Prediction of Resource Demand for Virtual Machines in Clouds

by

Mitra Khezli

A thesis submitted to the
Department of Computer Science
in conformity with the requirements for
the degree of Master of Science

Bishop's University
Canada
December 2025

Abstract

The emergence of cloud resources represents a gradual and powerful evolution in computing, stemming from a convergence of technological advancements. In fact, the rise of the cloud is a direct response to the limitations and inefficiencies of traditional self-hosted IT infrastructures. These resources include a wide range of on-demand services such as computing power, storage, databases, networking, and software delivered over the Internet. Cloud computing infrastructures are highly dependent on accurate forecasting of virtual machine resource demands to ensure optimal performance, cost-efficiency, and service quality. The prediction of cloud resource utilization is a critical determinant of an organization's operational costs, system performance, and service reliability. Forecasting future resource usage facilitates organizations in mitigating the risks associated with both excessive resource allocation (over-provisioning) and insufficient capacity (under-provisioning). To predict cloud resource demand, this paper introduces a hybrid deep learning architecture for forecasting cloud resource consumption, integrating Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM), Gated Recurrent Units (GRUs), and attention mechanisms. A comprehensive evaluation was conducted on the Bitbrain dataset, comparing the proposed model against established architectures, including LSTM, GRU, and others. The results indicate the notable performance improvements of the proposed model, particularly in its ability to accurately predict peak resource consumption, which is critical for maintaining cost-effective and reliable cloud infrastructures. For the specific task of CPU consumption forecasting, the model achieved an R^2 score of 0.93, representing a 27 percent improvement over a standard LSTM model. In addition, an ablation study and a feature importance analysis were performed. The study revealed that convolutional layers with different filter sizes were the most significant contributors to the high accuracy of the model. The feature analysis identified memory-related metrics as the most influential predictors of CPU consumption, while network-related metrics had the least impact.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Professor Madjid Allili, and Co-Supervisor Professor Mohammed Ayoub Alaoui Mhamdi for their invaluable guidance and support throughout this research. The faculty members of the Computer Science department have played a significant role in shaping my academic journey, and for that, I am genuinely thankful. I would also like to thank my husband for his unwavering support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Recently, the growth of cloud computing has fundamentally reshaped digital infrastructure, providing organizations across sectors with exceptional scalability, accessibility, and affordability. This paradigm shift has transformed how businesses manage and utilize data, enabling them to respond agilely to dynamic market demands and operational needs [39]. In the cloud, data is consolidated by third-party hardware such as remote cloud servers or local edge computers. At the core of cloud infrastructure are virtual machines (VMs), which enable users to dynamically allocate and manage essential computing resources, such as CPU, memory, disk I/O, and network bandwidth, according to their workload requirements [17]. These VMs grant on-demand access to virtualized resources, enabling flexible and optimized resource management. However, this flexibility introduces complex challenges in accurately predicting resource utilization. A strong correlation between predicted and actual demand is therefore essential. Accurate forecasts support proactive resource management, ensuring system reliability, cost optimization, and informed decision-making. In contrast, weak correlation undermines predictive models, rendering them unreliable for strategic planning. Overestimating demand leads to unnecessary expenditures that contradict the pay-as-you-go model of cloud services, while underestimation results in insufficient capacity, degraded performance, service outages, financial loss, reputational harm, and violations of Service Level Agreements (SLAs). To address this challenge, this study proposes a time-series–oriented model and benchmarks its performance against existing approaches using metrics such as R^2, MSE, RMSE, and MAE. Beyond general accuracy, the model emphasizes forecasting peak demand periods, which are more critical than average usage predictions. Failure to anticipate surges results in slowdowns, errors, or outages, ultimately leading to user dissatisfaction. Thus, this research contributes a model that not only improves overall demand prediction but also enhances the ability to forecast critical peak utilization periods, thereby ensuring both operational reliability and cost efficiency in cloud environments [36].

Traditional time series forecasting models, such as the Autoregressive Integrated Moving Average (ARIMA) and Holt-Winters methods, have laid the foundation for early workload prediction systems. Nonetheless, their reliance on linearity and stationarity assumptions limits their effectiveness in capturing the complex, high-frequency, and nonlinear behaviors typical of modern cloud environments[18]. In response, deep learning models such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRU) have demonstrated superior capabilities in modeling sequential dependencies. However, when deployed in isolation, these models still face challenges in maintaining long-term memory, handling vanishing gradients, and scaling effectively across large, multistep, or multivariate prediction scenarios. Moreover, the existing literature has primarily centered on forecasting individual resource types, most often CPU utilization, while neglecting the need to jointly model other critical metrics such as memory usage, disk I/O, and network throughput. This narrow focus diminishes the practical applicability of such models in real-world data centers, where comprehensive and coordinated resource provisioning is essential. For example, while Shaikh et al. extended forecasting to include network throughput alongside CPU usage, their model does not incorporate memory or disk I/O, both of which play pivotal roles in performance optimization and cost control [22]. Additionally, recent research has highlighted the importance of attention mechanisms in enhancing model performance and interpretability. Models such as BHyPreC, which employ hybrid BiLSTM-BiGRU architectures, have demonstrated strong predictive accuracy but explicitly acknowledge the absence of attention mechanisms as a limitation, particularly in terms of dynamic feature prioritization and temporal relevance. Similarly, ensemble-based approaches proposed by Shetty et al, address multivariate forecasting but rely heavily on manual feature extraction and lack unified temporal modeling capabilities [28].

To address these challenges, namely, the inability to model multiple resource types jointly, the lack of dynamic temporal focus, and limited interpretability, this thesis proposes a stacked hybrid deep learning model. The architecture integrates Convolutional LSTM (ConvLSTM), attention mechanisms, LSTM, and GRU components to form a comprehensive forecasting framework. Specifically, the proposed model jointly forecasts CPU, memory, disk I/O, and network throughput to provide a holistic multivariate prediction; incorporates attention layers to dynamically weigh relevant temporal patterns, enhancing both forecasting accuracy and model transparency; and leverages the combined strengths of ConvLSTM for spatial-temporal correlation and recurrent layers for capturing short- and long-term dependencies. Through this approach, the thesis aims to enhance methods for cloud resource demand forecasting.

1.2 Thesis Outline

Chapter 2 focuses on the literature review concerning resource allocation in cloud computing's virtualized environments, emphasizing the significant role of efficient resource allocation and the inherent challenges in forecasting. This chapter comprehensively analyzes various predictive models, including traditional statistical methods and machine learning approaches, and introduces a hybrid model. It discusses in detail the models utilized in this thesis.

Chapter 3 elaborates on the methodology and implementation details of the study, beginning with a thorough description of the new deep-stack model and the dataset collection process, including preprocessing steps. It discusses feature engineering and preprocessing techniques, with an explicit focus on data-denoising methods to prepare the data for modeling. It concludes with an evaluation of the metrics used for model performance assessment and comparison with statistical and other deep-learning models to underscore the proposed model's effectiveness.

In Chapter 4, the thesis concludes by summarizing its contributions to forecasting cloud usage metrics. It highlights the methodology and models introduced, discussing their significance in enhancing predictive accuracy. The chapter also acknowledges the limitations of the current study, offering a candid discussion of areas that require further research. It outlines potential future directions, suggesting avenues for advancing the field of cloud resource forecasting through refined models and innovative approaches.

Chapter 2

Literature Review

As mentioned, predicting resource usage in cloud environments is a cornerstone of achieving operational efficiency, cost-effectiveness, and high user satisfaction in cloud computing. Resource prediction in cloud computing has evolved from traditional statistical methods to advanced deep-learning approaches. Early studies primarily focused on time-series models such as ARIMA, Holt-Winters, and Unobserved Component Models (UCM), which provided a solid statistical foundation for forecasting workload demand. For example, Kirchoff et al. (2019) [19] evaluated ARIMA, MLP, and GRU for long-term workload forecasting using NASA HTTP traces. While GRU demonstrated superior efficiency over ARIMA, these models required frequent updates and struggled with capturing highly dynamic cloud workloads.

To address the limitations of traditional models, researchers have explored hybrid machine learning techniques. Maswood et al. (2021) [16] proposed BHyPreC, a hybrid architecture that combines Bi-LSTM, GRU, and 1D convolution layers, significantly improving multi-step-ahead workload predictions. Their method also introduced an optimized grid search technique for CPU usage forecasting, demonstrating greater adaptability than standalone deep learning models. However, the reliance on recurrent networks still posed challenges for long-term sequence modeling due to vanishing gradient issues.

In response to these challenges, convolutional neural networks (CNNs) have emerged as a powerful addition to deep-learning models for time-series forecasting. Unlike recurrent models, CNNs excel at extracting spatial patterns and temporal dependencies in a more computationally efficient manner. Khan et al. (2022) [17] explored hybrid architectures that combined CNNs with LSTM and GRU, demonstrating improved feature extraction and faster processing times. Similarly, in another study, Shetty et al. (2023) [28] proposed an ensemble model combining Vector Autoregression (VAR) with LSTM [42], showcasing the advantages of integrating statistical and deep-learning methods. Despite their accuracy, these models often suffer from high computational overhead, making them less scalable for real-world cloud environments.

To further refine predictive accuracy while maintaining efficiency, St-Onge et al. (2021) [31] have explored newer research in advanced hybrid deep-learning stacks. This workload estimation approach integrates the Hull-White stochastic differential equation with a Genetic Algorithm (GA), which provides a novel way to model cloud resource fluctuations dynamically. In another study, Shaikh et al. (2024)[20] compared deep-learning models, such as BiLSTM, against traditional regression techniques on cloud datasets, confirming that deep networks significantly outperform classical statistical methods in handling complex temporal patterns. To enhance cloud resource management and reduce power consumption, a deep learning-based hybrid model is proposed for the accurate forecasting of Virtual Machine (VM) workloads. The findings indicate that the proposed CNN-LSTM model achieves superior performance, outperforming both standalone CNN and LSTM models [21]. Zhang et al. utilize a hybrid CNN-BiLSTM model to predict server load by forecasting CPU utilization. The objective is to establish a dynamic prediction framework that enables data centers to adjust resource allocation promptly and effectively. The findings indicate that the proposed CNN-BiLSTM model demonstrates high predictive accuracy and significantly outperforms other established models, such as standalone CNN and CNN-LSTM architectures [40]. In response to the inadequacy of classical machine learning models for predicting time-varying and non-linear cloud workloads, the study by Patel et al. asserts the critical need for a multivariate approach that considers the interdependencies between resource metrics, rather than a univariate method. To this end, the paper proposes MAG-D, a novel deep learning model designed to accurately forecast chaotic workloads in cloud data centers. The presented MAG-D architecture is a sophisticated framework that integrates several advanced techniques, including Multivariate Analysis, an Attention Mechanism, stacked bi-GRU and bi-LSTM layers, and Clustering. The research concludes that the MAG-D architecture can reliably model volatile workload variations, making it an effective solution for real-time forecasting scenarios [23].

Building upon these advancements, our research proposes a deep-learning stack model that integrates multiple architectures, including LSTM, GRU, and CNN. Unlike previous approaches that rely solely on individual models or simple hybrid combinations, our deep-stack framework leverages the strengths of each model type to improve cloud resource forecasting. By combining CNNs for feature extraction, recurrent networks for capturing short-term trends, and transformers for handling long-range dependencies, our approach achieves higher predictive accuracy, improved scalability, and enhanced adaptability to dynamic cloud workloads.

To provide a more straightforward comparative overview of the studies discussed above, Table 2.1 summarizes their key characteristics, including the prediction methods, datasets used, evaluation intervals, and the best-performing models. This tabular representation reinforces the progression from traditional statistical approaches to more complex hybrid and deep-learning-based solutions.

Table 2.1: Summary of research on cloud predictive models

Refrences	Prediction Approach	Implemented Models	Dataset	Time Intervals	Results	Best-performed Model
[19]	Two months	Auto-Regressive Integrated Moving Average (ARIMA), Multilayer Perceptron (MLP), Gated Recurrent Unit (GRU)	NASA HTTP Traces	Every 5 minutes	MSE=16.51	GRU
[16]	Long-term traces	LSTM, Bi-LSTM, BHyPreC (Hybrid), GRU	Bitbrain servers	Every 5 minutes	RMSE=0.006	BHyPreC
[3]	Short term	WGAN-gp Transformer (Transformer, WGAN-gp)	Custom cloud workload traces	Every 5 minutes	RMSE=1.81, MAPE=1.49, MAE=1.40	Transformer
[28]	Short-term	VAR, LSTM, Weighted gride search (VAR+LSTM)	Bitbrains Servers	Every 5 minutes	MSE=(0.3-0.8)	Weighted Ensemble (VAR+LSTM
[31]	Short-term	Hull-White SDE, Genetic Algorithm (GA), Support Vector Regression (SVR)	Bitbrains Servers	Every 5 minutes	MSE=0.5, Mape=0.071	Hull-White-GA
[27]	Short-term	Linear Regression, Decision Tree Regression, Gradient Boosting, Support Vector Regression, LSTM, BiLSTM	Bitbrains Servers, Microsoft Azure Traces	Every 5 minutes	RMSE=0.049, R^2=0.804	BiLSTM
[5]	Short-term/Long-term	RNN, LSTM, GRU, 1D-CNN, attention-based LSTM	Bitbrains Servers, Google Trace, Alibaba Trace	Every 5 minutes	RMSE=0.1-0.2	GRU under transfer attacks
[23]	MAG-D	LSTM, BiLSTM,GRU, CNN, attention	Google cluster dataset	Every 5 minutes	RMSE=0.00378	MAG-D

Chapter 3

Methodology and Implementations

3.1 Introduction

This chapter describes the methodology employed in this study, providing a comprehensive overview of the research environment and dataset. It introduces the performance evaluation criteria, explaining the metrics used to assess the accuracy and reliability of the proposed model. The chapter also concludes with a comparative analysis of statistical models. To improve the readability of the study, the sections describing the Holt, UCM, LSTM, GRU, and Attention models have been placed in Appendix A.

3.2 Proposed Stack Hybrid Model

Stacking (or Stacked Generalization) is an advanced machine learning technique that enhances model accuracy by combining multiple base models. In this method, several base models are mixed, and their results are fed into a meta-model to make the final prediction. The stacking method enhances prediction by combining base models through meta-models, resulting in more precise and stable outcomes. When traditional empirical models are appropriately integrated, they can produce more accurate predictions. The fundamental concept of stacking is to train multiple diverse base models and then use a meta-model to merge their predictions, ultimately yielding the final prediction. Constructing a stacking model requires two key components: (1) base models, which are trained on the training data, and (2) a meta-model, which is designed to integrate the outputs of the base models. The role of the meta-model is to learn a function that effectively combines the predictions of the input features from the models to produce the final prediction. In this study, a meta-regressor has been used, which is trained using the outputs of the base models

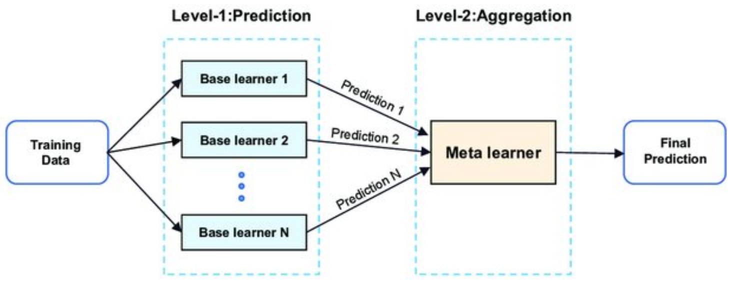

Figure 3.1: Stack model

as its inputs, effectively treating these outputs as new features. By doing this, the meta-regressor learns the strengths and weaknesses of each base model, adjusting its predictions to emphasize the most reliable outputs while compensating for any individual model's inaccuracies [30]. Figure 3.1 overviews a general diagram of a hybrid stack model.

