Machine Learning to Predict Risk Management Applications Performance

Laura De Giorgi
Department of Electronics
Information and Bioengineering
Politecnico Di Milano
Tinvention s.r.l.
Email: laura.degiorgi@polimi.it

Danilo Ardagna
Department of Electronics
Information and Bioengineering
Politecnico Di Milano
Email: danilo.ardagna@polimi.it

***Abstract*—Machine Learning is increasingly crucial for predicting application performance, offering a black-box approach that does not require a deep understanding of the application internal workings. This method enables accurate predictions without delving into complex system models. Our study utilized ML to forecast the execution time of an industrial application dealing with risk measures as part of the Solvency II regulations for insurance companies. By conducting a comparative analysis of multiple models, XGBoost was identified as the most effective, achieving a Mean Absolute Percentage Error of 23%. The results demonstrated robust accuracy for intermediate durations, though limitations were observed for shorter and significantly longer times due to data scarcity. Overall, this study highlights the significant potential of ML in improving prediction accuracy for complex industrial applications, offering valuable insights for resource allocation and performance management.**

***Index Terms*—Performance, Machine Learning, XGBoost**

I. Introduction

Predicting software execution time and performance metrics is essential in software-related endeavors, particularly with established Service Level Agreements (SLAs). Accurate resource allocation based on expected workloads is crucial for job scheduling and data center management. The complexity of modern software, including big data and High-Performance Computing (HPC) applications, challenges traditional analytical methods. These applications involve multiple software layers and operate across cluster nodes, making performance analysis intricate. In such scenarios, black-box techniques like Machine Learning (ML) are preferred due to their ability to map input/output relationships to performance indicators without requiring internal system knowledge. This study compares several ML models developed to predict the execution time of an industrial risk calculation application. Starting from simpler cases and progressively addressing more complex scenarios, our approach was incremental. The application processes various financial reports, requiring a predictive model capable of handling variability in the report numbers processed by individual requests. Our dataset spans from September 2022 to February 2024, containing 43,480 runs. Initial analyses considered executions requiring only one report, extending to runs requiring up to 100 reports. This methodology yielded an average percent error of 11% for one-report cases and 23% for 100-report cases. This paper is organized as follows. Section II reviews current Machine Learning (ML) approaches for forecasting ICT system performance. Section III provides a brief overview of the application functionality. In Section IV, statistical analysis reports the distribution of run times and the number of reports in our historical dataset. Section V details the methodology and experimental results, explaining how the variable demands associated with each execution were addressed. Finally, conclusions are drawn in Section VI.

II. Related work

ML is integral in forecasting ICT system performance, with applications ranging from video streaming platforms, where it evaluates service quality through QoD (Quality of Delivery) metrics [4], to cloud systems, AI models, communication networks, and FaaS systems. Maros et al. [3] used ML to predict Apache Spark job performance, outperforming native Spark models for various workloads. In cloud-based IoT management, Nawrocki and Osypanka [15] combined multiple ML models with anomaly detection to ensure compliance with QoS constraints. Jialun et al. [5] improved resource efficiency in cloud datacenters using TCN and GNN (temporal convolutional network and graph neural network) for workload prediction. Kirchoff et al. [6] compared different ML techniques for HTTP server workload prediction, highlighting their predictive capabilities. Guanba et al. [7] optimized resource allocation in cloud datacenters using TCNs and GNNs, and introduced the FaaSDeliver framework, which applies a Tree-structured Parzen Estimator (TPE) to efficiently distribute FaaS functions, significantly reducing execution costs. In hybrid SDN (Software-Defined Networking) architectures, Chandra et al. [11] enhanced routing efficiency with a bio-inspired smell detection algorithm, outperforming conventional algorithms and improving network reliability. Despite the proliferation of AutoML, which automates ML workflows and offers various libraries for building systems [8], existing software often requires coding or ML expertise and lacks specific focus on application performance analysis. This work employs aMM-Library [1], an open-source Python library, to effectively train performance models, streamlining the process of selecting and optimizing ML algorithms for performance prediction tasks.

