Time Series Predictions for Cloud Workloads: A Comprehensive Evaluation

Anna Lackinger‡, Andrea Morichetta‡, Schahram Dustdar‡,
‡TU Wien {a.lackinger, a.morichetta, dustdar}@dsg.tuwien.ac.at

2024 IEEE International Conference on Service-Oriented System Engineering (SOSE) | 979-8-3315-3958-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/SOSE62363.2024.00011

***Abstract*—Predicting workloads in cloud computing is a significant challenge due to their complex, multidimensional, and highly variable nature. Assessing the accuracy of these predictions is critical, as this directly impacts management decisions that the infrastructure manager has to take in real-time to optimize resource utilization and meet Service Level Agreements (SLAs). Researchers and practitioners approached workload prediction as a time series problem using both statistical and Machine Learning (ML) methods. However, due to the volatile nature of the resource utilization patterns and the fact that new workloads constantly appear, developing robust solutions is still an open challenge. Furthermore, current solutions often only predict one single workload, completely lacking the capability for generalizing over new workload time series. These approaches fully counter the advantages of leveraging complex methods, as for every new workload, they need to train, validate, and test a separate model. In this paper, we offer a generalizable approach based on Transfer Learning concepts. We comprehensively evaluate different methods (statistical, Machine Learning, and Deep Learning based) for predicting the resource utilization of cloud workloads by testing them on new, unobserved time series data, thereby assessing their potential for practical applications through Transfer Learning. Specifically, we inspect the algorithms' performance in predicting one or multiple timestamps ahead, considering both CPU and memory usage. Our main findings indicate that the Deep Learning methods Long Short-Term Memory (LSTM) and Transformer are the most suitable methods for predicting different metrics and timestamps ahead for test data. Through Transfer Learning principles, we investigate how the models' performance varies with out-of-distribution data. In predicting new, unseen workloads, complex models show some limits, even if LSTM still proves to work in specific cases. Overall, our research offers valuable insights that can help infrastructure managers make correct design decisions when predicting cloud workload resource usage.**

***Index Terms*—workload prediction, cloud, LSTM, Transformer, Transfer Learning**

I. Introduction

In recent years, cloud platforms have gained a leading role in providing infrastructure and computing resources to companies and organizations. Cloud computing offers a key advantage by replacing fixed capital investment with pay-on-demand services, enabling firms to scale and reorganize according to their requirements [9]. Contextually, the critical nature of the applications and services deployed on these platforms, such as Amazon Web Services (AWS), Microsoft Azure, IBM Cloud, Alibaba Cloud, Google Cloud Platform (GCP), and many more, calls for accurate measures for timely management, as most companies consider managing cloud spend as one of their top challenges.[1] Finding ways to act before malfunctions or inefficient actions occur is essential, not only from a business perspective but also from the security and reliability ones. In addition, accurate prediction of workloads can reduce waste in the cloud infrastructure and thus reduce carbon emissions into the environment [12]. Therefore, a notable challenge in cloud computing lies in effectively processing and predicting the runtime behavior of the applications' workload. This task is critical for service providers and infrastructure managers, seeking to optimize resource allocation, enhance system performance, and ensure seamless scalability and responsiveness to dynamic workloads [30]. Specifically, from the infrastructure manager's perspective, having accurate predictions of resource consumption of applications workload is essential for precise resource provisioning and efficient management of cloud workloads. In this context, the predictions can directly fuel autoscaling strategies that are indeed often modeled as a time series problem [20], [40]. However, prevailing methods are unable to effectively predict the complex and highly variable nature of cloud workloads, resulting in either a waste of resources or an inability to meet Service Level Agreements (SLAs) [6]. Therefore, it is essential to give the infrastructure manager performative tools to develop accurate strategies, providing highly precise predictions.

Traditionally, time series forecasting has been treated through statistical methods such as Autoregressive Integrated Moving Average (ARIMA) [4]. However, the context of cloud computing monitoring brings stringent requirements of rapidly adapting to changes. Therefore, in the literature, these methods have been overpowered by, in turn, Machine Learning (ML) and Deep Learning (DL) models [13], [39]. In general, the latter approaches often build on regression methods, heuristic algorithms, and traditional Neural Network (NN). In particular, NN solutions are networks with a limited number of layers, such as Multi-Layer Perceptron (MLP) and Radial Basis Function (RBF) [43]. Still, even if these methods can be effective for identifying clear patterns in workloads, they struggle to capture complex correlations. In the latter case, more complex NN models can be necessary to minimize the prediction errors [43]. At the same time, such complex models tend to suffer from overfitting and have higher computational complexity in training and inference.

Furthermore, most of the proposed solutions for cloud workload resource usage forecasting limit their methodology in observing and predicting a single workload. This approach, as highlighted by Morichetta et al. in [29], has three clear

[1] https://info.flexera.com/CM-REPORT-State-of-the-Cloud

979-8-3315-3958-0/24/$31.00 ©2024 IEEE
DOI: 10.1109/SOSE62363.2024.00007

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 24,2024 at 16:08:07 UTC from IEEE Xplore. Restrictions apply.

disadvantages. First, a new model must be trained for different groups of workloads or individual workloads, which involves both time and resource costs. Secondly, these methods cannot be utilized in the data collection phase. Thirdly, this strategy only works for long-lasting workloads, as for short ones, there would not be enough time to collect sufficient data. In addition, previous contributions lack an accurate analysis of the results and a clear definition of the approach configuration.

With this work, we aim to cover the existing gaps, **forecasting** the resource requirements of new jobs **without** relying on **extensive data collection** for model training. Furthermore, our objective is to identify a method that accurately predicts resource requirements and is generalizable, enabling practical application across various resource metrics and job types without requiring extensive historical data to train a new model or enough information to cluster jobs. This approach is inspired by Transfer Learning, i.e., using source models and information to improve a target predictive function, as described by Weiss et al. [42]. In this paper, we leverage Transfer Learning by training methods on existing jobs and applying these methods to predict new jobs. In this way, we use the knowledge we have gained from previous resource utilization data to improve the predictions for new jobs. We provide to the infrastructure manager **clear** and **practical tools** for workload resource usage forecasting by comprehensively exploring a range of time series prediction methods, considering various categories, and testing prominent algorithms. We evaluate these approaches primarily on their prediction error minimization for upcoming job resource requirements and their computational complexity. This assessment extends beyond single-step predictions to include the more challenging task of multi-step ahead prediction, which proves beneficial as it provides more time to allocate resources by revealing the short-term future workload behavior. Furthermore, by testing various techniques, we seek to identify the most generalizable model that can reliably predict resource utilization for various jobs, contributing to a more efficient and adaptable resource provisioning system. Our approach is shown in Figure 1.

