JURNAL RESPONSIVE Teknik Informatika

Juni 2026 | Vol. 10 | No. 01
E-ISSN : 2614-7602
DOI: 10.36352/jr.v10i1

Api Response Time Prediction Using a Deep Learning Model on Web Application Load Testing Simulation Data

Defnizal[1]*, Atman Lucky Fernandes[2], Aprizal Y[3], David Saro[4], Ghea Paulina Suri[5], Nofri Yudi Arifin[6], Romiko Afriantoni[7], Willy Rizki Perdana[8]

[1]Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Putra Indonesia YPTK, Padang, Indonesia

[2,3,4,5]Program Studi Teknik Informatika, Fakultas Sains dan Teknologi, Universitas Ibnu Sina, Batam, Indonesia

[6,7,8]Program Studi Sistem Informasi, Fakultas Sains dan Teknologi, Universitas Ibnu Sina, Batam, Indonesia

e-mail: defnizal@upiyptk.ac.id,

Abstract

Web application performance, particularly application programming interface (API) response time, is one of the primary determinants of user experience and system reliability, especially under high load conditions. Conventional load testing is reactive, as it can only measure actual response time after a given load has been applied to the system, creating a need for a predictive approach capable of estimating API response time from load and server resource conditions. This study applies a deep learning model in the form of a Multi Layer Perceptron (MLP) with three hidden layers to predict API response time, and compares its performance against two baseline models, namely Linear Regression and Random Forest Regressor. Due to limited access to sensitive real-world production data, this study uses a simulated load testing dataset of 500 samples generated programmatically, comprising ten load and server-resource features such as concurrent users, requests per second, payload size, database query count, cache hit ratio, and CPU and memory utilization, with non-linear patterns representing resource contention effects under high load. The data was split into 70% training and 30% testing, then evaluated using Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), and the coefficient of determination (R^2). Experimental results show that the Deep Learning (MLP) model achieved the best predictive performance with an RMSE of 24.86 ms, MAPE of 19.33%, and R^2 of 0.9009, outperforming Linear Regression (RMSE 26.25 ms, R^2 0.8895) and Random Forest (RMSE 27.90 ms, R^2 0.8751), although Linear Regression's MAE was slightly lower (19.34 ms versus 19.53 ms). Feature-importance analysis indicates that the number of concurrent users, database query count, and CPU utilization are the most dominant factors influencing API response time. To ground the simulated scenario in a concrete example, a small real demonstration web application was also built and load-tested; its qualitative behavior (response time increasing with concurrency) is consistent with the assumptions underlying the simulated dataset. These findings indicate that the deep learning model is better able to capture non-linear patterns caused by resource contention compared to a linear model or a tree based ensemble, suggesting potential use as a predictive component in auto-scaling strategies or web-application capacity planning, with the caveat that these results remain a preliminary study requiring further validation using real production data.

Keywords—response time prediction, deep learning, load testing, API performance, simulated data

INTRODUCTION

Modern web applications, particularly those built on microservices and application programming interface (API) architectures, are required to remain responsive even when facing surges in user numbers or high request volumes. Slow API response time can degrade user experience, increase abandonment rates on e-commerce applications, and potentially violate the service level agreement (SLA) agreed upon between a service provider and its users. Load testing has become a standard practice for evaluating system resilience against various load scenarios before an application is released to a production environment.

Nevertheless, conventional load testing is reactive: testing can only measure actual response time after a given load has genuinely been applied to the system, requiring development teams to run test scenarios repeatedly to understand the relationship between load conditions and response time. Machine learning and deep learning based approaches offer a predictive alternative, namely building a model capable of estimating API response time from load parameters and server resource conditions without having to run an actual test for every scenario combination, thereby supporting capacity and auto scaling decisions more proactively.

This study aims to apply a deep learning model in the form of a Multi Layer Perceptron (MLP) to predict API response time based on simulated web-application load testing data, covering load parameters such as concurrent users, requests per second, payload size, endpoint complexity, and server resource conditions such as CPU utilization, memory utilization, and cache hit ratio. To objectively position the deep learning model, this study also compares its performance against two baseline models commonly used for regression problems, namely Linear Regression as a linear baseline and Random Forest Regressor as a non-linear, tree-based ensemble baseline.

The main contributions of this study are: (1) providing a web application load testing simulation framework that represents non linear resource contention patterns, is transparent, and is free from copyright or production data confidentiality issues; (2) applying and evaluating a deep learning model (MLP) for the API response-time prediction problem and comparing it against linear and ensemble baselines; and (3) analyzing the contribution of each load and serverresource feature to API response time through a feature-importance approach.

LITERATURE REVIEW

Prediction of response time and web service performance (Quality of Service/QoS) has become a widely studied topic as the complexity of service-based application architectures has increased. Zheng, Li, Tang, Xie, and Lyu (2020) surveyed various collaborative filtering approaches for web-service QoS prediction, including the response time attribute, and identified sparse data and the dynamic, ever changing nature of the service environment as key challenges.

