SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

Software Architecture Design for Scalable and Distributed Systems

John Bennet Johnson[1], Anjali Bhardwaj[2], Chandrashekhar Ramesh Ramtirthkar[3], Keerthika K[4], Athira K[5], Sunila Choudhary[6], Rohit Goyal[7], Dr. R. Rajalakshmi[8]

[1] Assistant Professor, Department of Computer Science and Engineering, Presidency University, Bangalore, Karnataka, India. Email: John.bennet@presidencyuniversity.in ORCID: 0009-0005-5457-5832
[2] Assistant Professor, Department of Computer Science & Engineering, Noida International University, India. Email: anjali.bhardwaj@niu.edu.in
[3] Associate Professor, Department of Mechanical Engineering, Vishwakarma Institute of Technology, Pune, Maharashtra 411037, India. Email: chandrashekhar.ramtirthkar@vit.edu
[4] Assistant Professor, Department of Computer Science, Meenakshi College of Arts and Science, Meenakshi Academy of Higher Education and Research, India. Email: keerthikak@maher.ac.in
[5] Assistant Professor, Department of Commerce, Meenakshi College of Arts and Science, Meenakshi Academy of Higher Education and Research, India. Email: athirak@maher.ac.in
[6] Centre for Research Impact & Outcome, Chitkara University Institute of Engineering and Technology, Chitkara University, Rajpura, Punjab 140401, India. Email: sunila.choudhary.orp@chitkara.edu.in ORCID: 0009-0000-2253-5557
[7] Associate Professor, Department of Computer Application, Dev Bhoomi Uttarakhand University, Dehradun, Uttarakhand, India. Email: hod.ca@dbuu.ac.in ORCID: 0000-0001-7978-4079
[8] Professor, Department of Electronics and Communication Engineering, Panimalar Engineering College, Chennai, Tamil Nadu, India. Email: rajeeramanathan@gmail.com ORCID: 0000-0002-4399-4071

Abstract
Modern distributed computing environments demand software architectures that are not only scalable but also capable of adapting dynamically to fluctuating workloads and heterogeneous operational conditions. This research presents a self-adaptive software architecture design for scalable and distributed systems, integrating the Monitor Analyze Plan Execute Knowledge feedback loop with a deep learning-based decision engine to enable intelligent system adaptation. Architecture incorporates the use of microservices in a distributed environment where adaptive agents monitor the system performance. The use of Interquartile Ranges (IQR) preprocessing helps in removing the outliers for stability, while the use of Exponential Moving Average (EMA) helps in detecting temporal patterns by smoothing out the variations. These observations are processed by a centralized controller, which coordinates adaptation strategies using a Green Anaconda optimized Deep Gated Recurrent Unit (GAO-DeepGRU) for optimal decision-making. The integration of DeepGRU enhances the system's ability to perform predictive scaling, workload balancing, and resource reconfiguration while minimizing risks such as over-provisioning and performance degradation. The architecture is evaluated against traditional DL methods, demonstrating superior performance in terms of convergence speed, adaptation accuracy, and system stability using Python. Experimental results indicate that the proposed architecture significantly improves scalability, fault tolerance, and responsiveness with an accuracy of 93.95%, a precision of 93.90%, a recall of 94%, and an F1-score of 93.95%. This research highlights the effectiveness of combining DL-driven intelligence with modern software architectural patterns to build autonomous, efficient, and highly scalable distributed systems suitable for cloud and edge computing environments.

Keywords: Software Architecture, Distributed Systems, Microservices, Deep Learning (DL).

1. Introduction

The software industry significantly grown and strives to meet increasingly high customer demands for software quality and reliability. To achieve this, the field of Software Engineering emerged as a discipline that addresses all phases of software development. It provides structured methodologies for developing software products and managing projects, processes, and resources to ensure system efficiency and reliable execution [1]. The rapid expansion of cloud, edge, and distributed computing is creating a strong demand for a unified orchestration model capable of managing heterogeneous, geographically distributed, and resource-constrained environments [2]. However, traditional cloud-based architectures exhibit significant limitations. It is designed to support autonomous, decentralized decision-making across diverse computing layers, and it also faces challenges in seamlessly integrating non-container-native devices and legacy systems [3].The advancement of technology, with an increasing number of devices, is now embedded with intelligent computing capabilities. Many of these are implemented as embedded systems, which are specialized computer-based components designed to perform dedicated tasks within the larger systems. Such systems often operate as subsystems for automated monitoring and control in complex infrastructures [4]. The adaptive feedback mechanisms and system performance cannot continuously improve, particularly in dynamic and highly variable cloud-edge environments. Therefore, the development of self-adaptive, secure, and interpretable scheduling and orchestration mechanisms is essential for next-generation multi-cloud and distributed computing ecosystems [5]. These systems aim to support stakeholders across the software value chain by integrating planning, development, provisioning, deployment, operation, and maintenance processes into a unified intelligent model, thereby improving efficiency, reliability, and overall system coordination [6]. The Distributed Computing Continuum is a new paradigm that emerged as cloud computing gave way to edge computing. This paradigm combines the close proximity and diversity of edge devices with the cloud's almost limitless processing power, enabling seamless collaboration across multiple layers of computing infrastructure. As a result, infrastructure becomes a first-class concern in system design, playing a more critical role than in traditional Internet-based distributed systems [7, 8].

