MLOLET - Machine Learning Optimized Load and Endurance Testing

An industrial experience report

Arthur Vitui*
RedHat Inc.
Montreal, Quebec, Canada
avitui@redhat.com

Tse-Hsun (Peter) Chen
Software PErformance, Analysis and Reliability (SPEAR) Lab
Concordia University
Montreal, Quebec, Canada
peterc@encs.concordia.ca

ABSTRACT

Load testing is essential for ensuring the performance and stability of modern large-scale systems, which must handle vast numbers of concurrent requests. Traditional load tests, often requiring extensive execution times, are costly and impractical within the short release cycles typical of contemporary software development. In this paper, we present our experience deploying MLOLET, a machine learning optimized load testing framework, at Ericsson. MLOLET addresses key challenges in load testing by determining early stop points for tests and forecasting throughput and response time trends in production environments. By training a time-series model on key performance indicators (KPIs) collected from load tests, MLOLET enables early detection of abnormal system behavior and provides accurate performance forecasting. This capability allows load test engineers to make informed decisions on resource allocation, enhancing both testing efficiency and system reliability. We document the design of MLOLET, its application in industrial settings, and the feedback received from its implementation, highlighting its impact on improving load testing processes and operational performance.

ACM Reference Format:
Arthur Vitui and Tse-Hsun (Peter) Chen. 2024. MLOLET - Machine Learning Optimized Load and Endurance Testing: An industrial experience report. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3691620.3695258

1 INTRODUCTION

Today's modern large-scale systems need to handle a very large number of concurrent requests. Any malfunctions or service degradation resulting from a spike in the load may cause companies losses in the millions or even billions of dollars [10]. Therefore, it is crucial for the IT operations teams to be able to load test the system to ensure its performance and monitor the production system to ensure smooth operation.

Understanding system behavior under varying loads is typically achieved through load testing and endurance testing. These tests yield critical metrics, such as the number of requests processed per unit time (e.g., seconds) and the corresponding average response times. Such data is invaluable for load test engineers in defining service-level agreements and determining the optimal system configurations, including the number of nodes required to handle the anticipated load effectively. Once systems are deployed in production, load test engineers must collaborate with IT operators to continuously monitor the system. This ongoing vigilance is necessary to identify and respond to sudden spikes in load, such as a surge in requests, by potentially increasing computing resources to maintain performance and stability.

However, there are two main challenges in running such load tests and monitoring production system performance. ***First, load tests often require extended execution times to accurately assess system behavior***. In the context of modern software development, where release cycles are typically only a few weeks, conducting long-running load tests can be costly and time-consuming, especially if the load test engineers need to test the performance under different configuration settings. Therefore, if a load test is likely to fail, it is more economical to terminate the test early. ***Second, it is crucial for load test engineers to forecast the system's performance trends in production***. Accurate forecasting enables engineers enough time to take proactive measures, such as increasing the number of nodes, to ensure the system maintains optimal performance and stability during sudden load spikes.

In this paper, we present our experience deploying a time-series-based load testing framework, MLOLET, developed to assist load test engineers at Ericsson. MLOLET addresses two main challenges: 1) determining early stop points for load tests and 2) forecasting throughput and response time trends in production. For both cases, we focus on detecting and forecasting spikes in KPIs. Detecting an excessive number of spikes during a load test may mean the test is failing and needs to be stopped (i.e., spike detection), while forecasting spikes in production helps take preventative actions by adding more computing resources (i.e., spike forecasting). We train a time-series model using the key performance indicators (KPIs) collected periodically from load tests conducted with a limited set of configurations. This model is then applied to load tests with

*Arthur Vitui is a Senior AI Specialist Solutions Architect at RedHat Canada. The work described in this paper was done when Arthur was a Senior Solution Architect at Ericsson, Canada.

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695258

varying configurations to detect abnormal system behavior, such as significant spikes in certain KPI values over time. We also apply the model to monitor and forecast spikes in production environments under different configurations.

This experience report is a result of working on a large-scale mission-critical system developed by Ericsson Inc, which handles millions of concurrent users around the world everyday. During the short six-week release cycle[1], load test engineers at Ericsson need to assess the robustness of the software ecosystem by observing the system's behavior under specific loads. The systems at Ericsson are composed of several components with millions of lines of code, featuring complex configurations, including functional, performance, and environmental settings. These components can run under the same application server or be distributed across multiple compute nodes, with configurations that may include local and geographical redundancy. Due to the scale and complexity of these systems, it is challenging to conduct comprehensive load and endurance tests within the limited six-week release cycle. Moreover, covering different combinations of configuration parameters is nearly impossible due to their complexity and sheer number. Hence, by leveraging MLOLET, load test engineers can enhance the efficiency and effectiveness of load testing, ensuring that systems are robust and capable of handling dynamic load conditions under different configurations.

Due to the non-disclosure agreement (NDA), we can only disclose limited details about the experiment results and about the functionality of the enterprise system. We discuss, however, the techniques and algorithms we used for spike detection as well as for system throughput and response time trend forecasting. We also discuss the data processing pipeline, including the hyper-parameter tuning case studies. For reproducibility, we conducted experiments on an open-source system, with setup guides and source code available in a GitHub repository [28].

In summary, the paper makes the following contributions:

- We discuss the challenges our industrial partner encounters during load testing and share the design of our approach to resolve these challenges.
- We present the evaluation of MLOLET on both an industrial system and an open-source system. We also share our experience with our industrial partner's adoption of MLOLET.
- We review various open-source technologies that practitioners can use to implement a load testing framework similar to MLOLET.

We hope that MLOLET's blueprint provides valuable insights to both researchers and practitioners. We also hope that practitioners may adopt MLOLET to help improve their load testing processes.
Paper organization. Section 2 describes the studied system and the case study setup. Section 3 describes our methodology. Section 4 presents the results. Section 5 discusses the lessons learned and our experience from the industrial environment. Section 6 discusses threats to the validity. Section 7 discusses related work. Section 8 concludes the paper.

[1]https://www.Ericsson.com/en/press-releases/2017/9/Ericsson-offers-continuous-software-updates

2 STUDIED SYSTEMS

In this section, we present the studied enterprise system and the available metrics that can be monitored to implement our proposed framework, within the limits of the NDA boundaries. Due to NDA restrictions, we also test our hypothesis on an open-source system.

2.1 Studied Industry System

The evaluated industry system contains many components and is maintained by a large number of software developers. The system provides Business-to-Business (B2B) and Business-to-Consumer (B2C) messaging functions and has integrated Ericsson's in-house developed products, open-source software, and third party commercial products. As the provided functions are business critical, the system is designed for high availability with local as well as geographical redundancy across several datacenters. The system is used on a daily basis by millions of users around the world and processes tens of millions of requests.

