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

Platform Engineering for Scalable and Distributed System Architectures

Ambika P[1], Dr. S. Prince Mary[2], Rahul Bhatt[3], Divya Paikaray[4], Ding Shengrong[5], Surabhi Kesarwani[6], Dr. R. Rajalakshmi[7], Priyadharshini K[8]

[1] Assistant Professor, Department of Commerce, Meenakshi College of Arts and Science, Meenakshi Academy of Higher Education and Research, India. Email: ambikap@maher.ac.in
[2] Professor, Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Chennai, Tamil Nadu, India. Email: princemary.cse@sathyabama.ac.in ORCID: 0000-0002-6840-8921
[3] Assistant Professor, Department of Computer Science & Engineering, Dev Bhoomi Uttarakhand University, Dehradun, Uttarakhand, India. Email: socse.rahul@dbuu.ac.in ORCID: 0009-0004-0486-9414
[4] Assistant Professor, Department of Computer Science & IT, Arka Jain University, Jamshedpur, Jharkhand, India. Email: divya.p@arkajainuniversity.ac.inORCID: 0000-0001-7886-1538
[5] Research Scholar, School of Engineering and Built Environment, Lincoln University College, Malaysia. Email: shengrong.phdscholar@lincoln.edu.my
[6] Assistant Professor, Department of School of Engineering & Technology, Noida International University, India. Email: surabhi.kesarwani@niu.edu.in
[7] Professor, Department of Electronics and Communication Engineering, Panimalar Engineering College, Chennai, Tamil Nadu, India. Email: rajeeramanathan@gmail.com ORCID: 0000-0002-4399-4071
[8] Assistant Professor, Department of Management Studies, Meenakshi College of Arts and Science, Meenakshi Academy of Higher Education and Research, India. Email: kpriyamba@maher.ac.in

Abstract
Platform engineering plays a pivotal role in enabling scalable and distributed system architectures for modern data-intensive applications. The main challenges are inefficient scalability, resource utilization, and performance degradation in distributed machine learning (ML) systems when handling large-scale, heterogeneous workloads. This research proposes a unified platform engineering framework that supports efficient deployment and execution of large-scale ML workloads across distributed environments. Unlike traditional paradigms such as MapReduce, the proposed system adopts fine-grained scheduling and dynamic resource orchestration to accommodate heterogeneous and evolving workloads. To enhance data quality and computational efficiency, distributed data preprocessing is performed using Min–Max normalization, while Principal Component Analysis (PCA) is incorporated as a feature extraction technique to reduce dimensionality and improve workload optimization. It leverages inherent characteristics of Emperor Penguins Colony -tuned Stochastic Decision tree (EPC-SDT) algorithms such as iterative convergence, stochastic optimization, and error tolerance to design adaptive mechanisms for bounded-latency synchronization and dynamic load balancing. EPC algorithm optimizes resource allocation and load balancing in distributed ML systems, enhancing convergence and performance through efficient search space exploration. SDT reduces computational complexity in large-scale ML decision-making while preserving accuracy in distributed environments. Experimental evaluation demonstrates that the proposed approach achieves significant improvements in accuracy (98.5%), training time(1.4min), inference time(0.02ms), and memory usage(1.0MB) compared to conventional distributed systems. They were implemented in Python. Overall, this research highlights how platform engineering principles can bridge the gap between system design and application requirements, providing a robust foundation for next-generation scalable and distributed architectures.

Keywords: Platform Engineering, Distributed Systems, Scalable Architecture, Machine Learning Systems, Distributed Machine Learning, Data Parallelism.