Recent advances in cloud resource forecasting have highlighted several persistent challenges, including the inability of traditional and simple deep learning models to effectively capture multi-scale temporal patterns, prioritize critical events, and maintain computational efficiency. Previous models, such as ARIMA, LSTM, GRU, and hybrid CNN-RNN combinations, often suffered from vanishing gradient problems, limited scalability, and an equal treatment of all input sequences regardless of their importance. To address these limitations, the proposed StackModel integrates multi-scale ConvLSTM layers, an attention mechanism, and parallel LSTM-GRU pathways, enabling dynamic focus on key temporal segments, robust modeling of short- and long-term dependencies, and enhanced scalability. By strategically combining these complementary architectures, StackModel overcomes critical shortcomings in prior work and offers a more resilient and accurate solution for dynamic cloud resource forecasting.

Figure 3.2 provides a detailed illustration of our proposed model, which is further explained in the subsequent sections.

The proposed model integrates ConvLSTM, Attention Mechanism, LSTM, and GRU to enhance time-series prediction by effectively capturing spatial-temporal dependencies. The architecture comprises the following key components:

Multi-Scale ConvLSTM Layer To extract hierarchical spatial-temporal features as shown in Figure 3.3, a convolutional LSTM is first positioned to capture spatial-temporal features. Since traditional LSTMs and GRUs focus on sequential relationships, the input data is processed through three parallel ConvLSTM layers with distinct filter sizes (3, 5, and 7). This multi-scale approach enables the model to capture patterns at different granularities. Each ConvLSTM layer generates three outputs: Query (Q), Key (K), and Value (V), which are subsequently used in the

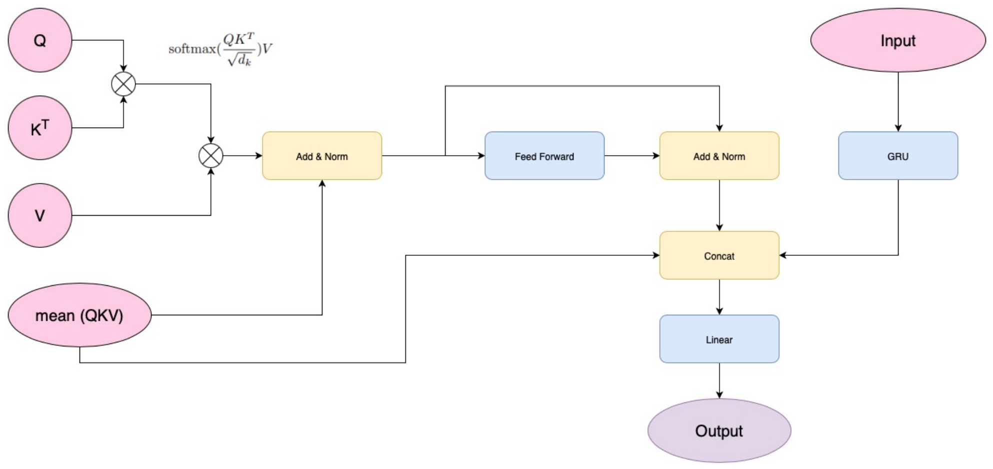

Figure 3.2: Proposed Stack model

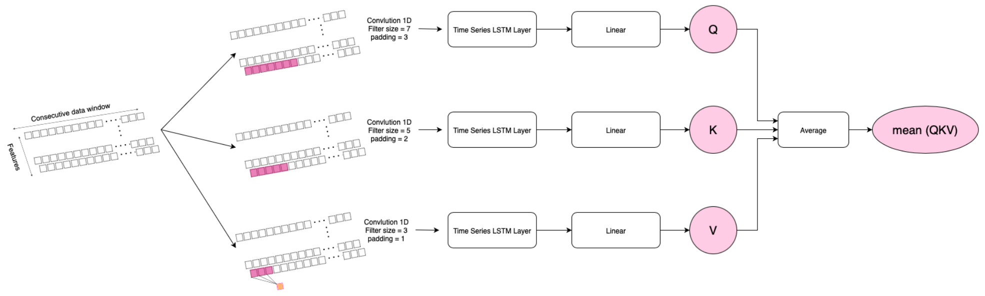

Figure 3.3: Feature Extraction Process

attention mechanism [19].

Attention Mechanism To improve the model's ability to focus on important temporal dependencies, an Attention Mechanism is applied to the ConvLSTM outputs. The attention weights are computed using the Scaled Dot-Product Attention, which has been explained in detail in Chapter 2, formulated as follows:

$$\text{Attention}\,(Q,\ K,\ V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.1}$$

where d represents the dimensionality of the Key vectors. The attention output is then passed through a linear transformation, followed by a residual connection and layer normalization to stabilize training and improve convergence [24].

LSTM Layer The concatenated Q, K, and V outputs from the attention module are fed into an LSTM layer to capture long-term temporal dependencies. The resulting

LSTM output is further enhanced by incorporating a residual connection with the attention output, ensuring effective retention of information.

GRU Layer Simultaneously, the input data is processed through a Gated Recurrent Unit (GRU) layer to efficiently model sequential dependencies while mitigating vanishing gradient issues [16]. The GRU output serves as an additional feature representation to be incorporated into the final prediction stage.

Final Aggregation and Prediction The outputs from the LSTM and GRU layers are concatenated to form a unified feature representation. A fully connected linear layer as a meta-learner, applied to transform the concatenated vector into the final output prediction [20].

Model structure This study's model processes time-series data through a sequence of one-dimensional Convolutional, LSTM, and Linear layers to produce Q, K, and V values with various filter sizes. An attention value is then computed from these using Equation 3.1. This attention output is calculated by averaging Q, K, and V, and the result is then normalized. Following this, the data is passed through a two-layer Linear network, with its output also being combined with the initial input and normalized. Finally, this processed output is concatenated with outputs from separate ConvLSTM and GRU layers and fed into a Linear meta-learner to generate the final predictions.

3.3 Dataset description and collection

The dataset used in this study originates from the Bitbrains data center, one of the most widely recognized real-world cloud workload datasets. Due to its richness and reliability, Bitbrains has been extensively employed in prior research on modeling and predicting cloud resource utilization. For example, several studies [2, 8, 20, 32, 35] employed the Bitbrains traces to develop and evaluate forecasting models, demonstrating the dataset's relevance and robustness for cloud resource prediction.

The Bitbrains data center hosts and manages enterprise workloads, making it a reliable source for analyzing cloud resource demand and performance." [8]. The dataset consists of virtual machine usage data (VM), with each VM having its own dataset (CSV) file. This trace comprises data from 1,250 virtual machines (VMs), all interconnected via a fast storage area network (SAN). Each VM file contains performance metrics, and overall, the fast storage dataset spans 5,446,811 CPU hours, includes 23,214 GB of memory, and utilizes 5,501 cores.

Table 3.1: Summary of dataset variables and statistics

Variable	**Distinct**	**Mean**	**Minimum**	**Maximum**
Network Received Throughput [KB/s]	40	0.1377	0	237.5333
Disk Write Throughput [KB/s]	265	10.2954	1.4444	1201.4666
Disk Read Throughput [KB/s]	162	0.6560	0	633.9333
Memory Usage [KB]	923	260410.0676	164974.9333	350000.0000
Memory Capacity Provisioned [KB]	2	2097150.448	2083753.455	2097152
CPU Usage [MHz]	280	33.8243	27.7333	259.9993
CPU Capacity Provisioned [MHz]	22	2599.9993	2599.9987	2599.9996
Timestamp [ms]	8633 (100%)	1377610964	1376314846	1378906798

3.3.1 Dataset Overview

Each data point in the Bitbrains dataset includes a time-stamp in Unix format, uniquely identifying the moment of data capture and allowing for precise tracking of changes over time. The dataset records the number of CPU cores available to each VM and the total CPU capacity provisioned, measured in megahertz (MHz). These CPU-related metrics are complemented by actual usage data, indicating how Memory resources are similarly detailed, with metrics on the total memory capacity provisioned for each VM, recorded in kilobytes per second (KB/s).

Data throughput is another critical aspect captured, encompassing both disk and network performance. Disk read throughput measures the rate at which data is transferred from the disk to the VM, while disk write throughput records data being written from the VM back to the disk. Both are measured in kilobytes per second (KB/s). Network performance is tracked through two separate metrics: network received throughput, which measures the rate at which the VM gets data from the network, and network transmitted throughput, which tracks data sent from the VM to the network. Table 3.1 provides a comprehensive dataset summary detailing the distinct values, means, minimums, and maximums of each variable, followed by two graphs depicting the resource consumption pattern.

Figures 3.4 and 3.5 display the averaged values for each resource metric across all virtual machines in the Bitbrain dataset.

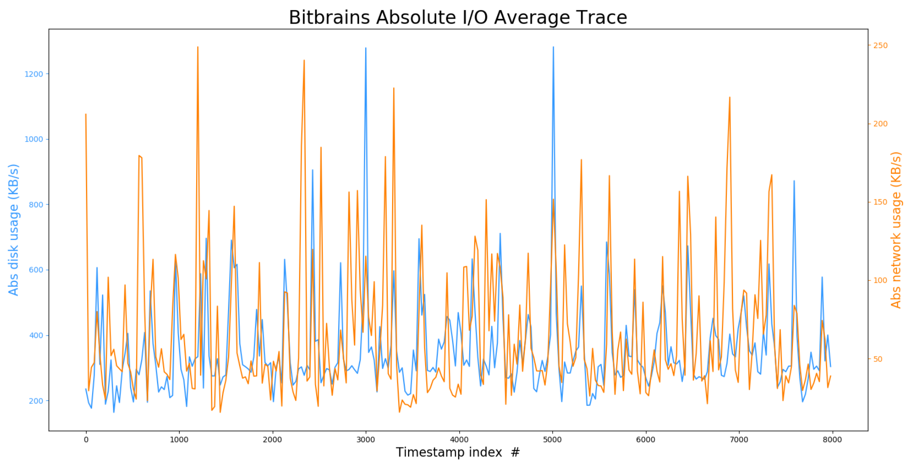

Figure 3.4: DataSet Trace Disk-I/O

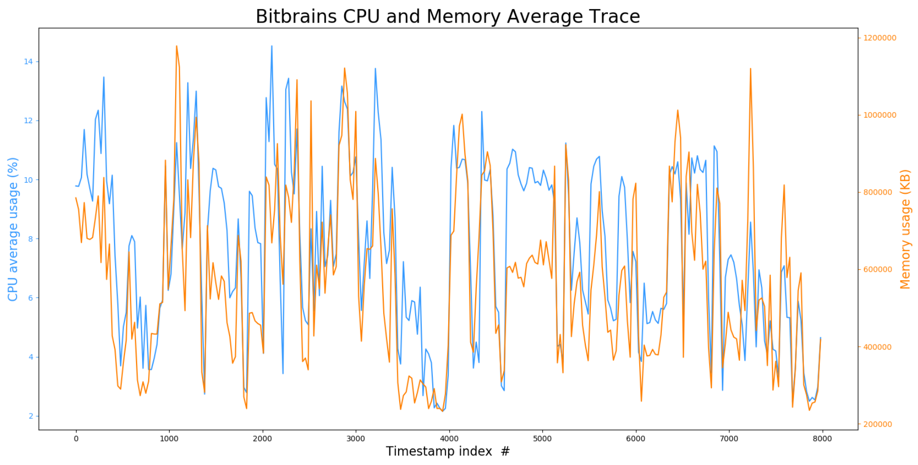

Figure 3.5: DataSet Trace CPU-Memory

3.4 Data Normalization and Preprocessing

The Bitbrain dataset is structured around four resource metrics: CPU, memory, network, and disk utilization. The objective of this study is to predict the resource consumption per virtual machine (VM) for each resource. To achieve this, each resource category for a given VM is modeled independently, resulting in the creation of four separate datasets per VM. Developing a dedicated model for each resource category has been found to yield better performance compared to utilizing a single, comprehensive model for all categories.

Although the raw data was originally captured at millisecond granularity, it has been resampled by averaging over 5-minute intervals. Consequently, the effective temporal resolution of the data used in this analysis is 5 minutes per timestamp.

Before applying predictive algorithms, it is crucial to normalize the data to mitigate bias arising from the differing scales of the dataset's features. In this study, Min-Max normalization was employed to scale each feature to a fixed range, typically between 0 and 1. This transformation enhances the comparability and performance of the predictive models. The normalization is defined as:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3.2}$$

where x is the original value, and $\min(x)$, $\max(x)$ represent the minimum and maximum values of the feature, respectively.

To train time series models, it is necessary to provide the model with historical context for each prediction. Accordingly, we constructed input sequences using a sliding window approach with a window size of 30 consecutive data points, corresponding to a 2.5-hour time interval (since each point represents 5 minutes of aggregated data).

Due to the temporal dependency inherent in time series data, traditional random sampling methods for model validation are inappropriate. Instead, the dataset was chronologically split into training and validation sets. Specifically, 80% of the earliest portion of the dataset was used for training, while the remaining 20%, representing the most recent, unseen data—was used to evaluate the model's predictive performance.

3.5 Implementation Environment

In this study, we employed a robust suite of tools and frameworks to facilitate our research in predictive modeling. PyTorch provides a powerful and flexible environment for designing and training neural network models. Preliminary data analysis was efficiently executed using Pandas Profiling, while data normalization and model evaluation were supported by Scikit-Learn, which offered reliable methods. Visual insights into the model's performance were generated using Matplotlib

Table 3.2: Hyperparameters of Statistical Model

Hyper-parameters	**Values**
Exponential Smoothing Seasonal Period	30
Trend Type (Holt-Winters)	Additive
Seasonality Type (Holt-Winters)	Additive
UCM Model Components	Trend, Seasonal (30), Cycle (AR=3)
UCM Damping Factor	Enabled
Scaler Used	MinMaxScaler (0,1)
Evaluation Metrics	MSE, RMSE, MAE

and Seaborn, facilitating the practical interpretation of results. Google Colab served as our primary platform for model deployment, leveraging its cloud-based environment to ensure the accessibility and scalability of our computational resources. This comprehensive approach not only streamlined our research processes but also bolstered the accuracy and efficiency of our predictive models.

3.5.1 Summary of Hyperparameters

Statistical

Table 3.2 outlines the specific hyperparameter configurations employed for the statistical forecasting models used in this study. Hyperparameters are crucial settings that are configured before the model training process begins. The statistical models examined in this study include the Holt-Winters and UCM models.

Proposed Deep Learning Time Series Models

The deep learning time series models include GRU, LSTM, Convolutional LSTM, and Deep Hybrid Stack. Table 3.3 reveals the hyperparameter information for these deep learning time series models. To ensure a fair comparison, a uniform set of hyperparameters was applied to all models evaluated in this study. The dataset was structured with each observation representing a 5-minute time interval. To generate a prediction for any given record, the model is fed the time series data from the previous 30 records. Regarding the training process, the Mean Squared Error (MSE) was selected as the loss function. The QHAdam optimizer was utilized to adjust the network's weights during the training process, and all models were trained for a total of 700 epochs.

