Future Generation Computer Systems 175 (2026) 108064

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

An online algorithm for power consumption prediction of HPC workload

Francesco Antici [a,*], Andrea Borghesi [a], Zeynep Kiziltan [a], Jens Domke [b], Andrea Bartolini [a]

[a] University of Bologna Bologna, Italy
[b] Riken Center for Computational Science Kobe, Japan

ARTICLE INFO

Keywords:
High-performance-computing
Machine learning
Natural language processing
HPC Workload
Job-level prediction
Green computing

ABSTRACT

As modern High-Performance Computing (HPC) systems push the boundaries of computational capabilities, their power consumption becomes a serious threat to environmental and energy sustainability. In such a context, accurate prediction of the jobs' power consumption is instrumental to develop efficient power management strategies acting at the system level. To this end, in this paper, we present an *online* prediction algorithm to predict job power consumption in a production HPC system, prior to job execution. Our solution employs machine learning tools, and it is able to predict the *minimum*, *average* and *maximum* power consumption of a job, aggregated per node throughout its execution. Our approach leverages only information which is available at the time of job submission, and it is validated on two datasets extracted from production supercomputers, namely F-DATA from Supercomputer Fugaku and PM100 from Marconi100. Our experimental results show that our prediction algorithm outperforms state-of-the-art techniques, and it can accurately predict job power consumption, by obtaining an error of less than 12 % on F-DATA and less than 22 % on PM100.

1. Introduction

High-Performance Computing (HPC) systems have become fundamental to modern scientific research and industrial applications, enabling the execution of large-scale computations. However, their increasing power consumption poses significant challenges related to energy efficiency, cost, and environmental sustainability [1]. To guarantee a sustainable development of current and future HPC systems, it is thus fundamental to develop efficient power management strategies to improve the system's energy efficiency while guaranteeing optimal performance of the system.

To this end, one effective solution is to work at the system's workload level [2–4], which is composed of thousands of jobs executed on the system's resources every day. When the system resources perform operations (i.e. when jobs are executed on them), their power consumption grows (in comparison to their idle state when they are not performing operations), ultimately increasing the overall system energy consumption. Hence, accurate prediction of job power consumption, prior to their execution, allows us to devise energy-aware workload management strategies (job scheduling or resource allocation techniques) to improve the energy efficiency of the system [3,5,6].

Recent advancements in Machine Learning (ML) have enabled predictive modeling techniques that accurately estimate the power consumption of jobs executed on HPC systems [7–9]. To make power-aware decisions on the job scheduling, the prediction has to be performed before the job execution. To achieve so, the prediction algorithm can leverage only features available at job submission time, such as the *user name, job name* and *# of requested resources*; this information is typically available in the majority of modern workload dispatchers [10]. Past work [11,12] demonstrated that to obtain optimal prediction performance, the prediction algorithm should work in an *online* fashion, meaning that the predictive models are updated recurrently on data of more recent job executions to adapt to the changes in system workload [13].

In this work, we propose an end-to-end *online* predictive algorithm for the prediction of *minimum*, *average* and *maximum* job power consumption (aggregated per node throughout its execution), at the time of job submission. Our approach works on real production workloads, without filtering any jobs (unlike prior work such as [9]). We rely on a sliding-window *online* learning framework, with daily retraining of lightweight ML regression models. The learning framework is tuned across multiple settings, offering both accuracy and retraining efficiency, which are essential for deployment on real systems. This paper builds on our previous work [14], which we extend in several ways. First, we modify the methodology to predict also the *minimum* job power consumption, in addition to the *average* and *maximum*. Second, we study

* Corresponding author.
E-mail addresses: francesco.antici@unibo.it (F. Antici), andrea.borghesi3@unibo.it (A. Borghesi), zeynep.kiziltan@unibo.it (Z. Kiziltan), jens.domke@riken.jp (J. Domke), a.bartolini@unibo.it (A. Bartolini).

https://doi.org/10.1016/j.future.2025.108064
Received 5 April 2025; Received in revised form 19 July 2025; Accepted 30 July 2025
Available online 5 August 2025

extensively the *online* algorithm parameters, aiming to find the best setup to maximize the prediction performance. Third, we validate the predictive algorithm on two publicly available datasets extracted from two production HPC systems with different architectures, namely supercomputers Fugaku and Marconi100. The Fugaku dataset[1] (F-DATA) comprises more than 1 million jobs executed on Fugaku between December 2023 and May 2024, while the Marconi100 dataset[2] (PM100) contains the data of ∼ 230K jobs executed between May and October 2020. Finally, we perform a comprehensive multi-level evaluation (job-level, user-level, and system-level), showing robustness across prediction granularity and operational perspectives. We compare the performance obtained by employing different regression models and different ways to encode job-related information for prediction purposes. Our experimental results show that the *online* algorithm setting outperforms the classic *offline* approach on both datasets across all prediction tasks. The *online* algorithm indeed improves the prediction performance up to the 10 % and the R^2 score up to the 700 %. Therefore, this algorithm is essential for obtaining accurate job power consumption predictions and ultimately equipping the scientific community with a tool to develop efficient system power management strategies.

The rest of the paper is organized as follows. In Section 2, we introduce the datasets and the ML/NLP models used in our work, and review related work. Then, we describe in Section 3 the prediction algorithm and its pipeline. We present our experimental evaluation in Section 4. We discuss the possible limitations of our approach in Section 5 before we conclude in Section 6.

2. Background and related work

2.1. Datasets

2.1.1. HPC Systems

The datasets are extracted from two production HPC systems, namely Supercomputer Fugaku[3] and Marconi100.[4] Fugaku is hosted at the RIKEN Center for Computational Science in Japan. The system was developed by RIKEN and Fujitsu, and it can reach a peak performance of 537 PFlops/s through around 160k interconnected computational nodes, each of them endowed with 48 Arm cores and 32 GiB of high-bandwidth memory. Marconi100 (now dismantled) was hosted by CINECA, one of Europe's largest supercomputing centers. It was a pre-exascale supercomputer based on IBM POWER9 processors and NVIDIA Volta V100 GPUs, connected through a high-speed NVLink architecture, enabling efficient GPU-accelerated computing.

As can be observed in Table 1, the systems present significant structural differences. Fugaku is a CPU-only system, while Marconi100 is endowed with 4 GPUs per node. However, Fugaku has over a hundred times more nodes than Marconi100, enabling it to achieve a substantially higher peak performance (537 PFLOP/s versus 30 PFLOP/s), as reflected in the top500 ranking of the world's most powerful supercomputers.[5] Fugaku was ranked as $1st$ for around a year and a half after its deployment in production. Conversely, Marconi100 made the cut for only the top 9 supercomputers at the time of deployment in production. Nowadays, after around 5 years from its deployment, Fugaku is still in the top positions ($7th$ in June 2025), while the latest Marconi100 positioning (before being dismantled) was $26th$ in June 2023.

2.1.2. F-DATA

This is a comprehensive workload dataset[1] comprising approximately 24 million jobs executed on Fugaku over three years of public usage (March 2021 to April 2024). It was created through a specialized management software[6] installed on the system, which provides job management functionalities (e.g., job manager and scheduler) and facilitates the recording and storage of job execution details. For privacy concerns, sensitive data is provided both in anonymized and encoded formats, with the encoding leveraging an NLP model to preserve certain job information essential for predictive modeling purposes without compromising data privacy.

Each job entry includes a rich set of features such as *exit code, duration, power consumption*, and performance metrics (e.g., *# floating-point operations, memory bandwidth, operational intensity*, and *performance class*), facilitating diverse job characteristic predictions, including power consumption prediction. For each job execution, the minimum (*minpcon*), average (*avgpcon*) and maximum (*maxpcon*) power consumption values are available as single integer values, which are computed as the sum of the minimum, average and maximum power consumption of the nodes allocated to the job during its execution, respectively. We note that Fugaku does not allow node sharing, meaning that the power consumption values refer to a single job execution.

The dataset is organized in 38 monthly `.parquet` files. For the purposes of this work, we consider only the data stored in the last 5 files (i.e. `23_12.parquet`, `24_01.parquet`, `24_02.parquet`, `24_03.parquet` and `24_04.parquet`), referring to the 1.3 million jobs executed on Fugaku between December 2023 and April 2024. Fig. 1 illustrates the temporal distribution of job submissions in this portion of the dataset. The number of submitted jobs consistently exceeds 10k per day, with an average of 20k jobs submitted daily. The only exceptions occur in early February and April, when scheduled system maintenance led to temporary shutdowns.

Table 1
Fugaku and Marconi100 system architecture.

	Fugaku	Marconi100
Architecture	Armv8.2-A SVE	IBM POWER9
OS	RHEL	RHEL
Job scheduler	Proprietary	SLURM
#Nodes	158,976	980
#Cores (per node)	48 (+4 assistant cores)	32
#GPUs (per node)	0	4 NVIDIA V100
Memory (per node)	32 GB	256 GB
Peak performance	≈ 537 PFlop/s	≈ 30 PFlops/s
Peak top500 rank	$1st$ (06/2020-11/2021)	$9th$ (06/2020)

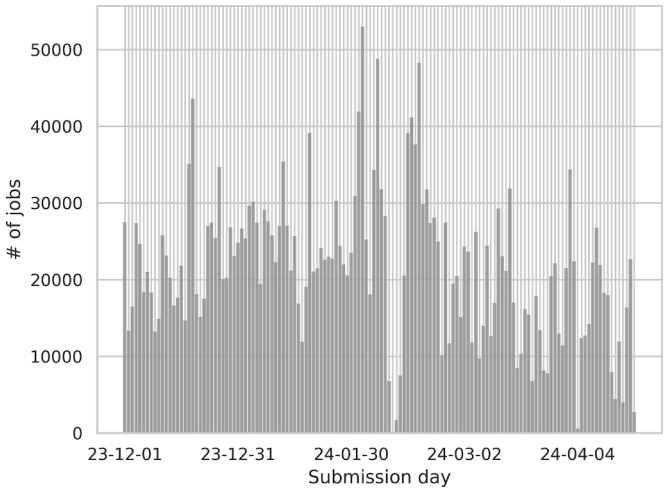

Fig. 1. Distribution of jobs submitted to Fugaku between Dec.'23 and May'24.

[1] https://zenodo.org/records/11467483
[2] https://zenodo.org/records/10127767
[3] https://www.fujitsu.com/global/about/innovation/fugaku/
[4] https://www.hpc.cineca.it/hardware/marconi100
[5] https://www.top500.org/lists/top500/2025/06/
[6] https://www.fujitsu.com/global/about/resources/publications/technicalreview/2020-03/article10.html#cap-03

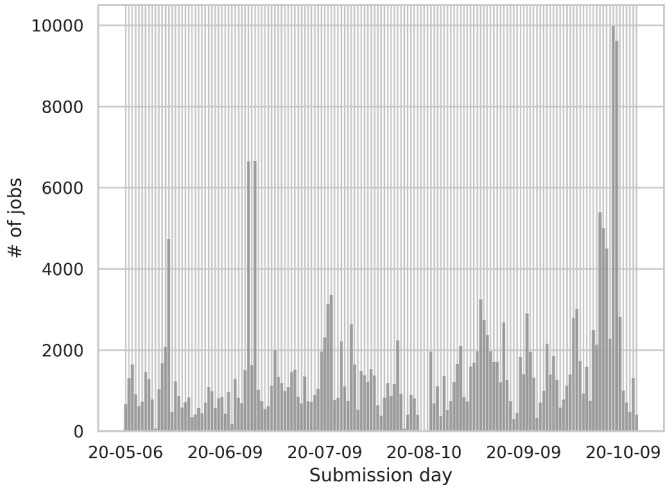

Fig. 2. Distribution of jobs submitted to Marconi100 during May-Oct'20.

