

# Machine Learning Based Resource Utilization Prediction in the Computing Continuum

Christian Bauer\*, Narges Mehran<sup>†</sup>, Radu Prodan<sup>‡</sup> and Dragi Kimovski<sup>§</sup>

University of Klagenfurt

Klagenfurt, Austria

Email: \*christian.bauer@aau.at, <sup>†</sup>narges.mehran@aau.at, <sup>‡</sup>radu.prodan@aau.at, <sup>§</sup>dragi.kimovski@aau.at

**Abstract**—This paper presents UtilML, a novel approach for tackling resource utilization prediction challenges in the computing continuum. UtilML leverages Long-Short-Term Memory (LSTM) neural networks, a machine learning technique, to forecast resource utilization accurately. The effectiveness of UtilML is demonstrated through its evaluation of data extracted from a real GPU cluster in a computing continuum infrastructure comprising more than 1800 computing devices. To assess the performance of UtilML, we compared it with two related approaches that utilize a Baseline-LSTM model. Furthermore, we analyzed the LSTM results against User-Predicted values provided by GPU cluster owners for task deployment with estimated allocation values. The results indicate that UtilML outperformed user predictions by 2% to 27% for CPU utilization prediction. For memory prediction, UtilML variants excelled, showing improvements of 17% to 20% compared to user predictions.

**Index Terms**—Utilization Prediction, Machine Learning, Computing Continuum, Cloud.

## I. INTRODUCTION

The pervasive influence of the Internet and the high availability of computational devices has extended its reach across all societal domains. Consequently, the volume of data generated and transported over the Internet network continues to grow exponentially. This encompasses various applications that generate vast amounts of data, including healthcare systems, smart cities, communications, media, and entertainment. However, much of this data is unstructured and requires processing to be transformed into usable and analyzable formats.

As the demand for processing vast amounts of data intensifies and more complex structures, like video streams, are transmitted over the Internet, robust computing resources become paramount. This led to the emergence of large consolidated computing infrastructures, such as the Cloud. The Cloud can rapidly scale application executions based on resource utilization spikes and deliver the necessary computing power for efficient analysis. Nevertheless, relying solely on a Cloud system for data processing has disadvantages, including unreliable Internet connections, unsecured data transfers, and high energy requirements.

These challenges gave rise to the Computing Continuum, encompassing resources from the Cloud to the network's edge [1]. The computing continuum addresses the drawbacks of relying solely on Cloud systems by moving the data processing services closer to the data sources. However, the Computing Continuum has a very high heterogeneity in terms of various

computing resources, usually under the control of different commercial or private entities.

Consequently, the high heterogeneity and different control domains adversely impact the overall utilization of resources within the Computing Continuum, resulting in decreased efficiency and suboptimal resource allocation [2]. Therefore, resource prediction based on machine learning can provide valuable insights and support informed decision-making by allowing organizations to plan better and allocate resources. However, providing accurate resource utilization estimates is a significant problem. The study by Thonglek et al. [3] found that tasks provided by users overestimate resource utilization by an average of over 30%. This leads to unnecessary resource allocation and instability in the infrastructure since applications may not be deployed on appropriate resources if all their capacity is already allocated.

Therefore, this paper introduces UtilML, a novel resource utilization prediction system that harnesses the power of long short-term memory (LSTM) machine learning model. To achieve this, we analyze real-data traces provided by Cloud and Edge providers to build fine-tuned predictors for application execution over the Computing Continuum. The aim is to improve existing prediction methods, such as user-provided estimations, which estimate the resource utilization required for task deployment. We use regression metrics to measure prediction performance and make informed decisions.

The main contributions of this paper are:

- Analysis of publicly available monitoring traces to enhance data quality, transform datasets, and design better-suited machine learning models.
- Development of a proof of concept machine learning approach UtilML that improves resource utilization prediction in the Computing Continuum compared to other methods.
- Evaluation of enhanced models and the UtilML based on data insights by using regression metrics to compare with existing prediction methods.

The paper has six sections. Section II surveys the state-of-the-art. Section III presents the UtilML approach, followed by an architecture design in Section IV. Section V describes the experimental evaluation. Section VI concludes the paper.

