Mathematical Modelling of Engineering Problems
Vol. 12, No. 11, November, 2025, pp. 3895-3903
Journal homepage: http://iieta.org/journals/mmep

IntelliStream: A Machine Learning Framework Based on Regression for Improving Broker Performance and Throughput via Log Analysis

G. Vijayakumar*, R. K. Bharathi

Department of Computer Applications, JSS Science &Technology University, University of Mysore, Mysore 570006, India

Corresponding Author Email: vijayakumar.gundappa@gmail.com

https://doi.org/10.18280/mmep.121115

Received: 9 July 2025
Revised: 3 September 2025
Accepted: 12 September 2025
Available online: 30 November 2025

Keywords:
garbage collection, machine learning, streaming, tuning, regression

ABSTRACT

In modern data-driven ecosystems, platforms such as Apache Kafka are essential for handling continuous, high-volume event streams across domains including IoT, finance, e-commerce, and industrial monitoring. These applications rely on real-time processing for tasks such as anomaly detection, fraud analysis, recommendation systems, and predictive maintenance. As workloads vary widely, multiple specialized pipelines are often required, adding complexity and increasing operational overhead. Tuning such systems is challenging, as parameters related to memory, concurrency, replication, and batching must be balanced to maintain high throughput, low latency, and stable performance under dynamic conditions. Misconfigurations can trigger fast failures, making automated, adaptive tuning essential. The study proposes a machine learning framework that dynamically adjusts real-time streaming platforms to achieve better performance. By analyzing garbage collection (GC) logs and broker logs, the method employs regression models to detect bottlenecks and predict optimal configuration settings. Key metrics are extracted from logs, used to train regression models, and applied to adjust parameters dynamically. Experimental evaluation shows that Linear and Ridge Regression achieved an R^2 of 0.9999 with a Mean Squared Error (MSE) of 4.84E-06, delivering over 99% accuracy in predicting throughput trends. The method dynamically optimizes performance, cutting manual tuning and enabling more intelligent, self-managing streaming systems.

1. INTRODUCTION

The need for real-time data streaming and processing has significantly increased in today's data-driven society. Platforms designed for these tasks must efficiently and reliably handle a vast amount of data. Apache Kafka, a widely used distributed event streaming platform, has emerged as a key technology for building real-time data pipelines and streaming applications [1]. It remains quite challenging to optimize Kafka's performance to meet the demands of a changing workload, despite its robust design.

Recent studies comparing manual, heuristic, and machine learning–based tuning methods in distributed systems are supported by several journal articles and surveys. Manual tuning of Kafka parameters generally shows only modest throughput improvements, whereas heuristic methods achieve stronger results, and machine learning approaches often provide the largest performance gains while also reducing latency variability [2-4]. Manual tuning in distributed systems like Apache Kafka generally results in limited performance improvements due to its static adjustment nature. The studies [3, 4] presented that manual tuning requires administrators to change distributions and configurations directly and typically only achieves around a 10% improvement in throughput, as changes are not adaptive to dynamic workload conditions. Heuristic-based methods, using rules or profiling to guide tuning, can deliver higher throughput gains (up to 25%) but still struggle to adapt under fluctuating workloads [4-6]. Whereas Machine learning techniques, such as reinforcement learning and regression-based optimization, routinely surpass manual and heuristic methods in both throughput and latency stability (with throughput improvements of 40–50%) [2, 6-8].

Performance tuning of Kafka involves adjusting numerous configuration parameters that influence throughput, latency, and stability. These parameters, such as heap size, buffer sizes, and concurrency settings, can interact in complex ways, making manual tuning a daunting and often inefficient task [9]. Moreover, the dynamic nature of workloads in production environments necessitates continuous and adaptive tuning to maintain optimal performance [10]. The problem has a promising answer in machine learning. Machine learning models can find patterns and correlations in operational logs, particularly garbage collection (GC) logs and broker logs, which are not immediately visible using conventional techniques. These insights can then be used to predict the effects of various configurations on performance metrics and to identify optimal settings [11].