2.1.3. PM100

This dataset[2] comprises a total of 231,116 jobs executed on Marconi100 between May and October 2020. It was created by aggregating and processing the raw data from the original M100 dataset [15]. The data collection on Marconi100 was achieved through ExaMon [16], a monitoring framework which collects several system usage metrics from different hardware sensors (e.g. node power consumption, room temperature and resource utilization) and software components (e.g. workload manager).

The dataset is stored as a single `.parquet` file, with each entry containing detailed information of a job execution, such as *exit code*, *duration* and *# requested resources*. A key feature of PM100 is the presence of different per-job *power consumption* values sampled every 20 seconds during the job execution. The *power consumption* of each job is recorded at the node, CPU, and memory levels.

Fig. 2 shows the distribution of the job submissions. Differently from F-DATA, the submission load of the system is significantly lower, with an average of less than 2000 jobs submitted per day. PM100 includes the data of the first months of production of Marconi100 (which entered in production in May). This explains both the low load of the first months, when the systems was mainly used for testing, and the high load of the last weeks, when the system was fully operational and external users started to use it.

2.2. ML and NLP models

2.2.1. ML classifiers

We exploit ML for prediction purposes, rather than more sophisticated Deep Learning (DL) models, as our approach is designed for online deployment in production HPC environments, where predictions must be made at job submission time, ideally within milliseconds. Traditional ML models offer extremely fast inference (as discussed in Section 4), which is crucial to avoid job scheduling latency. In contrast, DL models-especially those using high-dimensional embeddings or multilayer architectures-can introduce non-negligible runtime overhead, reducing the practicality of frequent updates. In preliminary experiments (not reported in the paper), we explored using fully connected feed-forward neural networks. We observed no performance gains over ensemble models like XGBoost, despite increased training time, tuning complexity, and inference cost. This aligns with observations reported in prior work (e.g., [7,9]) where traditional ML often outperforms or matches DL in tabular HPC workload data with structured features. Next, we briefly introduce the ML models used in our algorithm.

The Random Forest (RF) is a well-known ensemble ML algorithm leveraging several independent Decision Tree (DT) model instances for regression tasks. An RF model is trained on historical data to compute statistical correlations between input feature values and a given prediction target. Each DT is trained individually by tuning the internal parameters on a random subset of the training data and a random subset of the input features. At inference time, a majority voting among the trained DTs is carried out. This is done to make up for the tendency of individual DTs to overfit on the training data and thus obtain less error-prone prediction performance, as explained in [17]. In general, the RF benefits from having a voluminous amount of data, since the computation of statistical correlation becomes more accurate for the given sample.

XGBoost (XG) is a gradient boost algorithm designed for several prediction tasks, such as classification and regression. XG relies on an ensemble of predictive models (usually DTs), with each model attempting to correct the mistakes made by the other ones. Leveraging on gradient-based optimization, XG aims to optimize a specific loss function defined for the task. Moreover, XG includes regularization techniques to control model complexity and prevent overfitting. Due to these characteristics, XG excels in handling large-scale datasets, capturing complex interactions among features and delivering high predictive performance, as discussed in [18].

The k-Nearest Neighbors (KNN) is a widely used instance-based algorithm for regression tasks. Unlike typical ML-based methods, KNN does not require a training set to build a prediction model, as it does not rely on internal parameters. During model building, it stores a fixed amount of data points in a given feature space. Then, at inference time, it computes the k-nearest neighbours as the most similar elements among the stored ones, according to a distance metric, such as Minkowski. Because of this, the minimum amount of data needed to perform prediction is k.

2.2.2. Sentence bert

To feed the textual job information into the aforementioned ML classifiers, we need to convert them into a numerical format. For this purpose, we employ Sentence Bert (SBert), which is a state-of-the-art sentence embedding model, obtained by fine-tuning pre-trained BERT (Bidirectional Encoder Representations from Transformers) in sentence similarity tasks [19]. BERT is trained on millions of textual documents to understand language patterns. The model can generate word-level embeddings, namely a semantically meaningful floating-point array representation. However, when working with pieces of text or strings in regression tasks, this representation is impractical [20,21]. Conversely, SBert generates meaningful sentence-level embeddings, i.e. a 384-dimensional floating-point array.

2.3. Related work

Several past work tackled the job power consumption prediction task through the analysis of workload data, but they differ from our approach in various aspects. For instance, [22,23] explored the use of workload manager information to perform power-related prediction on job execution. However, their solutions rely on job data beyond what is available at submission time, making prediction before execution infeasible and thus unsuitable for our purposes. In [7,24], the authors predict job average power consumption per node by using an RF model trained on historical data. Both approaches train the model only once, without updating the model with the most recent data. Even if their solutions are suitable for retraining the models, they do not provide insights on retraining mechanism (e.g. temporal window of the retraining data and retraining frequency). Instead, our *online* prediction algorithm relies on an automated and continuous retraining process, which we evaluate extensively. As shown in [11,12] and validated by our experimental results in Section 4.3, the approach of [7,24] is less accurate than ours.

The algorithms presented in [8,9] predict average job power consumption per node and the models are updated periodically on more recent data. However, their approach differ from ours in many ways. The models are based on exponential smoothing of similar past jobs'

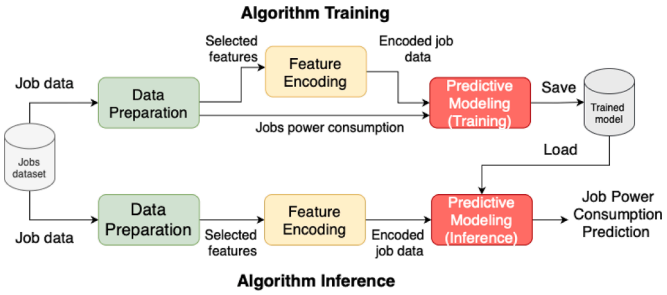

Fig. 3. High-level functioning of the prediction algorithm.

power consumption. Moreover, [8] relies only on categorical job features (*user id*, *group id*, *# tasks per node*) and this approach was shown to be outperformed by an RF model trained only once on all the available data (like that of [7]). While the algorithm of [9] is tested on Marconi100 data (which is different from PM100), it is evaluated only on jobs which ran for more than one minute, due to limitations in the proposed approach. In a real scenario, this would require knowledge on job duration before job execution, which is unrealistic. Additionally, this would mean removing more than half of the job executions on Marconi100, as discussed in [25]. A prediction algorithm that cannot act on any data would would provide incomplete and noisy information for job scheduling strategies or system power estimation. Differently, our approach does not require job duration information and works with any type of job.

Our work differs from all the studies cited above in additional respects. First, we predict job power consumption at different granularity, namely the *minimum*, *average* and *maximum*. All these values are useful for power-aware job scheduling and resource allocation [3,26,27]. Second, we validate our algorithm's robustness on two distinct large datasets, extracted from different production systems. Finally, [8,9] focus only on predicting power consumption at the job level, while we also consider the system level, which is a relevant information for power management. This was done only in [7] for the *average* system power consumption. As we show in Section 4.3, our approach is more accurate, as it obtains an error of 5 % against the 9 % obtained in [7].

3. Methodology

In this section, we describe our methodology for the power consumption prediction of the jobs submitted to HPC systems.

3.1. Algorithm overview

The goal is to build a prediction algorithm that estimates the power usage of a job by using only the information available at job *submission time*. To do so, we define a predictive pipeline composed of three algorithmic steps: *Data Preparation*, *Feature Encoding*, and *Predictive Modeling*, as depicted in Fig. 3. The figure shows how our algorithm employs the three steps in two different operational modes, namely *Algorithm Training* and *Algorithm Inference*. In the former, the algorithm leverages historical job execution data and the three algorithmic steps to generate an instance of a trained prediction model. In *Algorithm Inference*, the trained model is used to generate a prediction for a single unseen job using its submission time data. Again, the three algorithmic steps are employed sequentially to achieve this goal.

The only implementation requirement of our algorithm is the presence of a *Jobs dataset*, containing the data of past job executions. This dataset should contain features regarding job submission (such as *requested resources*, *user information*, *job name*), job execution (such as *duration*, *#nodes allocated*, and *power consumption*) and job completion time. The job execution and completion time features are available only upon job completion. This is not a strict requirement, since modern systems are typically endowed with monitoring software that permits the collection and dump[7] of job data [28–30], including power consumption [31,32].

In this paper, we focus on job power consumption per node. Nevertheless, our methodology can easily be configured to predict power consumption at different granularity (e.g. CPU, GPU or memory level), or other job execution characteristics (e.g. duration, failure or performance metrics). Moreover, the same methodology can be used to estimate job energy consumption per node, computed as the predicted power consumption multiplied by the job duration set by the user (e.g. wall-time). In the following, we describe each step of the pipeline in detail.

3.2. Data preparation

We perform a series of data engineering steps starting from the raw job data extracted from the *Jobs dataset* to isolate the information we need to address the prediction task. Specifically, we select a subset of job features that effectively represent job characteristics. Then, we define prediction target, which is required for the model training phase, and then create additional job features with the target values.

3.2.1. Feature selection

In order to perform the prediction before the job is executed, we can only rely on job *submission time* features [11,12]. Such features – such as *user name*, *job name*, and *environment variables* – are the information available when a job is submitted, hence that can be retrieved without further modification to the normal workload submission workflow. These features are instrumental in capturing job similarity, consisting of analogous computational patterns, hardware allocations, and akin power consumption.

In general, job power consumption is influenced by the computational operations performed and the amount of hardware utilized. Consequently, jobs executing similar operations and leveraging similar hardware configurations are likely to exhibit comparable power consumption patterns. Moreover, as explained in [11,12], in HPC production systems, users tend to submit jobs in batches containing similar experiments. Jobs submitted in the same batch are prone to have similar names, characteristics and perform similar operations. Given that the power consumption of a job depends on the computational operations it performs, jobs performing the same or similar operations will have similar power consumption. Therefore, features like the *user name, job name* and *environment variables*, might be the key to identify similar jobs, and consequently, perform accurate job power consumption prediction.

This feature set can be decided upon empirical evaluation on historical job data. Once settled, every time a job's data is inputted in the prediction pipeline, such features are extracted and inferred in the prediction model.

