INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2025, VOL. 71, NO. 4, PP. 1-10
Manuscript received August 20, 2025; revised September 2025. doi: 10.24425/ijet.2025.155471

Performance evaluation of selected ML algorithms in GC and AWS cloud environments

Grzegorz Blinowski, and Marcin Bogucki

***Abstract*—In this paper, we analyze the performance of common machine learning (ML) algorithms executed in Google Cloud and Amazon Web Services environments. The primary metric is training and prediction time as a function of the number of virtual machine cores. For comparison, benchmarks also include a "bare metal" (i.e. - non-cloud) environment, with results adjusted using the "Multi-thread Score" to account for architectural differences among the tested platforms.**

Our focus is on CPU-intensive algorithms. The test suite includes Support Vector Machines, Decision Trees, K-Nearest Neighbors, Linear Models, and Ensemble Methods. The evaluated classifiers, sourced from the scikit-learn and ThunderSVM libraries, include: Extra Trees, Support Vector Machines, K-Nearest Neighbors, Random Forest, Gradient Boosting Classifier, and Stochastic Gradient Descent. GPU-accelerated deep learning models, such as large language models, are excluded due to the difficulty of establishing a common baseline across platforms.

The dataset used is the widely known "Higgs dataset," which describes kinematic properties measured by particle detectors in the search for the Higgs boson.

Benchmark results are best described as varied—there is no clear trend, as training and prediction times scale differently depending on both the cloud platform and the algorithm type. This paper provides practical insights and guidance for deploying and optimizing CPU-based ML workloads in cloud environments.

***Keywords*—Public Cloud; ML/AI algorithms; performance evaluation**

I. Introduction

A. AI/ML computing in the public cloud – state of the art

The rapid development of Artificial Intelligence (AI) and Machine Learning (ML) algorithms, methods, and tools has significantly increased the demand for scalable, flexible, and cost-efficient computing resources. Public cloud platforms [1] such as Google Cloud Platform (GC), Amazon Web Services (AWS), and Microsoft Azure have emerged as dominant providers offering comprehensive computational infrastructure for AI and ML workloads.

Public cloud computing provides on-demand access to computing resources, enabling scalable storage, data processing, and algorithm execution [2,3]. Moreover, public cloud AI/ML services lower entry barriers for organizations and researchers seeking to adopt these technologies. AI and ML workloads, particularly deep learning applications, often require substantial computational power, which cloud platforms provide through elastic compute instances and specialized hardware such as Graphics Processing Units (GPUs).

In general, public cloud platforms offer three models of service delivery [4]:

- Infrastructure as a Service (IaaS): Provides raw compute, storage, and net-working resources.
- Platform as a Service (PaaS): Offers pre-built software frameworks and development environments.
- Software as a Service (SaaS): Delivers ready-to-use applications through the cloud.

This study focuses exclusively on the IaaS model, as it offers the greatest flexibility and is currently the most prevalent approach for AI and ML applications in both academic and industrial settings.

Public cloud Infrastructure as a Service (IaaS) operate primarily under usage-based, scalable payment models, enabling cost alignment with resource consumption. The most common cost models for public cloud IaaS include:

- Pay-as-you-go (On-Demand) Model: Users are billed based on actual re-source consumption (e.g., compute hours, storage usage, data transfer) with no long-term commitment.
- Reserved Instances (Commitment-based) Model: Users commit to purchasing specific resources (e.g., virtual machines, GPUs) for a defined period (typically 1 to 3 years) in exchange for significant discounts compared to on-demand pricing.
- Spot/Preemptible Instances Model: Providers offer spare capacity at significantly reduced rates, but with the caveat that instances can be terminated with little notice.

Because of flexibility for irregular usage, lack of long-term commitment as well as ease of access via grants and credits (such as for example [5] – in academic environments, the pay-as-you-go (on-demand) cost model is by far the most popular for public cloud IaaS usage, particularly for AI and ML research workloads. This cost-related concerns often become vital in case of resource and time-consuming AI/ML application – as we will elaborate in the next section.

B. Challenges, research gaps and scope of research

While public cloud platforms simplify AI/ML adoption for both scholarly research and enterprise use, several challenges persist [6], two of which are the focus of this study:

- Performance Variability: Cloud instances frequently exhibit variable, inconsistent, or unpredictable performance,

This work was supported by the Statutory Grant of the Polish Ministry of Science and Higher Education.

Grzegorz Blinowski and Marcin Bogucki are with Institute of Computer Science, Warsaw University of Technology, Warszawa, Poland (e-mail: grzegorz.blinowski@pw.edu.pl, marcin.bogucki.stud@pw.edu.pl).

particularly for compute-intensive AI and ML tasks. This variability can significantly impact execution time, scalability, and overall workload efficiency.
- Cost Optimization: Achieving an optimal balance between computational performance and cost efficiency remains a complex challenge. This is further com-plicated by the lack of transparent, consistent data regarding algorithm execution times across different cloud environments.

This study investigates the performance of selected CPU-based machine learning (ML) algorithms executed within public cloud IaaS environments, with a comparative analysis between Google Cloud (GC) and Amazon Web Services (AWS) platforms. To provide an objective performance baseline, an additional non-cloud "bare metal" (BM) environment is included in the evaluation.

Our focus is on classification algorithms that are widely used and accessible through established, open-source libraries such as scikit-learn and ThunderSVM. GPU-based models, deep learning frameworks, and large language models (LLMs) are intentionally excluded due to their distinct hardware requirements and the difficulty of establishing a unified baseline across platforms.

