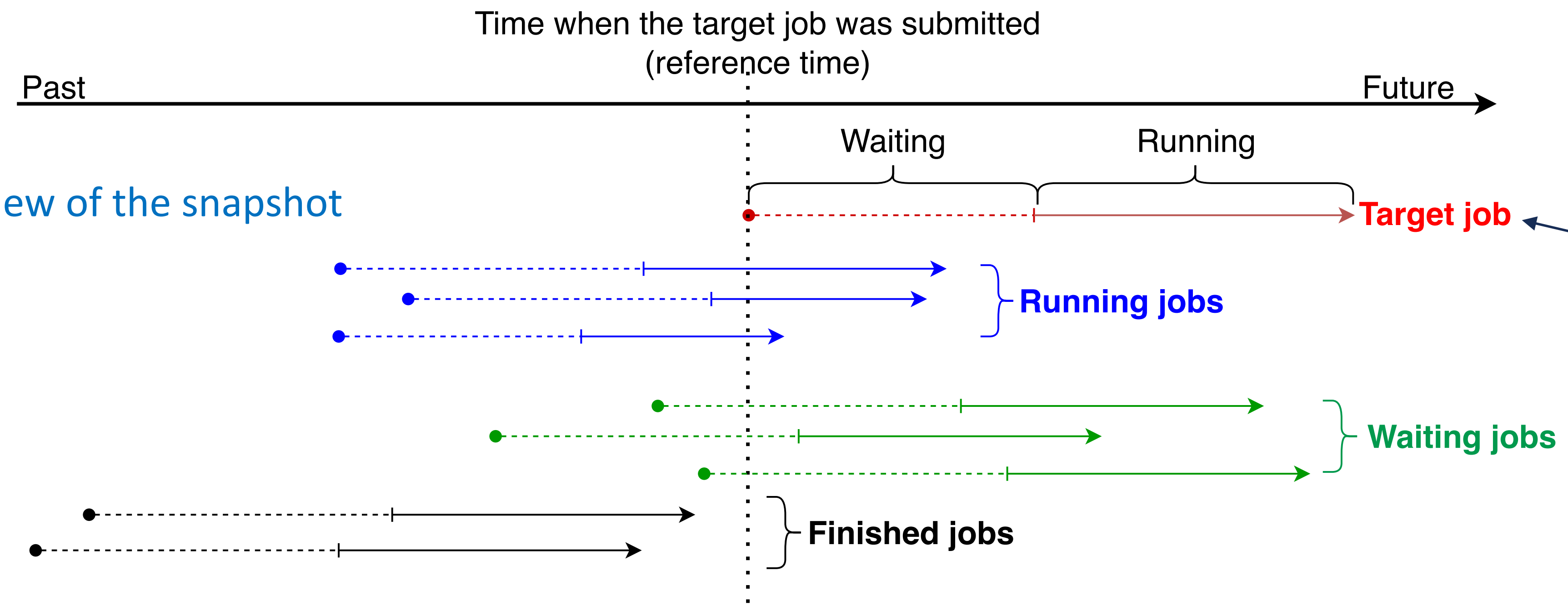
Name	Job scheduler	Training data	Validation data	Test data
SDSC_BLUE	Catalina [1]	186,050 [2000–04–30 to 2002–05–30]	23,256 [2002–05–30 to 2002–08–29]	23,256 [2002–08–29 to 2002–12–30]
HPC2N	Maui [4]	162,297 [2002–08–01 to 2005–04–13]	20,287 [2005–04–13 to 2005–06–13]	20,287 [2005–06–13 to 2006–01–16]
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 2002–09–01]
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

- 6 datacenters are selected to examine different types of job schedulers
- Datasets are split into training, validation, and test data with a ratio of 80%:10%:10%
  - DL model needs to acquire the capability of the prediction in completely different time range
  - E.g) SDSC_BLUE dataset: `2000-04-30 to 2002-05-30` is train data, and `2002-08-29 to 2002-12-30` is test data

Snapshots

- Snapshots of the scheduler are reconstructed from the accounting data
  - 1 snapshot = 1 input data of DL model

