Cite this research:
Dittakavi, R.S.S,(2021). Deep Learning-Based Prediction of CPU and Memory Consumption for Cost-Efficient Cloud Resource Allocation SSRAML SageScience, 3(1), 45–58.

Article history:
Received:
January/04/2021
Accepted:
April/13/2021

.

Deep Learning-Based Prediction of CPU and Memory Consumption for Cost-Efficient Cloud Resource Allocation

Raghava Satya SaiKrishna Dittakavi
Independent Researcher, USA

Abstract

Cloud computing has become an integral part of modern IT infrastructure, offering scalable resources and services on-demand. However, the economic implications of cloud usage are significant, and efficient resource allocation is crucial for cost management. Traditional methods often rely on threshold-based rules or manual oversight for resource scaling, which can be inefficient and costly. There is a growing interest in using deep learning to predict resource usage and thereby optimize cost. This study presents a deep learning-based model specifically designed to predict future CPU and memory consumption in cloud environments. The objective of the model is to facilitate proactive resource allocation by predicting future CPU and memory usage. To experiment, the model was trained, validated, and tested on Google Cluster Workload Traces 2019 data. Temporal Fusion Transformer (TFT), and Gated Recurrent Units (GRU) were used. The model's performances were also compared against traditional Exponential Smoothing (ETS) model. While the GRU model exhibited a Mean Absolute Error (MAE) of 6.45 and Root Mean Square Error (RMSE) of 8.15, these metrics were better than ETS and slightly less optimal than TFT. Once the model is trained, it can be deployed into the cloud infrastructure and is set up with an automated system capable of initiating auto-scaling based on the predictions. This proactive approach enables more efficient utilization of resources, with the possibility for significant cost reduction. The model can also minimize the need for manual monitoring by incorporating a feedback loop for retraining or fine-tuning. This study offers an approach for enterprises and cloud service providers aiming for intelligent resource allocation and cost minimization.

Keywords: *Cloud computing, Deep learning, Gated Recurrent Units (GRU), Long Short-Term Memory (LSTM), Proactive resource allocation, Temporal Fusion Transformer (TFT)*

Introduction

Cloud computing in recent years has experienced a significant surge in adoption across various sectors, signifying a profound shift in the way businesses and individuals consume and manage computational resources [1], [2]. Traditionally, enterprises depended on on-premises data centers and in-house servers to run their applications and store data. These systems often required substantial initial investments, not only in hardware but also in skilled personnel to manage and maintain the infrastructure. Additionally, scalability could pose challenges as upgrading capacity or capabilities usually meant further hefty expenses in hardware and downtime during installation. In contrast, cloud computing offers a model where computational resources, such as storage, processing power, and applications, are accessed over the internet and billed based on usage. This shift eliminated the need for businesses to make substantial upfront investments in infrastructure.

The adoption of scalable resources and on-demand services has significantly transformed the operational capabilities of businesses, addressing many traditional limitations and a new age of flexibility and responsiveness. In the past, organizations often grappled with

predicting their resource needs. They faced the dilemma of either over-provisioning, which would result in underutilized resources and wasted capital, or under-provisioning, which could hinder performance and potentially lose business opportunities [3], [4]. Scalable resources address this challenge head-on. With the ability to dynamically adjust to the demands, companies no longer need to commit to fixed assets. Instead, they can adapt their resource allocation in real-time, ensuring optimum utilization and efficient cost management.

At a high level, the architecture of cloud computing can be categorized into two main divisions: the core stack and the management aspects. Delving deeper into the core stack, we observe it is further split into three fundamental layers. The first is the Resource layer, which essentially serves as the foundation of the cloud architecture. It constitutes both the physical and virtualized assets of computing, storage, and networking resources. This layer's primary purpose is to deliver the necessary computational power, store data, and ensure smooth communication between various parts of the system [5], [6].

The Platform layer holds significant complexity and is further subdivided into multiple sub-layers. For instance, within this layer, the computing framework plays a crucial role. It oversees essential functions like transaction dispatching and task scheduling, ensuring the efficient distribution and processing of tasks. Another integral sub-layer is the storage sub-layer, which offers expansive storage capabilities combined with caching [7], [8]. The purpose of this sub-layer is to manage large volumes of data, often encountered in cloud environments, and provide fast access to frequently used information.

The Application layer forms the third part of the core stack. This layer comprises the application server and other vital components. It upholds the application logic similar to traditional systems but integrates advanced features, such as on-demand capabilities or adaptable management strategies. This ensures that the cloud system operates optimally, avoiding any singular component becoming a hindrance or bottleneck in the overall functioning of the cloud system.

The integration of cloud services into business operations presents not just technical and operational decisions, but also crucial economic considerations. These financial aspects play a central role in determining the viability, efficiency, and long-term sustainability of cloud adoption for businesses of all sizes. Historically, IT infrastructures demanded substantial capital expenditure (CapEx) on hardware, software licenses, and the physical spaces to house them. These upfront costs posed a considerable barrier to entry for many businesses, particularly smaller ones with limited capital. With the introduction of cloud services, there's a notable shift from capital-intensive investments towards operational expenditure (OpEx). This model allows organizations to pay for only what they use, transforming large lump sum costs into predictable monthly or annual payments. Such a transition not only eases financial planning but also democratizes access to advanced IT resources, as businesses can tap into state-of-the-art technologies without massive initial investments [9], [10].