3.2.2. Job power consumption

In this work, we focus on the *minimum, average* and *maximum* job power consumption. In general, the job power consumption values can range from few to millions of Watts, depending on the amount of resources allocated to the job execution. This makes the prediction task very hard and the possible relative prediction error very high. In the light of that, we decide to perform some data pre-processing to make the target more suitable for the regression task, as outlined hereafter. As done also in [7], we normalize the power consumption of a job ($pcon_j$) on the # of nodes allocated to the job execution, as shown in Eqs. (1)–(3). This normalization allows us to predict power consumption as if each job were running on a single node, aiming to reduce the potential for prediction errors. Moreover, this allows to account for the fact that the power consumption of the same jobs varies significantly by

[7] Export of the database's data and structure at a given moment in time, typically stored in a file (e.g. csv, parquet, json, etc).

changing the number of nodes allocated. By removing this factor, we can treat the power consumption as an intrinsic property of the job, which is more dependent on the job's operations rather than on the number of nodes allocated to it. This strategy has been previously employed in related studies [7,33] and has proven effective, particularly in workload scheduling applications [5,34].

$$minpcon_j = \frac{min(pcon_j)}{\#nodes\_allocated_j} \quad (1)$$

$$avgpcon_j = \frac{avg(pcon_j)}{\#nodes\_allocated_j} \quad (2)$$

$$maxpcon_j = \frac{max(pcon_j)}{\#nodes\_allocated_j} \quad (3)$$

The output of our prediction tasks, for a given job, will thus be the *minpcon*, *avpcon* or *maxpcon*. When working with historical data, these features must be created for all the past job execution, so as to use them as the ground truth for the training and evaluation of the prediction model.

3.3. Feature encoding

The regression models presented in Section 2.2 require a numerical representation of the job feature values which are heterogeneous in their type (e.g. integer, string, float). We suggest to encode them in a standard numerical representation by applying two strategies. In the INT encoding, an integer is assigned to the values which are not numerical, i.e. *user name*, *job name*, and *job environment*, while setting all the missing values in the other fields to a default value of -1. This encoding is a standard in the field of ML, and it is particularly suited for regression tasks [35].

In the SB encoding, all the feature values are first concatenated into a comma-separated string, e.g. *user1, job1, … , 1, 1, env1*. Then the string is encoded with SBert, obtaining a 384-dimensional floating-point array. SBert extracts more fine-grained insights about job features expressed in natural language (e.g. *user and job name*), in the context of job-level classification, as demonstrated in [11,12], This is because SBert is designed to represent sequences, with semantically similar contents, with similar encodings. As we discussed in Section 3.2, jobs with similar names and users could belong to the same submission batch running similar operations, therefore, such features could reveal important patterns on the nature of the job and its workload. Moreover, SBert allows to address privacy concerns on user data. Indeed, user data are usually considered sensitive and their disclosure or third-party usage is generally undesired. To this end, SBert projects the information on a latent space, thus complicating the association between the original user-sensitive information and the encoded data used for training [36,37].

3.4. Predictive modeling

For predictive modeling, we leverage ML models, which exploit historical data, originating from the *Jobs dataset*, to learn patterns and relationships in past job executions (training) to ultimately make accurate predictions on new, unseen job data (inference). To this end, our algorithm employs an instance of an ML regression model, since the prediction target (power consumption) is a numerical value.

3.4.1. Training

For model training, we use the encoded job data and the power consumption values of historical job traces, and define two temporal learning settings. In the *offline* setting, the model is trained only once using the entire historical data, without taking into account the temporal job execution information (e.g. the *submission time*, *start time* or *end time*), nor temporal subsets of the data. This setting is a standard in ML-based predictive modeling for its simplicity.

In the *online* setting, we exploit the temporal nature of the historical data, as a consequence of constant job submission, execution and completion in a system. We thus treat the job data as live and streaming in time, and retrain the model periodically using a sliding window of most recent jobs. This approach is expected to capture evolving workload patterns for a better prediction and is more suitable for adoption in production HPC systems [11]. The model is initially trained on the data of the jobs executed in the last α days. After that, the model is retrained periodically every β days, using only the data of the jobs executed in the last α days. As we proceed in time, the model is continuously re-trained with different training data, which are more recent in time.

As discussed in Section 3.2, the workload of an HPC system is usually submitted in batches having similar characteristics. Our *online* algorithm is expected to exploit this similarity. Moreover, it allows for the models to dynamically adapt to the change in the workloads that arise in production HPC systems due to the arrival of new users and the start of new research projects or campaigns [38].

3.4.2. Inference

The model inference relies on a trained instance of the ML model to generate a prediction for a new, unseen job. The input of the trained model is the job data which is first extracted from the *Jobs dataset* and then processed by the *Data Preparation* and *Feature Encoding* steps. The output is the power consumption prediction for the given job, which depending on the prediction task can be either the *minimum, average* or *maximum* job power consumption.

The frequency of the inference depends on the use case. It can be called periodically (e.g. after a certain number of jobs are submitted or after a given time span) or when needed, without any periodicity.

4. Experimental study

In this section, we present our experimental design and results. In the experiments, F-DATA and PM100 act as *Jobs dataset* and we will evaluate the prediction accuracy of our prediction algorithm. The same set of experiments is performed on each dataset alone, without merging together the two data sources. We do that because we want to test our algorithm performance on the data of two different systems; investigating transfer learning strategies is outside the interests of this work. Moreover, as explained in Section 2.1, the two datasets exhibit inherently different characteristics (especially in terms of job power consumption), hence making the evaluation of transfer learning strategies non-trivial.

We run the experiments on a machine with two AMD EPYC 7302 CPUs with 64 cores and 512 GB of RAM. The RF and KNN algorithms are implemented with the *scikit-learn*[8] Python library, while the XG implementation is retrieved from the *xgboost*[9] library. The sequence encoder model is provided by the *sentence transformers* library,[10] while the weights for SBert are pulled from huggingface.[11] We use the pre-trained model *all-MiniLM-L6-v2*,[12] since it is the best trade-off between prediction performance and speed [39]. All the models are instantiated with the default setting provided by the libraries. The implementation and the details of the Python version and its packages are available in a GitHub repository.[13]

4.1. Data pre-processing for the prediction tasks

To test our prediction algorithm on F-DATA and PM100, we need to pre-process the raw data, following our methodology described in Section 3.2. We note that this process does not remove any data from the original datasets. This means that failed, short (less than 1 minute of

[8] https://scikit-learn.org/stable/
[9] https://xgboost.readthedocs.io/en/stable/index.html
[10] https://www.sbert.net
[11] https://huggingface.co
[12] https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2
[13] https://github.com/francescoantici/online-jpcp

duration), and multi-node jobs are all included in the evaluation, without exceptions. In a real production system, it is not possible to filter out specific jobs before their execution, as all job related information can be obtained only after job termination. Moreover, we do not distinguish between PM100 jobs that run on CPUs only and those that also use GPUs. We intentionally adopt this approach to ensure our methodology reflects a realistic production environment, where all types of jobs coexist. Restricting predictions to jobs with specific characteristics would undermine the robustness of our prediction algorithm.

4.1.1. Job feature selection

In an initial phase, we evaluated models' prediction performance using different subsets and combinations of the job features obtainable at submission time. We performed the evaluation on each dataset separately, in order to find their best feature set. Given that we use two different feature encoding techniques, and different ML models, we cannot rely on statistical metrics (e.g. feature importance provided by RF or feature correlation) alone. To extract the optimal feature set, we evaluated the prediction performance of different algorithmic combinations on a subset of the datasets. Then, we extracted the smallest feature set which achieved the best prediction performance overall.

In F-DATA, the subset of features which yields the best predictive performance is composed of *user name, job name, # cores requested, # nodes requested, node frequency requested* and *environment*. Conversely, in PM100 data, the optimal feature set turns out to be composed of *user name, job name, partition, # cores requested, # GPUs requested, quality of service and amount of memory requested*. The two feature sets confirm our intuition about which information is most important for predicting job power consumption. Indeed, the feature sets have some common elements, such as the *user name* and *job name*. The information provided by the *environment* and *node frequency requested* features of F-DATA can be correlated to that provided by the *quality of service* and *partition* of PM100, as all of these features represent a machine setup for the job execution. Similarly, both feature sets present information on the hardware allocated, i.e. *# cores requested* in both, *# nodes requested* in F-DATA, and *# GPUs requested* and *amount of memory requested* in PM100. Clearly, the architectural differences of the systems are reflected to the relevant features. Marconi100 mounts GPUs (which are very power hungry components) and allows for node sharing, thus *# nodes requested* is less relevant per se w.r.t. *# GPUs requested* and *amount of memory requested*. Instead, Fugaku does not have GPUs and does not support node sharing, meaning that the *# nodes requested* is very informative about the resource requirements of a job.

4.1.2. Job power consumption

As explained in Section 2.1, the minimum, average and maximum power consumption recorded throughout the entire execution of a job j are available in F-DATA features. Conversely, in PM100 these three features are not present, and need to be derived. Each job j in PM100 has a $node\_power\_consumption_j$ feature, which is a time-series containing the power consumption of the job, sampled every 20 seconds between its start and end time. Each element is obtained as the sum of the power consumption of all the nodes allocated to the job. Hence, the $minpcon_j$, $avgpcon_j$ and $maxpcon_j$ features are generated for each job j by taking the *minimum, average* and *maximum* of $node\_power\_consumption_j$. As explained in Section 3.2, we normalize the power consumption values on the # of nodes allocated to the job execution, thus generating *minpcon*, *avgpcon* and *maxpcon* for each job data of the two datasets.

Fig. 4 reports the distribution of these values in F-DATA and PM100. In F-DATA, the majority of the values fall in the 40W-110W interval, with a maximum value located around 40W. For values greater than 140W, the *avgpcon* is shifted to the left of *maxpcon*, while the *minpcon* are almost not present. This is explainable by the fact that for a single job, the *maxpcon* is always greater or equal to the *avgpcon*, and the same holds for *avgpcon* and *minpcon*. Thus, jobs consuming high amounts of power can obtain higher peaks of power consumption (maximum),

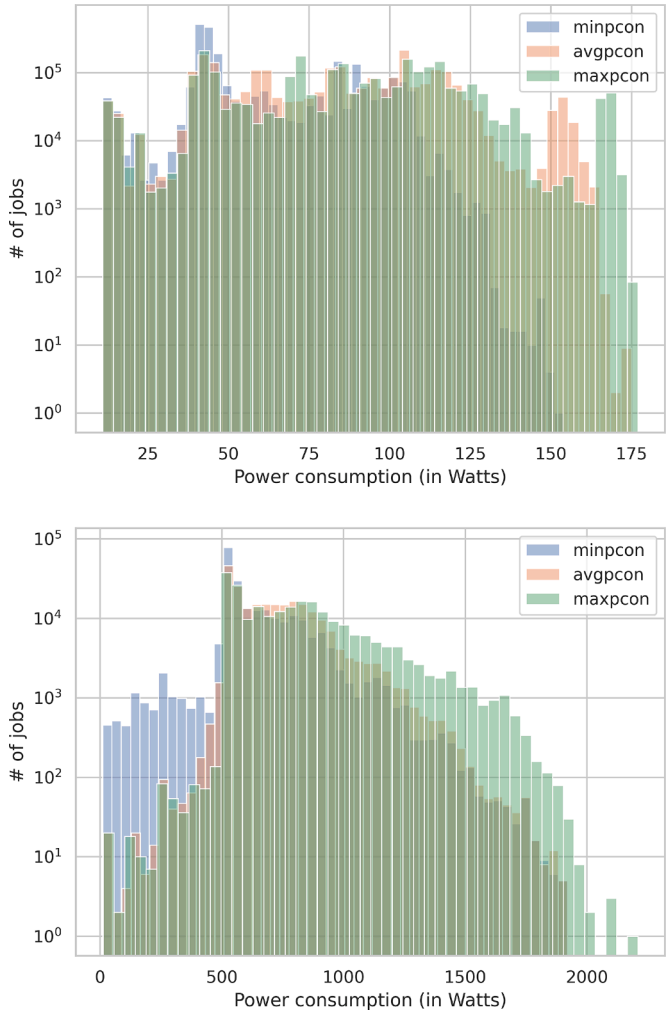

Fig. 4. Distribution of *minpcon*, *avgpcon* and *maxpcon* values of the jobs in F-DATA (above) and PM100 (below).

while keeping a lower mean power consumption throughout their execution. Concerning PM100, the job power consumption values span from a few watts to more than 2000 W. This is due to the presence of power hungry components like GPUs in the nodes, which makes the power consumption reach significant peaks. Here, job power consumption is more variable than in F-DATA, as witnessed by the fact that *minpcon* values are generally lower than 500 W, while *avgpcon* and *maxpcon* are mainly greater than 600 W. As in F-DATA, we observe the shift towards greater power consumption values of *maxpcon* w.r.t. *avgpcon* and *minpcon*.