2.2 Studied Open Source System

To verify our findings and ensure reproducibility, we also conducted our experiments on an open-source software system. For data generation purposes, we use a custom-developed load generator, for which the source code and description are available online [25]. We created our custom load generator to have better control over the load, including spike-emitting capability and the collection of near-real-time metrics.

The test subject is a WireMock [45] mock application extended with the Prometheus [30] metric generator and a global random string payload ResponseTransformer. Depending on the deployment configuration, the test subject application may recursively call itself multiple times. This approach generically models numerous business applications using a black box method, reflecting their common characteristics: receiving a request, processing it, and returning a response. The source code and full description for the test subject application are available online [46]. The test results related to MLOLET and the Machine Learning model prototypes used by MLOLET are also available online [28].

2.3 Measuring System Performance

In general, many metrics are available to measure a complex system's performance during load testing. Common performance metrics include CPU, memory, and disk usage and application-specific metrics such as throughput per unit of time (transactions per second—TPS and transactions per minute—TPM) and the associated average component response time for the selected unit of time. Due to the nature of the industry system, we measure system performance using two primary metrics: 1) **Throughput:** How many requests the software component can process within a desired time unit. 2) **Response Time:** How fast the software component processes those requests.

In this paper, we focus on modeling spike detection (both upward and downward) and trend forecasting for throughput and processing request response time as our main performance metrics. These metrics are generic system capacity indicators applicable to a wide range of systems. They also depend on the system's internal and hardware configurations, providing a comprehensive view of the

overall system performance generally applicable to many software systems.

3 THE MLOLET FRAMEWORK

In this section, we present MLOLET (Machine Learning Optimized Load and Endurance Testing), which aims to assist our industry partner with the load testing process by addressing the previously mentioned challenges.

To address the first challenge (i.e., long running load tests), we propose detecting spikes in the load test data using an online detection approach and acting upon them as quickly as possible. The approach needs to detect spikes on the fly, without having access to the entire test result, to stop a test early. A load test may fail for several reasons. One such case is when the system's response time is outside of the normal distribution, within a given period. If the response time is much higher than usual, it indicates the system is taking more time to process information, suggesting an overload that could lead to an imminent system crash. Conversely, if the response time is much lower than usual, it could mean another system within the business processing flow has failed and is no longer accessible, thus shortening the measured component response time. This does not necessarily mean the measured component is failing; however, the overall business logic may be compromised. Having this insight allows load test engineers to stop failing tests, saving time and cost in the testing process.

To solve this challenge, we work with our industry partner to develop a configurable component in the load testing practice that provides the following functions:

- Alert the load testing team (via email or on a central monitoring system dashboard) if the number of spike events in the load or endurance test exceeds a certain threshold.
- Stop the load or endurance test if the number of spike events in the test exceeds a certain threshold.
- Begin a new load testing cycle if the number of spike events in the test exceeds a certain threshold.

It is noteworthy that counting the number of spike events may be global (meaning over the entire duration of the load test) or windowed (meaning that the counter is reset after a certain time has passed).

To address the second challenge (spike forecasting), we propose a time-series based load testing framework that:

- Forecasts the trend of the system in an on-line setting.
- Increases the efficiency and project economics of the load testing team.
- Improves the operational performance and economics of the operations team with regards to service up-keeping and associated SLA fulfillment.

Next, we describe the design of MLOLET in details.

3.1 The Overall Process and the Building Blocks of MLOLET

The industry system is composed of multiple interconnected components, and the components may be tested independently or jointly (depending on the applicable user stories and use cases). Hence, to increase the efficiency and reproducibility of the tests, we need a base framework for data collection and analysis that is portable and flexible to adapt to the various software components (between environments and projects). Figure 1 depicts the overall process used in our setup.

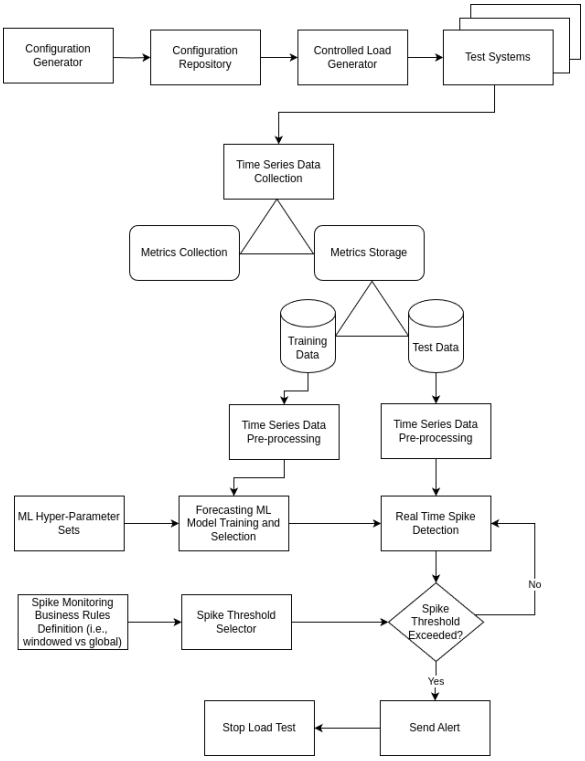

Figure 1: MLOLET framework overview.

One of the most important building blocks of the framework is the 'Real Time Spike Detection' one. The pseudo-code for this procedure is detailed in Algorithm 1.

Several different tools are available for each stage of the process blueprint. Due to the NDA with our industrial partner, we cannot disclose which tools were used in each phase. Nevertheless, we provide a general depiction of the process and suggest tools that may be used at each stage.

The configuration generator is a component-specific tool that may be written by software engineers (testers and/or developers alike) using any available scripting or programming language (e.g., C/C++, Python, bash-scripting, Java, Ruby). Its purpose is to provide a base set of configurations. The generated configurations may be stored in a repository, which may be a database (relational or NoSQL), a git repository, or a simple folder with text-like documents (YAML, CSV, XML, or JSON formatted documents).

