Black Sea Journal of Engineering and Science
doi: 10.34248/bsengineering.1963790

Open Access Journal
e-ISSN: 2619 – 8991

Research Article
Volume 9 - Issue 4: 1905-1922 / July 2026

FROM WAIT TIME TO SLOWDOWN: A HURDLE FRAMEWORK FOR PERFORMANCE-CENTRIC METRIC PREDICTION IN HPC CLUSTERS

Ömer DAKKAK[1]*

[1]*Karabük University, Faculty of Computer and Information Sciences, Department of Computer Engineering, 78050, Karabük, Türkiye*

Abstract: The demand for High-Performance Computing (HPC) has been growing since the rise of AI in the last few years. Supercomputers are in demand more than ever across many fields for running generative models, bitcoin mining, gaming services, and other computing applications that are increasingly run in the cloud. Scheduling in HPC has always been one of the most interesting research areas, especially in the 2000s, when grid computing emerged as a promising new technology in the domain. However, research in this area has been in the shadows for a while as the grid has matured into cloud-based systems in practice. Scheduling for HPC seemed to be a saturated research area, while new research trends have taken off in all directions. The recent big leap in AI models has revived this domain for two main reasons. Many classic, heuristic, and metaheuristic scheduling algorithms can be reconceptualized by integrating them with machine learning-based algorithms or frameworks, likely leading to better performance than the classical models used previously. Additionally, the urgent need for supercomputing for running the AI models that are used widely and daily, almost for every task, urges the area of scheduling algorithms itself. This paper proposes two novel AI models. The first model is based on the hurdle framework. In contrast, the second model is based on a stacking classifier followed by a stacking regression for two real datasets, using three performance metrics: wait time, slowdown, and response time, compared with the Random Forest solo model. The Hurdle model achieved the best performance on zero-inflated targets, while the Two-Stacking Stage showed better temporal robustness. When zero inflation is absent, simpler models can be more effective.

Keywords: High-performance computing, Job scheduling, Wait time prediction, Zero-inflated data, Hurdle model, Stacking regression

***Corresponding author:** Karabük University, Faculty of Computer and Information Sciences, Department of Computer Engineering, 78050, Karabük, Türkiye
E mail: omardakkak@karabuk.edu.tr (Ö. DAKKAK)
Ömer DAKKAK https://orcid.org/0000-0001-9767-5685

Received: June 04, 2026
Accepted: July 02, 2026
Published: July 15, 2026

Cite as: Dakkak, Ö. (2026). From wait time to slowdown: a hurdle framework for performance-centric metric prediction in hpc clusters. *Black Sea Journal of Engineering and Science*, 9(4), 1905-1922.

1. Introduction

High-performance computing (HPC) systems are widely used for many large and complex computational systems. Scientific fields with heterogeneous, dynamic environments that require the computational resources of today's HPC platforms are emerging more than ever. HPC systems are implemented in many critical scientific applications such as climate modeling, high-energy physics, and astronomy (Berriman et al., 2006; Chang, 2018; Kurth et al., 2018; Kolker-Hicks et al., 2023). While some fields focus on efficiency, others target productivity, leading to scripts that are not well optimized or tuned for HPC hardware (Gainaru et al., 2019). HPC systems handle hundreds of thousands of jobs simultaneously, and these jobs need to be scheduled as optimally as possible to achieve reasonable execution times. This is conducted via a centralized HPC job scheduler that runs these jobs in order. This order is based on the user's preferences. For instance, to achieve shorter waiting times, First-Come-First-Serve (FCFS) is used, whereas if job length is a priority, Shortest-Job-First (SJF) is considered. However, traditional scheduling algorithms do not fully utilize the system's resources. This is due to the static nature of these algorithms, which prioritize scheduling a high-priority job even when sufficient resources are not available. This massively underutilizes the system, whereas other ready jobs can be scheduled on available resources yet remain in the queue. In a given HPC environment, this can become quite costly.

Most current HPC schedulers are based on a resource-reservation system. This means that the user declares the requirement for running the application upon submission (Bailey Lee et al., 2004; Mu'alem and Feitelson, 2001). For optimal performance, the user must provide a decent level of accuracy. Otherwise, fewer resources demanded by the user might lead to many jobs being terminated (killed), while overestimation results in overallocation, which is a significant waste of valuable computational resources (Gaussier et al., 2015). Therefore, most HPC systems overestimate their resources, so idle time is filled with small jobs using

backfilling algorithms (Kolker-Hicks et al., 2023).

The current HPC systems are overwhelming users with a wide and diverse range of resources and services, including heterogeneous nodes with non-uniform memory hierarchies, burst buffers, vectorization units, and unified virtual memory. This diversity has made the user's estimation of resources more challenging than before, even though the scientific application is quite familiar to them. That complexity has led to widespread backfilling in today's systems to overcome the resource-estimation dilemma. However, the performance of backfilling algorithms is also affected by the accuracy of job execution time estimates, as several studies have shown (Feitelson et al., 2004; Gainaru et al., 2019).

Here, the concept of "backfilling" shines. This technique involves scheduling low-priority jobs earlier to wait for resources allocated to pending high-priority jobs. The performance of this approach relies on the "job runtime" to compute the start time of ready jobs without further delaying them. It is a common belief that more accurate estimates of job runtime will lead to better backfilling and more effective scheduling.

Backfilling might lead to issues: the backfilled job might delay jobs ahead in the schedule, violating the Conservative Backfilling (CONS) concept. The Extensible Argonne Scheduling System (EASY) algorithm only guarantees that the first job in the queue will not be delayed, while other jobs might be starved. This algorithm uses an aggressive backfilling approach and is among the first introduced. It is still widely used in many HPC systems (Brevik et al., 2006).

HPC clusters are integrated into modern service models. Due to their computational capabilities, these systems have surged in fields such as big data and AI training for medium- and large-scale models (Park, 2019), driving major trends in HPC systems and energy consumption (Gaikwad et al., 2025). SLURM, PBS, LSF, and HTCondor are the most widely used batch schedulers for resource management. Efficient implementation of these schedulers is crucial for saving energy; therefore, the backfilling technique is important but not sufficient, as its performance is affected by users' job runtime estimates. Hence, backfilling alone or static heuristics-based optimization approaches might lead to lower accuracy for unpredictable job wait times (Gaikwad et al., 2025).

Hence, the use of Machine Learning (ML) as an intuitive solution for predicting job runtime for better backfilling performance is introduced (Kolker-Hicks et al., 2023). Still, in real HPC environments, jobs arrive in the system one after another. Hence, jobs either start immediately or wait in the schedule until sufficient resources become available. This will lead to a significant number of jobs being scheduled almost immediately, with negligible waiting time (zero or almost zero). In contrast, a considerable number of jobs need to wait for short or long (or very long) waiting times in the worst-case scenario. This environment inherently creates a zero-inflated learning problem. This triggers an initial decision: whether to wait or not at all and whether to use zero or non-zero, alongside predicting the estimated waiting time when the job waits in the schedule. In such a structure, a single trained regression model will struggle to perform well for both behaviors efficiently. Therefore, this paper proposes the hurdle model to separate the zero and non-zero values. Additionally, it proposes two phases of stacking rather than relying solely on single stacking regression. The first phase classifies values as zero or non-zero using a stacking classifier, while the second phase, for positive values only, applies stacking regression. Those two models were evaluated against a single Random Forest and stacking regression alone under strict and fair scheduling criteria.

The main contribution of this paper is predicting user-facing QoS metrics in HPC systems where scheduler-induced zero-inflation makes conventional regression inadequate. For this aim, the objective breakdown has been structured as follows:

1- This study proposes a Two-Stage approach based on a Hurdle model, which splits the prediction into classification and regression, alongside a Two-Stacking Stage model that enhances temporal robustness. This approach is essential since single-stage regression collapses sharply when there is a zero-inflated dataset.

2- The study presents a novel, systematic approach to three quality-of-service criteria across two radically different workloads in the HPC environment. We evaluate the efficiency of the proposed models using the targeted metric. The two-stage regression approach is superior when there are many zeros. However, its performance is more vulnerable to metrics that include few zeros or few positive values.

3- Rigorous time verification is applied using Time Block CV, which reveals that the classifier is stable with respect to time, whereas the regressor is more sensitive to temporal shifts. The Two-Stacking Stage model shows greater resilience of 63–96%.

4- The experiments demonstrate that datasets with high proportions of zero-inflated zeros are the result of an efficient scheduling policy rather than an incidental feature of a given dataset, which means that the need for a two-stage regression approach extends to any prediction scenario after efficient scheduling

The rest of this paper proceeds as follows: Section 1.2 presents the related work. Datasets, features selection, and temporal split are shown in Section 2. The proposed prediction models, including the Hurdle model, and Two-Stacking Stage model as well as their validation using Time-Block Cross-Validation are demonstrated in Section 2.4.5. Section 3 evaluates and discusses the performance of the proposed work via selected performance metrics. Finally, the paper concludes with possible research directions in Section 4.

1.2. Related Work

In HPC systems, the performance is significantly impacted by efficient job scheduling. HPC systems now provide heterogeneous resources more than ever as they

evolve and the scheduling decisions taken are now crucially affecting both system efficiency and user experience (Tirmazi et al., 2020; Borghesi et al., 2023). As diversity resources grow with workload variability, several studies have proposed different metrics (scheduling objective functions) to guide scheduling policies (Rudolph and Smith, 2000; Frachtenberg and Feitelson, 2005; Leung et al., 2010; Verma et al., 2014; Goponenko et al., 2022; Boëzennec et al., 2023; Boëzennec et al., 2024). These metrics can be categorized by system perspective or end-user perspective. The system performance metrics concern system utilization, throughput, and reducing job fragmentation, while end-user metrics focus on reducing waiting time, slowdown, and job response time. Classical approaches are simple to implement and have low computational complexity. Traditional scheduling policies favor a single objective with a straightforward approach. However, these algorithms cannot achieve a balanced performance in real system conditions with uncertainties (Wang et al., 2026).

Constraint Programming (CP) is naturally applied for solving scheduling problems on HPC systems. This type of programming involves providing variables and constraints, while the algorithm is responsible for finding a solution to the scheduling problem that satisfies those constraints and the available resources. As HPC systems consist of many powerful end supercomputers, known as "nodes", a software scheduler is required to manage the large number of incoming jobs and process them according to a policy. The scheduling problem is NP-hard (Du and Leung, 1989); hence, the scheduler applies approximate algorithms, such as backfilling algorithms (Jackson et al., 2001; Klusáček et al., 2008; Jette and Wickberg, 2023), which are swift but produce suboptimal schedules. The efficiency of backfilling is even worse in modern HPC systems with heterogeneous resources (Fan et al., 2019; Hovestadt et al., 2003), while the need for economic and environmental sustainability calls for efficient schedulers (Etinski et al., 2011; Ensmenger, 2018). Especially in closely related fields such as grid and cloud computing (Abraham et al., 2008; Tsai and Rodrigues, 2013; Houssein et al., 2021; Dakkak et al., 2015a), there is a strong demand for more thoughtful scheduling policies (Goponenko et al., 2024).