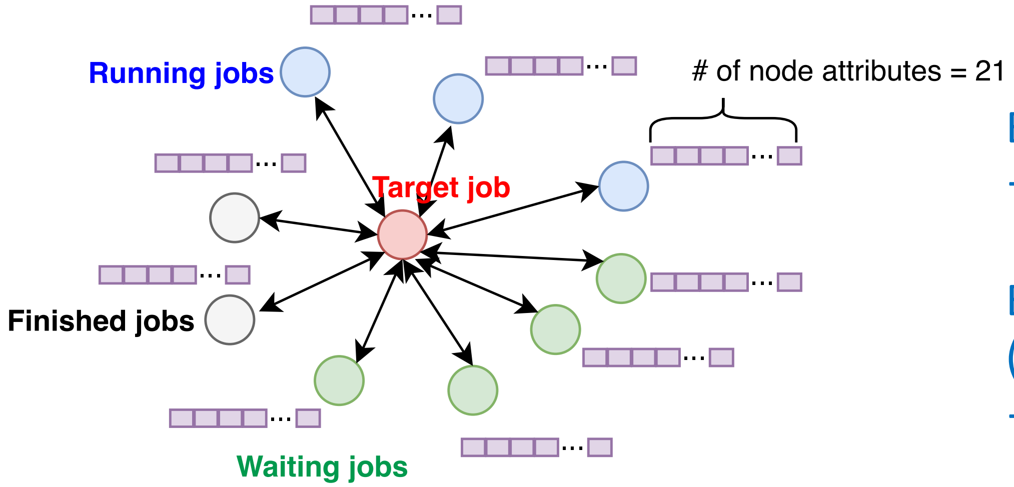
Overview of the snapshot



The job of interest that we aimed at predicting the wait time

Input variables and graph data

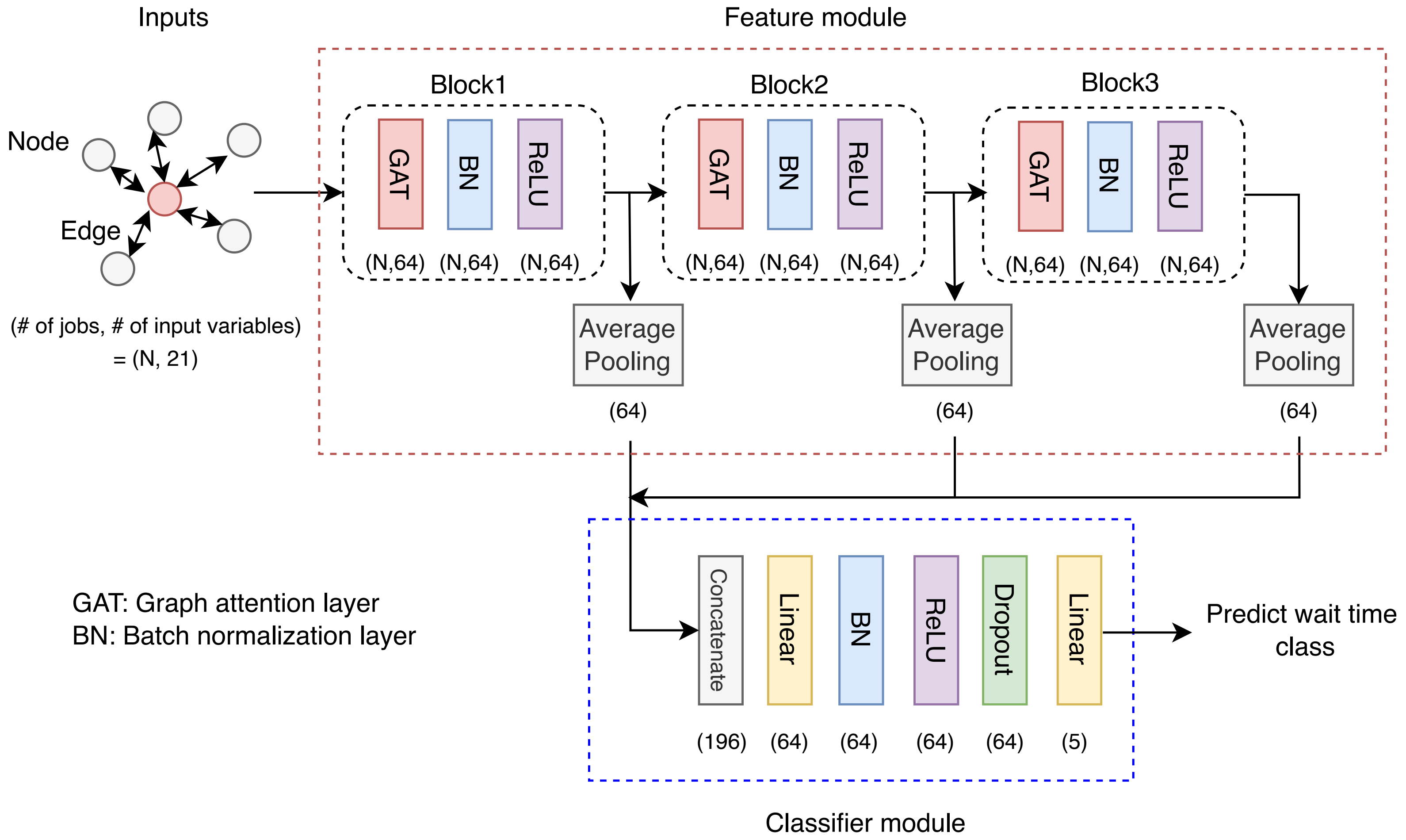
- 21 input variables are defined for each job
  - USER_ID, REQEST_CORE, REQUEST_TIME… etc
  - E.g.) if there are 10 jobs in the snapshot, the total number of input variables is 21 x 10 = 210
- Graph structure data are prepared from the snapshot