Subsequently, the controlled load generator will use the configuration repository to select available configurations, apply them to the tested components, and then initiate a load test. Automation is key in this case, and several specialized tools are available to accomplish the necessary steps. For example, specific component configuration settings (i.e., the parameters of the tested software

Algorithm 1 Real Time Spike Detection

Require: Y_pred_train ▷ The list of predictions on the train data
Require: Y_true_train ▷ The true values of the train data
Require: $error\_selector$ ▷ The error function selector
Require: y_pred_test ▷ The prediction of the test/new data
Require: y_true_test ▷ The true value of the test/new data
Require: $is\_train\_configuration\_flag$ ▷ Is 1 if the test values are from the training set

```
function CALCULATE_ERROR(Y_true, Y_pred)
    if error_selector is Absolute_Error then
        errors ← | Y_true − Y_pred |
    else if error_selector is Squared_Error then
        errors ← (Y_true − Y_pred)^2
    return errors
errors ← CALCULATE_ERROR(Y_true_train, Y_pred_train)
err_mean ← mean(errors)
std_deviaton_err ← std(errors)
up_threshold ← err_mean + 3 * std_deviation_err
low_threshold ← err_mean − 3 * std_deviation_err
repeat
    deviation ← CALCULATE_ERROR(y_true_test, y_pred_test)
    if is_train_configuration_flag ≠ 1 then
        deviation ← deviation + std_deviation_err
    if low_threshold ≤ deviation ≤ up_threshold then
        spikes ← unchanged
    else
        spikes ← increased
until there is new data.
```

component) and component scalability values within the test environment may be controlled by specialized tools such as Ansible [1], Chef [5], or Puppet [31], or in-house automated scripts or configurator applications. Selecting the right tool for the job depends on the specific behavior of a software component regarding reconfiguration: some software components may expose a reconfiguration interface (e.g., JMX interface) where they accept commands that lead to the internal parameter setting. Other software components require setting specific values in a configuration file and may need a restart (if the configuration is not automatically reloaded upon a detected file change).

The load testing step may be accomplished using either custom-developed applications or specialized tools such as JMeter [20], Jenkins [19], Soap-UI [37], etc. To ensure the system is as portable as possible, we collect the desired metrics – both the computing nodes' resources and the application's processing metrics (throughput per unit of time and associated average response time)–in a non-intrusive fashion. Such methods include log parsing and aggregation using tools such as LogStash [26] or ElasticSearch [8], or even by in-house scripts. A wide range of modern tools aid in the collection and storage of application metrics in the form of time-series data, such as InfluxDB [17], Prometheus [30], LogStash [26], or ElasticSearch [8]. It is noteworthy that each independent load test will generate independent time-series data sets. These data sets are split into test and training sets and used later by MLOLET for spike detection and forecasting.

MLOLET applies machine learning to help solve the two aforementioned challenges. For data preprocessing and machine learning-based modeling, there are many libraries and platforms available such as Apache Spark [2], TensorFlow [41], PyTorch [32], or DeepLearning4J [7]. The model serving can be achieved using any of the following: Apache Spark [2], DeepLearning4J [7], TensorFlow-Serving [42], Seldon [35], Flask [11], etc. This step is, however, conditioned, to a certain extent, by the modeling tools and approach selected in the previous step.

Noteworthy is the fact that MLOLET's spike detection relies on an online detection approach. Machine learning offers several architectures that function in an online setting; however, this is not a prerequisite as other techniques may be used as long as they function in an online setting (e.g., the VARMA statistical method [27]). The process we described should be general enough to be executed in any type of environment: physical, virtualized, or containerized (e.g., in a Kubernetes [22] based setup).

3.2 Running Load Tests

We run load tests with different configurations. During the load tests, we kept track of the system configuration variables, which included both internal configuration parameters of the components and deployment variables (e.g., the number of instances of various components). We also monitored the actual load applied to the system for each configuration.

With these settings, we obtained a pool of time series data for each of the targeted, generalizable application metrics (throughput per unit of time and associated average response time) after running multiple load tests with different workload types and configurations. This allowed us to study whether we could train a time-series model using data from one particular test and evaluate the model on similar tests with different configurations.

3.3 Model Classes in MLOLET

Based on the online testing/evaluation requirement of MLOLET, there are two types (classes) of spike detection algorithms for time series analysis that may be used here: traditional (also known as statistical) models, and, artificial neural network based models. The MLOLET process does not impose the use of a specific model. In fact, as the process is horizontally extensible, several different models may be used simultaneously (if needed) for different scenarios.

3.3.1 Traditional (or statistical) models. We use the Vector Auto-Regression Moving Average (VARMA) [27] as our traditional reference model for time series analysis. The VARMA method is a generalized version of the ARMA model for multivariate stationary time series. It uses a combination of Vector Auto-Regression and Vector Moving Averages together with ARMA to model the next step in the multivariate time series. We have selected VARMA as a reference model for two of its characteristics that help it compare with newer deep learning based algorithms.

Similar to a deep neural network (DNN [48]), the VARMA algorithm can accept an ad-hoc defined input time series sequence to predict the next value. Also, similar to a DNN, the length of the input sequence used by VARMA may be adjusted (as a hyper-parameter) in the training process. These two similarities with neural network based algorithms make it easy to use VARMA in an

online prediction setting much like a DNN-based model would be used. Because of this similarity it is more straightforward to make a comparison between the results of the models from these two classes of algorithms.

3.3.2 Artificial Neural Network (ANN) based models. Artificial neural networks have proven to be very good in generalizing any type of dataset and modeling time series makes no exception. Prior studies [18, 24, 40] showed that there are certain types of neural networks better suited for time series analysis. In our tests, we evaluated the following architectures: Convolutional Neural Networks (CNN) [48], classical Recurrent Neural Networks (RNN) [48] and variants such as Long Short-Term Memory network (LSTM) [48] as well as Bidirectional Recurrent Neural Network [4]. For more advanced architectures, we have also evaluated ResNet [15] type and LSTM-AutoEncoder [50] based approaches.

A Recurrent Neural Network is a ANN where connections between nodes form a directed graph along a temporal sequence. They are derived from feed forward neural networks and have an internal state that is used as memory to process sequences of variable length. An LSTM is an RNN variant that, during the training process, remembers the order of the data from the sequences [9, 47].

A major shortcoming of the standard RNN is that they have access to past information but not to a future context. The Bidirectional Recurrent Neural Network [33] provides a solution to this problem by using two separate recurrent hidden layers on the input sequence. One layer operates in forward direction and the second one in the backward direction and with both layers connected to the same output they provide context to both input directions. Bidirectional LSTM (B-LSTM), combines the principles of bidirectional networks and LSTM [4].

As pointed out by Zhu and Laptev [50], neural networks may suffer from prediction uncertainty, and they proposed an autoencoder-based architecture to mitigate false anomaly alerts. For an analogy to our case, we consider spikes as an anomaly, although technically they are not. The purpose of the encoder-decoder layer is to extract representative embeddings from the input time series, as autoencoders are used to perform either dimensionality reduction or feature extraction [3]. In short, the autoencoder acts as an intelligent feature extraction blackbox [50].