1. Introduction

Platform engineering is a core concept that enables the development of highly scalable and distributed computing platforms that can support today's enterprises' applications and services. Cloud computing, virtualization, microservice architecture, and container orchestration tools, including Kubernetes, have made it possible to build resilient and scalable systems in hybrid and multi-cloud environments [1]. Such systems leverage dynamic resource management, load balancing, service discovery, and automated deployment techniques to increase availability and efficiency. Moreover, DevOps culture and automation of infrastructure management have streamlined software delivery, further strengthening the role of platform engineering in building scalable distributed systems [2]. These days, the need for scalable and distributed computing platforms has increased due to the expanding use of AI/ML applications and solutions in a number of industries, including healthcare, the financial sector, manufacturing, and e-commerce. Adaptive resource allocation, scalable infrastructure support, and efficient workload scheduling are necessary to guarantee excellent computational performance with low operating costs and minimal energy consumption in an AI-based task. Cloud computing technologies offer on-demand access to computational resources, providing an effective platform for executing AI and ML workloads using dynamic scaling and resource allocation [3]. Predictive analytics, federated learning, and hybrid optimization techniques also help enhance workload allocation, resource utilization, and overall system efficiency in distributed environments [4-5]. It is due to these developments that this research seeks to design an adaptive platform engineering framework aimed at efficient implementation of distributed ML workloads using optimized scheduling, scalable orchestration, and effective resource management techniquesDespite all these innovations, several challenges are yet posed in distributed ML in terms of performance and scalability. Some of these challenges are workload fluctuations, heterogeneous resources, complexity due to multi cloud, security concerns, and heavy energy consumption [6]. The conventional resource management schemes lack proper management of workload, latency minimization, and scalability. In addition, there are inefficiencies in the utilization of resources and low performance in distributed computing environments [7]. Research aim: In this research, we have employed a wide range of techniques such as data and model parallelism, fine-grained scheduling, Min-Max normalization, PCA-based feature extraction, EPC-SDT optimization, and many others to overcome the aforementioned problem.

Research Organization: This research is divided into several parts. In Section 1, the importance of scalable and distributed computer architecture in data intensive applications is considered. The literature related to optimization of resources and engineering platforms is reviewed in Section 2. Section 3 describes the methodology and elaborates on the proposed model, PCA, and Min-Max normalization. Performance analysis in terms of execution time and scalability is the main focus of Section 4 and Section 5 concludes with key contributions.

2. Related work

The performance analysis of present machine learning (ML) and deep learning approaches for resource allocation and task scheduling in distributed computing is presented in Table 1. Enhanced execution time, energy utilization, resource allocation, and service level agreement (SLA) adherence have been noted.

Table 1: Previous research of ML and Optimization Techniques Workload Scheduling

Ref	Objective	Methodology	Key Results	Limitations
[8]	Minimize job completion time and reduce interference in ML clusters	Deep reinforcement learning using actor-critic + reward prediction model	16%–42% reduction in job completion time on the Kubernetes cluster	Improve scalability, generalization, and dynamic job handling
[9]	Improve resource utilization in deep learning systems	Deep Neural Network (DNN) aware sharding	Up to 279% throughput gain, 94% latency reduction	Improve dynamic allocation and scalability for larger environments
[10]	Optimize task-node mapping in clusters	Adaptive Genetic Orthogonal Convolutional Swarm (AGOCS) framework + ML classifiers + ensemble voting	98% accuracy with low misclassification	Reduce retraining cost, improve scalability, and reduce overfitting

[11]	Improve High-Performance Computing (HPC) scheduling, energy efficiency, and placement.	Virtualization + ML-based scheduling + real-time analysis	Better scheduling speed and optimized node placement	Limited in Field-Programmable Gate Array (FPGAs)
[12]	Optimize Virtual Machine (VM) placement, Service Level Agreement (SLA), and energy efficiency	Deep Q-Learning for workload prediction	Better CPU utilization, reduced energy usage, fewer SLA violations	Extend to diverse workloads, add optimization methods
[13]	Reduce energy consumption and improve task scheduling in IoT	Q-learning + distributed Reinforcement Learning (RL) + Dynamic Voltage and Frequency Scaling (DVFS)	52.26% energy reduction, 38% higher success rate	Improve scalability and communication overhead handling
[14]	Predict workload and energy for resource optimization	Linear Regression (LR), and the DL model Gated Recurrent Unit (GRU) + clustering methods	GRU lowest root mean square error(RMSE), 87.48% accuracy	Improve the energy model, and implement dynamic adaptation
[15]	Compare ML models for workload prediction	Taxonomy of 5 ML categories + unified benchmarking framework	Identifies trade-offs in accuracy, adaptability, and efficiency	Improve robustness, real-time prediction, and hybrid DL models
[16]	Monitor and predict network performance for distributed AI training workloads.	ML-based network performance monitoring and prediction models	Improves the prediction accuracy of network behavior	Needs improvement in real-time adaptability, scalability for large AI clusters

Research Gap: Despite progress in ML-based schedulers and resource allocation, issues persist in scalability, dynamic allocation, and energy efficiency, especially in complex workloads. Several existing approaches do not have real-time execution capabilities and optimized integration features. The approach presented in this research, which utilizes EPC-SDT, Min-Max normalization, and dynamic load balancing techniques, is considered a comprehensive platform to improve scalability, energy efficiency, and resource usage for big data ML applications.