Scheduling in the HPC domain, like other domains, benefited from the promise and rise of AI. This is stated by Microsoft and the Pacific Northwest National Laboratory (PNNL) collaboration, which demonstrated that HPC empowered by AI can shorten the time required for material synthesis from weeks to years (Baker, 2024). Similarly, NVIDIA's latest HPC centers equipped with GPUs and AI models have pushed the boundaries of HPC beyond expectations (Pathak, 2025). With machine learning, resource utilization, job scheduling, load balancing, and simulation in HPC can be further optimized. This will be reflected in the system's overall efficiency, leading to better performance for users.

In Tanash et al. (2019), to address the overestimation of the resources required executing user jobs, a supervised machine learning model was integrated into the Slurm resource manager simulator via Constraint Programming (CP) dispatchers. The proposed model was trained to predict both memory and required computational time. In Bridi et al. (2016) proposed an approach to mitigate job allocation using a constraint satisfaction programming method for solving optimization problems. The study was evaluated on virtual machines for real and synthetic workloads, with different metrics. Still, with the most recent CP-based dispatchers (Bartolini et al., 2014; Borghesi et al., 2015), there are limitations in job-prediction time, which hinder adaptation in HPC systems. The work presented by Galleguillos et al. (2017) aimed to extract useful features from a workload using predictive models to improve scheduling decisions. Job runtime was their main focus. Therefore, they developed a method to predict job runtime using data available at submission time. Many cloud service providers allow users to access GPUs exclusively, which reduces system performance. This was a motivation factor for the work introduced in (Oh and Kim, 2020) by Oh and Kim. The proposed method adopted reinforcement learning to predict how long resources need to be occupied for a given application, using past logs as input and a deep reinforcement learning model (DQN). The key element here was predicting which jobs will not detract from the performance when executed simultaneously.

Chen et al. (2020), applied ensemble learning to predict HPC job runtime. In the base learner, five machine learning algorithms were used: Linear regression (LR), support vector regression (SVR), random forest regression (RFR), XGBoost, and LightGBM across three workloads. Evaluations revealed that LightGBM has low time complexity and high accuracy. Other methods imported from other domains, such as the topic modeling approach from natural language processing, have also been implemented in HPC scheduling by DeLucia and Baseman in DeLucia and Baseman (2018). The idea is to provide log messages to anticipate the HPC jobs starting for the final model prediction. Before predicting job states, data manipulation is applied to identify the correspondence between log records and syslog messages. In the following step, the latent Dirichlet allocation (LDA) method is used to extract features (Pathak, 2025).

2. Materials and Methods

This Section presents a description of the workloads implemented in this study. The zero-wait time in each workload, the temporal split, and the features that the models have been trained on.

2.1. Datasets

Both datasets used in this study share the same nine raw columns, derived from HPC simulation and planner output. Table 1 describes each field. Two independent datasets were used in this study, both of which represent

HPC job traces. Table 2 summarizes the key properties of each dataset. The META dataset contains 347,000 records with a moderate zero-inflation rate of 64.5%. On the other hand, the WAGAP dataset contains only 10,000 records with an extreme zero-inflation rate of 92.81%. Furthermore, the validation split of WAGAP contains only a single positive-wait example. This means that threshold calibration on this dataset is severely limited.

Both datasets were parsed with European decimal comma handling and validated for any missing or negative values. After that, the records were sorted by arrival time. The RAM field was found to be constant across all records and was therefore removed. It should be noted that no rows were deleted during cleaning. Moreover, sorting by arrival time was mandatory before applying feature engineering in order to prevent temporal leakage. Zero distribution in META and WAGAP workloads are presented in Figure 1.

2.2. Temporal Split

A strict 70/10/20 temporal partition was applied in chronological order. Table 3 shows the details of this split. It can be observed that the zero-wait rates vary across the different partitions, which reflects the temporal non-stationarity in these datasets. In the case of WAGAP, the validation set contains only one positive wait case. This nearly eliminates the possibility of meaningful threshold selection.

2.3. Features Engineering and Selections

A total of 23 features were constructed, grouped into five categories. All of these features are computed causally, which means that only information available at job arrival time is used. The categories are described below:

Job-Level (2 features): CPUs, walltime_limit.

Derived Temporal (4 features): arrival_rel, gap_length, cpu_intensity (CPUs/walltime_limit), work (CPUs × walltime_limit).

Global System-State (7 features): running_jobs, running_work, queued_jobs, queued_work, arrivals_last_3600s, arrivals_last_14400s, prev_wait_mean_100.

Queue-Local (6 features): q_running_jobs, q_running_work, q_queued_jobs, q_queued_work, q_arrivals_last_3600s, q_prev_wait_mean_100.

Pressure Ratios (3 features): ratio_q_jobs, ratio_q_work, ratio_q_work_queue.

Based on the feature importance of analysis, system-state features consistently ranked highest. The queue was one-hot encoded, the numeric features were median-imputed, and no scaling was applied since all the learners used in this study are tree-based.

Table 1. Raw data fields

Field	Description
giID	Job identifier (not used)
arrival	Job arrival time (simulation seconds)
wait	Job waiting time — prediction target (seconds)
runtime	Actual execution time (not used as feature)
CPUs	Number of requested processor cores
RAM	Requested memory — constant; removed
userID	User identifier (not used)
queue	Queue name (categorical feature)
walltime_limit	Maximum allowed wall-clock time

Table 2. Summary characteristics of META and WAGAP datasets after cleaning

Property	META	WAGAP
Raw records	347,000	10,000
Records after cleaning	347,000	10,000
Records removed	0	0
Raw columns	9	9
Unique queues	21	10
Zero-wait rate	64.50%	92.81%
Positive-wait rate	35.50%	7.19%
Mean waiting time (s)	10,583.77	171.32
Median waiting time (s)	0.00	0.00
Maximum waiting time (s)	235,228.20	69,469.00
Mean CPUs requested	2.82	9.08
Median CPUs requested	2.00	2.00
Maximum CPUs requested	125	256

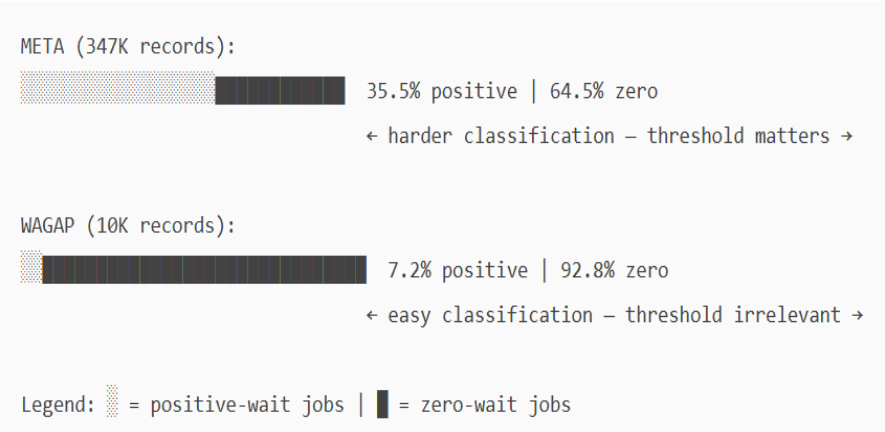


Figure 1. Zero distribution in META and WAGAP workloads.

Table 3. Temporal split summary

Partition	META Rows	Zero-Wait Rate	Positive Cases	WAGAP Rows	Zero-Wait Rate	Positive Cases
Train (70%)	242,899	64.14%	87,097	7,000	94.41%	391
Validation (10%)	34,701	52.89%	16,347	1,000	99.90%	1
Test (20%)	69,400	71.55%	19,744	2,000	83.65%	327

2.4. The Proposed Prediction Models

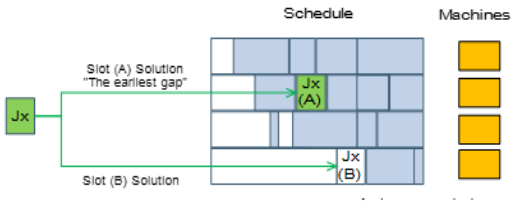
This Section introduces the proposed models in this study. The first proposed model is the Hurdle, which generally achieves the best possible performance. The two-stage stacking comes next, but with a notable performance advantage over the Hurdle model. The stand-alone Random Forrest is the worst. However, in terms of response time, this model's performance improves, since this metric, unlike wait-time, is not based on zero values. It is worth mentioning that the planner is based on the Best-Gap algorithm (Dakkak et al., 2015b; Klusáček et al., 2016;) and implemented in (Dakkak et al., 2021), which is the input to be trained for all the proposed models.

2.4.1. The planner

For all models proposed and implemented in this study, the Best-Gap algorithm was used as a planner. The Best-Gap algorithm is a filling algorithm that improves end-user QoS in the HPC environment. By QoS criteria, waiting time, slowdown, response time, and fairness are precisely the targets this algorithm aims to improve. The algorithm uses a planning approach (unlike queues). This data structure enables it to plan a place for the incoming job based on runtime estimates provided by the user (if available). The algorithm checks each incoming job to see if there is a suitable gap for it. Once a suitable gap is available, the job will be allocated to that machine. Then, the algorithm continues searching for another gap. Once another gap is detected, a job move evaluation will be calculated using the weight function. If the new gap is considered a better solution than the current one, the job will be moved; if not, it will remain in its current gap. This loop continues for all arrival jobs. The calculated weight function that evaluates the move is the sum of the weights for waiting time, slowdown, response time, and fairness. Algorithm 1 presents a summary of the steps applied, while Figure 2 shows how the planner keeps/moves the job.

Algorithm 1 Planner working steps

1. for each machine $m_i \in M$ do
2. if m_i is suitable and has a gap for J then
3. place J in gap, compute: score = $\Sigma\, w_k \cdot (prev_k - cur_k)/prev_k$
4. if score > 0 or first assignment then accept placement
5. else reject and restore previous best placement;
6. return best machine



$prev_{fair} = 0.8, prev_{rt} = 200, prev_{wt} = 100, prev_{sd} = 5.0$

$cur_{fair} = 0.7, cur_{rt} = 180, cur_{wt} = 80, cur_{sd} = 5.5$

$diff_{fair} = \frac{0.8 - 0.7}{0.8} = +0.125, diff_{rt} = \frac{200 - 180}{200} = +0.10$

$diff_{wt} = \frac{100 - 80}{100} = +0.20, diff_{sd} = \frac{5.0 - 5.5}{5.0} = -0.10$

$score = (2 \times +0.125) + (1 \times +0.10) + (1 \times +0.20) + (1 \times -0.10) = +0.45$

Accept placement, update best machine.