Performance in our study is assessed based on algorithm training and prediction times, with particular emphasis on scalability in relation to the number of available virtual CPU cores. Additionally, benchmark results are normalized using a multi-core efficiency score derived from independent hardware performance data to account for architectural differences among platforms.

The research employs the well-established, balanced Higgs dataset, ensuring relevance for large-scale classification tasks while maintaining practical computation times. The primary objective is to provide actionable insights into the scalability, performance limitations, and cost implications of deploying CPU-based ML workloads in cloud environments, with particular consideration given to the pay-as-you-go pricing model prevalent in academic research.

The remainder of this paper is organized as follows: Section 2 discusses related work. The research methodology and experimental setup are detailed in Section 3. Section 4 presents the experimental results, which are analyzed in Section 5. Finally, Section 6 provides conclusions and future work.

II. Related Work

Benchmarking the performance of public cloud providers is a well-established research topic. Numerous studies have evaluated the performance of different cloud platforms across a variety of use cases, experimental setups, and operational constraints. Most existing research focuses on low-level performance assessments, where specific subsystems—such as compute, memory, storage, or networking—are evaluated in isolation. Other popular areas of investigation include the performance of specific application domains, such as High-Performance Computing (HPC), simulations, big data analytics, and Internet of Things (IoT) platforms.

To date, however, the authors are not aware of any comprehensive studies that specifically address the performance of AI and ML algorithms executed within public cloud IaaS environments, particularly with respect to common classification algorithms running in CPU-only configurations. Below, we summarize selected studies relevant to our work in terms of platform scope, environment, methodology, and key findings.

Leitner and Cito [7] present a large-scale, systematic literature review on the predictability of performance in public Infrastructure-as-a-Service (IaaS) clouds. Their analysis revealed substantial performance differences between cloud providers, with multitenancy emerging as a key factor influencing both performance variability and predictability. Importantly, the impact of multitenancy was shown to vary significantly between providers.

Sadooghi et al. [8] conducted an extensive quantitative study of Amazon EC2, focusing on its suitability for running scientific applications. Their methodology involved low-level benchmarking of memory, networking, and I/O subsystems, followed by an evaluation of complete scientific workloads. The performance of AWS was compared to that of a private cloud infrastructure (FermiCloud), highlighting inherent limitations in public cloud environments for certain computational workloads. However, this study focused exclusively on a single public cloud provider (AWS) and did not address AI or ML applications.

In another relevant study, Ericson et al. [9] investigated performance variability across public cloud services, including Microsoft Azure and AWS. The authors emphasized that performance fluctuations should be a concern for cloud service consumers, as such variability may adversely affect Quality of Service (QoS), user experience, and ultimately, pricing efficiency.

Collectively, these studies underscore the importance of understanding cloud performance characteristics but also reveal a clear gap in the literature concerning the performance of ML algorithms across public IaaS platforms, particularly for CPU-based workloads. Addressing this gap is the primary objective of this work.

III. Experimental setup and Methodology

A. Algorithm and Dataset selection

The scope of this work is limited to classification algorithms, which was an intentional decision given the wide variety of AI and ML methods available. We have explicitly excluded GPU-based algorithms and regression models from the study. Despite these exclusions, the chosen approach still covers a large and practically relevant subset of AI/ML problems.

The selection of the dataset for benchmarking required a careful compromise between small and large datasets. Smaller datasets, with limited records and low complexity, often result in very short computation times, making it difficult to measure resource consumption accurately due to system overhead influencing the timing. Conversely, very large and highly complex datasets can lead to prolonged, day-long computations, which may significantly impact both the cost and the efficiency of the experiments.

Our dataset choice and the corresponding algorithms used in this study are presented in Table I.

TABLE I
ALGORITHMS USED IN THE BENCHMARK

Algorithm abbreviation	Description	Codebase
TSVC	Support Vector Classifier	ThunderSVM SVC
KNC	K-Nearest Neighbors Classifier	scikit-learn neighbors
KNC_2	Tuned K-Nearest Neighbors Classifier (with non-standard hyperparameters) from scikit-learn library	scikit-learn neighbors
RFC	Random Forest Classifier	scikit-learn ensemble
ETC	Extra Trees Classifier	scikit-learn ensemble
SGDC	SGD Classifier	scikit-learn linear model
HistGBC	Hist Gradient Boosting Classifier	scikit-learn ensemble
HistGBC_2	Tuned Hist Gradient Boosting Classifier (with non-standard hyperparameters)	scikit-learn ensemble

The dataset selected for this benchmark represents a necessary compromise between small and large datasets. Datasets with a limited number of records and low complexity typically lead to very short computation times, which makes it difficult to accurately measure resource consumption due to system overhead influencing the timing results. In contrast, datasets with a very large number of records and high complexity often result in prolonged, resource-intensive computations that can span many hours or even days, which negatively impacts both the cost and efficiency of experimental work.

To strike this balance, we selected the widely recognized Higgs dataset [10], which describes kinematic properties measured by particle detectors in the search for the Higgs boson. The key characteristics of the dataset are as follows:

- 940160 instances
- 25 features
- 2 solution classes ("0" – background noise; "1" – signal)
- Dataset is balanced, meaning the classes have the same number of elements for both solution classes

The scikit-learn library [11], initially developed by David Cournapeau in 2007 developed in Python, serves as the primary tool for implementing the benchmark suite. Scikit-learn is a widely used, open-source machine learning library released under a BSD license, making it suitable for both academic and commercial applications. Scikit-learn library integrates seamlessly with other core Python scientific libraries, including SciPy, NumPy, and matplotlib. Scikit-learn focuses on providing efficient implementations of various machine learning algorithms, and it is extensively applied in both academic research and industry settings. The library offers comprehensive support for: data preprocessing; supervised and unsupervised ML models; Hyperparameter tuning; model selection and extraction of features. Since its inception, the library has seen substantial improvements and growth thanks to large worldwide com-munity and contributors.