Table 3.3: Hyperparameters of the deep learning time series models

Hyper-parameters	**Values**
Window Size (look_back)	30
Time Frame	5
Batch Size	256
Time series Layers	GRU, LSTM, ConvLSTM, ConvGRU
Activation Function	ReLU
Optimizer	QHadam
Loss Function	MSE
Metrics	MSE, RMSE, MAE, MAPE
Total EpochEpochs	700
Scaler Used	MinMaxScaler (0,1)

3.6 Evaluation Metrics

Metrics are essential tools for evaluating the performance of time series prediction models, providing a quantitative and objective measure of how accurately a model's forecasts match the actual outcomes. We use various evaluation metrics to estimate the accuracy of the prediction methods more accurately. In this section, Y_i represents the actual value and $\widehat{Y_i}$ denotes the predicted value. Moreover, n denotes the number of values.

The Mean Squared Error (MSE) is a metric that quantifies the average of the squares of the differences between observed and estimated values. A lower MSE score indicates that the prediction model is more precise, implying that forecasted values closely align with the real ones. The actual formula for MSE is given as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \widehat{Y_i})^2 \tag{3.3}$$

Root Mean Squared Error (RMSE) is calculated as the square root of the Mean Squared Error.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}\left(\widehat{Y_i} - Y_i\right)^2}{n}} \tag{3.4}$$

The Mean Absolute Error (MAE) calculates the mean of the absolute discrepancies between forecasted and observed values.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left|Y_i - \widehat{Y_i}\right| \tag{3.5}$$

In the fields of statistics and machine learning, the Mean Absolute Percentage Error (MAPE) is a popular metric for evaluating the performance of a predictive model. It represents the mean deviation, in percentage terms, between the forecasted and the actual values.

$$MAPE = \frac{1}{N}\sum_{t=1}^{N}\left|\frac{A_t - F_t}{A_t}\right| \tag{3.6}$$

The variable A_t represents the value for a given data point t, while F_t signifies the corresponding forecasted or predicted value.

The Coefficient of Determination, also known as the R^2 score, quantifies the percentage of the variance in the dependent variable that the independent variables can explain. It reflects how well the model fits the data.

$$R^2 = 1 - \frac{RSS}{TSS} \tag{3.7}$$

Where Y_i is the actual value, $\hat{Y}_i$ is the predicted value, RSS is the sum of squares of residuals, and TSS is the total sum of squares.s

3.7 Model Parameters and Comparison

Practical tuning of model parameters is critical for optimizing predictive performance. The parameters for this study are summarized in Table 3.4. The simulation involved using 80% of the dataset for training, while the remaining 20% was reserved exclusively for testing. This ensures that the model's generalization capability is evaluated on completely unseen data, providing a reliable measure of its actual performance in the real world.

To ensure comparability with existing research in the real world, the same dataset selections were used in our thesis as in previous studies from academic papers and publicly available GitHub repositories. Specifically, subsets of the Bitbrains dataset were chosen based on established benchmarks from prior works, ensuring that the model's results could be directly compared with those of other forecasting approaches. This alignment with existing research enhances the credibility and relevance of the findings.

The Mean Squared Error (MSE) was selected as the primary error calculation function to emphasize the penalty on larger errors, thereby enhancing model accuracy. Hyperparameter tuning was performed manually to ensure direct control over model adjustments, optimizing performance without relying on automated tuning methods. Additionally, early stopping was not required, as the model demonstrated stable convergence across training epochs, reinforcing its reliability.

Table 3.4: The simulation parameter

Parameter	Description
Training data volume	80% of the dataset
Test data volume	20% of the dataset
Dataset Selection	Based on prior studies on Bitbrain servers and GitHub repositories
Error calculation function	Mean Squared Error (MSE)

Table 3.5: CPU evaluation result

metrics	ucm	holt
MSE	5.9393	5.9393
RMSE	0.0937	0.0958
MAE	0.0762	0.0800

3.7.1 Statistical Models Evaluations

In this section, we visually compare the performance of two statistical models, UCM and Holt-Winters, which are configured to predict up to 50 time steps, corresponding to a forecasting duration of 8 hours and 20 minutes. This predictive capability is demonstrated through a series of line charts, each depicting a separate component of resource usage. The visualizations reveal limitations in the UCM and Holt-Winters forecasting approaches. These models demonstrate overlapping lines with actual data, failing to consistently predict peak periods and sudden changes in resource usage, which are key aspects that are critical in cloud environments for maintaining efficiency and reducing operational costs. This limitation underscores the limitations of traditional statistical models for today's complex and dynamic cloud resource management tasks. This comparative graphical analysis clearly illustrates why advanced predictive techniques, such as the deep-learning models proposed in this thesis, are essential for achieving higher accuracy and reliability in forecasting cloud resource metrics. Below, detailed graphs and evaluation metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE), are illustrated in detail In Figures 3.6, 3.7, 3.8, 3.9, and Tables 3.5, 3.6, 3.7, 3.8.

3.7.2 Deep Learning Models Evaluations

Figure 3.11 presents a comparative analysis of the performance of our proposed model against each baseline model, all of which were trained and evaluated on the same 1000 datasets. To enable a more interpretable and visually balanced bar chart, we applied scaling adjustments: the Mean Squared Error (MSE) values were

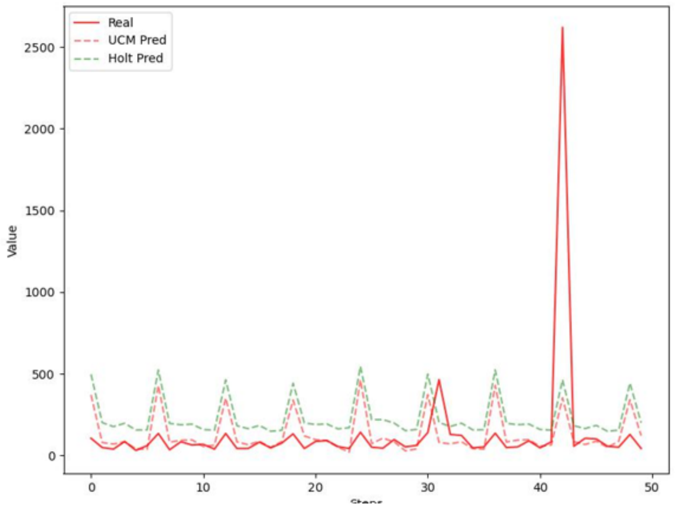

Figure 3.6: CPU usage prediction [KB/S]

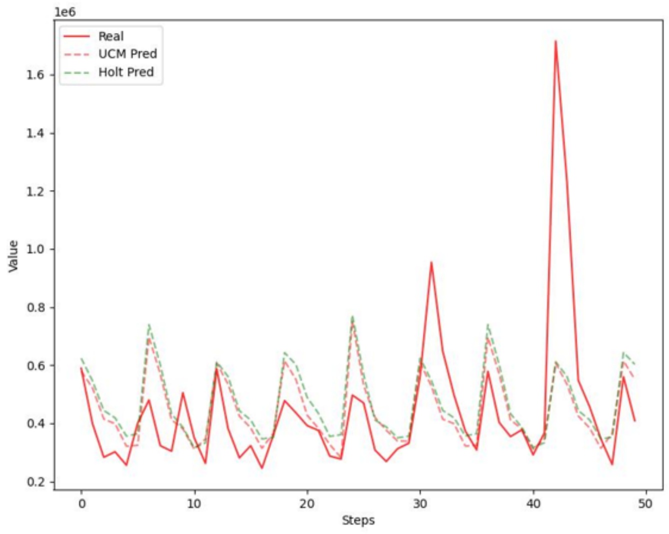

Figure 3.7: Memory usage prediction [KB/S]

Table 3.6: Memory evaluation result

metrics	**ucm**	**lstm**
MSE	9.3247	7.6911
RMSE	3.0536	2.8447
MAE	2.4582	2.2675

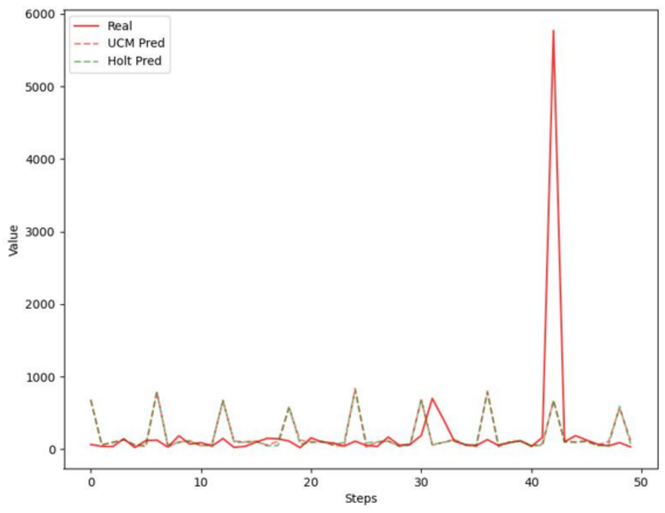

Figure 3.8: Disks read throughput prediction [KB/S]

Table 3.7: Disk read evaluation result

metrics	**ucm**	**holt**
MSE	5.5413	6.0915
RMSE	2.3540	2.4681
MAE	0.7558	1.0777

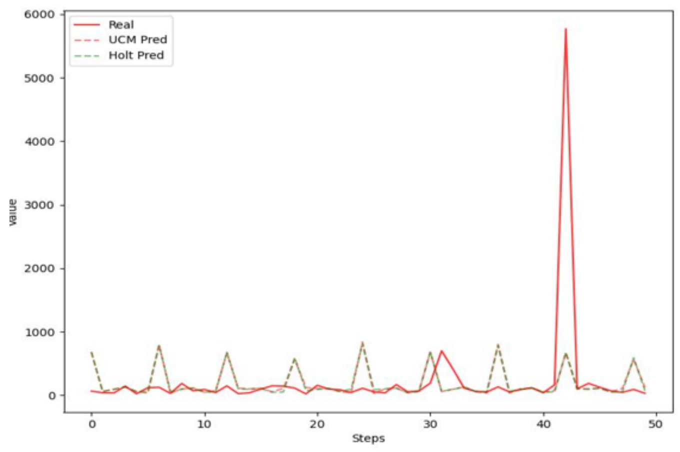

Figure 3.9: Disk write throughput prediction [KB/s]

Table 3.8: Disk write evaluation result

metrics	**ucm**	**holt**
MSE	3.69246	2.1173
RMSE	1.6201	1.6100
MAE	0.9459	0.9624

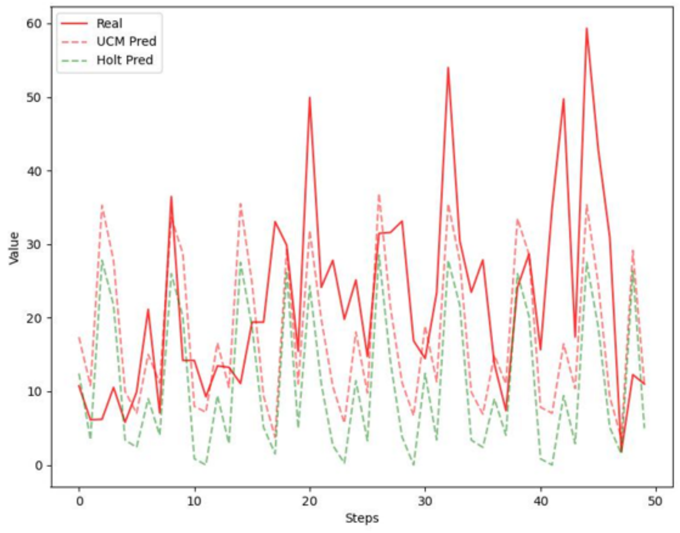

Figure 3.10: Network transmit throughput prediction [KB-S]

Table 3.9: Network transmitted throughput evaluation result

metrics	**ucm**	**holt**
MSE	0.1437	0.1462
RMSE	0.3791	0.3823
MAE	0.2357	0.2380

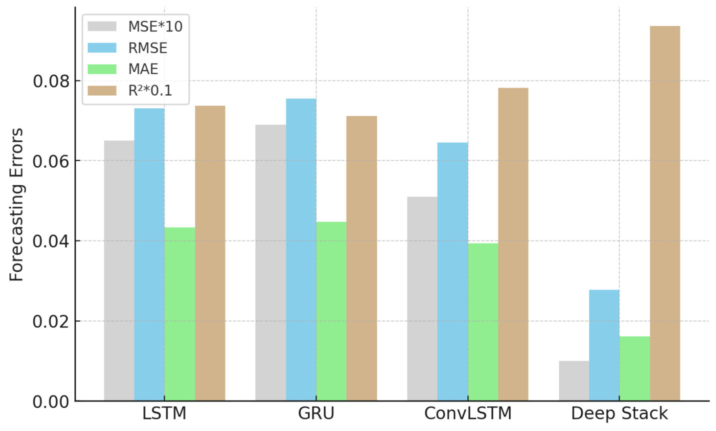

Figure 3.11: Comparative Error Analysis of Prediction Models

multiplied by 10, and the R^2 values were scaled down by a factor of 0.1. These transformations were necessary due to the inherent disparity in magnitude; MSE values are typically small, while R^2 values tend to be comparatively large. As illustrated in the figure, our proposed Deep Stack Model consistently achieves the lowest error across all evaluation metrics. In contrast, the LSTM model exhibits the highest forecasting errors. These results clearly demonstrate that our model outperforms the baseline approaches in accurately predicting VM resource usage.

Table 3.10 presents the detailed numerical results corresponding to each model and error metric, providing further support for the visual comparison given in the bar chart. For each category, the model's performance is assessed using key error metrics, Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE), whose functionalities were previously explained in Section 3.5.

The comparative analysis suggests that Table 3.10 highlights the effectiveness of the proposed Deep Stack model relative to conventional baselines (LSTM, GRU, and Convolutional LSTM). Among the models tested, the Deep Stack achieves the lowest error rates across all metrics (MSE, MAE, RMSE) while simultaneously obtaining the highest coefficient of determination (R^2). This indicates that the Deep Stack is not only more accurate but also more reliable in capturing the variance of resource demand patterns.

In contrast, the LSTM and GRU models show higher forecasting errors, which suggests that they are less effective at modeling the temporal complexity of VM workloads when used in isolation. The Convolutional LSTM performs moderately better than LSTM and GRU, benefiting from its ability to capture both spatial and temporal correlations.

Table 3.10: Deep- Learning comparison results

Model	**MSE**	**RMSE**	**MAE**	R^2
LSTM	0.0065	0.0731	0.0433	0.7369
GRU	0.0069	0.0755	0.0447	*0.7111*
Convolutional LSTM	0.0051	0.0645	0.0393	0.7811
Deep Stack	0.001	0.0278	0.0161	0.9362

Table 3.11: Deep-Stack detail results

Target	**MSE**	**MAE**	**RMSE**	R^2
CPU usage [MHZ]	0.001	0.0161	0.0278	0.9362
Memory usage [KB/s]	0.0003	0.0089	0.0133	0.9791
Disk throughput [KB/s]	0.0015	0.0110	0.0232	*0.9824*
Network throughput [KB/s]	0.0019	0.0089	*0.0426*	0.8068

Detailed Performance Analysis of the Deep Stack Model Across Resource Metrics

Table 3.11 summarizes the model's performance across CPU, memory, disk, and network usage. The model achieved high accuracy for disk (R^2 = 0.9824) and memory (R^2 = 0.9791), with CPU predictions also strong (R^2 = 0.9362). Network throughput showed lower accuracy (R^2 = 0.8068) and higher error values, reflecting the greater variability of network traffic; however, the results remain within an acceptable range. Figure 3.12 presents the training and validation curves for the model's performance using MSE, MAE, and RMSE as evaluation metrics.cs.

As evident from the graphs, the close convergence between training and validation loss across all system metrics confirms the model's practical learning and generalization capabilities. The consistent downward trend in error values over training epochs highlights the model's ability to learn meaningful patterns with minimal overfitting, further validating its robustness and reliability in predicting resource usage.