III. Application

In this Section, we will provide an overview of the application under study. The application primarily focuses on computing risk measures within the context of an Internal Model implementation of the Solvency II regulations for insurance companies [16]. It covers the main risk categories: Financial and Credit Risk, Life Underwriting Risk, Non-Life Underwriting Risk, and Operational Risk across the entire

979-8-3503-6104-9/24/$31.00 ©2024 IEEE

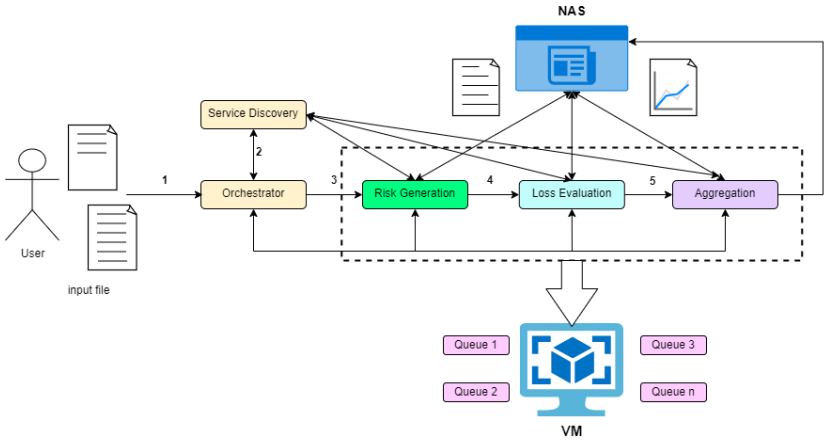

Fig. 1. Application architecture.

company portfolio. Users can request different types of reports, and each user has the flexibility to request an arbitrary number of reports during each run. A simplified representation of the application is shown in Figure 1. The application consists of two main steps: Loss Simulation, divided into the Risk Generation and Loss Evaluation Phases, and Aggregation. It operates on multiple queues, each with distinct resource allocations, ensuring efficient processing. Each phase is run on diverse virtual machines (VMs) within each queue, with its infrastructure. The input is a text file containing specific financial report requests. This Section is structured as follows: Section III-A shows the application structure, describing all the phases that constitute it, while Section III-B describes the different queues on which the application runs.

A. *Phases*

The process implemented by the application consists of two consecutive macro-phases: Loss Simulation and Aggregation. The Loss Simulation phase comprises two steps: Risk Generation and Loss Evaluation. Simulating risk scenarios, such as market curves and volatilities, is crucial in Risk Generation. This task involves sampling from an $N(0, C)$ probability distribution, where C is a user-defined n-dimensional correlation matrix. The risks result from applying a suitable transformation to each standard variable. The Loss Evaluation step uses the results from the previous step, evaluating polynomial functions, whose variables are the generated risks, to approximate the value of a portfolio for each simulated risk scenario. The Aggregation phase produces aggregated values and measures as reports. It is the most complex due to several factors. The number of requested reports varies, posing a challenge in handling a variable set of features for ML models. Unlike the first two phases, this phase involves many calculations among shared reports and varied calculations for different report types. Consequently, the dependency of execution time on input data becomes nonlinear. Some computations are universal across all reports, others specific to certain reports, while others diverge based on the report type. Unfortunately, the code lacks proper instrumentation, so we do not have times for individual phases and report generation.

B. *Queues*

The application operates on several queues, each tailored with distinct resource allocations for every phase of its operation and specified permissions for user access. In our analysis, we study a configuration set up with 6 queues. Users are directed to specific queues based on priority, ensuring those with higher importance encounter shorter waiting periods. This prioritization directly impacts the duration between job submission and actual processing. For each queue, the application tailors a virtual machine to accommodate every phase (i.e., risk generation, loss evaluation, and aggregation). Each virtual machine handles a specific phase with an adjustable number of associated threads and virtual cores. The most complex phase of the application is the aggregation phase, which has received the most attention throughout the article to understand how to construct the model. Since it is not possible to instrument this phase, additional effort is required to effectively manage it within the model. Furthermore, optimizing resource allocation necessitates establishing criteria for processing runs to supersede the existing 6-queue structure. However, to achieve this, we had to focus on the total execution times of the application, as partial phase times were not available.

