DOI: 10.21917/ijsc.2026.0577

DEEP LEARNING AND MACHINE LEARNING BASED HYBRID OPTIMIZATION FRAMEWORK FOR ENHANCED WEBSITE PERFORMANCE TUNING

P.C. Geethu[1], Deena Jose[2], N.M. Shejina[3] and M.B. Meethu[4]

[1,2]*Department of Computer Science and Engineering, Federal Institute of Science and Technology, India*

[3,4]*Department of Computer Science and Engineering, IES College of Engineering, India*

Abstract

The performance of modern websites has become a critical factor in determining user engagement, service reliability, and computational efficiency. Background studies have shown that increasing traffic load, dynamic content delivery, and heterogeneous user behavior have significantly reduced responsiveness and scalability of web systems. The problem of inefficient resource utilization and delayed response time has remained persistent, especially in cloud-hosted and data-intensive web environments. To address this issue, a Hybrid Deep Reinforcement Learning and Gradient Boosted Optimization (HDRL-GBO) method has been proposed to improve website performance through adaptive resource allocation and predictive load balancing. The proposed HDRL-GBO method integrated deep reinforcement learning for real-time decision optimization and gradient boosted machine learning for performance prediction. The reinforcement component dynamically adjusted caching strategies, request routing, and server allocation, while the boosting model estimated workload patterns and latency variations. The system has been trained using historical traffic logs and real-time simulation data. The proposed HDRL-GBO framework has significantly reduced average latency to 85–140 ms, compared to 105–240 ms in baseline methods. Throughput has improved to 520–600 req/s, while existing methods achieve 340–470 req/s. CPU utilization has reached 92% balanced efficiency, outperforming EHGA at 90% and PSO at 88%. These results confirm that the proposed system has improved performance stability, scalability, and response efficiency under dynamic web traffic conditions.

Keywords:

Website Optimization, Deep Learning, Machine Learning, Reinforcement Learning, Web Performance

1. INTRODUCTION

The evolution of web technologies has significantly transformed digital service delivery systems. Modern websites are no longer static entities but highly dynamic platforms that serve real-time content, multimedia data, and interactive services. Background research has shown that increasing user demand and distributed cloud environments have created substantial pressure on web servers and content delivery networks [1]. These systems often experience unpredictable traffic fluctuations that directly affect latency and resource utilization. As web applications become more complex, performance optimization has become an essential research area in both academic and industrial domains [2]. In contemporary distributed systems, scalability and responsiveness are primary performance indicators. However, traditional optimization techniques often rely on static rules or threshold-based configurations, which are insufficient in handling dynamic workloads. Studies have shown that such conventional methods fail to adapt efficiently to sudden traffic spikes and heterogeneous request patterns [3]. Consequently, adaptive and intelligent optimization approaches have gained attention for improving real-time decision-making in web environments.

Despite several advancements, multiple challenges still exist in optimizing website performance. One major challenge is the unpredictability of user traffic patterns, which makes it difficult to allocate computational resources efficiently. Another challenge is the latency overhead introduced by centralized optimization mechanisms, which often delays decision execution [4]. Additionally, balancing load distribution across geographically distributed servers remains complex due to network variability and inconsistent data propagation delays [5].

The core problem addressed in this study is the inefficiency of existing web optimization frameworks in handling dynamic, large-scale traffic environments [6]. These systems often lack predictive intelligence and fail to integrate learning-based decision models that can adapt over time. As a result, performance degradation occurs during peak load conditions, affecting user experience and system reliability.

To address these limitations, the primary objective of this research is to design an intelligent optimization framework that integrates deep learning and machine learning techniques for enhancing website performance. The study aims to develop a predictive and adaptive system capable of optimizing caching, load balancing, and request routing in real time. Another objective is to reduce latency and improve throughput by utilizing learned behavioral patterns from historical and live traffic data.

The novelty of the proposed approach lies in the integration of Deep Reinforcement Learning (DRL) with Gradient Boosted Machine Learning models. The DRL component continuously interacts with the web environment and learns optimal policies for resource allocation, while the boosting model provides accurate performance forecasting that supports proactive optimization decisions. This dual-layer architecture ensures both predictive intelligence and adaptive control within the system.