3. Methodology

For large-scale ML implementation in a distributed computing system, the platform engineering design architecture has been introduced, which includes resource management and fine-grained scheduling. The combination of data parallelism and model parallelism is used to maximize scalability and flexibility. For performance optimization, PCA is applied to feature selection and Min-Max scaling is used for data preprocessing. Load balancing and latency synchronization are achieved through EPC-SDT algorithms, as illustrated in Figure 1.

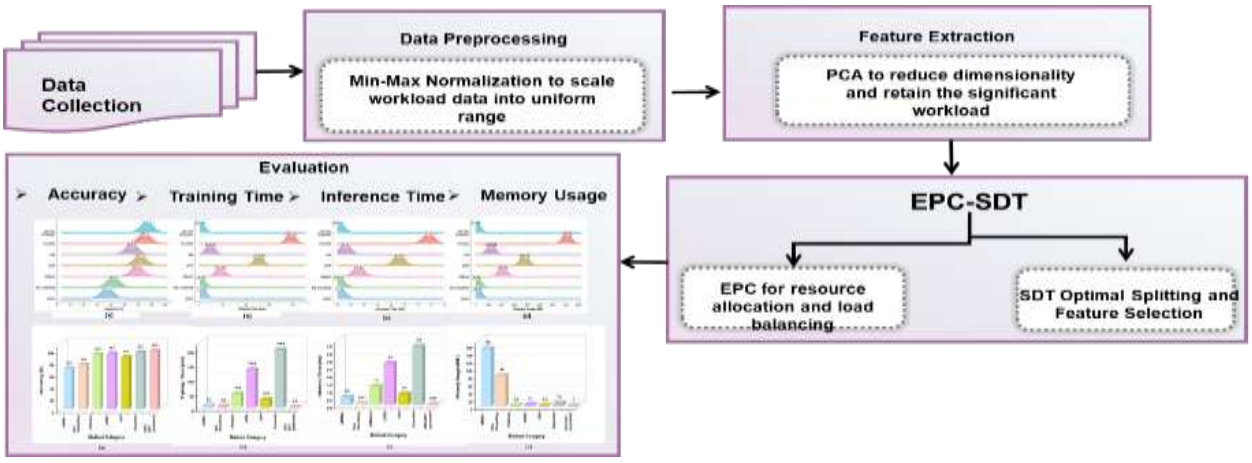

Figure 1: Methodology Flow of proposed model

3.1 Description of Dataset

A dataset for distributed workload placement for scalable machine learning was acquired from an open-source platform (https://www.kaggle.com/datasets/colabsss/adaptive-distributed-ml-workload-dataset). It has a total of 8800 entries of data with other metrics such as orchestration, synchronization, resource utilization, and performance metrics. It is divided into an 80/20 split between training and testing. This particular dataset may prove useful in research pertaining to distributed machine learning, load balancing, adaptive scheduling, and cloud-native platform development.

3.2 Data Preprocessing through Min-Max Normalization

Through the removal of the minimum value and then dividing by the full range, the min-max normalization will normalize the data to fit within a certain range, usually 0 to 1. The aim of this research is to investigate whether the min-max normalization technique can be applied to improve data handling during distributed machine learning processes.

$$U' = \frac{u-min}{max-min}\ (New\_max - New\_min)\_New\_min \quad (1)$$

In equation (1), where U' is output, u is input, max and min are the maximum and minimum values of the data, respectively. New_max and New_min are the newly defined maximum and minimum values.

3.3 Feature extraction using PCA

PCA stands for Principal Component Analysis which is an unsupervised machine learning approach used for dimensionality reduction to help reduce the number of features contained within a large dataset while retaining all essential features of the dataset. PCA is used in this research as a feature extraction method to improve the quality of the data. Construction of the Covariance Matrix, a square matrix indicating the relationship among several features, can be defined as:

$$N^{m*m} = n_{j,i}, (n_{j,i} = Cov(B_j, B_i)) \quad (2)$$

$$Cov(B_j, B_i) = \frac{\sum_{l=1}^{m}(B_{jl}-B_j')(B_{il}-B_i')}{m-1} \quad (3)$$

where B_j and B_i are the attributes, and m is the number of observations, B_j' and B_i' are the arithmetic of B_{jl} and B_{il}. $n_{j,}$ I is the element at row J and column i of the covariance matrix. Cov denotes the covariance between two variables B_j and B_i,$\sum_{l=1}^{m}$ represents summation over all $m$samples in the dataset used to compute covariance.

$$Nv = \delta v \quad (4)$$

where v is the original value, δ is a scalar value. Lastly, the new feature space is defined by choosing the eigenvectors that correspond to the highest eigenvalues to extract the principal components. The covariance matrix is used to determine eigenvalues and eigenvectors; the eigenvector v satisfies equation (4).

