POLITECNICO DI TORINO
Repository ISTITUZIONALE

A machine learning approach for an HPC use case: The jobs queuing time prediction

Original
A machine learning approach for an HPC use case: The jobs queuing time prediction / Vercellino, Chiara; Scionti, Alberto; Varavallo, Giuseppe; Viviani, Paolo; Vitali, Giacomo; Terzo, Olivier. - In: FUTURE GENERATION COMPUTER SYSTEMS. - ISSN 0167-739X. - ELETTRONICO. - 143:(2023), pp. 215-230. [10.1016/j.future.2023.01.020]

Availability:
This version is available at: 11583/2975716 since: 2023-06-21T06:50:38Z

Publisher:
Elsevier

Published
DOI:10.1016/j.future.2023.01.020

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

26 April 2026

Future Generation Computer Systems 143 (2023) 215–230

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A Machine Learning Approach for an HPC Use Case: the Jobs Queuing Time Prediction

Chiara Vercellino [a,b,*], Alberto Scionti [a], Giuseppe Varavallo [a,c], Paolo Viviani [a], Giacomo Vitali [a,b], Olivier Terzo [a]

[a] LINKS Foundation, Via P.C. Boggio 61, 10138 Turin, Italy
[b] Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy
[c] Università di Torino, Lungo Dora Siena 100A, 10153 Turin, Italy

ARTICLE INFO

Article history:
Received 20 January 2022
Received in revised form 27 October 2022
Accepted 28 January 2023
Available online 2 February 2023

Keywords:
High performance computing
Queues
Batch scheduler
Automatism
Machine learning
Uncertainty quantification

ABSTRACT

High-Performance Computing (HPC) domain provided the necessary tools to support the scientific and industrial advancements we all have seen during the last decades. HPC is a broad domain targeting to provide both software and hardware solutions as well as envisioning methodologies that allow achieving goals of interest, such as system performance and energy efficiency. In this context, supercomputers have been the vehicle for developing and testing the most advanced technologies since their first appearance. Unlike cloud computing resources that are provided to the end-users in an on-demand fashion in the form of virtualized resources (*i.e.*, virtual machines and containers), supercomputers' resources are generally served through State-of-the-Art batch schedulers (*e.g.*, SLURM, PBS, LSF, HTCondor). As such, the users submit their computational jobs to the system, which manages their execution with the support of queues. In this regard, predicting the behaviour of the jobs in the batch scheduler queues becomes worth it. Indeed, there are many cases where a deeper knowledge of the time experienced by a job in a queue (*e.g.*, the submission of check-pointed jobs or the submission of jobs with execution dependencies) allows exploring more effective workflow orchestration policies. In this work, we focused on applying machine learning (ML) techniques to learn from the historical data collected from the queuing system of real supercomputers, aiming at predicting the time spent on a queue by a given job. Specifically, we applied both unsupervised learning (UL) and supervised learning (SL) techniques to define the most effective features for the prediction task and the actual prediction of the queue waiting time. For this purpose, two approaches have been explored: on one side, the prediction of ranges on jobs' queuing times (classification approach) and, on the other side, the prediction of the waiting time at the minutes level (regression approach). Experimental results highlight the strong relationship between the SL models' performances and the way the dataset is split. At the end of the prediction step, we present the uncertainty quantification approach, *i.e.*, a tool to associate the predictions with reliability metrics, based on variance estimation.

© 2023 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

High-Performance Computing (HPC) paradigm is one of the major enabling factors for the scientific advancements we all have seen in the last decades. Indeed, supercomputers continuously raise the bar for performance in each generation, opening the door to get new insights into complex physical phenomena and allowing us to explore innovative designs for complex industrial systems at a faster pace than ever. Furthermore, with the progress of manufacturing technology, more powerful CPUs are used to complete, in a shorter time than before, ever-large numerical simulations. At the same time, adding specialised hardware co-processors (*i.e.*, GPUs, FPGAs, ASICs) further allows for gaining a large performance boost. As such, the next generation of supercomputers promises to crunch more than 10^{18} floating-point operations per second (*i.e.*, 1 ExaFlop/s), providing unprecedented computing power for better supporting engineering, drug designing and many scientific tasks, to mention a few.

Besides the historical task of supporting numerical simulations, supercomputers have recently started to embrace the domain of artificial intelligence (AI), where their computing power is used to train and run very large machine learning (ML) models.

* Corresponding author.
E-mail addresses: chiara.vercellino@linksfoundation.com (C. Vercellino), alberto.scionti@linksfoundation.com (A. Scionti), giuseppe.varavallo@linksfoundation.com (G. Varavallo), paolo.viviani@linksfoundation.com (P. Viviani), giacomo.vitali@linksfoundation.com (G. Vitali), olivier.terzo@linksfoundation.com (O. Terzo).

https://doi.org/10.1016/j.future.2023.01.020
0167-739X/© 2023 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Indeed, machine learning techniques are foreseen as a profitable way of improving traditional simulations in terms of results' quality and simulation speed. Fast technological advancements provided by heterogeneous hardware are also driving the research on the topic, with an increasing number of works focusing on better supporting deep learning (DL) models at very large scales. While recently, many application workflows have been designed to include ML/DL tasks, the usage of these techniques is not limited to the application layer. Many works leveraged machine learning to optimise specific aspects of the use of computing resources in large-scale systems, *e.g.*, improving energy efficiency [1,2], server consolidation [3], and optimising network topology [4].

Despite the architectural improvements and the advancements of the connected software stacks, the way supercomputing resources are provisioned to the end-users almost remained unchanged for a long time. As in the past decades, supercomputing resources still remain accessed through a *batch scheduler* [5], which applies an internal scheduling policy to guarantee the fair use of the system among the users. More in detail, batch schedulers provide the HPC administrators with the ability to define specific *queues* for provisioning specific resources (*e.g.*, access to GPU-accelerated nodes) or computing resources according to some specific policy. Then, sophisticated heuristic priority functions are used to prioritise and schedule jobs, still guaranteeing fair access to the requested resources by all the users. Generally, these functions are set by system administrators at the beginning but hardly adapt over time to the changes in the workloads and optimisation goals, potentially leading to the degraded overall efficiency of the system. Some alternatives for overcoming the limited flexibility offered by batch schedulers have already been proposed [6,7], but their large adoption is far away. Furthermore, a complete change in the way resources are provisioned that implies removing batch schedulers is further far away. Interestingly, some approaches based on ML techniques have been proposed, already showing promising results [8].

The motivation for this work emerged in the context of two European projects (*i.e.*, the H2020-LEXIS and the EuroHPC–JU ACROSS projects) focused on the execution of complex scientific workflows on HPC and cloud environments: in this context, being able to reliably predict when a job will be executed on an HPC cluster can improve the overall experience and may lead to novel scheduling strategies. As such, the capability of predicting queuing time becomes worth it in many cases; for instance, optimising the execution of a workflow by reducing the (wasted) time between the execution of dependent jobs, or in the case that a given job, with an execution time exceeding the maximum allowed time on a given queue, needs to be checkpointed and restarted from the previous state. As a matter of that, the objective of a job allocation policy based on the presented prediction capability is closer to the application perspective, since it can be implemented by an orchestration system which has the visibility of the entire workflow to be executed. Thus, such an orchestration system would rely on the evaluation of the makespan time for a given workflow as the optimisation criterion targeting the optimal schedule and submission of the jobs on the queuing system. Similarly, the minimisation of the overall execution time for a job exceeding the maximum allotted time on a specific queue can be used as an optimisation criterion. Similar requirements emerged in the context of the LEXIS project, where some workflows were required to execute 'long-running' jobs (Computational Fluid Dynamics simulations) whose overall execution time exceeded the maximum allotted time to any queue of the HPC centre. In a more general perspective, being able to predict the time jobs spend in a queue with reasonable accuracy will enable to set up orchestration strategies attempting to provide more deterministic execution of the application workflows, by hiding the timing variations coming with complex and dynamic systems as those associated to batch scheduler queues are.

Thus, this paper reports the investigation performed on historical data collected on HPC clusters, as well as the results from the ML techniques applied to predict the queue waiting time. Precisely, the main contribution is threefold:

- We propose an automated procedure, which combines Unsupervised Learning (UL) and Supervised Learning (SL) techniques, to predict queue waiting times in an HPC cluster. The UL part corresponds to the preprocessing phase, where State-of-the-Art approaches are used to discover patterns between jobs' features and individuate the key ones for the prediction purpose. The SL part regards the actual predictions of the target feature (*i.e.*, the queue waiting time).
- Among the many SL models, we investigated different classification and regression models to individuate candidate models on the analysed dataset. When targeting the classification approach, a major focus is on class definitions.
- Finally, we highlight critical aspects when dealing with similar case studies. The study on different dataset splittings evidences the relevance of the temporal components that impacts data distributions. Then an uncertainty quantification approach is proposed to evaluate the reliability of the predictions and find correlations with the errors' distributions.

The remainder of the paper is organised as follows. In Section 2, we describe the most relevant research works concerning the optimisation and prediction of queuing systems, focusing on HPC-related ones. Section 3 is devoted to describing the approach used to collect the dataset coming from one of the HPC systems exploited in the LEXIS and ACROSS projects, along with the main attributes that describe the jobs and which are closely connected to the information used by the batch scheduler to make decisions upon their execution. We also describe the way the resulting dataset has been organised and how it has been split into training and test sets supporting SL prediction models. While the attributes reported in this paper are linked to a specific batch scheduler system (Altair PBS Professional[1]), the proposed methodology applies to any other batch scheduler system. This section also introduces the different machine learning models used in the remainder of the paper, along with their performance and uncertainty metrics. Section 4 deeply analyses the results obtained using the different models on the acquired dataset and validates the methodology on the dataset from the Karlsruhe Institute of Technology ForHLR II (KIT FHII) System, whose workload files are available online[2]. It highlights the strong relationship between predictions' performance and how the training set is extracted from the overall collected data. It also shows how much uncertainty inherent in the data could affect the predictions. Finally, Section 5 summarises the main achievements described in this work and indicates some interesting future directions of investigation.

2. Related work

The provisioning of supercomputing resources is still dominated by the batch model, where jobs are submitted to a (batch)

[1] https://www.altair.com/pbs-professional/

[2] https://www.cs.huji.ac.il/labs/parallel/workload/ (accessed: 10/2022)

scheduler that decides which jobs to execute based on sophisticated heuristic priority functions. In this context, jobs are organised into different queues, allowing the system administrators to differentiate among kinds of resources to access and access priorities. As such, the optimal management of the jobs entering these queuing systems becomes the major factor in improving performance. To this purpose, previous works focused on distinct aspects of the 'optimal' management of the jobs: on one side, some works focused on optimising the schedule of the jobs; on the other hand, other works looked at how to make reliable predictions of the behaviour of the queuing systems.

Apart from the HPC context, the modelling of queuing systems and the implementation of associated prediction models have been considered in other domains, such as transportation systems and communication networks. For instance, the authors in [9] presented an analytical model of the aircraft departure process at an airport based on the transient analysis of the *D/E/1* queuing system. The authors showed how the proposed model could be successfully used to predict the expected runway schedule and takeoff times. Arnab et al. [10] deeply analysed the various models, effects, and management strategies involved in modern communication networks. Specifically, they surveyed different queuing models (*e.g.*, *M/M/1*, *M/M/m*, *M/G/1*, etc.) generally used to describe queuing systems' behaviours. In [11], the authors investigated the modelling and prediction of delays in communication networks with feed–forward neural networks. Similarly, Happ et al. [12] developed RouteNet, a graph neural network augmented to support multiple scheduling policies (*i.e.*, strict priority, deficit round robin, weighted fair queuing) and handle mixed scheduling policies, to estimate in advance the delays in the networks. Another domain where machine learning and deep learning techniques have been widely applied is that of cloud computing [13,14], where these techniques have been explored to overcome challenges such as finding the optimal allocation of virtual resource instances (*i.e.*, virtual machines—VMs, containers) to reduce overall energy consumption, increasing resource utilisation, guaranteeing Quality-of-Service (QoS) and Service Level Agreements (SLAs).