The contributions of this study can be summarized in two key aspects. First, it introduces a hybrid HDRL-GBO framework that combines reinforcement learning with supervised boosting techniques for web optimization. Second, it demonstrates a scalable architecture that improves response time, reduces latency, and enhances throughput under variable traffic conditions. The framework provides a practical and intelligent solution for modern web infrastructures that require continuous adaptation and high performance.

2. RELATED WORKS

Several studies have been conducted to improve website performance using optimization, machine learning, and intelligent

resource management techniques. Earlier research focused primarily on rule-based caching mechanisms and static load balancing strategies. These methods have shown limited adaptability in dynamic environments where traffic patterns frequently change [7]. Researchers have explored predictive modeling techniques using traditional machine learning algorithms for web performance enhancement. Regression-based models and decision tree approaches have been applied to estimate server load and response time. These methods have demonstrated moderate success in controlled environments but have struggled in large-scale distributed systems due to limited generalization capability [8]. Deep learning techniques have also been introduced for traffic prediction and resource optimization. Neural network models have been used to capture nonlinear relationships in web request patterns. These approaches have improved prediction accuracy compared to classical methods; however, they often require large training datasets and high computational resources [9]. Reinforcement learning-based optimization strategies have gained attention in recent years. Studies have shown that reinforcement learning agents can learn optimal caching and routing policies by interacting with web environments. These methods have provided adaptive solutions, but convergence time and training instability remain significant limitations [10]. Hybrid approaches combining machine learning and optimization algorithms have been proposed to overcome individual model limitations. For example, ensemble learning techniques have been used to improve prediction robustness in web traffic forecasting. These models have enhanced accuracy but still lack real-time adaptability in changing environments [11]. Some studies have integrated cloud computing frameworks with intelligent load balancing mechanisms. These systems distribute traffic dynamically across multiple servers based on resource availability. Although these approaches have improved scalability, they often rely on predefined heuristics that reduce flexibility [12]. Recent advancements have introduced deep reinforcement learning models for adaptive web resource allocation. These models continuously learn from environmental feedback and adjust system parameters accordingly. However, they often face challenges related to exploration-exploitation balance and computational overhead [13]. Gradient boosting techniques have been widely applied in performance prediction tasks due to their high accuracy and robustness. These models have effectively captured complex relationships in web performance metrics. Nevertheless, they are primarily static in nature and lack real-time adaptability [14]. These frameworks combine predictive analytics with adaptive control mechanisms. Although they represent a significant improvement over traditional methods, they often lack seamless integration between prediction and decision-making components [15].

3. PROPOSED METHOD: HYBRID DEEP REINFORCEMENT LEARNING AND GRADIENT BOOSTED OPTIMIZATION (HDRL-GBO)

The proposed HDRL-GBO method represents a hybrid optimization framework that integrates deep reinforcement learning with gradient boosted machine learning to improve website performance in dynamic environments. The method operates by learning adaptive control policies for resource allocation while simultaneously predicting system load and latency behavior. The reinforcement learning component interacts with the web environment and continuously updates decision policies based on reward feedback derived from latency reduction and throughput improvement. In parallel, the gradient boosted model estimates future traffic intensity and response behavior using historical web logs. The integration of both models enables proactive and reactive optimization within a unified architecture that improves scalability, reduces response delay, and enhances resource utilization across distributed web servers.

3.1 SYSTEM ARCHITECTURE AND PROBLEM FORMULATION

The system architecture defines a web environment that consists of a set of servers, incoming requests, caching layers, and a controller module that performs optimization. Let the web system be represented as a tuple: $\mathrm{W} = (S, R, C, D, \Pi)$, where S denotes the set of servers, R denotes incoming request streams, C denotes caching states, D denotes observed performance metrics, and Π denotes policy space. The objective function of the system is defined as:

$$\min_{\pi \in \Pi} \mathrm{J}\ (\pi) = \mathrm{E}\left[\alpha \cdot T_{lat} + \beta \cdot U_{cpu} + \gamma \cdot L_{queue} \right] \quad (1)$$

where T_{lat} denotes latency, U_{cpu} denotes CPU utilization, and L_q denotes queue length. The parameters α, β, and γ define trade-off weights that regulate optimization priority. The system state at time $t$is expressed as: $s_t = \{r_t, c_t, m_t, n_t\}$, where r_t represents request rate, c_t represents cache state, m_t represents memory usage, and n_t represents network delay. The action space is defined as:

$$a_t \in \{cache\_update, load\_shift, route\_change\} \quad (2)$$

The reinforcement agent learns an optimal policy $\pi(a_t|s_t)$ that minimizes system cost while maximizing throughput.