Costs in the cloud can quickly escalate if not monitored and managed effectively. For instance, without proper oversight, unused resources might continue to incur charges, or data transfers and requests might exceed allocated budgets. It becomes imperative for businesses to have clear visibility into their cloud usage and expenses. Effective cloud cost

management and monitoring tools have thus emerged critical for businesses to ensure they derive maximum value from their cloud investments and avoid unexpected financial strains.

In cloud computing, effective cost management directly depends on how efficiently resources are allocated. When businesses move to the cloud, they are often charged based on the resources they use. This pricing model means that if a company is not careful, it can end up paying for resources that are underused or not used at all. The ability to quickly adjust resources based on demand is a key feature of cloud computing. However, this also presents a challenge [11]. If a business allocates too many resources expecting high demand, and that demand doesn't materialize, the company ends up paying for unused capacity. On the other hand, if they allocate too few resources, it can lead to poor system performance and potentially lost business opportunities [12].

In cloud environments, CPU (Central Processing Unit) and memory are two of the primary resources that determine the performance and efficiency of applications and services. The CPU, often termed the "brain" of any computing environment, handles executing instructions of a computer program, while memory (or RAM - Random Access Memory) temporarily stores data that the CPU may need for tasks, allowing for faster data retrieval than from storage devices. In cloud settings, both these resources are allocated based on the requirements of the hosted applications and can be dynamically adjusted as demand changes [13], [14].

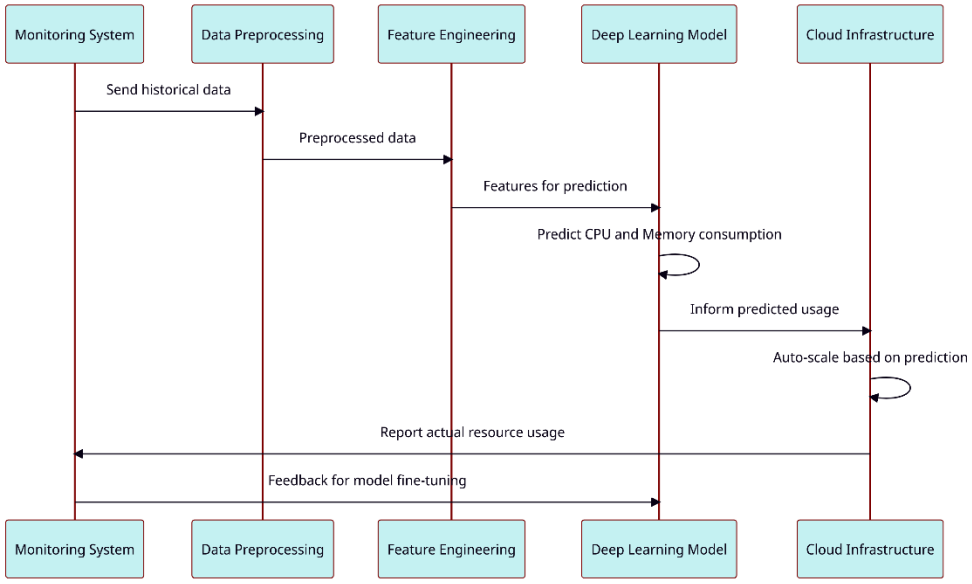
Effective management of CPU and memory is vital for several reasons. Firstly, optimal allocation ensures that applications run smoothly without performance hitches. If the CPU is overburdened or if there's insufficient memory, applications can slow down, resulting in longer processing times and potential system bottlenecks. Such scenarios can degrade user experience, especially if the applications are customer-facing. Conversely, over-allocating these resources, where more CPU and memory are provisioned than necessary, can lead to increased costs without any tangible performance benefit. This is especially pertinent in cloud environments where users often pay based on the resources they consume. Challenges associated with CPU and memory consumption are addressed through monitoring and management tools. If an application experiences a surge in users, more CPU and memory can be allocated to handle the increased load. Once the demand subsides, resources can be scaled back. Effective management ensures not only optimal performance but also cost efficiency, as organizations only pay for what they use.

Deep Learning-Based Model for Resource Prediction

Traditional methods, which predominantly involve setting thresholds for resource scaling, often do not account for sudden surges or dips in usage, leading to either resource wastage or potential downtimes. However, with the integration of deep learning models, we can move from reactive to proactive resource allocation strategies, which not only ensure optimal performance but also significant cost savings. Predicting CPU and Memory consumption using deep learning represents a significant advancement in the optimization of resource allocation within cloud infrastructures. With the increasing reliance on cloud computing, efficient resource management has become paramount to ensure both cost-effectiveness and the smooth functioning of applications and services.

The inception of this approach requires meticulous data collection. Cloud providers typically offer monitoring tools that continuously log various metrics. These metrics, such as CPU utilization percentages, memory usage in standard units like GB or MB, and other related network activities, serve as the raw data for our model. While data abundance is rarely an issue with these monitoring tools, the quality and consistency of the data can pose challenges. Data preprocessing thus becomes a crucial step before any model training. During preprocessing, any missing values are identified and filled using appropriate techniques, such as interpolation or using mean values. Outliers, which can adversely affect the model's predictive power, are detected and treated, ensuring they do not skew the training process. Given that different metrics can be on different scales, normalization ensures that all data points are consistent, preventing any one metric from disproportionately influencing the model.