When we move to the specific HPC context, as previously stated, some previous works targeted either ameliorating the scheduling strategy of a batch scheduler or making reliable predictions related to the submitted jobs. In the first case, recent works have successfully applied meta-heuristics and optimisation approaches. As such, in [15,16] the authors considered multiple objectives to be optimised at the same time as the target goal of the scheduling strategy. In particular, they modelled the scheduling decision as an optimisation problem that was solved at run-time (actually, jobs laying within a certain time window are considered for the schedule, and the others remain in the same position within their queues) by using a genetic algorithm (GA). Fan et al. [17] also compared different scheduling strategies to address some of the issues raised in co-scheduling on-demand, rigid, and malleable jobs on a single supercomputer. RLScheduler [8] is another example of a machine learning technique used to overcome the general limits of traditional State-of-the-Art batch schedulers; specifically, it relies on reinforcement learning (RL) to learn from the historical data the best scheduling policy. The paper shows the capability of the RL approach to adapt to the changes in the load and system configurations over time.

Focusing on the application of ML techniques for prediction purposes, other works can be found in the literature. Soysal et al. [18] defined and trained an ML model using the metadata information associated with the jobs submitted to a batch scheduler. The purpose of the devised model was the accurate prediction of the jobs' wall time. When compared with our approach, the authors targeted a different but rather challenging problem. Despite the presented results being encouraging, the higher variability of the target prediction feature made us lean towards the prediction of the queue waiting time. This choice also opens the door to the implementation of workflow orchestration strategies that aim at providing more deterministic execution of jobs with execution dependencies. Wang et al. [19] worked on run-time predictions: the authors collected data for three months, reaching nearly 26 thousand jobs' samples; then, their ML approach combined unsupervised learning methods, to cluster similar jobs, and supervised learning algorithms to predict the running time, their best results were obtained combining *K-nearest neighbour (KNN)* with a *support vector regressor (SVR)*. A major focus was on reducing the underestimation rate (*i.e.*, predicting run-time values that are lower than the actual ones). Some similarities can be found with our proposed methodology, which still combines unsupervised and supervised techniques; however, a major difference lies in our choice of using unsupervised techniques to perform an automated preprocessing step on the acquired input dataset. The run-time prediction has been also investigated in [20], where the information regarding the last two submitted jobs (by the same user) was exploited to estimate the run-times and enhance a back-filling strategy. Also, in [21], the authors performed run-time estimations on jobs submitted through the SLURM batch scheduler (nearly 10^5 samples were collected, with 25% of samples used as the test set). In this case, the best results emerged from a classification approach that exploited *Naïve Bayes classifier (NB)* and *support vector machine (SVM)*. Historical data were also used by Smith et al. [22] for run-time prediction, introducing similarity metrics on jobs' features and aggregating historical run-time as predictions for newly submitted jobs.

Apart from solely run-time predictions, another work focused on predicting also the memory footprint of the jobs in the HPC context. As such, in [23], different supervised ML techniques, *i.e.*, *Linear Regression*, *Lasso Regression*, *Ridge Regression*, *Decision Tree Regression (DTR)*, and *ElasticNetCV Regression*, are applied to both predict the concerned features, *i.e.*, the jobs run-time and their memory usage, and to establish which of the two could be the most relevant to predict to reduce the resources wasting. Conversely, concerning the literature on predicting the queue waiting time (hereafter referred to as 'queuing time') for the submitted jobs, the authors in [24] used a *Naïve Bayes classifier (NB)* on eight months based dataset, where the classifier was combined with classification probability adjustments to define classes boundaries. Furthermore, the analysis of the binning strategy used to classify the queuing time is a contact point with our work. Finally, Nurmi et al. [25,26] focused on finding boundaries, more specifically upper bounds (UB), on the queuing time, by exploiting the samples collected during 9 years from different supercomputers. Their methodology is based on time series (TS) and automated changing points detectors; a very relevant characteristic of their work regards the implementation of an automated downtime detector to identify systemic failures that affect jobs' queuing delay. In [27], the authors targeted the prediction of the queuing time by training a Hidden Markov Model (HMM) on historical data sampled from the PBSpro batch scheduler (the targeted supercomputer was the Nurion, which is managed by the National Supercomputing Center at the Korea Institute of Science and Technology Information—KISTI) over a period of six months (14,759 jobs in total). In this study, the dataset was filtered to eliminate the outliers before training the HMM model, and a preliminary analysis of the dataset was performed to determine the degree of correlation between the tracked job's features and the observed queuing times. Then, the HMM model is used to estimate the waiting time in the next period (experiments considered 6 prediction classes corresponding to predicting queuing

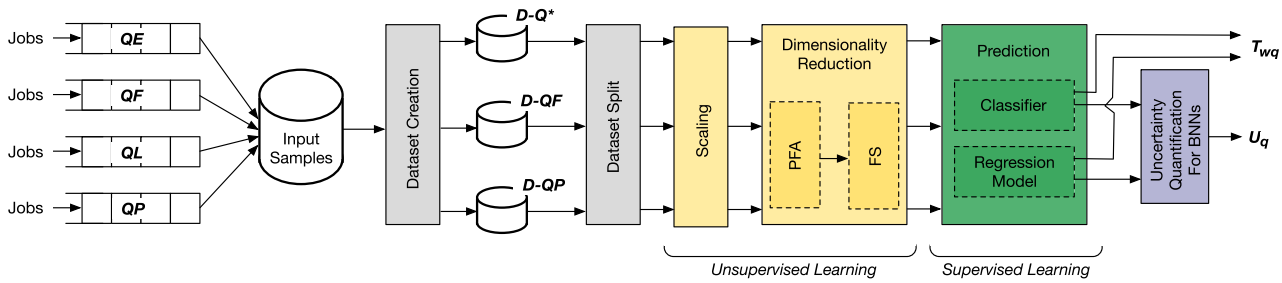


Fig. 1. Scheme of the proposed methodology from dataset collection to jobs' queuing times' prediction (T_{wq}) and uncertainty quantification (made for Bayesian Neural Network models) U_q.

Table 1
Related works summary: comparison of results and methodology of previous works concerning HPC predictions.

Ref.	Pred. target	ML Method	Performance
[18]	job walltime	AutoML [28]	Cumulative R^2 scores
[19]	job walltime	KNN + SVR	MAE: 46 min
[21]	job walltime	NB, SVM	F1 score: 0.79–0.81
[22]	job walltime	clustering + avg	MAE: 40–119 min
[23]	job walltime	DTR	R^2: 0.611
[23]	job memory usage	DTR	R^2: 0.638
[24]	job queuing time	NB	Accuracy: 0.78-0.86
[25]	job queuing time	BMBP for UB	95% level of CI
[26]	job queuing time	UB from TS	95% level of CI
[27]	job queuing time	HMM	Accuracy: 0.3891

time for 6 different time ranges, *e.g.*, less than 1 min, between 1 min and 30 min, etc.), based on the observed past waiting times.

Table 1 summarises the best results obtained by previous works, exploiting ML techniques to predict different features of interest in the HPC context.

3. Queuing time prediction with machine learning

Accurate queuing time predictions represent one of the enabling factors for improving the execution of modern HPC workflows, by allowing to devise the appropriate point in time when submitting the jobs to the batch scheduler. For this purpose, we figured out a pure machine learning approach, which mixes both unsupervised learning (UL) and supervised learning (SL) techniques to perform: (*i*) the preprocessing of the input dataset to extract the most relevant features for the queuing time prediction (here, we used UL techniques); (*ii*) the fitting of SL models to predict the queuing time of newly submitted jobs. Fig. 1 depicts the overall approach we followed. In the following, we deeply describe the methodology which covers the whole process, from the collection of data coming from a production supercomputer to the final steps that involve predictions and uncertainty quantification.

3.1. Dataset description

The input dataset is of paramount importance in using machine learning techniques, especially in the case of SL ones, since it determines the knowledge base from which the algorithms 'learn'. Datasets publicly accessible exist[3]; however, to provide significant results in the context of the ACROSS and LEXIS projects, we referred to a real-world supercomputer involved in both projects. Thus, we collected data for 34 days from an HPC cluster, consisting of 192 standard computational nodes, 8 GPU-equipped compute nodes and 1 fat node endowed with 6 TB RAM. The data were collected by periodically querying the batch scheduler (*i.e.*, in our case, it was the Altair PBS Professional, hereafter referred to as PBS), allowing us to keep track of the jobs' status evolution over time. Keeping in mind that HPC cluster resources are provisioned through a queuing system, we investigated the most peculiar characteristics of this cluster's queues. It is worth mentioning that, in some cases, access to dedicated queues is regulated by special authorisations. Given that, we exploited only data related to accessible resources: those associated with the freely accessible queue (hereafter referred to as *QF*), the production queue (we will refer to as *QP*), the long-running jobs queue (*QL*) and the short-running jobs queue (*QE*). It is relevant to notice that, even if these queues reflect the characteristics of specific types of HPC jobs, the proposed methodology is not directly linked to them. Therefore, the whole procedure can be replicated on other clusters which use different criteria to assign jobs to the relative partitions, *i.e.*, queues, and apply their custom limitations on users' requests.

Concerning the mentioned queues, PBS collects a large set of attributes that characterise the submitted jobs [29]. These attributes can be easily retrieved by querying the batch scheduler via a dedicated command (*qstatf*). This query provides JSON files containing all jobs as JSON elements. So, some further data engineering has been needed to map the samples to a dataset configuration that reflects the relationships among jobs: how many jobs precede in a queue for each job sample, what is the cluster status, concerning running jobs, when a new job is submitted, etc. Table 2 shows the overall attributes we associated with each job sample, including some hand-crafted features. Among these attributes, *Rattributes* and *Qattributes* are aggregated groups of attributes. They reflect jobs' characteristics at the cluster level (whenever the attribute is prefixed by *cluster*) or at the level of a queue (in this case, the attribute is prefixed by the queue's name). *Rattributes* refer to the running jobs, whilst *Qattributes* refer to jobs queued in the system and preceding the considered job.

From the batch scheduler command line interface (the PBS *qstat -xf* command in our case) is possible to collect the attributes of all the jobs that completed their execution; then, these attributes can be used as the targets (labels) of a prediction model. Among all the possible targets, we focused on predicting T_{wq}, *i.e.*, the of time jobs wait in the system queues before being scheduled for execution on the cluster's resources.

- T_{wq}: it indicates the total amount of time (measured in minutes) that a job has waited in the queue, and it is obtained from the *eligible_time* attribute.
- T_r: it indicates the total amount of time (measured in minutes) that a job has run, and it is obtained from the *resources_used:walltime* attribute.

[3] https://jsspp.org/workload/

Table 2
Main attributes related to the jobs submitted to PBS. Whenever Qattributes and Rattributes refer to a specific queue, this latter is selected by the *qname* belonging to one of the accessible queues, *i.e.*, *QE*, *QP*, *QL*, *QF*.

Attribute	Description	
priority	Priority score that incorporates queue priority, fair-share priority and eligible time	
rerunnable	Boolean attribute specifying if the job can be restarted from the beginning without harmful side effects	
qtime	Timestamp in which the job enters the queue	
queue	The queue to which the job is submitted	
state_history	A string that tracks the job's status changes	
walltime	The maximum running time requested by the user for the job	
fairshare	The fairshare priority score	
nodect	The number of execution nodes requested for the job	
ncpus	The number of CPUs requested for the job, it is a multiple of *nodect*	
Rattributes:		
	num_job	The number of jobs running
	nodect	The number of execution nodes requested by the jobs running
	used_cpupercent	The total CPU percentage of the jobs running
	used_cput	The total CPU time of the jobs running
	used_mem	The total physical memory used by the jobs running
	used_ncpus	The total number of CPUs used by the jobs running
	used_vmem	The total virtual memory used by the jobs running
	used_walltime	The total execution time actually used by the jobs running
	walltime	The total execution time requested for the jobs running
Qattributes:		
	priority	The total priority score
	eligible_time	The total waiting time, in minutes, of the jobs preceding the considered one
	ncpus	The total number of CPUs requested by jobs preceding the considered job
	nodect	The total number of execution node requested by jobs preceding the considered one
	num_job	The total number of jobs preceding the considered job
	walltime	The total walltime, in minutes, requested by jobs preceding the considered job

- CPU_{perc}: it expresses the percentage of CPU usage, summing up all the requested CPUs; it is obtained from the *resources_used:cpupercent* attribute.
- CPU_{time}: it indicates the total CPU time used by the job (expressed in minutes); it is obtained from the *resources_used: cput* attribute.
- Mem_{phy}: it provides the amount of physical memory used by a job (expressed in KB), and it is obtained from the *resources_used:mem* attribute.
- Mem_{vir}: it provides the size of virtual address space used by a job (expressed in KB), and it is obtained from the *resources_used:vmem* attribute.