3.2 GRADIENT BOOSTED PREDICTION MODULE

The gradient boosted model operates as a predictive layer that estimates future system load and latency. It constructs an additive model of weak learners:

$$\hat{y}_t = \sum_{k=1}^{K} f_k(x_t) \quad (3)$$

where f_k denotes a decision tree learner and x_t represents feature vectors extracted from web logs. The model minimizes a differentiable loss function:

$$\mathrm{L} = \sum_{t=1}^{N} \ell(y_t, \hat{y}_t) + \sum_{k=1}^{K} \Omega(f_k) \quad (4)$$

where l denotes prediction error and Ω denotes regularization term. Each boosting iteration updates the model using gradient descent in function space:

$$f_k(x) = f_{k-1}(x) - \eta \frac{\partial \ell(y, \hat{y})}{\partial \hat{y}} \quad (5)$$

where η represents learning rate.

The predicted output vector includes: $\hat{y}_t = [\hat{T}_{lat}, \hat{R}_{load}, \hat{B}_{band}]$, where $\hat{T}_{lat}$ denotes predicted latency, $\hat{R}_{load}$ denotes request load, and $\hat{B}_{band}$ denotes bandwidth consumption. The predicted values are passed to the reinforcement learning module for proactive optimization decisions.

3.3 DEEP REINFORCEMENT LEARNING OPTIMIZATION MODULE

The reinforcement learning component models the web optimization problem as a Markov Decision Process (MDP): $\mathrm{M} = (S, A, P, R, \gamma)$, where *S* is state space, *A* is action space, *P* is transition probability, *R* is reward function, and γ is discount factor. The agent learns a policy $\pi(a|s)$ that maximizes expected cumulative reward:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{6}$$

The Q-value function is defined as: $Q(s_t, a_t) = \mathrm{E}[G_t \mid s_t, a_t]$. The deep Q-network approximates Q-values using neural networks: $Q(s_t, a_t; \theta) \approx Q^*(s_t, a_t)$. The parameter update rule follows:

$$\theta = \theta + \alpha \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) - Q(s_t, a_t; \theta) \right) \tag{7}$$

The reward function is designed as: $r_t = -\lambda_1 T_{lat} - \lambda_2 U_{cpu} + \lambda_3 T_{thr}$, where T_{thr} denotes throughput. The agent continuously updates policy based on feedback and predicted system states from the gradient boosting module.

3.4 HYBRID DECISION FUSION MECHANISM

The hybrid module integrates predictions and reinforcement decisions into a unified optimization output. Let the predicted state be: $\hat{s}_t = f_{GBM}(x_t)$. The reinforcement policy is conditioned on predicted state: $a_t = \pi(a \mid s_t, \hat{s}_t)$. A weighted fusion function combines both outputs: $F_t = \omega_1 Q(s_t, a_t) + \omega_2 \hat{y}_t$. where ω_1 and ω_2 represent adaptive weights. The final decision rule is: $a_t^* = \arg\max_{a \in A} F_t$. This fusion ensures that predictive analytics guide reinforcement learning decisions, reducing exploration instability. The optimization objective becomes:

$$\min \mathrm{E}\left[\mathrm{L}_{lat} + \mathrm{L}_{load} + \mathrm{L}_{energy} \right] \tag{8}$$

where each loss term is dynamically adjusted.

3.5 ADAPTIVE CACHE AND LOAD BALANCING STRATEGY

The caching mechanism operates using a probabilistic update rule:

$$P(c_i) = \frac{e^{\beta Q(c_i)}}{\sum_j e^{\beta Q(c_j)}} \tag{9}$$

where c_i denotes a cache entry. The cache replacement follows:

$$c_{new} = \arg\min_{c_i} \left(\theta_1 T_{miss} + \theta_2 F_{req} \right) \tag{10}$$

where T_{miss} denotes cache miss penalty and F_{req} denotes frequency score. Load balancing is modeled as:

$$L_s = \frac{\sum R_s}{C_s + \hat{o}} \tag{11}$$

where L_s is server load, R_s is request count, and C_s is capacity. The system redistributes load using: $\Delta L = L_s - \bar{L}$. Servers with high ΔL are penalized in routing decisions.