3.4 Optimizing Resource Allocation and Load Balancing in Large-Scale ML Workloads using EPC - SDT Algorithms

EPC algorithm is inspired by the behavior of Emperor Penguins in terms of their huddling process to achieve the optimization of resource allocation and load balancing in scalable ML workloads. The EPC algorithm exploits thermal radiation and spiral movement to achieve the optimization of resources. The EPC algorithm incorporates stochastic optimization that ensures the adaptation of performance variations. Moreover, the SDT increases scalability in large-scale ML workloads by adopting random sampling and feature reduction. This approach combines the use of Min-Max normalization and PCA. Thus, both the EPC and SDT are enhanced with the dynamic resource orchestration technique to achieve high execution, scalability, and resource efficiency.

Improving Scalability and Efficiency in Distributed Systems for Large-Scale ML with SDT Approach: SDT enhances the scalability of distributed systems for large-scale ML workloads using random sampling and feature reduction techniques. The primary goal is to develop a scalable and efficient distributed system for large-scale ML workloads to enhance computational performance, model accuracy, and resource utilization. In addition, with the help of PCA and Min-Max normalization, SDT is integrated into the platform engineering system that improves data quality. The EPC-SDT promotes task management using iterative convergence and error tolerance.

$$\min_{e_i} \sum_{e_i=\{-1,1\}} \frac{|T_i^{e_i}|}{|T_i|} G(T_i^{e_i}) \quad (5)$$

$s.t\ e_i(w) = sign(w^o - th)$ (6)

In equation (4), the objective function minimizes the decision variable $\underset{e_i}{min}$, which is binary (-1 or 1) for node i. The dataset for node i is T_i, and $T_i^{e_i}$ is the chosen subset based on e_i. The term $\frac{|T_i^{e_i}|}{|T_i|}$ indicates the ratio of selected to total samples at node i, while $G(T_i^{e_i})$ is the cost function calculated on this subset using an entropy function. Equation (6) calculates e_i using a weight w^o and a threshold th, establishing a split when the weight surpasses the threshold.

Optimizing Resource Allocation in Scalable Machine Learning Processes Using the EPC Algorithm: The EPC algorithm refers to a population-based optimization algorithm that derives its concept from the Emperor Penguin clustering phenomenon and seeks to optimize energy consumption and minimize heat loss in distributed computing systems. It works towards optimizing resource allocation and load balancing in scalable ML tasks through adaptive processes. Some unique characteristics of this algorithm include iterative convergence, stochastic optimization, and fault tolerance.

$R_{penguin} = B\varepsilon\sigma S_t^4$ (7)

$R = B\varepsilon\sigma\ S_t^4 f^{-\mu w}$ (8)

$w_l = bf^{a\frac{1}{a}In\{(1-R)f^{ab}+Rf^{a\beta}\}Cos\{\frac{1}{a}ln\{(1-R)f^{a\alpha}} + Rf^{a\beta}\}\}$ (9)

$z_l = bf^{a\frac{1}{a}In\{(1-R)f^{ab}+Rf^{a\beta}\}sin\{\frac{1}{a}ln\{(1-R)f^{a\alpha}} + Rf^{a\beta}\}\}$ (10)

The radiation or energy output associated with the penguin system is represented by equation (7) $R_{penguin}$, which creates a baseline with important variables like a constant B, emissivity ε, the Stefan–Boltzmann constant σ, and a temperature-related term S_t^4. Equation (8) modifies this by introducing a frequency-dependent attenuation factor that includes frequency f, a damping coefficient μ, and a weighting factor w. Equations (9) and (10) involve transformed outputs w_l and z_l that incorporate scaling and nonlinearity parameters b and a, an input variable f, and balancing weights $(1-R)$and R. A logarithmic term (ln) is used for nonlinear scaling, while Cos and sin cosine and sine functions establish orthogonal projections, creating a transformed representation. Weighting parameters α and β modulate the influence and phase of the mixed components in the transformation for $w_l$and z_l.