It is worth mentioning that among the previously cited attributes, the T_{wq} of a considered job has a strong relationship with the T_r of all the jobs preceding it on the queuing system. So, in principle, being able to accurately predict T_r would imply an equally accurate prediction of T_{wq}. Anyway, this approach to predicting the queuing time brings along some drawbacks: (i) the prediction of T_r is highly dependent on the HPC jobs' specific use case, and there is no general relationship with other attributes that characterise the jobs, except maybe for the *walltime* that can be set by the users and may be significantly larger than T_r (ii) the batch scheduler implements its logic for HPC jobs allocation, that is usually quite complicated to reproduce faithfully, *e.g.*, back-filling strategies, urgent jobs submissions and jobs deletions, among the others should be handled carefully (iii) predicting directly T_{wq} may help in aggregating noise coming from scheduling strategies and users' overestimation of the needed *walltime*.

Table 3
The mean (measured in minutes) and standard deviation for the T_{wq} target attribute along with the explored queues.

Queue name	Mean [min.]	Standard deviation
QE	14	126.61
QF	192	514.15
QL	328	827.64
QP	122	215.95
all queues	156	409.68

The whole dataset has 90 columns, with 84 representing the features and 6 possible target labels. The dataset consists of 37,859 samples collected over 34 days and can be split as follows: 20,512 samples belong to the *QF* queue, 15,510 samples come from the *QP* queue, 1,692 samples have been collected from the *QE* queue, and finally, only 145 samples from the *QL* queue. These samples correspond to jobs whose entire execution process could be tracked, *i.e.*, from entering the queuing system to exiting it. Table 3 summarises some basic statistics (mean and standard deviation) on the T_{wq} attribute for the various accessed queues as, among the available target labels, it is the one of interest. From the samples' distribution among the queues, we can notice that 34 days would not be sufficient to train a separate prediction model for each of them: indeed, *QE* and *QL* have too few samples. However, some information could still be gathered through other queues' data to make predictions for these less-represented queues. The physical resources accessed via *QL* and *QE* partially overlap with those accessed through the other queues: out of the 192 nodes in the cluster, *QE*, *QP*, *QL* and *QF* have respectively access to 189, 187, 20 and 189 computational nodes. Thus several nodes can be accessed by more than one queue.

Given that, we built the following three datasets to evaluate our methodology:

- **D-Q***: this dataset contains the samples coming from all the monitored queues, thus allowing us to make predictions also for samples belonging to *QE* and *QL* queues, without the need of collecting further data for these specific queues.
- **D-QF**: this dataset only contains samples referring to jobs submitted to the *QF* queue.
- **D-QP**: as for the **D-QF** dataset, but with samples referring to the *QP* queue.

Among the possible targets, we focused our attention on the T_{wq} attribute, since it may strongly influence any strategy aimed at improving the execution of workflows on a batch scheduler-based system. A first observation that can be done is that T_{wq} is highly variant both in time (see Fig. 2(a)) and among jobs belonging to the same queue (see Fig. 2(b)), thus justifying the prediction purpose. Secondly, the collected data presents highly variable distributions over time, and this variability is reflected in the relationships between the target labels and the attributes used to make the predictions. As such, any dataset we intend to use for building the prediction model needs to be appropriately split. To point out how much this can be important, we split the three datasets according to the following strategies: (*i*) **random split** – the test set is sampled randomly and in the case of the **D-Q*** dataset the samples are stratified over the queues, accounting

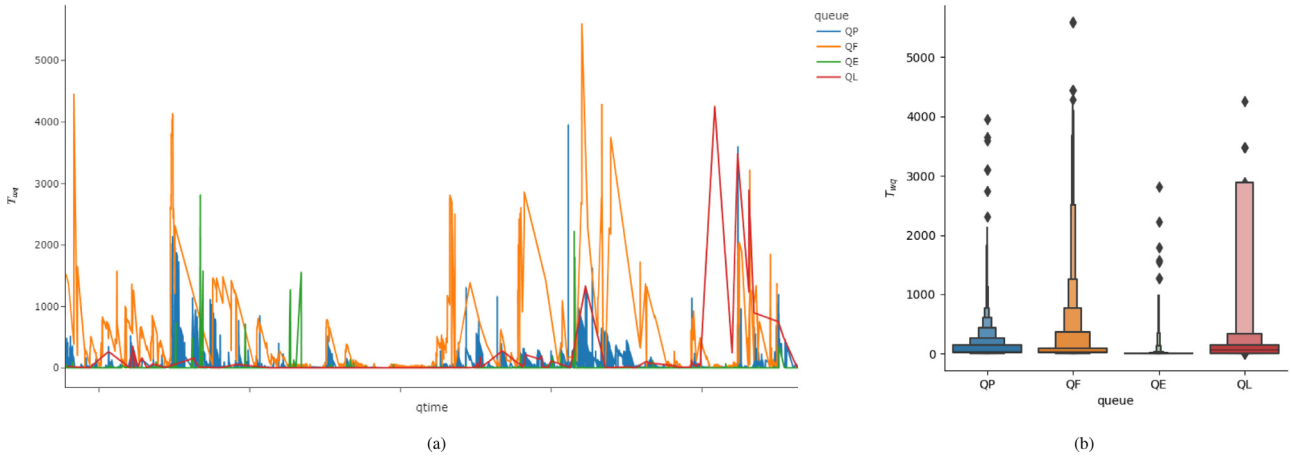


Fig. 2. Queuing time T_{wq} distribution plot for the jobs in the **D-Q*** dataset: on the left (a), there are the queuing times time-series collected over a 34-day time window for each queue; on the right (b), there is the boxplot of the queuing time for each queue.

for the 10% of the total samples; (*ii*) **last-day split** – the training set is made of all the data that correspond to jobs that ended their executions before the last day in the dataset, while the test set consists of the remaining jobs, only considering the ones that entered in the queue system after the last job in the training set has finished its execution; and (*iii*) **window-based split** – the training and the sets are iteratively formed using the temporal split described before (**last-day split**), but the time windows are of 7 days for the training set and 1 day for the test set. In this latter strategy, multiple training and test sets are defined and evaluated as the time-windows slide along with the whole 34-day dataset.

3.2. Data preprocessing phase

Each constructed dataset requires some preprocessing steps before being used by a learning model. This preprocessing aims at reducing the dimensionality of the features and improving the quality of the results. First, a *scaling* step is performed, the sample mean and the sample variance are computed on the training sets, and then they are used to normalise both the training and the test sets. After that, a *dimensionality reduction* step takes place. The following two approaches are combined to aggregate correlated features and select those that are relevant for the prediction purpose.

PCA-based Features Aggregation (PFA) The first approach we considered in the *dimensionality reduction* step is to perform the Principal Component Analysis (PCA) [30] on the input dataset, that at this step is yet split into training and test sets. The outcome of PCA is uncorrelated features, *i.e.*, the principal components (PCs), that are a linear combination of the original scaled features. PCs represent the maximum variance directions along which data are projected [31]. This allows summarising effectively high-dimensional data when the considered PCs are limited to the first N components, with N significantly lower than the original dimension [32]. In our case, the number of principal components (PCs) is selected considering the relative increment of explained variance obtained by adding new components. Subsequently, the scaled features are clustered according to the principal components' loadings, which are weights associated with the original features in the linear combinations. The clustering technique we chose is hierarchical clustering with Ward linkage, with the number of clusters selected by thresholding the clusters' distances. As such, we aggregate samples' features that are highly correlated with one another. The features belonging to the same cluster form a single new feature through the mean function. According to a non-anticipative strategy, the PCs' computation and the following features' aggregation are based exclusively on the samples belonging to the training set. Once the proper features' aggregations are computed on the training set, then the same aggregations take place on the test set features.

Features selection (FS). Feature selection is an important and challenging task in statistical modelling. After the **PFA** step, the most relevant features for predicting the target variable T_{wq} are selected by applying the Lasso regression [33–35]. Lasso regression automatically selects useful features while discarding ones not related to the target variable or that introduce redundancy. The approach relies on a linear regression model enhanced by a penalty strategy: the fitting of the statistical model minimises both the residual sum of squares of linear regression and the absolute values of the regression coefficients, thus shrinking to 0 the ones that correspond to predictive features that do not relate to the target variable. In the Lasso regression context, the absolute values of the regression coefficients play the role of relevance scores; the higher they are, the stronger the relationship with the target variable. So, for this capability of selecting relevant features, the Lasso regression is chosen to perform the features selection process: the variables that have a non-zero regression coefficient are selected as predictive attributes for the subsequent SL step, whereas the other ones (*i.e.*, those whose regression coefficient is zero) are considered unrelated with respect to the T_{wq} target and discarded. Feature selection is fit on the samples belonging to the training sets, then the selected features are also extracted from the test set.

To exemplify the whole preprocessing procedure, in Fig. 3, we report the results obtained on the **D-Q*** dataset with the **last-day split**. In this case, the PFA step found 21 feature clusters, with the features belonging to a certain cluster correlated to each other. For instance, *Cluster 5* (see the plot on the left in Fig. 3) contains all the features that represent the resources requested for the jobs belonging to the *QE* queue. As such, it aggregates *QE_Q_ncpus*, *QE_Q_nodect*, *QE_Q_num_job*, and *QE_Q_walltime*. On the other hand, *Cluster 14* aggregates *QP_R_used_mem* and *QP_R_used_vmem* features that summarise the HPC cluster memory usage for the running jobs submitted to the *QP* queue. As this first step is totally unrelated to the predictive attribute T_{wq}, the **FS** step is applied to find the relationships between the aggregated

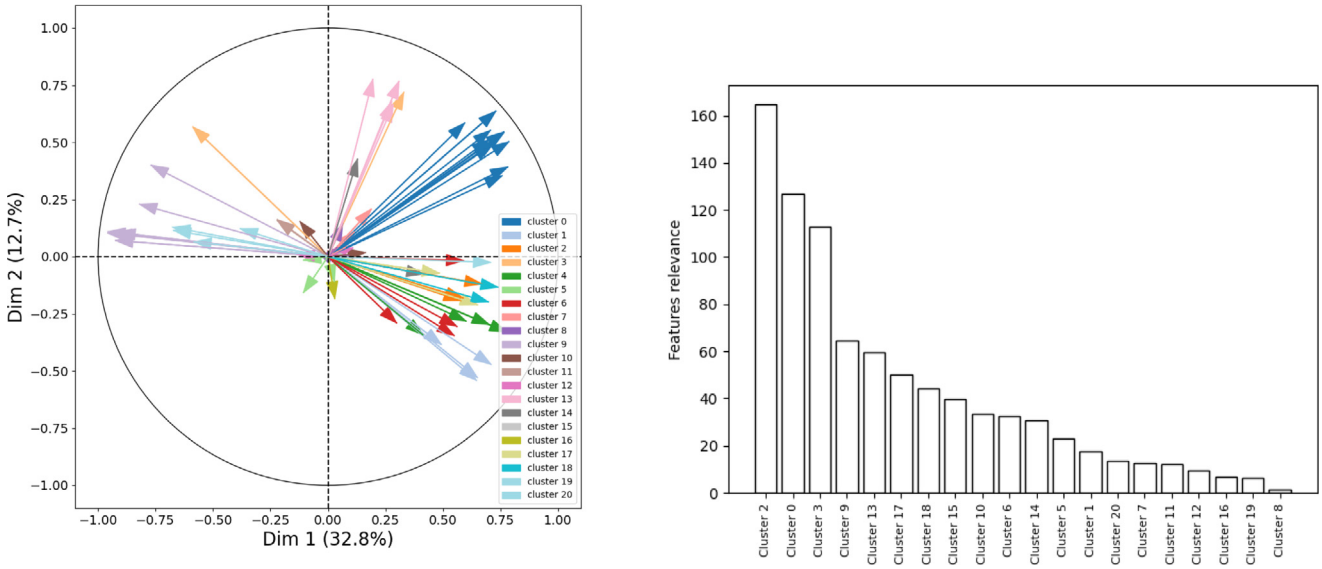


Fig. 3. PCA correlation circle (on the left) and features relevance (on the right), obtained from the preprocessing steps applied to the **D-Q*** dataset according to the **last-day split**.

features and the T_{wq}, thus also setting a common background with supervised learning techniques. Fig. 3, the right plot, reports the relevance score of the selected aggregated features. For instance, *Cluster 2* has the highest relevance score, it indeed includes the priority of the submitted jobs and the specific queue where they were submitted: these two features highly impact the waiting time (the higher the priority, the sooner the job execution begins) and have a high correlation (due to the considered HPC cluster policy, the computation of the priority scores takes into account the belonging queue). Still looking at the right plot in Fig. 3, we notice that the Lasso analysis excluded aggregated features represented by *Cluster 4*, which mixes up the features of running job submitted to the *QP* queue (*i.e.*, the *QP_R_nodect*, *QP_R_num_job*, *QP_R_used_cpupercent*, *QP_R_used_ncpus*, *QP_R_walltime*). Being willing to figure out the reasons for discarding *Cluster 4*, the reader can realise that they might be different. On one hand, there is the limit of the linear hypothesis that lies behind the Lasso approach; as such, the aggregated feature does not have a linear relation with the predictive attribute. On the other side, the aggregated features are discarded as the information concerning the running jobs submitted to the *QP* queue just marginally contributes to the prediction of the T_{wq} target.

