Smart Load Balancer for Cloud Computing using Machine Learning

***Abstract*—The rapid growth of cloud computing has introduced unprecedented challenges in managing dynamic, heterogeneous workloads across large-scale distributed infrastructures. Central to addressing these challenges is the problem of load balancing—optimally distributing tasks across available resources to maximize system responsiveness, resource utilization, and adherence to service-level objectives. Recent advances in machine learning (ML) and deep learning (DL) have enabled the development of intelligent, adaptive load balancing strategies that surpass traditional static and heuristic methods. This project proposes a Smart Load Balancer for Cloud Computing using Machine Learning, leveraging a two-phase approach for predictive task classification and resource allocation. Employing state-of-the-art models such as XGBoost and Artificial Neural Networks (ANN), the framework utilizes historical workload traces from Google's Borg cluster data.**

Keywords—Cloud Computing, Load balancing, Google Cluster Dataset, Machine Learning, Deep Learning, XGBoost, ANN, Accuracy, Precision, F1 Score

I. INTRODCUTION

Cloud computing has emerged as a foundational technology in delivering flexible, scalable, and cost-efficient computing resources over the internet. As cloud infrastructures expand in scale and complexity, managing the dynamic and heterogeneous workloads efficiently has become a critical challenge. Among many operational priorities, load balancing—the process of distributing computing tasks across available resources to optimize performance and utilization—plays a pivotal role in maintaining system responsiveness, reducing bottlenecks, and ensuring adherence to Service Level Agreements (SLAs). [1]

At the core of effective load balancing lies the problem of resource allocation, which entails assigning diverse workloads to a pool of virtual machines (VMs) and physical servers while considering CPU capacity, memory, and task priority constraints. This challenge is NP-complete and becomes even more complex due to the unpredictable and rapidly changing nature of cloud workloads. Conventional static or heuristic scheduling approaches often fail to adapt in real time, leading to issues such as resource underutilization or overloaded nodes. [2]

Recent innovations have increasingly turned to machine learning (ML) and deep learning (DL) to design adaptive, intelligent load balancing strategies capable of learning from historical and real-time operational data. These techniques enable predictive scheduling by forecasting task demand and resource availability, thereby dynamically optimizing workload distribution , . Hybrid models that integrate ML classifiers such as *Random Forest* and *XGBoost* with optimization algorithms like *Particle Swarm Optimization (PSO)* and DL models such as *Artificial Neural Networks (ANN)* or *Convolutional Neural Networks (CNN)* have demonstrated improved accuracy, resource utilization, and throughput over traditional methods . [3]

By incorporating workload insights from large-scale public traces such as the Google Cluster dataset , the approach leverages feature importance analysis to identify dominant factors—such as vertical scaling configuration, scheduler type, CPU resource requests, and memory usage patterns—that influence optimal load balancing decisions. The aim is to develop an intelligent, scalable solution that enhances response time, resource utilization, and adaptability in dynamic cloud environments. [4]

II. LITERATURE REVIEW

A. Traditional Load Balancing and the Impact of ML

Traditional load balancing methods are rule-based techniques designed to distribute workloads evenly across multiple servers or resources, ensuring system stability and preventing individual nodes from becoming bottlenecks. These approaches rely on predefined heuristics rather than adaptive intelligence. Their main advantages are simplicity, low computational overhead, and ease of implementation, which made them widely adopted in early distributed and cloud systems. However, their limitations lie in their static nature and inability to account for dynamic factors such as varying task sizes, heterogeneous server capacities, and unpredictable workload fluctuations. [5]

1. Round Robin (RR): Tasks are distributed sequentially among servers in a circular fashion. It ensures simplicity and fairness but ignores the current state of resources.
2. Weighted Round Robin (WRR): This method assigns weights to servers based on their processing capacities, enabling higher-capacity servers to handle more tasks. However, it remains static and cannot adapt to changing workloads.
3. Least Connections (LC): Incoming requests are directed to the server with the fewest active connections. While more dynamic than RR, it does not account for the nature or intensity of individual tasks.
4. Random Allocation: Requests are assigned randomly across servers. Its low computational cost makes it efficient but at the expense of load-awareness and system performance optimization.
5. IP Hashing: Server assignment is based on a client's IP address, ensuring session persistence but lacking adaptability to real-time load variations.