Research Aim: This research aims to create a self-adaptive software structure for environments with scalable and distributed microservices. To handle varying workloads and different conditions dynamically, by integrating the Monitor Analyze Plan Execute Knowledge (MAPE-K) feedback loop and a Green Anaconda Optimized Deep Gated Recurrent Unit (GAO-DeepGRU) decision engine. To enable predictive scaling with better workload balancing, fault tolerance, and overall efficiency for improvement in DL methods.

Research Organization: The research work is structured into five sections. **Section 1** provides the research background. **Section 2** focuses on reviewing existing research. **Section 3** discusses the process of gathering data, preprocessing, and feature extraction, as well as the proposed model. **Section 4** includes the results and discussion. **Section 5** presents the performance, constraints, and future work in the conclusion.

2. Related Work

A self-adaptive system for automatic maintenance of the quality requirements in dynamic environment changes was explored in [9] by integrating the use of the Q-Learning algorithm for maintaining distributed systems. Results showed that two self-adaptive systems learned faster, with an average speed-up of 33.7% and multiple adaptation actions. Investigating the entire new adaptation space, but only the collection of novel configurations, a self-adaptive system for improving distributed software quality in various settings was investigated in [10]. With the help of a decision tree (DT) model, for efficient software performance and quality, it showed high accuracy with a cluster size of 25.5%, and yet, it troubles with scalability issues, which remained limited in distributed systems. A self-adaptive DL model for ranking-based selection to improve the distributed software systems' latency was examined in [11]. Using the Ranked-Based Selection Particle Swarm Optimization (RSPSO) method, their results showed a 7.72% value for the ranking selection software with the RSPSO method. However, it needed lots of computational time to figure out the best hyperparameter settings. A self-adaptive strategy to boost fault tolerance in distributed software was developed in [12], using the Piezoelectric Vibration Absorbers (PVA) to enhance software quality. Compared to non-adaptive methods, it cut the maximum vibration amplitude by 30%. However, it works best for harmonic vibrations in propulsion systems and similar rotating equipment. Evaluation of a self-adaptive distributed software system for predictive accuracy and computational efficiency was conducted in [13]. By integrating a self-adaptive Differential Evolution–Kolmogorov–Arnold Network (SADE-KAN) to boost accuracy with the results reduced the Mean Absolute Percentage Error (MAPE) by up to 35%, but there are long-term dependencies, complex seasonal

patterns, and nonlinear system dynamics are all difficult to capture. An efficient self-adaptive software system for managing latency in distributed systems was analyzed in [14], using the Self-Adaptive Evolutionary Graph Regularized Broad Learning System (SaE-GBLS) to predict software accuracy. The results showed a 5.712% increase in accuracy. Nonetheless, it performed less effectively than DL methods due to limitations in feature extraction. Self-adaptive systems in edge cloud environments for refining their software behavior, which was explored in [15]. Using the Resistive Random-Access Memory (RRAM) to monitor how the software acts across the board. This resulted in higher accuracy and faster convergence toward the best performance. The internal threshold in RRAM hits an upper limit, and each agent only gets a set amount of time to figure things out through trial and error. Enhance self-adaptive software quality for microgrid optimization using the Self-Adaptive Gravitational Search Algorithm (SGSA) was conducted in [16]. It helps avoid early convergence and boosts solution quality. The result found that integrating battery storage cuts total generation costs by up to 49.7%. The application of Artificial Intelligence (AI) based fault tolerance using the DL model to improve the performance of distributed systems, was analyzed in [17]. Using the Visual Geometry Group 16 (VGG16) model to predict the performance of the system resulted in an accuracy of 92%. The use of the VGG16 model requires extensive computational power, making its implementation problematic on resource-constrained devices.