IV. Historical Data Statistical Analysis

In this section, we present analyses based on the available data. The dataset comprises historical records of the application's past executions. Specifically, we analyzed 43,480 runs performed between September 2022 and February 2024. In Figure 2, the cumulative distribution plot illustrates the

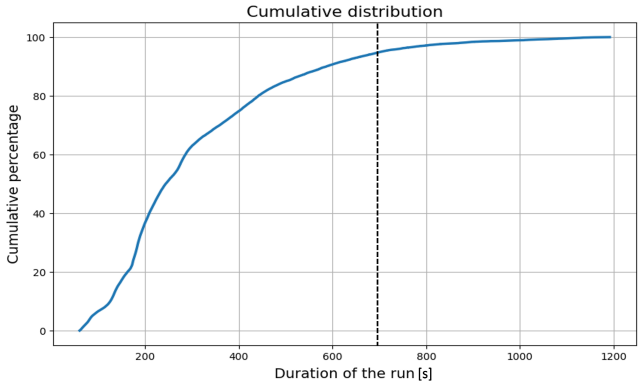

Fig. 2. Duration of the runs.

duration of all recorded runs. Runs exceeding 700 seconds constitute only 10% of the dataset. Given their minimal representation, these longer runs lack sufficient data for a predictive

model to accurately learn their behavior. Thus, we excluded these longer runs from our analyses. By concentrating on runs shorter than 700 seconds (39,532 in total), we aim to ensure a more statistically robust and reliable dataset for our predictive modeling efforts. We then analyzed the frequency of runs associated with various report numbers to identify the most common scenarios and establish a course of action for further analysis. The distribution of requests per number of reports is depicted in Figure 3. A significant portion of the runs (21,778)

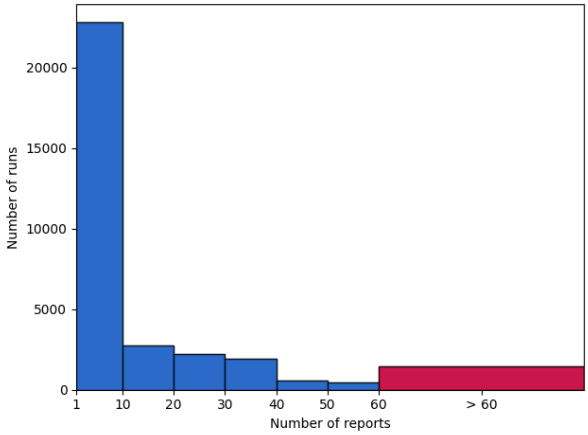

Fig. 3. Number of reports distribution.

require less than 10 reports, comprising 55.11% of the total. In particular, 11,603 (29.35%) requires only one report. Building on this insight, our analysis began by concentrating on runs requiring only a single report, before progressively advancing.

V. Machine Learning Performance Models

In this section, we discuss the data processing and model selection procedures. We outline the methods for preparing input data and comparing machine learning models using the `aMMLibrary`. It simplifies using various models and assists in feature selection via a configuration file, enabling Bayesian optimization to determine optimal hyperparameters. The library processes input from a CSV file, where each row details the execution of past runs. A sketch of the file structure is shown in Figure 7. Further details can be found in [1]. We outline the procedure conducted and display the results achieved for each case. Specifically, various strategies were employed to manage the variable number of features, depending on the required number of reports. Each report is associated with a specific set of features, so considering multiple reports increases the number of features proportionally. In Section V-B, we describe the approach taken when only one report is required. Section V-C covers the extension to two reports, while Section V-D discusses the scenario with up to five reports. Lastly, Section V-E delves into the scenario where requests extend up to 100 reports.

A. Data pre-processing and accuracy metrics

Historical data from 39,532 runs are available. Each run is associated with the following feature groups:

- **Hardware Configuration Setup**: Features related to the virtual machines used for each run (e.g., total number of virtual cores, total number of threads, etc.).
- **Loss Simulation Information**: Features related to the first phase of the application, such as the number of risk factors and scenarios where loss functions are evaluated.
- **Report Features**: Features related to the specific report required, such as the type of diversification and the number of computed percentiles.