Each node corresponds to each job
→ 21 input variables are assigned to node attributes

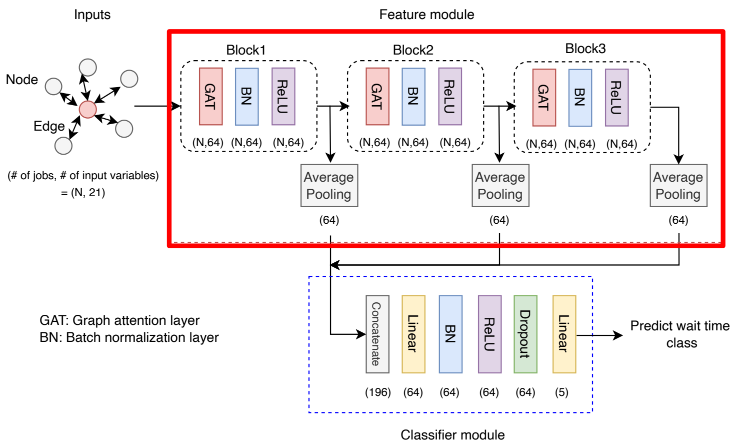
Edges are prepared between the target job and other jobs (bi-directional)
→ Job information will be exchanged along with the edges

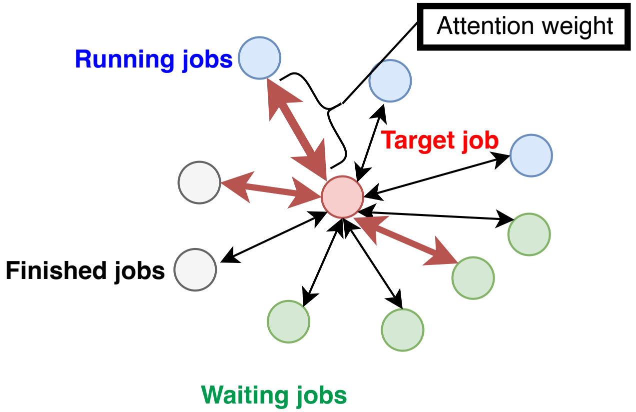
Model overview



GAT: Graph attention layer
BN: Batch normalization layer

Feature module

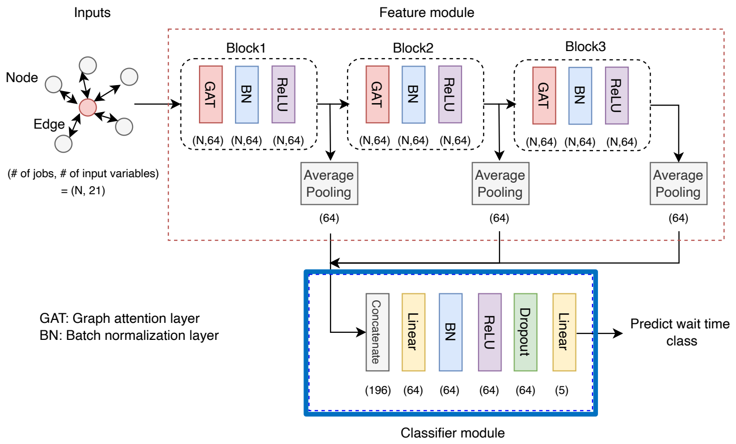




- Feature module is aimed at extracting global features of the snapshot
- Graph Attention Network (GAT) is employed
  - Importance of the relation between the target job and other jobs (attention weight) is learned as edge attribute

→ Improve the learning efficiency and explainability by visualizing the attention weights (will be discussed later)

Classifier module



- Classifier module is aimed at predicting the wait time classes
  - Fully connect layer, batch normalization, ReLU, dropout
  - 5 prediction classes are defined in this study

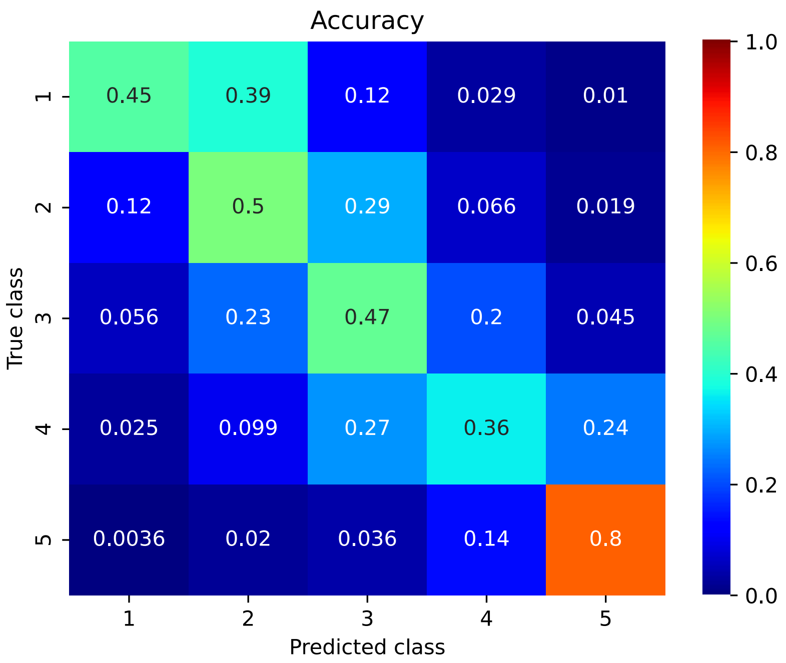
Prediction class index	Definition
1	(wait time) $<$ 1 minute
2	1 minute $\leq$ (wait time) $<$ 10 minutes
3	10 minutes $\leq$ (wait time) $<$ 1 hour
4	1 hour $\leq$ (wait time) $<$ 10 hours
5	10 hours $\leq$ (wait time)