The convergence of the hybrid system is analyzed using expected loss reduction:

$$\mathrm{E}[\mathrm{L}_{t+1}] \le \mathrm{E}[\mathrm{L}_t] - \delta \| \nabla \mathrm{L}_t \|^2 \tag{12}$$

where δ represents convergence rate constant. Stability is ensured through bounded reward function: $| r_t | \le R_{max}$. The system maintains equilibrium when: $\lim_{t \to \infty} \mathrm{E}[a_t] = a^*$, which indicates convergence to optimal policy. The final optimization objective that is combining the components of the entire system as $\mathrm{O} = \min\left(\alpha_1 T_{lat} + \alpha_2 L_{queue} + \alpha_3 E_{cons} \right)$, subject to constraints:

$$T_{lat} \le T_{max}, \quad U_{cpu} \le U_{max} \tag{13}$$

The system continuously adjusts parameters using gradient feedback: $\theta_{t+1} = \theta_t - \eta \nabla \mathrm{O}$. This ensures adaptive tuning across all layers.

4. DEEP REINFORCEMENT LEARNING OPTIMIZATION MODULE

The Deep Reinforcement Learning (DRL) Optimization Module forms the core adaptive intelligence of the proposed HDRL-GBO framework for website performance enhancement. This module models the web environment as a sequential decision-making system, where an autonomous agent continuously interacts with a dynamic server ecosystem. The primary aim of this module is to minimize latency, reduce queue congestion, and improve throughput through learned control policies that evolve over time. Unlike static optimization strategies, this module adapts to traffic fluctuations by learning optimal actions directly from environmental feedback. The module operates through a Markov Decision Process (MDP), neural approximation of value functions, policy optimization, and reward-driven learning. Each component contributes to a closed-loop adaptive system that refines decision-making in real time.

5. MARKOV DECISION PROCESS FORMULATION FOR WEB OPTIMIZATION

The web optimization environment is formally represented as a Markov Decision Process defined by: $\mathrm{M} = (S, A, P, R, \gamma)$, where, *S* represents the state space of the web system, *A* represents the action space available to the controller, $P(s_{t+1}|s_t,a_t)$ defines state transition probability, $R(s_t,a_t)$ defines reward function and $\gamma \in [0,1]$ defines discount factor. At any time instant *t*, the system state is defined as: $s_t = \{\lambda_t, \mu_t, q_t, c_t, b_t\}$, where, λ_t denotes incoming request arrival rate, μ_t denotes service rate of servers, q_t denotes queue length across servers, c_t denotes cache hit ratio and b_t denotes bandwidth utilization The action space is defined as:

$$a_t \in \{a^{cache}, a^{route}, a^{scale}, a^{balance}\} \quad (14)$$

where, a^{cache} represents cache update or replacement decision, a^{route} represents request routing decision, a^{scale} represents server scaling action and $a^{balance}$ represents load balancing adjustment. The transition function is expressed as:

$$P(s_{t+1} \mid s_t, a_t) = \Pr(s_{t+1} = f(s_t, a_t, \xi_t)) \quad (15)$$

where ξ_t represents stochastic network noise and external traffic variability. This formulation captures the dynamic and probabilistic nature of web systems, where identical actions may produce different outcomes due to fluctuating workloads.

5.1 STATE REPRESENTATION AND FEATURE ENCODING

The DRL agent requires a structured representation of the web environment. The state vector is encoded as: $s_t = [x_1^t, x_2^t, x_3^t, ..., x_n^t]$ where each feature corresponds to a measurable system metric. A normalized state representation is defined as:

$$\tilde{s}_t = \frac{s_t - \mu_s}{\sigma_s} \quad (16)$$

where μ_s and σ_s represent mean and standard deviation of observed system states. The feature transformation function is defined as:

$$\phi(s_t) = W_s s_t + b_s \quad (17)$$

where, W_s represents feature transformation weights and b_s represents bias vector. This transformation ensures that the neural network receives structured input for stable convergence. Additionally, temporal dependencies are modeled using a recurrent formulation: $h_t = f(h_{t-1}, s_t)$, where h_t represents hidden state capturing historical dependencies of traffic behavior.