Mourougaradjane and Dinadayalan (2022) conducted a systematic comparison of several machine-learning and deep-learning algorithms for predicting web service performance and successability, and concluded that deep learning approaches generally produced more accurate predictive models than conventional machine-learning algorithms, which are highly dependent on the quality of manual feature engineering.

In a closely related context workload prediction in cloud datacenter environments Kumar, Goomer, and Singh (2018) applied a Long Short Term Memory Recurrent Neural Network (LSTM-RNN) model to predict web and cloud server workload, reporting that the approach achieved a very low mean squared error compared to conventional statistical approaches, particularly for load patterns with long-term dependencies. That study reaffirmed the advantage of neural network architectures in capturing non-linear and temporal patterns in system-

performance data, although the present study instead uses an MLP based on independent load snapshot features rather than sequential time series data, since the simulated data here is constructed as independent samples.

Architecturally, the MLP as a foundational deep learning form relies on the backpropagation algorithm introduced by Rumelhart, Hinton, and Williams (1986) to update network weights based on the error gradient, while training optimization in this study uses the Adam algorithm introduced by Kingma and Ba (2015) for its ability to adaptively adjust the learning rate, yielding more stable convergence than conventional stochastic gradient descent. As a non linear, ensemble based comparator, Random Forest Regressor follows the ensemble learning principle introduced by Breiman (2001), while Linear Regression, as a linear baseline, follows the classical regression framework described by Hastie, Tibshirani, and Friedman (2009).

More recent work has extended performance prediction beyond the backend layer. Kaushik, Kumar, and Raj (2024) proposed a framework that combines micro frontends with backend microservices and applies a machine learning model to predict the performance of the resulting web applications, arguing that most prior work considers only backend services and therefore captures an incomplete picture of end to end response behaviour. In a follow up study, Kaushik, Kumar, and Raj (2025) addressed the reliability of microservices applications operating under dynamic workloads, reinforcing the view that load dependent behaviour rather than static configuration is the dominant determinant of perceived service quality.

Parallel developments have appeared in the autoscaling literature, which is the primary practical consumer of response time and workload predictions. Pintye, Kovacs, and Lovas (2024) showed that the choice of input metrics is decisive for machine learning based autoscalers: by selecting metrics through statistical analysis rather than relying on a fixed static set, their approach reduced quality of service violations by up to eighty percent and lowered virtual machine usage substantially. That finding is directly relevant to the present study, since it implies that identifying which load and server resource features actually drive response time is as important as the choice of predictive algorithm. Wang et al. (2024) reported a comparable result with DeepScaling, a system that autoscales microservices toward stable CPU utilization in large scale production clouds, while Alharthi, Alshamsi, Alseiari, and Alwarafy (2024) surveyed the field and identified prediction accuracy under non linear load as an open research direction.

On the modelling side, deep neural architectures have consistently outperformed classical baselines for cloud-workload problems. Xu, Song, Wu, Gill, Ye, and Xu (2022) introduced esDNN, a deep neural network for multivariate workload prediction, and Patel and Bedi (2023) proposed a multivariate attention network for cloud workload forecasting. Sabyasachi, Sahoo, and Ranganath (2024) compared CNN and LSTM approaches for the same task. These studies, however, uniformly treat workload as a temporal sequence. The present study differs by framing response time estimation as a cross-sectional regression over independent load snapshots, a formulation that is appropriate when the objective is to answer what-if capacity questions rather than to forecast the next time step.

Unlike prior studies that generally rely on real production log data or public benchmark datasets, this study adopts a programmatically generated load testing simulation approach that incorporates non linear resource contention effects a disproportionate spike in response time as CPU or memory utilization approaches capacity limits. This approach is relevant for development teams or researchers who wish to evaluate the feasibility of a predictive model before instrumenting and collecting performance data in a real production environment.

RESEARCH METHOD

3.1 Research Framework

This study was carried out through five main stages: (1) generation of simulated web application load testing data, (2) data preprocessing, (3) splitting of the data into training and testing sets, (4) training of the deep learning model along with two baseline models, and (5) evaluation and comparative analysis of the prediction results.

3.2 Simulated Dataset

The dataset used is entirely synthetic and was generated using the NumPy library, without copying or deriving values from any production log data or third party benchmark dataset. The dataset consists of 500 samples with 10 numeric features representing load testing parameters and serve resource conditions, as summarized in Table 1, along with a target variable representing API response time (response_time_ms) ranging from approximately 5 to 396 milliseconds. Figure 1 illustrates the testing-scenario architecture that served as the conceptual reference for generating this simulated data.