Training details

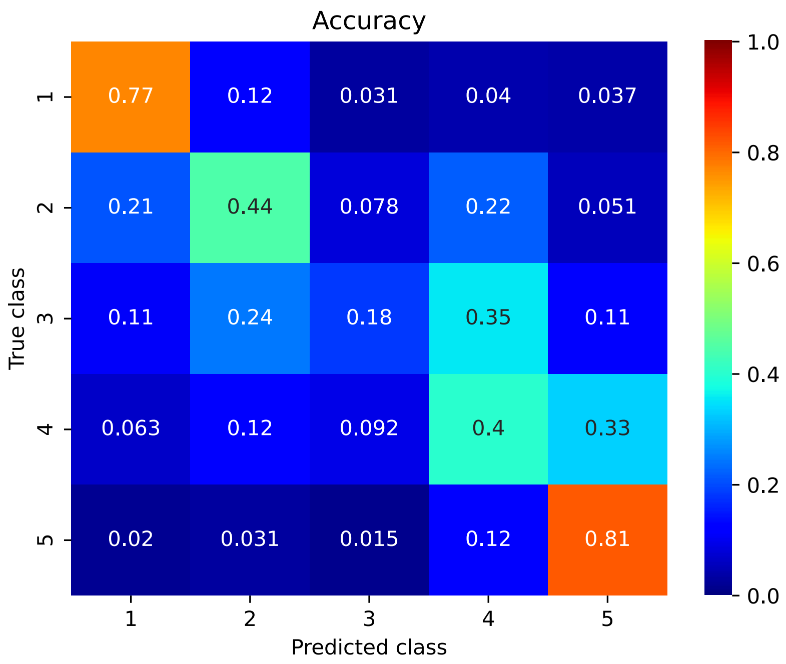
- Pytorch + DGL libraries are used, our codes are available in GitHub
- All executions used a local cluster of NVIDIA A100 graphics cards
  - 40GB GPU memory for each card
- The training is performed for up to 30 epochs
  - The best epoch for the validation data is used as the final weight parameters
  - Cross-entropy loss is used as loss function, and the SGD algorithm is used as optimizer
  - Batch size is 128, and the learning rate is 0.01
  - Other hyperparameters (e.g. # of nodes in GAT layer) are optimized by a grid search

Results: confusion matrix

SDSC_BLUE



HPC2N



- Confusion matrix for the test data
- As a global trend, middle range of classes is difficult to predict