## II. RELATED WORK

Machine learning-based resource prediction consists of applying a variety of algorithms, including linear regression [4], decision tree [5], random forest [6], and neural networks [7]. This section reviews the state-of-the-art for LSTM and deep learning-based resource utilization prediction.

### A. LSTM-based resource prediction

Improving resource utilization is especially promising for large-scale computational systems such as data centers and Cloud infrastructures. A hybrid architecture, based on the convolutional neural network (CNN) and LSTM models, predicts the server load in a Cloud computing infrastructure [8]. The algorithm selection highly depends on the specific problem and the available data type.

Tuli et al. [9] proposed an LSTM-based straggler-task prediction method in the Cloud infrastructure. This method reduces the application response time and minimizes the service violation between the application and resource providers.

Thonglek et al. [3] designed a neural network model based on LSTM as a type of recurrent neural network (RNN) to predict resource allocation based on historical data. This model has two LSTM layers, each learning the relationship between *i*) allocation and usage, and *ii*) CPU and memory. It aims to improve resource utilization in data centers by predicting the required resource for each user's application.

### B. Deep learning-based resource prediction

Oren et al. [10] represents resource utilization as a combinatorial optimization problem and uses a set of Monte Carlo decision processes (MDP) [11], deep Q-learning and graph neural networks (GNN) models as heuristics to improve the resource utilization.

Ngo et al. [12] explored multiple deep neural networks (DNN) models with varying complexity for data anomaly detection of Internet of Thing (IoT) data. This method explores selecting one of the models to perform autonomous detection in the Edge or Cloud infrastructures. The devices utilized in the evaluations consist of NVIDIA Jetson-TX2 and Devbox with four GPU TitanX, respectively, as the Edge and Cloud data centers.

Similarly, Chen et al. [13], [14] proposed a learning-based method that allocates resources aiming to minimize average service completion time and power consumption. Intelligent resource allocation framework (iRAF) [13], as a multitask deep reinforcement learning-based algorithm, predicts the resource utilization based on the training data generated from the searching process of the Monte Carlo tree search (MCTS).

Lastly, Tan and Hu [15] applied a deep reinforcement learning model to formulate the resource allocation optimization problem, where the parameters of caching, computing, and communication are optimized jointly.

**Ambition:** The current resource utilization prediction approaches are created explicitly for large-scale data Cloud centers and do not adequately model the whole Computing Continuum. Furthermore, these works do not use regression

metrics and traces from different Cloud and Edge providers to improve the prediction model.

## III. UTILML UTILIZATION PREDICTION MODEL

This section elaborates on the *UtilML* prediction model that utilizes long short-term memory [16] method. This involves describing the different layers, the steps, and the operations required to train and improve the prediction model (see figure 1). The structure and number of nodes inside the LSTM model are defined while instantiating it with the parameters *input size*, *hidden size*, and *number of LSTM layers*. The input size is defined as the number of feature set columns identified in the data preprocessing step (see Section IV). The hidden size value in the parameters also refers to the hidden layer size of the LSTM model that is used to variate the model's size and capabilities to remember long-term patterns in time series data. Finally, the number of layers refers to how many stacked LSTM layers should be applied. When using stacked LSTM layers, the size of the hidden layers of the model is doubled. This also includes the model output, where the *forward* function returns both hidden state outputs. The reason for applying the forward function is to analyze the hidden states of the intermediate model layers.

Afterward, we define sequential layers for both LSTM models and use Leaky Rectified Linear Unit (LeakyReLU) as the activation function between the sequential layers. This is an extension of the widely used ReLU activation function for transforming the summed weighted input from a layer into the activation of a node or output for that input. The model of the LeakyReLU function activates a node based on the value of the range of its input [17]:

$$g(z) = \max(0, z) + \text{negative\_slope} \cdot \min(0, z),$$

where  $z$  is the input of the nonlinear activation  $g$ , and *negative\_slope* is a non-zero coefficient bounded to 0.01 that manages the function's slope in negative  $z$  values.

The base function ReLU, which does not include a negative slope, is widely used because of its computational simplicity and representational sparsity. While LeakyReLU is computationally more complex and has less representational sparsity, it was a suitable choice for UtilML because of the following advantages. The advantage of LeakyReLU, as opposed to ReLU, lies in its specific utility within neural networks with large amounts of negative values. LeakyReLU avoids the *saturation* problem, i.e., if the gradient is zero, no adjustment of parameters can be made as the direction of the adjustment is unknown. Thus, the learning stops.