3. Methodology

The distributed microservices are self-adaptive by monitoring key performance metrics such as Central processing unit (CPU), memory, and resource utilization. In Figure 1, it illustrates how adaptation agents collected data from a distributed microservices environment that is implemented on cloud and edge nodes, involving several services and databases, as well as tracking the CPU usage and performance metrics.Then use Interquartile Ranges (IQR) for stabilizing data. Next, Exponential Moving Average (EMA) is used for feature extraction. The processed data gets analyzed by a GAO-optimized DeepGRU, which forecasts workloads, spots performance issues, and comes up with plans to adjust resources for scaling and load balancing with better performance.

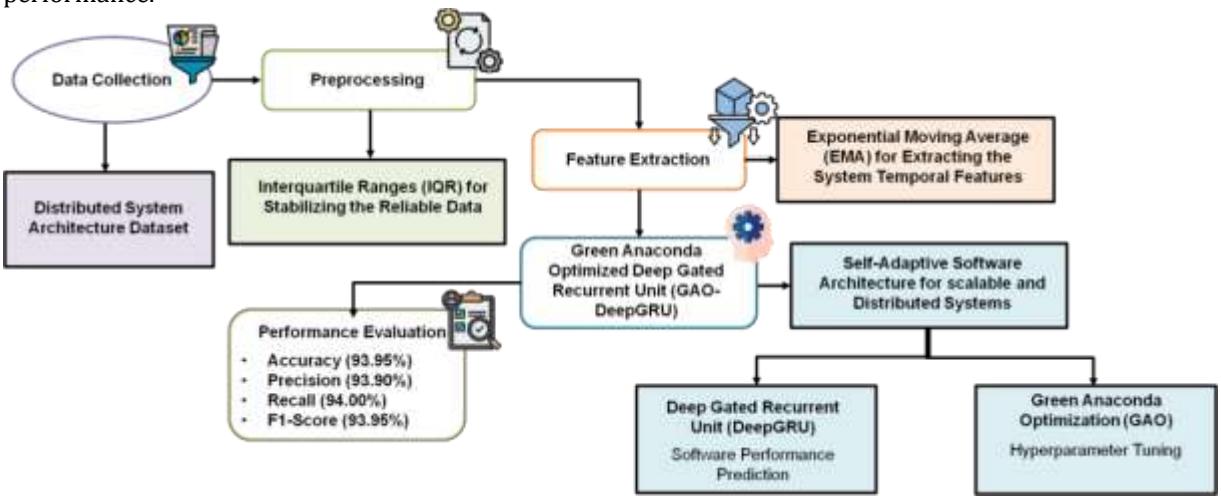

Figure 1: Methodology Workflow for the Self-Adaptive Software Architecture for Scalable Systems

3.1 Data Collection

A Self-Adaptive Software Architecture for Scalable and Distributed System dataset available at Kaggle (https://www.kaggle.com/datasets/niladriroy0/distributed-system-architecture-stress-and-failure), it consists of 200,000 snapshots and 21 columns. These include 11 decimal, 5 integer, and 5 string features. The dataset represents operating scenarios of different architectures like monolithic, microservices, event-driven, serverless, and hybrid systems. It tracks things such as deployment type, communication patterns, service numbers, databases, CPU usage, traffic, latency, and failures. The system states that root causes arise from complex interactions, and the data is great for predictive analysis, the dataset is split into 80% for training and 20% for testing, to evaluate the model.

3.2 Data Preprocessing using Interquartile Ranges (IQR) for stabilizing the Data with Reliable

Improving the robustness and reliability of distributed system performance data employed in a self-adaptive microservice architecture running on cloud-based multi-node infrastructure. The input includes raw data extracted from distributed adaptive modules, such as CPU usage and latency from various services and databases. The output is a reliable, processed dataset that guarantees a consistent performance of features for the software performance. This process employs the IQR method to filter out any outliers to workload spikes and network disturbances, with statistical smoothing using the mean aggregation formula presented in Equation (1).

$$\text{Mean} = \frac{b+2n+c}{4} \quad (1)$$

Where b represents the adjusted quartile values for the system performance metric from the distributed microservices in the cloud computing multi-node architecture. n indicates the median or the middle ground values for CPU utilization and latency from different services and databases within the self-adaptive system. c denotes the adjusted quartile values reflecting the maximum workload in the form of dynamic changes in traffic within the distributed system. The calculated Mean reflects a consistent state of the system by mitigating extreme variations.