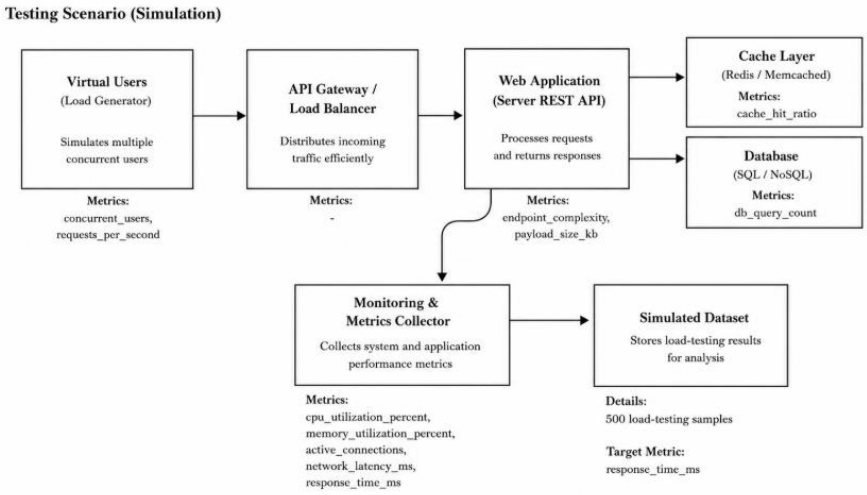

Figure 1. Illustration of the Simulated Load Testing Scenario Architecture

This scenario represents the general pattern of load testing on a REST API based web application: a number of virtual users send requests through an API gateway/load balancer to an application server, which in turn interacts with a caching layer and a database. Server resource conditions (CPU, memory, active connections) and the resulting response time for each load combination are then recorded by a monitoring component to form the simulated dataset. It should be emphasized that this diagram is illustrative, conceptually depicting the scenario context, and does not represent the architecture of any specific real production system.

Table 1. Load and Server-Resource Features in the Simulated Load-Testing Dataset

Feature	Description	Range
Concurrent users	Number of concurrent users sending requests simultaneously	5-500
Requests per second	Request intensity per second (RPS) toward the API	≈0.5-122
Payload size kb	Size of the request/response payload	≈1-169 KB
Db query count	Number of database queries triggered per request	0 -17

Endpoint complexity	Endpoint complexity (1 = simple, 5 = complex aggregation)	1-5
Cache hit ratio	Ratio of requests successfully served from cache	0 -1
Network latency ms	Baseline network latency before server-side processing	≈2 -52 ms
Active connections	Number of active connections in the server connection pool	≈1-475
Cpu utilization percent	Server CPU utilization while the request is being processed	≈5- 92%
Memory utilization percent	Server memory utilization while the request is being processed	≈5-93%

Response-time values were generated through an engineered formula combining a baseline network latency component, a computation cost that increases with the number of concurrent users, request intensity, payload size, and database query count, minus the positive effect of the cache hit ratio, plus a non-linear (power law) penalty that appears when CPU utilization exceeds 75% or memory utilization exceeds 85%, in order to simulate the resource contention phenomenon commonly observed on servers under high load. Proportional random (Gaussian) noise was also added so that the data is not fully deterministic and better represents the variability seen in real performance measurements.

3.3 Real Application Demonstration and Small Scale Load Testing (Illustrative)

To provide a concrete illustration of the testing context depicted in Figure 1, the author built a simple real demonstration web application ("SimpleShop Demo") using Flask and SQLite, consisting of a product-catalog page and several REST API endpoints (GET /api/products, GET /api/products/<id>, POST /api/orders). Figure 2 shows an actual screenshot of this application's interface, which was genuinely run locally for demonstration purposes.