  → Consistent with previous study by IN2P3 team

- Overlearning is main concern to improve the performance

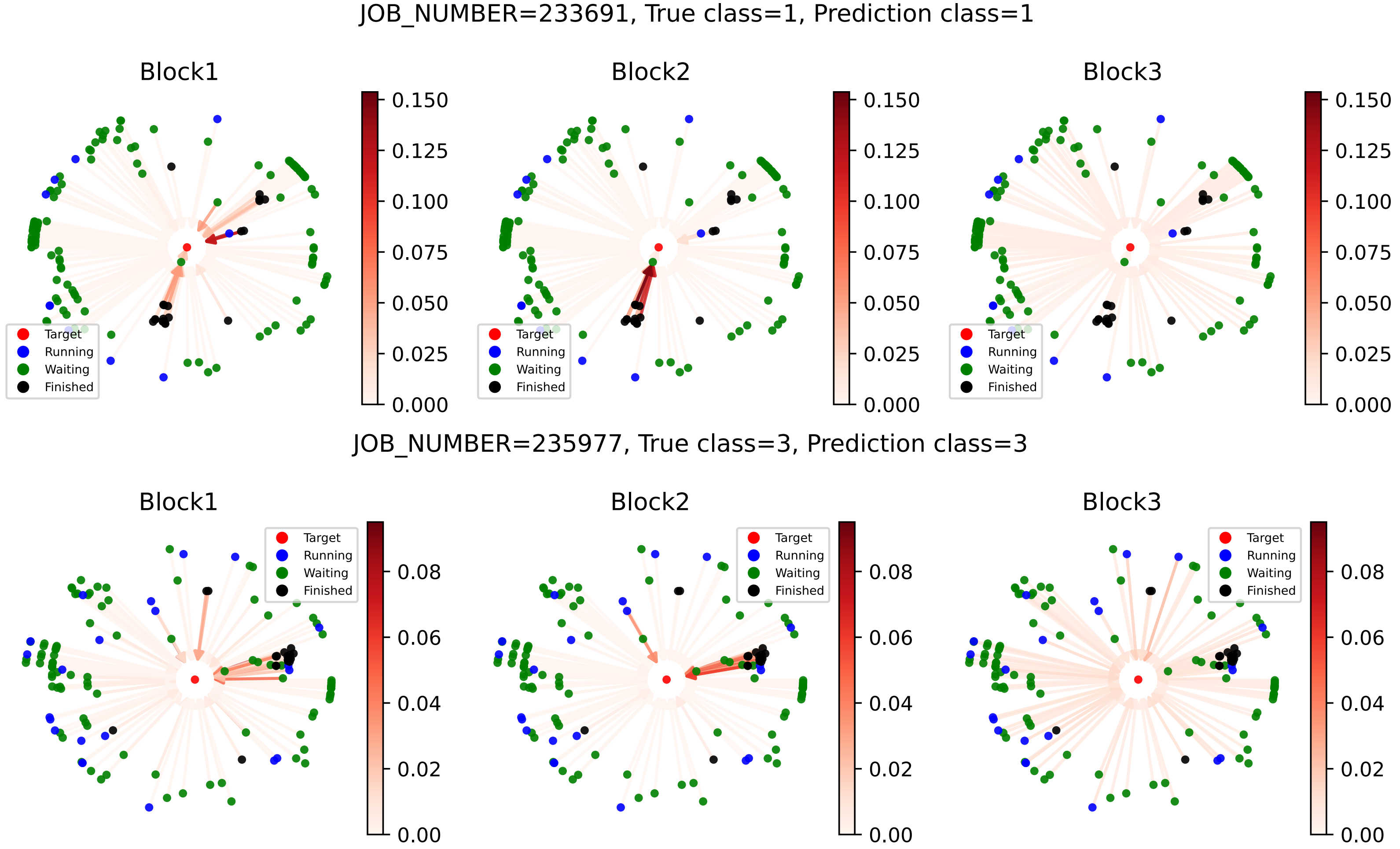
Results: comparison with other methods

- MLP and BDT models are executed and compared with our model
  - Need to prepare the fixed length of data, N jobs are selected from the snapshot

Dataset	Our model	MLP N=150	MLP N=300	BDT N=150	BDT N=300
SDSC_BLUE	**0.517 ± 0.016**	0.447 ± 0.012	0.446 ± 0.010	0.422 ± 0.017	0.418 ± 0.012
HPC2N	**0.522 ± 0.024**	0.457 ± 0.022	0.398 ± 0.010	0.421 ± 0.024	0.382 ± 0.020
ANL_Intrepid	**0.470 ± 0.020**	0.381 ± 0.028	0.388 ± 0.037	0.408 ± 0.018	0.408 ± 0.020
PIK_IPLEX	**0.322 ± 0.029**	0.307 ± 0.028	0.240 ± 0.026	0.262 ± 0.016	0.242 ± 0.024
RICC	**0.457 ± 0.026**	0.371 ± 0.028	0.373 ± 0.042	0.321 ± 0.035	0.332 ± 0.027
CEA_CURIE	**0.555 ± 0.030**	0.324 ± 0.030	0.311 ± 0.023	0.383 ± 0.017	0.365 ± 0.019

→ Our proposed model outperforms traditional methods
→ GNN can process our job information efficiently ☺

Results: attention weights



- Large attention weights for recently finished jobs

→ DL model seems to utilize past experiences (?)

Summary

- Proposed an efficient approach based on the GNN
  - Our model outperforms MLP and BDT models
  - Overlearning is a main concern:
    - Transfer learning is a feasible approach: SiteA → SiteB
  - The current study was submitted to JSSPP 2023 workshop: https://jsspp.org/
    - Acceptance rate is ~50%
- Future plans:
  - Latency of the prediction is not studied well yet
    - FPGA card (ALVEO) has been procured
  - KEKCC real accounting information (LSF) will be checked