Another type of network which is very good in feature extraction is the Convolutional Neural Network due to its automatic capability of detecting important features without human supervision [12, 21]. This property of a CNN network helps extracting and learning patterns from any sequential data and can therefore be leveraged to perform both spike detection as well as forecasting. ResNet (or Residual Networks architecture) was proposed by Microsoft Research in 2015 [15] and it is an advanced CNN type of network as it addresses the vanishing/exploding gradient problem. According to the research conducted by He et al. [15], the ResNet model is one of the top advanced CNN based architectures. It has been proven to perform well in time series analysis by Fawaz et al. [18].

We tested the efficiency of the different models using the same input length sequence, while each of the models was optimized according to their own set of hyper-parameters.

3.4 Defining Spikes

As previously mentioned, by spike detection, we are identifying those points in the datasets that differ from the majority of the data. Generally speaking, a spike is an event that does not fit in any determined pattern or data cluster. However, when we talk about spike detection for time series, one value may be considered a spike due to the time it appeared and the values before it. For example, if the typical value of the CPU load is around 10% for an application, an extreme event may cause the CPU load to jump to, say, 70%. On the other hand, if the load stabilizes around 70%, then this jump is no longer suspicious but is part of normal behavior. This leads to the fact that spike detection is very context-dependent.

Moreover, different loads may exert different average KPI values and spikes for a given configuration set. For instance, a configuration set X with an average load of 100 TPS might result in an average CPU utilization of 60%, accompanied by spikes of specific amplitude and duration. In contrast, increasing the average load to 150 TPS with the same configuration could lead to an average CPU utilization of 70%, with spikes that differ in both duration and amplitude. Similarly, the same average load of 100TPS may produce different (spike) data for configuration sets Y, Z, etc.

Following these examples, we found that simple threshold based monitoring is inadequate to handle complex situations in practice, and more advanced methods are required. One such method is based on the 3-sigma statistical rule (also known as the empirical rule) to determine whether the next point in the time series is a spike or not. The 3-sigma rule of thumb is considered in empirical science a conventional heuristic where most of the typical values lie within the three standard deviations of the mean (also known as the normal distribution) [16, 39, 49]. Therefore, first, we measure the mean and the standard deviation (σ) of the data points in the training data. Then, we define our error threshold as: $\epsilon = mean + /-3\sigma$. Any data point value larger than the threshold is considered a spike. This approach lets us apply the method to a large number of systems, without having any special domain knowledge about the underlying system's characteristics. Note that, we normalize the metrics when calculating the thresholds using the 3-sigma rule. Hence, we can apply the threshold on other tests after normalizing the metrics.

3.5 Data Preprocessing and Tuning Machine Learning Models

Prior studies found that inputs to neural network inputs are scale sensitive [36, 38]. Thus, we used several normalization methods: MinMaxScaler and StdScaler provided by the scikit-learn [34] library and logarithmic based scaling (i.e., log transformation). Furthermore, we also apply grid search optimization for hyperparameter tuning in the case of the neural network based models, where parameters are randomly chosen from a given range. We report the model that achieves the best performance in Section 4.

We used the same range of values for all the parameters in both hyperparameter tuning methods. We considered tuning common neural network parameters such as the number of neurons per layer [100 - 250], the learning rate [10e-4, 10e-3, 10e-2], and the number of hidden layers [1 - 10]. In time series analysis, additional hyperparameters include the input data window, the offset, and the signal dimensionality. Since our data consists of single point KPIs,

the feature set dimension is always $D = 1$. The input data window (T) represents the number of time steps used to predict the next value in the series. For each pass, we consider different data window sizes in the range [5 - 120]. The offset denotes the number of steps ahead we forecast from the chosen data window, (e.g., an offset of one predicts one step ahead). This concept is depicted in Figure 2. Additionally, we refer to offset $offset = 1$ as *1-step forecasting*, and $offset > 1$ as *n-step forecasting*.

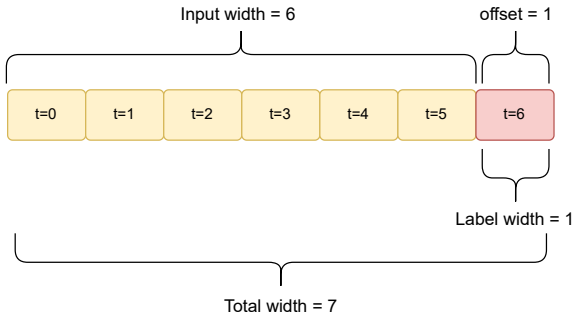

Figure 2: Data windowing.

3.6 Evaluating ML Models

We evaluated our time-series forecasting models using several metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE) and the Pearson Correlation Coefficient (PCC) value:

The **Mean Absolute Error (MAE)** is the average of the absolute errors, in other words it is the mean value of the absolute difference between the next predicted value and the actual value from the time series. Having n points in the testing set, it is calculated using the following formula:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n-1}} \tag{1}$$

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n} \tag{2}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid actual_i - predicted_i \mid \tag{3}$$

The **Mean Squared Error (MSE)** is a measure of the average of the squared distance between the next predicted value and the real data. Having n points in the testing set, it is defined by the following formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(actual_i - predicted_i)^2 \tag{4}$$

The **Mean Absolute Percentage Error (MAPE)** is also known as the mean absolute percentage deviation (MAPD). It is a statistical forecasting measure of the prediction accuracy of a model. It measures the size of the error in percentage terms and, for n points in the testing set, it is calculated by the following formula:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n} \mid \frac{actual_i - predicted_i}{actual_i} \mid \tag{5}$$

The **Pearson Correlation Coefficient (PCC)** is a measure of the linear correlation (in other words dependence) between two data sets (in our case the true values and the predicted ones). It is defined as the ratio between the covariance of the two variables and product of their standard deviations:

$$\rho_{act,pred} = \frac{cov(act, pred)}{\sigma_{act}\sigma_{pred}} = \frac{\mathbb{E}[(Actual - \mu_{act})(Predicted - \mu_{pred})]}{\sigma_{act}\sigma_{pred}} \tag{6}$$

In "(6)" μ, σ represent the standard deviation and the mean respectively for the actual (act) and the predicted (pred) sets, *cov* represents the covariance which can also be described with the help of the expectation function $\mathbb{E}$ of the product between the actual and predicted values adjusted by their respective means. The PCC is a measure of synchronicity of the variation of the two data sets: the actual and the forecasted values.

4 CASE STUDY RESULTS

In this section, we discuss the evaluation results of MLOLET. Due to space constraints, while we present the open source system results, we discuss only the results from the industry system. The complete results for the open-source system are available on GitHub [28]. The general results observed on the open-source system are the same as those for the industry system.

RQ1: How do different models compare in one-step forecasting?

Motivation: In order to implement the early stopping mechanism proposed by the MLOLET framework, we need to identify a performant forecasting model. The best performing model becomes the baseline for the subsequent research questions.