3.3 Exponential Moving Average (EMA) for Extracting the System Temporal Features

The purpose of this feature extraction involves identifying useful temporal patterns from the dynamically changing performance metric values of a distributed microservices architecture hosted on both cloud and edge nodes. Input for this procedure includes sequences of performance-related telemetry information collected from different services and their corresponding databases, such as response times and inter-service communication delays. A temporally stable representation of workload features, allowing the development of a more accurate prediction for software performance. This step employs the use of an EMA algorithm to identify any long-term trends by minimizing short-term fluctuations, is expressed in Equation (2).

$$z_s = \alpha \odot y_s + (1-\alpha) \odot z_{s-1} \quad (2)$$

z_s refers to the smoothed temporal value of the feature at time s. The smoothing factor is indicated by α and is used to determine the past observations under dynamic adaptive software load conditions. y_s denotes the current system performance value at time s in terms of response times and inter-service communication delays. These two inputs are represented by $\odot$ Hadamard Product. z_{s-1} refers to the smoothed value from the past and keeps the history of system behavior intact for distributed nodes.

3.4 Self-Adaptive Software Architecture for Distributed System to Predict System Performance using Green Anaconda Optimized Deep Gated Recurrent Unit (GAO-DeepGRU)

The proposed Self-Adaptive Software Architecture for Distributed Systems using GAO-DeepGRU enhances dynamic decision-making in microservices-based setups. It uses a modular method where loosely coupled services talk through lightweight protocols across nodes and databases. Dedicated adaptation system continuously monitors key system metrics, including CPU utilization, service workloads, and communication latency. The GAO-DeepGRU model then processes this data to predict when to scale, balance workloads, and reconfigure resources. As a result, it handles varying loads efficiently, boosting scalability and fault tolerance.

Deep Gated Recurrent Unit (DeepGRU) for predicting system performance: DeepGRU models how different parts interact over time in microservices-based systems, helping predict performance under various load metrics such as CPU use, latency between services, active services, and database usage. Seeing how these metrics behave lets the system handle diverse architectures and MAPE-K setups. Using this information, the self-adapting system to be able to perform tasks such as workload adjustment, resource tuning, and scaling decisions and efficient in terms of resource utilization, is described by Equation (3).

$$\hat{z}(s) = g^k(s) X_{out} \quad (3)$$

$\hat{z}(s)$ refers to the output prediction about the system performance characteristics at time s, comprising important metrics related to system performance, such as workload and database accesses. The expression $g^k(s)$ is used to refer to the hidden state at the end of the deep GRU layer k. This parameter allows capturing hierarchical relations and temporal dependences with various services within the distributed architecture. X_{out} denotes the software weight matrix of the output layer that is responsible for transforming obtained features into predictions regarding system performance characteristics. The hidden state of system performance is expressed in Equation (4).

$$g_j^k(s) = \hat{g}_j^k(s) y_j^k(s) + g_j^k(s-1)(1-y_j^k(s)) \quad (4)$$

$g_j^k(s)$ represents the value of the hidden state of neuron j on layer k at time s for the behavior of different microservices and database connections. $\hat{g}_j^k(s)$ denotes the new hidden state, obtained based on new system inputs, such as workload and database accesses.g_j^kstands for MAPE-K setup of software system state, and $y_j^k(s)$ refers to the update gate that controls the extent to which new behavior is incorporated or preserved through time.

Hyperparameter Tuning with Green Anaconda Optimization (GAO): The proposed self-adaptive distributed system, the GAO algorithm tunes hyperparameters to boost the DeepGRU model's predictive performance. GAO systematically explores the hyperparameter space, tweaking things like the learning rate, GRU units, dropout rate, batch size, and dense layer neurons. With the system metrics, including database load, patterns, the DeepGRU model with convergence, accuracy, and more reliable workload in distributed settings, is expressed in Equation (5).

$$Z = \{z_1, z_2, \dots, z_m\} \quad (5)$$

Where Z represents the population of hyperparameter candidates to be sampled for the GAO algorithm, each $z_1, z_2, \dots$ is the solution vector corresponding to a specific hyperparameter configuration that is assessed using system-level metrics obtained from the distribution of services, database usage, and CPU activity. The values of z_m correspond to DeepGRU hyperparameters like the learning rate, number of GRU units, batch size, and dropout ratio, and number of dense neurons. The optimizing function mathematically is expressed in Equation (6).

$$\text{fitness function} = \text{maximizing}(B) \quad (6)$$

Bdepicts to the predictive accuracy of the DeepGRU method, whereas the fitness function is the optimizing function of GAO that strives to maximize the performance metrics of the system, which include workload prediction accuracy, scalability efficiency, and response adaptability in distributed microservices, is expressed in Equation (7).