5.2 REWARD FUNCTION DESIGN FOR WEB OPTIMIZATION

The reward function defines the optimization objective of the DRL agent. The system aims to minimize latency while maximizing throughput and resource efficiency. The reward is formulated as:

$$r_t = \alpha_1 R_{thr} - \alpha_2 T_{lat} - \alpha_3 Q_{len} - \alpha_4 E_{cpu} \quad (18)$$

where, R_{thr} represents throughput reward, T_{lat} represents response latency, Q_{len} represents queue length penalty, E_{cpu} represents energy consumption penalty and α_i represents weighting coefficients. A normalized reward function is also defined as:

$$\hat{r}_t = \frac{r_t - r_{min}}{r_{max} - r_{min}} \quad (19)$$

To ensure stability, bounded reward constraint is applied: $|r_t| \le R_{max}$. The cumulative return is expressed as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \quad (20)$$

This formulation ensures that the agent optimizes long-term performance rather than short-term gains.

5.3 Q-VALUE FUNCTION APPROXIMATION USING DEEP NEURAL NETWORK

The DRL module uses Q-learning to estimate the value of actions in each state. The Q-function is $Q(s_t, a_t) = \mathrm{E}[G_t \mid s_t, a_t]$, Since exact computation is infeasible, a deep neural network approximates the Q-function: $Q(s_t, a_t; \theta) \approx Q^*(s_t, a_t)$, where θ represents neural network parameters. The input to the network is state-action pair: $z_t = [s_t, a_t]$. The output is Q-value:

$$Q(z_t; \theta) = f_\theta(W_2 \sigma(W_1 z_t + b_1) + b_2) \quad (21)$$

where, W_1, W_2 are weight matrices, b_1, b_2 are bias vectors and $\sigma(\cdot)$ is activation function (ReLU or tanh)

The network is trained using loss function:

$$\mathrm{L}(\theta) = \mathrm{E}\left[\left(y_t - Q(s_t, a_t; \theta)\right)^2\right] \quad (22)$$

where target value is:

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) \quad (23)$$

and θ^- represents target network parameters.

5.4 POLICY LEARNING AND ACTION SELECTION MECHANISM

The DRL agent selects actions based on an epsilon-greedy policy:

$$a_t = \begin{cases} \text{random action}, & \text{with probability } \dot{o} \\ \arg\max_a Q(s_t, a; \theta), & \text{with probability} 1 - \dot{o} \end{cases} \quad (24)$$

The exploration parameter ϵ decays over time: $\dot{o}_t = \dot{o}_0 e^{-\kappa t}$. This ensures that the system explores initially, and it exploits the learned knowledge later. The optimal policy is defined as:

$$\pi^*(a \mid s) = \arg\max_a Q^*(s, a) \quad (25)$$

A soft policy variant is also defined using softmax distribution:

$$\pi(a \mid s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}} \quad (26)$$

where τ represents temperature parameter controlling the randomness.

5.5 TEMPORAL DIFFERENCE LEARNING AND PARAMETER UPDATE

The Q-network is updated using Temporal Difference (TD) learning. The TD error is defined as:

$$\delta_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta) \quad (27)$$

The gradient descent update rule is:

$$\theta_{t+1} = \theta_t + \eta \delta_t \nabla_\theta Q(s_t, a_t; \theta) \quad (28)$$

where, η represents learning rate and ∇_θ represents gradient of Q-function.

To stabilize training, experience replay is used. The replay memory is defined as: $\mathrm{D} = \{(s_t, a_t, r_t, s_{t+1})\}$. Mini-batch updates are performed as:

$$\theta \leftarrow \theta - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \mathrm{L}_i \tag{29}$$

This reduces correlation between sequential samples and improves convergence stability.

5.6 TARGET NETWORK STABILIZATION STRATEGY

To avoid oscillations in training, a target network is maintained: $\theta^{-} \leftarrow \theta$ after fixed intervals T. Alternatively, soft update is applied: $\theta^{-} = \tau\theta + (1-\tau)\theta^{-}$, where $\tau \in (0,1)$ controls update smoothness. This mechanism ensures stable Q-value estimation and prevents divergence. The DRL module interacts with the web system in a continuous loop: $s_t \rightarrow a_t \rightarrow s_{t+1} \rightarrow r_t$. The control decision modifies: {*cache configuration*, *server allocation*, *routing paths*}. The system objective becomes:

$$\min \mathrm{E}\left[\sum_{t=0}^{T} \gamma^t \cdot (T_{lat} + Q_{len})\right] \tag{30}$$

This ensures that long-term system stability is preserved while improving instantaneous response behavior. The DRL system converges when: $\lim_{t\to\infty} Q(s_t, a_t) \rightarrow Q^*(s_t, a_t)$. The Bellman optimality condition holds:

$$Q^*(s,a) = \mathrm{E}[r + \gamma \max_{a'} Q^*(s', a')] \tag{31}$$

The system is stable when TD error approaches zero: $\delta_t \rightarrow 0$. This indicates that learned policy has reached equilibrium in dynamic web environment.

