Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

ATLAS - AI Tuned Load Assessment System

Kshitiz Vardhan
Department of Computer Science and Engineering
Chitkara University Institute of Engineering and Technology, Chitkara University
Punjab, India

Gunita Dhiman
Department of Computer Science and Engineering
Chitkara University Institute of Engineering and Technology, Chitkara University
Punjab, India

Anushka Sharma
Department of Computer Science and Engineering
Chitkara University Institute of Engineering and Technology, Chitkara University
Punjab, India

Krish Bahukhandi
Department of Computer Science and Engineering
Chitkara University Institute of Engineering and Technology, Chitkara University
Punjab, India

Dr. Anshu Singla
Department of Computer Science and Engineering
Chitkara University Institute of Engineering and Technology, Chitkara University
Punjab, India

***Abstract* - An enduring problem in software engineering is how to maintain consistent web service performance under changing demand. Traditional load testing approaches are post-hoc, giving measurements but no insight into how the system will degrade in the future. In this paper, we propose ATLAS (AI-Tuned Load Assessment System), a framework that combines adaptive asynchronous load generation with ensemble machine learning pipelines to provide predictive performance diagnostics.**

Keywords* - *Adaptive load testing, ensemble regression, failure probability estimation, Isolation Forest, digital twin, performance engineering, asynchronous concurrency.

I. INTRODUCTION

Web services are the backbone of modern computing environments and support millions of simultaneous users from geographically disparate locations. Such services are supposed to provide timely responses to users with reduced error rates and high throughput depending on demand conditions, which is not just a matter of quality but also one of business significance. Jiang et al. [1] demonstrated that even a single performance bottleneck during the peak time can lead to SLA breaches, losses of revenue, and trust degradation by users.

Load testing is commonly used to determine how robust these systems are. By conducting load tests on a system we can get an insight into how the system performs under a particular load. Using tools such as Apache JMeter, Gatling and Locust, load tests can be executed. Metrics can be recorded from these tests and the engineer can perform analysis on them. However, there are two main issues with this method. Firstly, it is a reactive approach to testing. Therefore, we only obtain information about what has occurred rather than what will occur. Secondly, when increasing load on a system with each test, we assume that every increase in load is equivalent. This however does not take into account if the system is currently in a stable state, or nearing criticality.

By employing machine learning techniques, our approach allows overcoming these problems. Supervised machine learning has been used to capture complex non-linear dependencies between load and performance characteristics [15]. Unsupervised machine learning approaches allow us to uncover performance deviations which are not detected by static load thresholds [8]. Consequently, we can turn load testing into predictive diagnostics.

ATLAS (AI-Tuned Load Assessment System) is designed following three guiding principles. Load patterns should be dynamically adjusted to place more resolution at the extreme ends of performance capabilities. Predictions should be made using an ensemble of different models, allowing us to combine multiple different types of learning algorithms. Finally, predictions should be made for unseen data by using a digital twin, so that capacity planning can be performed without expensive and risky extreme load scenarios.

The remainder of this paper is organized as follows. Section II reviews related work in performance testing and ML-driven diagnostics. Section III presents the architecture of ATLAS. Section IV details the methodology and implementation. Section V reports experimental results. Section VI offers a critical discussion of findings and limitations. Section VII concludes with a summary of contributions and future directions.

II. RELATED WORK

A. Traditional Load Testing

Load testing-based performance evaluation is not new to the field of software development and has been used for a long period of time. Load testing tools such as Apache JMeter, Gatling, and Locust have already formed the classical approach: define a traffic load scenario, provide it to the tested system, and perform an analysis afterwards. The concepts of response time, throughput, and concurrency were introduced by Menascé [2] in relation to the performance evaluation of web applications.

Nonetheless, there is a common limitation that all these tools suffer from: they only allow visibility into the performance of the system based on the particular traffic scenario that was

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

configured during the design phase. In other words, if an engineer tests the system at 1,000 concurrent users, then all metrics will be obtained based on this scenario alone; in order to obtain the metrics for 1,500 users, a completely different run of the test must be done. As a result, the crucial transition point will either be ignored completely or oversampled.