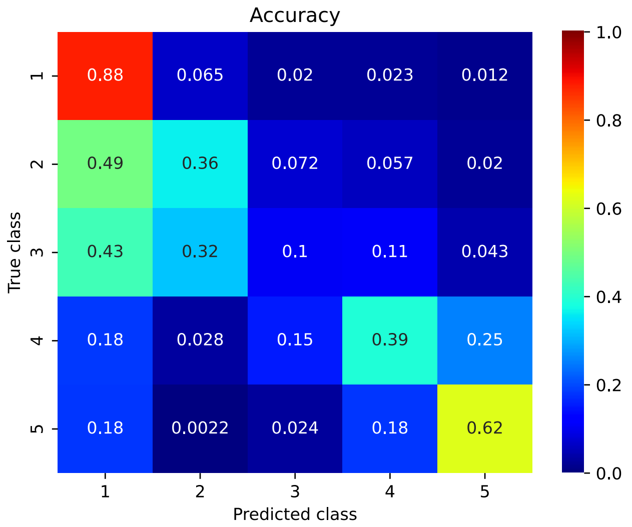
Backups

Input variables

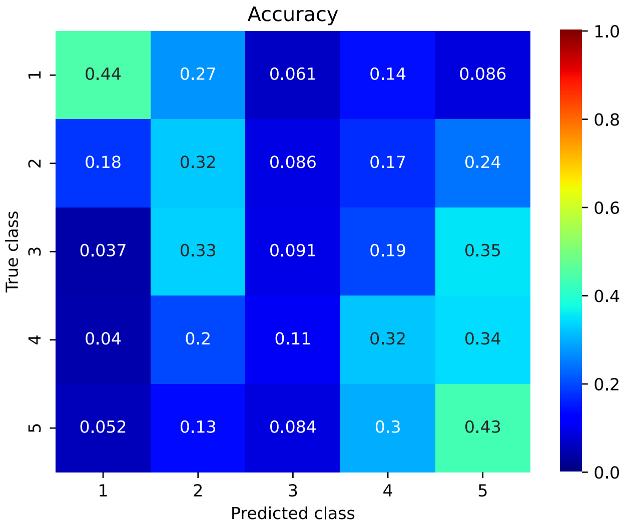
ID	Name	Description
1.	JOB_NUMBER*	A job identifier indicated by an integer.
2.	SUBMIT_TIME	The difference between the job's submission time and the reference time, in seconds.
3.	WAIT_TIME	The running and finished jobs: the difference between the job's submission time and the start time, in seconds. 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 in 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 represents 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.