We note that our analysis focuses on jobs executed on dedicated nodes without sharing them, as our datasets include only this type of jobs. Fugaku does not support node sharing. While Marconi100 supported it, the PM100 dataset filtered out such jobs. We retain that this does not pose any limitation to our approach, as no current method can determine jobs' node power consumption when jobs run concurrently on the same node [25]. On the contrary, this allows to have reliable per-job power consumption values, without the need of any modification to the recorded values.

4.2. Prediction algorithm evaluation

4.2.1. Testing on historical data

To test our prediction algorithm, we need to define the training and test sets and extract them from F-DATA and PM100. When working with

historical data, it is not realistic to do inference on a job by learning from the data of the future jobs submitted at a later time [12]. This is because in a real system, job data can be collected correctly only after the job execution and when the data collection process finishes successfully. Thus, whenever a job is submitted, we can use only the data of the jobs that finished their execution. To ensure that the training data always comes chronologically before the test data, we order the data in both datasets chronologically based on their *submission time*.

In the *offline* setting, since the model is trained only once, we split the job data into two. We take the first 70 % of the data as the training set and the remaining 30 % as the test set. As for the *online* setting, given that the model is retrained periodically, we create different training and test sets. We consider as the first training set all the jobs that are submitted in the first α days (and possibly finished after the αth day). Starting from the *submission time* of the first job not present in the first training set, we divide the data in batches in chronological order, where each batch contains the jobs submitted in the next β days. We then iterate over each batch, considering it as a new test set. At every iteration, the training set is updated with the data of the last α days and the models are retrained. To guarantee soundness of the setting, we ensure that all the jobs in the training splits end before the submission of the jobs in the test sets by using the *end_time* feature of the jobs.

4.2.2. Metrics and baselines

To evaluate the prediction quality, we compute the Mean Absolute Percentage Error (MAPE) and the R^2 score between the actual value (ground truth) and the predicted value. While MAPE is the mean of all absolute percentage errors on the predictions, the R^2 score represents how much of the variation in the target values is predictable from the model. R^2 is not suitable to evaluate the numerical error, but it is meaningful to evaluate the prediction model's quality and adaptability. Generally speaking, an accurate regression model should obtain a MAPE lower than 20 % [40,41], and an R^2 of at least 0.50 [24].

We compare our predictive algorithm to the state-of-the-art (s.o.t.a.) solution, which is an RF model, trained only once on historical data using the INT encoding for job features, identified as the best approach in [7,8,24]. We also define two simple baselines *MFREQ* and *MEAN*, which do not exploit learning, but rather return the most frequent and average power consumption value of the jobs in the training set, respectively. Comparison to these simple baselines allows us to estimate the complexity of the prediction tasks. If the baselines cannot obtain accurate prediction, it means that the prediction tasks are indeed non-trivial, and they require the use of a sophisticated ML-based prediction model.

While the approach of [9] is the most similar to ours, a fair comparison is not possible for several reasons. In [9], they filter out the jobs under 1-minute duration (due to limitations in their prediction pipeline). In our setting, this would exclude more than 50 % of the jobs in PM100. Our approach, by contrast, is designed to handle all job types, including ultra-short jobs, making it applicable in full production scenarios without bias. Moreover, they assume that at job submission time, it is possible to obtain the data of all the jobs terminated by that second. This is not realistic in a production scenario, as synchronization and data collection requires considerable time. Also, there is a big problem with real production data, i.e., the one of batch jobs. Batch jobs are submitted and executed together, this means that all the jobs have same or similar end time. If for a new prediction we have only data of batch jobs (or only 1 past job data is present) the formula to compute the prediction in [9] is not sound, and consequently the prediction is not obtainable. Even though we cannot include the method of [9] as a s.o.t.a. approach, we discuss a comparison under a specific setup in the following subsection.

4.3. Experimental results

We first conduct an initial study on the *online* algorithm, to find the best combination of α and β values for each model. We then test the performance of the *offline* and *online* algorithms for the prediction of the $minpcon$, $avgpcon$ and $maxpcon$ values and carry out different evaluations. In our evaluations, we distinguish between the job feature encodings (INT and SB) and the regression models (KNN, XG, RF). We will refer to each prediction model using its feature encoding strategy (INT or SB) "+" regression model (KNN, RF or XG). For instance, the s.o.t.a. solution is INT + RF employed in the *offline* setting.

4.3.1. Online algorithm tuning

We set α to 15, 30, 45 and 60, and β to 1, 2, 5 and 10. We avoid $\beta = 0$, i.e., retraining upon each new job submission, as it incurs excessive overhead, as well as exclude larger values of β so as not to delay model update for long. At the same time, we are not interested in using more than $\alpha = 60$, as otherwise the model would have to deal with a large amount of data. We then observe how different values of α and β impact prediction performance and overhead on the system operations for each prediction model. We evaluate the overhead as the time required to train a model. We consider the best setting to be the one in which the model achieves the lowest MAPE score (in case of equal MAPE values, we consider the higher R^2 score). If a model achieves the same performance (MAPE and R^2), we take the setting with the lowest α and highest β, to save model training time (less frequent training with lower amount of data).

Figs. 5–7 show the results. As expected, in both datasets, with greater α the MAPE improves (gets lower), and as β grows, the MAPE worsens.

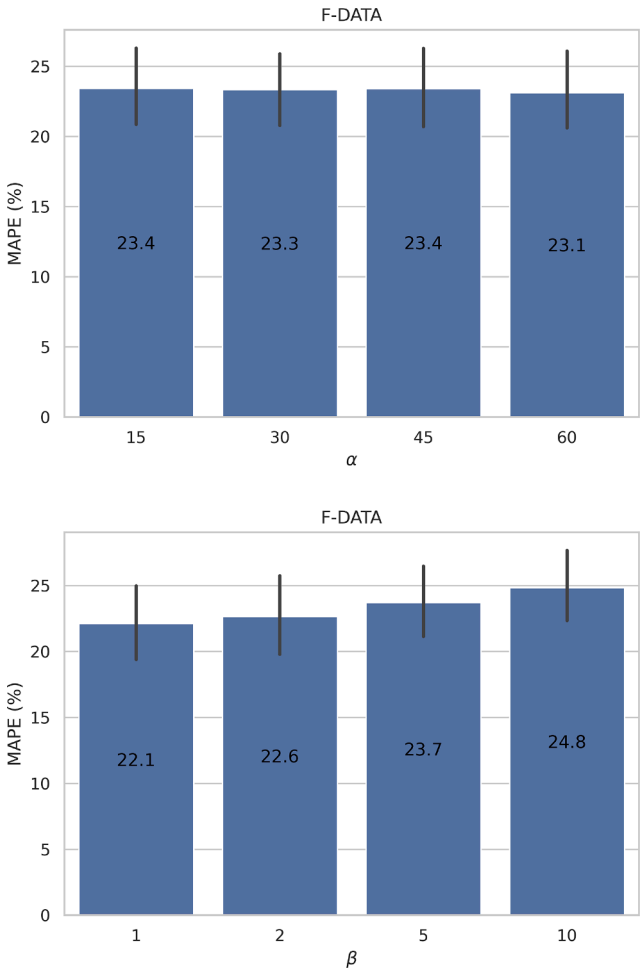

Fig. 5. MAPE scores of the prediction models on F-DATA, with different values of α (above) and β (below). The bars show the average and distribution of the MAPE scores aggregated from the different models. The values are aggregated by same α (above) and β (below). Lower values correspond to better results.

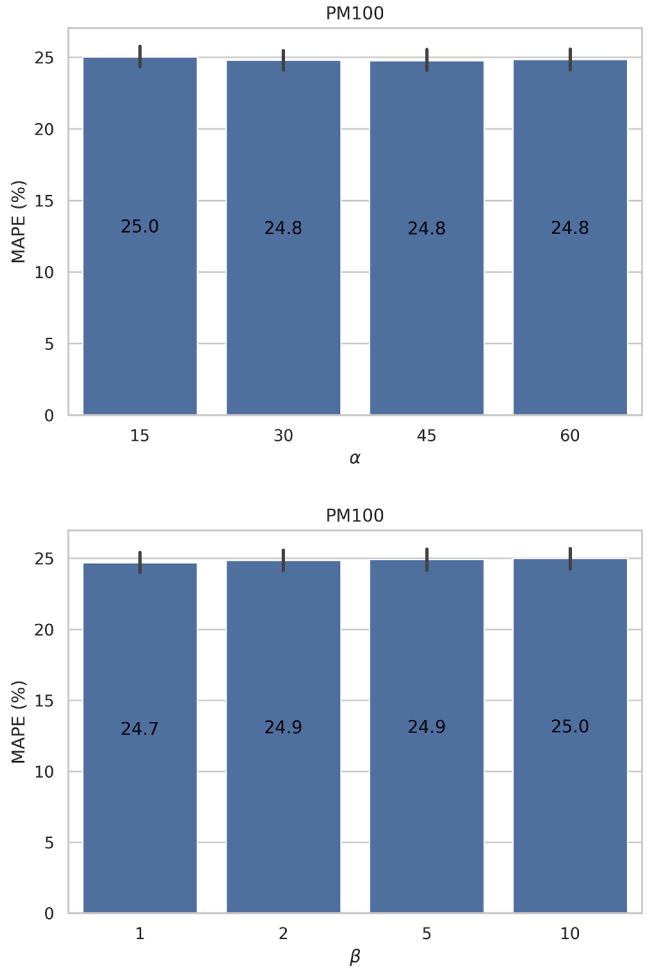

Fig. 6. MAPE scores of the prediction models on PM100, with different values of α (above) and β (below). The bars show the average and distribution of the MAPE scores aggregated from the different models. The values are aggregated by same α (above) and β (below). Lower values correspond to better results.