Approach: We evaluate different machine learning models implemented in MLOLET using a randomly selected load testing event. We use the same time series data for each model evaluation, as described in Section 3. We are interested in the 1-step time forecasting accuracy of the different machine learning algorithms and architectures.

Results: We find that, in general, all the models provide good 1-step forecasting results. Tables 1 and 2 summarize the best 1-step forecasting results for each type of model we experimented with on the enterprise and open source systems, respectively. We used the normalized values of the datasets (e.g., KPIs) when calculating the model results. We find that VARMA achieves the worst results in terms of all the evaluated metrics (i.e., MAE is 0.1156 and MSE is 0.0743). On the other hand, LSTM-based models achieve the best results (i.e., MAE ranges from 0.0386 to 0.0804, and MSE ranges from 0.0131 to 0.0207). In contrast, CNN and ResNet have a slightly higher MAE and MSE than Autoencoder LSTM. Our findings with regard to the performance of Autoencoder LSTM architectures follow the findings of Laptev and his team [23].

Table 1: Enterprise System - One step forecasting results (RQ1).

Model	MAE	MSE	MAPE (%)	PCC
VARMA	0.1156	0.0743	8.290	0.5965
RNN	0.0644	0.0227	1.347	0.8217
LSTM	0.0756	0.0207	1.641	0.8611
Bidirectional LSTM	0.0804	0.0254	1.720	0.843
Autoencoder LSTM	0.0386	0.0131	1.0826	0.8999
CNN	0.07173	0.0282	1.494	0.8094
ResNet	0.0687	0.0177	1.457	0.8851

Table 2: Open Source System - - One step forecasting results (RQ1).

Model	MAE	MSE	MAPE (%)	PCC
VARMA	0.0303	0.0236	26.6544	0.9888
RNN	0.0058	0.0010	3.5584	0.9888
LSTM	0.0072	0.0010	3.4166	0.9887
Bidirectional LSTM	0.0083	0.0010	6.0844	0.9887
Autoencoder LSTM	0.0078	0.0010	12.7623	0.9890
CNN	0.0086	0.0010	6.9289	0.9894
ResNet	0.0177	0.0013	55.3107	0.9854

Our results show that the ML models, in general, are able to predict the normalized KPIs in the next time window with good accuracy.

Although VARMA achieves the worst results, its training and execution time is significantly faster than that of DNN models. Since training and applying the model online may require substantial computing resources, VARMA may still be a good candidate model when such resources are limited. Nevertheless, advanced neural networks are still better at providing more accurate 1-step forecasting. Although we cannot show the ML model configuration values we found due to NDA (we describe our ML model configuration search/tuning in Section 3.5), we found that selecting the right time window size, T, is very important.

> We find that LSTM-Autoencoder achieves the best results. It is noteworthy that, depending on the required accuracy of the detection and the project perspective, simpler models like VARMA may still provide reasonable results.

RQ2: What is the generalizability of the forecasting model when system deployment settings change?

Motivation: Typically, spike mechanisms are put in place for static systems. By static, we mean the internal configuration of a system remains unchanged for different loads applied to that system and does not account for the number of nodes added for redundancy or capacity scaling purposes. In the practice of load testing, an application or an ecosystem of several applications requires parameter tuning. It would be impractical for load test engineers to retrain the model every time the internal configuration changes. Hence, in this research question (RQ), we compare the forecasting capabilities of each model we used for two use cases: 1) when the internal configuration of the system has changed, and 2) when the business request type has changed.

Table 3: Enterprise System - Forecasting results in an online setting using a previously trained model on new system configurations (RQ2).

Model	MAE	MSE	MAPE (%)	PCC
VARMA	0.1236	0.0775	8.376	0.5825
RNN	0.2231	0.1534	7.213	0.7339
LSTM	0.9777	0.9218	2.603	0.8008
Bidirectional LSTM	0.9752	0.1354	1.928	0.7941
Autoencoder LSTM	0.0610	0.1179	3.7872	0.7524
CNN	0.8649	0.0785	10.885	0.8458
ResNet	0.3561	0.1604	6.721	0.8023

Approach: As described in Section 3.2, each load test varies either the system configuration or the applied load or both. For this research question, we take a model trained with one of the randomly selected load tests data and verify the forecasting capabilities on all the remaining data sets. In other words, we take the model trained on a specific configuration and apply it to all other configurations and request types.

Results: Tables 3 and 4 summarize the prediction metrics for each model, on each studied system, respectively. Similar to RQ1, we normalize the values of the dataset when calculating the prediction metrics. Overall, we find that the errors increase compared to the results of RQ1. However, we find that the mean errors are still relatively small and most models yield similar result. Similar to RQ1, we find that, even though VARMA has one of the worst performances, the prediction results are still reasonable (i.e., MAE of 0.1236 and MSR of 0.0775). Hence, practitioners may still consider VARMA if they have limited resources to train more complex DNN models. We also see an increase in percentage error in convolution-based models (CNN and ResNet). This may be because these models require more training data to correctly recognize the patterns in the datasets. Finally, we find that LSTM-based models still achieve the best results. Our findings indicate that practitioners should consider LSTM-based models if they are concerned with prediction accuracy.

> Similar to RQ1, we find that LSTM-based models produce the best result. In contrast, we observed a larger decrease in forecasting results of the CNN-based models.

RQ3: How far in time can MLOLET forecast future trends?

Motivation: In this RQ, we aim to determine how far into the future the baseline model can predict. Such forecasting ability provide load test engineers with insights on whether they should add more resources in advance to account for increased load.

Approach: To solve this problem, a large number of model coefficients must be determined alongside the size of the sliding window. This is crucial in defining a model that uses the last X time steps

Table 4: Open Source System - Forecasting results in an online setting using a previously trained model on new system configurations (RQ2).

Model	MAE	MSE	MAPE (%)	PCC
VARMA	0.0401	0.0779	6.2584	0.9613
RNN	0.0690	0.0880	12.4745	0.9529
LSTM	0.0154	0.0037	4.7358	0.9525
Bidirectional LSTM	0.0124	0.0038	3.8204	0.9519
Autoencoder LSTM	0.0129	0.0036	3.8595	0.9528
CNN	0.0258	0.0044	8.5224	0.9451
ResNet	0.0751	0.0190	24.6865	0.8659

to predict the next N number of time steps. If a model can accurately forecast far enough into the future, the operations team may gain important insights that help them manage the software system more efficiently by scaling it up or down based on projected traffic requirements. A high-level view of the implied process is presented in Figure 3, where we can see that engineers may test how different models perform trend forecasting by comparing with the ground truth after every N step. Based on conclusions drawn from the comparison, engineers may devise new or update existing operational procedures to ensure the SLAs of the software systems are maintained within the desired boundaries.