B. Test Environment and hardware performance scaling

We conducted our tests across two major cloud environments: Amazon AWS and Google Cloud (GC), as a reference we used a non-cloud "bare metal" (BM) platform. A uniform test suite —comprising benchmark programs written in Python—was deployed consistently across all platforms.

The configurations of the cloud platforms and reference bare metal configuration (BM) are summarized below:

BM setup:

- Operating system: Ubuntu 24.04 LTS
- CPU: Intel i7-11700k with 8 physical cores, 16 threads, maximum frequency 5GHz,
- CPU Architecture: x86_64,
- Memory: 32GB virtual RAM (20GB + 12 GB swap)
- Storage: Viper M.2 NVME 512GB disk, 55GB ext4 partition used for Linux installation.

AWS setup:

- Virtual machine type: C6a-4xlarge [12,13], Compute Optimized,
- Operating system – Amazon Linux 2, lightweight Linux distribution, owned by cloud platform vendor [14],
- CPU: AMD gen. 3 EPYC 7R13, frequency up to 3.6GHz, 16 virtual cores, 16 threads,
- CPU Architecture: x86_64,
- Memory: 32GB virtual RAM,
- Storage: 25GB gp3,
- Enhanced networking.

GC setup:

- Virtual machine type: c2-standard-8 Compute Optimized in "pay as you go" model [15],
- Operating system: CentOS Stream 9, free, enterprise grade Linux distribution [16],
- CPU: Intel Xeon Scalable Processor (Cascade Lake) 2nd Generation, Intel® Xeon® Gold 6253CL Processor, 8 virtual cores, 8 threads, frequency up to 3.9GHz,
- Memory: 64GB virtual RAM memory.

C. Adjusting hardware performance parameters

It is not possible to select cloud platforms with identical hardware specifications, particularly with respect to CPU type, clock speed, and architecture. Consequently, we selected similar, but not identical, platforms across the different environments. To enable meaningful performance comparisons, we applied a hardware adjustment coefficient, calculated based on publicly available PassMark CPU benchmark data [17]. This approach allows us to account for the inherent performance differences between processor types.

Our benchmarking and normalization methodology is as follows:

- In the first step, we acquired data "as-is" (this data is not presented in this work due to space limitations)
- In the second step, we "normalize" the data, meaning the recorded computation times are multiplied by the values depending on the individual benchmarks of the

subsequent processors, the baseline being our "BM" platform, for which we assume a multiplier of 1.0.

The CPU benchmark results used to calculate the hardware coefficients are presented in Table II.

TABLE II
ALGORITHMS USED IN THE BENCHMARK

CPU Name	Intel i7-11700k	Intel Xeon Gold 6253CL	AMD EPYC 7R13
CPU Class	Desktop	Server	Server
CPU used on platform	**BM**	**GC**	**AWS**
Multi thread score	24455	28549	82158
% difference (multi thread)	100,0%	116,7%	336,0%
Multi-core efficiency score	**3057**	**1586**	**1712**
% difference (multi thread per core)	**100,0%**	**51,9%**	**56,0%**

After careful consideration of the benchmarking methodology, we concluded that the most appropriate metric for performance normalization is the Multi-thread Score, which we refer to as the multi-core efficiency score throughout this study. The rationale for this choice is as follows:

- Single-thread scores differ significantly between platforms. Since our primary interest lies in evaluating scalability, using single-thread scores for normalization would disproportionately favor the bare metal (BM) platform, which has superior single-core performance.
- The total multi-thread score reflects overall processor performance when all cores are fully utilized. However, in cloud environments, users typically only rent a fraction of the total available processor cores, making this score an inaccurate basis for comparison.
- The multi-core efficiency score, calculated as the ratio of the total multi-thread score to the number of physical cores, provides a fairer estimate of the computational power available per core. This metric better reflects the effective performance scaling potential across different platforms, making it the most suitable choice for normalizing our benchmark results.

D. Software: platform and libraries

To ensure a uniform execution environment for all benchmark calculations across platforms, it was necessary to install or upgrade certain default packages and applications to the latest available versions for each Linux distribution. For this purpose, the respective built-in system package managers were utilized: Aptitude for Ubuntu, and Yum for both Amazon Linux and CentOS. The complete list of software pack-ages, applications, and libraries used in the benchmark environment is provided in Appendix A.

Full code and intermediate data are available under the following link: https://github.com/gjbl/ML_Cloud_Perf

E. Measurement methods and result verification

As Python has been used with scikit-learn library, it allows to use several built-in tools for measuring calculation times and accuracy scores.

- Independent variable *n_jobs* establishes the number of vCPUs that are utilized in a given benchmark run.
- Time measurement is the basis for the primary metric – the given algorithm's total execution time. Python method *time.time()* from time library has been used to record the timestamp directly before and after the calculations.
- Additional metrics: methods of accuracy, F1 score, and confusion matrix [18] have been imported from *sklearn.metrics* library. While time-based performance is our primary concern accuracy metrics were checked to verify that all algorithms behaved correctly.
- CPU and memory: real-time observation using the standard Linux htop tool was conducted to ensure that the full system capacity was indeed being used.

All data related to timing was automatically recorded in csv files and averaged over multiple executions.