6. RESULTS AND DISCUSSION

The experimental environment is implemented using Python-based simulation integrated with TensorFlow and Scikit-learn libraries. The DRL environment runs on OpenAI Gym-like custom web simulation modules. The system is executed on an Intel Core i7 processor with 16 GB RAM and NVIDIA GTX 1650 GPU. The experiments simulate dynamic web traffic conditions using synthetic request streams derived from real-world HTTP log distributions.

Table.1. Simulation Parameters

Parameter	Value
Simulation Duration	1000 seconds
Request Arrival Rate	50–500 req/s
Cache Size	256–1024 MB
Server Count	5–20 nodes
Learning Rate (DRL)	0.001
Discount Factor (γ)	0.95
Replay Memory Size	50,000 samples
Batch Size	64
Boosting Trees	200
Exploration Rate (ε)	1.0 → 0.01

The parameters in Table.1 define the operational boundaries of the proposed HDRL-GBO framework. The simulation duration ensures sufficient convergence of reinforcement learning policies under varying traffic patterns. The request arrival rate models real-world web congestion scenarios. Cache size and server count influence load distribution efficiency. Learning rate and discount factor control DRL convergence stability. Replay memory improves sample efficiency, while boosting trees enhance prediction accuracy. Exploration decay ensures transition from exploration to exploitation.

6.1 PERFORMANCE METRICS

The evaluation uses key metrics:

1. **Latency (ms):** Measures average response time of web requests. Lower values indicate better performance.
2. **Throughput (req/s):** Measures number of successfully processed requests per second. Higher values indicate better scalability.
3. **CPU Utilization (%):** Measures computational resource usage across servers. Balanced utilization is preferred.
4. **Cache Hit Ratio (%):** Measures proportion of requests served from cache. Higher values indicate better efficiency.
5. **Load Balance Index:** Measures distribution uniformity across servers. Lower variance indicates better balance.

6.2 DATASET DESCRIPTION

The dataset represents a hybrid collection of synthetic and real-world-inspired web traffic patterns. It captures temporal variations in request loads, system response behavior, and caching efficiency. The time-series structure enables DRL-based sequential learning, while tabular features support gradient boosting prediction accuracy.

Table.2. Dataset Characteristics

Feature	Description
Dataset Name	Web Traffic Dataset
Data Source	Simulated + HTTP log traces
Total Samples	1,00,000 requests
Features	Request rate, response time, CPU load, cache status
Data Type	Time-series + tabular
Label	Latency and system metrics
Sampling Rate	1 second interval
Environment	Cloud-based distributed simulation

The performance comparison includes three baseline methods: MultiSVM Optimization Model, PSO-Based Load Balancing Algorithm, and Enhanced Genetic Load Scheduler (EHGA). These methods represent classical machine learning, swarm intelligence, and evolutionary optimization approaches commonly used for web performance enhancement and serve as benchmarks for evaluating the proposed framework.

Table.3. Latency Comparison Across Methods

Traffic Load	MultiSVM	PSO	EHGA	Proposed HDRL-GBO
50	120	110	105	85
100	150	135	125	95
150	180	160	150	110
200	210	185	170	125
250	240	210	195	140

The results in Table.3 show that latency increases across all methods as traffic load increases. However, the proposed HDRL-GBO method consistently maintains lower latency compared to baseline models. At 50 requests, the proposed method records 85 ms, while MultiSVM records 120 ms, PSO records 110 ms, and EHGA records 105 ms. This indicates a reduction of approximately 29.1%, 22.7%, and 19% respectively. At peak load of 250 requests, the proposed method achieves 140 ms latency, whereas MultiSVM reaches 240 ms. This represents a reduction of approximately 41.6%. The improvement occurs due to adaptive routing and reinforcement-based caching decisions that reduce queue buildup. The gradient boosting module further improves prediction accuracy of incoming load, which reduces delayed decision execution. The PSO and EHGA methods show moderate performance due to heuristic-based optimization, which lacks real-time adaptability. MultiSVM performs worst due to static classification boundaries that do not adjust to dynamic traffic conditions. The proposed model maintains stability even under high load due to continuous policy updates. Overall, latency reduction remains significant and consistent across all traffic levels.