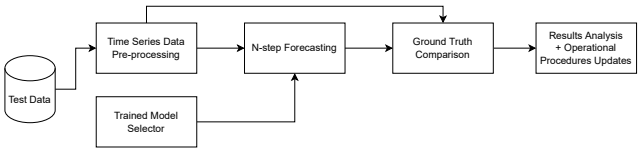

Figure 3: Traffic forecasting process overview.

Results: Although we cannot show the detailed results due to NDA, we share our findings on the maximum forecast steps (in seconds) that can still have a reasonable accuracy in our experiment (Table 5). The cutoff values were determined by system experts of the enterprise systems through visual inspection of the prediction plots. We provide, however the MAE and MSE details for the open source system. For our subject system, our test results show that most of the models have difficulties forecasting beyond 5-15 seconds in the future. One exception is the AutoEncoder-LSTM model, for which we obtained acceptable forecasts up to 120 seconds, followed by the B-LSTM architecture, for which we obtained forecasts up to 30 seconds. Although the forecasting capabilities of some models may seem limited (5-15 seconds), they still may be usable in practice. Many industry systems support features for online reconfiguration (e.g., JMX-based systems) or can be started in under 5 seconds, especially in containerized/Kubernetes-based environments. The open source system has similar results as seen in Table 6.

We also noticed that increasing the size of the input data window T (i.e., the number of time steps used to predict the next value in the series) does not always improve accuracy. Moreover, a larger size T usually results in a more complex network, often requiring more hidden layers rather than more neurons per layer.

Table 5: Enterprise System - Results of forecasting capabilities for different model architectures (RQ3).

Model Name	Max Forecast Steps (seconds)
VARMA	5
RNN	10
LSTM	10
Bidirectional LSTM	30
AutoEncoder LSTM	120
CNN	10
ResNet	15

Table 6: Open Source System - Results of forecasting capabilities for different model architectures (RQ3).

Model Name	Max Forecast Steps (seconds)	MAE	MSE
VARMA	15	0.2827	0.3178
RNN	30	0.1198	0.0369
LSTM	30	0.1072	0.0342
Bidirectional LSTM	45	0.1228	0.0330
AutoEncoder LSTM	120	0.5149	0.3170
CNN	15	0.1717	0.0441
ResNet	10	0.1572	0.0395

We find that the LSTM-Autoencoder model can forecast the trend with acceptable results up to 120 seconds for the industrial test subject system. We also find that using more time steps in the training process may not always help improve the model's accuracy.

5 DISCUSSIONS

In this section, we discuss the lessons that we learned from conducting the experiments and the feedback we received from our industrial partner.

5.1 Spike Effects and Online Spike Detection Observations

Dealing with spike misclassification is part of the model training where different metrics may be used to determine the dynamic thresholds, using the Mean Absolute Error or the Mean Squared Error. These two metrics provide a way to control the sensitivity of the spike detection with regard to the previous point (or sequence of points) observed by the model and are used to predict the next value in the sequence. In our experiments, we noticed that using the Mean Absolute Error as a measure of the prediction deviation provided better results for KPI spike detection.

In addition, software systems under load may exhibit spikes for various metrics. Neglecting spikes have often become more serious, causing outages [13]. Thus, it is important to understand the nature of the spike, its cause, and possible ripple effect in the system. In other words, having a good spike detection system in place and performing detection for various KPIs simultaneously can help

correlate events among the different components of a software ecosystem and make it easier to isolate them to study their effects.

These aforementioned factors play an important role in the software development lifecycle (SDLC) and solution operational processes. In the SDLC, the execution of the load test cycle becomes more efficient, and developers gain accurate insights into when an event occurred. This aspect is extremely useful when spikes cause ripple effects that destabilize later parts of the ecosystem, as those events are hard to reproduce without knowing the starting condition (e.g., some spike in the load of some component may have caused a buffer overrun, leading to loss of messages or corruption of other data).

Another important factor about spikes and their ripple effect is knowing when to act on them and what count threshold determines that a load test is failing, necessitating the decision to stop it. This may require expert knowledge of the software system to make correlations between the different data points and logs collected from the software system. The challenge here is to understand, for example, how the ripple effects of the spikes may propagate through a specific component or other components. For example, let's assume that during a load test, three detected problems do not occur at the same time. If the load test was stopped early after detecting the first problem, would the other two problems still occur in subsequent load test runs, or were they a consequence of the first error? By correcting the first error, would we never encounter the other ones? This problem does not easily generalize, and in the case of our industrial partner, it was often determined that by detecting a certain problem, using expert knowledge, test engineers could predict where the next problem might occur. Therefore, performing an early stop was more beneficial to project economics overall.

5.2 Transferable Models and Operational Insights

When referring to reducing load testing time while increasing process efficiency by using spike detection-based techniques, it is best to have a model that can be used across different configurations and multiple business case scenarios. In our experiments, we demonstrate that it is possible to train a model using load testing data from one system configuration and then use that trained model for spike detection in other configurations and business scenarios. We observe that when using different configurations, the signal reconstruction (from the time series data) is better when the predictions are adjusted with the mean of the training data.

Overall the outcome of these tests meant two very important things for our industrial partner:

- For future iterations, it is not required to train a spike detection model for each scenario, thus lowering the overall effort of the load testing process and improving project economics.
- The trained model may be used in production by the operations team, provided the load test data resembles production load conditions.

Continuing to refer to the operational insights, having a model capable of making forecasts for a sufficiently long future window (offset) can accommodate automatic scaling (sufficient time to start additional service instances). This effort may lower the overall energy consumption of the software ecosystem, which has two major benefits:

- It lowers the overall capital expenditure (CAPEX) cost in the data center by keeping the compute resources tuned to the forecasted load. Unfortunately, we are unaware of the actual cost reductions achieved, as determining such information requires immense effort and should be the subject of a future study.
- It aligns with our industrial partner's efforts in sustainability and corporate responsibility.

Several discussion points remained open with the industrial partner regarding the model's ability to forecast future load: how far into the future is long enough to predict, and how can false positive spikes affect the operational decisions to scale up or down the software ecosystem components.

To mitigate the first case, some components may require less time to act upon a scaling decision, while others may require more time. This could lead to having different model architectures for different components and perhaps even create new requirements for the development team to improve the application start-up SLAs.

To mitigate the second case, the frequency and spacing out between predicted spikes may be a decision factor for accepting the forecasting model, which is monitored with the help of a spike detection model. The two models may be different as they may use different input window sizes for making the next prediction. This area may warrant further investigation in the future, as creating the scaling rules can be very complex with many parameters to consider for making a scaling decision.