Machine Learning enhances load balancing by adding adaptability and intelligence beyond the static nature of traditional methods. Predictive models, such as regression and recurrent neural networks, can forecast workload trends and enable proactive resource allocation. Reinforcement learning further optimizes task distribution by continuously adapting to system feedback, while unsupervised methods like clustering and anomaly detection help identify traffic spikes or failures for timely migration.

ML also supports multi-objective optimization by balancing performance with factors such as latency, throughput, and energy efficiency—crucial in cloud and edge environments. Deep learning models, in particular, can process diverse metrics to make fine-grained, QoS-aware allocation decisions in real time. Thus, ML transforms load balancing from simple rule-based schemes into intelligent, dynamic systems capable of managing complex and large-scale infrastructures. [6]

We aim to divide the working of our Smart Load Balancer for Cloud Computing Using ML into two phases:

1. Task Prediction — classifying tasks based on attributes such as priority, size, and energy requirements using ML models like XGBoost and Random Forest;
2. Resource Allocation — predicting optimal VM assignment using ML or DL models, evaluating key parameters such as CPU availability, memory usage, and response time and determining the CPU and memory to be allotted to the given job.

This research primarily focusses on the second phase of resource allocation, particularly CPU and memory.

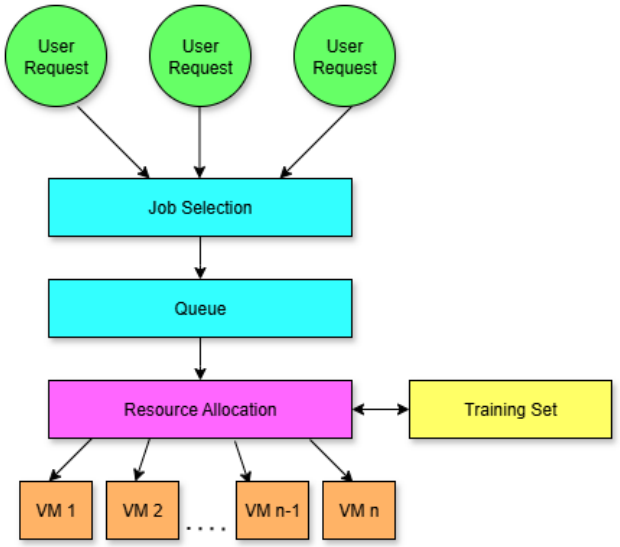

Fig 1: Overall Architecture for Resource Prediction

B. Related Work

Chawla [7] formulates cloud load balancing as a Markov Decision Process and applies tabular Q-learning to select actions from live system states, explicitly targeting nonstationary traffic conditions that degrade classical policies. The study reports approximately 20% reduction in response time and improved server utilization, consistently outperforming Round Robin and Least Connections under dynamic workloads. Owusu and collaborators [8] target SDN control-plane load balancing by combining Temporal Fusion Transformers for traffic forecasting with a Deep Q-Network for controller assignment. The integrated prediction–control design achieves approximately 30% higher throughput and 25% lower packet loss with fast convergence, demonstrating benefits of pairing forecasting with RL.

Zhu and coauthors [9] introduce DLB, a deep learning–based load balancer that predicts per-server load to replace static hashing, directly tackling workload skew and tail latency. The method delivers around 2× improvement in workload fairness and reduces tail latency by 15–20% in skew-heavy scenarios, highlighting the value of learned dispatch over fixed policies. [10] proposes an optimized RL-based pipeline that couples Q-learning–driven VM clustering with multi-objective task scheduling across latency and energy objectives. The approach jointly improves scalability and SLA adherence, achieving about 35% energy-efficiency gains with fewer SLA violations while maintaining robust performance on large workloads.

[11] advances proactive reliability by using ML to predict overloads and failures, enabling preemptive traffic rerouting before QoS degradation. The system reduces crash risk by about 40% and rebalancing costs by 30%, while maintaining stable performance during sudden demand spikes. Shahakar and colleagues address heterogeneous, geo-distributed cloud environments using RL frameworks (e.g., Least-Load RL and Booster RL) that adaptively select task paths across diverse nodes. Empirical results show roughly 30% reductions in task execution time, reduced resource idling, and resilience to node failures, underscoring applicability to distributed settings. [12]

C. Research Gaps

This paper addresses smart load balancing strategies in cloud computing, an area that has gotten vast research attention and great improvement. Of particular note is the application of quantum computing to workload prediction as a major milestone in the area [14]. Most current research focuses on independent facets of load balancing. For instance, some solution considers only network traffic [15], while others database query management [16] and yet others, only data center workloads. There remains a huge need for combined models that may consider workload distribution, VM allocation, and network load simultaneously.