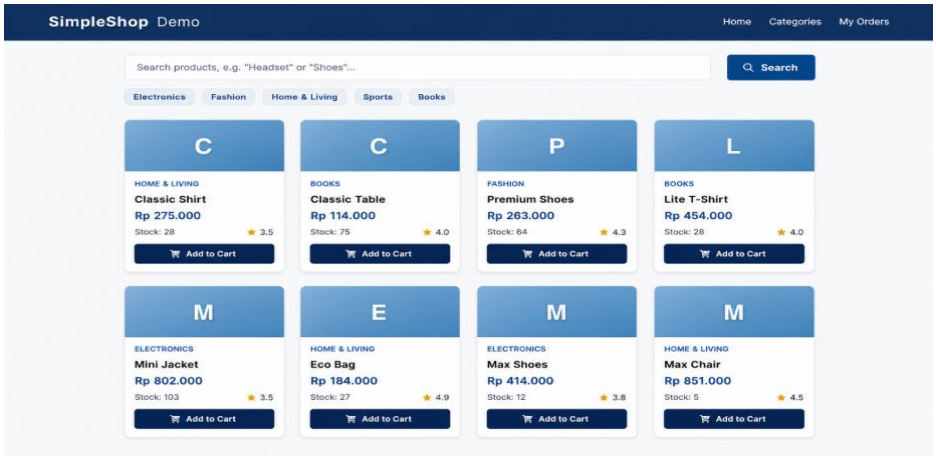

Figure 2. Screenshot of the Demonstration Application

To complete the illustration, the author also ran a real (non simulated) load test against the GET/api/products endpoint of this application using a Python script with ThreadPoolExecutor, sending 60 requests at each concurrency level (1, 5, 10, 20, 40, 60, and 80 concurrent requests) and measuring the actual response time of every request. Figure 3 shows an excerpt of the test execution log, while Figure 4 visualizes the measurement results.

```
python3 load_test.py

$ python3 load_test.py
Target        : http://127.0.0.1:5055/api/products
Requests/level: 60
----------------------------------------------------------------------
Concurrency |  Mean (ms) |  P50 (ms) |  P95 (ms) |  Max (ms) | Errors
----------------------------------------------------------------------
          1 |      24.19 |     24.88 |     32.54 |     33.22 |      0
          5 |      25.61 |     25.47 |     34.52 |     40.05 |      0
         10 |      30.00 |     30.47 |     39.45 |     43.95 |      0
         20 |      43.77 |     44.89 |     55.94 |     69.30 |      0
         40 |      75.47 |     73.37 |    103.82 |    114.54 |      0
         60 |      92.53 |     91.79 |    124.14 |    130.08 |      0
         80 |      73.78 |     77.10 |     91.85 |    102.13 |      0
Load test complete. Log and JSON results saved.
```

Figure 3. Execution Log Excerpt from the Real Load Test on the Demonstration Application

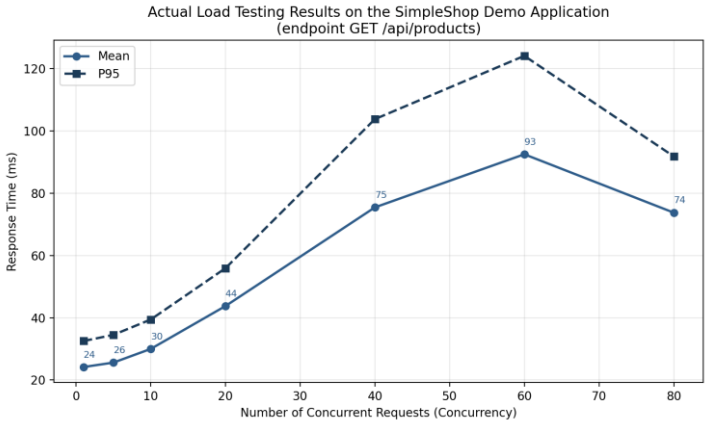

Figure 4. Actual Response-Time Measurement Results on the Demonstration Application

The results of this real, smallscale test show a pattern that is qualitatively consistent with the assumptions underlying the simulated dataset used in the main study, namely that response time tends to increase as the number of concurrent requests grows: mean response time rose from about 24.2 ms at 1 concurrent request to about 92.5 ms at 60 concurrent requests. Interestingly, at the highest concurrency level tested (80), mean response time actually decreased slightly to about 73.8 ms compared to the level 60 result, likely due to thread-scheduling characteristics of the Flask development server, whose genuine parallelism is limited by Python's Global Interpreter Lock (GIL); this non-monotonic pattern is a genuine real world phenomenon that differs from the strictly controlled pattern built into the simulated dataset.

It must be emphasized that the demonstration application and its real load-test results in this section are illustrative, intended to reinforce the context of the research scenario, and were not used as a source of training or testing data for the prediction models in this study. The dataset used for model training and evaluation continues to rely on the 500 sample simulated dataset described in Section 3.2, since testing a single small application in a local environment does not represent the broader variation in load, architecture, and server-resource conditions that the simulated dataset is designed to cover.

3.4 Data Preprocessing

The data was split into a training set (350 samples, 70%) and a testing set (150 samples, 30%) using a random split. All features were standardized using StandardScaler to have a mean of zero and a standard deviation of one, since the Deep Learning (MLP) and Linear Regression models are sensitive to feature scale. The target variable (response_time_ms) was also standardized specifically for training the Deep Learning model in order to stabilize the gradient-

optimization process, after which predictions were transformed back to the original millisecond scale (inverse transform) before evaluation. The Random Forest Regressor was trained on the data in its original scale, since tree-based models are not sensitive to feature scale.

3.5 Deep Learning Model Architecture

The deep learning model used is a Multi Layer Perceptron (MLP) with the following configuration:

1. Input layer, 10 neurons, corresponding to the number of load and server resource features.
2. Three hidden layers with 128, 64, and 32 neurons respectively, each using the ReLU (Rectified Linear Unit) activation function.
3. Output layer, 1 neuron with linear activation, appropriate for the regression problem.
4. Adam optimization algorithm with an initial learning rate of 0.001 and Mean Squared Error (MSE) as the loss function.
5. L2 regularization (alpha = 0.0001) and early stopping based on a 15% validation split with a patience of 30 epochs without improvement, to prevent overfitting.