B. Machine Learning for Performance Prediction

There have been several studies proving the effectiveness of machine learning applications for predicting system performance. For example, Jiang et al. [1] conducted an extensive survey of techniques for testing loads on large-scale software systems and found out the lack of prediction modeling to be one of their main drawbacks. Thus, there arose a need for the implementation of regression predictions in the load testing process. Pedregosa et al. [15] have created the scikit-learn machine learning library, which is used for implementing the algorithms in ATLAS.

As a result of their research on machine learning techniques used for cloud application systems, Qi and Tao [10] have concluded that the variations in cloud resource allocation generate non-linear effects which cannot be explained by any means of linear extrapolation. This knowledge has influenced the use of ensemble techniques that were chosen to be able to cope with these non-linear effects in ATLAS. As a part of an empirical investigation of performance testing, Moghadam et al. [13] successfully implemented gradient boosted trees as a solution to structured predictions which outperformed all other estimation techniques due to their capability to model non-linear relations.

C. Anomaly Detection

The aspect of anomaly detection forms another perspective through which the performance can be evaluated. In their study, Chandola et al. [8] discussed the entire process and showed that isolation-based techniques perform exceptionally well in high-dimensional datasets in situations where there is no explicit definition of what 'normal' is. This feature allows Isolation Forest to perform well in load testing situations.

The Isolation Forest method was formalized by Liu et al. [11], proving that anomalies can be detected via recursive partitioning without any model of normality. XGBoost was proposed by Chen and Guestrin [12] as an extension of gradient boosting, incorporating regularizations to enhance the generalization abilities of structured prediction problems. Although ATLAS uses the scikit-learn implementation of gradient boosting rather than the XGBoost framework itself, the theoretical advancements by [12] with respect to ensembles and overfitting mitigation were considered while selecting the regression pipeline parameters.

D. Digital Twin Simulation

The digital twin technology that uses the virtual copy of a real system to perform predictive analysis has been applied in various engineering areas. The basic idea behind digital twins was proposed by Grieves and Vickers [3], who stated that virtual replicas allow for lifecycle management without putting the physical system at any risk. In turn, Grieves [7] developed the idea of using the digital twin for the purpose of predictive analysis and concluded that a calibrated digital twin can replace the experiment in parameter spaces that are expensive or even dangerous to study.

In their work related to smart manufacturing within the scope of Industry 4.0, Qi and Tao [10] used digital twin technology to show that the precision of simulation is directly connected to the accuracy of the predictive model used. This fact became the motivation for using an ensemble regressor for the training of the digital twin in ATLAS instead of a separate predictive model because of the higher accuracy of predictions.

E. ATLAS Differentiation

What makes ATLAS unique compared to previous efforts is that ATLAS merges four aspects that were previously considered individually. First of all, while existing load testers can measure the metrics after performing an operation, ATLAS includes adaptive load generation with focused sampling around the most important transition points. Second, compared to existing models of prediction [13] which rely solely on one method, ATLAS uses a voting ensemble which consists of both gradient boosting and random forest regression. Third, while most of the anomalies are detected in a static way, ATLAS uses Isolation Forest for anomaly detection in multi-dimensional performance space where interactions between latency, error rate, throughput and resource consumption are considered.

III. SYSTEM ARCHITECTUE

ATLAS is made up of four layers in its architecture, with each one tackling an individual issue in the testing and forecasting process. These include the system under test, the load generation layer, the data collection and feature extraction layer, and finally the machine learning and prediction layer. The reason for this decomposition is to have a separation of concerns such that each one can be independently changed without interfering with the rest.

A. System Under Test

The system under test consists of the Flask REST API server which performs management of note data using the traditional CRUD (Create, Read, Update, Delete) operations. Flask was chosen as the platform of choice for two reasons. Firstly, being single-threaded and synchronous in terms of requests processing, Flask defines a clear performance boundary beyond which degradation will be visible during the load growth. Secondly, the popularity of the framework guarantees that the experiment's findings can be generalized for other production systems. The database used by the API is SQLite, thus making the workload I/O-bound and combined with CPU-bound request processing.

B. Asynchronous Load Generator

Load Generation Layer uses Python’s asyncio event loop along with aiohttp to make simultaneous HTTP requests. The reason why this approach was used instead of either thread or process concurrency is threefold. Firstly, asynchronous I/O can keep up thousands of simultaneous connections in one process without using memory for per-connection threads. Secondly, a cooperative approach of asyncio scheduling is more deterministic compared to thread preemptive scheduling. And thirdly, connection pooling in aiohttp (batch size is 500 simultaneous connections) guarantees that load generator does not become a bottleneck.