Consequently, we apply batch normalization, which is a generalization method used to increase the stability and reduce the training time of a neural network layer by re-centering and re-scaling the layer's inputs. Batch normalization standardizes the inputs to a layer for each mini-batch, and the resulting increase in stability reduces the number of required training epochs to train deep neural networks. Normalizing the input batch also has a regularizing effect, which reduces

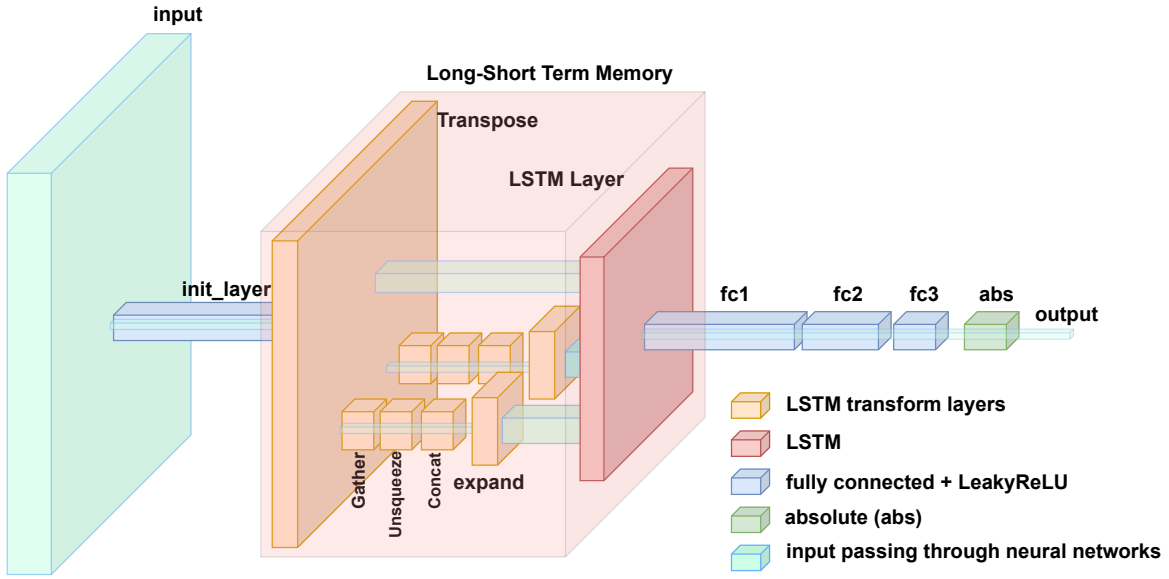


Fig. 1. LSTM Model.

generalization errors, similar to the usage of activation regularization. Batch normalization was proposed to mitigate internal covariate shifts [18]. This internal covariate shift occurs because inputs are forwarded to the neural network, and inputs from each layer have a corresponding distribution that affects the training process because of the randomness of the parameter initialization and input data [18]. Because of the resulting internal covariate shift, the succeeding network layers need to readjust their weights to update their state from the data forwarded from the proceeding layer with a bias regarding the distribution of activated nodes. This requirement to readjust to new distributions is even more severe in deep neural networks with many layers, and the increased number of layers amplifies this effect. Batch normalization reduces unwanted shifts propagated by layers, speeds up the training, and produces more reliable and general models.

Finally, we create the fully connected linear layers. The linear layers or *fully-connected layers* connect every input neuron to each output neuron and are commonly used in neural networks. A linear layer is defined by two parameters: the number of inputs and the number of outputs. Operations such as forward propagation, activation gradient computation, and weight gradient computations are directly expressed as matrix multiplications. The pivotal components of the prediction approach are the LSTM and sequential layers for estimating CPU and memory utilization. These layers are engaged in the process of forwarding inputs through each subsequent layer until the estimation is derived.

As depicted in figure 1, the model receives feature sets extracted from tasks as input first passed to an initial layer. The initial layer uses the input information to generate the initial hidden and internal states of the LSTM model. Thereafter, these states and input feature sets are passed to the LSTM components to predict CPU and memory utilization, respectively.