Algorithm 1: EPC–SDT-Based Distributed ML Framework

Input:
D → Dataset, MaxIter → Iterations, N → Population size, lb, ub → Bounds, R → Resources
Output:
O → Optimized scheduling
Begin
1. Load dataset D
2. For each Wi in D:
Apply Min-Max normalization:
$U' = \frac{u-min}{max-min}\ (New\_max - New\_min)\_New\_min$
Clean missing values, if any
End For
3. Apply PCA:
Compute the covariance matrix
$Cov(B_j, B_i) = \frac{\sum_{l=1}^{m}(B_{jl}-B_j')(B_{il}-B_i')}{m-1}$
$Nv = \delta v$
Select top eigenvectors → Feature set F
4. SDT Classification:
For each node Ti:
Compute split:
$\underset{e_i}{min} \sum_{e_i=\{-1,1\}} \frac{|T_i^{e_i}|}{|T_i|} G(T_i^{e_i})$
$e_i(w) = sign(w^o - th)$
Select/Remove features based on importance

```
    End For
5. Initialize EPC population:
    Bj = lb + rand(0,1)(ub - lb)
6. For t = 1 to MaxIter:
    For each penguin j:
        Compute R_penguin and R
        Update position using spiral movement
        Switch between exploration/exploitation
        Update solution
    End For
    Update the global best fitness
  End For
7. Perform:
    Resource orchestration
    Load balancing
    Scheduling deployment
8. Return O
End
```

As presented in Algorithm 1, the proposed EPC-SDT model unified platform for distributed machine learning is presented. The process includes data loading, Min-Max normalization, and PCA feature extraction. The EPC algorithm is used to optimize resource distribution, whereas SDT guarantees data distribution that maximizes its effectiveness based on an objective function and threshold conditions. The two techniques are optimized iteratively using population search and fitness assessment to produce a well-trained model.

4. Results and discussion

Experimental results for the suggested EPC-SDT model are discussed in this section. Experimental results also involve the comparison of the EPC-SDT model with machine learning models like XGBoost, LSTM, and classical time series forecasting models such as ARIMA model, exponential smoothing, and SVR [16]. The whole experimentation process was carried out using the Python programming language.

4.1 Metrics explanation

Accuracy: Model accuracy refers to the quality of forecast or classification made by the model as compared to the actual result. High accuracy of the model will mean the improved performance of the model and its capability to make decisions. Accuracy in machine learning tasks reflects the capability of generalization and allows using the model with any data sets. While assessing forecasts' accuracy in distributed systems, this characteristic is crucial. **Training Time:** Training time is the term for the duration of training of the model, which depends on the processing speed, the complexity of the algorithm, and the volume of the data set. As far as a faster and more effective model training process needs to be achieved, it is essential to consider low training time for the model. **Inference Time:** Inference time is the term describing the time of prediction of new values based on input data. **Memory Usage:** Memory usage is another critical aspect reflecting the volume of the computational memory used during training and inference processes.

4.2 Comparative evaluation

To guarantee that the proposed algorithm and baseline methods have an equal foundation of comparison, the recommended algorithm was trained and tested on an existing benchmark data set [16]. XGboost, LSTM, SVM, and Ensemble are examples of sophisticated algorithms that outperform the suggested technique in terms of accuracy, but at the cost of increased training time and memory usage. Of the six benchmark methods, Table 2 and Figure 2(a) show that the EPC-SDT model has the highest accuracy of 96.5%. Figure 2(b) demonstrates that the suggested model has good computational efficiency and requires less training time.Figure 2(c) reflects the extremely fast inference time for the proposed model, providing real-time prediction ability. Figure 2(d) shows memory usage by the suggested algorithm where EPC-SDT consumes the least memory.