$prev_{fair} = 0.8, prev_{rt} = 200, prev_{wt} = 100, prev_{sd} = 5.0$

$cur_{fair} = 0.9, cur_{rt} = 210, cur_{wt} = 95, cur_{sd} = 5.8$

$diff_{fair} = \frac{0.8 - 0.9}{0.8} = -0.125, diff_{rt} = \frac{200 - 210}{200} = -0.05$

$diff_{wt} = \frac{100 - 95}{100} = +0.05, diff_{sd} = \frac{5.0 - 5.8}{5.0} = -0.16$

$score = (2 \times -0.125) + (1 \times -0.05) + (1 \times +0.05) + (1 \times -0.16) = -0.41$

Reject placement, restore previous best.

Figure 2. Planner working approach.

2.4.2. The proposed hurdle model

The Hurdle Model is a two-stage predictive architecture that was designed specifically to handle zero-inflated targets. Instead of relying on a single model to learn both whether a job will wait and how long it will wait, the Hurdle approach separates these into two distinct prediction problems. In HPC environments, a job either starts immediately with wait = 0 if the required resources are available at arrival, or it enters the queue with wait > 0. These two situations are fundamentally different, and different factors govern them. The first depends on resource availability at the time of arrival, whereas the second depends on queue dynamics, backlog, and scheduling priorities. The main limitation of

applying direct regression in zero-inflated settings is that the model has to learn two very different behaviors simultaneously from a single optimization objective. Since 64–72% of training targets are exactly zero in META workload and 94% in WAGAP workload, the model tends to learn to predict near-zero values for most inputs. This suppresses the sensitivity to the genuine positive wait signal. This issue is considered a structural mismatch rather than a tuning failure, and the results presented in this study confirm this observation.

The first stage of the Hurdle Model consists of a LightGBM binary classifier. This classifier outputs $p_0(x)$, which represents the probability that wait equals zero. To address class imbalance, the scale_pos_weight parameter is used to correct it by weighing the minority class proportionally. Table 4 presents the hyperparameters that are used for this classifier.

Table 4. LightGBM classifier hyperparameters (Hurdle Stage 1)

Parameter	Value	Rationale
n_estimators	800	Sufficient trees for convergence
learning_rate	0.03	Conservative rate for stable gradient descent
num_leaves	127	Moderate complexity; controls overfitting
subsample	0.90	Row subsampling for variance reduction
colsample_bytree	0.90	Feature subsampling per tree
scale_pos_weight	Computed	Corrects for class imbalance
random_state	42	Reproducibility

The second stage estimates the positive waiting time. A LightGBM regressor is trained exclusively on jobs where > 0 wait. The target variable is transformed using $\log_{1+}$(wait) and reverse-transformed via expm1 before evaluation. The purpose of this log transformation is to compress the distribution's heavy tail into a more manageable range. Table 5 lists the hyperparameters for this regressor.

Table 5. LightGBM regressor hyperparameters (Hurdle Stage 2)

Parameter	Value
n_estimators	2000
learning_rate	0.03
num_leaves	255
subsample	0.90
colsample_bytree	0.90
random_state	42

The decision rule works as follows: if $p_0(x) > \tau$, then predicted_wait is set to 0; otherwise, predicted_wait is computed as expm1(regressor_prediction). In order to select the best threshold, two thresholds were identified through grid search over τ [0.30, 0.95]. The first threshold is θ_FP, which aims to minimise false-positive zero predictions. The second threshold is θ_MAE, which aims to minimise MAE_all on the validation set. It should be noted that on WAGAP, both thresholds collapsed to 0.30 since the validation set contained only one positive-wait sample. This means that meaningful threshold selection was not possible in this case.

Algorithm 2 works as follows:

Algorithm 2 The proposed Hurdle approach

1. split D into $D_0 = \{(x, wait) \in D : wait = 0\}$
2. $D_+ = \{(x, wait) \in D : wait > 0\}$
3. stage 1 — train LightGBM classifier on D
4. Label: y = 1 if wait = 0, else y = 0
5. set scale_pos_weight = $|D_0| / |D_+|$
6. Output: $p_0(x) = P(wait = 0 \mid x)$
7. stage 2 — Train LightGBM Regressor on D_+
8. Target: $\log_{1+}$(wait)
9. calibrate threshold τ via grid search over [0.30, 0.95]:
10. θ_FP = argmin FP_nonzero_as_zero(τ, validation)
11. θ_MAE = argmin MAE_all(τ, validation)
12. compute $p_0(x)$ using stage 1 classifier
13. if $p_0(x) > \tau$ THEN
14. predicted_wait = 0
15. else
16. predicted_wait = expm1(stage 2 regressor(x))
17. derive scheduling metrics:
18 response_time = predicted_wait + runtime
19. slowdown = response_time / max(runtime, 10)
20. return predicted_wait, response_time, slowdown

Lines 1-2: The dataset will be split into two groups based on whether the waiting time is positive or zero. Lines 3-6: A LightGBM Classifier is applied to determine whether the job will wait or be processed immediately. Lines 7-8: LightGBM Classifier will be trained to predict the waiting time for jobs with positive waits. Lines 9-11: A grid search is implemented to determine the optimal threshold for minimizing false positives and MAE. Lines 12-16: performing wait time prediction for an incoming job, if $po(x) \leq \tau$, that regressor will be applied to find the predicted wait time value; otherwise, the job's predicted wait time will be set to zero. Lines 17-20: based on the predicted wait, derive the slowdown and response time, and return their predicted values.

2.4.3. The proposed two-stacking stage model

The stacking architecture follows a similar structure to the Hurdle Model. However, each stage is replaced with an ensemble of models. In the classification stage, three base classifiers are used, namely LightGBM, Random Forest, and Logistic Regression. These classifiers produce out-of-fold probability predictions through a 5-fold TimeSeriesSplit. The generated predictions form the meta-feature matrix, which is fed into a Logistic Regression meta-classifier. Similarly, for the regression stage, an analogous three-model regression stack is applied (LightGBM, Random Forest, and Ridge). These base models are trained on $\log_{1+}$(wait) for positive-wait cases only, and a Ridge meta-regressor then combines the outputs. During inference, the classifier meta-learner

produces p_0. If $p_0 > \theta$, then the predicted wait is set to 0, and the regressor meta-learner provides the estimate. The same threshold criteria (θ_FP and θ_MAE) are applied here as well. Moreover, in the case of WAGAP, an additional numerical safety clip was applied before expm1 in order to prevent overflow issues.

Algorithm 3 works as follows: Lines 1-6: a stacking classifier is applied using three classifiers (LGBM, RF, LogReg). The classifier (base learner) here predicts whether the wait is zero or not; the meta-learner (Logistic Regression) sums the base learner's predictions and provides the final prediction of the waiting time (zero or not). , Lines 7-10: Time block series is applied over the positive wait time values only, then the regression stage starts with three regression learners (LGBM, RF, Ridge Regression), then the meta regressor (Ridge Regression) sums the predictions from the base learner and provides the final prediction of the waiting time. Lines 11-12: predicting if the job is going to wait or not, while in Lines 13-14: predicting how long the job is going to wait. Line: 15-18: if p0 is greater than the threshold, then wait is 0, otherwise wait is not zero. Threshold calibration results are presents in Table 6. Also, flowchart for the proposed Hurdle model, conceptual model of the Hurdle model and flowchart for the proposed two stacking stage model are presents in Figures 3, 4, and 5 respectively.

Algorithm 3 The proposed Two-Stacking Stage approach

```
1.  y_cls = (y == 0)
2.  x_pos, y_pos = x[y > 0], log1p(y[y > 0])
3.   for each fold in 5-Fold TimeSeriesSplit:
4.    for i, clf in [LGBM, RF, LogReg]:
5.     meta_cls[val, i] = clf.fit(xtr, y_cls_tr).predict_proba(xval)
6.   MetaCLF = LogReg.fit(meta_cls, y_cls)
7.   for each fold in 5-Fold TimeSeriesSplit:
8.    for i, reg in [LGBM, RF, Ridge]:
9.     meta_reg[val, i] = reg.fit(xtr_pos, y_pos_tr).predict(xval_pos)
10.  MetaREG = Ridge.fit(meta_reg, y_pos)
11. cls_meta = [clf.predict_proba(x_new) for clf in base_clfs]
12. p0 = MetaCLF.predict_proba(cls_meta)
13. reg_meta = [reg.predict(x_new) for reg in base_regs]
14. log_wait = MetaREG.predict(reg_meta)
15. if p0 > θ:
16.  wait = 0
17. else:
18.  wait = expm1(clip(log_wait, SAFE_MAX)
```

Table 6. Threshold calibration results

Threshold	Selection Criterion	META Value	WAGAP Value
θ_FP	Minimize FP_nonzero_as_zero	0.95	0.30
θ_MAE	Minimize MAE_all on validation	0.30	0.30

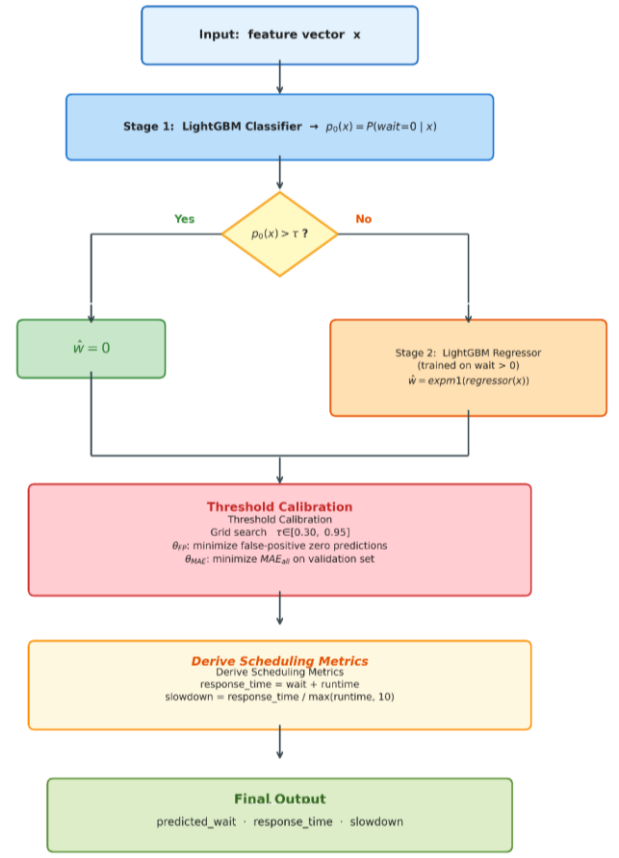


Figure 3. Flowchart for the proposed Hurdle model.