An execution request is a text file containing the previously explained information. We processed the data to eliminate unnecessary information and identify outliers. The dataset contains extensive information, including various features linked to a single aggregation report request. However, not all these features impact computation time; their values do not lead to variations in total computation time. Including them would lead to significant model complexity without meaningful impacts on predictive performance. Therefore, we excluded certain features from the data, either because they were unrelated to all reports or did not impact computation time. We assumed some features had minimal impact because they were specific to only certain types of reports. Subsequently, we divided all reports into two families based on operations performed during their execution: *capital* templates and *scenario by scenario* templates. This categorization was driven by domain expertise and the different operations performed in the two cases.

- **Capital Templates**: Designed to generate total output based on scenarios, which are combinations of various risk factors. The templates calculate percentiles from the outputs of each scenario and retain specific values, allowing detailed analysis on a scenario-by-scenario basis.
- **Scenario by Scenario Templates**: These templates operate differently from the capital templates. For instance, the feature indicating the number of percentiles calculated is associated only with the capital type family, not the scenario-by-scenario family.

For both template families, we meticulously selected features based on the specific types of operations performed. Scenario by Scenario Templates incorporate 4 distinct features, and Capital Templates utilize 6, capturing the categorical characteristics of the reports they generate. To accurately predict the execution time for a report, 18 key features from the first three phases are included (see Figure 7), providing comprehensive coverage of information about scenarios, risk numbers, and loss functions evaluated during the simulation. These features are numerical, eliminating the need for encoding. Categorical features are coded by assigning integers reflecting their significant influence on execution times; features associated with shorter durations are assigned lower numbers. The secondary feature set, tailored to differences between report families, manages discrepancies by setting non-relevant features to zero for each specific family, ensuring complete integration of every phase into each run. For instance, the feature *tax* is encoded as PRE (1) or POST (2) based on when taxes are calculated during the process. Due to confidentiality agreements with our partners, we cannot disclose the entire list of features used. Each run combines 18 initial phase features with 8 additional features relevant to the type of report being generated. The selection of the most suitable predictive model was performed using aMLLibrary [1], which applies advanced stochastic methods in classical machine learning. We conducted a thor-

ough analysis of the models, optimizing their parameters through Bayesian optimization integrated with hyperopt [1]. Models such as Ridge Regression, Decision Trees, Random Forest, and XGBoost were extensively evaluated, with their hyper-parameter ranges detailed in Tables I, II, III, and IV.

Hyperparameter	Min	Max	Distribution
alpha	0.01	2	loguniform

TABLE I
HYPER-PARAMETER TUNING RIDGE REGRESSION

Hyperparameter	Min	Max	Distribution
min_sample_leaf	0.01	1	loguniform
min_sample_split	0.01	1	loguniform
max_depth	5	30	randint

TABLE II
HYPER-PARAMETER TUNING DECISION TREE

Hyperparameter	Min	Max	Distribution
n_estimators	50	350	randint
min_sample_leaf	0.01	1	loguniform
min_sample_split	0.01	1	loguniform
max_depth	5	30	randint

TABLE III
HYPER-PARAMETER TUNING RANDOM FOREST

Hyperparameter	Min	Max	Distribution
n_estimators	50	350	randint
alpha	0.01	2	loguniform
lambda	0.01	2	loguniform
min_child_weight	1	5	randint
gamma	0.01	2	loguniform
learning_rate	0.01	2	loguniform
max_depth	5	30	randint

TABLE IV
HYPER-PARAMETER TUNING XGBOOST

The mean absolute percentage error (MAPE) was used to compare the different models. It is defined as follows:

$$MAPE(y, \hat{y}) = \frac{1}{N} \sum_{n=1}^{N} \left| \frac{y_i - \hat{y}_i}{\hat{y}_i} \right| \quad (1)$$

where y is the vector of true values and $\hat{y}$ is the vector of predicted values by the ML model.