Results: confusion matrix

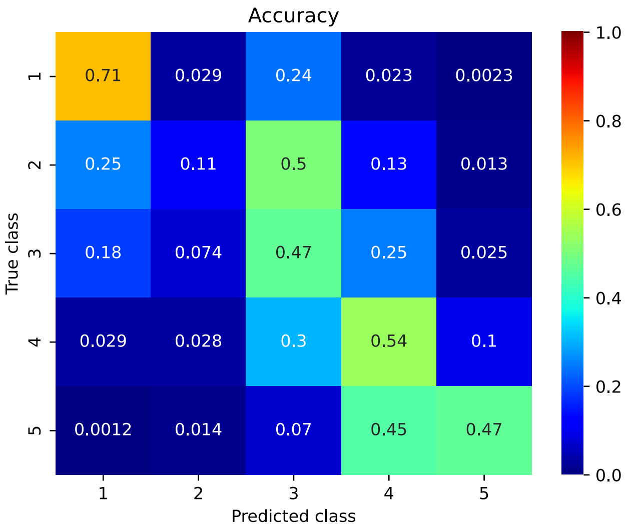
ANL_Intrepid



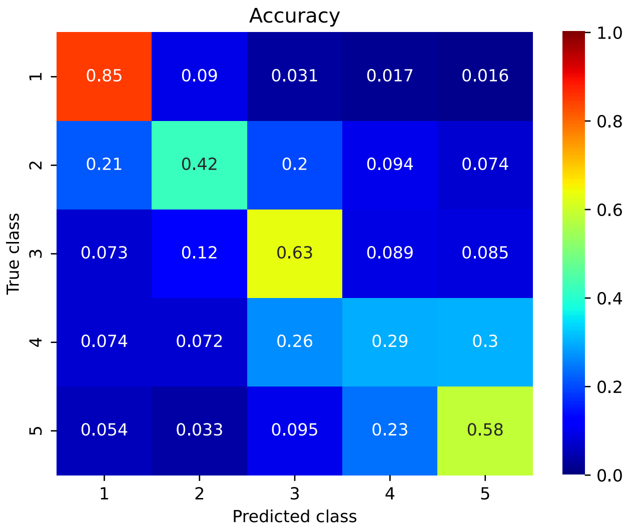
PIK_IPLEX



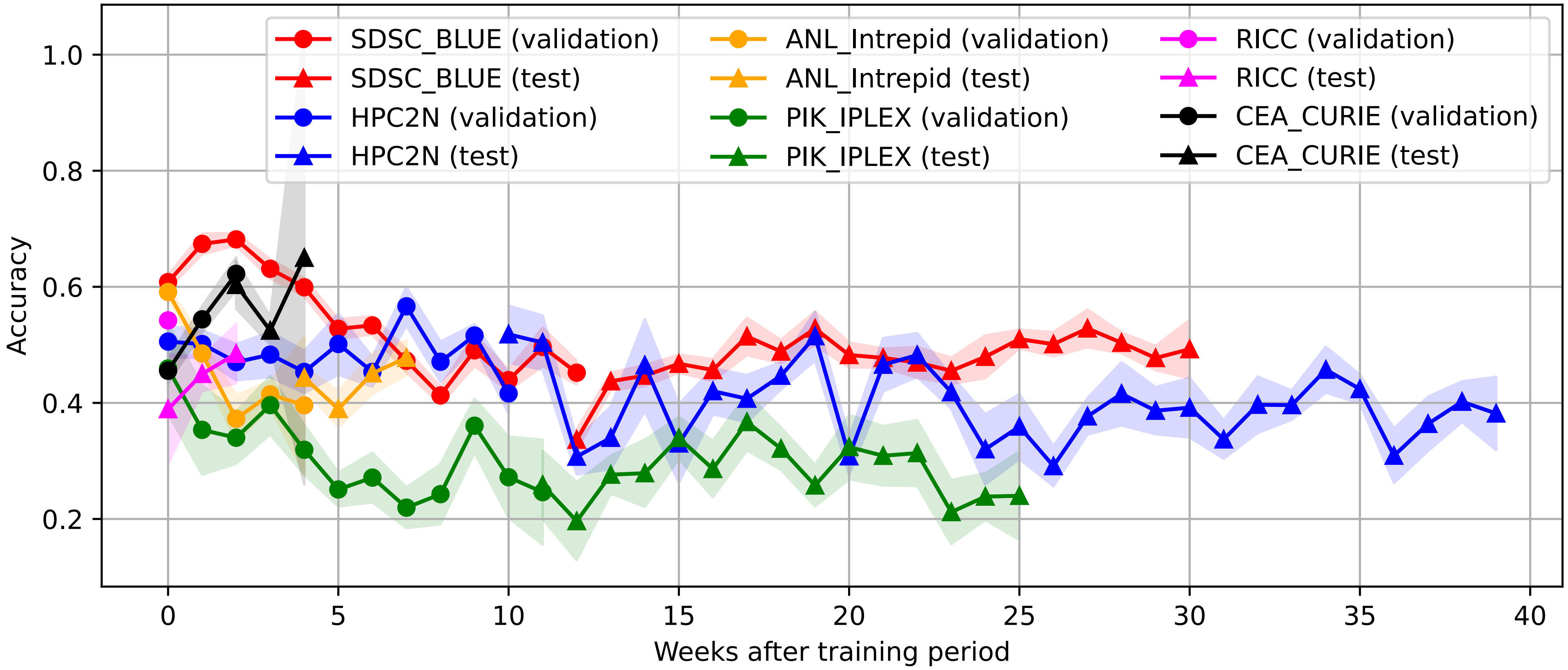
RICC



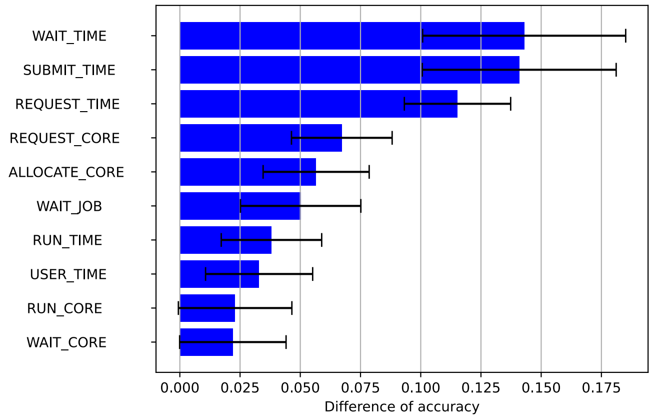
CEA_CURIE



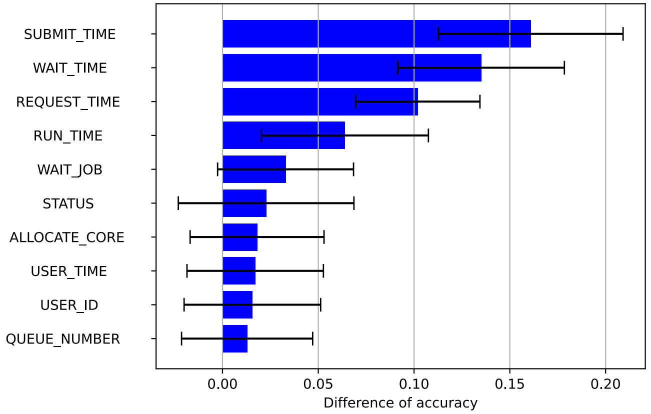
Results: time dependency



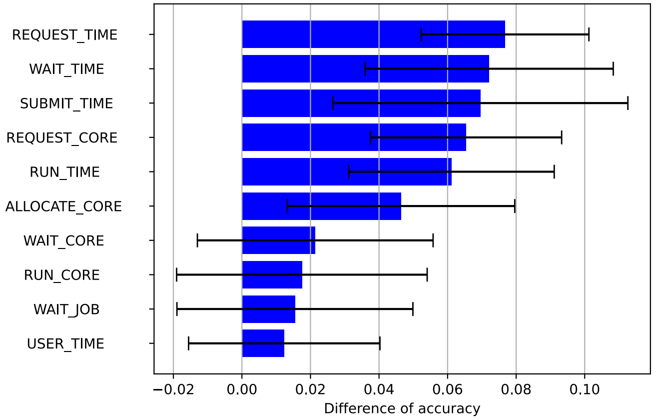
Results: PFI



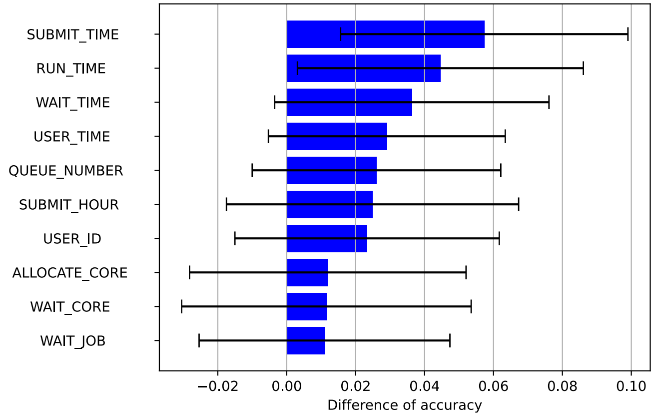
WAIT_TIME
SUBMIT_TIME
REQUEST_TIME
REQUEST_CORE
ALLOCATE_CORE
WAIT_JOB
RUN_TIME
USER_TIME
RUN_CORE
WAIT_CORE
0.000 0.025 0.050 0.075 0.100 0.125 0.150 0.175
Difference of accuracy



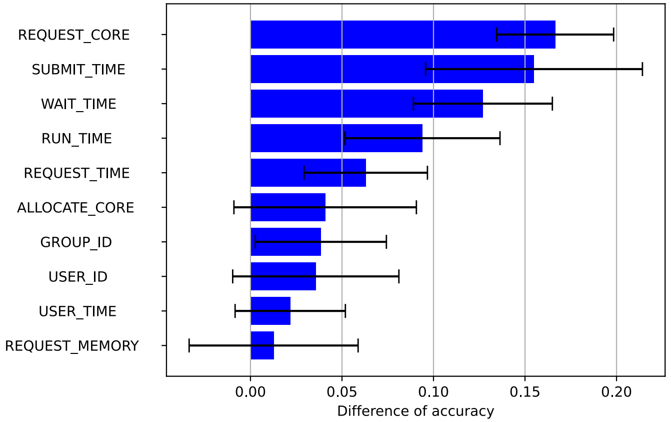
SUBMIT_TIME
WAIT_TIME
REQUEST_TIME
RUN_TIME
WAIT_JOB
STATUS
ALLOCATE_CORE
USER_TIME
USER_ID
QUEUE_NUMBER
0.00 0.05 0.10 0.15 0.20