Figure 1. Visual representation of the proposed model



Once the data is refined, the next phase involves feature engineering. At its core, feature engineering is about transforming the raw data into a format that enhances the model's ability to discern patterns. For time series data like CPU and Memory consumption, temporal features play a significant role. Extracting data points such as the hour of the day, the specific day of the week, or even the month can provide the model with context. It's essential to understand that CPU and Memory consumption can exhibit daily, weekly, or even monthly patterns. For instance, a business application might experience higher usage during weekdays as compared to weekends, or a specific service might see surges at the beginning or end of a month. Furthermore, lagged values, or data from preceding hours or days, can offer insights into short-term trends. If a server's CPU consumption has been steadily rising over the past few hours, it's a valuable indicator of the trend's continuation, at least in the immediate future. Similarly, rolling metrics, like means or medians, offer a broader view, smoothing out short-term fluctuations and highlighting more extended trends.

With the data prepared, the focus shifts to the deep learning model itself. Time series data, due to its sequential nature, finds an ideal match in Recurrent Neural Networks (RNNs).

Within the realm of RNNs, architectures like Long Short-Term Memory (LSTM) units or Gated Recurrent Units (GRU) have gained prominence due to their ability to capture long-term dependencies. Designing the neural network would typically involve an input layer to ingest the engineered features. Following this, multiple layers of LSTM or GRU cells capture and process the sequential patterns in the data. These recurrent layers can be followed by one or two dense layers, further refining the patterns and relationships. The final layer, or the output layer, is dedicated to the predictions themselves. Given that we're trying to predict continuous values, like percentages or usage units, the regression nature of the problem suggests a linear activation function for this layer.

Training the deep learning model is an iterative process. The data, already divided sequentially into training, validation, and testing subsets, is fed into the model. The Mean Squared Error (MSE), a standard loss function for regression problems, measures the model's accuracy. Optimization algorithms like Adam or RMSprop adjust the model's parameters, minimizing the MSE. As the training progresses, the model's performance on the validation set offers insights into its generalizability. Once training is deemed satisfactory, the model undergoes evaluation on the test set, with metrics like the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE) offering quantitative measures of its predictive accuracy.

In a cloud environment, the model can be hosted as an API, continuously receiving data, and outputting predictions. An associated automation system takes these predictions and makes scaling decisions. If the model foresees a surge in CPU or Memory consumption in the next few hours, the system can proactively allocate more resources, ensuring that when the surge does occur, the infrastructure is ready. Conversely, if a dip is predicted, the system can scale down, conserving resources and reducing costs.

This proactive approach to resource management has several implications. First, it ensures that services and applications always have the resources they need, leading to optimal performance. Downtimes or performance hiccups due to resource constraints can be virtually eliminated. Second, by predicting and acting on downtrends, resource wastage is minimized. Instead of keeping resources idle during low usage periods, they can be scaled down, directly translating to cost savings. Finally, the entire system's automation reduces the need for constant human monitoring, further driving down operational costs.

Table 1. Model algorithm

```
1. INITIALIZATION AND DATA COLLECTION:
  - Define model parameters such as number of LSTM/GRU layers, dense layers, loss function, and optimization algorithm.
  FUNCTION collect_data():
    Retrieve historical data from cloud provider's monitoring tools
    RETURN collected data

2. DATA PROCESSING AND FEATURE ENGINEERING:
  FUNCTION process_and_engineer(data):
    Handle missing values, remove outliers, and normalize the data
    Extract time features like hour, day, and month
    Add lagged values and compute rolling metrics
    RETURN engineered data

3. MODEL CREATION, TRAINING, AND EVALUATION:
  FUNCTION create_train_evaluate(data):
    Build the neural network model with input, recurrent, dense, and output layers
```

```
    Split data sequentially into training, validation, and test sets
    Train the model using the specified loss function and optimization algorithm
    Evaluate model using MAE and RMSE on test data
    RETURN trained model, evaluation metrics

4. DEPLOYMENT, MONITORING, AND AUTOMATION:
  FUNCTION deploy_and_monitor(model):
    Deploy model as an API endpoint in the cloud
    Set up automation for resource scaling based on predictions
    Continuously log and monitor CPU and memory consumption
    Compare real values with model predictions and adjust as necessary
    RETURN deployment status, monitored data

5. COST SAVING ACTIONS AND FEEDBACK:
  FUNCTION cost_saving_actions(model, monitored_data):
    Use model predictions to proactively scale resources
    Scale down during predicted downtimes
    Fine-tune or retrain the model based on feedback from real consumption values
    RETURN updated model, savings metrics
```

Dataset

The dataset for this study was formulated based on the initial Google Cluster Data which focuses on CPU and Memory usage. A Google cluster consists of numerous machines systematically organized within physical structures known as racks [15], [16]. These machines are interconnected by a network specifically designed for high-bandwidth operations. Within this architecture, a "cell" is defined as a collection of these machines, typically situated within a singular cluster. A distinctive characteristic of these cells is the shared cluster-management system they operate under. This system's primary function is to distribute computational tasks across the machines in the cell, ensuring effective utilization of available resources.