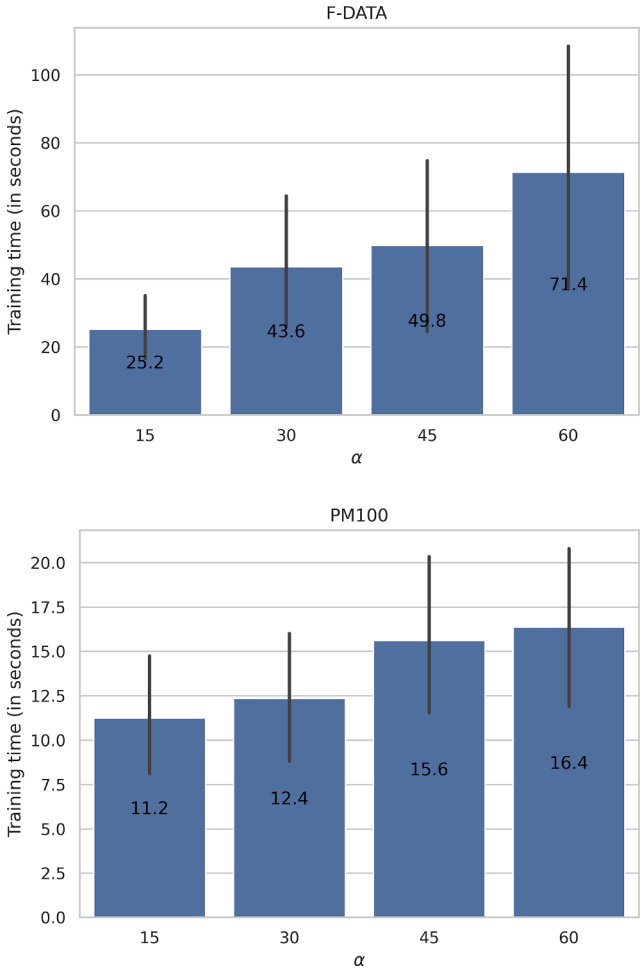

Fig. 7. Training time (in seconds) of the prediction models on F-DATA (above) and PM100 (below), with different values of α. The bars show the average and distribution of the training time aggregated from the different models. Lower values correspond to better results.

While the improvements obtained with higher values of α are minimal ($\alpha = 60$ improves the MAPE by $\sim 0.3\%$ w.r.t. $\alpha = 15$), the increase in the training time is significant, especially in F-DATA, where $\alpha = 60$ takes as almost 3 times the time required by $\alpha = 15$.

In Table 2, we observe that in both datasets, all the models need to be retrained daily ($\beta = 1$) for best prediction performance, confirming the results shown in Figs. 5 and 6. In general, the models using the SB encoding seem to perform better with higher α values, while the majority of the models using the INT encoding achieve the best prediction at $\alpha = 15$. This can be justified by the fact that the SB encoding has more elements than the INT (384 against less than 10); hence, the models need more data to learn correlation between input features and the prediction target.

In the rest of the experiments, we tune the *online* algorithm of each prediction model using the best α and β values derived in Table 2.

4.3.2. Job power consumption prediction

On F-DATA (Table 3), the *online* algorithm brings about significant improvements to MAPE and R^2 in all the prediction models, across all the prediction tasks. The two baselines *MFREQ* and *MEAN* provide poor results, proving that the prediction tasks are non-trivial and require the use of more sophisticated ML models. The benefits of using an *online* algorithm are particularly evident in the models using the SB encoding, where the MAPE is almost halved across all the tasks w.r.t. the *offline* algorithm. The best performing *online* model always lowers the MAPE of the best performing *offline* one by at least 5 %, specifically 5 % in *minpcon* prediction, 10 % in *avgpcon* prediction and 9 % in *maxpcon* prediction. Similarly, the R^2 score improves with the use of the *online* algorithm, which enhances the robustness of the models towards the variability of the job data and power consumption [24]. We observe that the s.o.t.a. solution (INT + RF in the *offline* setting) is outperformed by all the models employed in the *online* setting.

The best performing prediction model across tasks is the INT + RF in the *online* setting, which obtains the overall lowest MAPE values, namely 10 % for the *minpcon* prediction and 12 % for the *avgpcon* and *maxpcon* prediction. In terms of R^2, it is surpassed, by a small factor, only by the SB + RF in the *minpcon* prediction, which scores 0.77 against 0.75 of INT + RF.

Concerning the PM100 data, similar conclusions can be drawn. Here, the MAPE improvements of the *online* algorithm are less significant w.r.t. the ones obtained in F-DATA. This is probably due to the fact that PM100 contains only the jobs that ran exclusively on the nodes, as we discussed previously. This affects the effectiveness of the *online* methodology, as with missing job data it becomes harder to spot job execution patterns and understand how the workload characteristics evolve in time. Nevertheless, the *online* algorithm manages to enhance the prediction performance in all the models, across the three tasks. This is particularly noticeable from the R^2 values, which are always less than 0 with the best performing *offline* models (-0.10, -0.05 and -0.03 with SB + RF in the *minpcon*, *avgpcon* and *maxpcon* prediction, respectively), while the

Table 2
α and β values which yield the best prediction performance for each prediction model, on F-DATA and PM100.

	minpcon		*avgpcon*		*maxpcon*	
Approach	F-DATA	PM100	F-DATA	PM100	F-DATA	PM100
INT + KNN	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 45, \beta = 1$
INT + RF	$\alpha = 45, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 30, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 45, \beta = 1$	$\alpha = 15, \beta = 1$
INT + XG	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 60, \beta = 1$
SB + KNN	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 30, \beta = 1$
SB + RF	$\alpha = 60, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 30, \beta = 1$	$\alpha = 30, \beta = 1$	$\alpha = 60, \beta = 1$	$\alpha = 45, \beta = 1$
SB + XG	$\alpha = 30, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 60, \beta = 1$	$\alpha = 45, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 45, \beta = 1$
MFREQ	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 30, \beta = 1$	$\alpha = 30, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 30, \beta = 1$
MEAN	$\alpha = 15, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 30, \beta = 1$	$\alpha = 30, \beta = 1$	$\alpha = 15, \beta = 1$	$\alpha = 30, \beta = 1$

Table 3
Prediction performance in F-DATA. Lower MAPE values indicate better results, while for R^2 higher values are preferred. The best results are highlighted in bold.

	minpcon				*avgpcon*				*maxpcon*			
	offline		*online*		*offline*		*online*		*offline*		*online*	
Approach	MAPE (%)	R^2	MAPE (%)	R^2	MAPE (%)	R^2	MAPE (%)	R^2	MAPE (%)	R^2	MAPE (%)	R^2
INT + KNN	21	0.36	14	0.61	27	0.46	16	0.67	26	0.47	16	0.67
INT + RF	15	0.66	**10**	0.75	22	0.64	**12**	**0.78**	21	0.66	**12**	**0.79**
INT + XG	20	0.60	11	0.76	29	0.48	14	0.75	27	0.55	14	0.76
SB + KNN	24	0.13	12	0.68	31	−0.08	16	0.66	31	−0.02	16	0.65
SB + RF	25	0.39	11	**0.77**	28	0.32	14	0.76	29	0.24	14	0.76
SB + XG	24	0.36	12	0.76	30	0.27	14	0.77	31	0.22	15	0.75
MFREQ	28	−0.70	43	0.00	45	−1.53	49	0.04	45	−0.04	48	0.04
MEAN	40	−0.01	43	0.00	46	−0.04	48	0.02	45	−0.04	48	0.04

Table 4
Prediction performance in PM100. Lower MAPE values indicate better results, while for R^2 higher values are preferred. The best results are highlighted in bold.

	minpcon				*avgpcon*				*maxpcon*			
	offline		*online*		*offline*		*online*		*offline*		*online*	
Approach	MAPE (%)	R^2	MAPE (%)	R^2	MAPE (%)	R^2	MAPE (%)	R^2	MAPE (%)	R^2	MAPE (%)	R^2
INT + KNN	38	−0.74	30	−0.20	22	−0.16	21	−0.05	38	−1.02	26	−0.01
INT + RF	37	−0.60	28	**0.07**	20	−0.14	20	0.02	28	−0.19	25	0.07
INT + XG	36	−0.34	**27**	−0.48	20	−0.06	**19**	0.07	25	−0.06	**23**	**0.19**
SB + KNN	36	−0.53	30	−0.21	22	−0.16	21	−0.05	34	−0.63	26	0.00
SB + RF	33	−0.10	28	−0.02	20	−0.05	**19**	**0.08**	25	−0.03	24	0.13
SB + XG	35	−0.23	28	−0.03	20	−0.05	**19**	**0.08**	26	−0.08	24	0.13
MFREQ	34	−0.72	28	0.03	30	−1.91	22	0.03	35	−1.89	26	0.01
MEAN	34	−0.20	28	0.03	22	−0.25	22	0.03	24	−0.13	26	0.01

best performing *online* models always score more than 0.07 (0.07 with INT + RF in the *minpcon* prediction, 0.08 with SB + RF in the *avgpcon* prediction, and 0.19 with INT + XG in the *maxpcon* prediction). Once again, the s.o.t.a. solution (INT + RF in the *offline* setting) is outperformed by all the models employed in the *online* setting.

No specific model emerges as the overall best across tasks. For the *minpcon* prediction, we consider INT + RF as the best prediction model, as it obtains the highest R^2 (0.07) and the MAPE value (28 %) is only 1 % higher than the absolute best for this task, i.e., 27 % by INT + XG. While SB + RF and SB + XG obtain the same best results in the *avgpcon* prediction, we pick SB + RF, since its *online* setup requires a lower α (30 against 45 of SB + XG), as shown in Table 2. Finally, in the *maxpcon* prediction, INT + XG obtains the lowest MAPE (23 %) and the highest R^2 (0.19). The *MFREQ* and *MEAN* baselines obtain better results w.r.t. the experiments on F-DATA, however, they are still outperformed by the ML models.

Comparing Tables 3 and 4, it is clear that the MAPE values scored by the prediction models on PM100 are slightly higher, while the R^2 values are noticeably lower than those scored on F-DATA. This can be explained by the differences in the job power consumption values in PM100. As Fig. 4 shows, these values lie in a wider range (from few watts to almost 2000) compared to those in F-DATA (0–175 W), making the prediction task significantly harder. Moreover, as already mentioned, the problem is exacerbated by the fact that PM100 contains just a subset of the job executions.

A final observation is that, in the *online* algorithm, SB and INT encodings achieve similar results in both datasets. Despite being slightly less accurate, SB encoding allows to preserve data privacy, as the models are never trained on explicit user and job information. Moreover, the INT encoding is strictly related to the system, because the way the original values are mapped to integers is dictated by the historic information and system specifics. Conversely, the SB encoding does not depend on any previous knowledge on the jobs or the system. The SB encoding thus offers a general solution to share data among systems for analysis (e.g. clustering to find similar jobs or workload characterization) and transfer learning studies, while not violating user privacy.

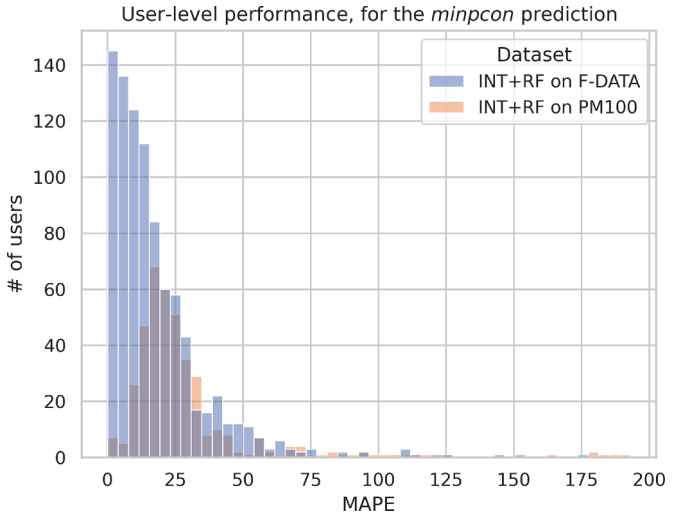

Fig. 8. Distribution of the MAPE per user for the *minpcon* prediction in both datasets.