Our contributions are as follows:

- Conversely from related work, we offer full transparency and reproducibility, clearly showing all the steps of our data selection and preparation process in detail.
- We conduct an experimental comparison of different approaches for predicting the workload of new jobs, not only one but *up to six*[2] timestamps ahead. The rationale is to capture immediate and near-future behaviors.
- Unlike previous approaches, we evaluate the method's performance by comparing their predictive capabilities using the Mean Squared Error (MSE) on both a test set and out-of-distribution data, ensuring the generalisability of the results and, therefore, the *transferability* of the developed solutions.

[2]Far-future predictions are not of interest as the boundary conditions can rapidly change in such dynamic environments and re-forecasting might become necessary.

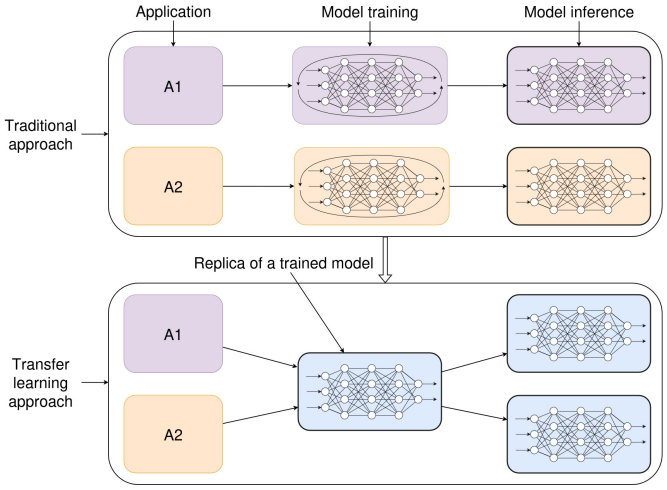


Fig. 1: Workload prediction approaches for two applications A1 and A2. The upper subgraphic illustrates the conventional method, whereas the lower subfigure depicts *our Transfer Learning* approach. In the upper subfigure, a new model is trained for each application, the trained model is then used for the prediction. In the lower one, a replica of a trained model is used for the prediction for each workload, which is not retrained for each new application.

- We provide our code as open source,[3] allowing other researchers to utilize it for their purposes.

The rest of the paper is organized as follows: Section II gives an overview of related work in the field. Section III presents the detailed steps of our methodology. Then, we evaluate the obtained results in section IV. Finally, the last section V presents the concluding remarks and the summary of the results.

II. Related Work

Cloud workload prediction typically involves forecasting how resource utilization patterns evolve, particularly for single tasks or jobs. Contrarily to related work, this paper aims to compare the efficiency of different methods on unseen jobs and jobs that are out-of-distribution data, meaning that they have different patterns.

In state-of-the-art work for time series prediction statistical methods, such as ARIMA are usually either used as a benchmark method [17], [28], [46] to compare results or they are combined with Machine Learning or Deep Learning methods, such as in [23].

Since the advent of popular libraries for implementing neural networks, many works leveraged methods based on Recurrent Neural Networks (RNNs). Solid results have confirmed their suitability and effectiveness in the context of time series forecasts for resource utilization [46]. More recent works frequently employ Long Short-Term Memory (LSTM) or Bi-Directional Long Short Term Memory (Bi-LSTM), which are a specific implementation of RNNs as these methods also

[3]*https://github.com/Lacki28/time-series-optimization*

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 24,2024 at 16:08:07 UTC from IEEE Xplore. Restrictions apply.

demonstrate strong performance in handling non-short-term time sequences [3], [17], [30], [39], [45]. The mentioned methods all use the workload of the Google cluster trace [34], but most of the mentioned methods need historical data of the workload for training to predict its continuation [3], [17], [30], [39], [46] and do not consider out-of-distribution data. Moreover, some methods only consider the prediction of the CPU [39], [45].

A different approach is to treat the workload sequence prediction as a translation problem, which can be realized using an Attention Seq2Seq-based technique, as in [2]. The mentioned work uses the Google cluster trace to predict memory and CPU values, but they also use historical data of the workload to predict its continuation and do not consider out-of-distribution data.

State-of-the-art applications frequently use hybrid methods that strategically combine a variety of Deep Learning techniques [6], [21], [31]. Beyond the field of standalone Deep Learning, there is a growing trend towards integrating these advanced techniques with traditional statistical methods [10], [21]. This fusion aims to leverage the strengths of both paradigms and create a comprehensive and robust framework that exploits the advantages of both techniques. However, these novel methods frequently lack thorough implementation details, which poses a challenge for others wishing to replicate or build on their findings. Moreover, the findings are usually not tested on out-of-distribution (OOD) data. For this paper, we have decided not to include a hybrid approach in our experiments, as the focus of our work is the comprehensive evaluation of standard methods on how well they predict unseen jobs and how accurately they predict multiple timestamps ahead.

Some time series prediction methods not only forecast the workload for the upcoming timestamp but also extend predictions for multiple timestamps ahead, providing sufficient time to schedule tasks based on the anticipated workload [10], [13], [29], [39], [45]. However, the effectiveness of these approaches is assessed based on their ability to predict future trends in specific time series data, given their history.

Moreover, some papers focus on the application of prediction methods, such as [27], [41]. Tour et al. present a scalable mechanism for monitoring and forecasting resource utilization in large-scale distributed systems [41], which integrates adaptive measurement transmission of local nodes, dynamic clustering, and temporal forecasting algorithms for a group of nodes, demonstrating improved performance in real-world experiments compared to baseline methods, with potential applications in resource allocation and system management. For each cluster, a different model is used, which is retrained every 24 hours. In their work, they consider CPU and memory values, but their models are trained on the history of the workload to predict its continuation. Moreover, they only compare two prediction methods, including a LSTM and an ARIMA.

Another area of research that is becoming increasingly popular in the field of workload prediction is Transfer Learning. Liu et al. [7] use an LSTM ensemble method for accurate workload prediction. However, this work only predicts short jobs that run for less than eight hours and they also use only a very small subset of 9 jobs. Singh et al. [37] combine feature extraction, clustering, and a deep learning model (N-Beats) to forecast workload. They analyze and cluster using machine similarities so that the prediction model can predict future usage without retraining, thereby reducing computational costs. They build generic models to forecast workload/CPU utilization across an entire cluster instead of individual machines. To train the models, 70% of the machines are used for training and the rest is then used for testing. However, they only consider CPU usage and they only predict one timestamp into the future.

III. Methodology

Prevalent experiments and solutions for time series forecasting in cloud workload data rely on the assumption that the model trained on a single job can generalize well. This is, however, a weak premise, as the model would likely need to be adapted – retrained, or at least updated in its hyperparameters. Furthermore, most of the previous contributions do not mention the details about their models' features, making it difficult, or impossible, to evaluate the validity of their solutions. This work aims to provide a comprehensive study and integrate different prediction methods to evaluate their effectiveness in predicting new time series data. Figure 2 depicts the individual steps of our methodology, which we will detail in this section.