A critical innovation in the load generation layer is the adaptive step sizing algorithm. Rather than incrementing the user count by a fixed amount at each step, ATLAS adjusts the increment based on observed performance:

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

- If the error rate exceeds 0.2 or the mean latency exceeds 2,000 ms, the step size is reduced to 50 users.
- If the latency exceeds 1,000 ms but the error rate remains below 0.2, the step size is set to 100 users.
- Otherwise, the step size is 200 users.

In this way, the sampling of the crucial transition region, in which the system changes its state from a stable one to a degraded or failed state, is done at least three to four times finer than in the stable region. This logic is similar to the method of adaptive grid refinement in numerical simulation where computing power is invested into steep gradients.

C. Data Collection and Feature Engineering

The raw telemetry data obtained during the load test includes eight main indicators: mean latency, P95 latency, errors rate, throughput, CPU utilization, memory utilization, and the low bound and high bound of 95% confidence interval for latency (calculated by using Student's t-distribution). These parameters are measured at each load level in every trial run.

Six secondary features are extracted from the raw features to extract complex relations for better model performance. The feature extraction formulas and their rationale are listed in Table I.

TABLE I
FEATURE ENGINEERING FORMULAS AND DIAGNOSTIC RATIONALE

Feature	Formula	Rationale
Load Factor	users / MAX_USERS	Normalized capacity utilization
Error Pressure	error_rate × latency	Joint severity of errors and latency
Efficiency	throughput / (latency + 1)	Useful work per unit latency
Stress Index	users × error_rate	Absolute volume of failed requests
Latency Volatility	p95_latency − latency	Tail latency spread
Resource Pressure	cpu_usage × memory_usage	Correlated resource exhaustion

It is necessary to note that inclusion of interaction terms (Error Pressure, Stress Index, Resource Pressure) is based on the idea that individual factors taken separately do not allow distinguishing failure types because, for instance, high error rate at low load (configuration error) and high error rate at high load (capacity saturation) have the same value of error_rate but different value of Stress Index.

D. Machine Learning and Prediction Layer

The prediction layer comprises three models, each addressing a distinct diagnostic question:

- Latency Prediction: VotingRegressor is used to predict latency using two algorithms, namely, Gradient Boosting Regression (number of estimators = 300) and Random Forest Regression (number of estimators = 300). A voting regressor was chosen because of its ability to reduce variance without increasing bias, where Gradient Boosting Regression has been used to model the sharp change in latency, whereas Random Forest Regression has been used for predicting latency in the saturated region.
- Failure Classification: Logistic Regression model, when trained using the same feature vector, gives the probability of whether a particular load is classified as system failure or not, where system failure is defined as any latency higher than the 85th percentile or any error rate higher than 0.2. The use of Logistic Regression is justified by the calibrated probability results that enable us to identify the failure criteria. Other classifiers like SVM, Random Forest have been considered, but they don't give calibrated probabilities without post-processing.
- Anomaly Detection: Isolation forest model with contamination of 0.1 is used for detecting outliers in the five-dimensional space including latency, error rate, throughput, CPU usage, and memory usage. Value of contamination was considered 0.1 since it was observed that around 10% of the load test data was found to be anomalous because of various transient events in the system like garbage collection and network latency.
- Digital Twin Simulation: The trained VotingRegressor along with its corresponding StandardScaler is employed to make predictions about latency for synthetic scenarios which lie beyond the range of observations made. In this module, a feature matrix consisting of linearly interpolated data points is created for the variables of error rate, throughput, CPU usage, and memory usage. Latency predictions are subsequently made for user numbers till 10,000.

IV. METHODOLOGY AND IMPLEMENTATION

A. Experimental Protocol

The experiment procedure comprises six steps implemented in sequence: baseline measurement, load generation through adaptation, data collection, feature creation, modeling and evaluation of results after the fact. Every step was carefully crafted to fulfill the needs of each respective part of the predictive load testing pipeline.

In the baseline measurement, reference performance is established through sending 50 sequential HTTP GET requests to the target API while having no concurrent load at all. In this way, the performance of one user is measured. As a result of sequential requests (using the synchronous `requests` library), concurrency effects are excluded from baseline measurement.