Table 2: Performance evaluation of the proposed model is trained using the existing dataset

Method Category	Accuracy	Training Time	Inference Time	Memory Usage

ARIMA [16]	67.3%	5.2 min	0.1ms	128 MB
Exp. Smoothing [16]	71.8%	2.8 min	0.05ms	64 MB
XGBoost [16]	89.3%	45.2 min	0.8ms	2.1GB
LSTM [16]	91.7%	127.8 min	2.3ms	3.8 GB
SVR [16]	87.4%	23.6 min	0.3ms	1.2GB
Ensemble [16]	94.7%	196.6 min	3.4ms	7.1GB
EPC-SDT [Proposed]	96.5%	1.6 min	0.03ms	1.1MB

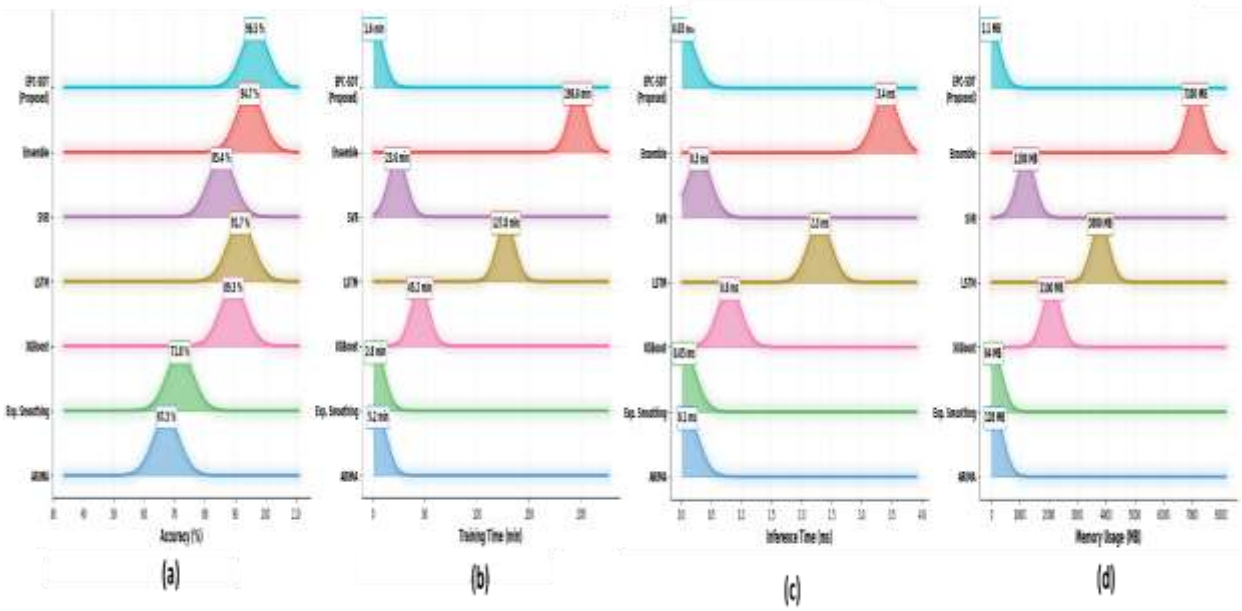

Figure 2: Graphical representation (a) accuracy, (b) training time, (c) Inference time, and (d) memory usage in existing data

To give a fair comparison, the adaptive distributed ML workload data set from the current research is used to evaluate the aforementioned models; this data set is shown in Table 3. Compared to the current model, EPC-SDT offers an exceptional decrease in training time and memory consumption, producing effective and scalable results in distributed workload scenarios. The enhanced performance of the suggested EPC-SDT model according to different performance criteria is shown in Figure 3. The suggested EPC-SDT outperforms all other models in the experiment with an accuracy of 96.5%, as shown in Figure 3(a).In Figure 3(b), EPC-SDT takes the shortest training time of 1.6 minutes, suggesting effective computational performance. Moreover, from Figure 3(c), EPC-SDT offers the lowest inference latency of 0.03 ms, allowing rapid real-time prediction. Finally, from Figure 3(d), EPC-SDT consumes minimal memory usage of 1.1 MB compared to several existing models.