The selection of GC logs and broker logs as primary indicators is motivated by their direct impact on system performance. GC logs capture information about memory allocation, collection frequency, and pause durations [12]. A high frequency of GC events or long GC pauses can

significantly reduce throughput by interrupting message processing and increasing latency due to stalled threads. Similarly, broker logs provide operational metrics such as message throughput, request latency, error counts, and partition reassignments [10], all of which directly reflect system stability and responsiveness. For example, sustained error rates or rising latencies in broker logs indicate declining stability under load, while steady throughput values reflect efficient resource utilization. By analyzing both GC and broker logs together, the study establishes a holistic view: GC behavior explains memory-level bottlenecks, while broker metrics capture system-level performance outcomes. This dual-log perspective ensures that the machine learning framework links configuration parameters to the most critical performance dimensions.

The dependent variables that represent performance are explicitly defined as throughput (MB/sec), latency (milliseconds), and stability (measured through error rates and consistency under varying workloads). These metrics capture efficiency, responsiveness, and robustness in distributed streaming. The independent variables are the Kafka configuration parameters, such as heap size, buffer size, concurrency levels, and GC behavior that directly influence performance outcomes. To connect these inputs with measurable system behavior, the study leverages GC logs, which provide details on memory allocation, pause times, and heap utilization, as well as broker logs, which record throughput, request latency, and error statistics. This explicit mapping of independent and dependent variables strengthens the analytical foundation of the proposed machine learning framework, ensuring that the research targets the most relevant and impactful performance indicators.

We utilize the publicly available JVM Logs Dataset from Kaggle [13], which provides GC and Kafka broker logs under diverse workload conditions and serves as a reproducible benchmark for performance modeling studies. which comprises 471 files and 2,880 features derived from GC and Kafka broker logs. This rich dataset captures diverse workload conditions, providing a robust foundation for machine learning–based performance modeling and auto-tuning.

Through a series of experiments, the model shows the effectiveness of the strategy, exhibiting notable gains in performance stability and throughput. In addition to increasing system efficiency, the automated tuning process eliminates the need for significant manual intervention, which lowers complexity and operating costs. Given the complexity and non-linearity of distributed system behavior, relying on a single regression technique may not adequately capture the diverse relationships between configuration parameters and performance outcomes. For instance, while linear regression offers interpretability, it oversimplifies non-linear effects; conversely, tree-based models such as decision trees or gradient boosting can capture non-linear patterns but may risk overfitting [14]. To address these limitations, an ensemble of regression algorithms is often more effective, as it combines the strengths of multiple models to improve predictive accuracy and generalizability. Approaches such as Random Forest Regression and Gradient Boosting Regression have shown significant promise in prior system optimization studies [15], reducing variance and bias while delivering more reliable performance predictions. Incorporating ensemble regression methods into auto-tuning frameworks thus provides a fine-tuned and adaptive means of modeling system performance under dynamic workloads. The objective of this research is to develop a machine learning-based auto-tuning framework that improves Kafka's throughput and stability while reducing manual intervention.

The paper proposes an automated, machine learning-based approach to fine-tune real-time data streaming platforms for better throughput and overall performance. The method involves collecting and preprocessing log data, extracting relevant features, and training regression models to predict performance outcomes. The models are then used to adjust configuration parameters dynamically, ensuring the system operates at peak efficiency under varying conditions.

The remainder of the paper is structured as follows: Section 2 reviews related work in the field of performance tuning using machine learning. Section 3 describes the methodology, including data collection, feature engineering, and model training. Section 4 presents the experimental setup and results. Section 5 concludes the paper with a discussion of the findings, and finally, Section 6 presents potential future work in the area.

2. RELATED WORK

The challenge of optimizing performance in distributed systems [16], particularly real-time data streaming platforms, has been the focus of extensive research. The section reviews existing literature on performance tuning using machine learning, with a specific focus on the analysis of operational logs and the application of regression techniques.

2.1 Performance tuning of distributed systems

Traditional approaches to performance tuning in distributed systems often involve heuristic methods and rule-based configurations [17]. These methods require significant expertise and manual intervention, making them labor-intensive and less adaptable to dynamic workloads. Recent advances have shifted towards more automated solutions, leveraging statistical and machine learning techniques to address these limitations [18].