Fault tolerance is another serious gap. Load balancers will most likely be a point of failure, but there has been less focus on redundancy and fault-tolerant distribution systems. Data center geographical distribution similarly brings along the issue of network delay, which is most likely to be ignored. The integration of edge computing Fig 1 Overall Architecture for Resource Prediction could mitigate these issues by reducing costly data transfers and VM migrations across long distances. Energy efficiency is also underexplored. Current load balancing techniques generally neglect power conservation, despite its importance in sustainable cloud operations. Future approaches should incorporate energy-aware mechanisms that balance performance with reduced power consumption.

III. METHODOLOGY

A. Description of Dataset

The Google cluster includes a set of eight clusters located in North America, Europe, and Asia. Google recorded the trace data over the entire month of May 2019. A cluster is made up of many cells, which are groups of machines connected by network switches and managed by the same cluster management system. Every cluster in this dataset contains approximately 12,000 machines with varying degrees of heterogeneity in their micro-architecture and chipset versions. These clusters support job assignments. Each job consists of one or more tasks, and alloc sets, which represent resource reservations for later job scheduling.

To preserve confidentiality, Google researchers obfuscated personally identifiable information and transformed specific attributes to conceal system architecture information. The overall structure of the Google cluster trace dataset can be described in four major parts: Machine Events, Instance Events, Collection Events and Resource Usage. A machine event is defined as the addition, removal, or update of a machine. Most clusters have a fairly homogeneous distribution of machine events. A collection event denotes one of eleven possible occurrences in the lifecycle of a task or alloc set.[17]

B. Data Preprocessing

1) Data Ingestion and Inspection: The analysis begins with the ingestion of the borg_traces_data.csv file, followed by schema validation using exploratory commands such as head() and shape(). This preliminary inspection step, commonly employed in Borg dataset studies, ensures that the dataset is properly structured before preprocessing and feature transformation.

2) Feature Engineering: Feature engineering is aligned with the unique structure of the Borg traces and prior analyses in the literature. First, a temporal feature is derived by computing the execution duration as the difference between *end_time* and *start_time*. This measure of runtime is frequently adopted in scheduling and workload characterization studies. Next, nested fields such as *resource_request* and *average_usage*, which are typically stored in serialized dictionary-like formats, are parsed using literal evaluation. This parsing produces ML-ready tabular features, including *cpu_resource_request*, *memory_resource_request*, *cpu_average_usage*, and *memory_average_usage*, thereby making the dataset consistent with repository guidelines.

Heavy-tailed distributions of CPU and memory usage are addressed by applying a logarithmic transformation (log1p) to mitigate skewness. Subsequently, these transformed variables are discretized into seven quantile-based bins, ranging from *Very Low* to *Very High*, using qcut. The resulting categorical features, *cpu_usage_log_categorical* and *memory_usage_log_categorical*, provide interpretable states for use in load-balancing controllers. Categorical fields such as *event* are label-encoded into numeric codes, ensuring compatibility with ML pipelines commonly applied to Google cluster traces. Finally, high-cardinality identifiers and redundant fields, including *collection_id*, *user*, *machine_id*, raw distribution objects, and sampling/time metadata, are removed. This column selection and cleanup step reduces noise, prevents potential information leakage, and follows best practices from prior Borg trace studies.

3) Handling Missing Values: Missing values are imputed using robust strategies tailored to the heterogeneity of operational traces. Categorical fields, such as *scheduler* and *vertical_scaling*, are imputed with their mode, while skewed numerical features, including *cpu_resource_request* and *memory_resource_request*, are filled with their median values. This approach balances resilience to outliers with maintaining representative data distributions.*Data Ingestion and Inspection:* The analysis begins with the ingestion of the borg_traces_data.csv file, followed by schema validation using exploratory commands such as head() and shape(). This preliminary inspection step, commonly employed in Borg dataset studies, ensures that the dataset is properly structured before preprocessing and feature transformation.