6 THREATS TO VALIDITY

Internal validity. The enterprise system continuously evolves. Therefore, from one release to another, the overall throughput of the system may be affected, depending on the particular changes made to the source code. In our tests, we used the same release of to avoid side effects introduced by such code changes.

External validity. We conducted our experiments on a large enterprise system composed of many components with different business functions. For reproducibility, we also tested our assumptions on one open-source system that models, in a generic way using a black box approach, a very large number of business applications given their common characteristic: receiving a request, processing it, and returning a response. Future studies may be needed to evaluate our approach on other more specific systems.

Construct validity. Prior studies [14, 29] show that model performance is very closely tied to the machine learning parameters. In our experiments, we used grid search as technique to tune these parameters to reduce the bias. While we mitigated multicollinearity and over-fitting by splitting the data into training, validation, and testing sets it may be possible that the models may still suffer from overfitting. However, in our experiments on external validation (RQ2), we find that models still perform well. We evaluated the ML models that we implemented and provided in MLOLET. However, there may be other models that can achieve better results. In our case, the models that we implemented could already help solve the challenges that we encountered. Future studies may include other ML models to evaluate their effectiveness.

7 RELATED WORK

Events forecasting and spike detection have been of great interest in the research community over the years. However, they have rarely been studied and applied in industrial settings simultaneously due to their different characteristics. For the general use case, there are studies that address the idea of anomalies in time series. While spikes could be treated as anomalies in the generic sense of time series data, they are also different in that a spike may have a long duration and could be natural behavior of the system, whereas anomalies are short in nature and not considered part of the normal operation of a software system.

Time series spike detection can be achieved through a number of methods, both offline and online. Given the business requirements from our industrial partner, our research focuses solely on online detection methods. Below, we discuss related work in two areas: time-series forecasting using deep learning models, and spike detection for time series in load test data.

7.1 Time Series Forecasting Using Deep Learning Models

Event forecasting is a current everyday challenge that has many business applications. Modern machine learning model architectures have been created and evaluated by prior research trying to solve various business problems. Wei et al. [43] used LSTM based auto-encoders (AE) in order to predict road traffic flow. Their experiments show that AE-LSTMs outperform other model architectures. Similarly, Laptev et al. [23] and Zhu and Laptev [50] use an AE-LSTM architecture to forecast the number of trips during special events such as Christmas or New Year's eve at Uber. In addition to forecasting, they use the model to also predict anomalies if the forecast falls outside of the predicted interval (during real-time data collection).

Our work differs from these prior studies in that we focus on a different domain of problem—load testing of software systems. Similar to the above-mentioned work, we also try different architectures and come to a similar conclusion that, in general, advanced ANN networks perform better than traditional approaches.

7.2 Spike Detection for Time Series in Load Test Data

Chen et al., [6] proposed SPIKE as a method based on regression trees to predict cloud resource usage spikes. They use a supervised learning method to classify anything above given threshold as an anomaly therefore contributing to a spike. While Chen and his team make use of similar features for modeling (i.e., response time for a service, transaction throughput, etc.), our work is different as we are performing an unsupervised learning approach for spike detection. As pointed out in Section 3.4, using a static threshold for spike detection may not be sufficient in today's ever increasing complex IT systems.

Wen and Keyes [44] discuss the criticality of anomaly detection in automated monitoring systems. Their work focuses on CNN architectures and they show the importance of transfer learning combined with CNN to extend models' utilization to other systems. In our work, we test how models trained with specific time series data perform when the system conditions have changed, however we use this information for unsupervised spike detection instead of supervised anomaly detection. More precisely, we verify a model's spike detection capabilities and new traffic forecasting on a new, never seen system configuration. Moreover, in our research, we test perform these verification on several model architectures in addition to CNNs.

8 CONCLUSIONS

Load testing is a critical activity in the software development lifecycle (SDLC) as it ensures the software system behaves correctly within defined SLA ranges under loads as close as possible to real world usage. As a result, it is recommended that load test engineers run as many tests as possible using different loads and system configurations. In this paper we propose an approach called MLOLET to:

- Help load test engineers overcome budget restraints (time and costs) by increasing the efficiency of the load test cycle with an early stopping mechanism for failing load tests.
- Increase operational efficiency and service level agreements (SLAs) by forecasting traffic trends

The process we propose is extensible and offers the flexibility to use a wide variety of tools for each step. It also provides the flexibility to select one or more machine learning models to address the two important challenges mentioned above.

We evaluated MLOLET on a large-scale enterprise system developed by our industrial partner and verified our assumptions formulated in our research questions regarding spike detection and time series signal reconstruction using load test data from the enterprise system. Additionally, we evaluated MLOLET on an open-source system by following a black box approach to model in a generic way a large number of business applications that have similar characteristics to the studied enterprise system: receiving a request, processing it, and returning a response.

By using MLOLET our industrial partner was able to:

- Speed up the load testing setup and execution processes by over 50% by implementing load test automation.
- Double the number of load and endurance tests by using the early stopping rules following 3-sigma/z-scores based spike detection process.
- Reduce the amount of time spent on load test data analysis with respect to spikes by 65%.

ACKNOWLEDGEMENT

We want to thank Ericsson for providing access to the enterprise systems that we used in our case study. The findings and opinions expressed in this paper are those of the authors and do not necessarily represent or reflect those of Ericsson and/or its subsidiaries and affiliation. Our results do not in any way reflect the quality of Ericsson's products.

REFERENCES

[1] Ansible 2022. Ansible Automation Platform. https://www.ansible.com/.
[2] ApacheSpark 2022. Apache Spark - Unified Engine For Large Scale Data Analytics. https://spark.apache.org.
[3] Yoshua Bengio. 2009. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning* 2, 1 (2009), 1–127. https://doi.org/10.1561/2200000006

[4] Raymond Brueckner and Björn Schuller. 2014. Social signal classification using deep blstm recurrent neural networks. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2014), 4823–4827.

[5] Chef 2022. Chef Automation Platform. https://chef.io/.

[6] Jianfeng Chen, Joymallya Chakraborty, Philip V. Clark, Kevin Haverlock, Snehit Cherian, and Tim Menzies. 2019. Predicting breakdowns in cloud services (with SPIKE). *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2019).

[7] DeepLearning4J 2022. DeepLearning4J - A suite of tools for running deep learning on the JVM. https://deeplearning4j.konduit.ai/.

[8] ElasticSearch 2022. ElasticSearch - A Distributed Free and Open Search and Analytics Engine. https://elastic.co.

[9] Tolga Ergen and Suleyman S Kozat. 2017. Online training of LSTM networks in distributed systems for variable length data sequences. *IEEE transactions on neural networks and learning systems* 29, 10 (2017), 5159–5165.