Table.4. Throughput Comparison Across Methods

Traffic Load	MultiSVM	PSO	EHGA	Proposed HDRL-GBO
50	420	450	470	520
100	400	430	460	540
150	380	410	440	560
200	360	390	420	580
250	340	370	400	600

The Table.4 illustrates that throughput decreases for baseline methods as load increases, while the proposed method maintains an increasing trend. At 250 request load, the proposed model achieves 600 req/s, whereas MultiSVM achieves 340 req/s. This represents an improvement of approximately 76.4%. The improvement is attributed to reinforcement learning-based load balancing that distributes requests efficiently across available servers. The caching strategy reduces redundant computations, which improves request processing speed. EHGA performs better than PSO and MultiSVM due to evolutionary optimization but lacks predictive intelligence. PSO shows moderate performance due to particle convergence limitations under dynamic conditions. The proposed HDRL-GBO system adapts its routing policy in real time, which prevents server saturation. The gradient boosting predictor ensures proactive scaling decisions, which reduces overload conditions. This leads to consistent throughput improvement across all traffic levels. The system demonstrates strong scalability under increasing request intensity.

Table.5. CPU Utilization Comparison

Traffic Load	MultiSVM	PSO	EHGA	Proposed HDRL-GBO
50	40	45	48	55
100	50	55	58	65
150	60	65	68	72
200	70	75	78	80
250	85	88	90	92

The Table.5 indicates that CPU utilization increases steadily across all methods. The proposed method achieves balanced utilization, reaching 92% at maximum load. This is higher than baseline methods, indicating better resource exploitation. MultiSVM shows inefficient utilization at lower loads due to static allocation. PSO improves distribution but suffers from oscillatory behavior in load balancing. EHGA achieves better stability due to evolutionary adaptation. However, it still lacks predictive control over workload spikes. The proposed HDRL-GBO system maintains controlled utilization growth through reinforcement-driven scaling actions. The system ensures that no server remains underutilized or overloaded. The improvement of approximately 2% over EHGA at peak load reflects optimized scheduling decisions.

6.3 DISCUSSION

The experimental evaluation demonstrates that the proposed HDRL-GBO framework consistently outperforms baseline models across all performance metrics. Latency reduces significantly by up to 41.6% compared to MultiSVM, as shown in Table.8. Throughput increases by approximately 76.4% compared to classical models, as observed in Table.9. CPU utilization remains balanced with improved efficiency over PSO and EHGA methods, as indicated in Table.10.

The improvement is achieved due to integration of deep reinforcement learning with gradient boosted prediction. The DRL module ensures adaptive decision-making, while the boosting model enhances prediction accuracy of traffic patterns. This combination reduces decision delay and improves system responsiveness. Traditional models such as PSO and MultiSVM lack predictive intelligence, which limits their adaptability under dynamic workloads. The hybrid architecture also improves cache efficiency and load distribution, which reduces queue congestion. The system demonstrates stable performance under increasing traffic conditions, indicating strong scalability. Overall, the results confirm that learning-based optimization provides a more robust solution for modern web performance enhancement compared to heuristic or static methods.

7. CONCLUSION

The study presents a hybrid HDRL-GBO framework for optimizing website performance under dynamic traffic conditions. The system integrates deep reinforcement learning with gradient boosted prediction to enhance latency reduction, throughput improvement, and resource utilization. Experimental

results confirm that the proposed method achieves superior performance compared to MultiSVM, PSO, and EHGA. The latency reduction reaches up to 41.6%, which demonstrates the effectiveness of adaptive caching and routing decisions. Throughput improvement reaches approximately 76.4%, indicating strong scalability under high request loads. CPU utilization remains balanced at 92%, which ensures efficient resource usage without system overload. The reinforcement learning module continuously adapts system behavior based on environmental feedback, while the boosting model provides accurate traffic prediction. This dual mechanism ensures proactive and reactive optimization simultaneously. The system maintains stability under fluctuating workloads and avoids performance degradation. The proposed framework demonstrates that hybrid learning-based optimization significantly improves web system efficiency compared to conventional techniques. It provides a scalable and intelligent solution for modern distributed web environments where real-time adaptability is essential.