3.6 Baseline Models

1. Linear Regression

a classical linear regression model using the Ordinary Least Squares method, used as a simple baseline that assumes a linear relationship between features and the target.

2. Random Forest Regressor

an ensemble of 200 decision trees (n_estimators=200) with a maximum depth of 14 levels, used as a non linear, tree-based baseline.

3.7 Evaluation Scheme

The predictive performance of each model was evaluated using four metrics: Mean Absolute Error (MAE), which measures the average absolute difference between actual and predicted values; Root Mean Squared Error (RMSE), which places greater weight on large errors; Mean Absolute Percentage Error (MAPE), which expresses error as a percentage relative to the actual value; and the coefficient of determination (R^2), which indicates the proportion of target variance explained by the model. Each model's training time was also recorded as a practical consideration. All experiments were run using the Python programming language with the scikit-learn library on the same computing environment.

RESULTS AND DISCUSSION

4.1 Comparative Model Performance

Table 2 presents the evaluation results of the three models on the test data (150 samples) based on MAE, RMSE, MAPE, R^2, and training time. Results are ordered from the lowest to the highest RMSE.

Table 2. Performance Comparison of API Response Time Prediction Models on the Test Data

Model	MAE (ms)	RMSE (ms)	MAPE (%)	R^2	Training Time (s)
Deep Learning (MLP)	19.53	24.86	19.33%	0.9009	0.143
Linear Regression	19.34	26.25	23.53%	0.8895	0.003
Random Forest	21.40	27.90	21.50%	0.8751	0.446

The Deep Learning (MLP) model achieved the best performance on three of the four main metrics: the lowest RMSE (24.86 ms), the lowest MAPE (19.33%), and the highest R^2 (0.9009), indicating that this model explains about 90% of the variance in API response time on the test data. Interestingly, Linear Regression's MAE (19.34 ms) was slightly lower than Deep Learning's (19.53 ms). The gap is 0.19 ms, which amounts to well under one percent of the mean response time and falls within the range that a different random seed or train test split could plausibly reverse. Because this study reports a single training run and did not apply a statistical significance test to the residuals, the MAE ranking between these two models should be read as indistinguishable rather than as evidence that the linear model is superior on that metric. Nevertheless, Linear Regression's RMSE and MAPE were markedly higher than Deep Learning's, indicating that Linear Regression tends to produce larger prediction errors on certain samples, most likely under high-load conditions where the relationship between features and response time is non-linear due to resource contention, a pattern the linear model is less able to capture.

Random Forest Regressor, although a non linear, tree based ensemble model, actually recorded the lowest performance among the three models on every metric. This is most likely due to the limited size of the training data (350 samples), which was insufficient to optimally train a 200-tree ensemble to capture the complex non linear interactions among features, so that the ensemble may have fitted local patterns in the training data without generalizing equally well to the test data. It should be stressed that this explanation is an interpretation rather than a tested hypothesis: the present study did not perform a learning curve experiment across varying training set sizes, and the claim therefore remains to be verified. In terms of computational efficiency, Linear Regression trained far faster (0.003 seconds) than Deep Learning (0.143 seconds) or Random Forest (0.446 seconds), making it a relevant fast baseline despite its lower accuracy.

4.2 Deep Learning Training Loss Curve

Figure 5 shows the training loss curve of the Deep Learning (MLP) model on the standardized target scale. Training stopped automatically at epoch 63 via the early-stopping mechanism after no improvement was observed on the validation set for 30 consecutive epochs. The curve shows a sharp decline in loss over the first 20 epochs, followed by a stable plateau with no sign of oscillation or divergence, indicating that the optimization process proceeded stably and that the model did not suffer significant overfitting on the training data.

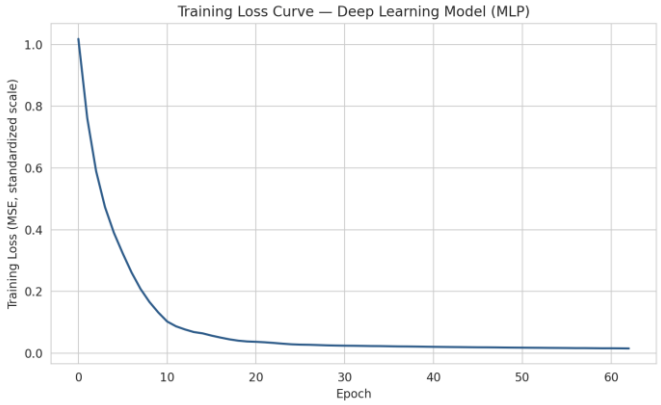

Figure 5. Training Loss Curve of the Deep Learning Model (MLP)

4.3 Actual vs Predicted and Residual Analysis

Figure 6 compares actual response time with the Deep Learning model's predicted response time on the test data. Most data points cluster tightly around the ideal reference line ($y = \hat{y}$) in the low to medium response-time range (roughly 0-250 ms), indicating good predictive

accuracy under normal to moderate load conditions. In the high response-time range (above 300 ms), which represents resource contention conditions caused by CPU or memory utilization approaching capacity limits, the model tends to slightly underestimate the actual value, likely because the number of samples under such extreme load conditions is relatively small in the training data.