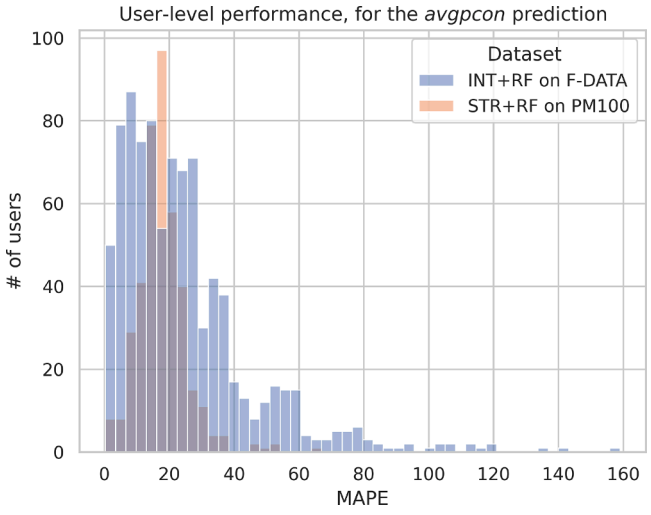

Fig. 9. Distribution of the MAPE per user for the *avgpcon* prediction in both datasets.

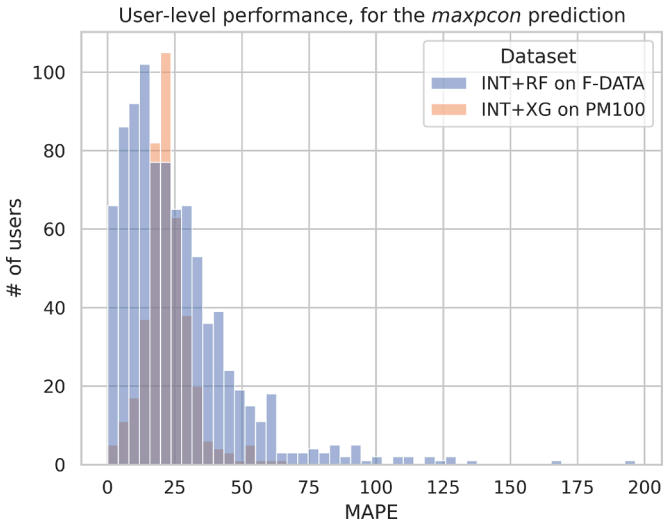

Fig. 10. Distribution of the MAPE per user for the *maxpcon* prediction, in both datasets.

4.3.3. Comparison to [9]

As discussed in Section 4.2, the method proposed in [9] cannot be compared to ours in a realistic setting. Nevertheless, we do this comparison under a different setting. First, we remove the jobs shorter than one minute. Our solution neatly outperforms the other, in the prediction of *minpcon* (MAPE of 27 % against 28 %, and 10 % against 17 %, in PM100 and F-DATA, resp.), *avgpcon* (MAPE of 16 % against 17 %, and 9 % against 17 %), and *maxpcon* (same MAPE of 23 %, and 9 % against 20 %).

Then we test the solution of [9] without removing any jobs. For the reasons outlined in Section 4.2, many jobs cannot be evaluated. Still, the method obtains a MAPE of 28 % and 17 % on PM100 and F-DATA respectively for the prediction of the *minpcon*, a MAPE of 19 % and 23 % for the prediction of the *avgpcon*, and MAPE of 24 % and 23 % for the prediction of the *maxpcon*. Our solution reveals significantly better results on F-DATA (Table 3). On PM100 (Table 4), we obtain a slightly lower MAPE for the prediction of the *minpcon* and *maxpcon* (28 % vs 27 % and 24 % vs 23 %), and same MAPE for the *avgpcon* prediction (19 %).

Overall, our method outperforms that of [9] and is capable of generating a prediction for any type of job. As such, we can conclude that our solution is more accurate, more robust and more suitable for real production systems.

4.3.4. User-level evaluation

Figs. 8–10 show the distribution of the MAPE values of the prediction models across all the users of the two datasets. For each prediction task, we consider the best prediction model reported in Tables 3 and 4. We observe that in all the tasks, the majority of the MAPE values obtained by the models on F-DATA are below 20 %, while on PM100 they exhibit higher density around 25 %. These values are in line with the overall results presented in Tables 3 and 4. The results suggest that our prediction algorithm can be effectively employed as a tool to inform the end-users about the expected power consumption of their jobs. As argued in [42], such a tool is fundamental to encourage the adoption of greener systems where energy cost of jobs needs to be made explicit and accounted for in the pricing scheme.

4.3.5. System power prediction

After testing the models at job level, we evaluate them at system level. We do this for two reasons, (i) to see if our models are capable of estimating the system power state accurately by considering only the submitted jobs, and (ii) because the prediction error on a single job is either an overestimate or an underestimate of the actual power consumption, we believe that such errors might cancel out each other at the system level, given the large amount of jobs running concurrently.

We consider all the jobs for which we computed the predictions, and we group them based on the hour of the day when they were running. For all the hours of all the days $d \in D$, we compute the system power consumption as the sum of the actual power consumption of all the running jobs. Then, we compute the daily system power consumption $psys_d$, as the average of all the hourly ones. By replacing the actual job power consumption with the generated prediction, multiplied by the number of nodes allocated to the jobs, we can also compute the predicted system power consumption $\overline{psys}_d$. We evaluate the numerical error of the predictions by computing the *mean error* score in Eq. 4, as defined in [7]. In addition, we also calculate the R^2 score.

$$mean\ error = \frac{1}{|D|} \sum_{d \in D} \frac{|psys_d - \overline{psys}_d|}{psys_d} * 100 \tag{4}$$

In Figs. 11–13 we show the performance of the best *online* prediction model for the prediction of the *minimum*, *average* and *maximum* system power consumption. The plots illustrate the effectiveness of our prediction models in forecasting system power consumption over time.

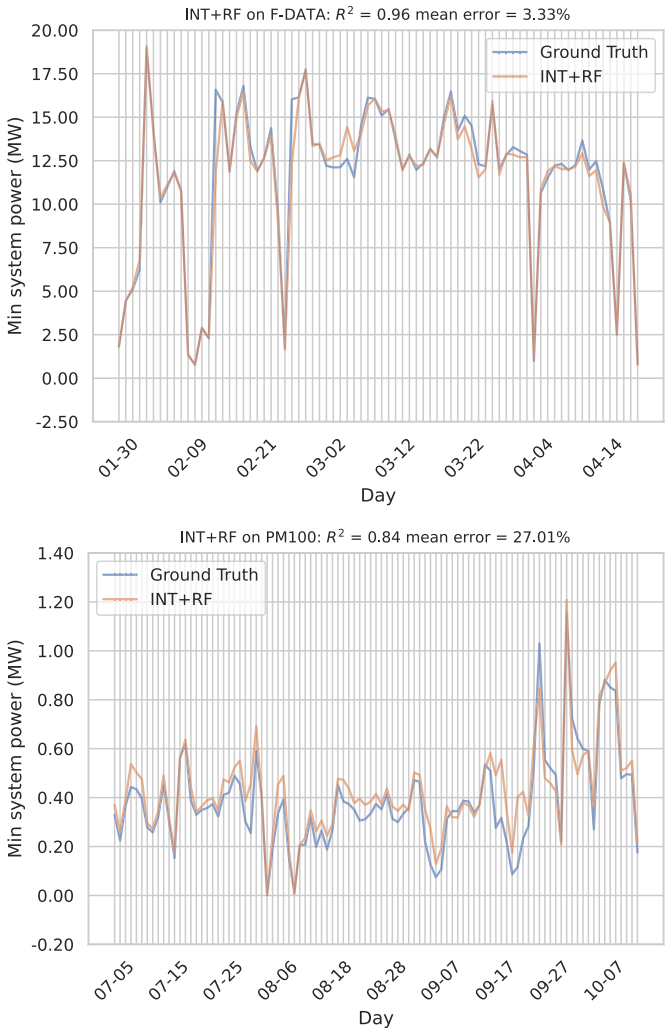

Fig. 11. System true and predicted power for the *mipcon* prediction on F-DATA (above) and PM100 (below) with the best prediction model.

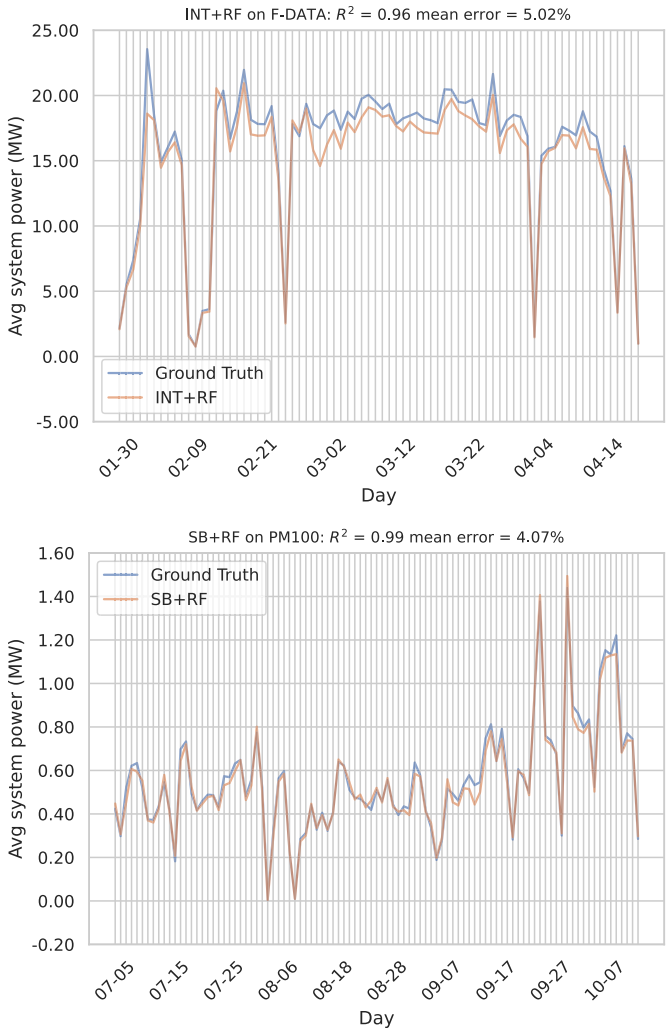

Fig. 12. System true and predicted power for the *avgcon* prediction on F-DATA (above) and PM100 (below) with the best prediction model.

On F-DATA, the models obtain an R^2 of 0.96, and a *mean error* of less than 5 %, across all the tasks. On PM100, the SB + RF is particularly suited for the estimation of the *average* system power consumption, as it obtains an R^2 of 0.99, and a *mean error* of 4 %. The estimation of the *maximum* system power consumption can be achieved effectively by the INT + XG model, which scores an R^2 of 0.94, and a *mean error* of 8 %. The only case where our models fall short is the estimation of the *minimum* system power consumption on PM100, where the INT + RF model obtains a *mean error* greater than 27 %.