The feature set is comprised of the capacity of the CPU and memory of a computing cluster, as well as the corresponding predicted values provided by users. Furthermore, we analyzed the different task types and separated them into 12 groups. The groups are represented as one-hot encoded columns in the feature set, and each task in the dataset is assigned to a single one-hot encoded column.

The general LSTM prediction model is split into two smaller LSTM components that each predict the utilization of one resource unit. A resource unit is either the CPU usage or the allocated memory.

The input is then transformed with multiple operations inside the LSTM model (see Figure 1) and passed from one stacked LSTM layer to the next. The final LSTM output of either resource unit is then sent to fully-connected sequential layers that use batch normalization as the generalization strategy. Since resource utilization cannot be negative, we omit negative values by calculating the absolute estimation output.

#### IV. UTILIZATION PREDICTION ARCHITECTURE

The section illustrates the architecture of the UtilML resources utilization prediction approach, depicted in figure 2.

*a) Monitoring:* utilizes the Prometheus [19] and Netdata [20] monitoring tools to extract data related to the utilization of the resources and the application execution in the Computing Continuum. The monitoring component collects real-time metrics, including CPU, memory, storage, and network bandwidth usage, without interrupting any of the application's runtime.

*b) Historical resource utilization database:* is populated using the monitoring data, which is essential for the training and fine-tuning of the LSTM prediction model. The database

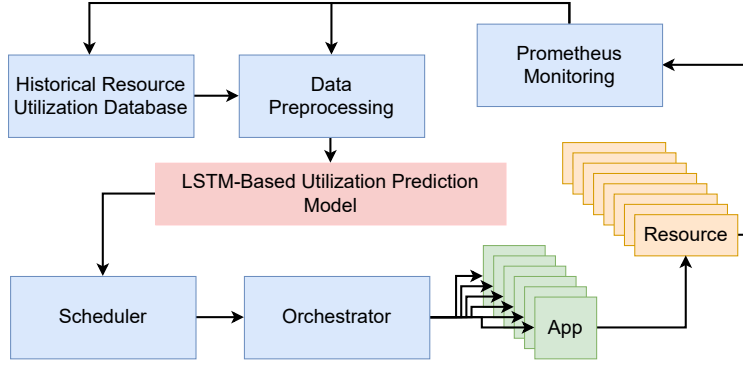


Fig. 2. UtilML Utilization Prediction Architecture.

TABLE I  
TRAINING TESTBED DESIGN.

<b>CPU</b>	Intel Xeon Gold 5218 Processor
<b>GPU</b>	2 × NVIDIA Quadro RTX 8000
<b>Memory</b>	754 GB
<b>Operating System</b>	Ubuntu Linux 18.04 LTS

aggregates the monitoring data from multiple executions and provides it for preprocessing.

*c) Data preprocessing:* transforms the data coming directly from the monitoring component or the historical database into the appropriate format for either model training, inference, or model fine-tuning. The data preprocessing includes feature extraction, feature cleansing, and feature analysis.

*d) LSTM utilization prediction model:* utilizes the pre-processed data to perform inference and identify possible future resources' under- or over-utilization. Thereafter, the model can be integrated with different schedulers or orchestrators to manage the resources based on the predicted utilization information.

*e) Scheduler:* maps the application to the resources. It supports multiple different scheduling algorithms. However, for the given purpose, we utilize the matching-theory-based  $C^3$ -MATCH scheduler [2].

*f) Orchestrator:* deploys and manages the execution of the applications based on the information from the LSTM model and the scheduler with the main purpose of optimizing the execution and avoiding resource under- or over-utilization. For the given component, we utilize Kubernetes orchestrator.

## V. EXPERIMENTAL EVALUATION

This section describes the experimental design and the evaluation results in detail.

### A. Training testbed

We conduct the training of the UtilML model on a testbed consisting of a GPU-enabled physical machine provided by the University of Klagenfurt. The machine specifications are summarized in Table I.

### B. Trace and monitoring data

We utilize monitoring traces from the Alibaba Cloud provider [21] to train and validate the UtilML variants. For the evaluation of the prediction performance of the LSTM model configurations, we choose the GPU cluster dataset `cluster-trace-gpu-v2020`. We analyze and characterize the GPU cluster presented in [22]. In addition, for the inference process, we use monitoring data from the GPU server described in Section V-A collected using the Prometheus and Netdata tools (see Section IV).