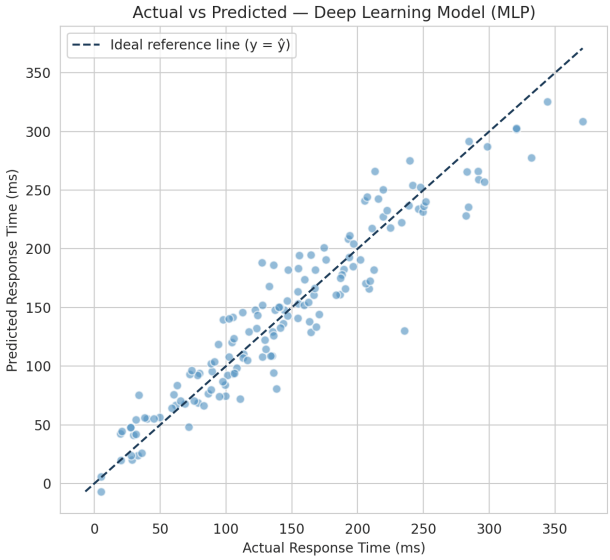

Figure 6. Actual vs Predicted Deep Learning Model (MLP)

Figure 7 shows the residual plot (actual minus predicted) against the predicted value. Residuals are scattered relatively evenly around the zero line across the whole prediction range without forming a clear funnel shape, indicating that the model does not exhibit significant systematic bias across different response time ranges. Most residuals fall within ±50 ms, with one outlier of about 105 ms at a mid range prediction (130 ms), suggesting one sample with an extreme non-linear pattern that was not fully captured by the model.

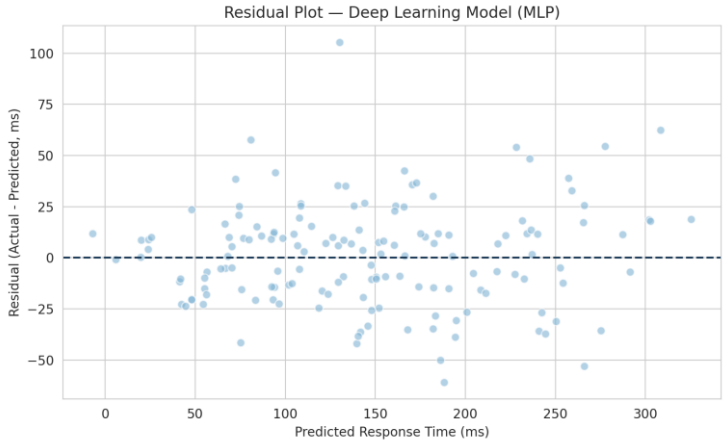

Figure 7. Residual Plot Deep Learning Model (MLP)

4.4 MAE and RMSE Comparison Across Models

Figure 8 visually summarizes the MAE and RMSE comparison across the three models. The MAE differences among the three models are relatively small, but the RMSE differences are more pronounced, particularly between Deep Learning and Random Forest, confirming that Deep Learning is more consistent at controlling large prediction errors than the two baseline models.

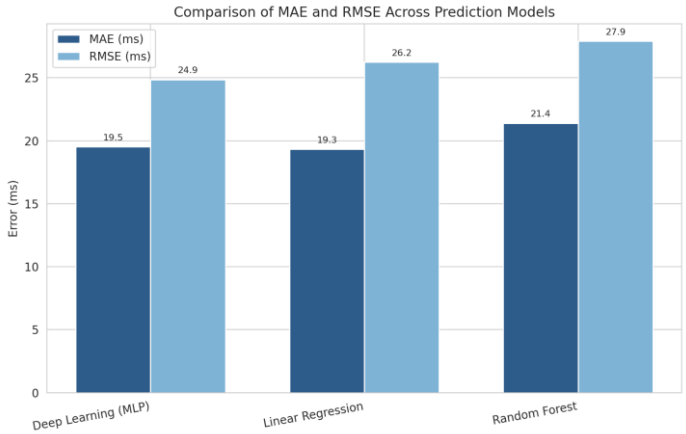

Figure 8. Comparison of MAE and RMSE Across Prediction Models

4.5 Feature Importance Analysis

Figure 9 presents feature importance scores based on the Random Forest Regressor model as an interpretability aid, since Random Forest produces built in, easily interpretable feature-importance estimates despite not being the model with the lowest RMSE in this experiment. The concurrent_users feature was recorded as the most dominant determinant, contributing nearly 46%, far ahead of the other features, followed by db query count (18%) and cpu_utilization_percent (12%). This pattern is consistent with the data-generation formula, in which response time is non-linearly influenced by the number of concurrent users and incurs a significant penalty once CPU utilization crosses a certain threshold. The active connections and endpoint_complexity features also contributed substantially, while features such as payload_size_kb and cache_hit_ratio showed relatively small contributions compared to the other load and computation factors.