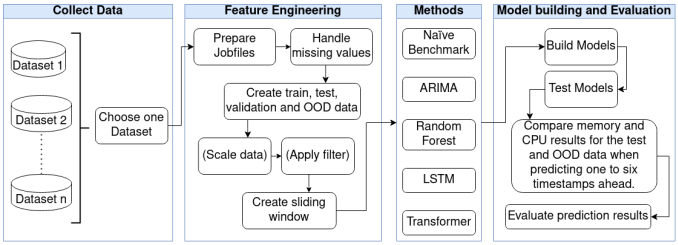


Fig. 2: Overview of our methodology.

A. Motivation

We aim at offering a comprehensive investigation of algorithms for time series forecasting of resource usage patterns for cloud workloads. In particular, we focus on investigating the capability of these methods of working with OOD data. A previous contribution in that sense is provided by Gao et al. [13]. In their work, they identify three main classes of algorithms that are typically employed for time series prediction. Despite providing an accurate analysis that offers an *m-gap* prediction of future resource usage, a model valid for each type of workload is built as well as specific models for workload categories identified through unsupervised learning. However, in a real case, it is not always possible to identify patterns of usage for all the workloads, especially in the early stages of the adoption of a platform, where different kinds of applications might submit their jobs. Therefore, it is essential to analyze the capability of these models over different, unseen

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 24,2024 at 16:08:07 UTC from IEEE Xplore. Restrictions apply.

data. Furthermore, despite the thorough analysis, the work in [13] does not explicitly discuss the adopted parameters for the models and the detail of which data they used, e.g., which jobs, with which characteristics. These aspects limit the comprehension and evaluation of the proposed results. Conversely, we provide a fully transparent and clear analysis of the approaches considered.

B. Selection of Predictive Approaches

To capture the most accurate analysis of time series prediction for cloud workloads, it is essential to include the most relevant approaches. In particular, we inspect relevant methods belonging to the three main categories identified by [13], namely statistical methods, ML algorithms, and DL techniques. In addition, we consider a Naïve Benchmark (NB). This method considers the value of the last observation as the prediction for the forthcoming resource usage steps. With NB we aim at having a ground truth, i.e., we want to verify that the more sophisticated methods outperform this vanilla approach. Despite facile, this "naïve" functioning is commonly used for comparison with advanced algorithms, including DL ones [11], [28]. Therefore, NB represents a useful benchmark.

Regarding the three categories, concerning statistical methods, we consider ARIMA, a widely used approach known for its effectiveness in predicting time series data, which is often used as a benchmark [17], [28], [30], [46]. For the ML class, we consider Random Forest (RF), which utilizes Bootstrap Aggregating, to construct multiple models and compute their collective average, thereby mitigating variance, as outlined in [36]. Being an ensemble ML algorithm, one benefit of the RF is that instead of conducting only one local search, multiple local searches from various starting points are conducted and the results are averaged. Moreover, it exhibits a reduced tendency to overfit [5]. Its performance has been shown in [22], where the RF outperformed an ARIMA when predicting influenza outbreak time series data. For the DL approaches, we have chosen a LSTM, which is frequently recurring as a solution for cloud workload resource usage prediction, making it an essential benchmark [17], [30], [39], [45]. The LSTM is a popular choice for time-series prediction because it has a memory cell that maintains state over time and non-linear gating units controlling information flow, which prevents the vanishing gradient problem that commonly appears in Recurrent Neural Network (RNN). A more detailed description of the LSTM architecture is given by Song et al. in [39] and in [25]. However, the last development and evolution of Transformer methods cannot be omitted [29], [35], [38]. The architecture of a Transformer consists of the encoder and the decoder, each of which can consist of several identical layers [29]. A crucial element is the integration of the attention mechanism, which enables the model to focus on relevant segments of the input sequence. To evaluate its potential in predicting new time series data, we include a Transformer in our analysis.

C. Selection of the Dataset

To evaluate the effectiveness of each of the investigated methods, it is essential to select a suitable dataset, considering its content, size, and scientific relevance. Several real-world datasets are available for this purpose, including the Azure dataset [16], the Google cluster dataset [14], and the Alibaba dataset [15]. To identify patterns, we want to run tests with data from jobs with a minimum duration of two weeks. The Alibaba dataset was excluded due to its limited time period. A search on Google Scholar to compare the citations between the papers detailing the datasets revealed that the Google cluster trace [34] is more commonly used by other researchers with 1037 citations compared to 646 in Azure [8] as of April 22, 2024.

The details about the trace selected for our experiments are sourced from [34] and [25]. Beginning at 19:00 EDT on Sunday, May 1, 2011, in one of Google's production cluster cells located in the Eastern time zone of the United States, the clusterdata-2011-2 trace was gathered for 29 days. About 12.500 distinct machines are part of this cluster [14]. In the context of Google's infrastructure, a cluster consists of computers grouped in racks and interconnected via a high-bandwidth cluster network. In contrast, a cell is made up of several devices that are usually part of a single cluster. As a result, the cluster management system that oversees the assignment of tasks to the computers in the cell is shared by these machines. Work arrives at a cell in the form of jobs. A job consists of one or more tasks. Each task is associated with a set of resource requirements used in the scheduling process to assign the tasks to specific computers in the cell. Individual tasks represent a Linux program, which might be several processes that are intended to operate on a single system. There are six different data tables provided [34] [26] [25].

1) *Machine events table*, which records each machine's events and capacity;
2) *Machine attributes table*, which are key-value pairs representing machine properties;
3) *Job events table*, which describes jobs' status and how latency-sensitive it is;
4) *Task event table*, which describes tasks' status, priority, constraints, and each task's resource request in the cloud;
5) *Task constraints table*, which records constraints for exactly each task;
6) *Task resource usage table*, which shows each task's detailed records, such as start and end time, Central Processing Unit (CPU) usage, memory usage, etc.

To provide an overview of the dataset, we list the most important values in Table I, which is taken from [25].

D. Feature Engineering

In this paper, we will focus on the prediction of jobs that have high-priority tasks, since these tasks will get preference for resources over tasks with lower priority numbers [34]. This section aims to give a detailed description of how the data

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 24,2024 at 16:08:07 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Overview of the Google cluster trace, taken from [25].

Number of machines	12.583
Number of jobs	672.074
Number of tasks	25.424.731
Number of jobs with scheduling class 0	257.275
Number of jobs with scheduling class 1	215.110
Number of jobs with scheduling class 2	194.513
Number of jobs with scheduling class 3	5.179
Average number of tasks each job has	215
Number of machines a job uses on average	35

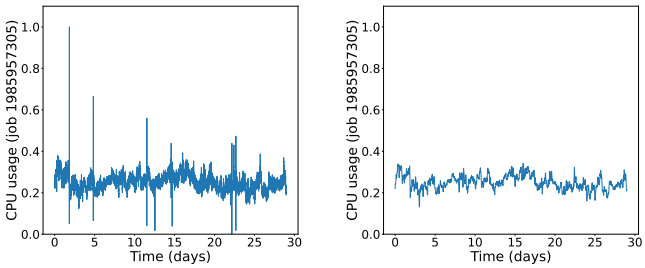
is chosen and prepared. There are 12 different priorities that range from 0 to 11 where 0 is the lowest priority [14]. Jobs that have priority 9 and above that are latency-sensitive should not be "evicted due to over-allocation of machine resources", according to the trace providers [33].