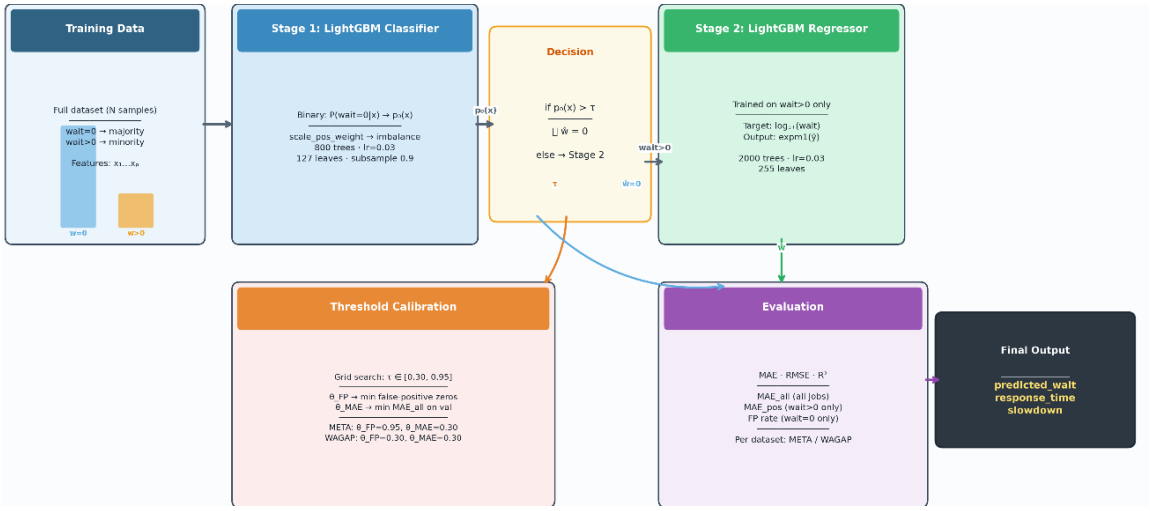


Figure 4. Conceptual model of the Hurdle model.

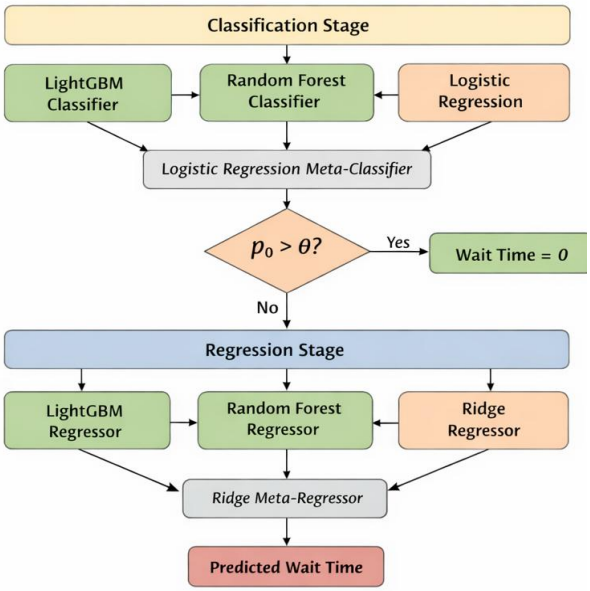


Figure 5. Flowchart for the proposed two stacking stage model.

2.4.4. Random forest solo model

The standalone Random Forest is trained directly on raw wait values, including zeros, without any decomposition. Negative outputs are clipped to zero. No log transformation or threshold tuning is applied in this model. The main purpose of including this model in the evaluation is to quantify the cost of ignoring the data's zero-inflated structure. Table 7 presents the hyperparameters. Algorithm 4 works as follows: Lines (1-2): for all jobs, the model is trained on the features described in Section 3.3. In lines (3-4), 300 trees are established with unlimited depth. The model will be trained based on the initial raw values in lines (5-6). In lines (7-8), for every incoming job, all 300 trees will predict, and the average of these predictions will be used. If the prediction value is negative, the result will be zero in line 9. The final value for every job will be returned in line 10. Conceptual model of the RF solo model are presents in Figure 6.

Table 7. Standalone RF hyperparameters

Parameter	Value
n_estimators	300
max_depth	None (unlimited)
min_samples_leaf	2
n_jobs	-1 (all cores)
random_state	42

Algorithm 4 Solo Random Forrest working steps

1. training set D_train = {(x_i, w_i)}
2. test set D_test = {(x_j)}
3. initialize RF model with:
4. n_estimators: 300, max_depth: None, min_samples_leaf: 2, n_jobs: -1, random_state: 42
5. train RF on raw wait values:
6. RF.fit (x_train, w_train)
7. for each test job x_j in D_test:
8. $\hat{w}_j \leftarrow$ RF.predict(x_j)
9. $\hat{w}_j \leftarrow \max(\hat{w}_j, 0)$
10. return $\hat{w}_j$

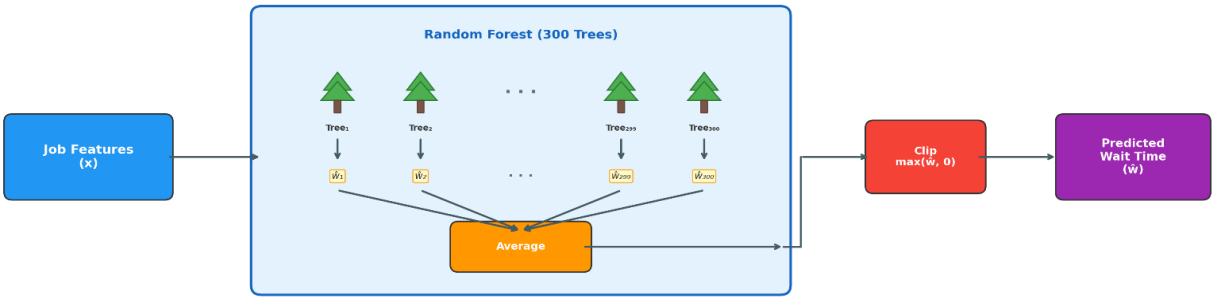


Figure 6. Conceptual model of the RF solo model.

2.4.5. Validation strategy

To properly evaluate the models, Time Block Cross-Validation with a 5-fold TimeSeriesSplit was used on the training set. The base learners were trained on the earlier temporal portion, and their predictions on the later portion formed the out-of-fold data that was used for meta-layer training. This approach prevents any form of temporal leakage. In order to assess the stability of both models across temporal periods, 5-fold Time Block CV using TimeSeriesSplit was conducted. Random-k-fold CV causes serious issues for data with a specific time order (arrival times in the META and WAGAP case) because this method allows the model to train on feature data and test on previous data, leading to temporal leakage and an outstanding but unreal performance. Time Block CV splits the data into ordered blocks rather than arbitrary splitting, leading to transparent performance for the model when time progresses (Doll et al., 2026; Roberts et al., 2017). Time block cross-validation (5-fold Expanding Window) is presents in Figure 7.

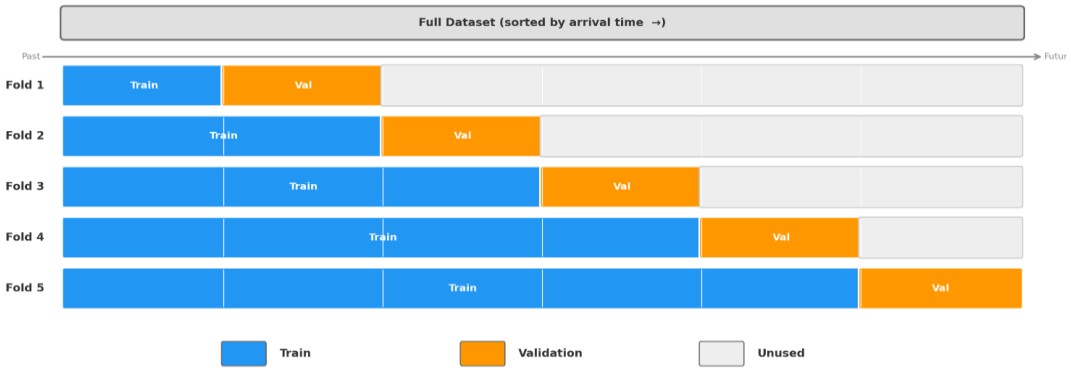


Figure 7. Time block cross-validation (5-fold Expanding Window).

2.4.5.1. Validation via time block cv (Hurdle model)

It can be observed that the ROC-AUC$_0$ ranges from 0.9846 to 0.9947 with a mean of 0.9906, confirming consistent zero/non-zero discrimination across all folds (Table 8). Fold 4 appears as an outlier with MAE_pos = 54,004.88 s, which is due to a distributional shift in this temporal window. On the other hand, Fold 3 achieves R^2_pos = 0.701, which is the best result in the Hurdle CV (Table 9, 10 and 11).

2.4.5.2. Validation via time block cv (two-stacking stage)

The Stacking classifier is equally stable with a mean ROC-AUC of 0.9910. Furthermore, Fold 3 achieves R^2_pos = 0.778, which is the highest value obtained in any CV fold across all models (Table 12 and 13). Time Block CV: Hurdle vs. Two Stacking Stages (META) are present in Figure 8.