IV. BENCHMARK RESULTS

A. Introduction

This section presents the results of the machine learning performance evaluation conducted across both bare metal (BM) and two public cloud platforms. The primary objective of this work was to assess the scalability of selected algorithms by measuring their training and prediction times as a function of the number of available CPU cores.

The algorithms listed in Table I were tested on the same dataset, with the number of CPU cores incremented to evaluate scalability. For each iteration, both training and prediction times were recorded. The central goal was to determine whether the performance of the selected algorithms improves with the addition of CPU cores. In addition, performance differences between execution on the BM platform and the cloud Virtual Machines (VMs) were analyzed and are reported. Due to space limitations, only the normalized results are presented below. For each algorithm, uniform plots are provided showing training and prediction times separately, as a function of the number of CPU cores used. All three platforms are shown together in each plot for direct comparison.

The number of cores (and hence of the *n_jobs* parameter) on the GC platform was limited to 8; however, as we will demonstrate in section 4.3 regarding gain, this limitation has a small, if any, practical impact on the final performance analysis.

The time results are presented using standard box-and-whisker plots, with values averaged over multiple runs. An exception is the TSVC algorithm, which, due to its significantly longer execution times, is treated separately. For reference and validation, the mean accuracy score (F1 score) for each algorithm is also reported.

B. Normalized benchmark results

The benchmarks result for algorithms exhibiting clear scalability are presented in Fig. 1 – 3. The rest of the algorithms is discussed briefly in the subsequent subsection.

Extra Trees Classifier from scikit-learn library (ETC)

- Mean Accuracy: 71,46%.
- Scalability: Yes, both for training and prediction processing.

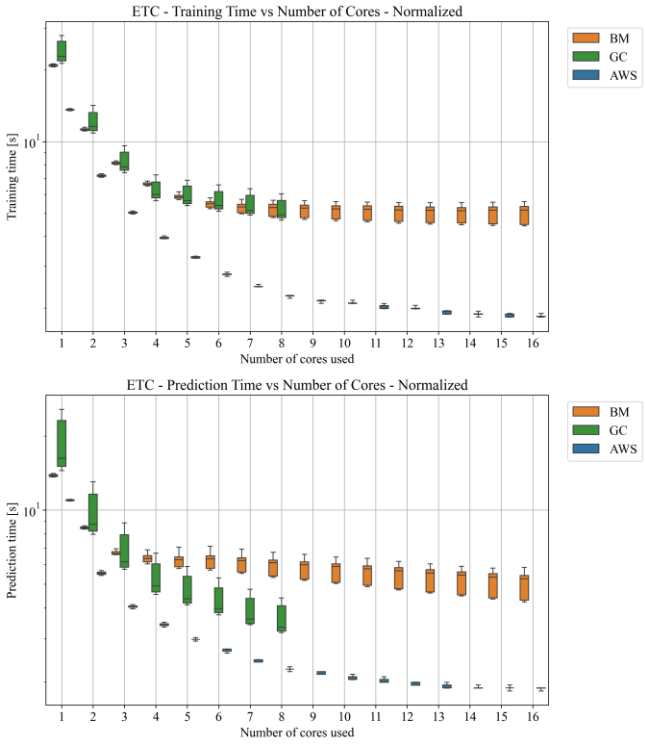

Fig. 1. Normalized Results for the ETC classifier

Support Vector Machine Classifier from ThunderSVM library (TSVC)

- Mean Accuracy score: 68,32%.
- Scalability: Yes, both for training and prediction processing

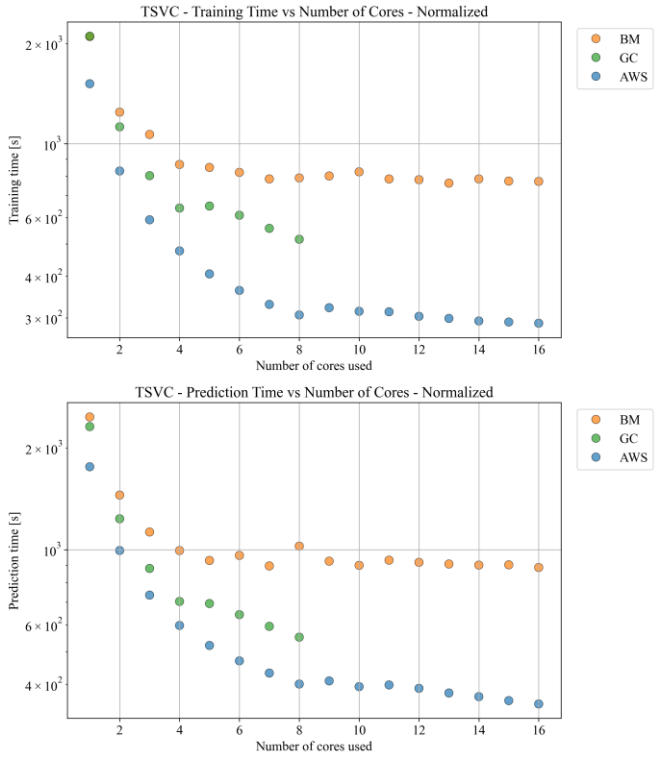

Fig. 2. Normalized Results for the TSVC classifier

Random Forest Classifier from scikit-learn library (RFC)

- Mean Accuracy: 72,57%.
- Scalability: Yes, both for training and prediction processing.