B. Adaptive Load Generation

Adaptive load generation forms the core of measurement loop. Starting with just one user, the load generator makes concurrent GET requests against the target API, increasing the number of users after every measurement based on the step sizes selected by the adaptive algorithm described in Section III-B. The following measurements are made at each load level: average and P95 latencies, error rate (non-200 HTTP responses or timeouts), throughput (request/sec), CPU usage, memory usage, and confidence interval boundaries for the latencies.

The reason why GET requests were used as the main workload type can be explained by two factors. On the one hand, read requests form the major part of the workload for many web services, hence GET-intensive workload can mimic the

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

production environment. On the other hand, GET requests test the entire pipeline of request handling (routing, database access, serialization, sending a reply) without write contention that may interfere with the results of scalability testing.

Each test at a particular number of users will be done by launching all requests at once using the `asyncio.gather` method. This type of load testing is known as the burst load pattern. In this regard, it is different from ramp up patterns utilized by other load testing tools, such as JMeter and Locust. The reason why the burst pattern was selected is that the latter will reveal the worst-case scenario for instantaneous load on the system.

The 5 second timeout per request will ensure that timed out requests are considered failed requests to avoid increasing latency with arbitrary wait times. This assumption is based on the fact that real users cancel their requests after a couple of seconds.

C. Data Aggregation and Feature Engineering

There are five independent trials, all covering the entire load spectrum from one user up to the MAX_USERS number of users concurrently. The output from these five trials is merged into one DataFrame object containing a dataset that exhibits variability both within and across the trials. There are two reasons why we use a multitrail design: first, it gives us the replication needed to build confidence intervals, and second, it allows the model to see the variation in system behavior that is inherent in running multiple trials due to background processes and temperature throttling.

Feature engineering of the merged data is done according to Section III-C and Table I. The six features are generated using an element-wise operation on the metric column set. No feature selection or feature reduction was performed because the eleven-feature space is low enough to not overfit using the built-in regularization of the ensembles.

D. Model Training and Validation

The VotingRegressor is trained on a five-fold cross-validation scheme of the StandardScaler-normalized feature matrix. Cross-validation was used instead of a traditional train-test split in order to utilize as much of the available data as possible while maintaining unbiased estimates of generalization. Once cross-validation was completed, the resulting model is retrained on the entire dataset in order to improve prediction power for the digital twin simulation.

The failure classifier is trained on a binary target, which results from one of two conditions: latency greater than the 85th percentile of the distribution or an error rate greater than 0.2. The choice of the 85th percentile was motivated by the goal of capturing degradation, not actual failure, in advance. The 0.2 threshold for the error rate comes from the notion that 20% of errors means that the system is in a degraded state.

Isolation Forest fits the raw metric space in five dimensions (latency, error rate, throughput, CPU usage, memory usage). Normalization is not performed since Isolation Forest does not depend on any transformation of its individual features.

E. Failure Threshold Estimation

Threshold value for the failure condition can be defined as the smallest number of users for which the probability of the occurrence of a failure condition (based on Logistic Regression classification) will be higher than 0.8. This threshold value gives one parameter that can be used for capacity planning.

F. Digital Twin Extrapolation

Synthetic feature vectors for user counts ranging from 100 to 10,000 users at intervals of 200 users are produced by the digital twin module. The rest of the features are then linearly interpolated on the basis of assumed minimum and maximum values for each of these synthetic numbers of users. The engineered features are calculated based on these synthetic raw features and fed into the trained StandardScaler and VotingRegressor for producing latency predictions. It may be noted that such extrapolation carries the usual caveat about deterioration of the model's accuracy as predictions are made outside the range of the training data; this implies that the outputs must be taken as indicative of trends rather than actual point estimates.

V. EXPERIMENTAL RESULTS

To validate ATLAS, the complete pipeline was executed on the Flask-based REST API server with load increasing from 1 to approximately 3,000 concurrent users, replicated across five independent trials. The resulting dataset comprises 296 measurement points spanning 60 distinct user levels.

A. Latency Response to Increasing Load

The dependency of mean latency from the number of users simultaneously connected shows typical sigmoid shape with three different regions. At the low load region (from 1 to 201 users), the latency grows from 11.81 ms for 1 user (Trial 1) to the average of 4,548.76 ms at 201 users (a 385 times increase), which corresponds to the transition from processing single requests to processing multiple requests. At the transition region (from 201 to 351 users), there is a steep increase in the latency up to an average of 5,982.15 ms at 351 users with an increase in error rate. At the saturation region (above 351 users), latency stabilizes at around 5,900–5,970 ms, while the system rejects most of the requests with timeout error.