The performance predictions of the LSTM, GRU, and Stack models on the validation set for VM number 890 are presented in Figure 3.13. The results show that the Stack model outperformed the other two, demonstrating a strong capability for forecasting peak points in CPU utilization. The accurate prediction of peak points in cloud resource consumption is a crucial task in cloud management, ensuring service dependability, optimizing operational costs, and maintaining a high-quality user experience. These peak points signify the highest level of demand for computational resources, such as CPU and network bandwidth, that an application is expected to handle. A failure to anticipate and provision for these high-demand periods can result in substantial operational and commercial challenges. As clearly

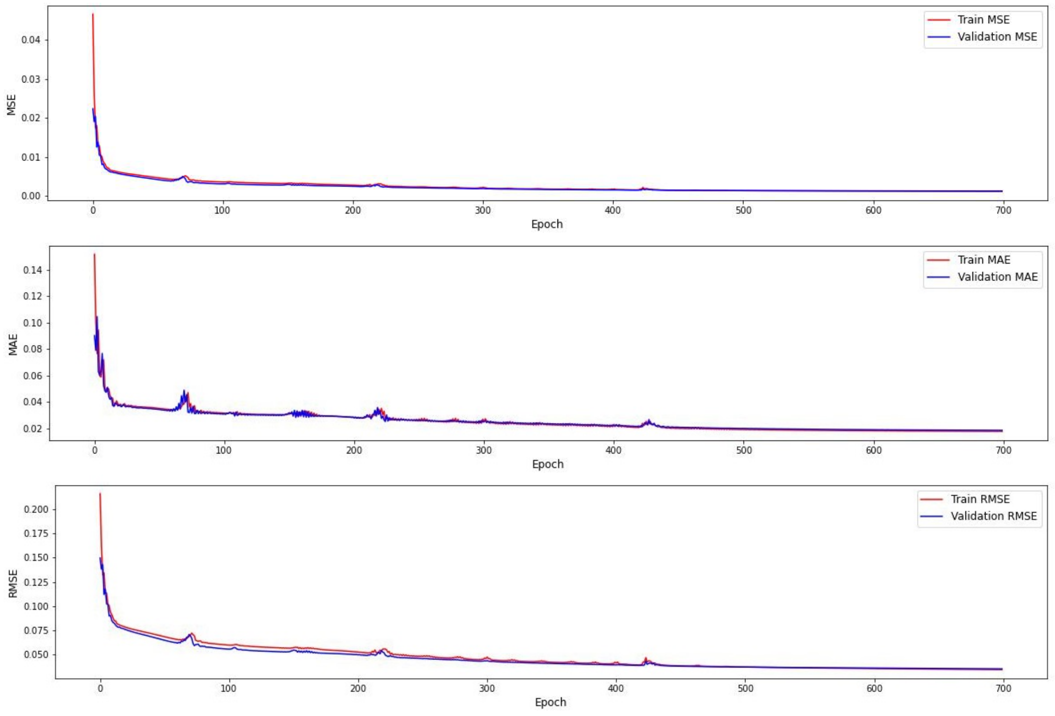

Figure 3.12: CPU metrics

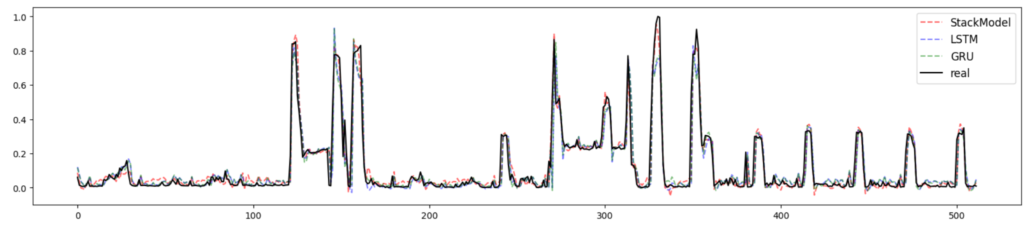

Figure 3.13: CPU usage prediction for VM number 890 validation dataset

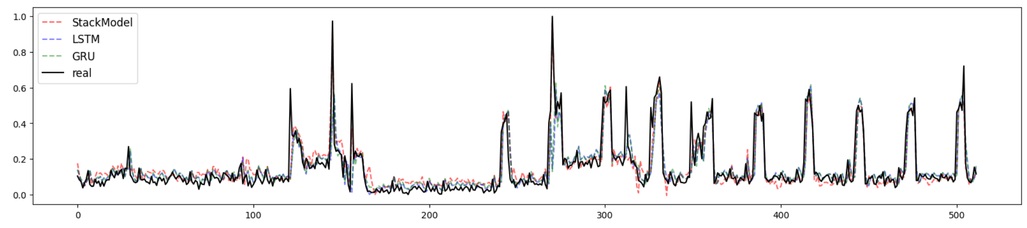

Figure 3.14: Memory usage prediction for VM number 890 validation dataset

demonstrated in Figure 3.13, the Stack model proposed in this study successfully forecasted values that closely align with the actual data at peak consumption points. Every

Figure 3.14 presents the memory usage prediction results for the validation data of VM 890. The figure clearly illustrates the strong correlation between the values predicted by the stacked model and the actual data.

Ablation Analysis

This section is written to compare the performance of various models, conduct an ablation study on the Stack model to determine the impact of its individual layers, and analyze the influence of input features. For this purpose, a dataset was constructed from 50 randomly chosen virtual machines from the Bitbrain dataset. Several models were trained on this data, including LSTM, GRU, ConvLSTM, ConvGRU, and three versions of the Stack model (with and without CNN and attention layers). A performance comparison was subsequently carried out on the validation set. Furthermore, Figure 3.15 visualizes the memory consumption patterns across these VMs.

As illustrated in Figure 3.15, the data from the selected virtual machines is heterogeneous. Accordingly, the models were evaluated based on their performance across these varied data states, specifically on both noisy and non-noisy data.

Figure 3.16 illustrates the distribution of CPU consumption data across the various VMs. This variety in the dataset allowed for a thorough evaluation of the model's performance in predicting CPU usage for diverse data structures.

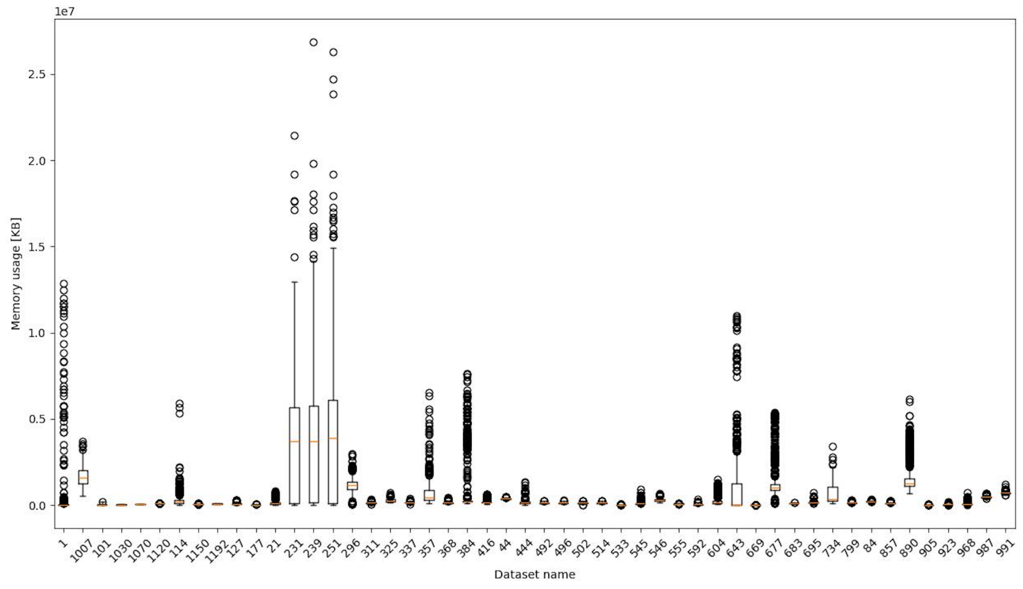

Figure 3.15: Memory consumption patterns across sample VMs

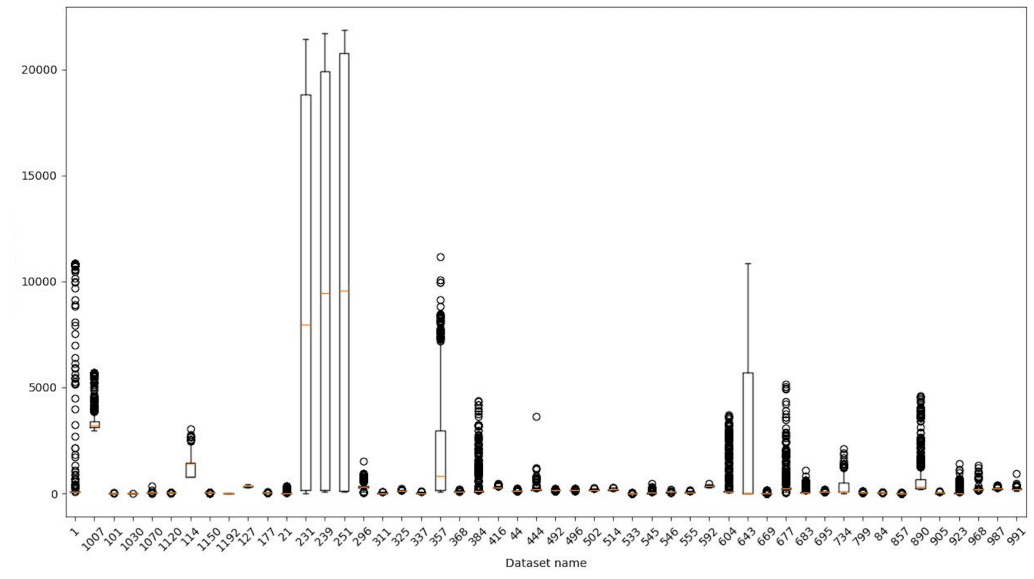

Figure 3.16: CPU consumption patterns across sample VMs

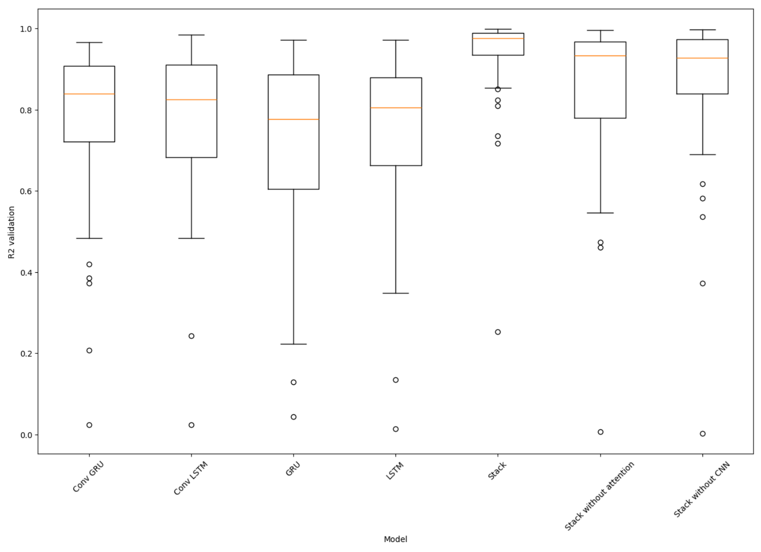

Figure 3.17: R^2 CPU Validation

To test the efficacy of our proposed model, its performance was quantitatively assessed by comparing it against several other models on a CPU prediction task, utilizing data from 50 virtual machines. The dataset was partitioned using the same 80/20 training-testing split strategy as before, with 80% allocated for model training and the remaining 20% reserved for testing on previously unseen data.

The outcomes of this comparison are depicted in Figure 3.17, which presents box plots of the R2 test parameter for each model. The R2 metric, fundamental to regression analysis, quantifies the proportion of variance in the dependent variable that is predictable from the independent variable(s).

Consequently, higher R2 values are indicative of a more accurate model. The box plots visualize the distribution of these scores, with each section corresponding to a quartile. The figure 3.17 clearly shows that the stack model outperforms the others, as it has both a higher minimum and a higher median R2 test score. This demonstrates more reliable and consistent performance. An analysis of its components reveals that the attention layer provides a performance boost (comparing stack to stack without Attention) and that the ConvLSTM layers are critical to its success. Furthermore, the benefit of a convolutional layer is evident when comparing the ConvLSTM model to the standard LSTM model, with the former showing improved results. The quantitative performance of each model on the hold-out test data is detailed in Table 3.12. For every model presented in this study, the table provides the resulting mean and variance for each metric. These statistics were derived from an evaluation conducted on 50 randomly selected virtual machines, using the terminal 20% of the time-series data which the models had not previously been exposed to. The data demonstrates the stack model's notable performance.

Table 3.12: Performance comparison of individual and stacked deep learning models (mean ± standard deviation).

Model	**MSE**	**MAE**	**RMSE**	**MAPE**	R^2
Stack	0.001 ± 0.0000	0.0161 ± 0.0001	0.0278 ± 0.0025	6.3361 ± 0.208	0.9362 ± 0.0135
Stack without CNN	0.0021 ± 0.0001	0.0247 ± 0.0002	0.0423 ± 0.0003	10.8925± 0.617	0.8637 ± 0.0329
Stack without attention	0.0021 ± 0.0000	0.0247 ± 0.0001	0.043 ± 0.0002	10.2961± 0.644	0.8445 ± 0.0378
Conv LSTM	0.0051 ± 0.0000	0.0393 ± 0.0006	0.0645 ± 0.0001	16.8638± 1.310	0.7811 ± 0.0357
Conv GRU	0.0055 ± 0.0000	0.041 ± 0.0005	0.0683 ± 0.0009	16.5897± 1.285	0.755 ± 0.0423
LSTM	0.0065 ± 0.0001	0.0433 ± 0.0007	0.0731 ± 0.0011	17.8771± 1.527	0.7369 ± 0.042
GRU	0.0069 ± 0.0001	0.0447 ± 0.0008	0.0755 ± 0.0012	18.0536± 1.479	0.7111 ± 0.0482

As shown in Table 3.12, the model achieved a variance of the MSE of 0.0135 and an average R^2 of 0.9362 on the test data, outperforming the other models. Similarly, its performance on the validation data, as demonstrated in Figure 3.18, indicates its high reliability, as it not only maintains lower MSE values but also produces fewer outliers.

In the models presented in this study, four categories of features were used to predict each target resource metric: CPU, memory, network, and disk usage. CPU prediction was selected as a representative task to analyze the importance of features. Since the proposed model is based on time-series forecasting, it utilizes data from all four feature categories over the 30 previous time steps to generate its predictions.

To assess the importance and impact of each feature category on model performance, an ablation study was conducted. Three additional models, identical in architecture, were trained, each excluding one of the feature categories. Figure 3.19 shows how removing each feature affects the accuracy of the CPU prediction, as measured by the Mean Squared Error (MSE). The results indicate that while all unrelated features contribute to impairing the model's performance, the presence of disk-related features leads to the most significant increase in error.