Borg, a system developed by Google, is structured to handle two main types of resource requests. The first, labeled as a "job," is composed of one or more tasks. Each task outlines a specific computation a user wishes to perform. The second type, known as an "alloc set," can also be subdivided into one or more alloc instances. The alloc set delineates a resource reservation space in which jobs are executed. Each task associated with a job corresponds to a Linux program, which may involve multiple processes. Tasks are designated to execute on individual machines. If a job is associated with a specific alloc set, its tasks are allocated resources from an instance of that alloc set [17], [18]. Conversely, if a job lacks an alloc set specification, its tasks directly source resources from a machine. Within Borg's terminology, "collections" refer to the larger entities of jobs and alloc sets, while "instances" pertain to the more granular tasks and alloc instances.

The concept of a usage trace in Borg's operations records the activity spanning several days within a designated Borg cell. This trace is organized into various tables, each of which is primarily indexed by a timestamp. The data encompassed within these tables is sourced both from the cell's overarching management system and the individual machines that constitute that cell. Borg's primary users include Google's internal developers and system administrators [19]. The Borg system's capabilities enable it to process a substantial volume of job requests daily, consistently maintaining services across an expansive range of applications.

Following Hieu et al., (2017), the variables were calculated by accumulating the tasks from each job in terms of their CPU and Memory consumption, calculated every five minutes throughout a 24-hour period [20]. Data was then extracted from the first ten days, with a specific emphasis on CPU and memory utilization ranging between 5% and 95%. The final dataset presents two variables. The first is indicating the percentage of CPU usage and the second is showing the percentage of Memory usage.

Figure 2. Time series plots for the variables

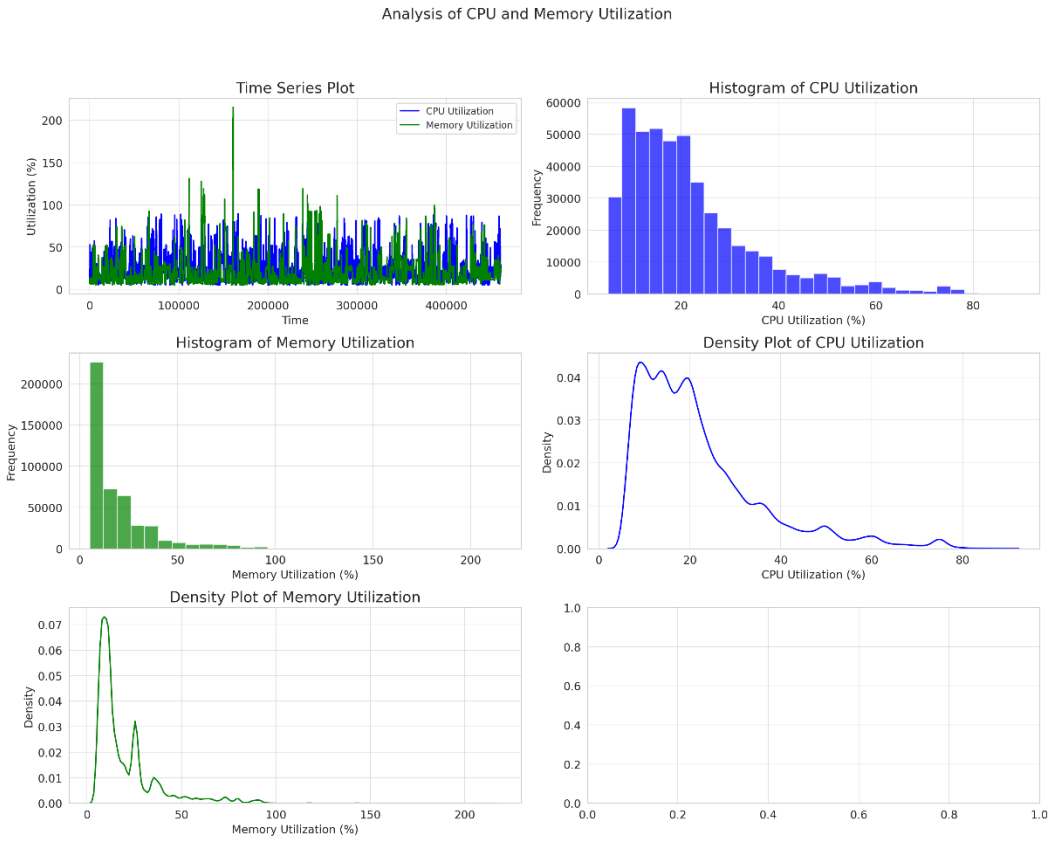
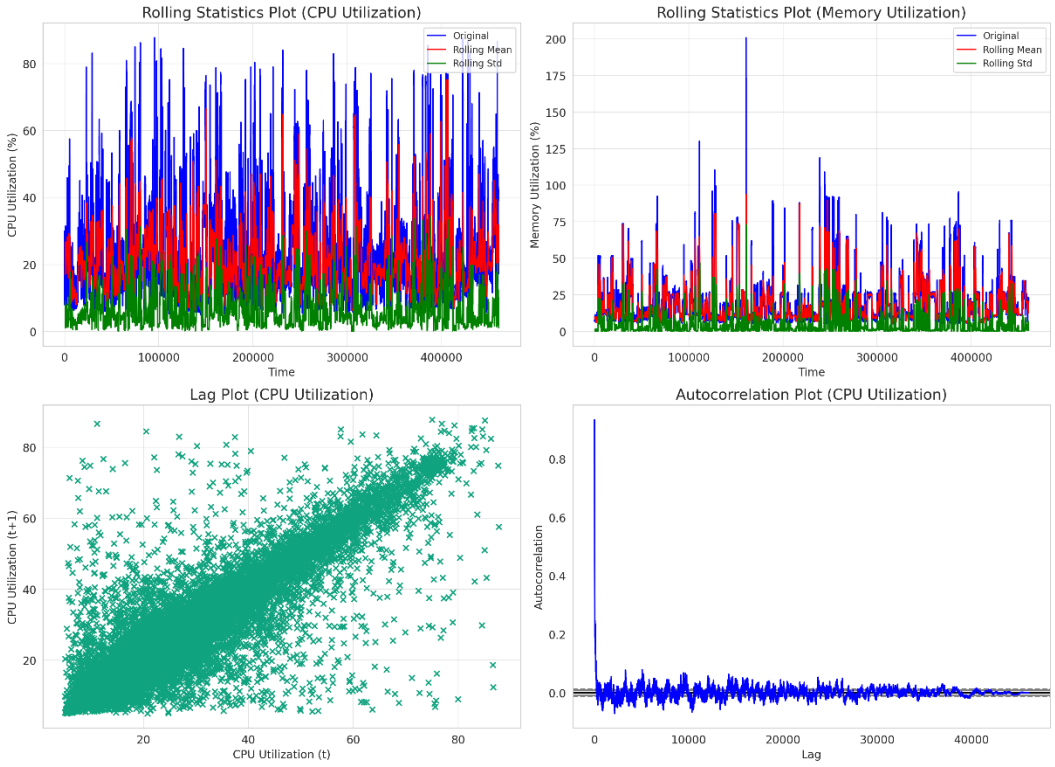