Table 3: Comparison of existing models and the proposed model trained on the proposed dataset

Method Category	Accuracy	Training Time	Inference Time	Memory Usage
ARIMA	68.3%	6.2 min	0.5ms	150 MB
Exp. Smoothing	74.8%	4.8 min	0.07ms	80 MB
XGBoost	92.3%	47.2 min	1.2ms	2.4 GB
LSTM	93.7%	130.8 min	2.7ms	4.1 GB
SVR	87.4%	27.6 min	0.7ms	1.5 GB
Ensemble	95.7%	199.6 min	3.8ms	7.6 GB
EPC-SDT[Proposed]	98.5%	1.4 min	0.02 ms	1.0 MB

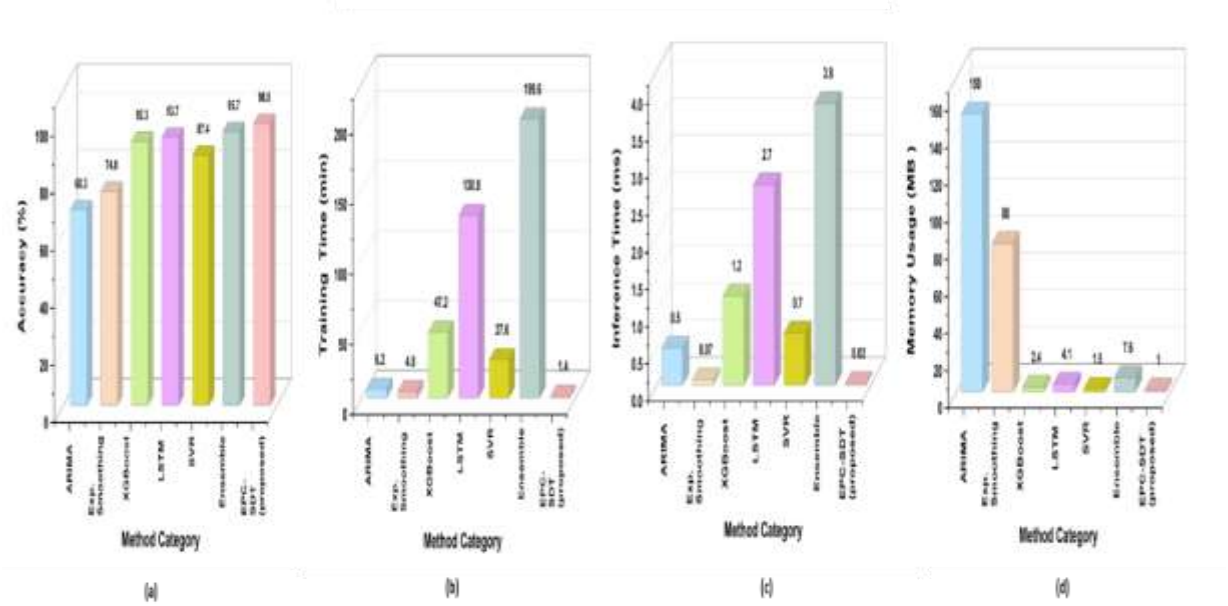

Figure 3: Graphical representation (a) accuracy, (b) training time, (c) Inference time, and (d) memory usage in existing data

There exist constraints in platform engineering and machine learning-based scheduling, including inefficient resource allocation, high communication costs, and limited adaptability when dealing with varying workloads in distributed and multicloud environments. However, these shortcomings can be effectively handled by the new proposed architecture that allows for dynamic and intelligent management of resources through scheduling and iterative improvements. EPC will improve the adaptability and balancing of the workload, while SDT ensures scalability with lower computational cost.

5. Conclusion

An extensive platform engineering framework integrating precise scheduling, flexible resource management, data parallelism, and model parallelism for machine learning deployment in a distributed system setup has been offered. In additions, new techniques such as the development of the EPC-SDT technique for dynamic resource allocation, latency alignment, load balancing, data normalization through Min–Max scaling, and PCA for dimensionality reduction have been introduced. The outcomes show that the proposed method performs better than typical distributed computing frameworks that were developed in Python and are suitable for cloud-native and data-centric environments, especially concerning accuracy (98.5%), training time (1.4 min), prediction time (0.02 ms), and memory utilization (1.0 MB). Its reliance on preprocessing, potential overhead in multicloud environments, and model retraining after workload change can be considered its limitations. Future work may involve extending the proposed model to energy-efficient applications, incorporating edge-cloud integration, and decreasing orchestration costs through reinforcement learning.