B. Case 1 report

First, we considered the case of runs that required exactly one report. This is because, in addition to being the base case, it also constitutes a relevant number of user-requested runs (29.35%). The initial step was to select the most suitable model for the problem being analyzed. Among the different models compared, XGBoost had the lowest error, highlighting the nonlinear nature of the problem. Hence, this was selected for subsequent analyses. The available data was divided into a training set comprising 80% of the data and a test set consisting of the remaining 20%. In the training phase, k-fold cross validation was performed considering 5 folds. The error obtained was 9.35% for the training set and 11.32% for the test set. The results obtained for this and the following cases are reported in Table V.

TABLE V
PERFORMANCE METRICS FOR DIFFERENT REPORT CASES

Report Case	All Features		XGBoost Feat. Selection	
	MAPE Train	MAPE Test	MAPE Train	MAPE Test
1 report	9.35%	11.32%	9.70%	10.82%
2 report	19.32%	22.60%	19.62%	22.92%
up to 5	12.00%	14.74%	12.36%	15.24%
up to 60	18.24%	22.34%	19.32%	23.06%

Next, we conducted feature selection to identify and remove any noisy features in the dataset. Given the accuracy of XGBoost in our previous tests, we proceeded with feature selection based on the feature importance weights provided by XGBoost. We systematically removed features starting with the one with the lowest weight. The results, illustrated in Figure 4, indicate that the error rate remains constant, even as we remove features with lower importance. It only begins to change after the removal of seven features. Consequently, we adopted the model with the seventh feature removed as our final model. This approach simplifies the model while maintaining the same error rate. Then, we considered the

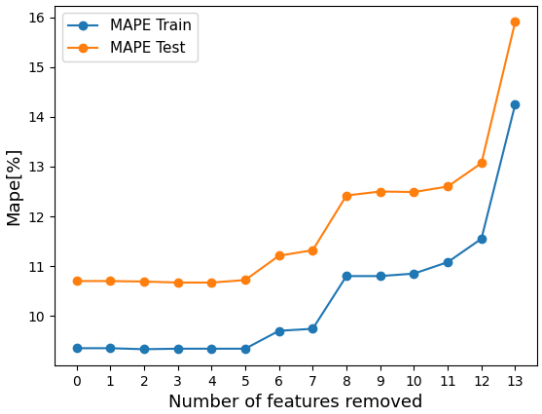

Fig. 4. MAPE feature selection - 1 report case

distribution of the errors w.r.t. the execution times and the distribution of the predicted times w.r.t. the real times as reported in Figures 5 and 6. Figure 5 displays both actual and predicted times, arranged in ascending order. Specifically, the data were first sorted by actual duration, and then the sorting index was plotted on the x-axis. What can be observed is that the model has better predictive capabilities for shorter times, while the error increases for longer durations. Consequently, we calculated the average percentage error for time intervals, as reported in Figure 6. The average percentage error remains below 22%, peaking around 700 seconds. The lowest error occurs for shorter runs.

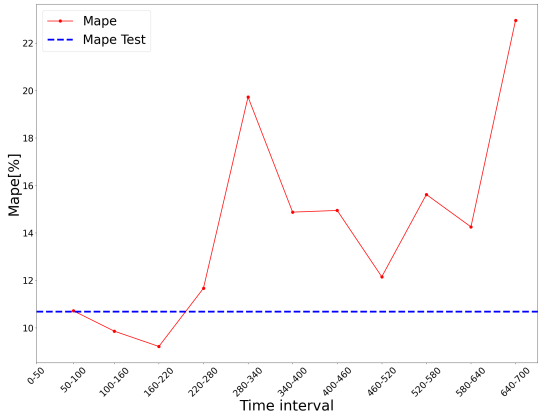

Fig. 5. Average error per interval - case 1 report.

C. Case 2 report

After analyzing the one-report case, we considered runs including exactly two reports, totaling 2,363 runs, i.e., 6.00% of the total. An initial challenge was that each run was linked to two reports. Given the minimal instrumentation of the application, only total process time was provided, not individual phase timings. Due to the application architecture, developing a model to predict time for the aggregation phase

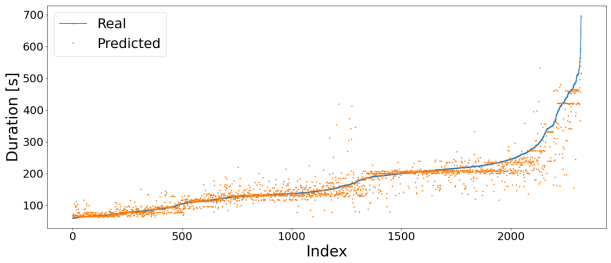

Fig. 6. Real and predicted values for test set - case 1 report.