Table 3.13 illustrates the results obtained when each feature is removed individually. The results demonstrate that incorporating all available features significantly enhances the model's performance. To illustrate, excluding memory-related features from the prediction of CPU metrics results in a drop in the R2 score from 0.9362 to 0.9071. The table also highlights that, among all features tested, those

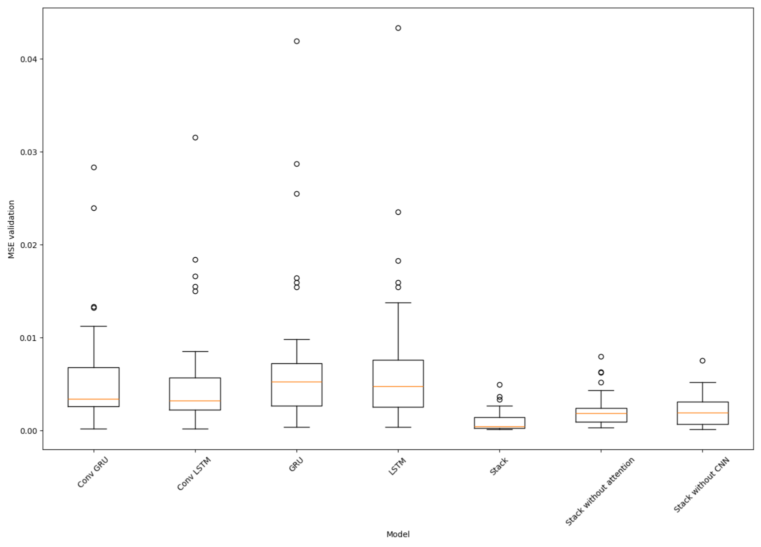

Figure 3.18: CPU MSE Validation

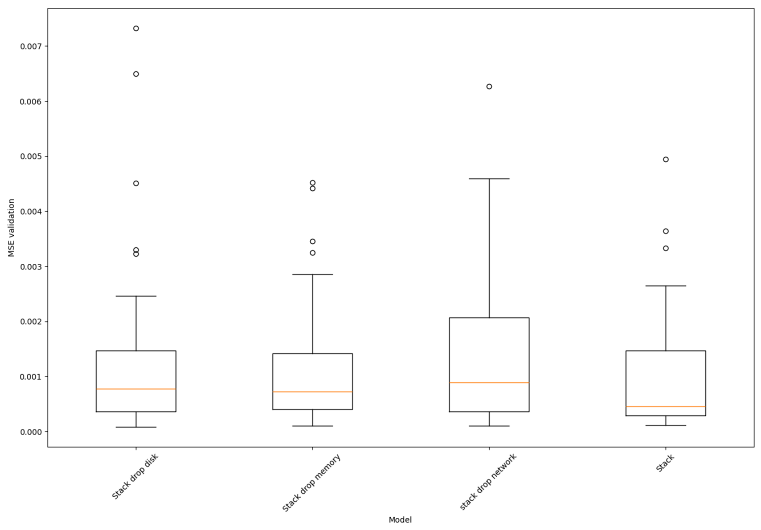

Figure 3.19: Feature importance

Table 3.13: Impact of removing individual feature categories on CPU prediction performance (mean ± standard deviation).

Model	**MSE**	**MAE**	**RMSE**	R^2
Stack	0.001 ± 0.0000	0.0161 ± 0.0001	0.0278 ± 0.0025	0.9362 ± 0.0135
Stack drop disk	0.0012 ± 0.0000	0.0182 ± 0.0001	0.313 ± 0.0003	0.8954 ± 0.0446
Stack drop memory	0.0011 ± 0.0000	0.0174 ± 0.0001	0.305 ± 0.0002	0.9071 ± 0.0381
Stack drop network	0.0014 ± 0.0000	0.0194 ± 0.0001	0.0335 ± 0.0003	0.8855 ± 0.0476

related to the network had the most significant influence on the model's accuracy when forecasting CPU consumption.

According to Table 3.14, which details memory prediction metrics for 50 virtual machines, the stack model outperforms both the ConvGRU and ConvLSTM models. While the standalone Attention model performs poorly, its integration with ConvLSTM substantially boosts performance. An ablation study comparing the metric values of the Stack Without CNN and Stack Without Attention models with those of the main stack model reveals the importance of combining the Convolutional and Attention layers.

Figure 3.20 illustrates the comparative performance of several models on a memory prediction task across 50 virtual machines. The results indicate that the Attention model, when used independently, achieves results comparable to those of the ConvGRU and ConvLSTM models. A high variance in its performance suggests a strong dependency on data characteristics; nevertheless, it exhibits a more favorable overall performance, evidenced by a higher median R^2 parameter on the test data. Furthermore, a stacked architecture that integrates Attention, Convolutional LSTM, and Convolutional GRU layers demonstrates markedly better performance, confirming that this combination of layers enhances predictive accuracy.

Figure 3.21 details the MSE results from models predicting memory usage on various datasets. This data suggests that omitting the CNN and Attention layers could degrade the efficacy of the stacked model, a hypothesis that is slated for statistical validation using a t-test. Furthermore, a comparison reveals that the stacked models demonstrate not only better performance but also greater consistency across datasets. This indicates a higher degree of reliability in the stacked model compared to the individual Attention, ConvLSTM, and ConvGRU architectures.

The t-test results for the mean outcomes of the different models are provided in Table 3.15 which evaluates the statistical significance of the differences in mean model outcomes, as measured by R^2. The null hypothesis (H0) assumed that the mean R^2 values were equal across all models, tested at a significance level of 0.05

Table 3.14: Performance comparison of stacked and individual deep learning models (mean ± standard deviation).

Model	**MSE**	**MAE**	**RMSE**	**MAPE**	R^2
Stack	0.0003 ± 0.0000	0.0089 ± 0.0001	0.0133 ± 0.0001	2.2537 ± 0.0230	0.9791 ± 0.0023
Stack without GRU	0.0010 ± 0.0000	0.0185 ± 0.0002	0.0271 ± 0.0003	3.20 ± 0.0919	0.9339 ± 0.0228
Stack without CNN	0.0033 ± 0.0000	0.0377 ± 0.0004	0.0528 ± 0.0005	7.0086 ± 0.3796	0.8541 ± 0.015
Stack without Attention	0.0028 ± 0.0000	0.0344 ± 0.0003	0.0489 ± 0.0004	5.8302 ± 0.2153	0.8592 ± 0.0205
Attention	0.0103 ± 0.0001	0.0662 ± 0.0011	0.0939 ± 0.0015	11.8932± 1.1870	0.5314 ± 0.1099
Conv LSTM	0.0087 ± 0.0000	0.0615 ± 0.001	0.0857 ± 0.0014	9.8699 ± 0.7359	0.644 ± 0.0617
Conv GRU	0.0088 ± 0.0000	0.063 ± 0.0008	0.0879 ± 0.0010	10.5631± 0.8893	0.6367 ± 0.0529

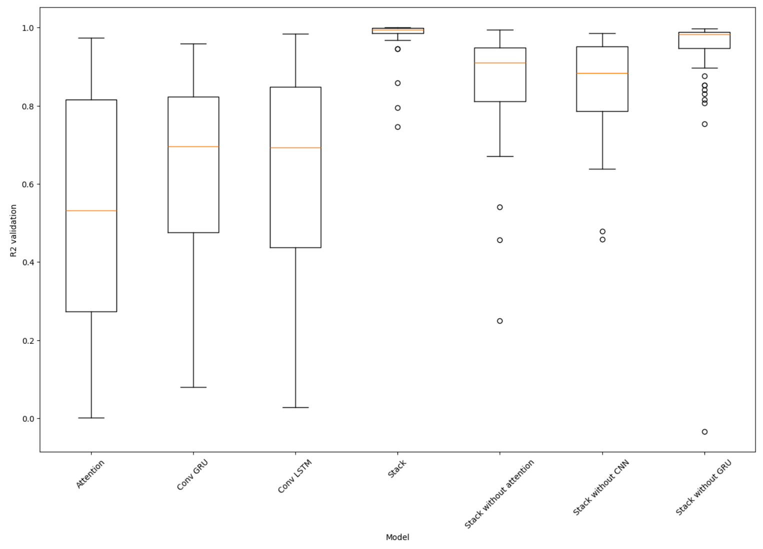

Figure 3.20: Memory R2 Validation

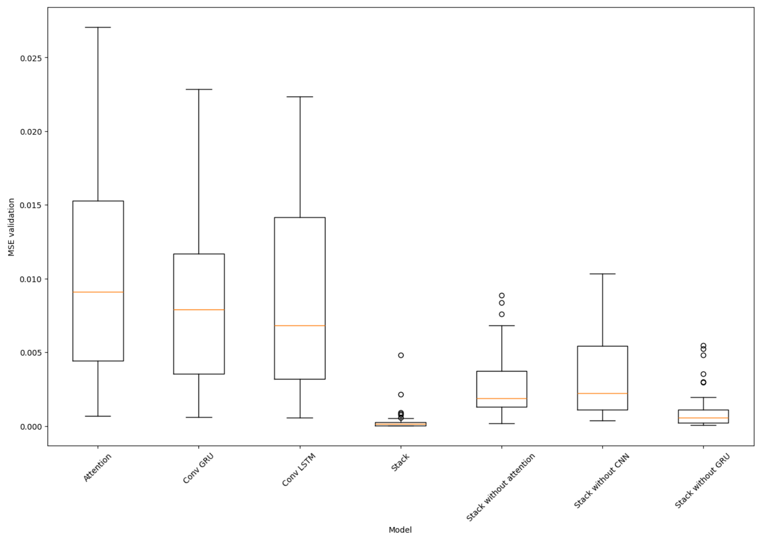

Figure 3.21: Memory MSE Validation

with 95% confidence. Rejection of H0 demonstrates that the observed performance differences are statistically significant. Conversely, if the null hypothesis is not rejected, it cannot be concluded that a statistically significant disparity in mean R^2 performance exists between the models, at a 95 percent confidence level.

The test results unequivocally demonstrate the strong performance of the stacked model, as indicated by its higher mean R^2 values compared to the standalone ConvLSTM, ConvGRU, and Attention models. Nevertheless, the primary objective of this analysis is to assess the statistical significance of each constituent layer—ConvLSTM, Attention, and GRU—within the integrated stacked architecture.

The t-test results in Table 3.15 indicate that hypothesis H0 is rejected with 95 percent confidence for the tests related to the Stack Without CNN, Stack Without GRU, and Stack Without Attention models, which reveals the importance of the CNN, GRU, and Attention layers. This means we can state with 95 percent confidence that there is a notable difference between the calculated R^2 value of the main stacked model and the R^2 values of the "Stack Without CNN", "Stack Without GRU", and "Stack Without Attention" models. Therefore, we can confidently conclude that the omission of either of these layers significantly degrades the model's predictive accuracy. Regarding the GRU layer, its contribution was found to be statistically significant at a 95% confidence level. Nevertheless, its impact on the model's performance is less than that of the other layers. Also, considering that the p-value of the t-test for the stack without the GRU model is 0.048, we cannot reject the null hypothesis H0—which states that the GRU layer has no effect—with 99% confidence. However, this hypothesis can be dismissed with 95% confidence. Visual analysis of Figure 3.20 indicates that omitting this layer can degrade the

Table 3.15: Statistical comparison between Stack and other models (mean values, p-values, and hypothesis rejection).

Model 1	**Model 2**	**Mean 1**	**Mean 2**	**P-value**	**Reject**
Stack	Stack without Attention	0.9573	0.8592	2.441×10^{-7}	TRUE
Stack	Stack without CNN	0.9573	0.8541	1.676×10^{-9}	TRUE
Stack	Stack without GRU	0.9573	0.9339	0.0488	TRUE
Stack	Attention	0.9573	0.5314	3.053×10^{-15}	TRUE
Stack	Conv LSTM	0.9573	0.6439	4.666×10^{-15}	TRUE
Stack	Conv GRU	0.9573	0.6367	4.444×10^{-17}	TRUE

R^2 performance on specific datasets. Consequently, the layer was retained in the stacked model to bolster its predictive accuracy.

The memory usage data from VM number 1192 is highly volatile, posing a significant challenge for prediction. Figure 3.22 presents the training and testing outcomes for the stack, ConvLSTM, and Attention models on this challenging dataset, where identifying peak consumption points is the primary objective.

An analysis of Figure 3.22 reveals the distinct performance characteristics of the models. The Attention model adeptly captures the general trend but lacks precision in predicting extreme values (maximums and minimums). The ConvLSTM model, although capable of modeling a broader range of volatility than the Attention model, exhibits deficiencies in trend identification in specific segments. The results for the stacked model suggest that the proposed integrated architecture overcomes these individual limitations. It demonstrates robust performance on datasets with significant fluctuations, successfully identifying both the overarching trend and instances of peak consumption.

For any well-architected cloud platform, forecasting all resource requirements is essential. However, the prediction of CPU and memory needs takes precedence over network and disk usage, as they represent the most common and direct performance constraints for the majority of applications. A deficit in CPU or memory severely and instantly impacts application functionality, resulting in degraded performance, errors, or system failure. Conversely, modern cloud infrastructure is typically engineered with greater elasticity for network and disk resources, making them less common points of critical failure. Consequently, this study omits a detailed performance analysis of the proposed model concerning network and disk utilization.

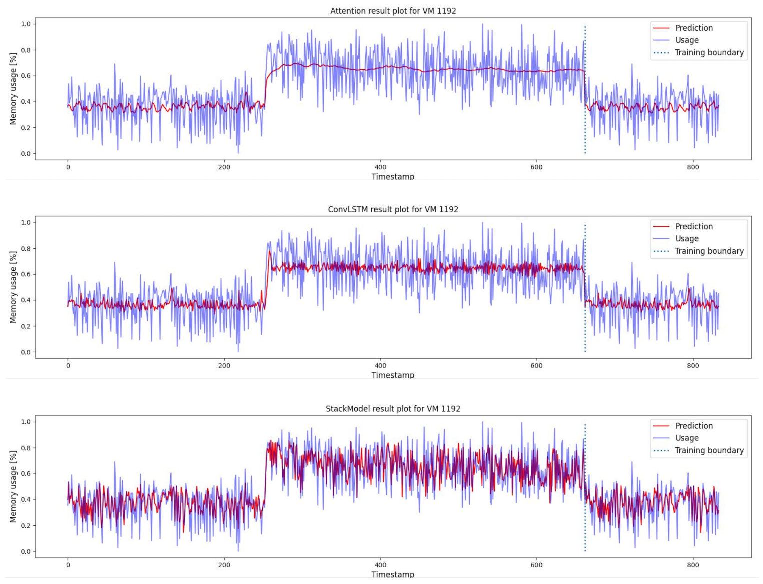

Figure 3.22: Memory MSE Validation

3.7.3 Comparison of Proposed Models with Related Studies

Table 3.16 compares the models discussed in the related work section with our model, focusing on their evaluation results for Mean Squared Error and R-squared when applied to the Bitbrains dataset. The proposed models, particularly Transformer and Deep Stack, demonstrate their robustness by achieving competitive MSE and R2 values. With an R2 of 0.9913, the Transformer model excels in explaining variance, indicating its strong predictive reliability. Similarly, the Deep Stack model achieves a low MSE of 0.0013, showcasing its ability to minimize error effectively, making it a powerful and practical choice for cloud resource prediction tasks on the Bitbrains dataset. These results highlight the advanced capabilities of the proposed models compared to traditional approaches.