We believe that the prediction of the *minimum* system power consumption is not more informative than the *average* and *maximum*, in the context of system power management. The *minimum* system power consumption can give an idea of the minimum electrical power required to sustain the system load. However, this information can also be extracted from the *average* and *maximum* power consumption, which represent the average and maximum electrical power required, respectively. Whereas, predicting the *average* and *maximum* system power consumption can be instrumental to estimate the electricity cost for budget allocation, and devise energy-based billing to charge users based on actual cost of their job executions [42]. Moreover, these values can help avoid undesired spikes in the system power consumption, which can potentially cause power outages, or damages to the electrical grid. This information is not retrievable from the *minimum* system power consumption.

The results suggest that our prediction algorithm can estimate system power consumption effectively, even without explicit knowledge of the system's internal operations. Our approach can thus be instrumental in identifying power efficiency trends and estimating system power consumption without requiring extensive instrumentation or direct access to system parameters.

4.3.6. Overhead

Another aspect to consider when working with an *online* prediction algorithm is its overhead on the system operations incurred by model training and inference time. Since a model is retrained at most daily, it can be executed in the background or asynchronously from the system operations not to interfere with the system normal functioning. Nevertheless, as Fig. 7 shows, the training time is negligible even in the worst case scenario, where it would still be of less than 2 minutes. Instead, the inference time is defined as the time required by the algorithm to generate a prediction for a given job. For the approach to be practical in a real production system, inference time should be as little as possible, not to slow down the system operations. One way to estimate how little this time should be is to compare it to the average waiting of the jobs in the system, i.e., the average time needed by the job scheduling software to make a scheduling decision after the job is submitted. If the inference time of the prediction algorithm is a fraction of the average waiting time, then the prediction algorithm can be seamlessly integrated in the workload submission pipeline.

We compute the average job waiting time for F-DATA and PM100 by subtracting the *scheduling_time* (i.e., the timestamp when the scheduling

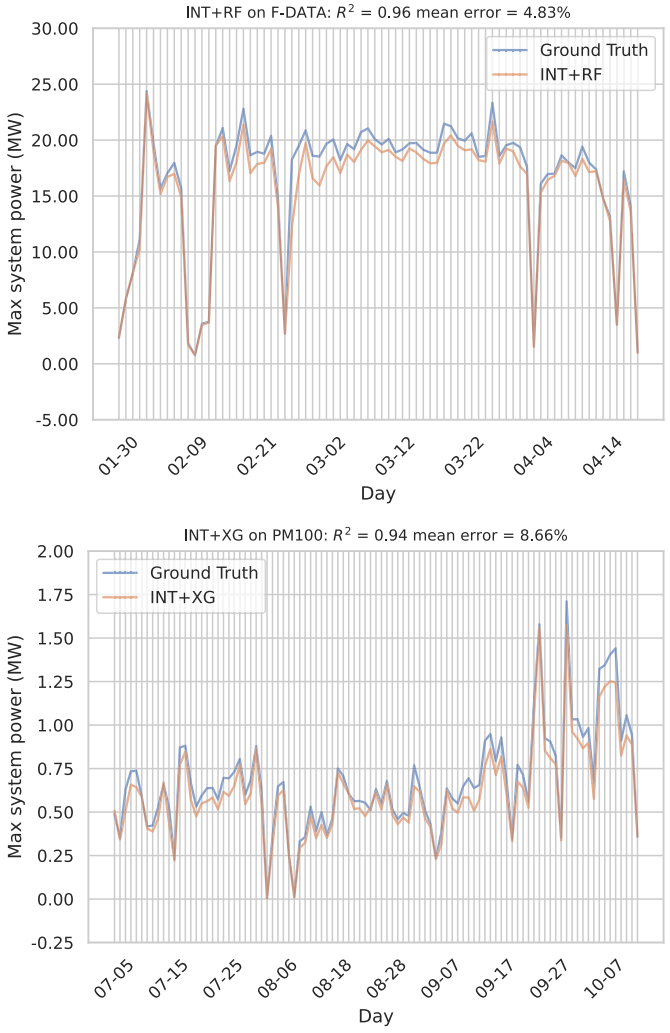

Fig. 13. System true and predicted power for the *maxcon* prediction on F-DATA (above) and PM100 (below) with the best prediction model.

decision is performed) and *submission_time* for all the jobs, and by computing the average of these values. We notice that in both datasets, the average value is greater than 10 minutes. In F-DATA, INT + RF (the best model in all the prediction tasks) performs inference in less than 0.12 seconds. Similarly, in PM100, INT + RF in *minpcon* prediction infers in 0.07 seconds , and the same holds for SB + RF in *avgpcon* (0.08 seconds) and INT + XG in *maxpcon* (0.03 seconds). We note that these times include also the time needed for *Data Preparation* and *Feature Encoding*. We can thus conclude that the prediction algorithm can be seamlessly used in a production system, as its inference time is negligible w.r.t. the average job waiting time.

We further evaluate the energetic overhead of our solution. Our algorithm consumes very low power and energy. Specifically, in the host machine, we observed power consumption of around 80 W during model training/inference. The daily operations of the models are the training plus the inferences on all the jobs. The average time to perform these operations is less than 20 minutes a day, meaning that the daily energy consumption of our solution is comparable to one of the shortest and lowest-consuming Fugaku jobs. Considering that the load of production systems (like Fugaku) is tens of thousands of job executions per day, we can conclude that the energetic impact of our solution is negligible.

5. Limitations

In this section, we discuss the possible limitations of our approach.

5.1. Job name feature

We acknowledge that a user may choose a meaningless *job name* (e.g. test1, run4), which may not provide useful information for prediction and possibly infer noise in the model training. A related consideration is that users may execute jobs with the same *job name* using different input, which may result in significant alterations in the job power consumption. Our approach is explicitly designed to handle such cases gracefully through the following mechanisms.

- **Encoding robustness.** In addition to traditional integer encoding (INT), we leverage SBert-based semantic embeddings (SB), which are robust to variations and can still capture latent similarities even when job names are generic. For instance, SBert can distinguish between different contextual uses of the term “test“ if accompanied by different environment descriptors, user profiles, or resource configurations.
- **Multi-feature generalization.** While *job name, user*, and *environment* are indeed key features, they are never used in isolation. Our feature set also includes quantitative parameters (e.g., *# cores*, *# GPUs*, *memory requested, QoS class, partition*, etc.) that capture hardware configuration and execution context. These features, together with periodic model retraining in our online learning setup, help the model adapt to job behavior-even when jobs with the same binary or job name are launched with different input or configurations.

In essence, while semantic features help in identifying recurring job patterns (e.g., within user batches), the model does not rely solely on textual *job name*, and we expect our approach to remain effective even when such fields are ambiguous or repeated across users. We also note that we are dealing with data of real production systems, where users tend to name their jobs properly based on the application and its input, to keep track of their experiments and for accountability reasons. This was also observed by us in a preliminary inspection of the *job name* values of the two datasets. Moreover, in both datasets, we have access to the *command* feature, which reports the explicit command line used to execute the job. In case of doubts on the validity of the *job name*, our methodology can be adapted to use the *command* feature, so as to have more reliable information on the application executed and its input. However, we observed during the feature selection phase that, for prediction performance, this feature is not more informative than *job name*.

5.2. Aggregated job power consumption

In this work, we target the prediction of the *minimum*, *average* and *maximum* power consumption of a job, aggregated throughout the job execution and normalized on the # of nodes allocated to the job execution. Such an approach yields several advantages, as well as two main limitations.

First, working with aggregated job power consumption (i.e., a single floating-point value), instead of the full power profile (i.e., the full time-series of power consumption values sampled periodically throughout the job’s execution), allows for an easy characterization of the job power consumption, which has been widely used for scheduling and accountability purposes [3,42]. It is not always feasible to extract the full power profile in production systems [25], and indeed F-DATA does not include this information. A possible limitation of our choice is that the temporal profiling of the job power consumption can be instrumental for real-world scheduling. For instance, long-running jobs may present multiple execution phases, with very different power demands. Such an information would be important to perform more fine-grained scheduling decisions, aiming at maximizing the resource utilization and minimizing the strain on the system resources. The prediction of the full power profile requires a completely different set of prediction algorithms, which we plan to investigate in future work using the PM100 data.

Second, normalizing the power consumption on the # of nodes allocated to the job execution allows to make the prediction less error prone and, again, it is a widely used representation for scheduling and accountability purposes [3,42]. However, this does not account for cases where each node is operating on a different load, possibly resulting in significantly different power consumption across nodes. In future work, we will explore job power consumption prediction on individual nodes by applying our algorithms to the data of each node's power consumption.

5.3. Job energy and duration prediction

As discussed in Section 3, our algorithm can be used to estimate the job energy consumption, which is necessary for energy- and carbon-aware scheduling algorithms [4,43]. This can be done by multiplying the predicted job power consumption by the wall-time (maximum job's duration) set by the user. However, the duration set by the user is often a huge overestimation [44]; hence the accurate prediction of the job's energy consumption requires the development of an auxiliary job duration prediction model, such as those proposed in [44,45]. To this end, the creation of a job duration prediction algorithm, and the study of the combination of different predictions, are separate research topics, which are outside the scope of this work. So is also the evaluation of the estimation of job energy consumption. We acknowledge that this can partially limit the applicability of our prediction algorithm for scheduling purposes. Nevertheless, the prediction of job power consumption alone is already fundamental information for power aware job scheduling [3,5] and enforcement of energy-aware practices [42] which do not require the job duration as input. In future work, we plan to expand our algorithm to also predict the job duration.

6. Conclusions

In this work, we presented a predictive algorithm to perform job power consumption in HPC systems, before their execution. Our solution is designed to be deployed in an *online* context, aiming to be suitable for a real production environment where job data from multiple users are live and streaming in time. The algorithm relies on standard ML regression models, which are re-trained over time to adapt to the change of workload in the system and optimize prediction performance. The accurate prediction of job power consumption, prior to their execution, is fundamental to develop energy-aware workload management strategies (e.g., job scheduling or resource allocation policies) to improve the energy efficiency of the system [5,6] and foster environmentally sustainable HPC practices.

We tested our prediction algorithm on two datasets extracted from production HPC systems, namely PM100 and F-DATA. For each dataset, we investigated the prediction of three different targets, namely the *minimum*, *average* and *maximum* power consumption, per job. For each task on each dataset, we empirically evaluated different algorithm settings to find the one which yields the best prediction performance. To this end, we experimented with different re-training frequency, amount of training data and the ML model used. We demonstrated that, with the optimal setting, our prediction algorithm is able to accurately predict *minimum*, *average* and *maximum* job power consumption across the datasets, obtaining an error of less than 12% on F-DATA and less than 22% on PM100. The prediction model is also proven effective in predicting the whole system power consumption, which is fundamental to devise strategies for system power management.

In future work, we foresee the integration of our solution in the workload submission pipeline of a production system, aiming to drive energy-aware workload management strategies to improve the environmental sustainability of present and future HPC systems. Moreover, we plan to investigate how variability modeling (e.g., via quantile regression or predictive intervals) and lightweight anomaly detection can be integrated without compromising the online, fast-inference nature of our pipeline, aiming to detect unexpected job behavior. An interesting direction is also the investigation of event- (such as the appearance of new users or applications) or statistical drift-based re-training strategies, rather than the daily one presented in this paper. Finally, we plan to expand our algorithm to the prediction of other job execution characteristics, such as duration, energy consumption and failure.