Our prediction targets are the memory and CPU since they are prominent metrics for understanding resource usage. The prediction of the CPU load is the target in multiple related papers [7], [21], [37], [39], [45]. Other works predict not only the CPU but also the memory, such as [2], [6], [29]–[31], [46].

For the preparation of the dataset, we follow the instructions taken from [25]. In total, 45.533 jobs contain high-priority tasks. For the training/test and validation data, we filter out jobs that have fewer than 4032 lines since these jobs run less than two weeks, which leaves us with 3793 jobs. For the OOD data, we select jobs with a different structure, that have a shorter duration. To be more precise, we select jobs with a duration of more than 1 hour but less than a week; therefore, we filter out files that have more than 2016 lines and less than 12. For this dataset we included jobs with high-priority tasks since they will get preference for resources over tasks with lower priority numbers, making their accurate prediction more important.

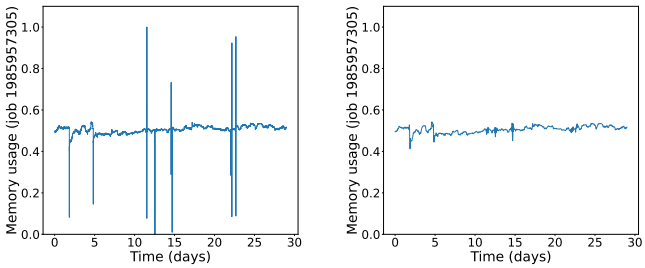
The subsequent phase involves dealing with missing values and preparing the datasets for the prediction. There are various techniques for dealing with missing values, which can be roughly divided into three types: Ignore, Delete, and Imputation. In this paper, we use the imputation approach as described in [32]. In particular, the k-nn imputation method is used because of its recognized effectiveness [1]. Since this method is very resource-intensive, an interpolation method is used that considers the nearest neighbors in both directions.

Afterward, we filter out rows that do not contain any meaningful data on resource consumption, such as cases in which the maximum CPU consumption is specified as 0. In order to align the tasks with a 5-minute interval and to facilitate subsequent grouping based on start times, the start times are rounded. After this phase, the tasks that start at the same time are summarized in a single line for all files, since we want to predict jobs and not individual tasks. The resource values in each line correspond to the cumulative total of all tasks. In the next step, a new column is then added indicating the number of tasks, followed by the inclusion of a column for the scheduling class in all files. After filtering files that do not have enough lines (the desired duration), the non-OOD dataset contains 2261 jobs, and the OOD dataset includes 3432 jobs. To prepare the data for the Machine and



(a) Normalized data (b) Savitzky-Golay filter

Fig. 3: Mean CPU usage of the job with the id 1985957305, after normalization with and without Savitzky-Golay filter spanning a duration of approximately 4 weeks. The y-scale in both images ranges from 0 to 1.0.



(a) Normalized data (b) Savitzky-Golay filter

Fig. 4: Canonical memory of the job with the id 1985957305, after normalization with and without Savitzky-Golay filter spanning a duration of approximately 4 weeks. The y-scale in both images ranges from 0 to 1.0.

Deep Learning models, we additionally used normalization and a Savitzky-Golay filter. This filter smooths a series, and it has been used in the papers [3], [10] and in the thesis [25]. The thesis showed, with several experiments, that the models achieved a higher accuracy when a Savitzky-Golay filter was used for the training data. We have used the same parameters for the filter as in [25] as we are working with similar data. Figure 3 illustrates the normalized mean CPU usage and Figure 4 depicts the canonical memory usage of a job, both before and after the application of a filter. Upon comparing the two images presented in Figure 3, it becomes evident that the application of this filter in the bottom Figure 3b effectively mitigated the noise inherent in the data, shown in 3a. The same pattern is shown in Figure 4 when comparing the two images 4a and 4b. This process enhances the visibility of the underlying patterns thereby facilitating the model in making predictions that are based on meaningful information and are not influenced by random noise.

To gain some understanding of the value distributions of the different datasets, we created violin plots, which are a combination of box plots and kernel density plots. These plots contain much of the information about boxplots and provide

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 24,2024 at 16:08:07 UTC from IEEE Xplore. Restrictions apply.

additional information about the shape of the distribution that is not obvious in boxplots [19]. In Figure 5 one can see a violin plot of the CPU value distribution of the different datasets. To better visualize the distribution, the extreme

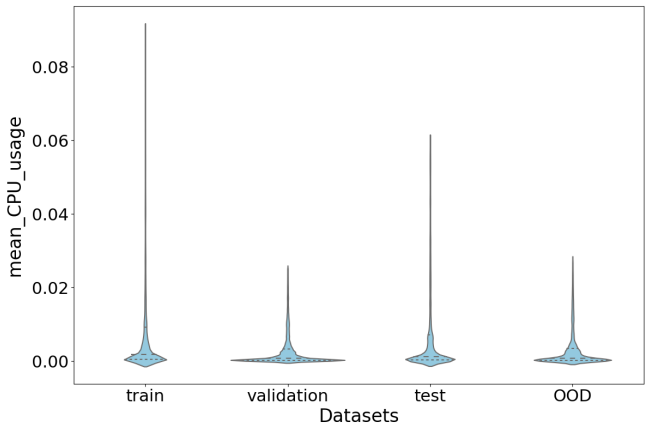


Fig. 5: Mean CPU usage distribution of the different datasets.

outliers have been removed. The maxima can be found in Table II. Looking at Figure 5 and Table II, one can see that the training data has higher CPU values than the other datasets. The validation data exhibits fewer instances of higher CPU values but demonstrates a high density at lower values.

TABLE II: Maximum values of the datasets.

Train data max CPU	245.9730
Train data max memory	222.8578
Validation data max CPU	13.5571
Validation data max memory	10.5724
Test data max CPU	72.2677
Test data max memory	39.7109
OOD data max CPU	14.0444
OOD data max memory	23.7272

Comparatively, the test data mirrors the distribution observed in the training data, featuring more occurrences of high values. The OOD data shows a similar distribution to the validation data but with one difference: the density at low CPU values is not as pronounced.

In Figure 6, one can see the distribution of the memory values in all four datasets. The training dataset has not only higher CPU values but also higher memory values, which can be seen in Figure 6 and Table II. The validation data and OOD data have again a comparable distribution with a high percentage of low memory values. The median of the test data is higher than the ones of the validation and OOD data, but lower than the one from the train data.