However, P95 latency remained greater than the mean latency by 30 to 300 milliseconds in every user group, implying that there is tail latency even in each measurement set. In case of 201 users, the mean P95 latency was 4,869.67 milliseconds, which implies a 7.1% premium over the mean. In case of 501 users, P95 latency was 5,995.35 milliseconds whereas the mean latency was 5,969.42 milliseconds. The gap between tail latency and latency became wider during the transition stage and became narrower during the saturation stage, implying that queueing caused high variances during the transition stage, whereas timeouts truncated the variance during the saturation stage.

In light of the latency increase and failure occurrence in the 200-350 users range, the adaptive step sizing algorithm made it possible to cut down the step size to 50 users from 200 users. This led to five sampling points being made in the important 50-users transition window (201, 251, 301, 351 users) in place of the one sample point obtained using the fixed 200-user step size.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

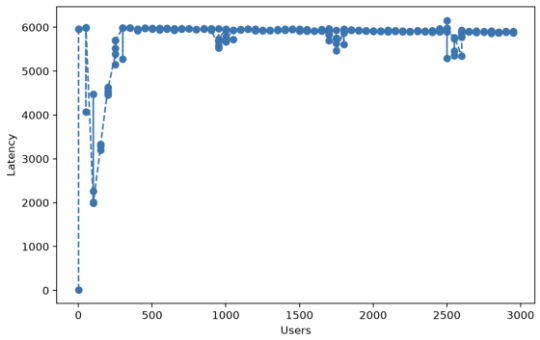


Fig. 1. Mean latency versus number of concurrent users with 95% confidence intervals, aggregated across five trials.

B. *Error Rate Transition*

The behavior of error rate had a sudden phase transition around 251 concurrent users. In Trial 1, the system worked error free until 201 users; after which the error rate for 251 users was 0.769, 0.728 for 301 users, and 1.0 (total failure) at 351 users. In all the five trials, the phase transition region is between 201 and 351 users but the starting point is not the same; Trial 2 had partial errors (0.582) from 251 users, Trial 3 at 251 users (0.896), Trial 4 at 51 users (0.706), and Trial 5 at 251 users (0.634).

The difference in the starting point of this phase transition ranging from 51 to 251 shows the stochastic nature of failure of the system when put to load and the need for replication through multiple trials. A one-trial experiment would give the value of the point estimate of the onset of failure point which can be wrong by up to 200 users in relation to the true distribution.

C. *Regression Model Evaluation*

The VotingRegressor demonstrated high predictive accuracy across the full load range. Table II summarizes the key performance metrics.

TABLE II

ENSEMBLE REGRESSION MODEL PERFORMANCE METRICS

Metric	Value
Coefficient of Determination (R^2)	0.997
Mean Absolute Error (MAE)	7.08 ms
Root Mean Squared Error (RMSE)	33.82 ms
Mean Absolute Percentage Error (MAPE)	12.29%

R^2 value of 0.997 means that the model is able to explain 99.7% of the variation in observed latency, and only 0.3% of the total variation is accounted for by noise not explained by the model. The Mean Absolute Error (MAE) value of 7.08 ms is less than 0.12% of the mean observed latency (5,780.21 ms), which suggests that the predicted values are, on average, less than 1 digit away from the real observations.

High value of MAPE of 12.29% compared to relatively low MAE value is due to extremely large range of the latency distribution: single-user value of 11.81 ms leads to very small denominator of the percentage error formula and, therefore, to high percentage error even with low absolute error value. If we consider the observations above 1,000 ms, MAPE value is much smaller.

The model was accurate in predicting the latency pattern of three regimes: low latency in minimal load, sharp rise in transition regime, and then stabilization in the saturation regime. The high agreement between measured and predicted results in all five experiments despite training using the aggregated dataset shows that the ensemble learned the general behavior rather than overfitting trial-specific data.