was impractical. Thus, we predicted the total time, including report generation and preceding phases. To address this, we used features from the first phase as initial features and duplicated the report-specific features. This approach managed the dual-report association effectively while accommodating the application structure. A brief sketch of this process is shown in Figure 7. The results obtained in this case are reported in

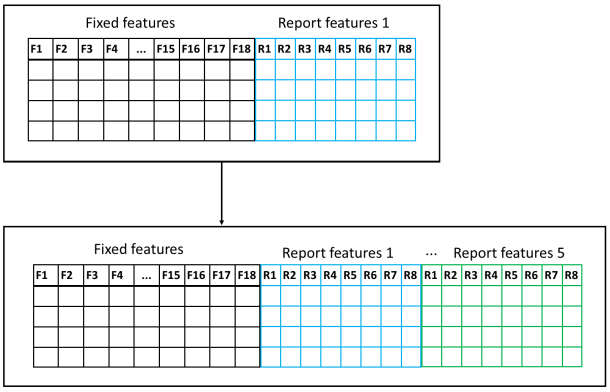

Fig. 7. Input file for the AMLLibrary for one and 5 report executions.

Table V, where the same features as in the one-report case are removed in feature selection. Further development included training models under mixed input files, containing data from both two-report and one-report runs. For runs requiring only one report, the features for the second report were set to zero. This approach significantly improved model accuracy, reducing the error to 10.84% for the training set and 13.20% for the test set. This improvement was due to the dataset higher dimensions and the inclusion of runs with only a single report, which demonstrated a lower error rate. To decide between adopting a mixed model or a model specific to the number of reports, a comprehensive comparative analysis was conducted. This included scenarios involving up to five reports to ensure robust decision-making based on extensive data evaluation.

D. Case up to 5 report

We followed the same procedure for runs requiring three to five reports. Initially, we trained separate models for each set, resulting in three distinct models: one for three reports, one for four reports, and one for five reports. Secondly, we considered a mixed case where the training set included all runs with up to five reports. For this scenario, if a report was not requested, its features were set to zero. The results for training and test errors can be found in Figure 8. We observed that errors in the test set remained consistent across both approaches, with occasional superior performance from the mixed case.

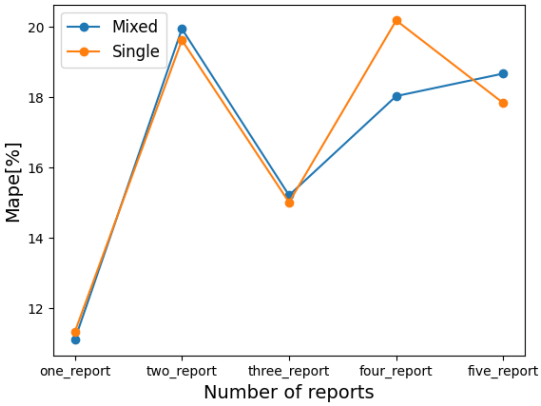

Fig. 8. Comparison single and mixed case with the full set of features.

Consequently, we determined the mixed case was optimal as it streamlines training and reduces the need to manage multiple models. The results, shown in Table V, indicated an error rate of 12.00% for training and 14.74% for testing. Feature selection was then performed, removing the same features as in the single-report case. The errors obtained were 12.36% for training and 15.24% for testing. We further analyzed the results by examining the error distribution relative to execution times, as shown in Figures 10. This analysis also considered the features removed for individual report cases. The final model chosen was the one with the reduced number of features. Analysis of these plots reveals that the mean error consistently stays below 30% exept for run with duration around 700s. Such an error margin is considered acceptable within the context of performance prediction, see, e.g., Lazowska et al. [12]. However, the error margin increases for longer durations due to a scarcity of data for these instances.