Difference of accuracy



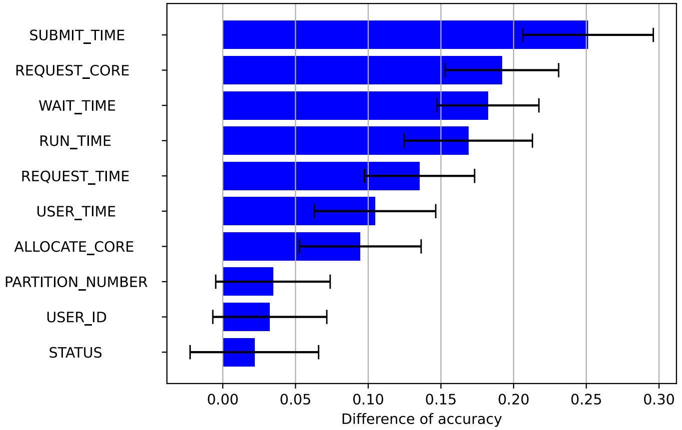
REQUEST_TIME
WAIT_TIME
SUBMIT_TIME
REQUEST_CORE
RUN_TIME
ALLOCATE_CORE
WAIT_CORE
RUN_CORE
WAIT_JOB
USER_TIME
−0.02 0.00 0.02 0.04 0.06 0.08 0.10
Difference of accuracy



SUBMIT_TIME
RUN_TIME
WAIT_TIME
USER_TIME
QUEUE_NUMBER
SUBMIT_HOUR
USER_ID
ALLOCATE_CORE
WAIT_CORE
WAIT_JOB
−0.02 0.00 0.02 0.04 0.06 0.08 0.10
Difference of accuracy



REQUEST_CORE
SUBMIT_TIME
WAIT_TIME
RUN_TIME
REQUEST_TIME
ALLOCATE_CORE
GROUP_ID
USER_ID
USER_TIME
REQUEST_MEMORY
0.00 0.05 0.10 0.15 0.20
Difference of accuracy



SUBMIT_TIME
REQUEST_CORE
WAIT_TIME
RUN_TIME
REQUEST_TIME
USER_TIME
ALLOCATE_CORE
PARTITION_NUMBER
USER_ID
STATUS
0.00 0.05 0.10 0.15 0.20 0.25 0.30
Difference of accuracy