3.3. Overview on the supervised learning models

As a matter of evaluating the capability of different supervised learning (SL) models on the task of predicting the T_{wq}, we investigated the application of models falling into two categories:

- *Classification* models highly depend on the binning of the queuing time (*i.e.*, the time a job spends waiting in the assigned queue); as such, they require to discretise the labels T_{wq}. Classifiers can be effectively exploited either for predicting the order of magnitude of the queuing time or when the classification bins can be defined to well-fit the data distribution.
- *Regression* models are the more natural choice for a continuous attribute. They can treat each job's submission as a sample, without requiring any particular data manipulation, thus preventing the introduction of approximation errors due to the discretisation process. As such, they are able to provide very precise predictions, thus allowing us to target near-optimal scheduling of jobs (when their performances are good enough), especially in those cases there is a logical dependence among them (*i.e.*, workflow's jobs).

3.3.1. The classification approach

The T_{wq} classification task requires a further data manipulation step. We investigated four different approaches to discretise the time, where the number of time-steps used in the discretisation process corresponds to the number of classes (N_{cls}) used by the classification models.

- Uniform time binning (*U-bin*): T_{wq} is discretised using a constant time-step for each class.
- Log-uniform time binning (*LU-bin*): first, we transform the T_{wq} into $T^*_{wq} = log_2(T_{wq})$; then, this transformed log-times (*i.e.*, T^*_{wq}) are binned with constant time-step. Finally, the classes are obtained by inverse transforming the boundary of the bins. In this case, the classes reflect mostly the order of magnitude of the queuing time.
- Log-KMeans time binning (*LK-bin*): the labels are transformed through the $log_2(\cdot)$ function as in the *LU-bin* case; however, here the bins are determined by applying a K-Means algorithm targeting a number of clusters equal to N_{cls}.
- Balanced time binning (*B-bin*): the boundaries of the bins for the T_{wq} classes are computed in such a way that the training samples represent balanced classes, *i.e.*, with nearly the same number of samples for each class.

It is worth noticing that the bins are computed using the data referring to the training set, while the classes are then obtained for the whole dataset. Table 4 summarises, for each of the above-described binning approaches, the number of samples belonging to each class, as N_{cls} changes. The tables also report, for each class and each binning approach, the mean queuing time (expressed in minutes), which has been computed with respect to the time boundaries of the classes. The interpretation of a job belonging to class C_i, $i \in \{1, \ldots, N_{cls}\}$ is that its queuing time is in the time interval $[T^i_a, T^i_b]$, where T^i_a and T^i_b ($T^i_a < T^i_b$) are the boundaries used to discretise the time domain. The reader can observe that

Table 4
Queuing times discretisation for classification models: for each class, we show the corresponding mean and standard deviation (std) of the queuing times, along with the number of samples. The mean times are reported in minutes. These tables refer to the case of classes build over all samples in **D-Q***.

	U-bin			*LU-bin*			*B-bin*			*LK-bin*		
Class	Samples	Mean	Std	Samples	Mean	Std	Samples	Mean	Std	Samples	Mean	Std
$N_{cls} = 4$												
C_1	37,187	698	197	6,672	5	3	9,424	7	4	6,992	4	3
C_2	440	2,096	450	19,453	42	16	9,511	24	6	17,439	32	13
C_3	217	3,493	412	9,519	361	142	9,444	70	20	8,694	170	66
C_4	15	4,891	438	2,215	3,119	984	9,480	2,847	690	4,734	2,938	837
$N_{cls} = 6$												
C_1	36,781	466	164	3,347	3	1	6,307	4	3	2,434	1	1
C_2	550	1,398	227	8,708	11	4	6,294	13	3	9,187	9	4
C_3	296	2,329	330	14,070	46	14	6,334	27	5	13,344	38	11
C_4	95	3,261	385	7,229	195	66	6,323	48	7	6,569	134	42
C_5	135	4,193	167	3,787	821	244	6,283	136	40	5,208	535	192
C_6	2	5,124	1	718	3,459	936	6,318	2,900	776	1,117	3,225	1025
$N_{cls} = 8$												
C_1	35,932	349	128	2,525	2	1	4,722	3	2	2,081	0	1
C_2	1,255	1,048	205	4,147	6	2	4,702	9	2	5,323	5	2
C_3	250	1,746	221	9,020	17	5	4,743	18	3	8,275	17	5
C_4	190	2,445	192	10,433	50	12	4,768	28	4	9,406	42	9
C_5	41	3,144	206	5,660	147	42	4,769	43	5	4,980	104	25
C_6	176	3,843	185	3,859	433	117	4,677	78	16	4,041	261	68
C_7	13	4,542	73	1,711	1,274	310	4,736	195	54	2,756	736	209
C_8	2	5,240	1	504	3,746	786	4,742	2,938	837	997	3,344	966

the *U-bin* approach produces the most unbalanced classes, with the unbalancing increasing with the number of classes. In this situation, a classifier that assigns to each sample the most representative class in the training set could easily reach a $0.94 - 0.98$ accuracy. This observation led us to evaluate the performance of the classification models considering the F1-score metric, which is less affected by the unbalancing of the classes than the accuracy score, and so it better represents the ability of the models to predict the correct classification. We also used a time-based error evaluation metric, called *Mean Time Absolute Error* (MTAE), that averages over the *Time Absolute Error* (TAE) samples defined as follows:

$$TAE = \begin{cases} 0 & C_i = \hat{C}_i \\ min(|\hat{T}_a^i - t_q|, |\hat{T}_b^i - t_q|) & C_i \neq \hat{C}_i \end{cases} \quad (1)$$

where $\hat{C}_i$, with time-boundaries $[\hat{T}_a^i, \hat{T}_b^i]$, is the predicted class, C_i is the true class, and t_q is the actual queuing time of the evaluated job.

Looking at some preliminary classification results, we realised that the *U-bin* approach generated a time discretisation able to drive the classification algorithms to the best results in terms of classification accuracy. However, the high unbalancing among the classes makes these results unreliable and suggests that this binning is more tailored to an outliers detection task than to a classification one. Indeed, the classification models were able to reach good performance basically by assigning all the test set jobs to the first two or three more representative classes. The *B-bin* approach puts jobs that are too different from each other within the same class: this strong balancing policy reduces the information about queuing times distribution, thus leading to the worst classification results. *LU-bin* seemed to provide reliable results (in terms of class balancing), without worsening too much the prediction performance. Moreover, considering the PCA outcomes of the preprocessing step, the projection on the first 3 PCs (see Fig. 4) suggests that the logarithmic scale of T_{wq}^* combined with a proper feature selection step could lead to interesting results. At last, the further improvement on the logarithmic binning provided by the *LK-bin* approach is our choice to evaluate the classification models, as it better reflects the distribution of the labels.

As previously mentioned, we evaluated different classification models in order to determine the best combination of UL and SL algorithms for predicting the target T_{wq} label. In the following, these algorithms are briefly presented.

Naïve Bayes classifier (NB). This classification model is based on the Naïve Bayes algorithms[4] (*i.e.*, the Multinoulli version for categorical variables and the Gaussian version for the continuous ones) and relies on the assumption that the features are conditionally independent of the classes' distribution. This assumption may affect the performance of the classifier. Anyway, despite this assumption being tough to fully satisfy, this model has previously been used for performing HPC queuing time predictions [24] with good results. Moreover, it can be easily adapted to perform real-time predictions without needing a (complex) continuous retraining process.

K-Nearest Neighbours classifier (KNN). This classifier is controlled by the positive integer hyper-parameter K. For each test sample x_i, the KNN classifier identifies the nearest K points in the training data that are closest to x_i and then it estimates the conditional probability as the fraction of points in the neighbourhood of x_i belonging to each class. Finally, KNN applies the Bayes rule to determine the class with the largest probability and classify the test observation.

Neural Network classifier (NN). This model does not require specific hypotheses on the input features and usually, it well-performs in approximating non-linear data distributions, hence it is broadly used in the ML context. As a drawback, standard NN-based models lack reliability metrics about their predictions, which are of crucial importance if critical decisions (*e.g.*, how workflows' jobs should be scheduled) are made based on the obtained predictions. Moreover, NNs are intrinsically dependent on hyper-parameters (*e.g.*, the number of hidden layers and the number of nodes in each layer) that are difficult to relate to the models' performance. Finally, continuous retraining of these models over time, according to data distribution changes to avoid their obsolescence, can be computationally expensive. From this

[4] https://pypi.org/project/mixed-naive-bayes

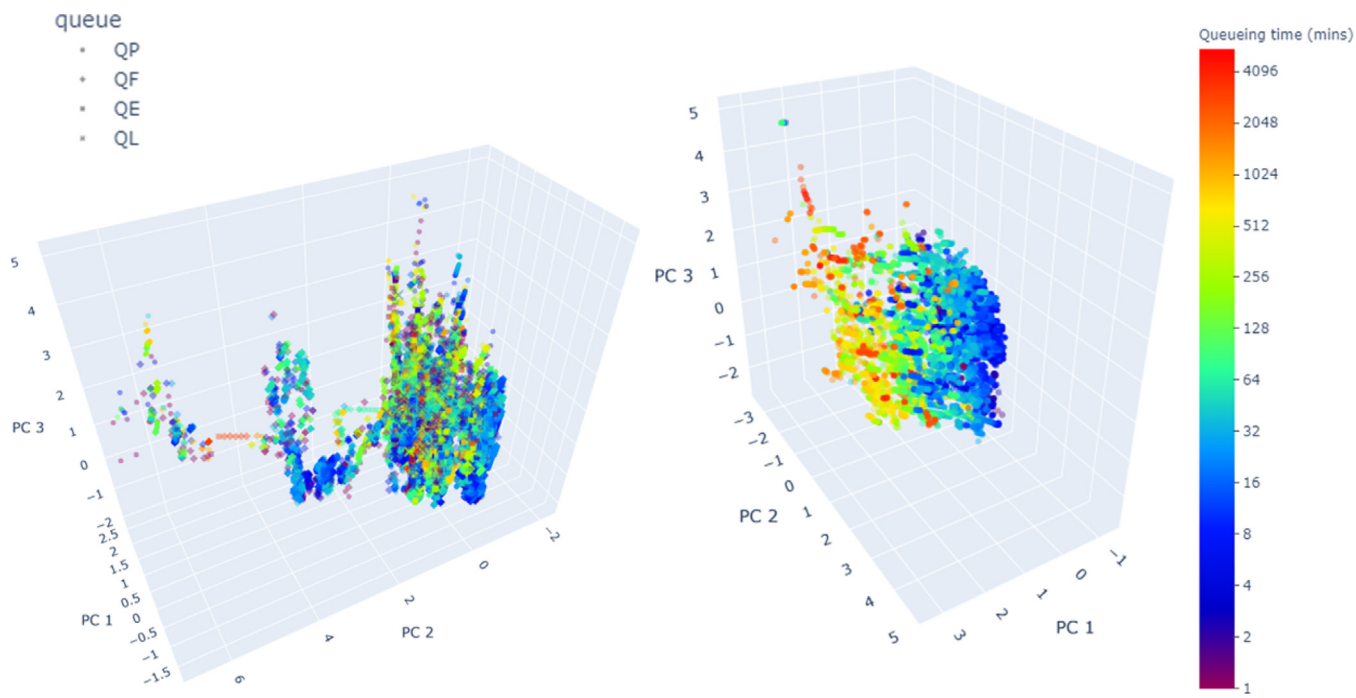


Fig. 4. Scatter-plot of the jobs' queuing time distribution, projected along with the first 3 principal components (PCs). On the right, the plot is related only to the **D-QF** dataset, on the left the one concerning **D-Q***. Each scatter-point is a job and the colour represents the job's queuing time (T_{wq}) in log_2-scale. Especially in the **D-QF** case, we can observe a relation between the queuing time distribution and the principal components: increasing values of *PC 1* and *PC 3* and decreasing the values of *PC 2* correspond to higher T_{wq}.

point of view, even not optimal hyper-parameters setting allows, in most cases, to achieve better results than other prediction models. Concerning the drawback of continuous retraining, *continual learning* could come in handy, allowing the trained model to learn new data features without forgetting the previous-learned ones [36]. As such, we let this possible development as an open door for future works. Looking at the chosen NN-classifier for our experimental evaluation, we opted for a multi-layer model, composed of 4 hidden layers consisting respectively of 256, 128, 64 and 32 neurons. The number of input neurons was set equal to the number of features generated by the preprocessing step, while the output layer, consisting of N_{cls} neurons, undergoes a soft-max activation function to obtain the classification probabilities. The loss function is a Cross-Entropy loss function with L2-regularisation applied to the weights and dropout regularisation (*i.e.*, dropout probability was set to 0.2 to better prevent over-fitting). The weights were updated by exploiting the Adam optimiser with the learning rate found using a learning rate finder function, relying on the cyclical learning rates technique [37]. Finally, the batch size has been fixed to 256, and the number of training epochs was set to 100.