2.2 Machine learning for system optimization

Performance tuning is essential for ensuring system efficiency and stability, and automating this process minimizes manual effort while enabling adaptive optimization in dynamic workloads. Machine learning has shown considerable promise in optimizing system performance [19]. Approaches such as reinforcement learning [20], supervised learning, and unsupervised learning have been applied to various tuning problems. For instance, reinforcement learning has been applied to resource management in cluster computing, resulting in improved resource utilization and faster job completion times [11]. Similarly, deep learning models have been employed to predict performance bottlenecks in cloud services, thereby enabling more effective resource allocation [12].

2.3 Log analysis for performance tuning

Log analysis is a critical aspect of performance tuning, providing insights into system behavior and performance metrics. Logs from GC and broker activities offer valuable data for understanding and predicting system performance. It

is demonstrated that the use of log mining techniques to diagnose performance issues in distributed systems highlights the potential of logs as a rich source of information for optimization tasks [21].

While log analysis provides the raw behavioral data, it must be coupled with predictive modeling to translate these signals into actionable tuning decisions.

2.4 Regression techniques in performance prediction

Regression techniques are particularly effective in modeling the relationships between configuration parameters and performance metrics [22]. In the context of system tuning, regression models have been used to predict the impact of configuration changes on performance metrics. Using regression trees to model the performance of database queries provides a basis for automatic tuning of query parameters. In this way, log analysis and regression complement each other: logs supply the empirical evidence of system behavior, and regression translates this evidence into predictive insights for tuning.

There are multiple measures to find the effectiveness of machine learning models, as in Eq. (1).

Mean Squared Error (MSE): MSE is a metric used to measure the average squared difference between predicted and actual values in a dataset. It quantifies the error by squaring the differences to ensure both positive and negative errors contribute equally, and then averaging these squared differences.

Mean Absolute Error (MAE): MAE is a metric that calculates the error by taking the absolute value of the differences between predicted and actual values, and then averaging these absolute differences. It is less sensitive to outliers than MSE and provides a linear measure of average error [23]. However, it does not emphasize larger errors, which can be critical in some applications.

Root Mean Squared Error (RMSE): RMSE is the square root of MSE, making it interpretable in the same units as the target variable [23]. However, it carries the same information as MSE on a different scale.

Mean Absolute Percentage Error (MAPE): MAPE measures error as a percentage, which can be helpful in interpretability in business contexts [24]. However, it is undefined for zero values and can be biased by small actual values.

Adjusted R-squared (R^2): Adjusted R^2 adjusts for the number of predictors in the model, preventing overfitting. While helpful, it is more complex to interpret compared to the standard R^2 for an initial comparison [25].

$$\left\{\sum_{i=1}^{n}\left(x_i, y_i\right)\right. \tag{1}$$

With n observations, where x_i represents the feature vector and y_i the target variable. The goal is to predict the target variable $\hat{y}$ using the regression model as depicted in Eq. (2):

$$\hat{y} = f(x) \tag{2}$$

MSE emphasizes large errors; sensitivity is particularly crucial for throughput predictions, where large prediction errors can lead to misleading conclusions. Given a dataset of observations where the goal is to predict Throughput MB/sec using a regression model, MSE and R^2 are proven to be effective metrics for model evaluation. MSE emphasizes larger errors more heavily [26], ensuring the model minimizes significant deviations, which is crucial for accurate throughput predictions. R^2 provides a clear measure of the model's explanatory power and facilitates comparison across different models [25]. In contrast, MAE, which measures the average magnitude of errors without considering their direction, does not penalize larger errors as heavily, making it less suitable when large errors have significant impacts. RMSE offers similar benefits to MSE but with added complexity due to the square root transformation, without additional interpretative value [27]. Therefore, MSE and R^2 are preferred for their ability to provide both absolute and relative measures of model performance [23], ensuring accurate and reliable assessments for predicting throughput MB/sec.

Several frameworks have been proposed for the auto-tuning of distributed systems using machine learning [7]. Machine learning is used to automatically tune database configurations, demonstrating significant performance improvements [28]. Similarly, new frameworks were employed with a combination of supervised learning and Bayesian optimization to tune cloud services, highlighting the potential for machine learning in automated system tuning [29].