3.7.4 Model Performance Evaluation

The model's performance is illustrated through accuracy, loss, precision, and recall plots, demonstrating its robust predictive capabilities as figure 3.23. Each graph displays the metric values across training epochs, with the x-axis representing epochs and the y-axis showing metric scores. The accuracy plot shows convergence for training and validation data, reaching approximately 87.5% and 87.3%, respectively, indicating effective generalization without overfitting. The loss plot shows a stable decrease in error, with minimal disparity between the training loss (0.4737) and the validation loss (0.4810), supporting the model's consistency and

Table 3.16: Summary of research on cloud predictive models

Refrences	**Prediction Approach**	**Implemented Models**	**Dataset**	**Time Intervals**	**Results**	**Best-performed Model**
[28]	Short-term	VAR, LSTM, Weighted gride search (VAR+LSTM)	Bitbrains Servers	Every 5 minutes	MSE=(0.3-0.8)	Weighted Ensemble (VAR+LSTM
[31]	Short-term	Hull-White SDE, Genetic Algorithm (GA), Support Vector Regression (SVR)	Bitbrains Servers	Every 5 minutes	MSE=0.5, Mape=0.071	Hull-White-GA
[27]	Short-term	Linear Regression, Decision Tree Regression, Gradient Boosting, Support Vector Regression, LSTM, BiLSTM	Bitbrains Servers	Every 5 minutes	RMSE=0.049, R^2=0.804	BiLSTM
[5]	Short-term/Long-term	RNN, LSTM, GRU, 1D-CNN, attention-based LSTM	Bitbrains Servers	Every 5 minutes	RMSE=0.1-0.2	GRU under transfer attacks
Proposed Stack Model	short-term	LSTM-CNN, GRU-CNN, GRU, Transformer,stack	Bitbrains Servers	Every 5 minutes	MSE=0.0013, RMSE=0253, MAE=0.0358, R^2=0.9786	stack of (LSTM-CNN, GRU-CNN, GRU, Transformer)

reliability. Precision, which measures the accuracy of positive predictions, stabilizes around 89% for both datasets, reflecting the model's ability to identify true positives with minimal false positives. Similarly, the recall plot, which measures the model's ability to capture all relevant positives, reaches approximately 86% across both training and validation sets, confirming its effectiveness in identifying actual positive instances without significant false negatives. The close alignment across all metrics between training and validation sets indicates that the model generalizes well, maintaining performance on unseen data. These results demonstrate that the model is both accurate and reliable, making it suitable for real-world applications where high precision and recall in identification are crucial.

3.7.5 Overfitting and Underfitting Analysis

Overfitting occurs when a model captures noise in the training data rather than generalizable patterns, resulting in a low training error but a high validation or test error. Conversely, underfitting happens when a model is too simplistic to learn meaningful trends, resulting in high error rates across both training and validation datasets [17]. To ensure that the proposed stack model achieves a balanced learning process without overfitting or underfitting, dropout regularization was employed. Dropout functions by randomly deactivating a fraction of neurons during training, preventing the model from relying excessively on specific features and encouraging generalization. This technique played a crucial role in ensuring that performance metrics, including accuracy, loss, precision, and recall, remained consistent between

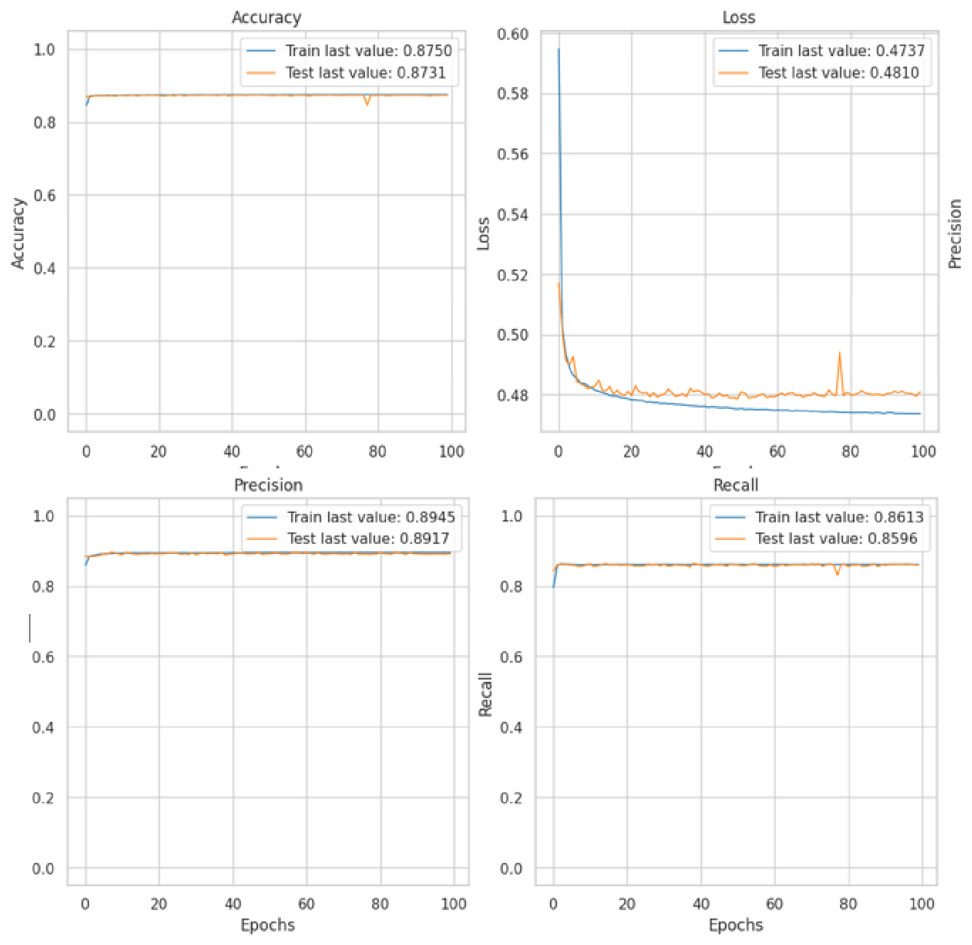

Figure 3.23: Evaluation Graphs

the training and validation datasets. The model's logarithmic error slope indicates that it neither overfits nor underfits the data, demonstrating a stable and generalized learning process. The close alignment of training and validation performance indicates that the model generalizes well, effectively capturing underlying patterns without memorizing noise. As a result, the model maintains robust predictive capabilities, making it suitable for real-world applications where high reliability and adaptability are essential.

Chapter 4

Conclusion and Future Work

4.1 Contributions and Implications

This thesis presents a method for predicting cloud usage metrics, including disk, CPU, memory, and network utilization, utilizing a deep-learning stack that uses LSTM, GRU, CNN, and Transformer models. Additionally, to verify the model's robustness and accuracy, statistical models were also deployed on the same dataset. The proposed model demonstrated notable improvements in forecasting accuracy, achieving more effective performance compared to statistical approaches, as well as several deep-learning models tested in prior studies and included in our baselines. Key metrics, including MSE, RMSE, and R2, validated its effectiveness and improved predictive accuracy over baseline models.

This study proposes a hybrid forecasting architecture for cloud resource consumption that integrates CNN, LSTM, GRU, and attention mechanisms. The model first extracts multi-scale features using parallel convolutional layers with distinct filter sizes. These features are then weighted by time-series LSTM layers, preceded by an attention layer, which enables the model to learn data trends and outperform benchmark models effectively.

A comprehensive performance evaluation was conducted on the Bitbrain dataset to compare our proposed model with various established architectures, such as LSTM, GRU, and convolutional LSTM. The comparative analysis revealed that the proposed model outperformed the other models. In particular, it excelled at accurately predicting peak resource consumption, which is a critical factor in building and maintaining a cost-effective, reliable, and high-performance cloud infrastructure. The proposed stacked model demonstrated strong performance on the Bitbrain dataset's CPU consumption task, achieving an R^2 score of 0.93. This indicates a 27 percent improvement on this metric compared to the standard Convolutional LSTM model.

This research also breaks down the architecture of the model presented to assess the importance of each component. We found that convolutional LSTM layers with different filter sizes provide the most effective performance gain, contributing significantly to the model's accuracy. In contrast, the attention layer offers only a minor improvement. In a separate feature analysis, we demonstrated that incorporating all available features significantly enhances the model's performance and identified the network-related metrics as the most substantial influence on predicting CPU usage.

Despite its contributions, this research has certain limitations. First, the evaluation relied solely on the Bitbrains dataset, which may not fully represent the diversity of cloud environments. Second, computational complexity remains a concern for real-time deployment, given the layered architecture of the stack model. Third, the comparison with related works was constrained by differences in evaluation metrics, limiting direct comparability.

4.2 Future Work

Future research should address these limitations by testing the model on additional datasets (e.g., Google and Microsoft Azure traces) and developing lightweight versions suitable for real-time applications. Enhancing the model's robustness to adversarial attacks and dynamic workloads is another critical direction for improvement. Furthermore, integrating the model into cloud-native systems could enable end-to-end automation of resource allocation. Ultimately, this research contributes to environmental sustainability by optimizing cloud resource utilization and minimizing operational waste, which minimizes idle server capacity, a major source of unnecessary energy consumption in data centers. Future work should quantify the model's ecological impact, providing concrete estimates of energy savings and reductions in Environmental impact. This research lays a foundation for more scalable, accurate, and sustainable solutions in cloud resource management by addressing these challenges.

Bibliography

[1] C. C. Aggarwal et al., *Neural Networks and Deep Learning*, Springer, 2018.

[2] O. Ali, A. Shrestha, J. Soar, and S. F. Wamba, *Cloud computing-enabled healthcare opportunities, issues, and applications: A systematic review*, International Journal of Information Management, 43 (2018), pp. 146–158.

[3] S. Arbat, V. K. Jayakumar, J. Lee, W. Wang, and I. K. Kim, *Wasserstein adversarial transformer for cloud workload prediction*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 12433–12439.

[4] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, arXiv preprint arXiv:1409.0473, (2014).

[5] L. Breiman, *Bagging predictors*, Machine Learning, 24 (1996), pp. 123–140.

[6] S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath, *An attentive survey of attention models*, ACM Transactions on Intelligent Systems and Technology (TIST), 12 (2021), pp. 1–32.

[7] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, *On the properties of neural machine translation: Encoder-decoder approaches*, arXiv preprint arXiv:1409.1259, (2014).

[8] G. Dhamane, *The Grid Workloads Archive-T-12 Bitbrains*. kaggle, accessed Dec 2025.

[9] T. G. Dietterich et al., *Ensemble learning*, The handbook of brain theory and neural networks, 2 (2002), pp. 110–125.

[10] J. Durbin and S. J. Koopman, *Time series analysis by state space methods*, OUP, 2nd ed., 2012.

[11] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, *Scalable object detection using deep neural networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 2147–2154.

[12] Y. Freund and R. E. Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting*, Journal of Computer and System Sciences, 55 (1997), pp. 119–139.

[13] A. Galassi, M. Lippi, and P. Torroni, *Attention in natural language processing*, IEEE transactions on neural networks and learning systems, 32 (2020), pp. 4291–4308.

[14] S. Goodarzy, M. Nazari, R. Han, E. Keller, and E. Rozner, *Resource management in cloud computing using machine learning: A survey*, in 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE, 2020, pp. 811–816.

[15] S. C. Gupta, A. B. Kunnumakkara, S. Aggarwal, and B. B. Aggarwal, *Inflammation, a double-edge sword for cancer and other age-related diseases*, Frontiers in immunology, 9 (2018), p. 2160.

[16] M. E. Karim, M. M. S. Maswood, S. Das, and A. G. Alharbi, *BHyPreC: A novel bi-LSTM based hybrid recurrent neural network model to predict the CPU workload of cloud virtual machine*, IEEE Access, 9 (2021), pp. 131476–131495.

[17] T. Khan, W. Tian, G. Zhou, S. Ilager, M. Gong, and R. Buyya, *Machine learning (ML)-centric resource management in cloud computing: A review and future directions*, Journal of Network and Computer Applications, 204 (2022), p. 103405.

[18] A. Khoshkroodi, H. Parvini Sani, and M. Aajami, *Stacking ensemble-based machine learning model for predicting deterioration components of steel W-section beams*, Buildings, 14 (2024), p. 240.

[19] D. F. Kirchoff, M. Xavier, J. Mastella, and C. A. De Rose, *A preliminary study of machine learning workload prediction techniques for cloud applications*, in 2019 27th Euromicro international conference on parallel, Distributed and Network-Based Processing (PDP), IEEE, 2019, pp. 222–227.

[20] J. Kolluri, V. K. Kotte, M. Phridviraj, and S. Razia, *Reducing overfitting problem in machine learning using novel L1/4 regularization method*, in 4th international conference on trends in electronics and informatics (ICOEI) (48184), IEEE, 2020, pp. 934–938.

[21] H. L. Leka, Z. Fengli, A. T. Kenea, A. T. Tegene, P. Atandoh, and N. W. Hundera, *A hybrid CNN-LSTM model for virtual machine workload forecasting in cloud data center*, in 2021 18th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), IEEE, 2021, pp. 474–478.

[22] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, *Statistical and machine learning forecasting methods: Concerns and ways forward*, PLOS ONE, 13 (2018), p. e0194889.

[23] Y. S. Patel and J. Bedi, *Mag-d: A multivariate attention network based approach for cloud workload forecasting*, Future Generation Computer Systems, 142 (2023), pp. 376–392.

[24] M. Pleños, *Time series forecasting using Holt-Winters exponential smoothing: application to abaca fiber data*, Zeszyty Naukowe SGGW w Warszawie-Problemy Rolnictwa Światowego, 22 (2022), pp. 17–29.

[25] H. Sak, A. W. Senior, and F. Beaufays, *Long short-term memory recurrent neural network architectures for large scale acoustic modeling*, in Proceedings Interspeech 2014, Singapore, Sep 2014, pp. 338–342.

[26] R. Selukar, *State space modeling using SAS*, Journal of Statistical Software, 41 (2011), pp. 1–13.

[27] A. Shaikh, M. Patel, and R. Khan, *Predicting CPU and network utilization in cloud environments using deep learning models*, Journal of Cloud Computing and Big Data, 12 (2024), pp. 127–144.

[28] J. Shetty, K. Cottur, G. Shobha, and Y. R. Prajwal, *A weighted ensemble of VAR and LSTM for multivariate forecasting of cloud resource usage*, Journal of Advances in Information Technology, 14 (2023), pp. 264–270.

[29] J. Shi, L. Wang, Z. Dai, L. Zhao, M. Du, H. Li, and Y. Fang, *Multiscale hierarchical design of a flexible piezoresistive pressure sensor with high sensitivity and wide linearity range*, Small, 14 (2018), p. 1800819.

[30] R. Sikora et al., *A modified stacking ensemble machine learning algorithm using genetic algorithms*, in Handbook of research on organizational transformations through big data analytics, IGi Global, 2015, pp. 43–53.

[31] C. St-Onge, S. Benmakrelouf, N. Kara, H. Tout, C. Edstrom, and R. Rabipour, *Generic SDE and GA-based workload modeling for cloud systems*, Journal of Cloud Computing: Advances, Systems and Applications, 10 (2021).

[32] Y. Tao, B. Yan, D. Fan, N. Zhang, S. Ma, L. Wang, Y. Wu, M. Wang, J. Zhao, and H. Zhang, *Structural changes of starch subjected to microwave heating: A review from the perspective of dielectric properties*, Trends in Food Science & Technology, 99 (2020), pp. 593–607.

[33] R. S. Tsay, *Analysis of Financial Time Series*, John Wiley & Sons, 2005.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, Advances in neural information processing systems, 30 (2017).

[35] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao, *Learning deep transformer models for machine translation*, arXiv preprint arXiv:1906.01787, (2019).

[36] C. Wolf and E. M. Halter, *Virtualization: From the Desktop to the Enterprise*, Apress, 2006.

[37] D. H. Wolpert, *Stacked generalization*, Neural Networks, 5 (1992), pp. 241–259.

[38] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, *Show, attend and tell: Neural image caption generation with visual attention*, in International conference on machine learning, PMLR, 2015, pp. 2048–2057.

[39] A. K. Y. Yanamala, *Emerging challenges in cloud computing security: A comprehensive review*, International Journal of Advanced Engineering Technologies and Innovations, 1 (2024), pp. 448–479.