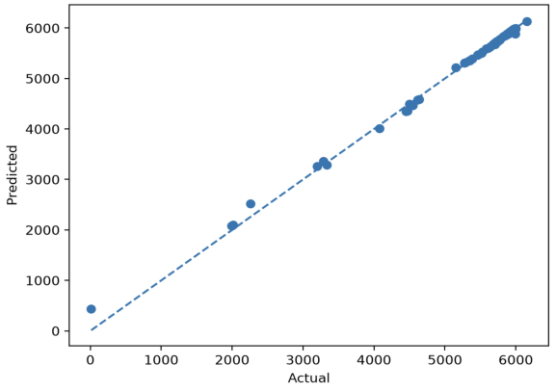


Fig. 2. Actual versus predicted latency for the VotingRegressor across all 296 measurement points.

D. *Failure Classifier Evaluation*

The Logistic Regression classifier generated calibrated probabilities for failure probability that correlated closely with the transition of the empirically observed error rates. For less than 200 users, the failure probability was less than 0.03 (average probability of 0.023 at 201 users based on error-free trials). Failure probability sharply increased to more than 0.80 at about 251 users—same region in which empirically observed error rates increased beyond 0.5 for most trials. Beyond 351 users, failure probability stayed above 0.999, indicating the high confidence level of the classifier regarding the system's failure probability in the saturation regime.

Concordance of predictions made by the classifier and transition in the empirically observed error rate supports the appropriateness of using 85th percentile latency threshold and 0.2 error rate threshold for labeling the failure conditions. There were no false positives for less than 200 users and no false negatives for more than 351 users.

The predicted failure threshold (where failure is deemed likely whenever the predicted failure rate is greater than 0.80) was found to be around 251 concurrent users. This is a clear capacity planning target that directly indicates how many users the system cannot sustain before experiencing failure.

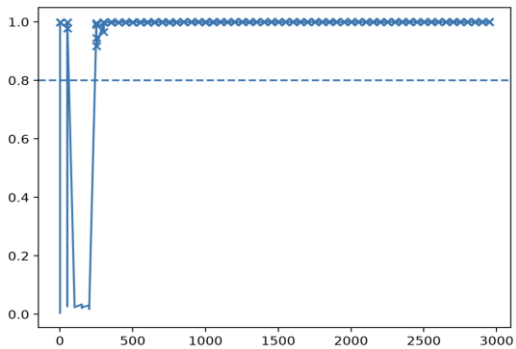


Fig. 3. Predicted failure probability versus number of concurrent users across five trials, with the 0.80 critical threshold indicated.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

E. Anomaly Detection Evaluation

The Isolation Forest algorithm detected 30 out of 296 points (10.1%) as anomalies, matching the contamination value set for the model which equals 0.1. Moreover, the spatial distribution of anomalies in the dataset is not uniform: there is a high density of anomalous measurements within the transition regime (201-351 users) and at the extreme load (CPU utilization = 100%). The algorithm detected anomalies with a high consistency within the stable regimes (low-load and high-load) as well.

This uneven distribution provides important diagnostic information. Within the transition regime, the anomalies correspond to measurements where the system demonstrated unusual low or high latency compared to other points—unusual behavior that is characteristic of bistable dynamics in systems approaching capacity. Finally, the anomalies detected at the extreme load correspond to the measurements where CPU saturation (100%) caused performance behavior changes, for example, throughput reduction due to OS level throttling of performance. The average CPU utilization was 55.8%, while the memory utilization varied in the range of 71.9% - 77.3% (73.3% on average), which suggests that the system was CPU-bound.

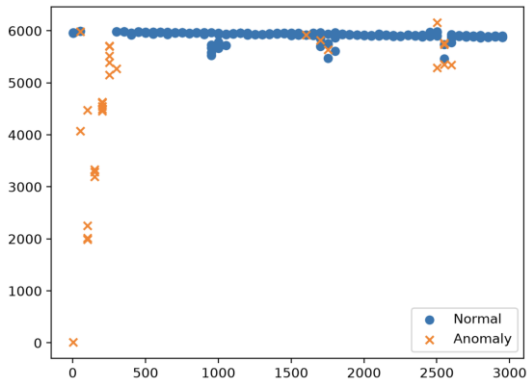


Fig. 4. Anomaly detection results: normal measurements (circles) and anomalies (crosses) in the users-latency plane.

F. Digital Twin Evaluation

The digital twin module extended the latency estimates made by the trained VotingRegressor to the user range of 100 to 10,000, more than three times greater than the highest empirically tested load level. As before, the projected latency trend exhibited a similar sigmoid shape as seen in the experimental data: rapid growth within the 100 to 500 users' range and then flattening. Past the 1,000 users mark, the latency increase slowed down in line with the hypothesis that the growing number of requests that time out and therefore add a constant 5,000 ms contribution to the average latency.