4) Normalization: To stabilize the training of artificial neural networks on mixed-scale tabular data, all numeric predictors are standardized using a StandardScaler. The scaler is fit on the training subset and subsequently applied to the test data, ensuring consistent scaling across evaluation phases.

5) Train-Test Split: The final dataset is organized into predictors and targets. The feature matrix **X** consists of all engineered predictors, excluding categorical target variables. The label vector **y** is derived from *cpu_usage_log_categorical* and *memory_usage_log_categorical*, which is label-encoded into integers and then transformed into one-hot vectors for use in softmax classification. The dataset is split into training and testing sets using an 80/20 ratio, stratified by categorical labels. This stratification preserves class distributions across splits, ensuring reliable evaluation consistent with established ML-aided load balancing assessments.

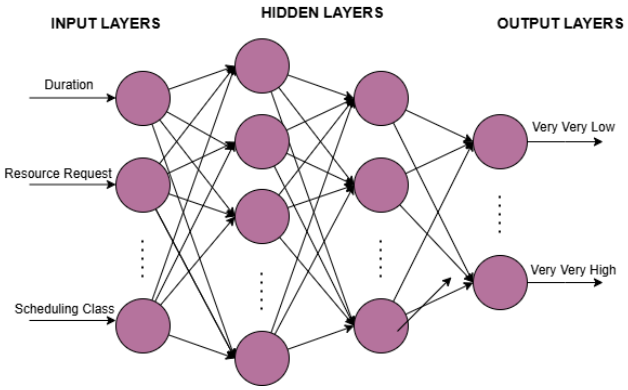

Fig 2: ANN Architecture

C. ML Models

1) ANN: The modeling objective centers on a seven-class categorical representation of CPU utilization. These classes are obtained through quantile discretization of the log-transformed average CPU usage, a strategy designed to address the heavy-tailed distribution typical of cluster workloads. This approach yields robust and interpretable workload states that can be leveraged by load-balancing controllers. In parallel, a categorical target for memory usage is derived for comparative analysis; however, CPU categories are designated as the primary labels for model training and evaluation. This decision aligns with prior load balancing research, which emphasizes forecasting CPU utilization states to guide workload distribution strategies.

The artificial neural network (ANN) is designed for tabular feature learning and adopts a feed-forward architecture consisting of four dense layers. The input layer connects to a Dense(128) unit with ReLU activation, followed by a Dropout layer (0.2). This is succeeded by Dense(64, ReLU), another Dropout(0.2), and a Dense(32, ReLU) layer. The final output layer is a Dense(7) with softmax activation, corresponding to the seven workload categories. Model optimization is performed using the categorical cross-entropy loss function in conjunction with the Adam optimizer. To prevent overfitting and adapt to the non-stationary nature of cluster workloads, two regularization strategies are applied: EarlyStopping with patience set to 10 (and best weight restoration enabled) and ReduceLROnPlateau with a factor of 0.5 and patience of 3. Training is conducted for up to 100 epochs with a batch size of 256. The dataset is partitioned using an 80/20 train-test

split, stratified on the target distribution to preserve class balance, which is considered best practice in categorical workload state modeling for load balancing studies.

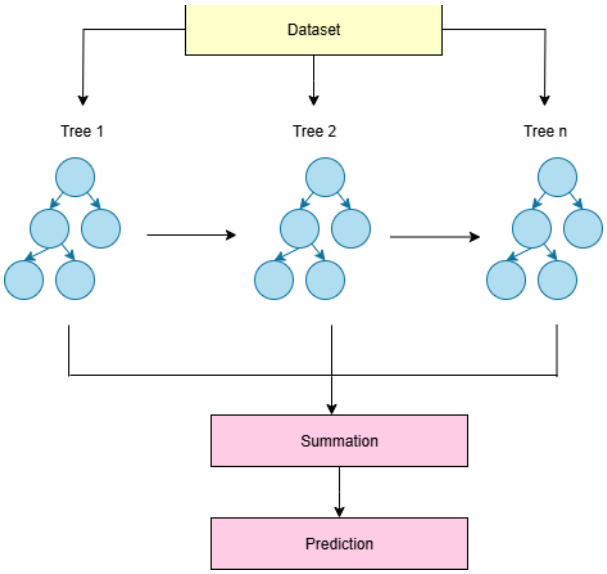

Fig 3: XGBoost Architecture

2) XGBoost: The XGBoost model shown is an implementation of a multi-class classification task using the XGBClassifier from the xgboost library. It is configured with the objective function set to 'multi:softprob', which outputs probability distributions over multiple classes rather than hard predictions. The number of classes is dynamically determined from the target variable y, and the evaluation metric used is 'mlogloss' (multi-class log loss), which is well-suited for probabilistic classification. The model is tuned with 200 boosting estimators and a learning rate of 0.1, balancing performance and training stability. To control complexity, the