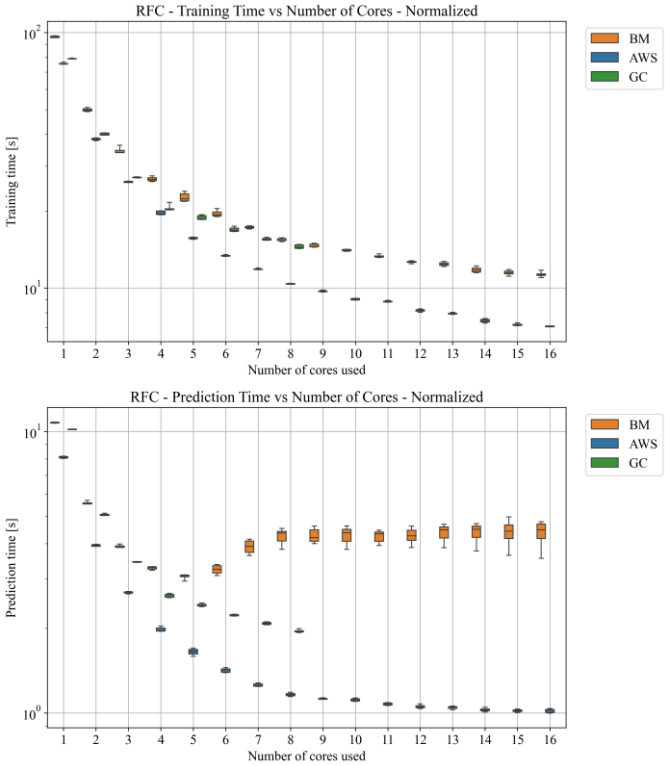

Fig. 3. Normalized Results for the RFC classifier

K-Nearest Neighbors Classifier from scikit-learn library (KNC)

- Mean Accuracy: 58,54%.
- Scalability: No scalability observed.

This algorithm did not exhibit significant scalability with respect to the number of CPU cores. The best results (both for training and prediction) were obtained for AWS, results for GC were approximately 40% worse.

K-Nearest Neighbors Classifier (with non-standard hyperparameters) from scikit-learn library (KNC_2)

- Mean Accuracy: 60,56%.
- Scalability: No scalability observed.
- Hyperparameters tweak: *n_neighbors=19* - improved accuracy and prediction time.

As its predecessor, this algorithm did not exhibit significant scalability with respect to the number of CPU cores. The best results (both for training and prediction) were obtained for AWS; results for GC were approximately 60% worse.

Stochastic Gradient Decent Classifier from scikit-learn library (SGDC)

- Efficient for this big dataset. The fastest training and prediction time.
- Mean Accuracy 62,78%.
- Scalability: No scalability observed.

This method also did not exhibit significant scalability with respect to the number of CPU cores. The best results for training

were obtained for AWS; results for GC were approximately 30% worse. In the case of prediction, the difference was even higher, with AWS being 60% better than GC.

Hist Gradient Boosting Classifier from scikit-learn library (HistGBC)

High performance, one of the fastest for training and predicting with a huge dataset. One of the highest accuracy scores.

- Mean Accuracy: 73,04%.
- Scalability: No scalability observed.
- Hyperparameters: Default parameters used.

This method also did not exhibit significant scalability with respect to the number of CPU cores. Training time was lower by 10% for GC (with respect to AWS), while prediction time was almost 2 times better in the case of AWS.

Hist Gradient Boosting Classifier (with non-standard hyperparameters) HistGBC_2

- Very low training times, improved performance with optimized hyperparameters.
- Mean Accuracy: 73,30%.
- Scalability: No scalability observed.
- Hyperparameters: *l2_regularization=1.0, learning_rate=0.2, max_depth=5, max_iter=300* - improved accuracy.

As the previous variant, this model also did not exhibit significant scalability with respect to the number of CPU cores. Training time was 15% lower for GC (with respect to AWS), while prediction time was 3 times better in case of AWS.

C. Algorithm specific gain and saturation points

To determine the optimal number of CPU cores for each algorithm with respect to computation time improvement, we introduce a simple gain formula, defined as follows:

$$gain_n = \frac{T_{n-1} - T_n}{T_{n-1}} \quad (1)$$

here:

- T_n = Mean execution time (training time or prediction time) for n CPU cores,
- *gain* = Relative decrease in execution time, expressed as a percentage.

The saturation point is defined as the lowest number of CPU cores for which the calculated gain falls below 10%. In other words, increasing the number of cores be-yond this point results in less than 10% improvement in execution time between consecutive core counts, suggesting that further scaling is not economically or computationally efficient.

The gain and saturation point calculations are based on the non-normalized data, as the actual (real) execution times are considered for these evaluations. The results for algorithms exhibiting clear scalability are presented in Fig. 4 – 6.

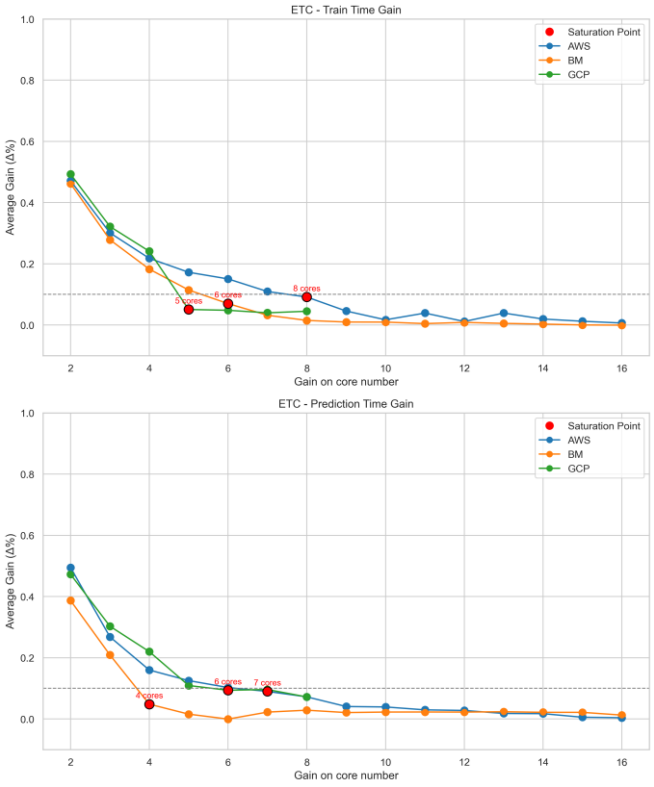

Fig. 4. Gain and saturation for ETC algorithm.