2.5 Application to Apache Kafka

A comprehensive framework that leverages log analysis and regression-based modeling for dynamic, machine learning–driven tuning of Kafka remains largely unrealized. Specific to Apache Kafka, research has explored various aspects of performance optimization, though there is limited work on comprehensive auto-tuning solutions [1]. There are experiments that examined the impact of different configuration settings on Kafka's performance, suggesting that machine learning could be beneficial in automating these adjustments [11]. However, a complete framework leveraging machine learning to analyze Kafka's GC and broker logs for dynamic tuning has not been fully realized.

The reviewed literature underscores the potential of machine learning techniques, particularly regression models, in automating the performance tuning of distributed systems [30]. While significant progress has been made in general system optimization, the specific application to Apache Kafka remains an open area for further research [31]. The aim is to bridge the gap by proposing a novel approach that integrates machine learning-based log analysis for the auto-tuning of Kafka, thereby enhancing its throughput and overall performance.

3. METHODOLOGY

The section outlines the methodology for using machine learning techniques to auto-tune real-time data streaming platforms by analyzing GC and broker logs [1]. The approach involves several key steps: data collection and preprocessing, feature extraction, model training, and the application of regression models for dynamic configuration tuning [32].

Specifically, the study evaluates multiple regression algorithms, including Linear Regression, Ridge Regression, Lasso Regression, Support Vector Regression (SVR), Decision Tree Regression, Random Forest Regression, and Gradient Boosting Regression, to capture both linear and non-linear relationships in the data. The dataset, constructed from

GC and broker logs collected from AWS EC2-based Kafka deployments, comprises several thousand log entries that encapsulate throughput, latency, memory usage, and garbage collection metrics. By benchmarking a diverse set of algorithms on this dataset, the methodology ensures not only robust model selection but also provides a plug-and-play framework where different regression techniques can be substituted depending on workload characteristics and prediction requirements.

A comprehensive collection of 471 files, including 226 raw log files, 244 structured CSV files, and 1 Python preprocessing script, comprised the publicly accessible JVM Logs Dataset on Kaggle [13], from which the experimental data were sourced. The dataset yields 2,880 columns of features after translation, comprising 1,204 decimal-valued metrics (such as latency, throughput, and memory utilisation) and 1,676 integer-valued parameters (such as counts and frequencies). By capturing crucial data from Kafka broker logs (throughput, latency, and error counts) and GC logs (heap consumption, halt periods, and collection frequency), these capabilities allow for the methodical modelling of configuration–performance interactions under various workloads.

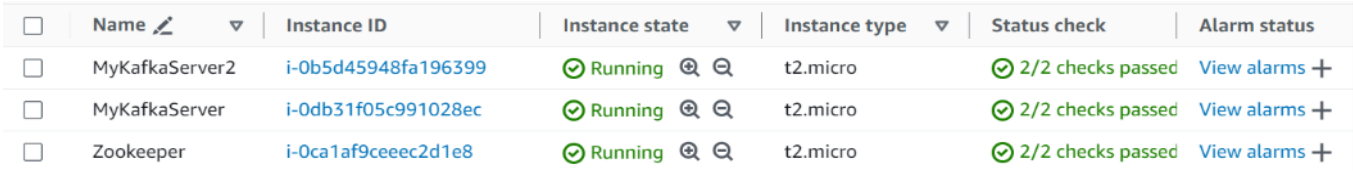

Figure 1. AWS EC2 instances

Table 1. The installation steps on AWS EC2

Steps	Description
Launch EC2 Instance	Launch an EC2 instance from the AWS Management Console. Select an Amazon Linux 2 AMI and a t2.micro instance type to stay within the Free Tier limits.
Connect to EC2 via SSH	Once the instance is running, connect to it via SSH using a terminal or an SSH client.
Update Package List	Update the package list using the package manager for the Linux distribution installed on the EC2 instance.
Install Java	Install Java, as Kafka requires it to run.
Download Kafka Binaries	Download the latest Kafka binaries from the official Apache Kafka website.
Extract Kafka Files	Extract the downloaded Kafka files to a directory on the EC2 instance.
Configure Kafka	Edit the server properties file to configure the Kafka server settings, such as broker ID, log directories, and Zookeeper connection string.
Start Zookeeper Service	Start the Zookeeper service, which Kafka depends on for coordination.
Start Kafka Broker	Start the Kafka broker, which will begin listening for incoming connections and processing messages.
Verify Installation	Verify the installation by creating a topic and sending test messages to ensure Kafka is functioning correctly.
Configure Security Groups	Configure security groups in the AWS Management Console to allow necessary inbound and outbound traffic for Kafka communication, ensuring appropriate network access.