Bayesian Neural Network classifier (BNN). This NN-based classifier has the same architecture and hyper-parameters as the previously described NN classifier. Unlike this latter, the dropout layers are activated also in the inference phase, thus providing a non-deterministic classification (hence, the name Bayesian Neural Network — BNN classifier). This model, with the dropout applied before every weight, the L2-regularisation and the Cross-Entropy loss function, is shown to be mathematically equivalent to an approximation of the probabilistic deep Gaussian process [38,39]. Therefore, it is possible to compute uncertainties metrics to evaluate the reliability of the model's predictions, without implementing a more complex variational NN model with twice the number of parameters. The final classification leverages Monte Carlo (MC) simulations; as such, we applied 200 MC iterations and we got as well 200 predictions for each sample to evaluate. Then, the final predicted class is the most representative of the predictions.

3.3.2. The regression approach

With the regression models, we target to predict T_{wq} with the granularity of minutes, thus, avoiding further manipulation of the overall features. The performance of the models is evaluated on the test set samples using the Mean Absolute Error (MAE) metric (expressed in minutes). As for the classification approach cases, in the following, we briefly describe the various models we evaluated.

K-Nearest Neighbours regression model (KNN). This model is the regression counterpart of the KNN classifier introduced in Section 3.3.1. The final prediction of test samples is obtained by averaging their neighbourhood computed on the training set.

Multi-Layer Perceptron (MLP). Similarly to the KNN case, this model architecture reflects the one described in the classification subsection. The only modifications concern the loss function and the output layer. In the former case, we exploited the mean squared error loss function, in the latter case the output consists of just one neuron with a ReLU activation function.

Bayesian Multi-Layer Perceptron (BMLP). This is the Bayesian version of the MLP, with the dropout layers also activated in the inference phase. This model allows for obtaining uncertainty scores, whose correlation with errors can be investigated. It is the regression counterpart of the BNN classifier (see Section 3.3.1); hence it shares the same architecture and the same training and inference hyper-parameters (*i.e.*, dropout probability, training epochs, batch size and MC iterations).

3.4. Uncertainty quantification

When using BNN models, we can rely on a standard approach for uncertainty quantification. In particular, we estimate the total predictive uncertainty $Var(Y|X)$ by exploiting the predictive distribution [40]. According to the law of total variance, $Var(Y|X)$ can be split into 2 components as follows:

$$Var(Y|X) = Var(\mathbb{E}[Y|X, \Theta]) + \mathbb{E}[Var(Y|X, \Theta)] \quad (2)$$

where Y represents the target variable distribution, X is the input features distribution and Θ models the random parameters within the Bayesian NN-based model. The two components in Eq. (2) represent respectively, the *epistemic uncertainty* $Var(\mathbb{E}[Y|X, \Theta])$, and the *aleatoric uncertainty* $\mathbb{E}[Var(Y|X, \Theta)]$. The former refers to the uncertainty of the model, and it can be decreased by introducing more training data, for instance, to better represent some classes in a classification task. Epistemic uncertainty can also be affected by the model architecture. Conversely, the latter concerns data's inherent randomness, which can be either constant along with data (homoscedastic) or a function of the data itself (heteroscedastic). Aleatoric uncertainty cannot be reduced, but it can be learned from data in its heteroscedastic component. We will estimate the total variance through Monte Carlo sampling both in the case of regression and classification models.

$$\begin{aligned} Var(Y|X = x_i) &= Var(\mathbb{E}[Y|\Theta; X = x_i]) + \mathbb{E}[Var(Y|\Theta; X = x_i)] \\ &\approx \frac{1}{T}\sum_{t=1}^{T} f(x_i, \theta_t)^2 - (\frac{1}{T}\sum_{t=1}^{T} f(x_i, \theta_t))^2 + \frac{1}{T}\sum_{t=1}^{T} S^2(x_i, \theta_t) \end{aligned} \quad (3)$$

To be more precise, Eq. (3) describes how to approximate the uncertainty for a given data sample x_i. Here, $f(\cdot, \cdot)$ is a forward evaluation of the Bayesian NN-based model, θ_t represents the set of parameters sampled by the Monte Carlo (MC) simulations exploiting variational dropout, T is the number of the MC iterations, and $S^2(\cdot, \cdot)$ is the predictive variance concerning the Y variable distribution. For instance, in the case of the classification model, where we have a Bernoulli distribution for each one of the possible classes, with $p_{c,i,t}$ representing the probability of assigning sample i to class c with a single forward evaluation t, the predictive variance is given by Eq. (4). Here, $\hat{c}$ is the predicted class.

$$S^2(x_i, \theta_t) = p_{\hat{c},i,t}(1 - p_{\hat{c},i,t}) \quad (4)$$

4. Results

In the following, we analyse the results obtained from a study on the performance of the various UL and SL approaches over the 34-day data we collected on a real-world production cluster and the KIT FHII dataset.

4.1. Classification results

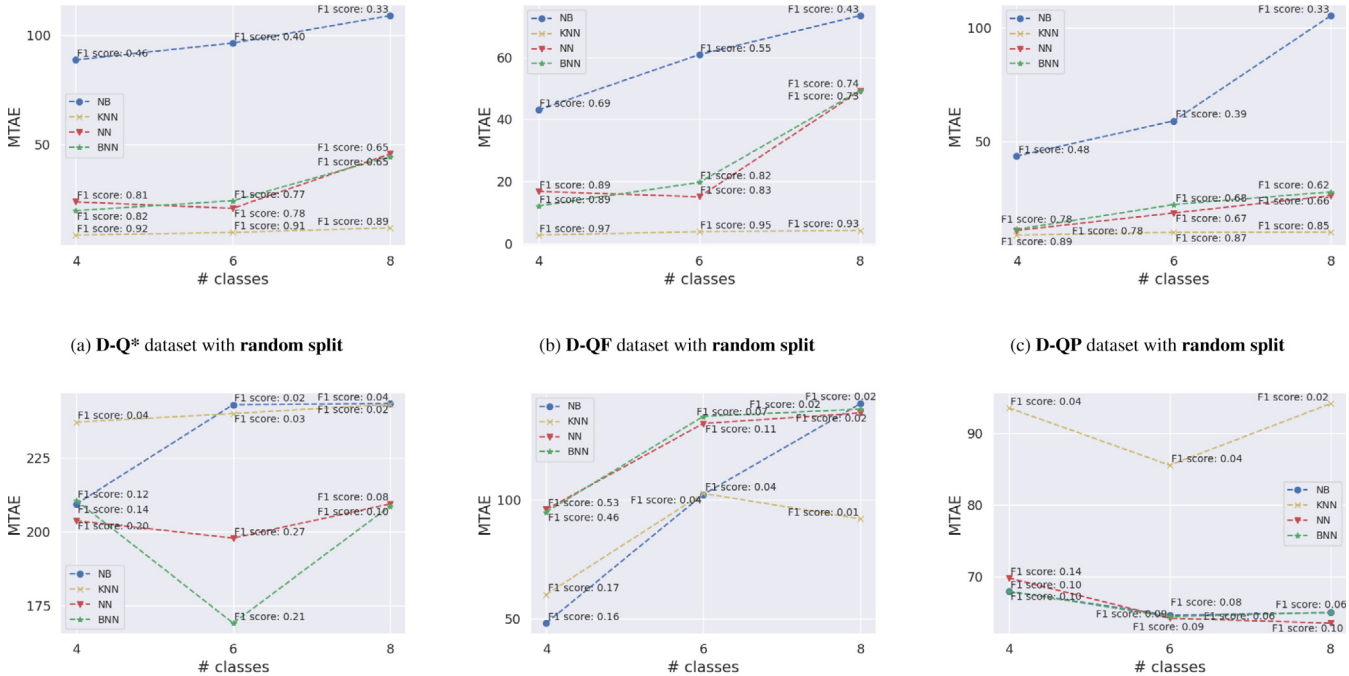
In this section, we compare the results obtained with the classification models applied to the **D-Q***, **D-QF** and **D-QP** datasets. Firstly, we want to highlight the relevance of the dataset splitting in training and test sets. In this context, job submissions that are very close in time to each other may refer to a status of the queues that have a negligible change. In this case, performing a random split in both training and test sets allows for obtaining artificially good predictions. This fact is confirmed by comparing Fig. 5 plots. KNN and NN-based models achieved good F1-score and MTAE performance when the test set is sampled randomly, whereas the performance considerably decreased for the predictions using a **last-day split**. Generally, the NB classifier seems to perform poorly, irrespective of the dataset split strategy used. Although poor performance is predictable when the **last-day split** is applied (as for the other classifiers, the temporal split introduces high variability between training and test set distributions), the low F1-scores obtained by the NB classifier, even in the case of the **random split**, might be a symptom that the model is not suitable for the proposed prediction task. A motivation for this can be found in the strong hypotheses of conditional independence that are the basis of the NB classifier and that are not consistent with the actual distributions of the predictive features and the target variable.

When moving to compare the performance of queue-specific (we took into consideration models for **D-QF** and **D-QP** datasets) classifiers with that of models treating all the queues at once, specialised models trained on *QF* queue data performed better. However, interestingly to notice, this well-performing behaviour is not reflected in the dedicated models trained on the *QP* dataset. Therefore, it is difficult to establish whether queue-specific models should be preferred to a unique model which refers to all the batch scheduler queues.

Starting from these considerations, we investigated the relationship between classification performances and data distribution through uncertainty quantification. Thus, we applied a **window-based** split, and, to avoid partially represented days, we removed the first day (Day 0) and the last day (Day 33) from the overall dataset; then, we made a weekly-based split of the remaining days (that is why the results in Fig. 6 cover days from the 8th to the 32nd). The classification results presented here concern the **D-Q*** dataset using 4 classes to assign labels since a higher number of classes lowered the classification performance. Since the NN-based models appear like a more natural choice when the model has to deal with a temporal-aware split, we addressed the BNN classifier. This latter has also the advantage that, in addition to its deterministic counterpart, it provides proper uncertainties metrics. In particular, referring to Fig. 6, we can highlight a connection between uncertainties among the test set predictions and the F1-score on the same test set. The histograms associated with each day represent the number of samples belonging to certain ranges of uncertainty. Each daily histogram has 15 bins and a variance limit range going from 0 to 0.3 (the top-left corner of each of these plots corresponds to the origins). We observe higher total uncertainty among samples corresponds to lower F1-score (see, for instance, Day-8, Day-25, Day-26, and Day-27); on the contrary, days for which the models achieved good classification performance correspond to lower uncertainty. It is worth noticing that the greater contribution to the total uncertainty derives from the aleatoric component. In particular, the predictions present a relevant heteroscedastic aleatoric component that highlights the presence of data-dependent noise.

Finally, as a general observation concerning the performance of the classification models and datasets with a $log_2(\cdot)$ transformation applied, we remark that the models we considered in this study tend to be less accurate in the predictions as N_{cls} increases; as such, the F1-score decreases and the MTAE metric increases accordingly. We interpreted this general behaviour as the consequence of the following two main facts:

- Increasing the number of classes (N_{cls}) makes it more difficult to distinguish between jobs' belonging to classes that are less separated in the time dimension. This is even more prominent when we consider the first classes since they are the ones nearest in time (keep in mind the log_2 transformation).
- Another significant consequence of increasing the number of classes is that the number of samples belonging to each class is reduced. On average, the 8-classes classification relies on a



(a) **D-Q*** dataset with **random split**

(b) **D-QF** dataset with **random split**

(c) **D-QP** dataset with **random split**

(d) **D-Q*** dataset with **last-day split**

(e) **D-QF** dataset with **last-day split**

(f) **D-QP** dataset with **last-day split**

Fig. 5. Classification results at the varying of the number of classes (N_{cls}) and the classification models.