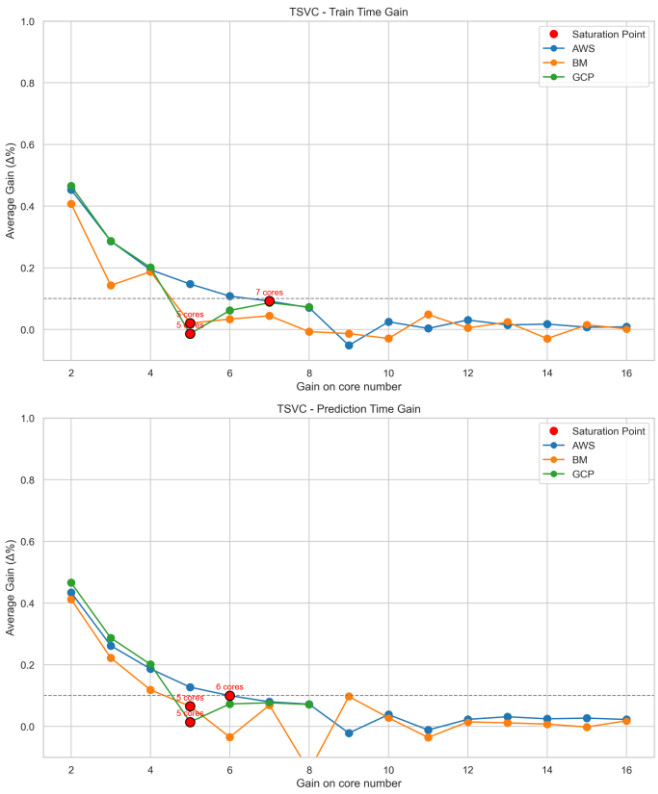

Fig. 5. Gain and saturation for ETC algorithm.

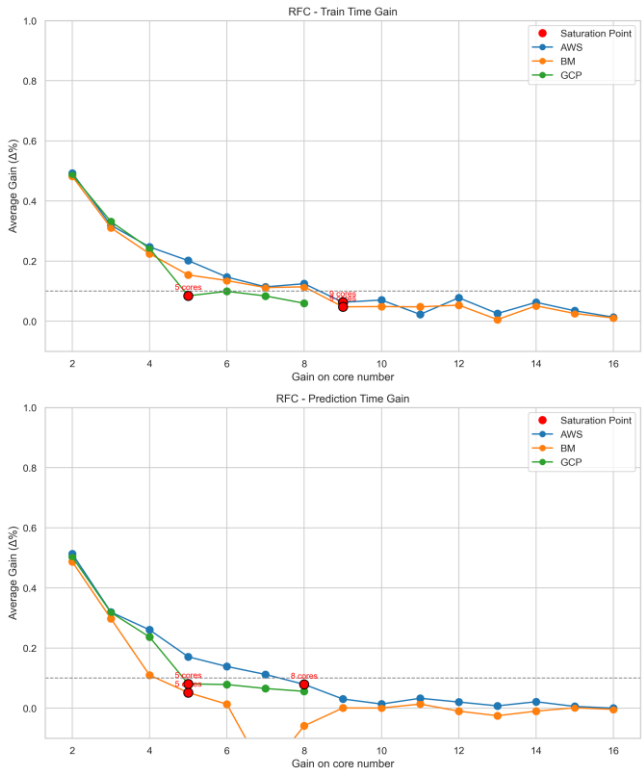

Fig. 6. Gain and saturation for RFC algorithm

We will provide a more detailed analysis of gain and saturation in the next section.

V. Discussion

A. Algorithm scalability

The results of our study clearly demonstrate that not all of the tested algorithms exhibit true scalability with respect to the number of virtual CPU cores (vCPUs). As indicated in the algorithm documentation, some methods perform computations in a purely sequential manner, meaning they are not designed to scale with additional cores. Among the tested algorithms, Extra Trees Classifier (ETC), Support Vector Ma-chine Classifier (TSVC), and Random Forest Classifier (RFC) show a clear improvement in both training and prediction times as the number of CPU cores increases:

- RFC and ETC are ensemble methods based on generating multiple independent decision trees. Each tree can be trained independently in parallel, which allows these algorithms to effectively utilize the available CPU computational resources.
- TSVC (ThunderSVM) is a library specifically optimized for multi-core CPU execution. It leverages the OpenMP parallelization framework, enabling the algorithm to distribute computations across multiple threads. As a result, increasing the number of available CPU cores has a noticeable and measurable impact on execution time, as reflected in the performance plots.