E. Evaluation Metrics

When it comes to regression, the model is usually evaluated with *MSE*, *Mean Absolute Error (MAE)* or the *Root Mean Squared Error (RMSE)* [18]. We have decided to focus on the MSE in our evaluation since it is frequently used in related work, such as [6], [24], [30], [39], [44]–[46] and it puts larger

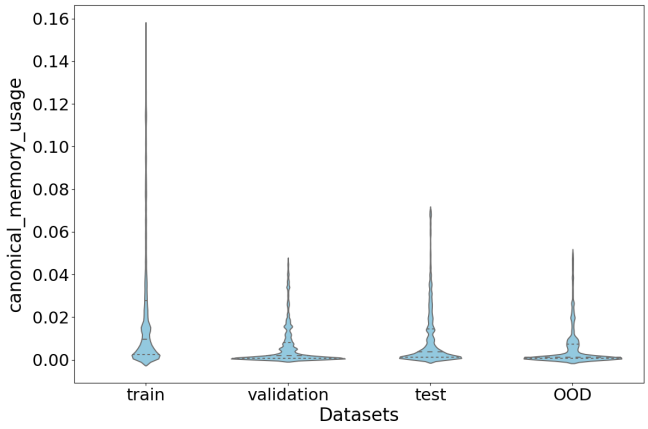


Fig. 6: Canonical memory usage distribution of the different datasets.

weight on higher errors, penalizing larger deviations between predicted and actual values more significantly than smaller errors.

F. Experimental Design

The steps conducted for the experiments are depicted in Figure 7. The various prediction approaches are shown in purple on the left-hand side. The next step of the experiments, highlighted in yellow, involves selecting suitable architectures. This step is exclusively for the two DL methods. The search for optimal hyperparameters (orange) and the subsequent phases of training and evaluation of the methods (green) are essential for all approaches except the NB. After the evaluation, there are two possible directions: Either proceed to the final step or resume the iterative process of refining hyperparameters and architectures. The final step, shown in blue, is shared across all the experiments. This is because the methods each predict new jobs and the results are systematically evaluated.

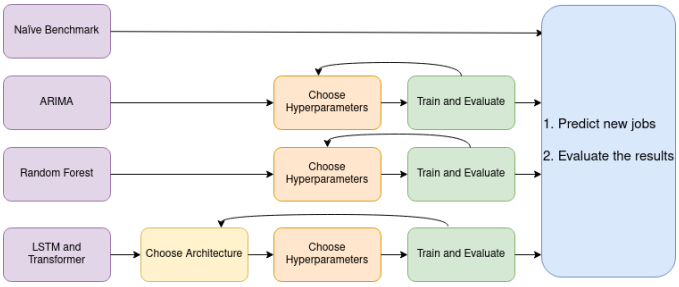


Fig. 7: Experimental design.

In the search for the optimal architectures and hyperparameters, tests have been conducted by using 10% of the jobs and the prediction target was the CPU usage. To determine the optimal hyperparameters for the ARIMA model, a systematic evaluation is conducted by assessing the performance on the validation and training dataset across all permutations of hyperparameter values, specifically considering binary configurations (0 or 1) for each hyperparameter. The three

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 24,2024 at 16:08:07 UTC from IEEE Xplore. Restrictions apply.

parameters for ARIMA are usually written as (p,d,q), where p is the number of lags of the stationarized series, d is the number of non-seasonal differences needed for stationarity and q is the number of lagged forecast errors. The best results were achieved using the parameters (1,0,0). The RF has two hyperparameters that can be optimized. These include the maximum depth of the individual trees and the total number of trees. The optimal maximum depth for the RF was determined by employing 100, 200, and 300 trees, comparing the results of using a maximum depth of 8, 16, 24, 32, and 64. The best results were achieved with 200 trees and a maximum depth of 16. The first step for optimizing DL methods is to select suitable architectures, then a grid search is used to find the best hyperparameters. The Long Short-Term Memory (LSTM) architecture is taken from [25] and comprises a variable number of LSTM layers. The final output is then processed through three fully connected layers, where in the first layer, a Rectified Linear Unit (ReLU) activation function is used. In Table III one can see the hyperparameters that lead to the best results. To find the best parameters, we trained the model for 100 epochs with a linear layer size of 500. We then tested different LSTM layer numbers and hidden sizes with a learning rate ranging from 0.01 to 0.00001 and batch sizes including 16, 32, and 64, until we received the optimal hyperparameters.

TABLE III: Hyperparameters of the LSTM model.

Hyperparameter	Value
hidden size	64
LSTM layers	1
learning rate	0.001/0.0001
batch size	32
linear layers size	500
epochs	10/13

Given the univariate nature of the data, a simple Transformer model is chosen, which is taken from [25]. The data is first embedded, and then an encoding layer is used. Finally, the data is passed to a decoder, which, in this context, is represented by a fully connected layer. The hyperparameters that lead to the best results are shown in Table IV. To find the best hyperparameters for the Transformer we trained it for 100 epochs. We then tested different hidden dimensions, number of heads, number of transformer encoder layers, and feed-forward dimensions with a learning rate ranging from 0.01 to 0.00001 and batch sizes including 16, 32, and 64, until we received the optimal hyperparameters.

TABLE IV: Hyperparameters of the Transformer model.

Hyperparameter	Value
hidden dimension	32
number of heads	4
feedforward dimension	32/64
number of Transformer encoder layers	1
learning rate	0.001
batch size	32
epochs	36/69

To train and evaluate the models, the training jobs were first used to train the model for a single epoch and then all the validation jobs were used for the evaluation. This process is repeated for a specified number of epochs and is shown in Algorithm 1.

Algorithm 1: Training process using multiple jobs

Input: t_l_list, v_l_list: PyTorch DataLoader lists
$model$: Model to be trained
$device$: Device for computation (CPU/GPU)
t: Number of prediction steps
opt: Optimizer algorithm

for $i \leftarrow 0$ **to** *nr_of_epochs* **do**
  for t_l *in* t_l_list **do**
    train_model(t_l, model, opt, device, t);
  mse = [test_model(v_l, model, opt, device, t) | v_l in v_l_list];
  sum_mse = sum(mse);

where:
- t_l: Training DataLoader
- v_l: Validation DataLoader

Once the appropriate hyperparameters are found, the data for the final predictions undergoes a partitioning process, dividing it into three distinct sets. 60% of the jobs are used for training, while 20% of the jobs are reserved for validation and additionally, 20% are chosen for testing. Therefore, our test data consists of 452 jobs, our validation data has 453 jobs, and the train data has 1356 jobs. Since our objective of the prediction is to estimate both the mean CPU usage and the canonical memory usage, a distinct model is constructed and trained for each target so that the input data for each model aligns with the respective goal it aims to predict. For the training and the prediction of individual jobs, each method takes six timestamps as input to predict the next six values as illustrated in Figure 8. Given the dataset, predicting up to six timestamps ahead is equivalent to predicting the resource usage of the next 30 minutes, leaving enough time to allocate resources.