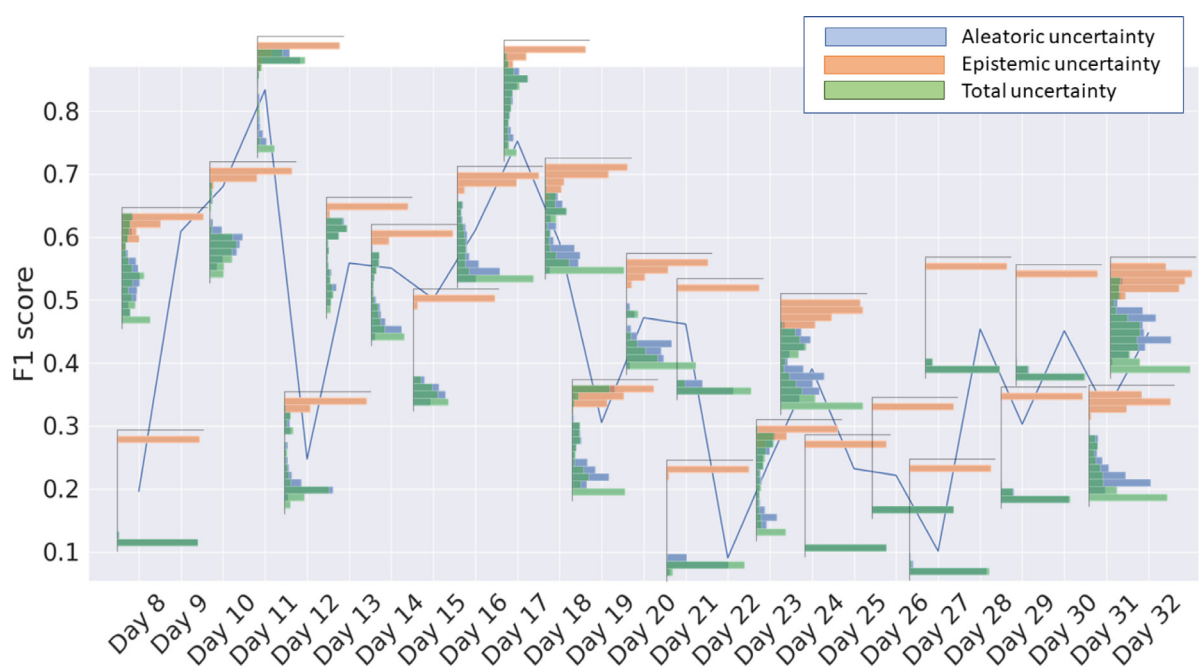


Fig. 6. Connection between classification performance (F1-score) and uncertainties of the predictions on the **D-Q*** dataset, with weekly training of the BNN classifier and N_{cls} set to 4.

number of samples per class that is halved compared to the 4-classes one. Moreover, as N_{cls} increases, the unbalancing between the classes increases as well. For instance, in the 4-classes classification, the less represented class has a number of samples equal to the 27% of the samples belonging to the most represented class; on the contrary, in the 8-classes classification, this ratio decreases to 10%.

As previously stated, all these considerations led us to investigate the regression models as a better alternative to their classification counterparts, since they are able to provide a prediction with

a minute-scale granularity. The following subsection details the experimental results we obtained using regression models.

4.2. Regression results

With the regression models, we evaluated the capability of supervised learning approaches to predict the target T_{wq} variable with a temporal scale of minutes. As in the previous subsection, we remark on the relevance of proper dataset splitting. To this end, the plots in Fig. 7 reflect the same splitting approaches already used in the classification task, *i.e.*, a **random split** on one hand (Figs. 7(a), 7(b), 7(c)), and a **last-day split** on the other hand (Figs. 7(d), 7(e), 7(f)). We considered the **D-Q*** dataset, the **D-QF** dataset and the **D-QP** dataset for our evaluations. All the reported plots represent the distribution of the Absolute Error (AE) expressed in minutes for the samples belonging to the test set, according to the used regression models.

First, the plots concerning the performance evaluation on the **D-Q*** dataset point out the variation of the errors among the queues. Specifically, in Fig. 7(a), we observe that all the regression models performed pretty well. In particular, the MLP regressor has lower outliers' errors on the *QF*, *QP* and *QE* queues than the other two models. Samples belonging to the *QL* queue are tough to predict correctly, as this is emphasised in Fig. 7(d). Indeed, in this case, the major contribution to the mean absolute error (MAE) on the test set is ascribed to the *QL* queue. This can be easily related to the very small number of samples belonging to this queue. Another worth-noticing behaviour concerns the (test) errors related to the samples belonging to the *QF* queue. Here, the small errors we see in the case of the **random split** become quite large when the **last-day split** is applied. This trend is maintained in the subsequent two plots (*i.e.*, Figs. 7(b) and 7(e)), where the samples in the test set exceed 250 min of AE and are definitely not negligible. On the contrary, the samples coming from *QP* present a more stable behaviour for the two different splits, so the outliers' errors are lower when the **last-day split** is applied (see Figs. 7(c) and 7(f)).

To better study the variability of the AEs, we also applied the window-based split to the **D-Q*** dataset, being careful of removing only the *QL* samples. In this case, the chosen regression model was the BMLP, as it provides variance scores and has performed quite equivalently to the other models. Looking at Fig. 8, for each subplot, we reported day-by-day the AEs on the y-axis and the variance on the x-axis. Moreover, we added information concerning the maximum AEs, to give a reference scale for the plots. These daily 2D histograms distinguish the contributions of each queue: we can notice that, across the days, the greatest errors are due to the queuing time prediction of the jobs belonging to the *QF* queue. Another interesting observation concerns the relationship between the variance values and the AEs. Indeed, discarding the predictions concerning the *QF* queue's samples (in this case, it seems that these samples bring a lot of erroneous and noisy predictions), we can see that usually, greater variance values correspond to greater errors on the test samples belonging to the *QE* and the *QP* queues.

4.3. Comparison with the KIT FHII dataset

This other dataset is obtained from workload files covering a period of approximately 1 year and 8 months on the KIT FHII System. The collected data refers to 2 HPC queues, respectively named *Q1* and *Q2*. Some basic statistics concerning the queuing times are summarised in Table 5. To provide a fair comparison with previous results, we elaborated the workload dataset to match the attributes available in our dataset as much as possible. Table 6 shows all the features we succeeded in obtaining for the new dataset. From a comparison with Table 2, the new dataset provides much less information on the cluster status each time a new job is submitted, *e.g.*, there is much less information on the usage of the resources for running jobs. Moreover, there is no reference to any priority score driving the scheduler allocation policy.

Table 5
The mean (measured in minutes) and standard deviation (std) for the T_{wq} target attribute along with the queues in the KIT FHII dataset.

Queue name	Mean [min.]	Std	Samples
Q1	149	2897	112006
Q2	6	74	2349
all queues	146	2868	114355

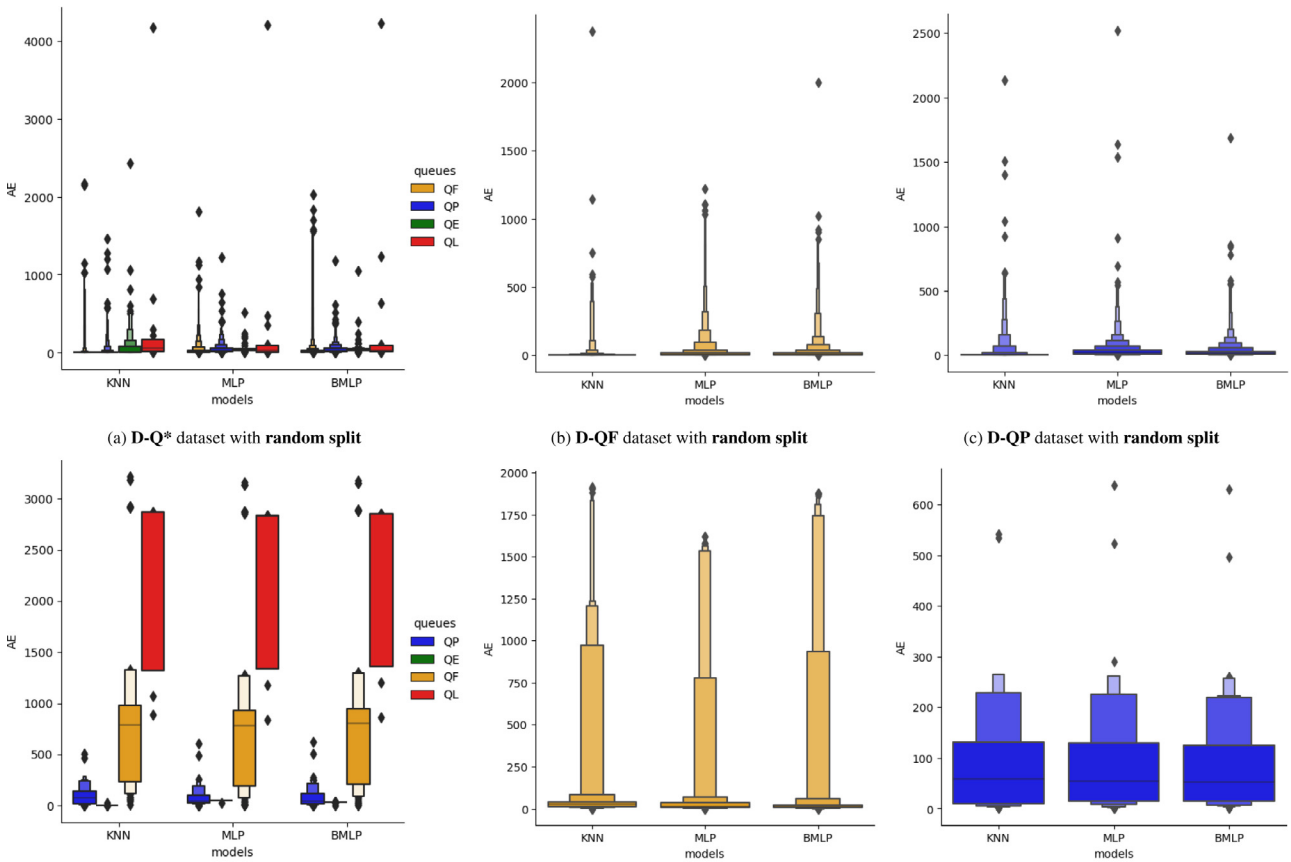
Table 6
Attributes available in the KIT FHII dataset. Whenever Qattributes and Rattributes refer to a specific queue, this latter is selected by the *qname* belonging to one of the accessible queues, *i.e.*, *Q1*, *Q2*.

Attribute	Description	
qtime	Timestamp in which the job enters the queue	
queue	The queue to which the job is submitted	
walltime	The maximum running time requested by the user for the job	
proc	The number of processors requested for the job	
Rattributes:		
	num_job	The number of jobs running
	proc_used	The number of processors used by the jobs running
	walltime	The total execution time requested for the jobs running
Qattributes:		
	num_job	The number of jobs running
	proc	The total number of processors requested by jobs preceding the considered job
	walltime	The total walltime, in minutes, requested by jobs preceding the considered job

As for the previously considered dataset, the prediction methodology consists of an automated preprocessing phase, followed by a classification or regression step. The results are reported in Tables 8 and 9, respectively. The importance of the temporal effect is again studied by applying both the **random split** and **last-day split** to obtain training and test sets. It is worth noticing that even if, given a dataset-splitting approach, the selected features are the same (see Table 7), the outcomes of the classifications and regressions lead to very different interpretations.

Concerning classification, the **last-day split** seems to produce a better performance. However, the high F1-scores of NN and BNN classifiers are obtained by classifying all the test set samples to the first class, to which actually the 90% of the test set samples belong. KNN and NB classifiers perform slightly better by correctly classifying the samples belonging to the last class. Nevertheless, these results are not sufficiently reliable to enhance an effective scheduling strategy. Interestingly, the NB, BNN and NN models' predictions do not improve when applying **random split**, NN and BNN still assign each test sample to the first class to which 77% of the test samples belong, NB classifier assigns most of the samples to either the first or the last class. On the other hand, KNN improves its classification performance. Thus, it is the best classification model for the **random split**.