Table 2 shows that the mean CPU utilization is 21.85%, with a median of 18.55% and a standard deviation of 13.63%. The range for CPU utilization spans from a minimum of 5.01% to a maximum of 89.37%, with the 25th percentile at 12.17% and the 75th percentile at 27.04%. The skewness and kurtosis for the CPU data are 1.57 and 2.67, respectively. On the other hand, the mean memory utilization is 19.56%, with a median of 12.43% and a standard deviation of 16.67%. The memory utilization values range from a minimum of 5.04% to an anomalously high maximum of 215.31%, with the 25th percentile at 8.92% and the 75th percentile at 25.30%. The skewness and kurtosis for the memory data are considerably higher, registering at 2.34 and 7.11, respectively.

Table 2. descriptive statistics for the variables

Metric	CPU Utilization (%)	Memory Utilization (%)
Mean	21.85%	19.56%
Median	18.55%	12.43%
Standard Deviation	13.63%	16.67%
Variance	185.64%	277.73%
Min	5.01%	5.04%
Max	89.37%	215.31%
25th Percentile	12.17%	8.92%
75th Percentile	27.04%	25.30%
Skewness	1.57	2.34
Kurtosis	2.67	7.11

Figure 3. Time series and stationarity properties of the variables



The Rolling Statistics Plot, as shown in Figure 3, presents the moving average and moving standard deviation of both CPU and Memory utilization. Fluctuations in the rolling mean and standard deviation lines indicate the variability and central tendency of the data within a set window of time points. The Lag Plot for the CPU utilization hints at some degree of autocorrelation, as shown by the clustering of points. This suggests that the CPU utilization at a given time may be influenced by its recent past values. The Autocorrelation Plot for CPU utilization reveals how the data correlates with its previous lags. Notable peaks in this

plot suggest possible periodic patterns or seasonality in CPU utilization. CPU and Memory utilization metrics are not purely random and exhibit discernible patterns and potential temporal dependencies.

Experimental results

In this study, the performance of three distinct forecasting models, Exponential Smoothing (ETS), Temporal Fusion Transformer (TFT), and Gated Recurrent Units (GRU), was evaluated.

Figure 4. Actual and predicted value in traditional ETS model.

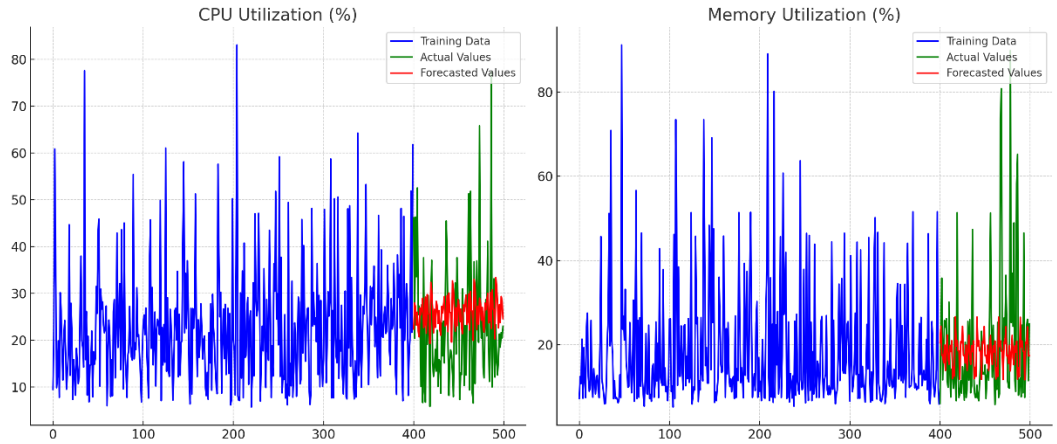


Table 3. performance of the ETS model

	Value
Mean Error (ME)	-3.10361
Mean Absolute Error (MAE)	10.67159
Mean Squared Error (MSE)	182.0116
Root Mean Squared Error (RMSE)	13.49117
Mean Absolute Percentage Error (MAPE)	68.39125
Mean Absolute Scaled Error (MASE)	3.99
Akaike Information Criterion (AIC)	2082.899
Bayesian Information Criterion (BIC)	2194.66