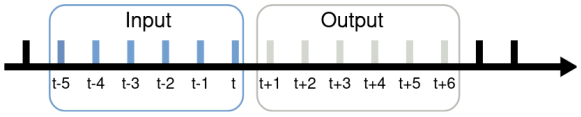


Fig. 8: Multistep-ahead prediction.

G. Hardware Setup

All tests conducted in this work ran on the same server that is equipped with 32 CPUs, specifically the AMD Ryzen 9 5950X 16-Core Processor, each with a maximum clock speed of 5083.3979 MHz. Additionally, the server incorporates 2 NVIDIA GeForce RTX 3090 GPUs, each featuring 24576 MB of memory.

IV. Results

In this section, we compare the results of the five different methods. We analyze the performance from one to six times-

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 24,2024 at 16:08:07 UTC from IEEE Xplore. Restrictions apply.

tamps (TS) ahead, looking at aggregated metrics for MSE, namely Mean ($\bar{x}$) and Standard Deviation (σ). To emphasize the most important values, we have highlighted the lowest values for $\bar{x}$ and σ in bold. First, we analyze the prediction results of the test data in predicting the CPU values and then compare the methods based on their predictions for memory. Then we identify how well the methods work on jobs with shorter execution time, i.e., running for less than a week. Lastly, we compare the prediction and training times.

1) Comparison of the methods based on predicting the CPU: Table V shows the mean values and standard deviations of the MSE of the CPU usage of the different methods when predicting one to six timestamps in advance.

TABLE V: Mean squared error (MSE) prediction results of the methods for predicting the mean_CPU_usage of the test data.

	1 TS		2 TS		3 TS	
Method	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
NB	0.0311	0.4356	0.0363	0.4415	0.0444	0.5269
ARIMA	0.0281	0.3607	0.0320	0.3795	0.0391	0.4572
RF	0.0308	0.3998	0.0406	0.4367	0.0536	0.5271
LSTM	**0.0251**	**0.3116**	**0.0307**	**0.3534**	0.0366	0.4160
Trans.	0.0288	0.3640	0.0336	0.3889	**0.0356**	**0.3951**
	4 TS		5 TS		6 TS	
Method	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
NB	0.051	0.6069	0.0506	0.5738	0.0627	0.7580
ARIMA	0.0421	0.4823	0.0460	0.5240	**0.0482**	**0.5419**
RF	0.0619	0.5751	0.0622	0.5656	0.0676	0.6823
LSTM	**0.0409**	**0.4621**	0.0438	0.4856	**0.0487**	**0.5463**
Trans.	0.0449	0.5257	**0.0426**	**0.4699**	**0.0486**	**0.5479**

Given this table, one can see that the LSTM and the Transformer have lower averages and standard deviations than the other methods in most cases, which is not surprising since they are known to better capture complex correlations. The ARIMA outperforms the NB, and when predicting 6 timestamps ahead, it has the lowest average MSE and the lowest standard deviation. It is worth mentioning that all methods, except for the RF have lower average MSE than the NB. One possible explanation for the bad performance of the RF is that its hyperparameters have been chosen on 10% of the jobs, and for more data, a more complex model may be needed. Overall the results on the test data are as expected, showing that the deep learning methods perform best.

2) Comparison of the methods based on predicting the memory: Table VI depicts the average MSE and the standard deviations of the prediction of the canonical memory usage with the test data.

When comparing the averages, the LSTM again outperforms the other methods when predicting only one and also when predicting multiple timestamps ahead. The Transformer model has significantly higher MSE averages than the other methods except for the RF when predicting one, two, and three timestamps ahead, but when predicting five and six timestamps ahead, it outperforms the NB. The ARIMA performs worse than the NB. One possible explanation for the bad performance of the ARIMA is the hyperparameters that have been chosen

TABLE VI: Average MSE prediction results of the methods, predicting the canonical_memory_usage of the test data.

	1 TS		2 TS		3 TS	
Method	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
NB	0.0013	0.0158	0.0016	0.0178	0.0018	0.0190
ARIMA	0.0020	0.0263	0.0021	0.0260	0.0025	0.0295
RF	0.0017	0.0238	0.0040	0.0634	0.0054	0.0895
LSTM	**0.0008**	**0.0084**	**0.0011**	**0.0107**	**0.0013**	**0.0129**
Trans.	0.0060	0.1016	0.0040	0.0628	0.0046	0.0700
	4 TS		5 TS		6 TS	
Method	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
NB	0.0021	0.0213	0.0023	0.0237	0.0026	0.0263
ARIMA	0.0027	0.0314	0.0030	0.0337	0.0035	0.0417
RF	0.0060	0.0977	0.0060	0.0916	0.0060	0.0890
LSTM	**0.0015**	**0.0155**	**0.0017**	**0.0177**	**0.0023**	**0.0234**
Trans.	0.0032	0.0421	0.0020	0.0188	0.0025	0.0240

for the prediction of the CPU, which do not generalize well for other metrics. The RF has a much higher average MSE when predicting multiple timestamps ahead, which can be expected, given its performance on the CPU values.

3) Out of Distribution (OOD): In the following section, our focus shifts to examining the performance of these methods in predicting 3432 jobs with a different pattern that have a duration of more than 1 hour, but less than a week. For the following tests, we take the models that we have trained on jobs with a longer duration. In Table VII one can see the prediction results of the five methods when predicting the CPU usage of short jobs. In this table, one can see that the

TABLE VII: Average MSE prediction results of the methods, predicting the mean_CPU_usage of the OOD data.

	1 TS		2 TS		3 TS	
Method	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
NB	**0.0811**	**0.6565**	**0.1317**	**1.0429**	**0.1774**	**1.4329**
ARIMA	0.1567	1.1068	0.2030	1.5101	0.2609	2.0749
RF	0.1474	1.3316	0.2696	1.8911	0.4239	2.7291
LSTM	0.1195	0.9464	0.1692	1.4073	0.2181	1.8761
Trans.	0.1306	1.0329	0.2014	1.6675	0.2382	2.0366
	4 TS		5 TS		6 TS	
Method	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
NB	**0.2091**	**1.7445**	**0.2363**	**1.9894**	**0.2791**	**1.7092**
ARIMA	0.3117	2.5703	0.3376	2.8420	0.3678	3.0966
RF	0.5181	3.3225	0.5538	3.7453	0.5255	3.7243
LSTM	0.2424	2.1315	0.2578	2.2705	0.2819	2.4382
Trans.	0.2619	2.2888	0.2510	2.1990	0.2847	2.4612

NB performs better than the other methods. Both DL methods perform worse than the NB as they have higher means and standard deviations, indicating that they do not generalize as well to new data. The ARIMA has higher averages than the NB and DL methods and the RF has the highest average loss when predicting multiple timestamps ahead. Still, the LSTM provides prediction with only a slightly higher MSE.