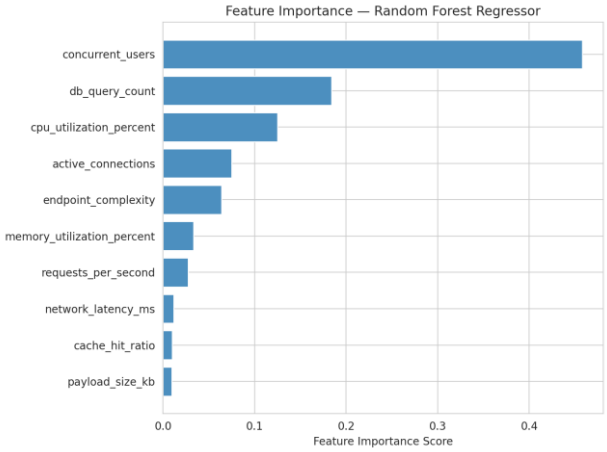

Figure 9. Feature Importance Random Forest Regressor

4.6 Discussion

Overall, the results of this study show that the Deep Learning (MLP) model offers the best balance between predictive accuracy and the ability to capture non linear patterns in the API response-time prediction problem, reflected in its superior RMSE, MAPE, and R^2 compared to Linear Regression and Random Forest. This advantage is consistent with the MLP's nature as a universal function approximator capable of learning complex non-linear relationships among

features without manual feature engineering, a finding also reported in the machine learning versus deep learning comparison study for web service performance prediction by Mourougaradjane and Dinadayalan (2022).

However, Deep Learning's advantage in this study does not hold absolutely across every metric, given that Linear Regression's MAE was in fact slightly lower. This finding underscores the importance of evaluating prediction models using more than one metric, since MAE and RMSE can paint different pictures depending on the error distribution: MAE treats all errors proportionally, whereas RMSE places greater weight on large errors, which are arguably more critical in the context of API response-time prediction, since a prediction that is far off under high-load conditions could lead to flawed auto scaling or capacity planning decisions.

Random Forest's performance, which trailed both other models, is also worth noting. This is most likely influenced by the relatively small dataset size (350 training samples), which limited the tree ensemble's ability to learn complex non linear interactions, particularly the cubic penalty pattern under resource contention conditions. A larger dataset could substantially improve Random Forest's performance, so this result should not be generalized into a conclusion that Random Forest is inherently unsuited to this problem. Confirming the data size explanation would require training the ensemble across a range of training set sizes and inspecting the resulting learning curve, which the present study leaves to future work.

This study also has several limitations worth noting. First, the data used is simulated and generated from an engineered formula designed by the researcher; although designed to represent general patterns reported in the performance testing literature, its characteristics do not fully capture the complexity of real production data, which is shaped by system architecture, network conditions, and actual user behavior. Second, each sample in this dataset was treated as an independent observation (snapshot), whereas in practice API response time often has temporal dependencies, such as load ramp up patterns or daily trends, as captured by the LSTM approach in the study by Kumar, Goomer, and Singh (2018). Third, the architecture and hyperparameters of the Deep Learning model in this study were not optimized through a systematic hyperparameter search (e.g., grid search or Bayesian optimization), so there remains room for further performance improvement. Fourth, and most consequentially for the interpretation of Table 2, all reported metrics derive from a single training run on one random traintest split. No repeated runs with different seeds, no k fold cross validation, and no paired significance test on the residuals were performed. The differences between models reported here should therefore be treated as indicative rather than as statistically established, particularly where the margins are small.

For future research, the most immediate priority is to establish the statistical reliability of the comparison itself: repeating each experiment across multiple random seeds, reporting mean and standard deviation for every metric, applying k-fold cross validation, and testing pairwise differences with a non parametric test such as the Wilcoxon signed rank test on the residuals. Beyond that, it is recommended to validate these findings using real production performance log data, enlarge the simulated dataset, conduct a learning-curve study to test the data size explanation for the Random Forest result, and explore time series deep learning architectures such as LSTM or Gated Recurrent Units (GRU) if performance data is collected sequentially over time. A well validated predictive model could potentially be integrated as a decision support component within auto-scaling systems or proactive web application infrastructure capacity planning.