$$BEK^j = \{Y_{l_j} : E_{l_j} < E_j \text{ and } l_j \neq j\} \quad (7)$$

BEK^j represents the competitive fitness-linked set of individual j, and Y_{l_j} refers to thecompetitors of software system solution j. E_{l_j} and E_j are fitness values of the competitor solution $l_j \neq j$competitors and the current software system solution j, which provides the system's performance with CPU efficiency, isexpressed in Equation (8).

$$j = 1, 2, \dots, M \text{ and } l_j \in \{1, 2, \dots, M\} \quad (8)$$

Where Mrepresents the number of candidate system solutions representing the total population size, j and $l_j \in \{1, 2, \dots, M\}$ refer to the individual GAO population. It ensures the distributed system load and deployment scenarios, is expressed in Equation (9).

$$QD_i^j = \frac{BEE_i^j - BEE_{max}^j}{\sum_{m=1}^{m_j} BEE_m^j - BEE_{max}^j} \quad (9)$$

QD_i^j represents the probability contribution made by candidate solution i for population j. Here, BEE_i^j and BEE_{max}^j stand for optimal configuration improving resource utilization in microservices architecture. BEE_m^j considers the system-level measurements, such as CPU usage. Where ∑represents the symbol for summation, m_j is the variable used to index each entity for the architecture of the distributed system. Here, the lower limit $m = 1$ refers to the system element being evaluated, the hyperparameter value expressed in Equation (10).

$$Y_{J,E}^{q1} = y_{j,e} + s_{j,e}\left(TG_e^j - J_{j,e}\right) \quad (10)$$

$Y_{J,E}^{q1}$ represents the updated solution vector, where $y_{j,e}$ denotes the current hyperparameter value for dimension e. The random coefficient,$s_{j,e}$ adds a stochastic component to the search algorithm. TG_e^j shows the effect of the scaling factor with the search process. Interaction influences $J_{j,e}$ represents the distributed system-level metrics such as CPU load andcommunication overhead, the iteration process is expressed in Equations (11 and 12).

$$Y_j = \begin{cases} Y_j^{q1}, G_j^{q1} < G_j, \\ Y_j, \quad \text{else}, \end{cases} \quad (11)$$

$$Y_{j,e}^{q1} = y_{j,e} + \left(1 - 2s_{j,e}\right)\frac{vc_e - mc_e}{s} \quad (12)$$

Y_j represents the system candidate solution at iteration j. In particular, Y_j^{q1} represents the system solution vector, and the criterion $G_j^{q1} < G_j$ ensures the inclusion in the better system performance, like increased

throughput, decreased latency, and balanced database load. In Equation (12), $Y_{j,e}^{q1}$ represents the updated solution vector and $y_{j,e}$ denotes the current hyperparameter value for dimension e. The random coefficient, and$s_{j,e}$, adds a stochastic component to the search algorithm. The updated hyperparameter value, where vc_e and mc_e denote the upper and lower bounds of the system bounds. The parameter s corresponds to the iteration number for convergence. The hyperparameter search can be performed for different distributed system loads and configurations, as expressed in Equation (13).

$$Y_j = \begin{cases} Y_j^{q2}, G_j^{q2} < G_j, \\ Y_j, \quad \text{else}, \end{cases} \tag{13}$$

Where Y_j represents the system candidate solution at iteration j, and Y_j^{q2} represents the optimized result obtained from the secondary update, while $G_j^{q2} < G_j$ signifies the fitness value for the optimized result. The above selection strategy makes sure that the best hyperparameters are preserved, which helps achieve better prediction. The hyperparameters of the proposed GAO-DeepGRU model for scalable and distributed systems is present in Table 1.

Table 1: Hyperparameters of Proposed GAO-DeepGRU for the Self-Adaptive Software Architecture for Scalable and Distributed Systems

Module	Key Parameter	Value / Range
Input	Feature dimension	Dataset features
GAO	Population / Iterations	20 / 30
	LR / Batch size	0.0001–0.01 / 16–64
	GRU units	64–256 / 32–128
	Dropout / Dense	0.10–0.50 / 32–128
DeepGRU	Units (final)	128 / 64 and 2 layers
	Activations	Tanh / Sigmoid / Softmax
Training	Loss / Val split	Cross-entropy / 0.20 and 50 epochs
Evaluation	Metrics	Accuracy, Precision, Recall, F1

4. Result and Discussion

The self-adaptive architecture based on the GAO-DeepGRU method proved to be better when implemented in a microservices-enabled distributed environment by handling several services and communication. The experimental setup of a software configuration for a Self-Adaptive Distributed Software System is presented in Table 2.