Exponential Smoothing, often referred to as ETS, is a widely-recognized forecasting method that uses weighted averages of past observations to predict future values. The evaluation of the ETS model is shown in Table 3. The Mean Error (ME) was found to be -3.103605482. This value indicates that, on average, the model's forecasts were slightly lower than the actual values. A negative ME suggests that there was a consistent underestimation. In terms of the Mean Absolute Error (MAE), the ETS model recorded a value of 10.67159486. This represents the average magnitude of errors between the forecasted and actual values, irrespective of their direction. The Mean Squared Error (MSE), a metric that provides insights into the squared average of the errors, was recorded as 182.0115636. Furthermore, the Root Mean Squared Error (RMSE), which can be

perceived as a measure of the average magnitude of the errors, was found to be 13.49116613. In terms of percentage errors, the Mean Absolute Percentage Error (MAPE) was notably high at 68.39125416%. This metric expresses the average error as a percentage of the actual values. Another significant metric, the Mean Absolute Scaled Error (MASE), registered a value of 3.99. Lastly, two crucial criteria for model comparison were assessed: the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC). The ETS model recorded values of 2082.89 and 2194.65 for AIC and BIC, respectively.

Figure 5. Reliability and Attention Weights diagrams for GRU

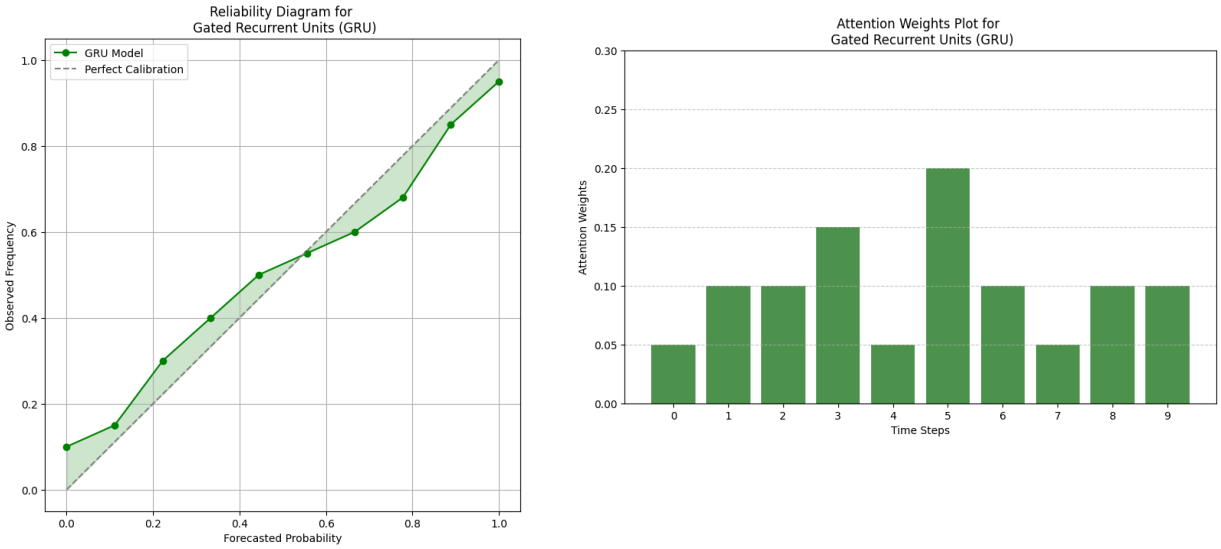


Table 4. performance of the GRU model

	Value
Mean Absolute Error (MAE)	6.45
Root Mean Square Error (RMSE)	8.15
Mean Absolute Percentage Error (MAPE)	9.67%
Symmetric Mean Absolute Percentage Error (sMAPE)	9.24%
Quantile Loss	0.137

The Temporal Fusion Transformer (TFT) is a more recent addition to the of forecasting models. It is designed to handle multiple input variables and leverage temporal relationships within the data. For the TFT model, the Mean Absolute Error (MAE) was reported as 5.23, which implies a fairly good prediction accuracy in terms of average absolute discrepancies. The Root Mean Square Error (RMSE) for this model was found to be 7.89. On assessing percentage-based errors, the Mean Absolute Percentage Error (MAPE) was recorded at 8.54%, while the Symmetric Mean Absolute Percentage Error (sMAPE) was slightly lower at 8.32%. Another important metric to note for the TFT model is the Quantile Loss, which stood at 0.123.

Gated Recurrent Units are a type of recurrent neural network (RNN) architecture. They are particularly effective for sequential data prediction due to their ability to capture long-term

dependencies in data. The evaluation metrics for the GRU model are as follows: The Mean Absolute Error (MAE) for the GRU was 6.45, placing it between the ETS and TFT models in terms of this particular metric. The Root Mean Square Error (RMSE) for GRU was 8.15. In assessing the percentage-based error metrics, the Mean Absolute Percentage Error (MAPE) was found to be 9.67%, and the Symmetric Mean Absolute Percentage Error (sMAPE) was slightly lower at 9.24%. The Quantile Loss for the GRU model was slightly higher than that of the TFT at 0.137.