E. Case up to 60 report

Previous methods faced scalability issues as the number of required reports increased. For cases with up to 60 reports, each iteration's feature set includes 18 fixed features and 8 variables per report, resulting in up to 480 additional features. This complexity required a large training dataset and an ensemble approach. We divided the input into 12 subsets, each containing 5 reports and fixed features, employing the mixed model from Section V-D designed for sets up to 5 reports. This strategy ensured accurate predictions for each group, creating a new dataset where columns represented predictions for each block of 5 reports, with actual execution time as the target variable. We chose XGBoost for its superior accuracy in reducing mean absolute percentage error over linear models. The dataset was split with 80% for training and 20% for validation, ensuring balanced distribution across different report counts. Post-hyperparameter tuning, training and test results were 15.42% and 18.49%, respectively. We analyzed the mean error results as depicted in Figure 11. It is clear that errors tend to increase with longer run durations.

VI. Conclusion

In this paper, we developed a predictive model for estimating the run duration of an industrial application used in risk prediction. We demonstrated the models effectiveness by considering the mean absolute percentage error during training and testing.

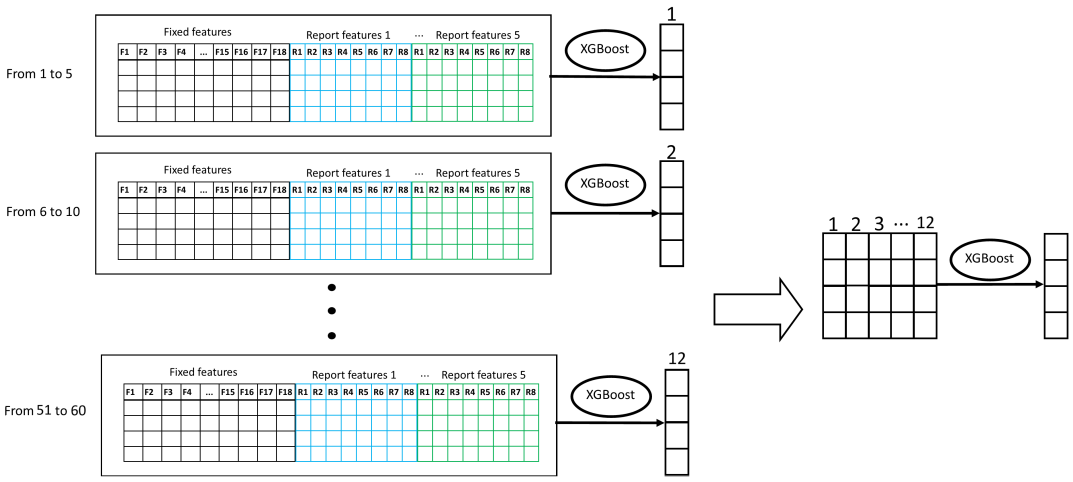

Fig. 9. Ensembling.

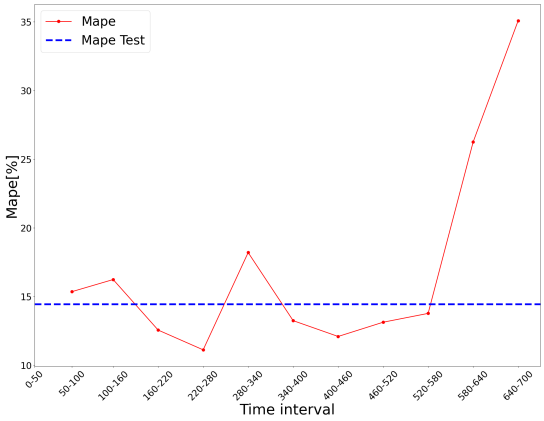

Fig. 10. Average error per interval - case up to 5 report.

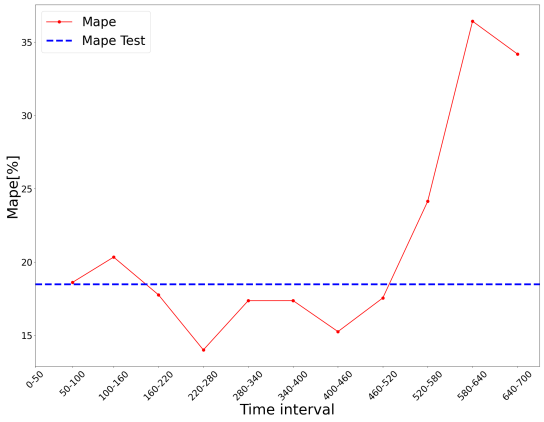

Fig. 11. Average error per interval - case up to 60 reports.