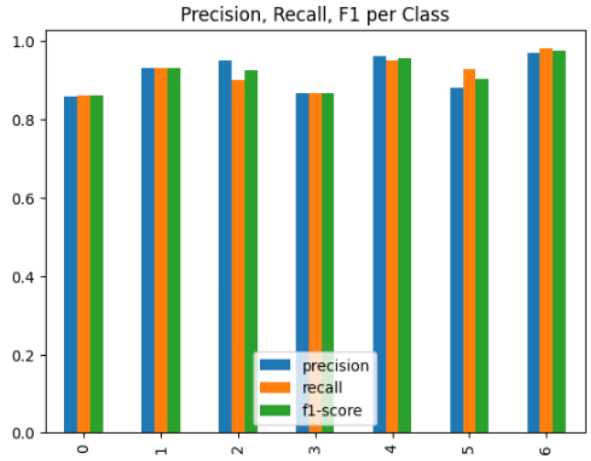

Fig 6: Class-wise performance metrics for CPU allocation

maximum tree depth is restricted to 6, while subsample=0.8 and colsample_bytree=0.8 introduce randomness in row and feature sampling, reducing overfitting. The random_state=42 ensures reproducibility of results. Overall, this configuration reflects a robust setup for handling high-dimensional, multi-class problems where probabilistic outputs are useful for downstream decision-making.

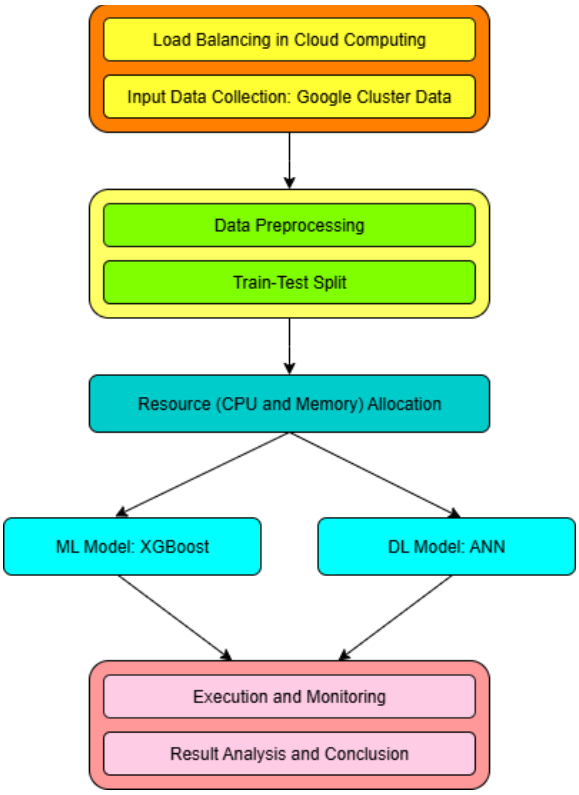

Fig 4: Work Flow

D. *Performance Evaluation*

The evaluation of the model is based on multiple metrics to ensure both accuracy and interpretability. The primary metric is test accuracy, chosen to align with the reporting standards of the IJST study on classification tasks. For a more

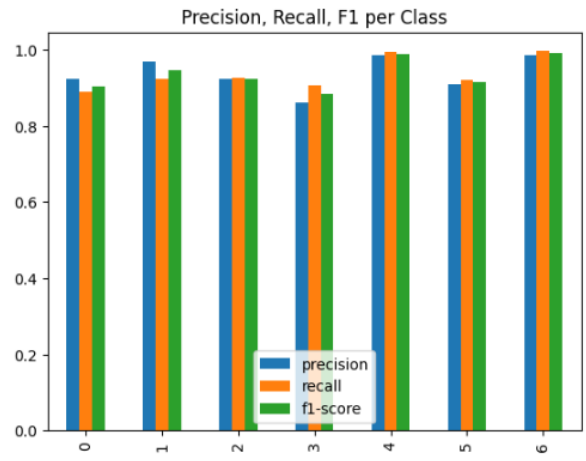

Fig 8: Class-wise performance metrics for memory allocation

detailed view, precision, recall, and F1-score are calculated for each class using a classification report, while a confusion matrix is employed to analyze error structures—a common practice in machine learning evaluations for load balancing. During inference, the predicted class is obtained by taking the argmax over the softmax probabilities, which is then

compared against the true labels to compute per-class as well as macro- and micro-averaged metrics.