In Table VIII, one can see standard deviations and averages when predicting the canonical memory usage of the OOD data. The LSTM and NB have the best results. The Transformer performs slightly worse and the ARIMA has the highest average losses. One possible reason for the bad performance of

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 24,2024 at 16:08:07 UTC from IEEE Xplore. Restrictions apply.

TABLE VIII: Average MSE prediction results of the methods, predicting the canonical_memory_usage of the OOD data.

	1 TS		2 TS		3 TS	
Method	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
NB	**0.1120**	**1.4906**	**0.1402**	**1.6806**	0.1860	**2.3878**
ARIMA	0.1829	2.0993	0.2763	3.1772	0.3363	3.8763
RF	0.1678	2.3880	0.2249	2.8418	0.2894	3.7589
LSTM	0.1323	1.7162	0.1660	2.1362	**0.1767**	2.4130
Trans.	0.1806	2.3340	0.1863	2.4042	0.1986	2.6897
	4 TS		5 TS		6 TS	
Method	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
NB	0.1140	1.3680	0.1183	1.4218	**0.0618**	**0.5241**
ARIMA	0.3226	3.7774	0.2118	2.3225	0.2385	2.5946
RF	0.2044	2.4902	0.2095	2.5475	0.2280	2.7050
LSTM	**0.0760**	**1.0191**	**0.0783**	**1.0406**	0.1094	1.3417
Trans.	0.0803	1.0941	0.0899	1.2286	0.1215	1.5126

the ARIMA is that it highly depends on the hyperparameters that may change when the data distribution changes.

To conclude the comparison of the methods on the OOD data, the NB performs best in most cases. The other methods have more difficulty predicting the resource usage of data with different patterns. The DL methods have slightly higher average MSE, but in some cases, they outperform the benchmark. In particular, the LSTM model keeps up with low MSE losses. Intuitively, giving the possibility to the LSTM model to adapt and improve over time, we can obtain a robust and fully generalizable model.

4) Comparison of the methods based on their training and prediction time: To assess how long the methods take for training and predictions, we have added the training and prediction times in seconds for predicting the CPU usage of the non-OOD data that are depicted in Table IX. The NB and the ARIMA do not need to be trained. The Transformer and the LSTM are characterized by the longest training duration, while the RF has the shortest training time among the methods requiring training. When it comes to predicting the test, train, and validation data of the non-OOD dataset, the NB is the fastest, followed by the RF. It is noticeable that the LSTM and the Transformer require significantly more time to predict the CPU values of the jobs compared to these other two methods. The ARIMA has the highest prediction time of all methods. Comparing the training and prediction times of the five methods, it is evident that the NB is the fastest method overall. The RF requires a long time for training but not as much for prediction. The ARIMA does not need to be trained, but it takes more time for prediction, resulting in a total time greater than that required for training and predicting using DL methods. Both DL methods have comparable prediction and training times.

TABLE IX: Training and prediction time (for CPU).

Method	Training (seconds)	Prediction (seconds)
Naïve Benchmark	No training needed	86
ARIMA	No training needed	287 672
RF	11 290	116
LSTM	46 946	14 360
Transformer	47 224	14 150

V. Conclusion

In this paper, we compared different methods in their ability to predict the workload of upcoming jobs. Inspired by Transfer Learning, we used trained models to predict new jobs, reducing the need to wait for training data and reducing computational costs. In our comparison, we considered different targets, different datasets, and the prediction of up to six timestamps ahead. The MSE was chosen as a metric to compare our results. In our comprehensive evaluation, we found that the LSTM performed well in predicting two different targets using the test data. However, when predicting the OOD data, the NB outperformed the LSTM in most cases. The RF did not perform well in our experiments as it had the highest average loss in most cases. When it comes to training times, both Machine Learning and Deep Learning methods took the longest. However, in terms of prediction times, the ARIMA method took significantly longer than the others.

The results of our work can guide the selection of an appropriate method for accurate resource forecasting. Furthermore, our open-source implementation is available for others to utilize as a foundation for further development. In the future, we aim to leverage our findings and deploy solutions in practical scenarios, extending previous work done in Polaris [29] as part of the Linux Foundation Centaurus project. We target management measures such as autoscaling or rescheduling for maintaining SLAs, which can be towards cost-, resource- or energy-awareness. In this direction, we plan to test the considered methods on different datasets that can display different patterns, expanding our generalization analysis. Furthermore, we want to explore the management of these methods over time, e.g., studying the best strategies to prevent model drifting. In conclusion, our comprehensive analysis and open-source implementation represent leverage for further implementations of resource usage forecasting, providing precious tools for researchers and practitioners.

VI. Acknowledgment

This work has been supported by the European Union's Horizon Europe research and innovation programme under grant agreement No. 101079214 (AIoTwin) and No. 101135576 (INTEND).

References

[1] Hyun Ahn, Kyunghee Sun, and Kwanghoon Kim. Comparison of missing data imputation methods in time series forecasting. *Computers, Materials and Continua*, 70:767–779, 09 2021.

[2] Mustafa M. Al-Sayed. Workload time series cumulative prediction mechanism for cloud resources using neural machine translation technique. *J. Grid Comput.*, 20(2), jun 2022.

[3] Jing Bi, Shuang Li, Haitao Yuan, and MengChu Zhou. Integrated deep learning method for workload and resource prediction in cloud systems. *Neurocomputing*, 424:35–48, 2021.

[4] Rodrigo N. Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya. Workload prediction using arima model and its impact on cloud applications' qos. *IEEE Transactions on Cloud Computing*, 3(4):449–458, 2015.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 24,2024 at 16:08:07 UTC from IEEE Xplore. Restrictions apply.

[5] Rui Cao, Zhaoyang Yu, Trent Marbach, Jing Li, Gang Wang, and Xiaoguang Liu. Load prediction for data centers based on database service. In *2018 IEEE 42nd annual computer software and applications conference (COMPSAC)*, volume 1, pages 728–737. IEEE, 2018.

[6] Zheyi Chen, Jia Hu, Geyong Min, Albert Y. Zomaya, and Tarek El-Ghazawi. Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning. *IEEE Transactions on Parallel and Distributed Systems*, 31(4):923–934, 2020.

[7] Liu Chunhong, Jie Jiao, Weili Li, Jingxiong Wang, and Junna Zhang. Tr-predictior: An ensemble transfer learning model for small-sample cloud workload prediction. *Entropy*, 24:1770, 12 2022.

[8] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 153–167, 2017.

[9] Timothy DeStefano, Richard Kneller, and Jonathan Timmis. Cloud computing and firm growth. *Review of Economics and Statistics*, pages 1–47, 2023.

[10] K. Lalitha Devi and S. Valli. Time series-based workload prediction using the statistical hybrid model for the cloud environment. *Computing*, Nov 2022.

[11] Maximilian Du. Improving lstm neural networks for better short-term wind power predictions. In *2019 IEEE 2nd International Conference on Renewable Energy and Power Engineering (REPE)*, pages 105–109, 2019.