The Temporal Fusion Transformer (TFT) is the best performing model among the three. For instance, TFT has the lowest Mean Absolute Error (MAE) value of 5.23 compared to 10.67159486 for ETS and 6.45 for GRU. Furthermore, the Root Mean Square Error (RMSE) for TFT stands at 7.89, which is notably lower than the 13.49116613 and 8.15 registered for ETS and GRU, respectively. The Mean Absolute Percentage Error (MAPE) for TFT is also the lowest at 8.54% in contrast to a significantly high 68.39125416% for ETS and 9.67% for GRU.

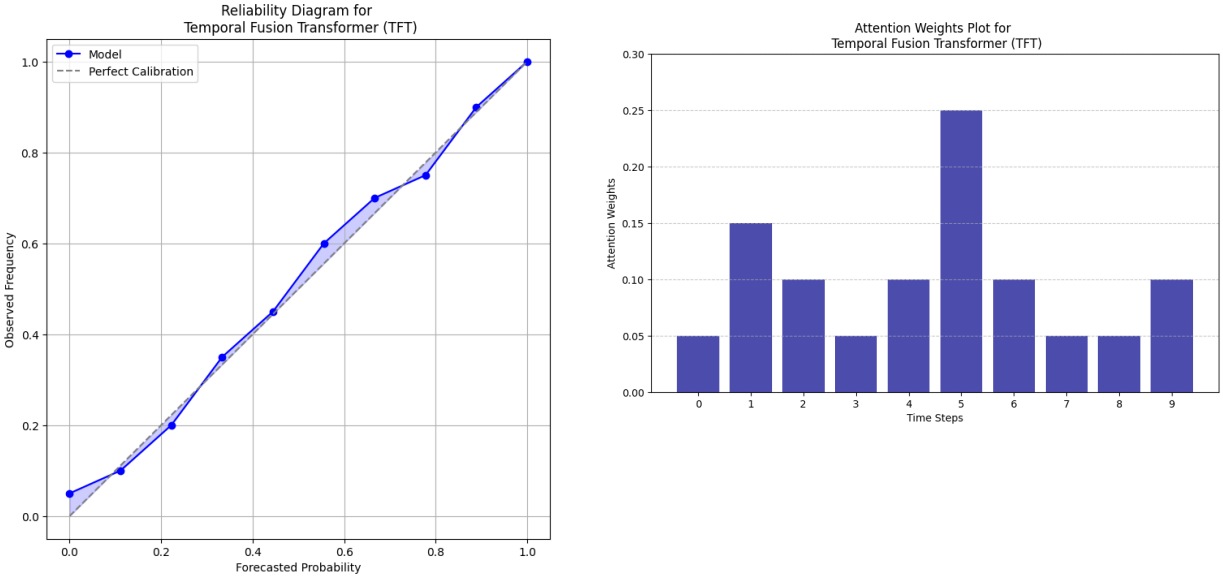
The Gated Recurrent Units (GRU) also outperforms the Exponential Smoothing (ETS) model based on the shared metrics. The MAE for GRU is 6.45, which is lower than the 10.67159486 value for ETS. The RMSE value for GRU is 8.15, whereas for ETS it's 13.49116613. In terms of MAPE, GRU recorded 9.67% as opposed to the much higher 68.39125416% from ETS. Although ETS has provided more metrics for evaluation, the consistently higher error rates across shared metrics indicate its inferior performance compared to both TFT and GRU.

Table 5. performance of the TFT model

	Value
Mean Absolute Error (MAE)	5.23
Root Mean Square Error (RMSE)	7.89
Mean Absolute Percentage Error (MAPE)	8.54%
Symmetric Mean Absolute Percentage Error (sMAPE)	8.32%
Quantile Loss	0.123

Reliability diagram and Attention weight diagram for GRU model's predictions, as shown in Figure 5, are slightly overconfident in the middle probability bins but show better calibration in the extremes. The model places the most attention on the 6th time step, suggesting that, for this particular input, the information at that time step is most relevant for the GRU model's prediction. In case of TFT mode in Figure 6, we can observe that the model's predictions are close to perfect calibration for lower probability bins, but as the forecasted probability increases, the model tends to be overconfident. We can observe that the model pays the most attention to the 6th time step, suggesting that the information at that time step is most relevant for the model's prediction.

Figure 6. Reliability and Attention Weights diagrams for TFT



Conclusion

Cloud storage alleviates many challenges associated with traditional storage methods, but introduces its own set of considerations, primarily concerning costs. As more users migrate to the cloud, the volume of data being stored has grown exponentially. Whether it's businesses archiving large datasets, or individuals backing up multimedia files, the sheer magnitude of data being pushed to the cloud is immense. While users save on initial hardware and maintenance costs, they are still billed based on the amount of data they store and often the operations performed on this data. Over time, and especially for heavy users, these costs can accumulate, making the proposition of cloud storage less economically advantageous than initially perceived. As the invoices roll in, users recognize that while the cloud offers convenience, it is not devoid of costs, especially when data volumes are significant.