When it comes to regression, the high MAEs suggest that none of the regressors is suited for this prediction task. Moreover, all models worsen their performance on **random split**: we observe that, even if the preprocessing step tries to maintain as many features as possible, they are not sufficient to reconstruct the HPC



(a) **D-Q*** dataset with **random split**

(b) **D-QF** dataset with **random split**

(c) **D-QP** dataset with **random split**

(d) **D-Q*** dataset with **last-day split**

(e) **D-QF** dataset with **last-day split**

(f) **D-QP** dataset with **last-day split**

Fig. 7. Regression results in terms of absolute errors (AE) in minutes at the varying of the regression models.

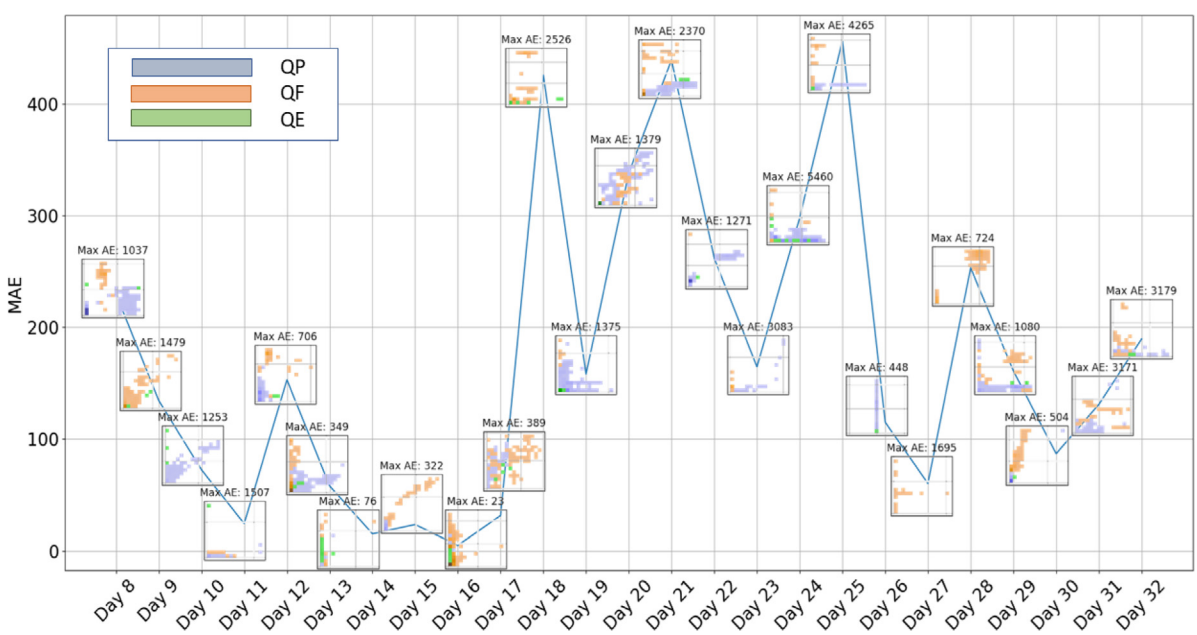


Fig. 8. Connection between the regression performance (MAE) and uncertainties of the predictions on the **D-Q*** dataset, except *QL* samples, with weekly training of the BMLP regressor.

Table 7
Aggregated features (***PFA***) selected by the ***FS*** step on the KIT FHII dataset. Both the **random split** and the **last-day split** are considered for the creation of the training (S_{train}) and test (S_{test}) sets.

random split: $\|S_{train}\| = 102919$, $\|S_{test}\| = 11436$	
cluster 1	*cluster_Q_num_job, Q1_Q_num_job*
cluster 2	*walltime*
cluster 3	*proc*
cluster 4	*cluster_R_proc_used, Q1_R_proc_used*
cluster 5	*cluster_Q_walltime, Q1_Q_walltime*
cluster 6	*queue*
cluster 7	*Q2_Q_proc, Q2_Q_walltime*
cluster 8	*Q2_R_proc_used*
cluster 9	*cluster_R_num_job, cluster_R_walltime, Q1_R_num_job, Q1_R_walltime*
last-day split: $\|S_{train}\| = 112216$, $\|S_{test}\| = 439$	
cluster 1	*cluster_Q_num_job, Q1_Q_num_job*
cluster 2	*walltime*
cluster 3	*proc*

Table 8
Classification results ($N_{cls} = 4$) on the KIT FHII dataset: both **last-day split** (Tmp) and **random split** (Rnd) are considered.

	NB		KNN		NN		BNN	
	Tmp	Rnd	Tmp	Rnd	Tmp	Rnd	Tmp	Rnd
MTAE	42.7	244.3	48.9	22.9	45.7	119.8	45.7	119.8
F1-score	0.85	0.48	0.84	0.91	0.85	0.67	0.85	0.67

Table 9
Regression results on the KIT FHII dataset: both **last-day split** (Tmp) and **random split** (Rnd) are considered.

	KNN		MLP		BMLP	
	Tmp	Rnd	Tmp	Rnd	Tmp	Rnd
Max AE	2841.6	12271.2	2801.2	8791.7	2804.0	8787.0
MAE	42.8	76.2	80.3	94.7	73.1	109.5

cluster status at the submission time of each job, this combined with a large time-period, which implies higher variability compared to the previous dataset, causes poor performance of the predictive models.

As mentioned, the KIT FHII dataset lacks some features compared to the **D-Q***, **D-QF** and **D-QP** datasets. Thus, it is interesting to analyse the impact on the predictions after maintaining only common features (since they have a similar meaning, we will consider *proc* ≡ *ncpus*).

Classification results are summarised in Table 10: maintaining a fixed set of features, performances still worsen when the **last-day split** is applied, only the NB classifier's results, from an F1-score point of view, are approximately the same, though they are not sufficiently accurate to provide meaningful predictions. When the dataset splitting approach is set, the removal of the non-common features has no significant effect in the case of **random split**; on the other hand, it reduces MTAEs and increases F1-score significantly in the case of a **last-day split**. So, this suggests that the removed features change their relationship with the prediction target over time. In particular, since their relation during the training days is no more valid for the test day, their removal enhances the prediction through more stable (concerning correlations along time) features' contribution.

Concerning regression results, see Table 11, the worsening of performances remains when the **last-day split** is applied, and the removal of the features not in common between the two datasets does not significantly impact the prediction. Nevertheless, since the prediction outcomes are still poor, this does not provide relevant insights for the choice of predictive features.

The results we reported here are evaluated on a dataset of interest in the context of the ACROSS and LEXIS projects and, to

Table 10
Classification results ($N_{cls} = 4$) on the **D-Q*** dataset: comparison starting with all the features (0-subscript) or considering only the features in common with the KIT FHII dataset (r-subscript). Both **last-day split** (Tmp) and **random split** (Rnd) are considered.

	NB		KNN		NN		BNN	
	Tmp	Rnd	Tmp	Rnd	Tmp	Rnd	Tmp	Rnd
$MTAE_0$	209.2	88.5	236.9	8.4	203.5	21.1	204.8	23.3
$MTAE_r$	152.4	82.7	179.6	9.4	190.7	21.5	178.6	20.2
$F1-score_0$	0.12	0.46	0.04	0.92	0.20	0.82	0.14	0.81
$F1-score_r$	0.44	0.48	0.29	0.92	0.28	0.81	0.26	0.80

Table 11
Regression results on the **D-Q*** dataset: comparison starting with all the features (0-subscript) or considering only the features in common with the KIT FHII dataset (r-subscript). Both **last-day split** (Tmp) and **random split** (Rnd) are considered.

	KNN		MLP		BMLP	
	Tmp	Rnd	Tmp	Rnd	Tmp	Rnd
MAE_0	242.2	20.7	224.6	39.9	213.9	41.2
MAE_r	218.1	24.1	218.4	42.0	209.6	43.9
$Max\ AE_0$	3214.9	4172.2	3170.4	4214.7	3170.7	4224.8
$Max\ AE_r$	3214.9	4205.2	3107.2	4230.5	3066.7	4220.4

provide some degree of generality, on another dataset available online. Further testing on other datasets could result in divergent conclusions: (i) the best prediction models may be different, and models' performance could vary a lot according to the considered data; (ii) data inherent noise may have a different impact on the predictions, generally lowering the noise allows for achieving better performance; (iii) different measured features can easily go through the UL step, as it is automatised, but may provide different correlation with the target variable, thus also impacting on the interpretation of the results.

5. Conclusion

This work presents a machine learning (ML) based methodology for predicting the time jobs, submitted to an HPC batch scheduler, have to wait before being executed. This specific prediction task still remains challenging, as only partially successful results have been achieved on the collected data. However, it is important to highlight that the lack of prediction performance on the test sets when applying a temporal split instead of a random split can be related to many aspects: (*i*) the data were collected at the turn of maintenance periods, which probably affected the data distribution; (*ii*) the random split introduces overfitting by assigning to the test set samples that are too near in time to the ones belonging to the train set; (*iii*) the correlation between the features and the target variable is not preserved over time or could not be properly addressed by the proposed models; (*iv*) there is an intrinsic noise in the data, which uncertainty analysis also highlighted; (*v*) the main contributors to this noise can be restricted to the samples belonging to one specific queue, in our case, it is the *QF* queue. This latter aspect is totally in line with the expected outcomes, concerning the considered HPC cluster configurations: the jobs submitted to the *QF* queue (*i.e.* the queue which refers to freely accessible resources) tend to fail more than those submitted to the production queue (*i.e.*, production jobs). Consequently, their running time is usually pretty different from the users' set walltime, thus highly impacting the predicted queuing time of the following jobs in the queue. The absence of such noise-affected queues, linked to a specific HPC cluster's queueing policy, could improve significantly the prediction performance.

By applying the entire methodology on the KIT FHII dataset, we extended the analysis outside the main scope of the study,

motivated by supporting scheduling strategies with ML predictions. In this way, we showed the generality of the approach and proposed further interpretations of the results, also following a comparison with the features in common between the two datasets.

Although there is still room for improvement, this work introduces some novelty over the State-of-the-Art machine learning techniques applied in the HPC field. The proposed methodology presents an uncertainty-based approach to evaluate predictions and the quality of data in the context of HPC queuing systems, as it paves the way for improving the prediction results through a proper selection of the predictions by relying on the uncertainty scores. The comparison of the different splits leads to other important considerations: the difference in the prediction performance among the split approaches highlights the importance of the temporal component and can be further investigated to find points in time that mark significant changes in the jobs' data distribution. Concerning the split approaches, even the results interpretation is different. In fact, predictions made when the **random split** is applied are not *predictions* in a strict sense, since they do not foretell future events. Instead, they are more related to a historical data analysis purpose and, in the limit of when the predicted labels are close to the real value, they could enhance a data augmentation task or the implementation of a data-driven queuing system simulator. Concerning the methodology per-se, the preprocessing step leads to aggregated features that are automatically generated and are not fixed a priori; thus, they are not constrained to the ones that can be obtained from the batch scheduler (*e.g.*, the PBS Professional as in our case). This allowed us to test our methodology on the KIT FHII dataset in a straightforward way. Moreover, the features are selected day-by-day in the case of the window-based split, so they are not even fixed in time, which comes in handy to adjust feature selections and aggregation towards more relevant features for the predictions. This allows modelling the relationships between features and labels more dynamically in time.

As a future activity, the methodology will be further investigated and improved. While on the preprocessing side, more complex features' extraction methods can be tested, such as a CNN-based approach presented in *LAXCAT* [41], on the prediction side, better modelling of the temporal component could be achieved through RNN models [42] or a continuous-time series approach. An example of this latter approach can be found in the *Prophet* model by Facebook [43], which can be combined with uncertainty quantification to individuate changing points in the data distribution.

CRediT authorship contribution statement

Chiara Vercellino: Methodology, Software, Formal analysis, Data Curation, Writing – original draft, Visualization, Conceptualization. **Alberto Scionti:** Writing – original draft, Writing – review & editing, Conceptualization. **Giuseppe Varavallo:** Formal analysis, Writing – review & editing. **Paolo Viviani:** Investigation, Resources, Conceptualization. **Giacomo Vitali:** Writing – review & editing, Conceptualization. **Olivier Terzo:** Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgements

This work has been supported by the LEXIS project funded by the EU's Horizon 2020 research and innovation programme (2014–2020) under grant agreement No 825532, and by the Euro-HPC–02-2019 ACROSS project, grant agreement No 955648. Data has been gathered thanks to The Ministry of Education, Youth and Sports of the Czech Republic from the Large Infrastructures for Research, Experimental Development, and Innovations project "e-INFRA CZ - LM2018140".