### C. Prediction Metrics

In this section, we used forecasting measurement metrics [23] to evaluate the accuracy of the trained models. These metrics compare different models, assess the predictions' quality, and identify improvement areas. We use three prediction metrics of RMSE, MAPE, and sMAPE to evaluate the models:

*a) Root Mean Square Error (RMSE):* [24] measures the average magnitude of the error between the predicted and actual values and provides a metric to compare the magnitude of the errors in different models:

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}},$$

where  $N$  denotes the number of data points (monitoring values for the CPU and memory). Moreover,  $Actual_i$  and  $Predicted_i$ , respectively, define the actual and predicted values of the data point  $i$ .

*b) Mean Absolute Percentage Error (MAPE):* [25] measures the average percentage difference between the predicted and actual values:

$$MAPE = \frac{1}{N} \cdot \sum_{i=1}^N \left| \frac{Actual_i - Predicted_i}{Actual_i} \right| \cdot 100,$$

where MAPE provides a percentage error, which simplifies interpreting and comparing forecasting models' accuracy.

*c) Symmetric Mean Absolute Percentage Error (sMAPE):* [26] measures the average percentage difference between the predicted and actual values and is symmetrical since it treats positive and negative errors equally. In our evaluation of the

TABLE II  
COMPARISON OF LSTM VARIANTS AND User-Predicted CPU [%].

	Actual	UtilML-LSTM	Baseline-LSTM	User
mean	516.073	454.205	392.630	632.809
std	881.832	579.213	705.771	496.245
min	1.023	2.395	3.030	5.000
25%	103.632	129.884	97.586	400.000
50%	208.749	249.392	118.014	600.000
75%	528.076	662.490	281.472	600.000
max	7790.371	5634.635	5793.996	6400.000

prediction model’s performance, this is essential as over- and under-utilization are present in all prediction variants:

$$sMAPE = \frac{1}{N} \cdot \sum_{i=1}^N \left| \frac{Actual_i - Predicted_i}{(Actual_i + Predicted_i) \div 2} \right| \cdot 100,$$

where sMAPE provides a percentage error similar to MAPE, simplifying the interpretation and comparing forecasting models’ accuracy. Unlike MAPE, sMAPE is symmetrical, as it remains defined even when the actual value is zero. Another distinction is its reduced sensitivity to outliers and anomalous data points. Therefore, a better sMAPE result is preferable to a better MAPE result in comparing estimations.

#### D. Results

In this section, we compare the UtilML utilisation prediction model, referred to as UtilML-LSTM, with the Baseline-LSTM variant [3] as well as the predictions provided by the users of the GPU cluster for both CPU and memory utilization.

1) *CPU prediction analysis*: In this section, we compare the CPU prediction of the UtilML-LSTM and Baseline-LSTM model as well as the *user*-provided prediction (also referred to as *user predicted* or *user*) are presented in Tables II and III. We observe in Table II, the *mean* of the LSTM model that additionally receives information about the task types is closer to the actual *mean* and even is the closest compared to the other prediction methods. The *standard deviation* of the UtilML-LSTM is smaller than the value of the Baseline-LSTM but closer to the actual compared to the value of the user prediction. The UtilML-LSTM predicted *minimum* value of 2.395 is closer to the actual value of 1.023 compared to both the Baseline-LSTM and user-provided prediction. In addition, the three quartiles of the UtilML-LSTM variant in combination are the closest to the actual quartiles, which indicates that the model enhanced its learning from the dataset distribution compared to the currently presented predictions. The *maximum* value predicted by the UtilML-LSTM is the farthest from the actual value, which could result from similar tasks with low utilization as opposed to the tasks with the maximum utilization. Therefore, the task knowledge alone might not be sufficient to predict utilization spikes in a GPU cluster.

The UtilML-LSTM outperformed the Baseline-LSTM and User-Predicted models regarding the RMSE metric. However, it performed better than the User-Predicted model regarding the MAPE metric and worse by an error increase

TABLE III  
REGRESSION METRICS BY UtilML-LSTM AND User-Predicted CPU [%].