Table 8. Hurdle classifier Time Block CV (META dataset)

Fold	Train rows	Val rows	Zero%	ROC-AUC$_0$	PR-AUC$_0$	Pos. train count
1	40,484	40,483	87.09%	0.9846	0.9975	6,755
2	80,967	40,483	58.55%	0.9932	0.9951	11,980
3	121,450	40,483	59.17%	0.9915	0.9941	28,761
4	161,933	40,483	33.19%	0.9889	0.9754	45,291
5	202,416	40,483	63.55%	0.9947	0.9960	72,339

Table 9. Hurdle regressor Time Block CV (META dataset)

Fold	Pos. train count	Pos. val count	MAE_pos (s)	R^2_pos
1	6,755	5,225	4,678.59	0.290
2	11,980	16,781	10,050.08	0.241
3	28,761	16,530	5,670.96	0.701
4	45,291	27,048	54,004.88	−0.071
5	72,339	14,758	6,899.13	0.371

Table 10. Hurdle Time Block CV (WAGAP dataset)

Fold	Pos. train	ROC-AUC$_0$ (clf)	MAE_pos (reg, s)	R^2_pos (reg)	Status
1	1	0.500	0.10	0.000	Insufficient positives
2	9	0.407	324.71	−0.088	OK
3	22	0.137	2,349.83	−7.625	OK
4	31	0.868	8,654.46	−1.215	OK
5	50	0.573	2,139.36	−0.455	OK

Table 11. Two-Stacking Stage classifier Time Block CV (META dataset)

Fold	Train rows	Val rows	Zero%	ROC-AUC (meta-clf)	PR-AUC (meta-clf)
1	40,484	40,483	87.09%	0.9892	0.9981
2	80,967	40,483	58.55%	0.9940	0.9956
3	121,450	40,483	59.17%	0.9923	0.9937
4	161,933	40,483	33.19%	0.9847	0.9676
5	202,416	40,483	63.55%	0.9948	0.9963

Table 12. Two-Stacking Stage Regressor Time Block CV (META dataset)

Fold	Pos. train	Pos. val	MAE_pos (meta-reg, s)	R^2_pos (meta-reg)
1	14,517	14,516	10,403.95	0.431
2	29,033	14,516	5,596.66	0.557
3	43,549	14,516	16,523.32	0.778
4	58,065	14,516	20,132.01	0.617
5	72,581	14,516	7,792.39	−0.193

Table 13. Two-Stacking Stage Time Block CV (WAGAP dataset)

Fold	ROC-AUC (meta-clf)	PR-AUC (meta-clf)	MAE_pos (meta-reg, s)	R^2_pos (meta-reg)
1	0.405	0.994	1,197.00	0.036
2	0.842	0.998	500.32	−0.045
3	0.850	0.998	843.31	0.478
4	0.885	0.996	336.89	0.215
5	0.888	0.954	712.70	0.873

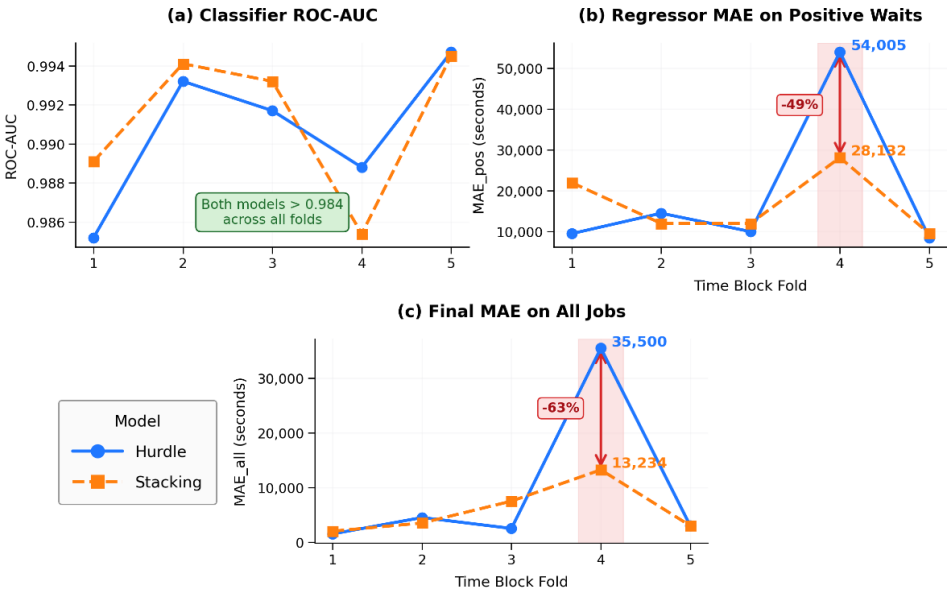


Figure 8. Time Block CV: Hurdle vs. Two Stacking Stages (META).

The Time Block CV on the META workload shows that the components of the proposed models have responded differently. The classifier has shown high stability for all five folds. The ROC-AUC remained between 0.985 and 0.995 for both the Hurdle and Two-Stacking Stages models. This means that the separation between jobs that have to wait (zero) and those that do not (non-zero) is stable and insensitive to time. At the same time, the regressor component has shown high sensitivity, especially in the fourth fold. For instance, in the Hurdle model, the error has registered 54000 seconds compared to 5000-10000 seconds in the other remaining folds. This high jump in the fold indicates that the distribution shift in the workload occurred at that time, or that the resources allocated to the jobs in the cluster changed. In the fifth fold, the performance returns to normal, which proves that the model itself is stable. The most remarkable observation is that the Two-Stacking Stages model has 63% less error in the same (fourth) fold than the Hurdle model (20000 seconds). This backs up the meta-learner's ability to adjust and absorb this large distribution shift in the data. However, the performance of the Two-Stacking Stages model in the third fold was not as good as the Hurdle model, and these points to the fact that improvement in the fourth fold compared to the Hurdle model might be at the expense of other folds.

WAGAP workload (in Appendix A) was more challenging for Hurdle since it consists of high number of zero values (almost 94%). The ROC-AUC values are 0.14-0.87 and this is due to few positive wait values in this workload which makes it difficult to learn due to the lack of jobs that have waited in this workload. The Two-Stacking Stages model has shown more stability, especially after the second fold (0.84-0.89) and this again returns to the meta-learner, which has proven that it can deal with severe lack of positive values. The regressor performance in Hurdle model follows the same behavior as META. In the fourth fold, the values increased to 8654 seconds during the fourth fold, whereas the Two-Stacking Stages model has shown better performance by 96% (337 seconds).

The conducted analysis over Time Block CV confirms that the Hurdle model is completely stable for workloads that include (64%-72%) zero values, such as the META workload. While in the WAGAP workload, which includes 94% of zero waiting time, the performance has struggled due to a lack of positive values. While the Two-Stacking Stages model has shown greater dynamicity in both datasets, indicating that the meta-learner adds robustness against temporal splits. The deterioration in Time Block CV for WAGAP workload is due to a lack of positive waits (not more than 78 values in the odd folds), not to a model performance issue. The lack of positive values in this workload poses a challenge for any proposed model, even for the Two-Stacking Stages model. The evidence for the success of the Hurdle model is the result of the test set (will be presented in Section 5.2) that has a total of 327 positive cases, as shown in Table 15, where the Hurdle outperforms all other models, especially the RF solo model, which performs the second best in this case.

3. Results and Discussion

This Section presents the results on META and WAGAP for waiting time, response time (RT = Wait + Runtime), and slowdown (Slowdown = RT/Runtime) across three models. Since RT and slowdown are linear transformations of wait, the MAE for RT equals the MAE for wait. Furthermore, analysis and discussion are introduced.

3.1. Waiting Time Evaluation

The Hurdle model remains the best on WAGAP with MAE_all = 303.52 s, outperforming the Stacking by 27.7% and the RF Solo by 19.4%. It should be noted that all Hurdle threshold variants produce identical results. This is because the validation set contained only one positive-

wait sample, which means that threshold calibration had no effect. Nevertheless, the classifier remains highly capable with ROC-AUC$_0$ = 0.9891. R^2_pos remains negative for all models on WAGAP as well, which is consistent with the META results.

On the META workload (Figure 9), the Hurdle achieved the best performance with (MAE_all ≈ 350–500 depending on the selected threshold). The Hurdle best performance was in thr_FP (MAE_all ≈ 350, R^2 ≈ −0.15). Two-Stacking Stage also achieved good results (MAE_all ≈ 350–600), hitting its best performance when thr_MAE (MAE_all ≈ 350). Both models have shown relatively robust and close performance (MAE_pos ≈ 2500–2900) when the wait is greater than zero (positive waits), indicating that they have nearly the same accuracy in predicting waiting time. The difference here depends on the classifier's accuracy in each model. RF Solo performance is the worst one (MAE_all ≈ 9.500, MAE_pos ≈ 21,000، R^2 ≈ −22), which means that the prediction means by 22 times. This confirms that single-stage regression cannot handle zero-inflated datasets (64–72% zeros). RF Solo aims to find a middle ground between zeros and large values, but it fails in both cases, whereas two-stage regression, applied in the Hurdle and Two-Stacking Stage models, solves this issue structurally. The classifier handles zeros, while regression focuses on positive values. R^2_all is negative in all models (from -0.15 to -0.50 for Hurdle and Two-Stacking Stage), reflecting the challenge of prediction in HPC and keeping this an open challenge due to high variance in the values and sensitivity to temporal changes. However, these values are close to zero, and that shows two-stage regression is close to the mean, which is quite crucial in significance compared to -22 in RF solo.

In the WAGAP workload (Figure 10), a remarkable R^2_all case shows up. This is the only case in which R^2_all is positive across all models. The Hurdle has 0.195, the Two-Stacking Stage has 0.137- 0.130, and the RF Solo has 0.130. This means that all models are indeed better than the mean prediction, which occurs only once in the conducted experiments. The justification stems from the nature of WAGAP. With dominant zero waiting time values (almost 94% of the workload), the model that predicts a wait time of zero will be right most of the time. Hurdle and Two-Stacking Stage models are perfectly getting the advantage here. The classifier classifies most of the job waiting time as zeros (which is true), and the errors are limited to a few positive values, keeping the total error low. The Hurdle model (MAE_all ≈ 305، R^2 = 0.195) outperforms Two-Stacking Stage (MAE_all ≈ 420–435، R^2 ≈ 0.130–0.137) and RF solo (MAE_all ≈ 380، R^2 = 0.130). RF Solo performance was not deteriorated, unlike META. In fact, its performance was even close to the Two-Stacking Stage. With 94% of zero waiting times, even a simple model such as RF solo can learn that the usual prediction is zero. However, on positive waiting time values (MAE_pos ≈ 1800–1900 for all models), the differences almost disappear, which highlights the challenge of wait time prediction with 391 positive values only. The negative R^2 values for positive-wait prediction are caused due to the heavy-tailed and zero-inflated data in these workloads. The distribution is really challenging to get a positive R^2. When we have thousands of jobs with zero waiting time, then suddenly we have a job that might wait for a significant time. This will result in a negative R^2 value. The hurdle model and the two stacking stages were close to the baseline.

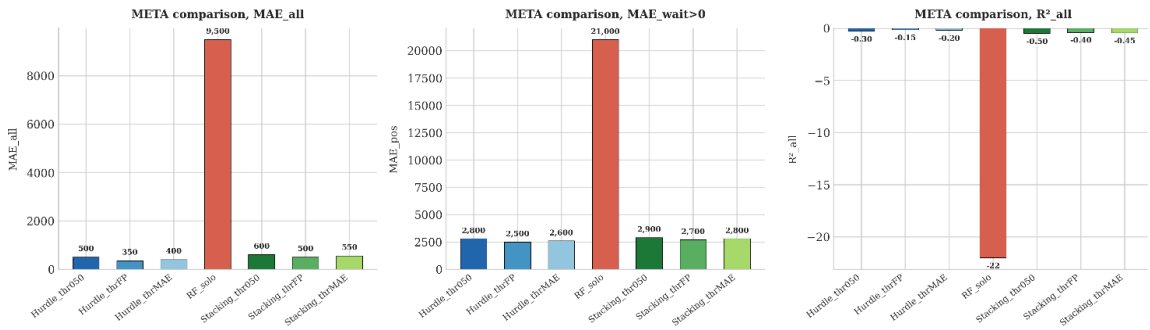


Figure 9. Waiting Time Evaluation (META).