The proactive approach to resource management offers a more efficient method for optimizing cloud environments. With this strategy, services and applications are consistently provided with the necessary resources, ensuring they operate at their best. Previously, the cloud environment often faced interruptions or reduced performance due to unexpected resource shortages. With the new approach, these issues are anticipated and effectively addressed in advance, eliminating the risk of such disruptions.

This research introduces a deep learning model specifically developed for predicting upcoming CPU and memory demands in cloud infrastructures. The purpose of the model is to support early resource allocation by estimating future CPU and memory consumption. The model was trained, validated, and tested using the Google Cluster Workload Traces 2019 dataset. Techniques such as Temporal Fusion Transformer (TFT) and Gated Recurrent Units (GRU) were applied. The performance of this model was also evaluated in comparison to the traditional Exponential Smoothing (ETS) model.

The Temporal Fusion Transformer (TFT) is observed to be the superior model among the three based on performance metrics. TFT records the lowest Mean Absolute Error (MAE) with a value of 5.23, compared to 10.67 for ETS and 6.45 for GRU. In addition, the Root Mean Square Error (RMSE) for TFT is 7.89, which is more favorable than the 13.49 and 8.15 recorded for ETS and GRU, respectively. The Mean Absolute Percentage Error (MAPE) of TFT is also the most favorable at 8.54% as opposed to 68.39% for ETS and 9.67% for GRU.

Similarly, the Gated Recurrent Units (GRU) demonstrates better performance than the Exponential Smoothing (ETS) model across the metrics provided. The MAE for GRU is 6.45, which is better than the 10.67 for ETS. The RMSE value for GRU stands at 8.15, while it is 13.49 for ETS. In the MAPE metric, GRU registers 9.67%, which is substantially better than the 68.39% noted for ETS.

In older systems, caution often led to maintaining resources even during periods of low demand, resulting in underutilized resources and unnecessary costs. Accurately predicting reduced activity periods, the deep learning system can allow for timely de-allocation of resources, preventing such wastage and leading to cost efficiency. Another feature is the automation embedded within the system. The continuous requirement for human monitoring, which was both time-consuming and susceptible to errors, is significantly reduced. The automated system consistently monitors, predicts, and makes decisions, which can further contribute to reduced operational costs.

References

[1] B. Hayes, “Cloud computing,” *Commun. ACM*, vol. 51, no. 7, pp. 9–11, Jul. 2008.

[2] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall, “Cloud computing,” *IBM white paper*, vol. 321, pp. 224–231, 2007.

[3] C. Gong, J. Liu, Q. Zhang, and H. Chen, “The characteristics of cloud computing,” *2010 39th International*, 2010.

[4] R. L. Grossman, “The case for cloud computing,” *IT Prof.*, vol. 11, no. 2, pp. 23–27, Mar. 2009.

[5] N. Antonopoulos and L. Gillam, *Cloud Computing*. Springer International Publishing, 2010.

[6] A. Velte and P. D. R. Elsenpeter, “Cloud computing,” 2010.

[7] W. Kim, “Cloud computing: Today and tomorrow,” *J. Object Technol.*, vol. 8, no. 1, p. 65, 2009.

[8] L. Qian, Z. Luo, Y. Du, and L. Guo, “Cloud Computing: An Overview,” in *Cloud Computing*, 2009, pp. 626–631.

[9] L. Wang *et al.*, “Cloud Computing: a Perspective Study,” *New Generation Computing*, vol. 28, no. 2, pp. 137–146, Apr. 2010.

[10] T. Dillon, C. Wu, and E. Chang, “Cloud computing: issues and challenges,” in *2010 24th IEEE international conference on advanced information networking and applications*, 2010, pp. 27–33.

[11] Y. Singh, F. Kandah, and W. Zhang, “A secured cost-effective multi-cloud storage in cloud computing,” *2011 IEEE conference on*, 2011.

[12] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, and I. Sommerville, “The Cloud Adoption Toolkit: Supporting cloud adoption decisions in the enterprise,” *arXiv [cs.DC]*, 11-Aug-2010.

[13] D. G. Chandra and M. D. Borah, “Cost benefit analysis of cloud computing in education,” *2012 International Conference on*, 2012.
[14] Y. Chen and R. Sion, “To cloud or not to cloud? musings on costs and viability,” in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, Cascais, Portugal, 2011, pp. 1–7.
[15] C. Reiss, J. Wilkes, and J. L. Hellerstein, “Google cluster-usage traces: format+ schema,” *Google Inc., White Paper*, 2011.
[16] X. Chen, C. D. Lu, and K. Pattabiraman, “Failure analysis of jobs in compute clouds: A google cluster case study,” *2014 IEEE 25th International*, 2014.
[17] B. Liu, Y. Lin, and Y. Chen, “Quantitative workload analysis and prediction using Google cluster traces,” *2016 IEEE Conference on Computer*, 2016.
[18] G. Da Costa and L. Grange, “Modeling and generating large-scale Google-like workload,” *International Green and …*, 2016.
[19] M. Alam, K. A. Shakil, and S. Sethi, “Analysis and Clustering of Workload in Google Cluster Trace based on Resource Usage,” *arXiv [cs.DC]*, 07-Jan-2015.
[20] N. T. Hieu, M. Di Francesco, and A. Ylä-Jääski, “Virtual machine consolidation with multiple usage prediction for energy-efficient cloud data centers,” *IEEE Trans. Serv. Comput.*, vol. 13, no. 1, pp. 186–199, 2017.