Table 2: Experimental Setup for the Self-Adaptive Distributed Software Systems

Component	Key Configuration
Processor	Intel i7 / AMD Ryzen 7 or higher and 16 GB+ RAM
GPU	NVIDIA with CUDA, 4 GB+ VRAM
OS	Windows 10/11 or Ubuntu 20.04+
Language / IDE	Python 3.10, Jupyter / Colab / VS Code
DL Framework	TensorFlow / Keras
ML / Data Libraries	Scikit-learn, NumPy, Pandas / Green Anaconda Optimizer
CUDA/cuDNN	CUDA 11.8+, cuDNN 8.x+

4.1 Exploratory Data Analysis

In Figure 2(a), it describes the types of Maps failures such as retry storms and network partitions according to network latency and CPU usage. It is used to indicate the operating conditions under which various types of adaptive recovery strategies are activated. In Figure 2(b), it displays the relation between the occurrence of Correlates errors and P95 latency in terms of the open/closed state of the circuit breaker. The figure demonstrates the efficacy of the automated scale in maintaining system stability during periods of high error

occurrence. In Figure 2(c), a comparison of the deployment modes (on-premise, cloud, and hybrid) in dealing with simultaneous CPU and memory stress is demonstrated. The Comparison between average and P95 latencies under various communication architectures (synchronous, asynchronous, and mixed) is shown in Figure 2(d).

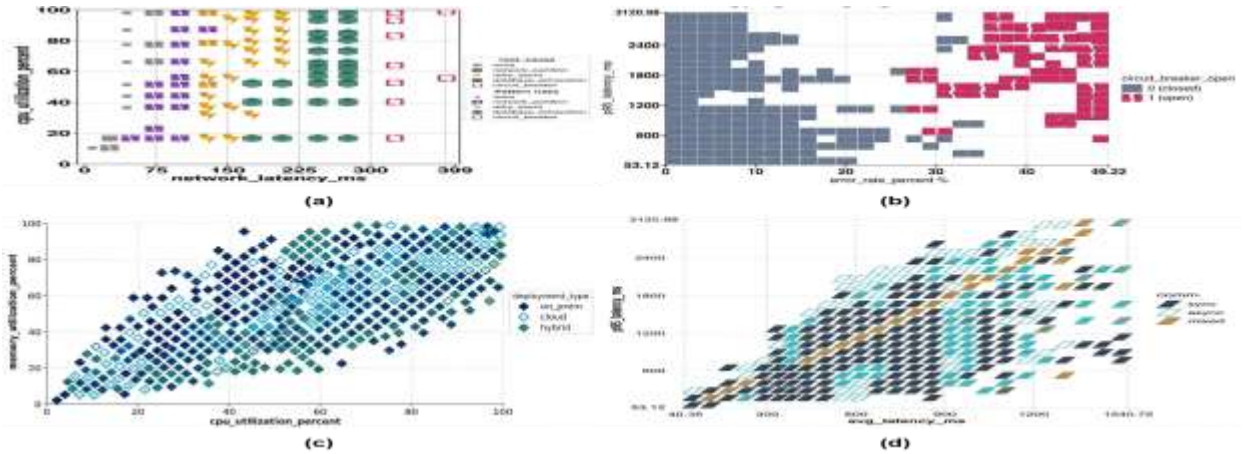

Figure 2: Data Visualization of (a) Root-Cause Mapping by Latency and CPU Usage, (b) Distribution of Circuit Breaker State, (C) Type of Deployment for Resource Utilization, and (d) Type of Communication with Latency Correlative Lattice for a self-adaptive architecture of scalable and distributed systems

4.2Performance Evaluation

The proposed self-adaptive distributed system architecture is based on evaluation metrics such as accuracy, precision, recall, and F1-score, which are used to measure GAO-DeepGRU, predicts system adaptations. **Accuracy** measures the predicted system states are correct across various deployments, communication, and resource uses. **Precision** checks how reliable adaptation actions are for particular services and databases, while **Recall** looks at the model's ability to adaptations under differing CPU loads and workloads. The **F1-score** gives a good balance between precision of reliable adaptation and recall of the model's ability in workload balancing. In Table 3 and Figure 3 a comparison evaluation of the existing and proposed models for the self-adaptive architecture of the distributed systems is shown. The existing VGG 16 [17] model achieves a high accuracy of performance prediction for scalable software. The Proposed GAO-DeepGRU model shows a better performance with the accuracy of 93.95%, precision of 93.90%, and recall of 94%, and F1-Score of 93.95% for a self-adaptive architecture in and distributed systems.