References

[1] H. Shoukourian, T. Wilde, D. Labrenz, A. Bode, Using machine learning for data center cooling infrastructure efficiency prediction, in: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, IEEE, 2017, pp. 954–963.

[2] Y. Ran, H. Hu, X. Zhou, Y. Wen, Deepee: Joint optimization of job scheduling and cooling control for data center energy efficiency using deep reinforcement learning, in: 2019 IEEE 39th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2019, pp. 645–655.

[3] K. Haghshenas, A. Pahlevan, M. Zapater, S. Mohammadi, D. Atienza, Magnetic: Multi-agent machine learning-based approach for energy efficient dynamic consolidation in data centers, IEEE Trans. Serv. Comput. (2019).

[4] S. Salman, C. Streiffer, H. Chen, T. Benson, A. Kadav, DeepConf: Automating data center network topologies management with machine learning, in: Proceedings of the 2018 Workshop on Network Meets AI & ML, 2018, pp. 8–14.

[5] Y. Etsion, D. Tsafrir, A short survey of commercial cluster batch schedulers, in: School of Computer Science and Engineering, Vol. 44221, The Hebrew University of Jerusalem, 2005, pp. 2005–2013.

[6] M.A. Salim, T.D. Uram, J.T. Childers, P. Balaprakash, V. Vishwanath, M.E. Papka, Balsam: Automated scheduling and execution of dynamic, data-intensive HPC workflows, 2019, arXiv preprint arXiv:1909.08704.

[7] D.H. Ahn, J. Garlick, M. Grondona, D. Lipari, B. Springmeyer, M. Schulz, Flux: A next-generation resource management framework for large HPC centers, in: 2014 43rd International Conference on Parallel Processing Workshops, IEEE, 2014, pp. 9–17.

[8] D. Zhang, D. Dai, Y. He, F.S. Bao, B. Xie, RLScheduler: An automated HPC batch job scheduler using reinforcement learning, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2020, pp. 1–15.

[9] I. Simaiakis, H. Balakrishnan, A queuing model of the airport departure process, Transp. Sci. 50 (1) (2016) 94–109.

[10] A. Roy, J. Pachuau, A. Saha, An overview of queuing delay and various delay based algorithms in networks, Computing (2021) 2361–2399.

[11] J. Behrmann, P. Vicol, K.-C. Wang, R. Grosse, J.-H. Jacobsen, Understanding and mitigating exploding inverses in invertible neural networks, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2021, pp. 1792–1800.

[12] M. Happ, M. Herlich, C. Maier, J.L. Du, P. Dorfinger, Graph-neural-network-based delay estimation for communication networks with heterogeneous scheduling policies, ITU J. Future Evol. Technol. 2 (4) (2021).

[13] H.M. Makrani, H. Sayadi, D. Motwani, H. Wang, S. Rafatirad, H. Homayoun, Energy-aware and machine learning-based resource provisioning of in-memory analytics on cloud, in: Proceedings of the ACM Symposium on Cloud Computing, 2018, pp. 517–517.

[14] T. Khan, W. Tian, G. Zhou, S. Ilager, M. Gong, R. Buyya, Machine learning (ML)–Centric resource management in cloud computing: A review and future directions, J. Netw. Comput. Appl. (2022) 103405.

[15] Y. Fan, ROME: A multi-resource job scheduling framework for exascale HPC systems, 2021, arXiv preprint arXiv:2108.13175.

[16] Y. Fan, Z. Lan, P. Rich, W.E. Allcock, M.E. Papka, B. Austin, D. Paul, Scheduling beyond CPUS for HPC, in: Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing, 2019, pp. 97–108.

[17] Y. Fan, Z. Lan, T. Childers, P. Rich, W. Allcock, M.E. Papka, Deep reinforcement agent for scheduling in HPC, 2021, arXiv preprint arXiv:2102.06243.

[18] M. Soysal, M. Berghoff, A. Streit, Analysis of job metadata for enhanced wall time prediction, in: Workshop on Job Scheduling Strategies for Parallel Processing, Springer, 2018, pp. 1–14.

[19] H. Wang, Y.-Q. Dai, J. Yu, Y. Dong, Predicting running time of aerodynamic jobs in HPC system by combining supervised and unsupervised learning method, Adv. Aerodynam. 3 (2021) 22, http://dx.doi.org/10.1186/s42774-021-00077-8.

[20] D. Tsafrir, Y. Etsion, D.G. Feitelson, Backfilling using system-generated predictions rather than user runtime estimates, IEEE Trans. Parallel Distrib. Syst. 18 (6) (2007) 789–803, http://dx.doi.org/10.1109/TPDS.2007.70606.
[21] M. Rezaei, A. Salnikov, Machine learning techniques to perform predictive analytics of task queues guided by slurm, in: 2018 Global Smart Industry Conference, GloSIC, IEEE, 2018, pp. 1–6.
[22] W. Smith, I. Foster, V. Taylor, Predicting application run times with historical information, J. Parallel Distrib. Comput. 64 (9) (2004) 1007–1016, http://dx.doi.org/10.1016/j.jpdc.2004.06.008, URL https://www.sciencedirect.com/science/article/pii/S0743731504000991.
[23] M. Tanash, B. Dunn, D. Andresen, W. Hsu, H. Yang, A. Okanlawon, Improving HPC system performance by predicting job resources via supervised machine learning, in: Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning), PEARC '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1–8, http://dx.doi.org/10.1145/3332186.3333041.
[24] V. Jancauskas, T. Piontek, P. Kopta, B. Bosak, Predicting queue wait time probabilities for multi-scale computing, Phil. Trans. R. Soc. A 377 (2142) (2019) 20180151, http://dx.doi.org/10.1098/rsta.2018.0151, URL https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2018.0151.
[25] J. Brevik, D. Nurmi, R. Wolski, Predicting bounds on queuing delay for batch-scheduled parallel machines, in: Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '06, Association for Computing Machinery, New York, NY, USA, 2006, pp. 110–118, http://dx.doi.org/10.1145/1122971.1122989.
[26] D. Nurmi, J. Brevik, R. Wolski, QBETS: Queue bounds estimation from time series, in: E. Frachtenberg, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 76–101.
[27] J.-W. Park, M.-W. Kwon, T. Hong, Queue congestion prediction for large-scale high performance computing systems using a hidden Markov model, J. Supercomput. (2022) 1–22.
[28] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, Adv. Neural Inf. Process. Syst. 28 (2015).
[29] L. Documentation, PBS attributes, 2021, https://linux.die.net/man/7/pbs_job_attributes. (Online: Accessed May 2021).
[30] G. James, D. Witten, T. Hastie, R. Tibshirani, An Introduction to Statistical Learning: with Applications in R, Springer, 2013, URL https://faculty.marshall.usc.edu/gareth-james/ISL/.
[31] B.M.S. Hasan, A.M. Abdulazeez, A review of principal component analysis algorithm for dimensionality reduction, J. Soft Comput. Data Min. 2.1 (2021) 20–30.
[32] A. Tharwat, Principal component analysis-a tutorial, Int. J. Appl. Pattern Recognit. 3.3 (2016) 197–240.
[33] V. Fonti, E. Belitser, Feature Selection Using Lasso, VU Amsterdam Research Paper in Business Analytics, Vol. 30, 2016, pp. 1–25.
[34] R. Jain, W. Xu, HDSI: High dimensional selection with interactions algorithm on feature selection and testing, PLoS One 16.2 (2021).
[35] J. Wang, et al., Feature selection using a neural network with group lasso regularization and controlled redundancy., in: IEEE Transactions on Neural Networks and Learning Systems, 2020, pp. 1110–1123.
[36] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A.A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, R. Hadsell, Overcoming catastrophic forgetting in neural networks, 2017, arXiv:1612.00796.
[37] L.N. Smith, Cyclical learning rates for training neural networks, 2017, arXiv:1506.01186.
[38] Y. Gal, Z. Ghahramani, Dropout as a Bayesian approximation: Representing model uncertainty in deep learning, in: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Vol. 48, ICML '16, JMLR.org, New York, NY, USA, 2016, pp. 1050–1059.
[39] A. Damianou, N.D. Lawrence, Deep Gaussian processes, in: C.M. Carvalho, P. Ravikumar (Eds.), Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, in: Proceedings of Machine Learning Research, vol. 31, PMLR, Scottsdale, Arizona, USA, 2013, pp. 207–215, URL http://proceedings.mlr.press/v31/damianou13a.html.
[40] J.D.J.Z. Jeremy Oldfather, Quantifying Uncertainty in Deep Learning Systems, AWS, 2020, URL https://d1.awsstatic.com/APG/quantifying-uncertainty-in-deep-learning-systems.pdf.
[41] T.-Y. Hsieh, S. Wang, Y. Sun, V. Honavar, Explainable multivariate time series classification: A deep neural network which learns to attend to important variables as well as informative time intervals, 2020, arXiv:2011.11631.
[42] E. Diaconescu, The use of NARX neural networks to predict chaotic time series, WSEAS Trans. Comput. Res. 3 (2008).
[43] S.J. Taylor, B. Letham, Forecasting at scale, PeerJ Prepr. 5 (2017) e3190.

Chiara Vercellino holds an M.Sc. degree in Mathematical Engineering (orientation: statistics and optimisation on networks and data), received from Politecnico di Torino. Currently, she is attending a Ph.D. in Computer and Control Engineering at Politecnico di Torino, and she is a researcher at the CPE (Advanced Computing, Photonics and Electromagnetics) area of LINKS Foundation, where her main focus is on optimisation algorithms, targeting, among the others, HPC/Cloud resources orchestration and Quantum optimisation for industrial applications. She participates in the ACROSS EuroHPC project and she is a member of the SRA-5 WG6 "Mathematics and Algorithms" at ETP4HPC.

Alberto Scionti holds a M.Sc. and a Ph.D. (European doctorate degree) in computer science and control engineering, both received from Politecnico di Torino, Italy. He is a senior researcher in the Advanced Computing, Photonics and Electromagnetics group at LINKS Foundation. His main research interests include management of heterogeneous HPC/Cloud infrastructures and quantum computing. He was involved in several national and EU-funded projects. He is co-author of more than 50 peer reviewed papers on international conferences and journals. He was editor of a book on heterogeneous computing architectures and is currently leading the orchestrator design in the EU-H2020 ACROSS project.

Giuseppe Varavallo Ph.D. Candidate in Innovation for the Circular Economy at the University of Turin. In 2017 he won a Research Grant in Data Science for Social Good at ISI Foundation, Turin. He was involved in a project to develop a composite indicator to measure Environmental Injustice in the World, using Sustainable Development Goals data. From 2019 to 2022, he worked as a researcher at LINKS Foundation (Turin), working on activities related to Data Mining, Web Development, and Distributed ledger technologies.

Paolo Viviani holds a M.Sc. in Theoretical Physics and a Ph.D. in Computer Science both achieved at University of Turin. He worked for more than 5 years as Research Engineer at Noesis Solutions NV on machine learning methods for engineering. He is now senior researcher at the Advanced Computing, Photonics and Electromagnetics area of LINKS Foundation, Turin, where he works on HPC applications, HPC and AI convergence, and Quantum computing. During his career, he participated in several EU and nationally funded research projects and published several papers at peer reviewed conferences.

Giacomo Vitali, currently Ph.D. student at Politecnico di Torino in Quantum Computing, he is graduated in Physics of Fundamental Interactions at the University of Pisa. After a year as a researcher at the Normale di Pisa where he worked for the LHCb experiment, he moved to LINKS foundation as a senior researcher in computer science, mainly dealing with HPC topics, like optimisation and Quantum Computing.

Olivier Terzo holds Ph.D. in Electronic Engineering and Communications and M.Sc. Degree in Computer Engineering from both achieved at Politecnico di Torino. He is currently the Head of the CPE Advanced Computing, Photonics and Electromagnetics Research Domain with a staff of 25 Researchers at the LINKS Foundation Applied Research Center. He was the Technical and Scientific coordinator of the OPERA H2020 project. He is the coordinator of the ACROSS EuroHPC project and the co-design and dissemination manager of the LEXIS project. He is a delegate member at ETP4HPC and an active member of the HiPEAC community. He peer-reviewed several journals.