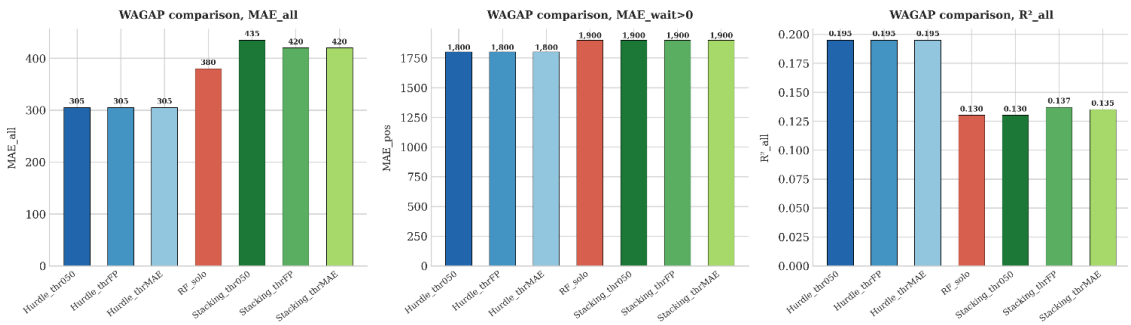


Figure 10. Waiting Time Evaluation (WAGAP).

3.2. Response Time Evaluation

Response time is the sum of the job's waiting time and running time. It represents the total time the user spent sending the job until completion. Unlike waiting time, this metric cannot be zero since at least the response time will be equal to the job runtime, even when the job has zero waiting time.

In the META workload (Figure 11), the Hurdle still outperforms all the other models (MAE_all ≈ 750–900), leading the Two-Stage Staking (MAE_all ≈ 1300), while RF solo has totally collapsed (MAE_all ≈ 8,000، R^2 ≈ −22). This downgrade in RF performance proves that one stage is not enough most of the time, even for response time. Although the intense zero values are absent from this metric, the Hurdle model performance achieved the best results.

The interesting case is in the WAGAP workload (Figure 12). The performance of the three models has been shifted. RF has turned the tables by achieving better performance (MAE_all ≈ 250) than the Two-Staking Stage (MAE_all ≈ 1310–1350) and the Hurdle model (MAE_all ≈ 310). This is due to the lack of positive values (391 out of 10000) in this load, given that the metric cannot be zero in the first place. In this particular case, the RF solo model takes advantage of its simplicity, while the two stages applied in Hurdle and Two-Staking Stage add more complexity without a significant positive impact on the performance in this case.

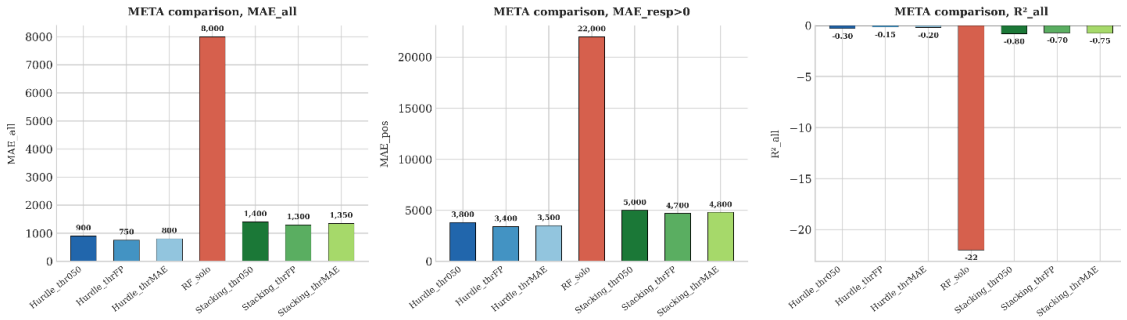


Figure 11. Response Time Evaluation (META).

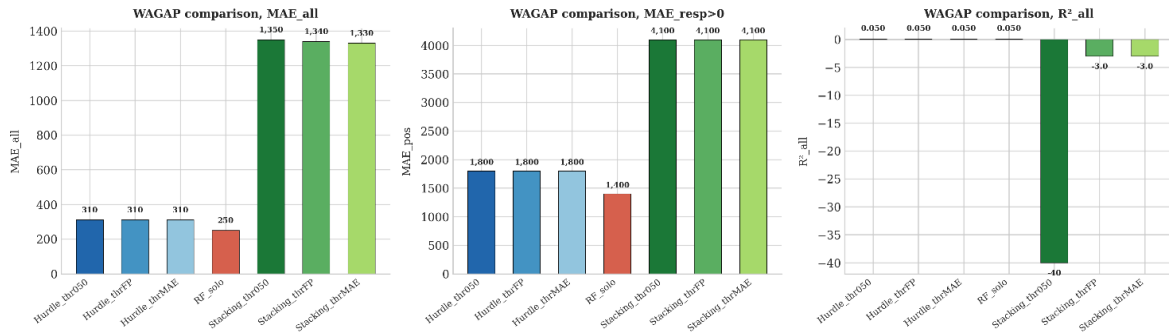


Figure 12. Response Time Evaluation (WAGAP).

3.3. Slowdown Evaluation

The slowdown (computed as 1 + Wait/Runtime). If the slowdown is 1.0, that means the job was executed immediately without waiting. The more the slowdown, meaning the longer the job has waited relative to the job volume. This metric is crucial because it reflects the relative delay for the job. For instance, if a job runtime is 10 seconds and it waited 100 seconds, it means the slowdown in this case is 11.0, which is much worse compared to the same waiting time (100 seconds) compared to a job that has 10.000 seconds as runtime (slowdown = 1.01).

In the META workload (Figure 13), the experiments have shown that the Hurdle model is the best (MAE_all ≈ 8–15) with a slight advantage compared to the Two-Stacking Stage (MAE_all ≈ 10–15), while RF solo is the worst MAE_all ≈ 850, R^2 ≈ −150). This is back to the nature of the metric: short jobs with long waiting times produce high slowdown values, and the simplicity of the RF model is unable to handle contrast. For jobs with a slowdown greater than 1 (slowdown > 1.0), the difference becomes more visible. Hurdle achieves (MAE_pos ≈ 20–30), while Two-Staking Stage achieves (MAE_pos ≈ 25–35). Nevertheless, the RF solo jumps to 2600, while the R^2_all is negative across all models, but it is close to zero for the Hurdle and Two-Staking Stage models, indicating it is close to the average. However, it is at -150 for RF Solo, indicating that the coefficient of determination is 150 times worse than the average. These findings demonstrate that the le-Stage Regression technique is not just a fancy design with marginal improvement, but an essential need when dealing with unusual metrics such as slowdown. The slightly better performance of the Hurdle model over the Two-Staging Stage model is consistent with response-time findings on the META workload, which indicates that the Hurdle model performs better than the overly complex Two-Staging Stage model when there are enough positive wait

values in the dataset. In the WAGAP workload (Figure 14), the same general pattern appeared. Both Hurdle and Two-Stacking Stage models have almost a tie (MAE_all ≈ 8.0, 7.0-8.0) respectively, while RF solo is much worse (MAE_all ≈ 35). However, the interesting thing here is that the Two-Stacking Stage is slightly better than Hurdle, unlike the case in the META workload, especially for positive values (MAE_pos ≈ 25 vs 50 for Hurdle). This aligns with the previous analysis conducted on the Time Block CV, whereas the Two-Stacking Stage showed greater stability on the WAGAP workload, but the Hurdle achieved better performance on the final test set. For R^2_all, it is negative for all models. This is quite anticipated in such a dataset that lacks positive values (391 values).

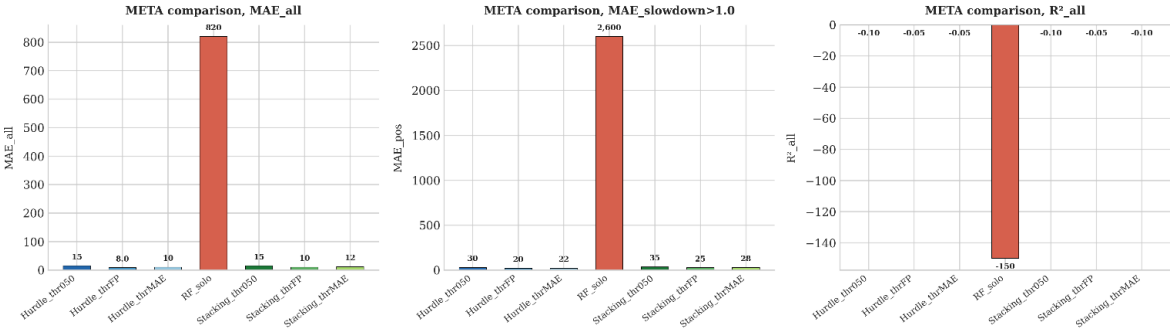


Figure 13. Slowdown Evaluation (META).

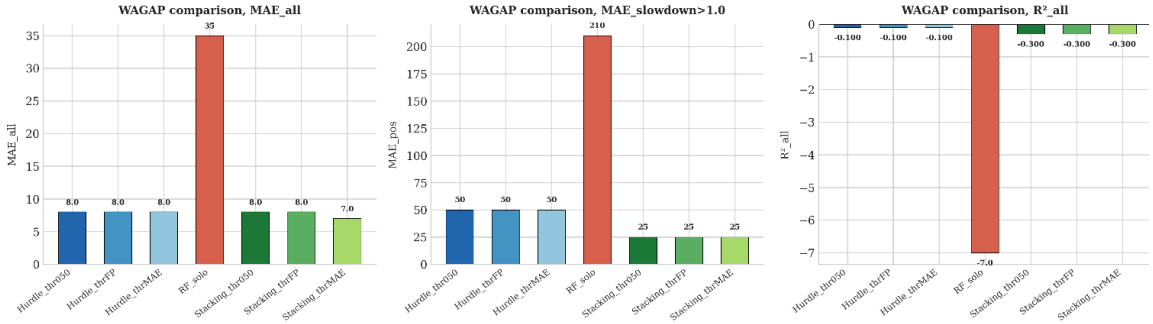


Figure 14. Slowdown Evaluation (WAGAP).