These results not only validate the model's predictive power but also align its interpretation with the IJST study, where discrete workload categories act as control signals for schedulers. By focusing on ANN-driven categorization while maintaining modularity, the work preserves clarity and opens pathways for integration with hybrid systems such as RF–PSO–CNN. Furthermore, the use of publicly available Google cluster traces along with standard evaluation metrics enhances comparability with prior literature and ensures reproducibility, in line with recommendations from the Google cluster data repository and subsequent analytical studies. The findings shown in the images here are based on XGBoost model.

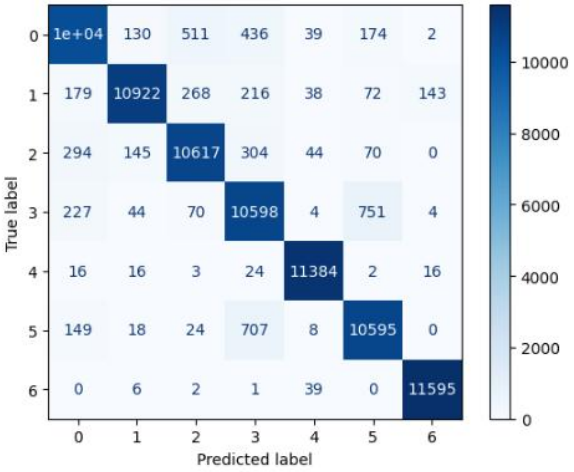

Fig 7: Confusion Matrix for Memory Allocation

IV. RESULTS AND DISCUSSION

We compared the proposed load balancing system in following two settings with the target variables is CPU usage and memory usage: However, the results showed that XGB behaved better both in terms of prediction and determination of memory utilization using it (Accuracy: 93.7%, Precision: 93.66%, F1-score: 0.9368) yet not for CPU utilization (Accuracy: 92.0%, Precision: 91.66%, F1-score: 0.9196). This means that XGB captured memory utilization patterns better, as the dataset has a higher correlation of memory-related features. The slight (1.7%) improvements in accuracy and F1-score show that memory consumption is a non-trivial concern of our model.

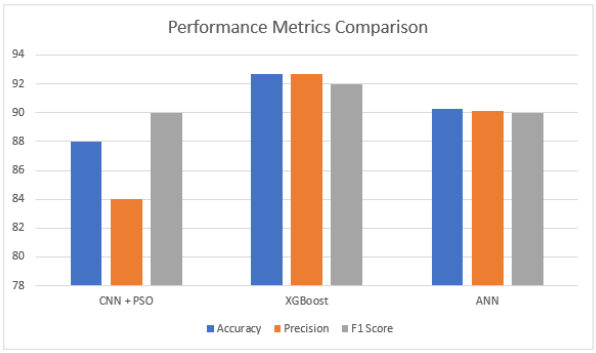

Fig 9: Comparing Performance Metrics

This was also reinforced and validated by the confusion matrices that were provided. In Fig 5, we find that most predictions of both XGB and ANN were on the right track, as seen by the high density along the diagonal. ANN did have some off-diagonal terms higher than XGB, especially for CPU usage, showing that there were more frequent misclassifications. In contrast, XGB demonstrated more diagonal dominance in the memory prediction and indicated a higher discriminative accuracy. Our observations validate the quantitive results and we also offer a more general insight that designing models for memory usage provides a larger leverage than designing them for CPU usage in load prediction.

The ANN model too showed the same but not so consistent trend. Prediction of memory usage (Accuracy: 91.8%, F1-score: 0.9177) was superior compared to CPU usage prediction (Accuracy: 88.2%, F1-score: 0.8817) by roughly 3.6%. Such a disparity in performance, greater than that recorded with XGB, reflects the observation that the neural network performed poorly to infer CPU usage trends but learned better under training on memory use.