[12] Brad Everman, Maxim Gao, and Ziliang Zong. Evaluating and reducing cloud waste and cost—a data-driven case study from azure workloads. *Sustainable Computing: Informatics and Systems*, 35:100708, 2022.

[13] Jiechao Gao, Haoyu Wang, and Haiying Shen. Machine learning based workload prediction in cloud computing. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, 2020.

[14] GitHub. google/cluster-data/clusterdata2011_2.md. https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md, 2021.

[15] GitHub. alibaba/clusterdata. https://github.com/alibaba/clusterdata, 2022.

[16] GitHub. Azure/azurepublicdataset. https://github.com/Azure/AzurePublicDataset, 2022.

[17] Shaifu Gupta, A. D. Dileep, and Timothy A. Gonsalves. Online sparse blstm models for resource usage prediction in cloud datacentres. *IEEE Transactions on Network and Service Management*, 17(4):2335–2349, 2020.

[18] Guy S. Handelman, Hong Kuan Kok, Ronil V. Chandra, Amir H. Razavi, Shiwei Huang, Mark Brooks, Michael J. Lee, and Hamed Asadi. Peering into the black box of artificial intelligence: Evaluation metrics of machine learning methods. *American Journal of Roentgenology*, 212(1):38–43, 2019. PMID: 30332290.

[19] Jerry L Hintze and Ray D Nelson. Violin plots: a box plot-density trace synergism. *The American Statistician*, 52(2):181–184, 1998.

[20] Byeonghui Jeong, Jueun Jeon, and Young-Sik Jeong. Proactive resource autoscaling scheme based on scinet for high-performance cloud computing. *IEEE Transactions on Cloud Computing*, PP:1–14, 10 2023.

[21] Ananya Kaim, Surjit Singh, and Yashwant Singh Patel. Ensemble cnn attention-based bilstm deep learning architecture for multivariate cloud workload prediction. In *Proceedings of the 24th International Conference on Distributed Computing and Networking*, ICDCN '23, page 342–348, New York, NY, USA, 2023. Association for Computing Machinery.

[22] Michael J Kane, Natalie Price, Matthew Scotch, and Peter Rabinowitz. Comparison of arima and random forest time series models for prediction of avian influenza h5n1 outbreaks. *BMC bioinformatics*, 15(1):1–9, 2014.

[23] Hisham Kholidy. An intelligent swarm based prediction approach for predicting cloud computing user resource needs. *Computer Communications*, 151, 02 2020.

[24] Jitendra Kumar, Ashutosh Kumar Singh, and Rajkumar Buyya. Self directed learning based workload forecasting model for cloud resource management. *Information Sciences*, 543:345–366, 2021.

[25] A. Lackinger. Towards accurate time series predictions for cloud workloads, 2023. reposiTUm.

[26] Bingwei Liu, Yinan Lin, and Yu Chen. Quantitative workload analysis and prediction using google cluster traces. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 935–940, 2016.

[27] Ning Liu, Zhe Li, Jielong Xu, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, and Yanzhi Wang. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, pages 372–382. IEEE, 2017.

[28] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020. M4 Competition.

[29] Andrea Morichetta, Vıctor Casamayor Pujol, Stefan Nastic, Thomas Pusztai, Philipp Raith, Schahram Dustdar, Deepak Vij, Ying Xiong, and Zhaobo Zhang. Demystifying deep learning in predictive monitoring for cloud-native slos. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*.

[30] Bhalaji Natarajan. Cloud load estimation with deep logarithmic network for workload and time series optimization. *Journal of Soft Computing Paradigm*, 3:234–248, 09 2021.

[31] Yashwant Singh Patel and Jatin Bedi. Mag-d: A multivariate attention network based approach for cloud workload forecasting. *Future Generation Computer Systems*, 142:376–392, 2023.

[32] Irfan Pratama, Adhistya Erna Permanasari, Igi Ardiyanto, and Rini Indrayani. A review of missing values handling methods on time-series data. In *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, pages 1–6, 2016.

[33] Charles Reiss. Towards understanding heterogeneous clouds at scale : Google trace analysis. 2012.

[34] Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, 1, 2011.

[35] Andrea Rossi, Andrea Visentin, Steven Prestwich, and Kenneth N Brown. Clustering-based numerosity reduction for cloud workload forecasting. In *International Symposium on Algorithmic Aspects of Cloud Computing*, pages 115–132. Springer, 2023.

[36] Sarikaa S, Niranjana S, and Sri Vishnu Deepika K. Time series forecasting of cloud resource usage. In *2021 IEEE 6th International Conference on Computing, Communication and Automation (ICCCA)*, pages 372–382, 2021.

[37] Gurjot Singh, Prajit Sengupta, Anant Mehta, and Jatin Bedi. A feature extraction and time warping based neural expansion architecture for cloud resource usage forecasting. *Cluster Computing*, pages 1–20, 01 2024.

[38] Gurjot Singh, Prajit Sengupta, Anant Mehta, and Jatin Bedi. A feature extraction and time warping based neural expansion architecture for cloud resource usage forecasting. *Cluster Computing*, pages 1–20, 2024.

[39] Binbin Song, Yao Yu, Yu Zhou, Ziqiang Wang, and Sidan Du. Host load prediction with long short-term memory in cloud computing. *J. Supercomput.*, 74(12):6554–6568, dec 2018.

[40] Tejas Subramanya and Roberto Riggio. Centralized and federated learning for predictive vnf autoscaling in multi-domain 5g networks and beyond. *IEEE Transactions on Network and Service Management*, 18(1):63–78, 2021.

[41] Tiffany Tuor, Shiqiang Wang, Kin K Leung, and Bong Jun Ko. Online collection and forecasting of resource utilization in large-scale distributed systems. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 133–143. IEEE, 2019.

[42] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3:1–40, 2016.

[43] Minxian Xu, Chenghao Song, Huaming Wu, Sukhpal Singh Gill, Kejiang Ye, and Chengzhong Xu. Esdnn: Deep neural network based multivariate workload prediction in cloud computing environments. *ACM Trans. Internet Technol.*, 22(3), aug 2022.

[44] Qiangpeng Yang, Chenglei Peng, He Zhao, Yao Yu, Yu Zhou, Ziqiang Wang, and Sidan Du. A new method based on psr and ea-gmdh for host load prediction in cloud computing system. *J. Supercomput.*, 68(3):1402–1417, jun 2014.

[45] Qiangpeng Yang, Yu Zhou, Yao Yu, Jie Yuan, Xianglei Xing, and Sidan Du. Multi-step-ahead host load prediction using autoencoder and echo state networks in cloud computing. *J. Supercomput.*, 71(8):3037–3053, aug 2015.

[46] Weishan Zhang, Bo Li, Dehai Zhao, Faming Gong, and Qinghua Lu. Workload prediction for cloud cluster using a recurrent neural network. In *2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI)*, pages 104–109, 2016.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 24,2024 at 16:08:07 UTC from IEEE Xplore. Restrictions apply.