3.4. Further Discussion

Table 14 reports the primary evaluation metrics on the META test set, which consists of 69,400 jobs representing the last 20% chronologically. It can be observed that the Hurdle model with θ_MAE = 0.30 achieves the best overall performance on META, recording MAE_all = 994.47 s and R^2_all = 0.370. This means that on average, the predicted waiting time deviates by fewer than 17 minutes from the actual value. The Two-Stacking Stage model is closely competitive with MAE_all = 1,011.22 s, which is 1.67% higher. The standalone RF performs dramatically worse, with MAE_all = 9,096.95 s, an 815% increase over the Hurdle model. Furthermore, R^2_all = −21.86, which indicates that the predictions are far worse than simply using the training-set mean. This is a structural failure caused by the data's zero-inflated nature. The classifier quality in both two-stage models is exceptional. The ROC-AUC_0 is 0.9951 for the Hurdle and 0.9947 for the Stacking, with PR-AUC_0 ≥ 0.997 in both cases. However, R^2_pos remains negative across all configurations, indicating that the regression stage still struggles to explain variance among positive-wait jobs. Across both datasets, the Hurdle model achieves the lowest MAE_all. On META, it shows an 89.1% reduction compared to the RF Solo and 1.7% compared to the Stacking. On WAGAP, the reduction is 19.4% compared to RF Solo and 27.7% compared to the Stacking. On META, the Two-Stacking Stage model is competitive, within 1.7% of the Hurdle. However, on WAGAP, it degrades and performs worse than even the RF Solo. This suggests that the added complexity of the Stacking architecture becomes a liability when insufficient data is available for training. R^2_pos remains negative on the test sets for all models. This indicates that the system-state features capture aggregate congestion but are insufficient to model the scheduler's fine-grained priority logic. Therefore, improving the positive-wait prediction is considered the most important direction for future work. The zero-wait rate varies significantly across temporal windows. In META, it ranges from 33% to 87%, whereas in WAGAP, it ranges from under 70% to over 99%. This level of non-stationarity makes the prediction task more challenging.

Table 15 reports the evaluation metrics on the WAGAP test set, which consists of 2,000 jobs with a zero-wait rate of 83.65%.

Table 14. Test-set performance on META (69,400 jobs)

Model	θ	MAE_all (s)	R^2 (all)	MAE_pos (s)	R^2 (pos)	MAE_tail top-10% (s)	ROC-AUC_0	PR-AUC_0
Hurdle (θ_MAE)	0.30	994.47	0.3696	3,406.09	−0.077	8,582.06	0.9951	0.9976
Hurdle (θ=0.50)	0.50	1,002.44	0.3690	3,404.99	−0.076	8,570.10	0.9951	0.9976
Hurdle (θ_FP)	0.95	1,038.69	0.3643	3,403.73	−0.075	8,565.46	0.9951	0.9976
Two-Stacking Stage (θ_MAE)	0.55	1,011.22	0.3490	3,494.23	−0.117	8,523.63	0.9947	0.9970
Two-Stacking Stage (θ=0.50)	0.50	1,010.73	0.3488	3,494.73	−0.117	8,524.37	0.9947	0.9970
RF Solo	—	9,096.95	−21.858	21,302.52	−30.517	26,890.80	—	—

Table 15. Test-set performance on WAGAP (2,000 jobs)

Model	θ	MAE_all (s)	R^2 (all)	MAE_pos (s)	R^2 (pos)	MAE_tail top-10% (s)	ROC-AUC_0	PR-AUC_0
Hurdle (all θ)	0.30	303.52	0.1946	1,802.08	−0.107	9,970.50	0.9891	0.9974
Two-Stacking Stage (θ_MAE)	0.30	419.91	0.1361	1,912.79	−0.150	10,280.92	0.9851	0.9968
Two-Stacking Stage (θ=0.50)	0.50	433.84	0.1292	1,910.89	−0.150	10,280.92	0.9851	0.9968
RF Solo	—	376.43	0.1281	1,927.58	−0.101	9,996.37	—	—

Based on the results, the Hurdle Model with θ = 0.30 is recommended for general use. However, when false-zero predictions carry a high cost, θ = 0.95 should be used instead. The standalone RF should not be used for zero-inflated HPC prediction since it fails structurally. Moreover, regular retraining on rolling temporal windows is recommended to address non-stationarity. More detailed results are provided in the Appendices.

4. Conclusion

This study has introduced the Two-Stacking Stage approach by proposing the Hurdle model and the Two-Stacking Stage model for predicting QoS from the end-user perspective (waiting time, response time, and slowdown) for HPC environment. This study emerged after the backfilling scheduler BestGap was simulated on real workloads (META and WAGAP). The simulation resulted in a dataset with a plethora of zero waiting time values, since the scheduler finds suitable gaps immediately most of the time. These zero-inflated data are not a flaw or noise in the original datasets, but rather a reflection of BestGap's good scheduling performance. This leads to the fact that single-stage regression structurally is not sufficient for this study. To overcome this, a two-stage regression is required to determine the job's positive waiting times.

The experiments have been conducted over two real datasets that have different attributes in terms of job records (347000 jobs for META) and (10000 recorded jobs for WAGAP), and then zero inflated data with 64%-72% in META and 94% - 99% in WAGAP. The study has revealed that the double-stage regression approach is a structural necessity, not merely a marginal improvement. Direct single-stage regression (RF solo) collapsed catastrophically on both waiting time (R^2 = −22 on META) and slowdown (R^2 = −150 on META), while Hurdle and Two-Stacking Stage performed tens of times better. This confirms that ignoring the zero-structure arising from efficient scheduling results in unusable predictions. The effectiveness of the approach is linked to the target structure and data size. For waiting time (a sharp zero target), the Hurdle model clearly outperforms the others. However, for response time that loses its sharp zero nature (because each job has a positive runtime), the advantage of decompression diminishes—and specifically with WAGAP workload, the simple RF Solo outperforms all others, while Stacking collapses. This reveals that adding architectural complexity to scarce data without a sharp zero can be counterproductive.

The waiting time on WAGAP was the only instance out of six trials where R^2 yielded positive values for all models (Hurdle model: 0.195). This is due to the very high percentage of zeros, which makes predicting zero for the

majority very close to reality, reflecting a striking paradox: the more efficient the scheduler (more zeros), the easier the overall prediction becomes, but predicting positive cases becomes more difficult because of their rarity.

Time Block CV analysis revealed a clear decoupling of the model components: the classifier maintained high stability (ROC-AUC: 0.985–0.995 on META), while the regressor showed sensitivity to temporal shifts at specific folds. Stacking significantly reduced these jumps (63% on META, 96% on WAGAP), confirming that the meta-learner adds a layer of robustness against time-dependent operating mode variations. The fluctuations observed on WAGAP reflect a statistical evaluation challenge stemming from the scarcity of positive samples at individual folds, not a weakness in the model itself, a finding corroborated by the final test set results, where the Hurdle outperformed the RF Solo. The main message of this study is that any efficient scheduler will inherently produce a high percentage of zero-times; therefore, the need for a two-stage framework is not specific to any particular dataset but is a structural feature of the problem of predicting wait times after efficient scheduling. The choice of predictive architecture must be driven by the structure of the target: sharp zeros require two-stage decomposition, while continuous targets may suffice with simpler models—especially when data is scarce.

For future work, this study revealed several challenges that open the door to promising future developments. Time Block CV analysis showed that the WAGAP model evaluation suffers from statistical instability due to the scarcity of positive samples in individual folders. Therefore, we recommend adopting stratified temporal splitting, which guarantees a minimum number of positive samples per folder. This will provide a more reliable estimate of model performance over time and enable fair comparisons between architectures, even with extremely limited data. The sharp regressor jumps observed in certain folders (especially Fold 4 on META) indicate that cluster operating patterns change over time. Exploring adaptive/online retraining would allow the model to periodically update itself as operating patterns evolve, reducing the regressor's sensitivity to temporal shifts and maintaining long-term prediction accuracy. Additionally, this study was limited to the BestGap scheduler and two datasets.

To confirm that excessive zeroing is a general phenomenon related to scheduling efficiency and not specific to any particular environment, we recommend applying the framework to other schedulers (such as Easy Backfilling and Conservative Backfilling) and various HPC systems. This expansion will reveal whether the level of zeroing varies with the scheduling policy and identify the conditions under which binary decomposition becomes necessary versus those where simple models suffice.

Author Contributions

The author reviewed and approved the final version of the manuscript

	Ö.D.
C	100
D	100
S	100
DCP	100
DAI	100
L	100
W	100
CR	100
SR	100
PM	100
FA	100

C= concept, D=design, S=supervision, DCP=data collection and/or processing, DAI=data analysis and/or interpretation, L=literature search, W=writing, CR=critical review, SR=submission and revision, PM=project management, FA=funding acquisition.

Conflict of Interest

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

Ethical Consideration

Ethics committee approval was not required for this study because there was no study on animals or humans.

References

Abraham, A., Liu, H., Grosan, C., & Xhafa, F. (2008). Nature inspired meta-heuristics for grid scheduling: Single and multi-objective optimization approaches. In F. Xhafa & A. Abraham (Eds.), *Metaheuristics for scheduling in distributed computing environments* (pp. 247–272). Springer.

Bailey Lee, C., Schwartzman, Y., Hardy, J., & Snavely, A. (2004). Are user runtime estimates inherently inaccurate? In D. G. Feitelson, L. Rudolph, & U. Schwiegelshohn (Eds.), *Job scheduling strategies for parallel processing* (pp. 228–243). Springer.

Baker, N. (2024). Unlocking a new era for scientific discovery with AI: How Microsoft's AI screened over 32 million candidates to find a better battery. *Microsoft Azure Quantum Blog*. https://azure.microsoft.com/en-us/blog/unlocking-a-new-era-for-scientific-discovery-with-ai/

Bartolini, A., Borghesi, A., Bridi, T., Lombardi, M., & Milano, M. (2014). Proactive workload dispatching on the EURORA supercomputer. In B. O'Sullivan (Ed.), *Principles and practice of constraint programming* (pp. 744–759). Springer.

Berriman, G. B., Laity, A. C., Good, J. C., Katz, D. S., Jacob, J. C., Deelman, E., & Prince, T. A. (2006). Science applications of the Montage image mosaic engine. *Proceedings of the International Astronomical Union*, *2*(S14), 621–621. https://doi.org/10.1017/S174392130600495X

Boëzennec, R., Dufossé, F., & Pallez, G. (2023). Optimization metrics for the evaluation of batch schedulers in HPC. In D. Klusáček, W. Cirne, & N. Desai (Eds.), *Job scheduling strategies for parallel processing* (pp. 3–23). Springer.

Boëzennec, R., Dufossé, F., & Pallez, G. (2024). Qualitatively

analyzing optimization objectives in the design of HPC resource manager. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, *9*(4), 1–28. https://doi.org/10.1145/3675392

Borghesi, A., Collina, F., Lombardi, M., Milano, M., & Benini, L. (2015). Power capping in high performance computing systems. In G. Pesant (Ed.), *Principles and practice of constraint programming* (pp. 531–547). Springer.