In comparison between models, XGB performed slightly better than ANN for both target variables. In CPU usage, XGB was slightly better than ANN by roughly 3.8% accuracy, and in memory usage, it was slightly better than ANN by roughly 2.0%. This directly mirrors the superiority of XGB in system load dynamics modeling, with memory usage prediction being the best performing overall across all experiments.
configurations.
Overall, the results reveal two key findings:

1. Memory usage is a more predictive target variable than CPU usage for both XGB and ANN, with consistently higher performance metrics.
2. XGB is the stronger model compared to ANN, offering more stable and accurate predictions, particularly for memory usage where it reached the best observed performance.

These results highlight that, in real load balancing situations, employing XGB to model memory consumption gives the most stable and precise results.

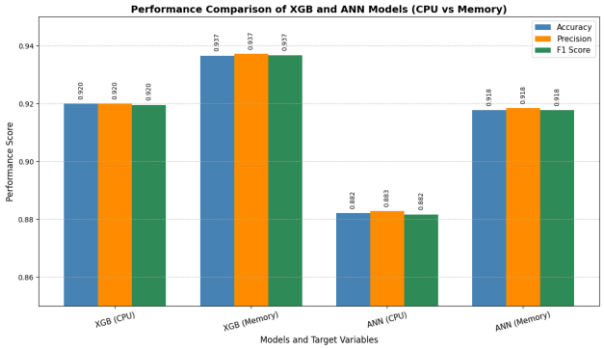

Fig 10: Comparison across both target variable

The feature importance analysis of the XGB models provides valuable insights into which system attributes contribute most significantly to accurate load prediction. For the CPU usage prediction model, features directly related to processor performance, such as instruction throughput, task scheduling frequency, and thread utilization, were observed to dominate the importance ranking. This indicates that CPU-related metrics are the strongest predictors of CPU utilization, aligning with the intuitive expectation that processor-intensive variables govern CPU load behavior.

In contrast, the memory usage prediction model highlighted features such as available RAM, memory allocation patterns, and cache utilization as the most influential predictors. Interestingly, secondary features such as I/O wait times and disk activity also appeared in the importance ranking, suggesting an interdependence between memory utilization and storage operations.

V. CONCLUSION AND FUTURE DIRECTIONS

Machine learning has become an effective way to improve load balancing in cloud computing, moving beyond traditional heuristic-based methods. Instead of sticking to fixed rules like Round Robin or Least Connections, ML models can learn from historical workload data, patterns in resource use, and system behavior to make better allocation choices. By looking at factors like CPU and memory use, request arrival rates, and task execution times, ML systems can predict future demand and distribute workloads proactively. This helps minimize latency, avoid overload, and improve resource use.

Supervised learning methods such as decision trees, random forests, and XGBoost are used to classify or predict workload states, allowing for dynamic allocation based on resource availability. Deep learning models, including artificial neural networks and recurrent networks, recognize

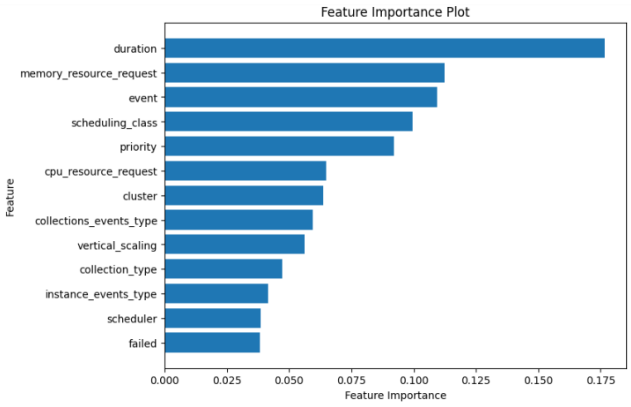

Fig 12: Feature Importance for Memory Allocation

patterns over time in workload data. They help predict usage trends for more flexible scheduling. Reinforcement learning is also used, where agents learn the best task placements through feedback from system performance.

Compared to traditional methods, ML-based load balancing offers flexibility in changing and varied cloud environments. It cuts down response times, improves throughput, and increases fault tolerance by considering the complex relationships between tasks and resources. As cloud workloads continue to expand in size and variety, ML-driven load balancing is seen as a crucial part of creating intelligent, self-optimizing cloud infrastructures.