In contrast, the remaining algorithms—KNC, KNC_2, SGDC, HistGBC, and HistGBC_2—do not exhibit significant scalability for either the training or prediction phases. Nevertheless, all methods produced stable results across the conducted measurements:

- KNC and KNC_2: These algorithms support parallel processing via the *n_jobs* parameter; however, this only applies to the prediction phase. Despite enabling this functionality during testing, no substantial performance improvement was observed. The overhead associated with parallelism appears to offset any potential time savings in this case.
- SGDC (Stochastic Gradient Descent Classifier): While the *n_jobs* parameter is supported for parallelization, it is primarily intended to accelerate regression or multi-class classification tasks. As this study focused on a binary classification problem, the parameter had no measurable effect on either training or prediction times.
- HistGBC and HistGBC_2: These algorithms utilize the OpenMP parallelization library internally; however, the implementation of the *n_jobs* parameter differs from other scikit-learn methods. Despite theoretical support for parallelism, the algorithms did not demonstrate significant scaling behavior within the tested con-figuration.

B. Optimal number of VCPUs - gain and saturation points

By calculating the gain variable and analyzing the execution time plots, we can identify the optimal number of virtual CPU cores (vCPUs) for each algorithm. In this study, a reduction in execution time of less than 10% between consecutive core counts is considered economically insignificant, and thus the corresponding number of cores is defined as the saturation point.

Extra Trees Classifier (ETC):
- Training: Average gain drops below 10% for each added core at 5 cores calculations (GC), 6 cores (BM) and 8 cores (AWS).
- Prediction: Saturation points are: 6 (gc), 4 (BM), 7 (AWS).

Support Vector Machine Classifier (TSVC):
- Training: Saturation points are: 5 (GC), 5 (BM), 7 (AWS).
- Prediction: Saturation points are: 5 (GC), 5 (BM), 8 (AWS).

Random Forest Classifier (RFC):
- Training: Saturation points are: 5 (GC), 8 (BM), 8 (AWS).
- Prediction: Saturation points are: 5 (GC), 5 (BM), 8 (AWS).

For the remaining algorithms, which did not exhibit significant scalability, the saturation point occurs at 2 cores, reflecting purely sequential or near-sequential execution behavior. The only exception is the KNC algorithm during the prediction phase, where a slight improvement was observed with up to 3 cores, beyond which no further time reduction was achieved.

C. Platform differences

- BM platform was generally the fastest in cases when algorithms were not showing any scalability trends. That is because those algorithms were calculated only on 1 core. In this case, BM presents the strongest benchmark score.
- AWS shows the most stable results, having the lowest standard deviation score.
- The mean accuracy of the tested algorithms did not vary between platforms or with different numbers of CPU cores. This consistency in accuracy validates the correctness and reliability of the implementation across all environments.

D. Result summary

- As expected, the non-normalized execution times on the BM platform were generally shorter compared to the cloud platforms, primarily due to the high single-core performance of the local CPU. Conversely, the CPUs installed in cloud virtual machines are server-class processors optimized for total multi-core computational throughput rather than single-thread performance. Cloud platforms, therefore, exhibit a performance advantage for algorithms that effectively utilize multiple cores.
- For scalable algorithms, the BM platform demonstrates a significant performance advantage when using a lower number of cores. However, this ad-vantage diminishes as the number of cores increases.
- The Time Gain plots are the final proof for the presence or lack of scalability of the given algorithm.
- To facilitate a meaningful evaluation of scalability, we introduced the concept of the saturation point, defined as the number of cores beyond which execution time improvement falls below 10%. Beyond this threshold, further increasing the number of cores is considered economically inefficient, particularly in cloud environments where pricing is closely tied to the number of allocated cores.
- For non-scalable algorithms, the saturation point consistently occurs at 2 cores, indicating that increasing the number of cores beyond this has no significant impact on performance.
- For scalable algorithms, saturation points were observed to fall between the 4th and 9th cores, depending on the specific platform and algorithm. Based on the 10% gain threshold, utilizing more than 9 cores may not yield sufficient performance improvements to justify the additional cost, especially when selecting cloud virtual machines where pricing scales with core count.
- One contributing factor to these observations may be the use of hyper-threading, which can inflate the apparent number of available cores without delivering equivalent physical performance.
- Notably, the AWS platform consistently reached saturation points at higher core counts compared to other platforms, indicating superior scalability under the tested conditions.
- Table III summarizes best obtained execution times (both for training and prediction) together with the name of the platform. The last column provides the number of vCPUs – as calculated by the saturation & gain formula.

TABLE III
SUMMARY OF PLATFORM RESULTS AND COMPARISON, TIMES ARE NORMALIZED FOR PERFORMANCE COEFFICIENT

Algorithm	Minimal train time (best platform)	Minimal prediction time	Optimal number of vCPUs (train / predict)
ETC	1,84 (AWS)	1,8 (AWS)	8 / 7
TSVC	288,9 (AWS)	349,5 (AWS)	7 / 6
RFC	6,96 (BM)	1,0 (AWS)	5 / 8

CONCLUSION

To summarize, the primary consideration when selecting a cloud platform for AI computation should be the evaluation of the algorithm's scalability. This assessment depends not only on the type of algorithm but also on the nature of the problem and the characteristics of the dataset. Hyper-threading should not be assumed to be as efficient as scaling with dedicated vCPUs. Finally, if performance stability is as important as computation runtime, AWS has demonstrated higher efficiency; however, this may not hold true for different sets of algorithms.

As outlined in this study, our benchmarks were limited to a selected group of classification algorithms, with an intentional exclusion of neural network models, which represent a distinct class of machine learning methods that require specialized hardware, such as GPUs or TPUs, for optimal performance.

Potential extensions of this work include:

- Non-binary classification tasks, where some of the tested algorithms may exhibit different scalability characteristics, including greater parallelism potential.
- GPU-accelerated algorithms, such as deep learning models or large language models (LLMs), are becoming increasingly relevant for both research and practical AI applications.