[40] H. Zhang, J. Li, and H. Yang, *Cloud computing load prediction method based on CNN-BiLSTM model under low-carbon background*, Scientific Reports, 14 (2024), p. 18004.

Appendix A

Concepts and Fundamentals

A.1 Foundations of Predictive Modeling in Cloud Environments

Given the dynamic and complex nature of cloud environments, predictive modeling of cloud resource usage necessitates a robust understanding of both statistical and machine learning approaches. This section outlines the foundational concepts underpinning our research, detailing their relevance and evolution in the development of our forecasting model.

A.1.1 Statistical Models in Time Series Forecasting

Statistical methods have long dominated the field of time series forecasting, offering a diverse array of models, each designed to analyze specific data patterns [24]. These models have been foundational in numerous applications, from economic forecasting to energy consumption modeling. In the context of cloud resource management, statistical models provide essential benchmarks against which modern approaches can be evaluated. The following section explores two statistical models: Holt-Winters Exponential Smoothing and the Unobserved Component Model (UCM), examining their mechanics and capabilities. These models set the stage for a comparative analysis with our proposed deep-learning model, highlighting the evolution and necessity of more advanced predictive technologies in managing the dynamic nature of cloud computing environments.

Holt-Winters Exponential Smoothing:

Holt-Winters Exponential Smoothing is one of the oldest time series analysis and statistical methods that has been in use for a long time. It is named after two people who contributed to its development, Charles Holt and Peter Winter. This technique considers both the trend and seasonality when making predictions. The predictions made using Holt-Winters Exponential Smoothing are based on three components of a seasonal time series: level, trend, and seasonal coefficient. Exponential smoothing is a technique used to smooth a time series of data by assigning weights that decrease exponentially over time. This means that old data is given less importance by assigning lower weights to it, while more weight is given to recent data [24]. This method employs two smoothing equations, one for the level and one for the trend, in addition to a forecast equation [18]. The Holt-Winters Exponential Smoothing model integrates three key components to forecast seasonal time series data effectively. The Level component represents the baseline value of the series, indicating the average outcome within a fixed period. The Trend component accounts for the series' increasing or decreasing values over time. Lastly, the Seasonal Coefficient captures the repeating short-term cycles within the series, which is essential for adjusting predictions to seasonal fluctuations [24].

$$\begin{array}{ll} \text{Forecast equation:} & \widehat{y}_{(t+h|t)} = l_t + hb_t \\ \text{Level equation:} & l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1})) \\ \text{Trend equation:} & b_t = \beta^* \ (l_t - l_{t-1}) + \left(1-\beta^*\right) b_{t-1} \end{array} \tag{A.1}$$

where $\hat{y}_{(t+h|t)}$ represents the predicted value of the series at time $t+h$, given the information available up to time t. The notation $\hat{y}$ indicates that this value is an estimate or forecast, distinguishing it from observed or actual values of y. This differentiation is crucial for understanding the model's outputs in relation to real data, emphasizing the predictive nature of the model.

l_t represents the estimated level of the series at time t; b_t represents the estimated trend of the series at time t, α and β are the smoothing parameters for the level and trend, respectively, at $0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$.

The Holt-Winters model can capture trend and seasonality in time series data, making it helpful in forecasting in certain situations; it can handle additive and multiplicative seasonality for diverse time series, but despite its widespread use, its linear approach can be limiting when dealing with the inherently non-linear patterns of cloud resource usage, necessitating more complex models [14].

UCM Model

State space modeling is a powerful approach for analyzing time series data. It assumes that unobserved variables influence the system's behavior over time. These variables are related to the observed data through a state space model. The goal of state space analysis is to estimate the characteristics of the unobserved variables based on the observed data. A specific type of state space model commonly used in time series research is the unobserved component model. The system under study is divided into several unobserved components in the UCM, each of which is represented by a state variable, including trend, seasonality, and error term. A measurement equation is then used to connect the observations to these unobserved components [10].

$$\begin{aligned} y_t &= \mu_t + \gamma_t + \psi_t + \sum_{j=1}^{m} \beta_j x_{jt} + \varepsilon_t \\ \epsilon_t &\sim i.i.d.N\left(0, \sigma_\varepsilon^2\right) \end{aligned} \tag{A.2}$$

The components μ_t, γ_t, and ψ_t respectively represent the trend, seasonal, and cyclical components of the model; the term $\sum_{j=1}^{m} \beta_j x_{jt}$ gives the contribution of regression variables with fixed or time-varying regression coefficients [26].

The Unobserved Component Model, also known as the Structural Time Series Model, is a popular approach for analyzing patterns in time-based data by breaking it down into distinct components, including trends, seasonality, and cycles. However, it has some limitations. It assumes that the trend changes gradually over time, which might not always be true. While the model assumes that the trend changes slowly over time, it can still help in detecting shifts in trends. The model's handling of seasonal changes may not always accurately reflect real-world patterns, which can impact its predictions. Similarly, it may not accurately capture complex cycles. Therefore, while this model is beneficial for time series analysis, we must be cautious of its limitations and consider alternative models or adjustments as needed [33].

A.1.2 Emergence of Machine Learning in Predictive Analytics

Traditional statistical models face several limitations, including their inability to capture complex and nonlinear data relationships, making them less effective in rapidly changing environments. They rely on assumptions like stationarity and normality, which, if violated, can lead to inaccuracies. These models struggle with high-dimensional data and are prone to overfitting, especially when handling complex or non-repetitive patterns. While computationally efficient for simple tasks, their performance degrades with complexity, and they generally excel in short-term forecasts but often falter over longer horizons. These drawbacks have driven the shift towards more flexible and adaptive machine learning and deep learning methods [47]. Machine learning has emerged as a powerful successor, offering greater flexibility and capability in handling diverse data types and patterns. Deep learning, a branch within the broader field of machine learning, emphasizes feature-based learning. It consists of approaches that empower a system to independently acquire the critical features needed for task detection derived from training data [11].

Artificial Neural Networks (ANN)

At the heart of deep learning lies the artificial neural network (ANN), a model inspired by the neural architecture of the human brain. It is a complex communication network composed of smaller entities known as neurons. These neurons possess the unique capability to retain knowledge gained from experience and apply it in various contexts. An ANN consists of several computational layers arranged in a sequence. Each subsequent layer is responsible for learning more abstract features. As the processing progresses into deeper layers, these layers identify and extract more complex and abstract features at higher levels. There are two primary forms of neural networks:

Feedforward Neural Networks (F-F) In this type, data flows linearly from the input layer through various hidden layers and finally to the output layer; there is no cycling or looping back of information, which characterizes these networks as direct acyclic. The formula is described as follows:

$$f(x) = \sigma\left(w^t x + b\right) \tag{A.3}$$

where w represents the weight vector, b is the bias, σ is the activation function, and x is the input vector to the neuron.

Recurrent Neural Networks (RNNs) Unlike feedforward networks, RNNs incorporate loops into their architecture, enabling them to retain a state from one iteration to the next. This feature is crucial for tasks that depend on sequential information, such as time series analysis or natural language processing. In the RNNs formula mentioned below, each neuron's output at a given time t, denoted by h_t, is determined by the current input x_t and the output from the previous time step, h_{t-1}. This feedback mechanism allows the network to maintain a form of memory that influences future outputs, effectively incorporating previous states into current decisions.

$$h_t = \sigma(W.[h_{t-1}, x_t] + b) \tag{A.4}$$

Deep-Learning Models

LSTM Model Short-Term Memory Networks (LSTMs) are an advanced type of Recurrent Neural Networks (RNNs) designed to learn long-term dependencies. Unlike basic RNNs, which typically consist of a single neural layer, LSTMs feature a complex architecture with four interacting layers. Central to these layers are unique structures known as memory blocks. Each memory block contains memory cells with self-connections, which preserve the cell state over time, thereby maintaining the network's memory. Additionally, LSTMs utilize specialized units called gates, which regulate the flow of information into and out of the cells. These gates ensure that the network retains important information and discards non-essential data, enabling it to make accurate predictions over extended periods [25].

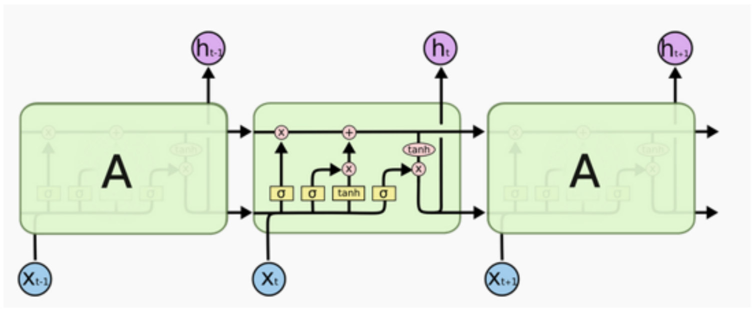

Figure A.1: LSTM network module

The most prominent feature of LSTM is its ability to learn long-term dependencies, which is impossible with Recurrent Neural Networks. To predict the next time step, the network must update its weight values, which requires retaining information from the initial time steps. A Recurrent Neural Network can only learn a limited number of short-term dependencies; however, long-term time series like LSTMs can accurately learn these long-term dependencies.

The cell state, which is the core component of LSTMs, is represented as a horizontal line running through the top of the diagram in Figure A.1. The cell state can be envisioned as a conveyor belt running from the beginning to the end of the sequence or chain, with minor linear interactions in motion (meaning its structure is very simple and undergoes minor changes [32]. The LSTM network can add new information to the cell state or delete its information. This is accomplished by precise structures called gates [32]. Gates are a way for the optional entry of information. They are composed of a layer of a sigmoid neural network along with a point-to-point multiplication operator [29]. The output of the sigmoid layer is a number between zero and one, indicating how much of the input should be sent to the output. A value of zero means that no information should be sent to the output, while a value of one means all inputs should be sent to the output.

Forget gate output The initial phase as shown in Figure A.2, Long Short-Term Memory (LSTM) networks involves determining what information should be discarded from the cell state. This critical decision is executed by a component known as the "forget gate," which is a sigmoid layer. This layer assesses the data and decides the extent to which each element of the cell state should be retained or removed as the network processes data. This gate, considering the values of h_{t-1} and x_t, outputs a zero or one for each number in the cell state c_{t-1}. A value of one means completely carrying over the current value of the cell state c_{t-1} to c_t, and a value of zero means completely erasing the information of the current cell state c_{t-1}

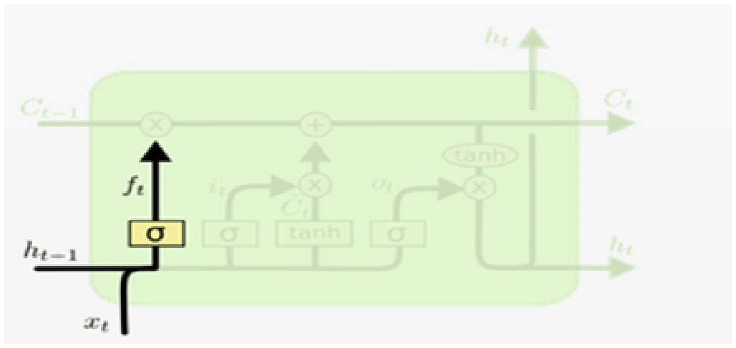

Figure A.2: LSTM forget gate

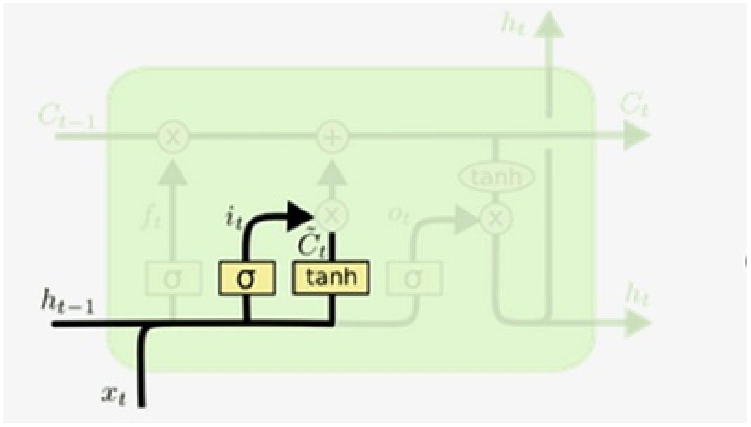

Figure A.3: LSTM input gate

and not transferring any of it to c_t.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{A.5}$$

Input gate The next step is to decide what new information we want to store in the cell state. This decision is twofold. First, we have a sigmoid layer called the input gate that decides which values will be updated. The next step is a hyperbolic tangent layer that creates a vector of new candidate values, denoted a $\hat{C}_t$, which could be added to the cell state. In the next step, we combine these two stages to update the cell state value as in A.3.

$$\begin{aligned} i_t &= \sigma\left(W_i \cdot \left[h_{(t-1)}, x_t\right] + b_i\right) \\ \hat{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \tag{A.6}$$

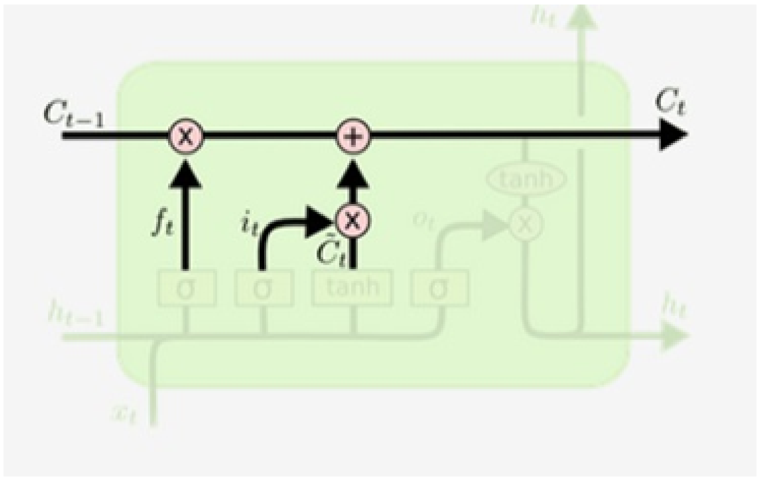

Figure A.4: LSTM reset gate

Reset gate Now as illustrated in A.4, the time has come to update the old cell state C_{t-1} to the new cell state C_t. The previous steps involved making decisions, and now we need to implement those decisions. We multiply the old cell state value by $f_{t,}$, meaning we forget the information we decided to forget earlier. Then, we add $i_t * \hat{C}_t$ to it. Now, the new cell state values are determined based on the decisions made earlier.

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{A.7}$$

Output gate In the final stage of the Long Short-Term Memory (LSTM) processre, Figure A.5, the crucial decision is what information to send to the output. This output is derived from the cell state, but it doesn't go directly to the output. Instead, it is filtered through a particular mechanism, ensuring that only the most relevant and significant information from the cell state is passed on as output. This selective filtering is key to the effectiveness of LSTM networks in processing and retaining important information over time. Initially, we have a sigmoid layer that decides which parts of the cell state should be sent to the output. Then, we pass the cell state value (updated in the previous steps) to a hyperbolic tangent layer (so the values are between -1 and +1) and multiply it by the output of the previous sigmoid layer to ensure only the intended parts are sent to the output.