	RMSE	MAPE	sMAPE	OE	UE
UtilML-LSTM	688.089	212.796	83.017	48.14	51.86
Baseline-LSTM	797.289	206.062	85.626	41.94	58.06
User-Predicted	812.497	355.049	89.466	72.80	27.20

TABLE IV  
COMPARISON OF LSTM VARIANTS AND User-Predicted MEMORY [GB].

	Actual	UtilML-LSTM	Baseline-LSTM	User
mean	17.203	29.134	29.904	26.895
std	74.761	63.342	39.634	15.259
min	0.003	0.156	1.951	2.000
25%	2.160	4.537	22.178	14.648
50%	7.699	14.679	24.620	29.297
75%	15.976	27.924	24.620	29.297
max	1992.484	698.983	550.056	146.484

of 7% compared to Baseline-LSTM. An explanation for the lower performance is the distribution of over- and under-estimated (also referred to as the over- and under-allocated memory) values in the datasets. As discussed in section V-C0b, the MAPE metric is biased towards higher values; therefore, overestimated values result in a higher overall loss. The UtilML-LSTM improved the sMAPE metric compared to both the Baseline-LSTM and the user prediction methods. Moreover, the distribution of over- and underestimated (referred to as OE and UE, respectively) values of the UtilML-LSTM predictions is close to equal.

Therefore, providing categorical data can positively influence the prediction performance of the machine learning model. Moreover, we observe this in the predictions of the UtilML-LSTM model, in which the additional knowledge about the task types enables the ML model to create more accurate predictions of CPU utilization.

2) *Memory prediction analysis*: The memory prediction analysis of the UtilML-LSTM model and the user-provided prediction are compared in Tables IV and V. The *mean* memory utilization prediction of both the LSTM variants performed slightly worse than the prediction provided by users. All three prediction variants were approximately 9 GB – 12 GB overestimated on average. For the *standard deviation*, the UtilML-LSTM prediction did achieve the closest result, which indicates its capability of adapting the prediction more actively than the other two prediction variants. Similarly, the UtilML-LSTM model did achieve to predict a *minimum* value close to the actual one as opposed to the other two variants. In the first and second quartiles, the UtilML-LSTM performed more accurately than the Baseline-LSTM and the user predictions but did perform worse than the Baseline-LSTM in the third quartile. The actual *maximum* could be predicted better with the UtilML-LSTM, yet the estimation is still approximately 3 times lower.

As described in Table V, the UtilML-LSTM improved the

TABLE V  
REGRESSION METRICS BY UtilML-LSTM AND User-Predicted MEMORY [GB].

	RMSE	MAPE	sMAPE	OE	UE
UtilML-LSTM	61.897	32287.182	119.715	56.66	43.34
Baseline-LSTM	77.902	7711.391	109.870	77.8	22.2
User-Predicted	73.853	3672.256	97.613	80.80	19.20

RMSE metric among the three variants. However, for both the *MAPE* and *sMAPE* metrics, it did perform worse. An explanation for the high *MAPE* value of the UtilML-LSTM is the high *standard deviation* in combination with the too high *mean* value. In other words, the UtilML-LSTM, while being more proactive in predicting utilization spikes, also is more likely to predict higher utilization. Predicting utilization spikes, when none occur, results in such high loss values of the *MAPE*.

## VI. CONCLUSION

This paper proposes a novel UtilML approach for solving large-scale computing clusters' resource utilization prediction problem. It utilizes a machine learning model based on LSTM neural networks and constantly defines a set of regression metrics to improve the prediction. We evaluated the UtilML approach on a local GPU server with real-life data containing monitoring traces of a GPU cluster of approximately 1800 devices.

We compared the acquired inference results with a dataset containing the CPU and memory resource utilization. Moreover, we analyzed the LSTM variant results with the User-Predicted values the GPU cluster owners provided for each task to be deployed with the estimated values. We identified that UtilML outperformed the user predictions for CPU utilization by 2% – 27%. Moreover, for memory utilization, the UtilML variants performed 17% – 20% better, and the other variants were slightly 5% worse than the User-Predicted method.