[10] FastCompany. 2016. How one second could cost Amazon 1.6 billion sales. http://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales. Last accessed March 3 2016.

[11] flask 2022. Flask - A micro web Python framework. https://flask.palletsprojects.com/.

[12] Dario García-Gasulla, Ferran Parés, Armand Vilalta, Jonathan Moreno, Eduard Ayguadé, Jesús Labarta, Ulises Cortés, and Toyotaro Suzumura. 2018. On the Behavior of Convolutional Nets for Feature Extraction. *J. Artif. Intell. Res.* 61 (2018), 563–592.

[13] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. 2016. Why Does the Cloud Stop Computing?: Lessons from Hundreds of Service Outages. *Proceedings of the Seventh ACM Symposium on Cloud Computing* (2016).

[14] Huong Ha and Hongyu Zhang. 2019. DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. 1095–1106.

[15] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 770–778.

[16] Xiaolei Hua, Lin Zhu, Shenglin Zhang, Zeyan Li, Su Wang, Dong Zhou, Shuo Wang, and Chao Deng. 2022. GenAD: General Representations of Multivariate Time Seriesfor Anomaly Detection. *ArXiv* abs/2202.04250 (2022).

[17] InfluxDB 2022. InfluxDB - An Open Source Timeseries Database. https://influxdata.com/.

[18] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: a review. *Data mining and knowledge discovery* 33, 4 (2019), 917–963.

[19] Jenkins 2022. Jenkins Automation Server. https://www.jenkins.io/.

[20] JMeter 2022. Apache JMeter Load Testing Tool. https://jmeter.apache.org/.

[21] Manjunath Jogin, Mohana, M S Madhulika, G D Divya, R K Meghana, and S Apoorva. 2018. Feature Extraction using Convolution Neural Networks (CNN) and Deep Learning. In *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. 2319–2323. https://doi.org/10.1109/RTEICT42901.2018.9012507

[22] Kubernetes 2019. Kubernetes - An open-source system for automating deployment, scaling, and management of containerized applications. https://kubernetes.io.

[23] Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. 2017. Time-series extreme event forecasting with neural networks at uber. In *International conference on machine learning*, Vol. 34. sn, 1–5.

[24] Bryan Lim and Stefan Zohren. 2021. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A* 379, 2194 (2021), 20200209.

[25] Load Generator 2023. Load Generator - load testing controller with spike load capability. https://github.com/eartvit/load-generator.

[26] Logstash 2022. Logstash - An Open Source Data Collection Engine. https://elastic.co.

[27] Helmut Lütkepohl. 2005. *Specification and Checking the Adequacy of VARMA Models*. Springer Berlin Heidelberg, Berlin, Heidelberg, 493–514. https://doi.org/10.1007/978-3-540-27752-1_13

[28] MLOLET 2023. MLOLET: Machine Learning Optimized Load and Endurance Testing example implementation on Openshift. https://github.com/eartvit/mlolet.

[29] Andrew Y. Ng. 2004. Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning* (Banff, Alberta, Canada) *(ICML '04)*. Association for Computing Machinery, New York, NY, USA, 78. https://doi.org/10.1145/1015330.1015435

[30] Prometheus 2022. Prometheus - Monitoring System and Timeseries Database. https://prometheus.io/.

[31] Puppet 2022. Puppet Automation Platform. https://www.puppet.com/.

[32] Pytorch 2022. Pytorch - An open source machine learning framework that accelerates the path from research prototyping to production deployment. https://pytorch.org.

[33] M. Schuster and K.K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681. https://doi.org/10.1109/78.650093

[34] SciKit-Learn 2022. SciKit Learn - Machine Learning in Python. https://pypi.org/project/psutil.

[35] Seldon 2022. Seldon - Deploy, Monitor and Explain Machine Learning Models. https://seldon.io.

[36] Bikesh Kumar Singh, Kesari Verma, and A. S. Thoke. 2015. Investigations on Impact of Feature Normalization Techniques on Classifier's Performance in Breast Tumor Classification. *International Journal of Computer Applications* 116 (2015), 11–15.

[37] SoapUI 2022. SoapUI API Testing Tool. https://soapui.org/.

[38] J. Sola and Joaquin Sevilla. 1997. Importance of input data normalization for the application of neural networks to complex industrial problems. *Nuclear Science, IEEE Transactions on* 44 (07 1997), 1464 – 1468. https://doi.org/10.1109/23.589532

[39] Siwoon Son, Myeong-Seon Gil, Yang-Sae Moon, and Hee-Sun Won. 2016. Anomaly Detection of Hadoop Log Data Using Moving Average and 3-Sigma.

[40] Yang Syu, Chien-Min Wang, and Yong-Yi Fanjiang. 2018. A survey of time-aware dynamic QOS forecasting research, its future challenges and research directions. In *International Conference on Services Computing*. Springer, 36–50.

[41] TensorFlow 2022. TensorFlow - An End-to-end Machine Learning Platform. https://tensorflow.org.

[42] TFX 2022. TensorFlow Serving - A high-performance serving system for machine learning models. https://tensorflow.org.

[43] Wangyang Wei, Honghai Wu, and Huadong Ma. 2019. An AutoEncoder and LSTM-Based Traffic Flow Prediction Method. *Sensors (Basel, Switzerland)* 19 (2019).

[44] Tailai Wen and Roy Keyes. 2019. Time Series Anomaly Detection Using Convolutional Neural Networks and Transfer Learning. *ArXiv* abs/1905.13628 (2019).

[45] WireMock 2022. WireMock: Mock the APIs You Depend On. https://wiremock.org/.

[46] WireMock Metrics 2023. WireMock Metrics - extended WireMock with Prometheus metrics and global random string payload ResponseTransformer. https://github.com/eartvit/wiremock-metrics2.

[47] Martin Wöllmer, Florian Eyben, Björn Schuller, Ellen Douglas-Cowie, and Roddy Cowie. 2009. Data-driven clustering in emotional space for affect recognition using discriminatively trained LSTM networks. In *Proc. Interspeech 2009, Brighton, UK*. 1595–1598.

[48] Giancarlo Zaccone, Md. Rezaul Karim, and Ahmed Menshawy. 2017. *Deep Learning with TensorFlow*. Packt Publishing Ltd.

[49] Chunkai Zhang and Ao Yin. 2019. Anomaly Detection Algorithm Based on Subspace Local Density Estimation. *Int. J. Web Serv. Res.* 16 (2019), 44–58.

[50] Lingxue Zhu and Nikolay Pavlovich Laptev. 2017. Deep and Confident Prediction for Time Series at Uber. *2017 IEEE International Conference on Data Mining Workshops (ICDMW)* (2017), 103–110.