REFERENCES

[1] R. Haryana, "Improving the Performance of Hybrid Models using Machine Learning and Optimization Techniques", *International Journal of Membrane Science and Technology*, Vol. 10, No. 2, pp. 3396-3409, 2023.

[2] M. Ghattas, A.M. Mora and S. Odeh, "A Novel Approach for Evaluating Web Page Performance based on Machine Learning Algorithms and Optimization Algorithms", *AI*, Vol. 6, No. 2, pp. 1-35, 2025.

[3] S.K. Gupta, M. Tripathi and J. Grover, "Hybrid Optimization and Deep Learning based Intrusion Detection System", *Computers and Electrical Engineering*, Vol. 100, pp. 1-8, 2022.

[4] J. Wang, T. Lu, L. Li and D. Huang, "Enhancing Personalized Search with AI: A Hybrid Approach Integrating Deep Learning and Cloud Computing", *Journal of Advanced Computing Systems*, Vol. 4, No. 10, pp. 1-13, 2024.

[5] K.N.G. Veerappan, J. Perumal and S.J.N. Kumar, "Categorical Data Clustering using Meta Heuristic Link-based Ensemble Method: Data Clustering using Soft Computing Techniques", *Dynamics of Swarm Intelligence Health Analysis for the Next Generation*, pp. 226-238, 2023.

[6] T. Yang and J. Sun, "A hybrid Ensemble Deep Learning Framework with Novel Metaheuristic Optimization for Scalable Malicious Website Detection", *Scientific Reports*, Vol. 15, No. 1, pp. 1-17, 2025.

[7] N.S.K.M. Tirumanadham, S. Thaiyalnayaki and M. Sriram, "Improving Predictive Performance in E-Learning through Hybrid 2-Tier Feature Selection and Hyper Parameter-Optimized 3-Tier Ensemble Modeling", *International Journal of Information Technology*, Vol. 16, No. 8, pp. 5429-5456, 2024.

[8] A. Karim, M. Shahroz, K. Mustofa, S.B. Belhaouari and S.R.K. Joga, "Phishing Detection System through Hybrid Machine Learning based on URL", *IEEE Access*, Vol. 11, pp. 36805-36822, 2023.

[9] S.K. Sriramulugari, V.A.K. Gorantla, V. Gude and K. Gupta, "Exploring Mobility and Scalability of Cloud Computing Servers using Logical Regression Framework", *Proceedings of International Conference on Disruptive Technologies*, pp. 488-493, 2024.

[10] S. Parvathareddy, A. Yahya, L. Amuhaya, R. Samikannu and R.S. Suglo, "A Hybrid Machine Learning and Optimization Framework for Energy Forecasting and Management", *Results in Engineering*, Vol. 26, pp. 1-15, 2025.

[11] S. Simaiya, U.K. Lilhore, Y.K. Sharma, K.B. Rao, V.V.R. Maheswara Rao, A. Baliyan and R. Alroobaea, "A Hybrid Cloud Load Balancing and Host Utilization Prediction Method using Deep Learning and Optimization Techniques", *Scientific Reports*, Vol. 14, No. 1, pp. 1-18, 2024.

[12] J. Isabona, A.L. Imoize and Y. Kim, "Machine Learning-based Boosted Regression Ensemble Combined with Hyperparameter Tuning for Optimal Adaptive Learning", *Sensors*, Vol. 22, No. 10, pp. 1-22, 2022.

[13] A. Ampavathi and T.V. Saradhi, "Multi Disease-Prediction Framework using Hybrid Deep Learning: An Optimal Prediction Model", *Computer Methods in Biomechanics and Biomedical Engineering*, Vol. 24, No. 10, pp. 1146-1168, 2021.

[14] U.K. Lilhore, S. Simaiya, Y.K. Sharma, A.K. Rai, S.M. Padmaja, K.V. Nabilal and H. Alsufyani, "Cloud-Edge Hybrid Deep Learning Framework for Scalable IoT Resource Optimization", *Journal of Cloud Computing*, Vol. 14, No. 1, pp. 1-27, 2025.

[15] U.K. Lilhore, S. Simaiya, Y.K. Sharma, A.K. Rai, S.M. Padmaja, K.V. Nabilal and H. Alsufyani, "Cloud-Edge Hybrid Deep Learning Framework for Scalable IoT Resource Optimization", *Journal of Cloud Computing*, Vol. 14, No. 1, pp. 1-27, 2025.