However, the UtilML model could not accurately predict both CPU and memory spikes in computing clusters. One reason is the need to be fed information about the number of task instances that will be run. Therefore, in future, we plan to address the issue if insufficient information on the task.

## ACKNOWLEDGMENT

This work received funding from the *European Union's* research and innovation program, grant 101016835 (Horizon 2020, DataCloud) and the *Austrian Research Promotion Agency (FFG)*, grant agreement FO999897846 (GAIA).

## REFERENCES

- [1] D. Kimovski, R. Mathá, J. Hammer, N. Mehran, H. Hellwagner, and R. Prodan, "Cloud, fog or edge: Where to compute?," *IEEE Internet Computing*, 2021.
- [2] N. Mehran, Z. N. Samani, D. Kimovski, and R. Prodan, "Matching-based scheduling of asynchronous data processing workflows on the computing continuum," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 58–70, IEEE, 2022.
- [3] K. Thonglek, K. Ichikawa, K. Takahashi, H. Iida, and C. Nakasan, "Improving Resource Utilization in Data Centers using an LSTM-based Prediction Model," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 1–8, Sept. 2019.
- [4] S. Weisberg, *Applied Linear Regression*, vol. 528. John Wiley & Sons, 2005.
- [5] S. B. Kotsiantis, "Decision trees: A recent overview," *Artificial Intelligence Review*, vol. 39, pp. 261–283, 2013.
- [6] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.
- [7] J. A. Anderson, *An Introduction to Neural Networks*. MIT press, 1995.
- [8] E. Patel and D. S. Kushwaha, "A hybrid CNN-LSTM model for predicting server load in cloud computing," *The Journal of Supercomputing*, vol. 78, no. 8, pp. 1–30, 2022.
- [9] S. Tuli, S. S. Gill, P. Garraghan, R. Buyya, G. Casale, and N. Jennings, "Start: Straggler prediction and mitigation for cloud computing environments using encoder lstm networks," *IEEE Transactions on Services Computing*, 2021.
- [10] J. Oren, C. Ross, M. Lefarov, F. Richter, A. Taitler, Z. Feldman, D. Di Castro, and C. Daniel, "SOLO: Search online, learn offline for combinatorial optimization problems," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 12, pp. 97–105, 2021.
- [11] F. James, "Monte Carlo theory and practice," *Reports on progress in Physics*, vol. 43, no. 9, p. 1145, 1980.
- [12] M. V. Ngo, T. Luo, H. Chaouchi, and T. Q. Quek, "Contextual-bandit anomaly detection for IoT data in distributed hierarchical edge computing," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1227–1230, IEEE, 2020.
- [13] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "iRAF: A deep reinforcement learning approach for collaborative mobile edge computing IoT networks," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 7011–7024, 2019.
- [14] J. Chen, S. Chen, S. Luo, Q. Wang, B. Cao, and X. Li, "An intelligent task offloading algorithm (iTOA) for UAV edge computing network," *Digital Communications and Networks*, vol. 6, no. 4, pp. 433–443, 2020.
- [15] R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 10190–10203, 2018.
- [16] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997.
- [17] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.
- [18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.
- [19] Prometheus, "Overview — Prometheus." <https://prometheus.io/docs/introduction/overview/>.
- [20] Netdata, "Getting started — Learn Netdata." <https://learn.netdata.cloud/docs/getting-started/>, Mar. 2023.
- [21] Q. Weng and H. Ding, "Alibaba Cluster Trace Program." Alibaba, Apr. 2023. <https://github.com/alibaba/clusterdata>.
- [22] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp. 945–960, USENIX Association, 2022.
- [23] A. Botchkarev, "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology," *arXiv preprint arXiv:1809.03006*, 2018.
- [24] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)?—Arguments against avoiding RMSE in the literature," *Geoscientific model development*, vol. 7, no. 3, pp. 1247–1250, 2014.
- [25] A. De Myttenaere, B. Golden, B. Le Grand, and F. Rossi, "Mean absolute percentage error for regression models," *Neurocomputing*, vol. 192, pp. 38–48, 2016.
- [26] V. Kreinovich, H. T. Nguyen, and R. Ouncharoen, "How to estimate forecasting quality: A system-motivated derivation of symmetric mean absolute percentage error (SMAPE) and other similar characteristics," 2014.