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \tag{A.8}$$

In other words, the candidate cell state $\hat{C}_t$ is calculated using the input data x_t and the previous hidden state h_{t-1}. The cell memory, or the current cell state c_t, is

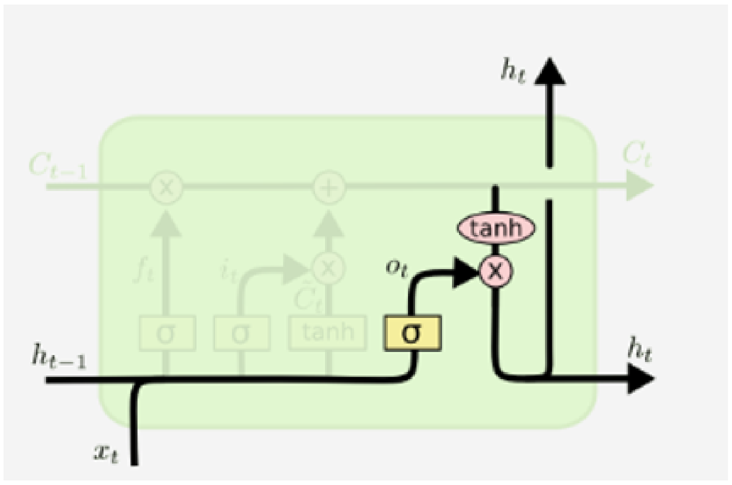

Figure A.5: LSTM output gate

calculated using the forget gate f_t, the previous cell state c_{t-1}, its input gate, and the candidate cell state C_t.

GRU Model Cho et al. introduced Gated Recurrent Units (GRUs) in 2014 [4]. They were designed to solve key problems encountered in traditional RNNs while simplifying the model complexity found in LSTMs. GRUs are considered a variant of the LSTM as they provide similar functionality but with a more efficient architecture, often achieving comparable performance in tasks involving sequential data [15].

GRU is a streamlined version of Long Short-Term Memory (LSTM) networks, employing fewer parameters through only two gates: the update gate and the reset gate. The update gate regulates the rate at which the hidden state is updated, enabling the network to determine how much of the past information to retain. Conversely, the reset gate determines how much of past information to discard, which is critical in modeling sequences where the network needs to forget irrelevant parts of the past data. The special thing about these gates is that they can be trained to retain information from very previous time steps without changing over time [1].

For a better understanding of this topic, the architecture of a recurrent neural network is shown in Figure A.6.

$$\begin{aligned} h_t &= \tanh(W_1\, X_t + W_2\, h_{t-1} + b_h) \\ o_t &= \text{softmax}(W_3\, h_t + b_o) \end{aligned} \tag{A.9}$$

The diagram depicted in Figure A.7 provides a visual representation of the GRU cell.

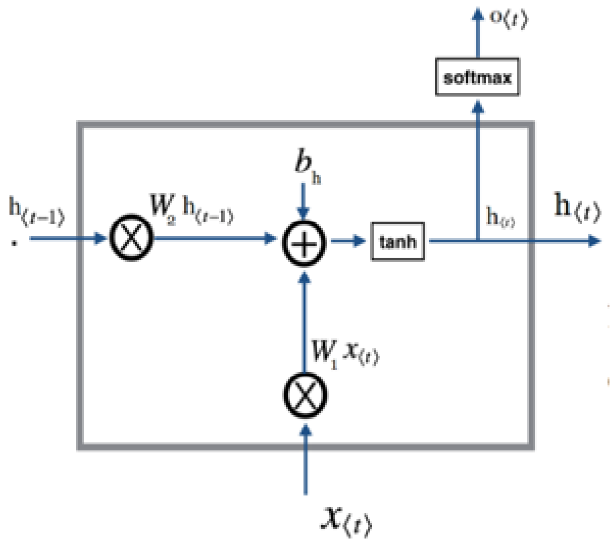

Figure A.6: Architecture of GRU network

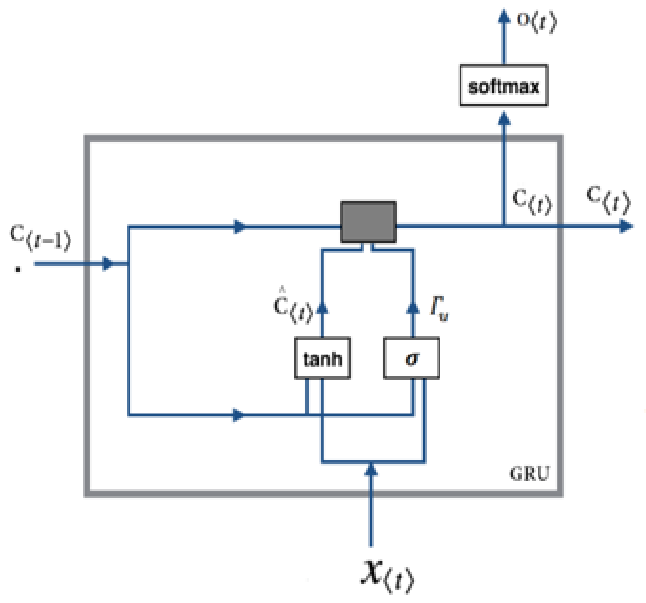

Figure A.7: A Cell in GRU

$$\begin{aligned}
\hat{C}_t &= \tanh\left(W_1.\left[C_{t-1}, X_t\right] + b_c\right) \\
\Gamma_u &= \sigma(W_2.\left[C_{t-1}, X_t\right] + b_u) \\
C_t &= \Gamma_u.\hat{C}_t + (1 - \Gamma_u)\,.C_{t-1} \\
o_t &= \text{softmax}(W_3\, C_t \,+\, b_o)
\end{aligned} \tag{A.10}$$

Figure A.7 illustrates that the operation in recurrent neural networks and GRUs is essentially the same; however, there are also two additional operations, referred to as the update gate and the reset gate, which exist in GRUs but not in traditional recurrent neural networks [1].

Update Gate The update gate operates like a switch, using a weighted combination of the previous state and the new candidate state, determined by the update gate's output.

$$C_t = \Gamma_u.\hat{C}_t + (1 - \Gamma_u)\,.C_{t-1} \tag{A.11}$$

The formula provided illustrates this process. Here, Γu is the update gate's output, which ranges from 0 to 1. When Γu is close to 0, the network tends to retain more of the previous state C_{t-1} ; when it is close to 1, it favors incorporating more of the new candidate's state $\hat{C}_t$, which is calculated based on the current input and the previous state. This adaptability enables the GRU to retain essential historical information while incorporating new data, thereby effectively managing the sequence's context over extended periods. This mechanism helps GRUs overcome the problem of vanishing gradients that plague traditional RNNs, thus improving their capacity to learn from data with long-term dependencies [1].

Transformer Model The Transformer model represents a significant advancement in deep learning by introducing a multi-head attention mechanism, which fundamentally differs from the traditional recurrent units found in models like the Long Short-Term Memory (LSTM). Unlike recurrent models that process data sequentially and depend on the internal states generated at each timestep, the Transformer processes entire sequences in parallel. This parallel processing capability significantly reduces training times and enables the model to handle more extended sequences more effectively, addressing the limitations imposed by memory constraints in earlier architectures [9].

Attention Mechanisem At the core of the Transformer's effectiveness is its sophisticated attention mechanism, initially developed for machine translation tasks [35]. This model has since evolved into a prevalent concept in the neural network literature. In the Artificial Intelligence (AI) community, attention has been recognized as an integral component of neural network architecture, with applications in Natural Language Processing (NLP), Speech, and Computer Vision (CV) [13].

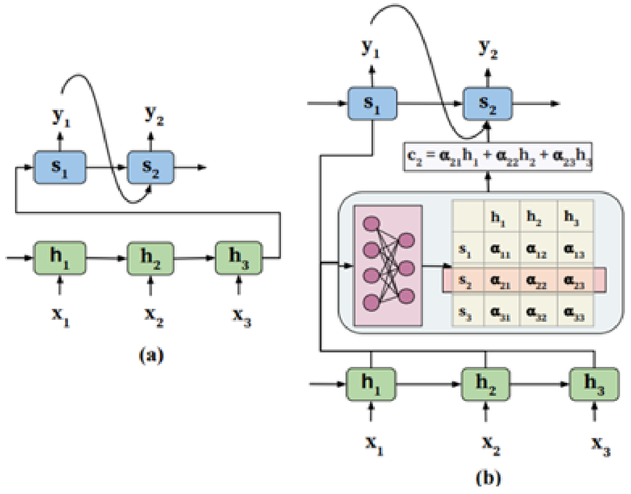
y_1
y_2
s_1
s_2
h_1
h_2
h_3
x_1
x_2
x_3
(a)
$c_2 = \alpha_{21}h_1 + \alpha_{22}h_2 + \alpha_{23}h_3$
α_{11} α_{12} α_{13}
α_{21} α_{22} α_{23}
α_{31} α_{32} α_{33}
s_3
(b)

Figure A.8: Encoder-Decoder architecture: (a) traditional (b) with attention model [6]

The concept of attention in neural networks is best understood by drawing parallels with human biological systems, particularly our visual processing. Just like how our eyes and brain selectively concentrate on specific portions of an image while disregarding less relevant parts, the attention mechanism in neural networks operates similarly. This selective focus helps in better comprehension and processing of the information at hand, enabling the network to handle and interpret complex data inputs efficiently [38]. Top of Form

The Attention Mechanism (AM) was first introduced in 2014 by Cho et al. [4] as a solution for sequence-to-sequence modeling tasks. A sequence-to-sequence model consists of an encoder-decoder architecture, as illustrated in Figure A.8; the encoder is an RNN that receives an input sequence of tokens $\{x_1,x_2,...,x_t\}$ and encodes them into vectors. The decoder is also an RNN, subsequently receiving a fixed-length vector h_t as its input and generating an output sequence. $\{y_1,y_2,...y_{t'}\}$ where t' is the length of the output sequence. At every position, t, h_t, and s_t, respectively, represent the hidden states of the encoder and the decoder [7].

The paper titled "Attention Is All You Need," published in 2017, introduced an encoder-decoder architecture based on attention layers, which the authors called the Transformer. One of the primary differences in this architecture is that it can process the input sequence in parallel, enabling the effective use of the GPU and consequently increasing training speed. Additionally, this architecture utilizes multi-headed attention, which allows it to overcome the vanishing gradient problem with ease. For instance, in a translator consisting of a simple RNN, the sequence or sentence is input continuously, with one word entered at a time to generate

-

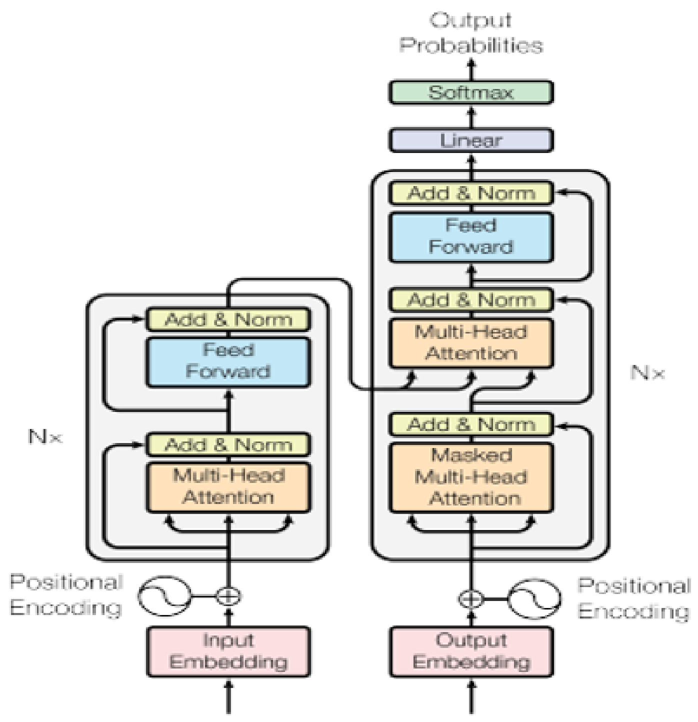

Figure A.9: Encoder-Decoder

the word embedding. Since each word depends on the previous work, its hidden state operates accordingly; therefore, it must be fed in one step at a time. In a Transformer model, every word in a sentence is processed simultaneously, allowing for the concurrent determination of word embeddings. This is achieved through a mechanism that assigns different levels of importance to each word in relation to others in the sentence, enabling the model to understand the context and meaning of each word in parallel.

The general schema of a Transformer is provided in Figure A.9.

Encoder-Decoder In the Transformer model, each encoder and decoder block features a series of layers designed to process information efficiently as presentes in Figure A.9. The encoder consists of multiple layers, including a multi-head self-attention mechanism and a fully connected feed-forward network, both of which are supplemented by residual connections and layer normalization. This structure facilitates the parallel processing of sequences, thereby enhancing the model's ability to handle complex patterns in data.

Similarly, the decoder includes layers that not only mirror the encoder's structure but also introduce an additional attention mechanism that focuses on the encoder's outputs. This enables the decoder to generate accurate and contextually relevant outputs effectively based on the encoded information.

Such architectural choices enable the Transformer to excel in various tasks across natural language processing, speech recognition, and even computer vision, leveraging its unique ability to simultaneously focus on different parts of the input data.

Scaled Dot-Product Attention: This mechanism computes the attention scores by scaling the dot product of queries Q and keys K by the inverse square root of the dimensionality of the keys $\sqrt{dk}$ and applying a softmax to prioritize certain values V. The formula is given by:

$$\text{Attention}\left(Q,\ K,\ V\right) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{A.12}$$

Multi-Head Attention: To enhance the model's ability to focus on different positions, the Transformer employs multi-head attention. This method projects the queries, keys, and values multiple times with different learned linear projections. This results in multiple attention heads, which are then concatenated and linearly transformed to produce the final values. The process can be summarized as:

$$\begin{aligned} &\text{MultiHead}\left(Q,\ K,\ V\right) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right)W^O \\ &\text{where: head}_i = \text{Attention}\left(QW_i^Q,\ KW_i^K, VW_i^v\right) \end{aligned} \tag{A.13}$$

Queries Q are vectors that query corresponding key vectors to produce attention scores. Keys K are vectors compared against queries to determine relevance. Values V are vectors containing the information to be aggregated based on attention scores. Projection matrices $W_i^Q,\ W_i^K, W_i^v$ are learned matrices that project queries, keys, and values into different subspaces for each attention head. Each attention head $head_i$ outputs the attention scores and weighted values for its specific subspace. The final projection matrix W^O projects the concatenated outputs of all attention heads back to the original dimensionality. This mechanism allows the Transformer to process multiple attention distributions, capturing various contextual relationships and dependencies more effectively.

In conclusion, the Transformer architecture not only accelerates training times due to its ability to process data in parallel but also significantly improves performance on tasks that require understanding of long-range dependencies within the data. As a result, the Transformer has become a foundational model for developing more sophisticated AI systems [34].

A.1.3 Hybrid Models

Overview of Ensembling Techniques Ensembling is a fundamental approach in machine learning aimed at improving predictive performance by combining multiple models. By aggregating predictions, ensemble methods reduce errors and enhance generalization. The three primary ensembling approaches are:

Bagging (Bootstrap Aggregating): Bagging involves training multiple models on different subsets of the dataset, created through bootstrapping, and aggregating their predictions, typically by averaging or majority voting. This method is effective in reducing variance and combating overfitting. A classic example of this approach is the Random Forest algorithm, which combines multiple decision trees [5].

Boosting: Boosting sequentially trains models, where each successive model attempts to correct the errors made by its predecessors. By focusing on hard-to-predict instances, boosting reduces bias and improves model accuracy. Examples include Gradient Boosting Machines (GBMs) and AdaBoost [12].

Stacking (Stacked Generalization): Stacking is a sophisticated ensembling technique that combines the outputs of multiple base models through a meta-model. Unlike bagging or boosting, which aggregate predictions through simple rules, stacking trains a meta-model to learn the optimal way of combining base model predictions [37].