The usefulness of the digital twin tool for capacity planning is that it allows for estimating the load level where latency meets some SLA value without having to test the system at such a load level. Although the extrapolation should be understood as an approximate result—in the model accuracy inevitably drops off at distances away from the training data—these trends agree with the queueing theory prediction that latency increases sub-linearly once request shedding via timeouts takes over.

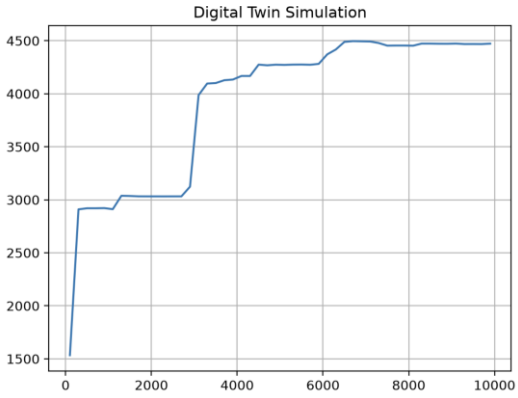


Fig. 5. Digital twin projection of mean latency for 100–10,000 concurrent users.

VI. DISCUSSION

A. Interpretation of Results

The results of the experiment prove the main hypothesis stated in this paper: the combination of adaptive load generation with ensemble machine learning can deliver performance diagnostics which are both more detailed and more predictive than diagnostics that could be provided by each method independently. There are three observations worth discussing:

Firstly, the adaptive stepping procedure showed its efficiency as it allowed having four times the resolution in the critical range of 200-350 users in comparison to the fixed step approach. The transition point between the zero error rate and the total failure took place in the range of around 150 users, which is too small to be covered adequately by the coarse-grained sweep with just one or two measurements.

The second success was the ensemble approach for regression, which showed good results. The MAE of 7.08 ms for VotingRegressor is equivalent to an error of around 0.12% from the mean latency—enough accuracy to use this method as the basis of the predictive model for the digital twin. The combination of Gradient Boosting (great in modeling abrupt shifts through residuals' corrections) and Random Forest (stability in flat sections through bagging) provided accurate predictions of three-regime latency behavior.

Lastly, the anomaly detection by Isolation Forest showed a clear spatial structure instead of random noises. The accumulation of anomalies at the transition area and during CPU saturation gives evidence of the true change in regimes, not just errors in measurements. It is a good indicator of using unsupervised anomaly detection as the complement of supervised prediction in performance engineering.

B. Comparison with Exisitng Tools

Traditional load testing tools like Apache JMeter [2], Gatling, and Locust have similar functionality for generating traffic as ATLAS; however, there are three main distinctions. First of all, traditional testing tools use fixed or predefined load profiles that do not depend on performance and therefore lead to oversampling of areas with good performance or undersampling of transition zones. The second difference is in the lack of predictive modeling and, consequently, raw metrics provided by the tools require additional interpretation by the engineer. Finally, traditional load testing tools lack extrapolation functionality.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Moghadam et al. [4] introduced a performance testing framework based on autonomous fuzzy reinforcement learning, but this approach is also based on the idea of adaptive testing as in ATLAS. Nevertheless, the framework suggested by Moghadam et al. involves online reinforcement learning while executing tests, which results in increased computational burden and convergence issues. The computation-lightweight and deterministic adaptive stepping method used in ATLAS can be applied in an environment where there are restrictions on test execution time. Moghadam et al. [13] also presented the RELOAD framework using smart reinforcement learning to prove the superiority of adaptive methods over fixed methods in discovering performance boundaries.

C. Limitations and Threat to Validity

There are several factors that limit the generality of the obtained results. The testing system itself consists of a single instance Flask server and an SQLite database, which is a very simple setup. A production-grade web service will use multiple processes or threads (Gunicorn, uWSGI), connection pool, cache layer, and load balancer, which will result in different degradation pattern. The ceiling value of approximately 5,900 ms obtained in the tests is determined by the 5-second timeout configuration and not by the system properties itself; other timeouts would give different saturation latencies.