Table 3: Comparison analysis of Existing and Proposed Models for Self-Adaptive Distributed Systems

Methods	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
VGG16 [17]	92	91.73	91.70	91.70
GAO-DeepGRU [Proposed]	93.95	93.90	94.00	93.95

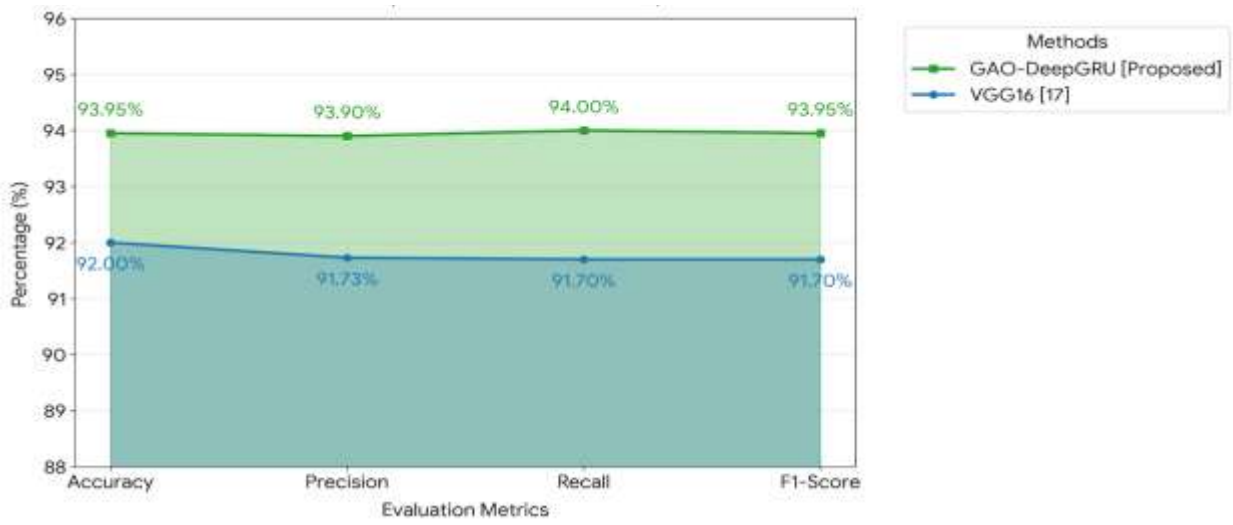

Figure 3: Performance Evaluation of Existing and Proposed Models for Distributed Systems

4.3 Discussion

Developing a self-adaptive software architecture for scalable and distributed systems to improve dynamic resource management, fault tolerance, and optimization of performance in cloud-edge ecosystems, the existing research using the VGG 16 model [17] involves the distributed deployment of the system on multiple nodes with services being loosely coupled using lightweight APIs for better scalability. But such architectures are characterized by high CPU load, communication overheads, and poor coordination between services and databases when dealing with changing workloads. To overcome these Problems the MAPE-K loop and the proposed GAO-deepGRU model decision-making algorithm were used. By applying this model, to minimize CPU load, ensure better service coordination, and optimize interactions with databases in distributed systems.

5. Conclusion

Designing a self-adaptive software architecture that uses a distributed model based on a microservices architecture deployed in multi-node cloud environments is the aim of this research. The data related to the system for extracting the distributed adaptation agents that measuring latency, throughput, and CPU usage on services and databases processed using the IQR technique. Extraction of features using EMA, and the proposed GAO-DeepGRU model can help establish intelligent communication-driven adaptation on loosely coupled services. The proposed GAO-DeepGRU model showed an accuracy rate of 93.95%, precision of 93.90%, recall of 94%, and F1-score of 93.95%. The designed GAO-DeepGRU self-adaptive system faces limitations due to the centralization of the controller in such a way that, in very large systems, there to be a bottleneck due to the centralized controller. The system incurs high computational costs during monitoring of several databases and service requests, and high CPU usage and workload. Future research focus to consider fully decentralized self-adaptation, self-adaptive decision-making agents at the edge, and optimizations in inter-service communication with others.