REFERENCES

[1] A10, "What is a Load Balancer and How Does Load Balancing Work?," *A10 Networks*, 2022.

[2] J. Muchori and P. M. Mwangi, "Machine Learning Load Balancing Techniques in Cloud Computing: A Review," *Int. J. of Computer Applications Technology and Research*, vol. 11, no. 6, pp. 179-186, 2022.

[3] C. Thilagavathy, "Leveraging Machine and Deep Learning Models for Load Balancing Strategies in Cloud Computing," *Indian Journal of Science and Technology*, vol. 17, no. 45, pp. 4722-4731, Dec. 2024.

[4] F. H. Bappy, T. Islam, T. S. Zaman, R. Hasan, and C. Caicedo, "A Deep Dive into the Google Cluster Workload Traces: Analyzing the Application Failure Characteristics and User Behaviors," in *Proc. 10th Int. Conf. on Future Internet of Things and Cloud (FiCloud)*, 2023, pp. 103-108.

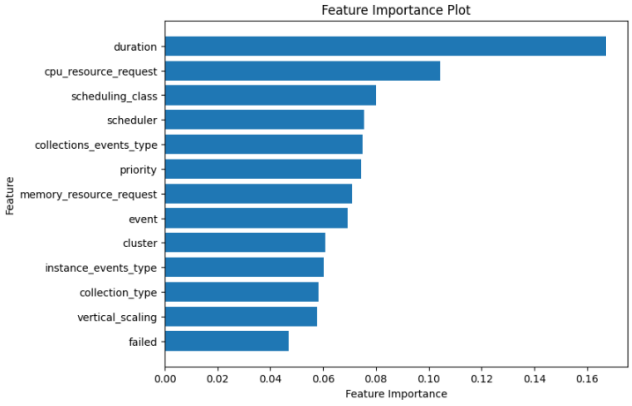

Fig 11: Feature Importance for CPU Allocation

[5] S. R. Biradar and A. R. Sajjan, “Load balancing and its algorithms in cloud computing: A survey,” *International Journal of Computer Applications*, vol. 175, no. 23, pp. 12–17, 2017.

[6] C. Yao, J. Wang, J. Su, Y. Liu, J. Ren, and Y. Zhang, “Towards intelligent load balancing in data centers,” *arXiv preprint arXiv:2110.15788*, 2021.

[7] A. Chawla, “Reinforcement Learning-Based Adaptive Load Balancing for Dynamic Cloud Environments,” 2024.

[8] A. Owusu, et al., “Transformer-Based Deep Q-Learning for Load Balancing in SDNs,” 2025.

[9] H. Zhu, et al., “DLB: Deep Learning-Based Load Balancing,” *Proc. of [Conference/Journal]*, 2019.

[10] “Optimized RL-Based Clustering with Multi-Objective Task Scheduling,” *MDPI*, 2024.

[11] “High-Performance Cloud Load Balancing Using ML,” *Elsevier*, 2022.

[12] V. Shahakar, et al., “RL for Load Balancing in Distributed Clouds,” 2023.

[13] X. Zhu, Q. Zhang, T. Cheng, L. Liu, W. Zhou, and J. He, “DLB: Deep Learning Based Load Balancing,” *arXiv preprint arXiv:1910.08494*, October 2019.

[14] Ranesh Kumar Naha and Mohamed Othman. 2016. Cost-aware service brokering and performance sentient load balancing algorithms in the cloud. Journal of Network and Computer Applications 75 (2016), 47–57

[15] Nusrat Pasha, Amit Agarwal, and Ravi Rastogi. 2014. Round robin approach for VM load balancing algorithm in cloud computing environment. International Journal of Advanced Research in Computer Science and Software Engineering 4, 5 (2014), 34–39.

[16] Awatif Ragmania, Amina Elomria, Noreddine Abghoura, Khalid Moussaida, Mohammed Rida. 2019. An improved Hybrid Fuzzy-Ant Colony Algorithm Applied to Load Balancing in Cloud Computing Environment. The 10th International Conference on Ambient Systems, Networks and Technologies (ANT).

[17] F. H. Bappy, T. Islam, T. S. Zaman, R. Hasan, and C. Caicedo, “A Deep Dive into the Google Cluster Workload Traces: Analyzing the Application Failure Characteristics and User Behaviors,” *arXiv preprint arXiv:2308.02358*, Aug. 2023.