CONCLUSION

This study applied a Deep Learning model in the form of a Multi Layer Perceptron (MLP) to predict API response time on a 500-sample simulated web application load testing dataset, and compared its performance against Linear Regression and Random Forest Regressor. Experimental results show that Deep Learning (MLP) achieved the best performance on RMSE

(24.86 ms), MAPE (19.33%), and R^2 (0.9009), although Linear Regression's MAE was slightly lower (19.34 ms versus 19.53 ms). Random Forest Regressor recorded the lowest performance among the three models, plausibly due to the limited size of the training data, although this explanation was not tested directly. Because all figures come from a single training run without cross-validation or a significance test, the narrow MAE gap between Deep Learning and Linear Regression should not be read as a meaningful ranking on that particular metric. Feature importance analysis indicates that the number of concurrent users, database query count, and CPU utilization are the most dominant factors influencing API response time, consistent with the resource contention pattern simulated in the dataset. A small real demonstration application and an accompanying genuine load test further showed qualitatively consistent behavior response time increasing with concurrency reinforcing the plausibility of the simulated scenario, although this demonstration was not used as training or evaluation data for the models.

These findings indicate that deep learning models have potential as a predictive component within auto scaling strategies or web application capacity planning, particularly for anticipating non linear response time spikes caused by resource contention. Future research is recommended to first place the model comparison on firmer statistical ground through repeated runs, cross-validation, and paired significance testing, and subsequently to validate these findings using real production performance data, enlarge the dataset, systematically optimize hyperparameters, and explore time-series deep-learning architectures such as LSTM when sequential performance data is available.

REFERENCES

1. Alharthi, S., Alshamsi, A., Alseiari, A., & Alwarafy, A. (2024). Auto scaling techniques in cloud computing: Issues and research directions. Sensors, 24(17), 5551. https://doi.org/10.3390/s24175551
2. Bansal, S., & Kumar, M. (2023). Deep learning-based workload prediction in cloud computing to enhance the performance. In 2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC) (pp. 635–640). IEEE. https://doi.org/10.1109/ICSCCC58608.2023.10176790
3. Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324
4. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Springer.
5. Imdoukh, M., Ahmad, I., & Alfailakawi, M. G. (2019). Machine learning based auto scaling for containerized applications. Neural Computing and Applications, 32, 9745–9760. https://doi.org/10.1007/s00521-019-04507-z
6. Kaushik, N., Kumar, H., & Raj, V. (2024). Micro frontend based performance improvement and prediction for microservices using machine learning. Journal of Grid Computing, 22(2), 44. https://doi.org/10.1007/s10723-024-09760-8
7. Kaushik, N., Kumar, H., & Raj, V. (2025). A novel approach for enhancing reliability of microservices applications under dynamic workloads. Cluster Computing, 28, 669. https://doi.org/10.1007/s10586-025-05471-1
8. Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Proceedings of the 3rd International Conference on Learning Representations (ICLR).
9. Kumar, J., Goomer, R., & Singh, A. K. (2018). Long short term memory recurrent neural network (LSTM-RNN) based workload forecasting model for cloud datacenters. Procedia Computer Science, 125, 676–682. https://doi.org/10.1016/j.procs.2017.12.087

10. Mourougaradjane, P., & Dinadayalan, P. (2022). Prediction of web service performance and successability using comparative analysis of machine learning and deep learning algorithms. International Journal of Intelligent Systems and Applications in Engineering, 10(3), 322–328.
11. Patel, Y. S., & Bedi, J. (2023). MAG-D: A multivariate attention network based approach for cloud workload forecasting. Future Generation Computer Systems, 142, 376–392. https://doi.org/10.1016/j.future.2023.01.002
12. Pintye, I., Kovács, J., & Lovas, R. (2024). Enhancing machine learning-based autoscaling for cloud resource orchestration. Journal of Grid Computing, 22, 68. https://doi.org/10.1007/s10723-024-09783-1
13. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533–536. https://doi.org/10.1038/323533a0
14. Sabyasachi, A. S., Sahoo, B. M., & Ranganath, A. (2024). Deep CNN and LSTM approaches for efficient workload prediction in cloud environment. Procedia Computer Science, 235, 2651–2661.
15. Wang, Z., Zhu, S., Li, J., Jiang, W., Ramakrishnan, K. K., Yan, M., Zhang, X., & Liu, A. X. (2024). DeepScaling: Autoscaling microservices with stable CPU utilization for large scale production cloud systems. IEEE/ACM Transactions on Networking, 32(5), 3961–3976. https://doi.org/10.1109/TNET.2024.3400953
16. Xu, M., Song, C., Wu, H., Gill, S. S., Ye, K., & Xu, C. (2022). esDNN: Deep neural network based multivariate workload prediction in cloud computing environments. ACM Transactions on Internet Technology, 22(3), 1–24. https://doi.org/10.1145/3524114
17. Zheng, Z., Li, X., Tang, M., Xie, F., & Lyu, M. R. (2020). Web service QoS prediction via collaborative filtering: A survey. IEEE Transactions on Services Computing, 15(4), 2455–2472. https://doi.org/10.1109/TSC.2020.2995571