Additionally, future experiments could broaden the scope of platforms evaluated to include other major public cloud providers, such as Microsoft Azure, Oracle Cloud, or emerging providers, to enable a more comprehensive comparison across the industry.

ACKNOWLEDGEMENTS

We are very grateful to anonymous reviewers for their appropriate and constructive suggestions to improve this paper.

APPENDIX A
APPLICATION, PACKAGE AND LIBRARY VERSIONS USED BY THE BENCHMARK SUITE.

Application / Distribution name	Description	Platform	Version
Linux / Ubuntu 24.04	Operating system	BM	Ubuntu 24.04,2 LTS kernel 6.11.0-19
Linux / Amazon Linux 2		AWS	Amazon Linux 2, kernel 5.1.230 Virtualizer: Amazon
Linux / CentOS Stream 9		GC	CentOS Stream 9 kernel 5.14.0-539 Virtualizer: KVM
Python	Programming language necessary for implementation of ML algorithms, calculation of tables and plot drawings.	BM	3.12.7
		AWS	3.12.9
		GC	3.9.21
Bash	System built-in shell / command interpreter – Required for writing starting script for Virtual Machines.	BM	5.2.21(1)
		AWS	4.2.46
		GC	5.1.8(1)
PIP	Python Package Manager - required for install and update additional Python packaged.	BM	25.0.1
		AWS	25.0.1
		GC	25.0.1
scikit-learn (Python package)	Machine learning algorithms optimized implementations library.	BM	1.6.0
		AWS	1.6.0
		GC	1.6.1
ThunderSVM-cpu (Python package)	SVC parallelized implementation.	BM	0.2.0
		AWS	0.2.0
		GC	0.2.0

REFERENCES

[1] P. Borra, “Comparison and analysis of leading cloud service providers (AWS, Azure and GCP)”, in International Journal of Advanced Research in Engineering and Technology (IJARET) Volume, 15, 266-278, 2024. https://doi.org/10.17605/OSF.IO/T2DHW

[2] M. Armbrust, A. Fox, A., et al., “A view of cloud computing”, in Communications of the ACM, 53(4), 50-58. 2010. https://doi.org/10.1145/1721654.1721672

[3] A. Rashid, A., A. Chaturvedi, “Cloud computing characteristics and services: a brief review”, in International Journal of Computer Sciences and Engineering, 7(2), 421-426, 2019. https://doi.org/10.26438/ijcse/v7i2.421426

[4] Q. Zhang, I. Cheng, R. Boutaba, “Cloud computing: state-of-the-art and research challenges”, in Journal of internet services and applications, 1, 7-18, 2010. https://doi.org/10.1007/s13174-010-0007-6

[5] Amazon Web Services, “AWS Cloud Credits for Research”, Retrieved from https://aws.amazon.com/grants/, 2023.

[6] M. Goswami, “Challenges and Solutions in Integrating AI with Multi-Cloud Architectures”, in International Journal of Enhanced Research in Management & Computer Applications ISSN, 2319-747, 2021.

[7] P. Leitner, J. Cito, “Patterns in the chaos—a study of performance variation and predictability in public IAaS clouds “, in ACM Transactions on Internet Technology (TOIT), 16(3), 1-23, 2016. https://doi.org/10.1145/2885497

[8] Sadooghi, I. Martin, et al., “Understanding the performance and potential of cloud computing for scientific applications”, in IEEE Transactions on Cloud Computing, 5(2), 358-371, 2015. https://doi.ieeecomputersociety.org/10.1109/TCC.2015.2404821

[9] J. Ericson, M. Mohammadian, F. Santana, “Analysis of performance variability in public cloud computing”, in 2017 IEEE International Conference on Information Reuse and Integration (IRI) (pp. 308-314). IEEE, 2017. https://doi.org/10.1109/IRI.2017.47

[10] OpenML, "Higgs Boson dataset description," [Online]. Available: https://www.openml.org/search?type=data&sort=runs&id=44129&status=active , [Accessed 20.01.2025]

[11] F. Pedregosa, G. Varoquaux, et al., “Scikit-learn: Machine learning in Python”, in the Journal of machine Learning Research, 12, 2825-2830, 2011.

[12] Amazon AWS, "AWS EC2 Instance Types," [Online]. Available: https://aws.amazon.com/ec2/instance-types/c6a/ . [Accessed 25.03.2025].

[13] Amazon AWS, "AWS EC2 CPU Options," [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/cpu-options-

supported-instances-values.html#cpu-options-compute-optimized [Accessed 25.03.2025].

[14] Amazon AWS, "AWS Amazon Linux 2 - Product Landing Page," [Online]. Available: https://aws.amazon.com/amazon-linux-2 [Accessed 25 03 2025].

[15] Google Inc., "Google Compute Optimized Machines," [Online]. Available: https://cloud.google.com/compute/docs/compute-optimized-machines#c2_machine_types [Accessed 25 03 2025].

[16] The CentOS Project, "Official CentOS Project Website," [Online]. Available: https://www.centos.org/ [Accessed 01.04.2025].

[17] PassMark, "CPU Benchmark Comparison," [Online]. Available: https://www.cpubenchmark.net/compare/3896vs4539vs6160/Intel-i7-11700K-vs-Intel-Xeon-Gold-6253CL-vs-AMD-EPYC-7R13-64-Core [Accessed 25 03 2025].

[18] D. M. Powers, “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation”, arXiv preprint arXiv:2010.16061, 2020.