References

1. Priyadarshini, S., Sawant, T.N., Bhimrao Yadav, G., Premalatha, J., and Pawar, S.R., 2024. Enhancing security and scalability by AI/ML workload optimization in the cloud. Cluster Computing, 27(10), pp.13455-13469. https://doi.org/10.1007/s10586-024-04641-x
2. Yang, L., Wen, F., Cao, J., and Wang, Z., 2022. EdgeTB: A hybrid testbed for distributed machine learning at the edge with high fidelity. IEEE Transactions on Parallel and Distributed Systems, 33(10), pp.2540-2553. https://doi.org/10.1109/TPDS.2022.3144994
3. Kumar, A. and Priyadarshini, S., 2024. Adaptive AI Infrastructure: A Containerized Approach For Scalable Model Deployment. International Research Journal of Modernization in Engineering Technology and Science, 6(11), pp.5827-5834. https://www.doi.org/10.56726/IRJMETS64700

4. Hosseinzadeh, M., Haider, A., Rahmani, A.M., Gharehchopogh, F.S., Rajabi, S., Khoshvaght, P., Porntaveetus, T., and Lee, S.W., 2025. SDN-Based NFV deployment for multi-objective resource allocation in edge computing: A deep reinforcement learning for IoT workload scheduling. Sustainable Computing: Informatics and Systems, p.101218. https://doi.org/10.1016/j.suscom.2025.101218
5. Mizan, T. and Taghipour, S., 2022. Medical resource allocation planning by integrating machine learning and optimization models. Artificial Intelligence in Medicine, 134, p.102430. https://doi.org/10.1016/j.artmed.2022.102430
6. Rajawat, A.S., Goyal, S.B., Kumar, M., and Malik, V., 2024. Adaptive resource allocation and optimization in cloud environments: Leveraging machine learning for efficient computing. In Applied Data Science and Smart Systems (pp. 499-508). CRC Press. https://dx.doi.org/10.1201/9781003471059-64
7. Thota, R.C., 2023. Optimizing Kubernetes workloads with AI-driven performance tuning in AWS EKS. International Journal of Science and Research Archive, 9(2), pp.1-11. https://doi.org/10.30574/ijsra.2023.9.2.0546
8. Bao, Y., Peng, Y. and Wu, C., 2022. Deep learning-based job placement in distributed machine learning clusters with heterogeneous workloads. IEEE/ACM Transactions on Networking, 31(2), pp.634-647. https://doi.org/10.1109/TNET.2022.3202529
9. Jin, X., Bai, Z., Zhang, Z., Zhu, Y., Zhong, Y., and Liu, X., 2024. Distmind: Efficient resource disaggregation for deep learning workloads. IEEE/ACM Transactions on Networking, 32(3), pp.2422-2437. https://doi.org/10.1109/TNET.2024.3355010
10. Sliwko, L., 2024. Cluster workload allocation: A predictive approach leveraging machine learning efficiency. IEEE Access, 12, pp.194091-194107. https://doi.org/10.1109/ACCESS.2024.3520422
11. Dakić, V., Kovač, M. and Slovinac, J., 2024. Evolving high-performance computing data centers with Kubernetes, performance analysis, and dynamic workload placement based on machine learning scheduling. Electronics, 13(13), p.2651. https://doi.org/10.3390/electronics13132651
12. Ahamed, Z., Khemakhem, M., Eassa, F., Alsolami, F., Basuhail, A. and Jambi, K., 2023. Deep reinforcement learning for workload prediction in federated cloud environments. Sensors, 23(15), p.6911. https://doi.org/10.3390/s23156911
13. Oustad, E., Younesi, A., Ansari, M., Safari, S., Soleimani, M.A., Henkel, J., and Ejlali, A., 2025. DIST: distributed Learning-Based Energy-Efficient and reliable task scheduling and resource allocation in fog computing. IEEE Transactions on Services Computing, 18(3), pp.1336-1351. https://doi.org/10.1109/TSC.2025.3568255
14. Khan, T., Tian, W., Ilager, S. and Buyya, R., 2022. Workload forecasting and energy state estimation in cloud data centres: ML-centric approach. Future Generation Computer Systems, 128, pp.320-332. https://doi.org/10.1016/j.future.2021.10.019
15. Saxena, D., Kumar, J., Singh, A.K. and Schmid, S., 2023. Performance analysis of machine learning-centered workload prediction models for cloud. IEEE Transactions on Parallel and Distributed Systems, 34(4), pp.1313-1330. https://doi.org/10.1109/TPDS.2023.3240567
16. Li, J., Ren, W., and Wu, X., 2025. Machine Learning-Based Network Performance Monitoring and Prediction for Distributed AI Training Workloads. Journal of Advanced Computing Systems, 5(10), pp.1-17. https://doi.org/10.69987/JACS.2025.51001