However, challenges persist with extended duration runs and handling variable report numbers more efficiently. Hence, we aim to explore advanced techniques such as transformers or state space models, suited for handling variable tokens and dimensions. The goal is to leverage this model by integrating it into a scheduling algorithm to efficiently reorganize the submitted jobs, decreasing users average waiting time.

ACKNOWLEDGMENT

We kindly thank Assicurazioni Generali for their support in accessing the data required for our analysis. We also extend our gratitude to Riccardo Lena for his valuable contribution.

REFERENCES

[1] Guindani Bruno, Lattuada Marco and Ardagna Danilo *AMLLibrary: An AutoML Approach for Performance Prediction*, 37th International Conference on Modelling and Simulation (ECMS)

[2] Didona, Diego and Romano, Paolo *Using analytical models to bootstrap machine learning performance predictors*, 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)

[3] Alexandre Maros, Fabricio Murai, Ana Paula Couto da Silva, Jussara M. Almeida, Marco Lattuada, Eugenio Gianniti, Marjan Hosseini, Danilo Ardagna *Machine Learning for Performance Prediction of Spark Cloud Applications*, 2019 IEEE 12th International Conference on Cloud Computing (CLOUD)

[4] George Margetis, Grigorios Tsagkatakis, Stefania Stamou and Constantine Stephanidis *Integrating Visual and Network Data with Deep Learning for Streaming Video Quality Assessment*, *Sensors*, 2023, Volume 23, Issue 8, Page 3998.

[5] Jialun Li, Diying Yang, Hairui Guo, Xuan Mo, Weigang Wu *Forecasting resource usage pattern changes in clouds via contrast graph-evolution learning*, : www.elsevier.com/locate/fgcs

[6] Kirchoff, Dionatra F and Xavier, Miguel and Mastella, Juliana and De Rose *A preliminary study of machine learning workload prediction techniques for cloud applications*, : 2019 27th Euromicro international conference on parallel, Distributed and Network-Based Processing (PDP)

[7] Guangba Yu, Pengfei Chen, Zibin Zheng, Jingrun Zhang, Xiaoyun Li, Zilong He *FaaSDeliver: Cost-Efficient and QoS-Aware Function Delivery in Computing Continuum* , : IEEE TRANSACTIONS ON SERVICES COMPUTING, VOL. 16, NO. 5, SEPTEMBER/OCTOBER 2023

[8] He, Xin and Zhao, Kaiyong and Chu, Xiaowen *AutoML: A survey of the state-of-the-art*, : Knowledge-Based Systems

[9] S. S. Vinod Chandra e S. Anand Hareendran *Modified smell detection algorithm for optimal paths engineering in hybrid SDN*, : Journal of Parallel and Distributed Computing

[10] Lazowska, Edward D and Zahorjan, John and Graham, G Scott and Sevcik, Kenneth C *Quantitative system performance: computer system analysis using queueing network models*, :Prentice-Hall, Inc.

[11] S. S. Vinod Chandra e S. Anand Hareendran *Modified smell detection algorithm for optimal paths engineering in hybrid SDN*, : Journal of Parallel and Distributed Computing

[12] Lazowska, Edward D e Zahorjan, John e Graham, G Scott e Sevcik, Kenneth C *Quantitative system performance: computer system analysis using queueing network models*, : Prentice-Hall

[13] Dietterich, Thomas G *Ensemble methods in machine learning*, : International workshop on multiple classifier systems

[14] Chen, Tianqi and Guestrin, Carlos *XGBoost: A scalable tree boosting system*, : Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining

[15] Nawrocki, Piotr and Osypanka, Patryk *Cloud resource demand prediction using machine learning in the context of qos parameters*: Journal of Grid Computing

[16] *https://www.eiopa.europa.eu/browse/regulation-and-policy/solvency-ii_en*