REFERENCES

1. Aguayo, O., Sepúlveda, S. and Mazo, R., 2024. Variability management in self-adaptive systems through deep learning: A dynamic software product line approach. Electronics, 13(5), p.905. https://doi.org/10.3390/electronics13050905
2. Deligiannakis, N., Papataxiarhis, V., Loukeris, M., Hadjiefthymiades, S., Touloupou, M., Ul Hassan, S.M., Herodotou, H., Moustakas, T., Bampis, E., Ioannidis, K. and Michailidis, I.T., 2026. DACCA: Distributed Adaptive Cloud Continuum Architecture. Future Internet, 18(2), p.74. https://doi.org/10.3390/fi18020074
3. Andersson, J., Caporuscio, M., D'Angelo, M. and Napolitano, A., 2023. Architecting decentralized control in large-scale self-adaptive systems: J. Andersson et al. Computing, 105(9), pp.1849-1882. https://doi.org/10.1007/s00607-023-01167-9

4. Ciopiński, L., 2025. Methodology of an energy efficient-embedded self-adaptive software design for multi-cores and frequency-scaling processors used in real-time systems. Electronics, 14(3), p.556. https://doi.org/10.3390/electronics14030556
5. Zhu, H., Guo, Y. and Tan, X., 2024. Self-adaptive moving least squares measurement based on digital image correlation. Optics, 5(4), pp.566-580. https://doi.org/10.3390/opt5040042
6. Dobaj, J., Riel, A., Macher, G. and Egretzberger, M., 2023. Towards devops for cyber-physical systems (cpss): Resilient self-adaptive software for sustainable human-centric smart cps facilitated by digital twins. Machines, 11(10), p.973. https://doi.org/10.3390/machines11100973
7. Casamayor Pujol, V., Morichetta, A., Murturi, I., Kumar Donta, P. and Dustdar, S., 2023. Fundamental research challenges for distributed computing continuum systems. Information, 14(3), p.198. https://doi.org/10.3390/info14030198
8. Pahl, C., Barzegar, H.R. and El Ioini, N., 2025. Quality Management for AI-Generated Self-Adaptive Resource Controllers. Machines, 14(1), p.25. https://doi.org/10.3390/machines14010025
9. Metzger, A., Quinton, C., Mann, Z.Á., Baresi, L. and Pohl, K., 2024. Realizing self-adaptive systems via online reinforcement learning and feature-model-guided exploration. Computing, 106(4), pp.1251-1272. https://doi.org/10.1007/s00607-022-01052-x
10. Wohlrab, R., Cámara, J., Garlan, D. and Schmerl, B., 2023. Explaining quality attribute tradeoffs in automated planning for self-adaptive systems. Journal of Systems and Software, 198, p.111538. https://doi.org/10.1016/j.jss.2022.111538
11. Luo, X. and Oyedele, L.O., 2022. A self-adaptive deep learning model for building electricity load prediction with moving horizon. Machine Learning with Applications, 7, p.100257. https://doi.org/10.1016/j.mlwa.2022.100257
12. Paixão, J., Foltête, E., Sadoulet-Reboul, E., Chevallier, G. and Cogan, S., 2024. Self-adaptive piezoelectric vibration absorber with semi-passive tunable resonant shunts. Journal of Sound and Vibration, 583, p.118424. https://doi.org/10.1016/j.jsv.2024.118424
13. Abbas, M., Che, Y., Maqsood, S., Yousaf, M.Z., Abdullah, M., Khan, W., Khalid, S., Bajaj, M. and Shabaz, M., 2025. Self-adaptive evolutionary neural networks for high-precision short-term electric load forecasting. Scientific Reports, 15(1), p.21674. https://doi.org/10.1038/s41598-025-05918-w
14. Cheng, L., Xiong, J., Duan, J., Zhang, Y., Chen, C., Zhong, J., Zhou, Z. and Quan, Y., 2024. SaE-GBLS: an effective self-adaptive evolutionary optimized graph-broad model for EEG-based automatic epileptic seizure detection. Frontiers in computational neuroscience, 18, p.1379368. https://doi.org/10.3389/fncom.2024.1379368
15. Bianchi, S., Muñoz-Martin, I., Covi, E., Bricalli, A., Piccolboni, G., Regev, A., Molas, G., Nodin, J.F., Andrieu, F. and Ielmini, D., 2023. A self-adaptive hardware with resistive switching synapses for experience-based neurocomputing. Nature Communications, 14(1), p.1565. https://doi.org/10.1038/s41467-023-37097-5
16. Ahmed, E.M., Zaki, Z.A., Kamarposhti, M.A., Barhoumi, E.M. and Colak, I., 2026. Probabilistic operational management of a renewable-based microgrid considering uncertainties using the self-adaptive gravitational search algorithm. Scientific Reports. https://doi.org/10.1038/s41598-026-42839-8
17. Gogineni, A., 2023. Artificial intelligence-driven fault tolerance mechanisms for distributed systems using deep learning model. J. Artif. Intell. Mach. Learn. Data Sci, 1(4). https://doi.org/10.51219/JAIMLD/anila-gogineni/519