Acknowledgement

This research was partly supported by the HORIZON-RIA DECICE project (g.a. 101092582), the HE EU Graph-Massivizer project (g.a. 101093202), the EuroHPC JU SEANERGYS project and the Spoke "FutureHPC & BigData" and "Multiscale Modelling & Engineering Applications" of the ICSC-Centro Nazionale di Ricerca in "High Performance Computing, Big Data & Quantum Computing", funded by the EU - NextGenerationEU.

CRediT authorship contribution statement

Francesco Antici: Writing - review & editing, Writing - original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Andrea Borghesi:** Writing - original draft, Validation, Conceptualization; **Zeynep Kiziltan:** Writing – review & editing, Validation, Conceptualization; **Jens Domke:** Writing – review & editing, Validation, Conceptualization; **Andrea Bartolini:** Writing – review & editing, Validation, Conceptualization.

Data availability

The data is already public, the code is made available.

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.

References

[1] B. Li, R. Basu Roy, D. Wang, S. Samsi, V. Gadepally, D. Tiwari, Toward sustainable HPC: carbon footprint estimation and environmental implications of HPC systems, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2023, pp. 1–15.

[2] S. Loganathan, R.D. Saravanan, S. Mukherjee, Energy aware resource management and job scheduling in cloud datacenter, Int. J. Intell. Eng. Syst. 10 (4) (2017).

[3] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, Scheduling-based power capping in high performance computing systems, Sustain. Comput. Inf. Syst. 19 (2018) 1–13.

[4] B. Kocot, P. Czarnul, J. Proficz, Energy-aware scheduling for high-performance computing systems: a survey, Energies 16 (2) (2023) 890.

[5] B. Qureshi, Profile-based power-aware workflow scheduling framework for energy-efficient data centers, Future Gen. Comput. Syst. 94 (2019) 453–467.

[6] M. Shaukat, W. Alasmary, E. Alanazi, J. Shuja, S.A. Madani, C.-H. Hsu, Balanced energy-aware and fault-tolerant data center scheduling, Sensors 22 (4) (2022) 1482.

[7] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, Predictive modeling for job power consumption in HPC systems, in: High Performance Computing: 31st International Conference, ISC High Performance 2016, Frankfurt, Germany, June 19–23, 2016, Proceedings, Springer, 2016, pp. 181–199.

[8] T. Saillant, J.-C. Weill, M. Mougeot, Predicting job power consumption based on rjms submission data in hpc systems, in: High Performance Computing: 35th International Conference, ISC High Performance 2020, Frankfurt/Main, Germany, June 22–25, 2020, Proceedings 35, Springer, 2020, pp. 63–82.

[9] D. Carastan-Santos, G. Da Costa, M. Poquet, P. Stolf, D. Trystram, Light-weight prediction for improving energy consumption in HPC platforms, in: J. Carretero, S. Shende, J. Garcia-Blas, I. Brandic, K. Olcoz, M. Schreiber (Eds.), Euro-Par 2024: Parallel Processing, Springer Nature Switzerland, Cham, 2024, pp. 152–165.

[10] B. Wang, Z. Chen, N. Xiao, A survey of system scheduling for hpc and big data, in: Proceedings of the 2020 4th International Conference on High Performance Compilation, Computing and Communications, 2020, pp. 178–183.

[11] F. Antici, A. Bartolini, Z. Kiziltan, O. Babaoglu, Y. Kodama, MCBound: an online framework to characterize and classify memory/Compute-bound HPC jJobs, in: To appear in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2024.

[12] F. Antici, A. Borghesi, Z. Kiziltan, Online job failure prediction in an HPC system, in: Euro-Par 2023: Parallel Processing Workshops: Euro-Par 2023 International Workshops, Limassol, Cyprus, August 28–September 1, 2023, Revised Selected Papers, Springer Nature, 2023.
[13] S. Madireddy, P. Balaprakash, P. Carns, R. Latham, G.K. Lockwood, R. Ross, S. Snyder, S.M. Wild, Adaptive learning for concept drift in application performance modeling, in: Proceedings of the 48th International Conference on Parallel Processing, 2019, pp. 1–11.
[14] F. Antici, K. Yamamoto, J. Domke, Z. Kiziltan, Augmenting ML-based predictive modelling with NLP to forecast a job's power consumption, in: Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, 2023, pp. 1820–1830.
[15] A. Borghesi, C. Di Santi, M. Molan, M.S. Ardebili, A. Mauri, M. Guarrasi, D. Galetti, M. Cestari, F. Barchi, L. Benini, et al., M100 Exadata: a data collection campaign on the CINECA's marconi100 tier-0 supercomputer, Sci. Data 10 (1) (2023) 288.
[16] A. Bartolini, F. Beneventi, A. Borghesi, D. Cesarini, A. Libri, L. Benini, C. Cavazzoni, Paving the way toward energy-aware and automated datacentre, in: Workshop Proceedings of the 48th International Conference on Parallel Processing, ICPP Workshops '19, Association for Computing Machinery, New York, NY, USA, 2019. https://doi.org/10.1145/3339186.3339215
[17] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.
[18] T. Chen, C. Guestrin, Xgboost: a scalable tree boosting system, in: Proceedings of the 22nd ACM Sigkdd International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.
[19] J. Devlin, M.-W. Chang, K. Lee, et al., BERT: pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 NAACL: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186.
[20] H. Choi, J. Kim, S. Joe, Y. Gwon, Evaluation of bert and albert sentence embedding performance on downstream nlp tasks, in: 2020 25th International Conference on Pattern Recognition (ICPR), IEEE, 2021, pp. 5482–5487.
[21] Y. Chu, H. Cao, Y. Diao, H. Lin, Refined SBERT: representing sentence BERT in manifold space, Neurocomputing 555 (2023) 126453.
[22] B. Bugbee, C. Phillips, H. Egan, R. Elmore, K. Gruchalla, A. Purkayastha, Prediction and characterization of application power use in a high-performance computing environment, Statist. Anal. Data Mining ASA Data Sci. J. 10 (3) (2017) 155–165.
[23] K. Yamamoto, Y. Tsujita, A. Uno, Classifying jobs and predicting applications in HPC systems, in: High Performance Computing: 33rd International Conference, ISC High Performance 2018, Frankfurt, Germany, June 24–28, 2018, Proceedings 33, Springer, 2018, pp. 81–99.
[24] A. Sîrbu, O. Babaoglu, Power consumption modeling and prediction in a hybrid CPU-GPU-MIC supercomputer, in: Euro-Par 2016: Parallel Processing: 22nd International Conference on Parallel and Distributed Computing, Grenoble, France, August 24–26, 2016, Proceedings 22, Springer, 2016, pp. 117–130.
[25] F. Antici, M. Seyedkazemi Ardebili, A. Bartolini, Z. Kiziltan, PM100: a job power consumption dataset of a large-scale production HPC system, in: Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, 2023, pp. 1812–1819.
[26] D. Bodas, J. Song, M. Rajappa, A. Hoffman, Simple power-aware scheduler to limit power consumption by hpc system within a budget, in: 2014 Energy Efficient Supercomputing Workshop, IEEE, 2014, pp. 21–30.
[27] D. Rajagopal, D. Tafani, Y. Georgiou, D. Glesser, M. Ott, A novel approach for job scheduling optimizations under power cap for arm and Intel hpc systems, in: 2017 IEEE 24th International Conference on High Performance Computing (HiPC), IEEE, 2017, pp. 142–151.
[28] G. Juve, B. Tovar, R.F. Da Silva, D. Król, D. Thain, E. Deelman, W. Allcock, M. Livny, Practical resource monitoring for robust high throughput computing, in: 2015 IEEE International Conference on Cluster Computing, IEEE, 2015, pp. 650–657.
[29] N. Sukhija, E. Bautista, Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus, in: 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation, IEEE, 2019, pp. 257–262.
[30] J. Li, G. Ali, N. Nguyen, J. Hass, A. Sill, T. Dang, Y. Chen, Monster: an out-of-the-box monitoring tool for high performance computing systems, in: 2020 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2020, pp. 119–129.
[31] M. Colmant, P. Felber, R. Rouvoy, L. Seinturier, Wattskit: software-defined power monitoring of distributed systems, in: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), IEEE, 2017, pp. 514–523.
[32] J. Muraña, S. Nesmachnow, F. Armenta, A. Tchernykh, Characterization, modeling and scheduling of power consumption of scientific computing applications in multicores, Cluster Comput. 22 (2019) 839–859.
[33] T. Patel, A. Wagenhäuser, C. Eibel, T. Hönig, T. Zeiser, D. Tiwari, What does power consumption behavior of hpc jobs reveal?: demystifying, quantifying, and predicting power consumption characteristics, in: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2020, pp. 799–809.
[34] H. Khaleghzadeh, R. Reddy Manumachu, A. Lastovetsky, Efficient exact algorithms for continuous bi-objective performance-energy optimization of applications with linear energy and monotonically increasing performance profiles on heterogeneous high performance computing platforms, Concurrency Comput. Prac. Exper. 35 (20) (2023) e7285.
[35] K.S. Sree, J. Karthik, C. Niharika, P. Srinivas, N. Ravinder, C. Prasad, Optimized conversion of categorical and numerical features in machine learning models, in: 2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), IEEE, 2021, pp. 294–299.
[36] Z. Xu, Z. Guo, N. Cristianini, On compositionality in data embedding, in: International Symposium on Intelligent Data Analysis, Springer, 2023, pp. 484–496.
[37] F. Antici, A. Borghesi, J. Domke, Z. Kiziltan, UoPC: a user-based online framework to predict job power consumption in HPC systems, in: ISC High Performance 2025 Research Paper Proceedings (40th International Conference), 2025, pp. 1–12.
[38] W. Yang, J. Yu, G. Xing, Long-term analysis for job characteristics on the supercomputer, in: Proceedings of the 2021 5th High Performance Computing and Cluster Technologies Conference, 2021, pp. 1–12.
[39] N. Reimers, I. Gurevych, Sentence-bert: sentence embeddings using siamese bert-networks, arXiv preprint arXiv:1908.10084 (2019).
[40] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, I. Stoica, Ernest: efficient performance prediction for {Large-Scale} advanced analytics, in: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), 2016, pp. 363–378.
[41] K. Assogba, E. Lima, M.M. Rafique, M. Kwon, PredictDDL: reusable workload performance prediction for distributed deep learning, in: 2023 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2023, pp. 13–24.
[42] A. Borghesi, A. Bartolini, M. Milano, L. Benini, Pricing schemes for energy-efficient HPC systems: design and exploration, Int. J. High Perform. Comput. Appl. 33 (4) (2019) 716–734.
[43] H. Badri, T. Bahreini, D. Grosu, K. Yang, Energy-aware application placement in mobile edge computing: a stochastic optimization approach, IEEE Trans. Parallel Distrib. Syst. 31 (4) (2019) 909–922.
[44] A.B. Faisal, N. Martin, H.M. Bashir, S. Lamelas, F.R. Dogar, When will my ML job finish? Toward providing completion time estimates through predictability-centric scheduling, in: 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), USENIX Association, Santa Clara, CA, 2024, pp. 487–505. https://www.usenix.org/conference/osdi24/presentation/bin-faisal.
[45] Y.-C. Wang, T. Chen, M.-C. Chiu, An explainable deep-learning approach for job cycle time prediction, Decis. Anal. J. 6 (2023) 100153.