Borghesi, A., Di Santi, C., Molan, M., Ardebili, M. S., Mauri, A., Guarrasi, M., Cavazzoni, C., & Benini, L. (2023). M100 ExaData: A data collection campaign on the CINECA's Marconi100 Tier-0 supercomputer. *Scientific Data*, *10*(1), Article 288. https://doi.org/10.1038/s41597-023-02174-3

Brevik, J., Nurmi, D., & Wolski, R. (2006). 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* (pp. 110–118). ACM. https://doi.org/10.1145/1122971.1122990

Bridi, T., Bartolini, A., Lombardi, M., Milano, M., & Benini, L. (2016). A constraint programming scheduler for heterogeneous high-performance computing machines. *IEEE Transactions on Parallel and Distributed Systems*, *27*(10), 2781–2794. https://doi.org/10.1109/TPDS.2016.2516990

Chang, C. (2018). *Multiphysics magnetic fusion reactor simulator, from hot core to cold wall* [Doctoral dissertation, Princeton University]. Princeton University Repository.

Chen, X., Zhang, H., Bai, H., Yang, C., Zhao, X., & Li, B. (2020). Runtime prediction of high-performance computing jobs based on ensemble learning. In *Proceedings of the 2020 4th International Conference on High Performance Compilation, Computing and Communications* (pp. 28–32). ACM. https://doi.org/10.1145/3391811.3391823

Dakkak, O., Arif, S., & Nor, S. A. (2015a). A critical analysis of simulators in grid. Jurnal Teknologi (Sciences & Engineering), 77(4), 111–117. https://doi.org/10.11113/jt.v77.6050

Dakkak, O., Arif, A. S. B. C. M., & Nor, S. A. (2015b). Resource allocation mechanisms in computational grid: A survey. ARPN Journal of Engineering and Applied Sciences, 10(15), 6662–6667

Dakkak, O., Fazea, Y., Nor, S. A., & Arif, S. (2021). Towards accommodating deadline driven jobs on high performance computing platforms in grid computing environment. *Journal of Computational Science*, *54*, Article 101439. https://doi.org/10.1016/j.jocs.2021.101439

DeLucia, A., & Baseman, E. (2018). Work in progress: Topic modeling for HPC job state prediction. In *Proceedings of the First Workshop on Machine Learning for Computing Systems* (pp. 1–4). ACM.

Doll, F., Habbel, B., Liesch, T., Wetzel, M., Kunz, S., & Broda, S. (2026). Validation strategies for deep learning-based groundwater level time series prediction using exogenous meteorological input features. *Geoscientific Model Development*, *19*(7), 2657–2675. https://doi.org/10.5194/gmd-19-2657-2026

Du, J., & Leung, J. Y.-T. (1989). Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, *2*(4), 473–487. https://doi.org/10.1137/0402040

Ensmenger, N. (2018). The environmental history of computing. *Technology and Culture*, *59*(4), S7–S33. https://doi.org/10.1353/tech.2018.0148

Etinski, M., Corbalan, J., Labarta, J., & Valero, M. (2011). Linear programming based parallel job scheduling for power constrained systems. In *2011 International Conference on High Performance Computing & Simulation* (pp. 389–398). IEEE. https://doi.org/10.1109/HPCSim.2011.5999847

Fan, Y., Lan, Z., Rich, P., Allcock, W. E., Papka, M. E., Austin, B., & Paul, D. (2019). Scheduling beyond CPUs for HPC. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing* (pp. 201–212). ACM. https://doi.org/10.1145/3307681.3325409

Feitelson, D. G., Rudolph, L., & Schwiegelshohn, U. (2004). Parallel job scheduling—A status report. In D. G. Feitelson, L. Rudolph, & U. Schwiegelshohn (Eds.), *Job scheduling strategies for parallel processing* (pp. 1–16). Springer.

Frachtenberg, E., & Feitelson, D. G. (2005). Pitfalls in parallel job scheduling evaluation. In D. G. Feitelson & L. Rudolph (Eds.), *Job scheduling strategies for parallel processing* (pp. 257–282). Springer.

Gaikwad, B., Simakov, N. A., Furlani, T., White, J. P., & Patra, A. (2025). Predictive modeling of HPC job queue times: Improving user decision-making and resource utilization. In *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration* (pp. 1–4). ACM. https://doi.org/10.1145/3708579.3708595

Gainaru, A., Aupy, G. P., Sun, H., & Raghavan, P. (2019). Speculative scheduling for stochastic HPC applications. In *Proceedings of the 48th International Conference on Parallel Processing* (pp. 1–10). ACM. https://doi.org/10.1145/3337821.3337905

Galleguillos, C., Sîrbu, A., Kiziltan, Z., Babaoglu, O., Borghesi, A., & Bridi, T. (2017). Data-driven job dispatching in HPC systems. In G. Nicosia, G. Pardalos, & R. Umeton (Eds.), *Machine learning, optimization, and big data* (pp. 245–257). Springer.

Gaussier, E., Glesser, D., Reis, V., & Trystram, D. (2015). Improving backfilling by using machine learning to predict running times. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1–10). ACM. https://doi.org/10.1145/2807591.2807641

Goponenko, A. V., Lamar, K., Allan, B. A., Brandt, J. M., & Dechev, D. (2024). Job scheduling for HPC clusters: Constraint programming vs. backfilling approaches. In *Proceedings of the 18th ACM International Conference on Distributed and Event-based Systems* (pp. 154–165). ACM. https://doi.org/10.1145/3629104.3661559

Goponenko, A. V., Lamar, K., Peterson, C., Allan, B. A., Brandt, J. M., & Dechev, D. (2022). Metrics for packing efficiency and fairness of HPC cluster batch job scheduling. In *2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)* (pp. 41–50). IEEE. https://doi.org/10.1109/SBAC-PAD55451.2022.00016

Houssein, E. H., Gad, A. G., Wazery, Y. M., & Suganthan, P. N. (2021). Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends. *Swarm and Evolutionary Computation*, *62*, Article 100841. https://doi.org/10.1016/j.swevo.2021.100841

Hovestadt, M., Kao, O., Keller, A., & Streit, A. (2003). Scheduling in HPC resource management systems: Queuing vs. planning. In D. G. Feitelson, L. Rudolph, & U. Schwiegelshohn (Eds.), *Job scheduling strategies for parallel processing* (pp. 1–20). Springer.

Jackson, D., Snell, Q., & Clement, M. (2001). Core algorithms of the Maui scheduler. In D. G. Feitelson & L. Rudolph (Eds.), *Job scheduling strategies for parallel processing* (pp. 87–102). Springer.

Jette, M. A., & Wickberg, T. (2023). Architecture of the Slurm workload manager. In D. Klusáček, W. Cirne, & N. Desai (Eds.), *Job scheduling strategies for parallel processing* (pp. 24–41). Springer.

Klusáček, D., Rudová, H., Baraglia, R., Pasquali, M., & Capannini,

G. (2008). Comparison of multi-criteria scheduling techniques. In T. Priol & M. Vanneschi (Eds.), *Grid computing: Achievements and prospects* (pp. 173–184). Springer.

Klusáček, D., Tóth, Š., & Podolníková, G. (2016). Complex job scheduling simulations with Alea 4. In *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques* (pp. 114–121). EAI. https://doi.org/10.4108/eai.31-8-2016.2265039

Kolker-Hicks, E., Zhang, D., & Dai, D. (2023). A reinforcement learning based backfilling strategy for HPC batch jobs. In *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis* (pp. 1386–1393). ACM. https://doi.org/10.1145/3624062.3624208

Kurth, T., Treichler, S., Romero, J., Mudigonda, M., Luehr, N., Phillips, E., Mahesh, A., Matheson, M., Deslippe, J., Fatica, M., & Prabhat. (2018). Exascale deep learning for climate analytics. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 649–660). IEEE. https://doi.org/10.1109/SC.2018.00057

Leung, V. J., Sabin, G., & Sadayappan, P. (2010). Parallel job scheduling policies to improve fairness: A case study. In *2010 39th International Conference on Parallel Processing Workshops* (pp. 217–224). IEEE. https://doi.org/10.1109/ICPPW.2010.38

Mu'alem, A. W., & Feitelson, D. G. (2001). Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, *12*(6), 529–543. https://doi.org/10.1109/71.932709

Oh, J., & Kim, Y. (2020). Job placement using reinforcement learning in GPU virtualization environment. *Cluster Computing*, *23*(3), 2219–2234. https://doi.org/10.1007/s10586-020-03082-x

Park, J.-W. (2019). Queue waiting time prediction for large-scale high-performance computing system. In *2019 International Conference on High Performance Computing & Simulation (HPCS)* (pp. 111–117). IEEE. https://doi.org/10.1109/HPCSim40371.2019.00028

Pathak, A. R. (2025). Highlights on utilizing machine learning for high performance computing systems. *Procedia Computer Science*, *258*, 1242–1253. https://doi.org/10.1016/j.procs.2025.07.158

Roberts, D. R., Bahn, V., Ciuti, S., Boyce, M. S., Elith, J., Guillera-Arroita, G., Hauenstein, S., Lahoz-Monfort, J. J., Schröder, B., Thuiller, W., & Dormann, C. F. (2017). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography*, *40*(8), 913–929. https://doi.org/10.1111/ecog.02881

Rudolph, L., & Smith, P. H. (2000). Valuation of ultra-scale computing systems. In D. G. Feitelson & L. Rudolph (Eds.), *Job scheduling strategies for parallel processing* (pp. 235–248). Springer.

Tanash, M., Dunn, B., Andresen, D., Hsu, W., Yang, H., & Okanlawon, A. (2019). Improving HPC system performance by predicting job resources via supervised machine learning. In *Practice and Experience in Advanced Research Computing 2019: Rise of the Machines (Learning)* (pp. 1–8). ACM. https://doi.org/10.1145/3332186.3333157

Tirmazi, M., Barker, A., Deng, N., Haque, M. E., Qin, Z. G., Hand, S., Harchol-Balter, M., & Wilkes, J. (2020). Borg: The next generation. In *Proceedings of the Fifteenth European Conference on Computer Systems* (pp. 1–14). ACM. https://doi.org/10.1145/3342195.3387517

Tsai, C.-W., & Rodrigues, J. J. (2013). Metaheuristic scheduling for cloud: A survey. *IEEE Systems Journal*, *8*(1), 279–291. https://doi.org/10.1109/JSYST.2013.2268316

Verma, A., Korupolu, M., & Wilkes, J. (2014). Evaluating job packing in warehouse-scale computing. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)* (pp. 48–56). IEEE. https://doi.org/10.1109/CLUSTER.2014.6968735

Wang, L., Rodriguez, M. A., & Lipovetzky, N. (2026). MetaPilot: A DRL-based controller for dynamic adaptation to shifting scheduling objectives in HPC systems. *Future Generation Computer Systems*, *178*, 1–15. https://doi.org/10.1016/j.future.2025.108269