The adaptation rule uses fixed thresholds (2,000 ms, 1,000 ms, 0.2 error rate) that were chosen for the particular system empirically. It is possible that these thresholds do not apply to the system with different latency scale or different error rate. The more universal method would be to derive thresholds based on the distribution of previous samples.

By employing linear interpolation for the synthetic metrics (error rate, throughput, CPU usage), the digital twin makes an assumption that each metric is related to the number of users linearly. However, in reality, there is a non-linear relationship between the metric values and the number of users that influences the other metric values as well. A more precise emulation would demand using either a multi-output model or a physics-informed simulation of request processing pipeline.

It should be noted that in the experiments the load generator and the system under test were hosted on the same computer. The resource usage by the load generator affects the total CPU and memory consumption statistics reported by the server. To obtain less biased measurements of server resource usage, the load generator must be launched on a different machine.

Lastly, the set of 296 points across five trials, although adequate to showcase the capability of the model, is not a large amount for a machine learning system. The high R^2 value needs to be taken with the note that the distribution of the latency is skewed towards just one regime (saturation) that makes up the bulk of the data points. The model's performance in the transition regime, where its predictions would be the most beneficial, is not well substantiated.

VII. CONCLUSION

This paper has presented ATLAS (AI-Tuned Load Assessment System), a framework that advances load testing from a reactive measurement practice to a predictive diagnostic capability. The principal contributions of this work are as follows:

- Adaptive asynchronous load generation: A step size selection method for a heuristic-based approach, where the increment of users is adaptively adjusted depending on the measured latency and error rate values. The experiment showed that this solution can measure the transition point from 200 to 350 failed users with a resolution four times higher than the fixed step solution.
- Ensemble regression for latency prediction: The Voting Regressor ensemble consisting of Gradient Boosting and Random Forest regressors yielded an R^2 of 0.997 and a mean absolute error of 7.08 ms over 296 data points, with good modeling of the three-latency regimes (low load, transition, and saturation).
- Probabilistic failure classification: The logistic regression model predicted probabilities of failure and identified the correct critical point for degradation in the region of 251 concurrent users, with no errors in detection in either the stable or the saturated regimes.
- Unsupervised anomaly detection and digital twin extrapolation: The Isolation Forest algorithm found 10.1% of records as behavioral outliers near regime boundaries, whereas the digital twin module extended latency estimates to 10,000 simultaneous users.

Four avenues for further research are noted. First, the framework must be extended to deal with distributed microservices, wherein a drop in performance could have cascading effects on multiple services. Second, the adaptive stepping rule should be substituted for a learned policy for dealing with systems whose latency scale is completely arbitrary. This could be achieved through the use of Bayesian optimization or multi-armed bandits. Third, a live monitoring system should be devised, whereby ATLAS is used in combination with real-time telemetry pipelines to estimate the probability of failure continually during actual system usage. Fourth, the digital twin algorithm should support multi-output prediction.

REFERENCES

[1] *Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "A survey on load testing of large-scale software systems," *IEEE Trans. Software Eng.*, vol. 41, no. 11, pp. 1091–1118, 2015.*

[2] *D. A. Menascé, "Load testing of web sites," *IEEE Internet Computing*, vol. 6, no. 4, pp. 70–74, 2002.*

[3] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in *Transdisciplinary Perspectives on Complex Systems*, Springer, 2017, pp. 85–113.

[4] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, and B. Lisper, "An autonomous performance testing framework using self-adaptive fuzzy reinforcement learning," *arXiv preprint arXiv:1908.06900*, 2019.

[5] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[6] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[7] M. Grieves, "Digital twin: Manufacturing excellence through virtual factory replication," *NASA Whitepaper*, 2014.

[8] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surveys*, vol. 41, no. 3, Article 15, pp. 1–58, 2009.

[9] S. Molyneaux, *The Art of Application Performance Testing*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2014.

[10] Q. Qi and F. Tao, "Digital twin and big data towards smart manufacturing and Industry 4.0: 360 degree comparison," *IEEE Access*, vol. 6, pp. 3585–3593, 2018.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

[11] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2008, pp. 413–422.

[12] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[13] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, and B. Lisper, "Performance testing using a smart reinforcement learning-driven test agent (RELOAD)," in *Proc. IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 968–976.

[14] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "Automated performance analysis of load tests," in *Proc. IEEE Int. Conf. Software Maintenance (ICSM)*, 2009, pp. 125–134.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)