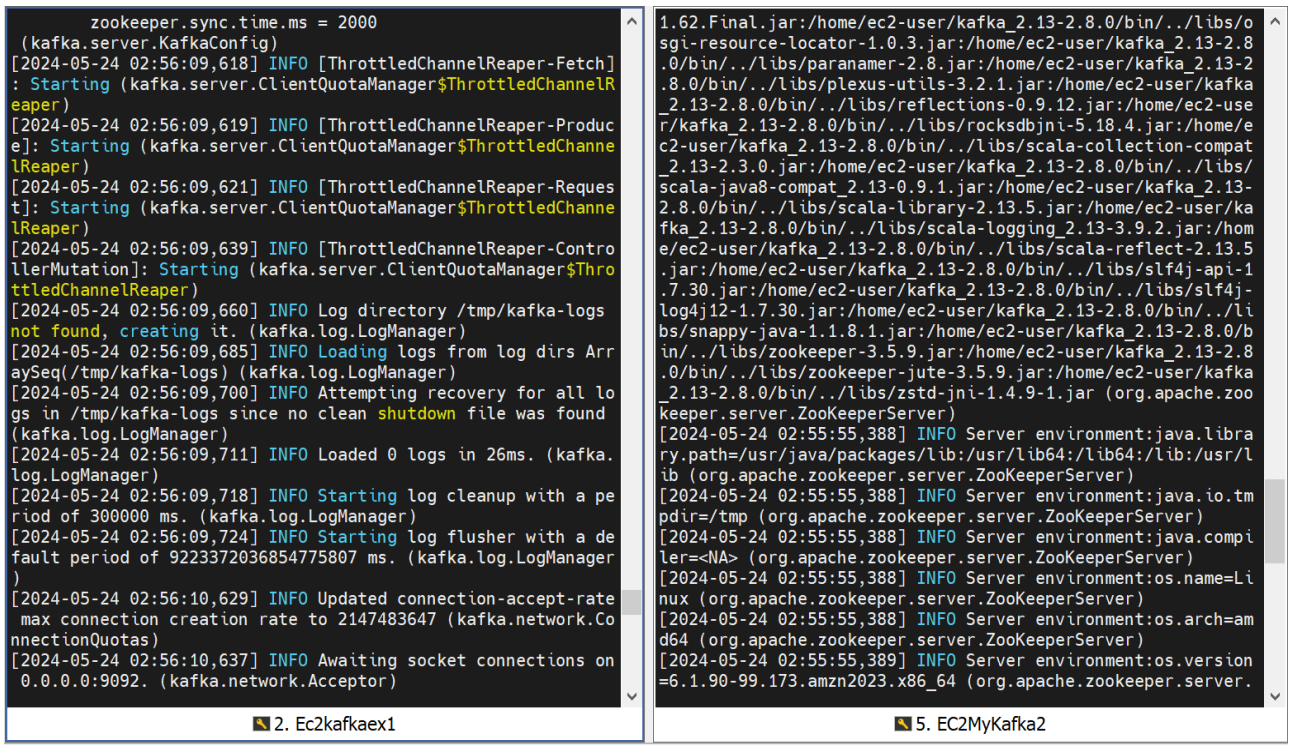

Figure 2. Zookeeper and Kafka broker running on AWS EC2

3.1 Data collection and preprocessing

Log collection: The first step involves collecting logs from the data streaming platform [33]. Specifically, gather garbage collection (GC) logs and broker logs. GC logs provide insights into memory management and garbage collection events, while broker logs contain information about broker activities, including message throughput, latency, and error rates. Figure 1 shows the AWS EC2 [34] instances that were provisioned with the steps mentioned in Table 1.

Preprocessing: Raw logs are often noisy and unstructured. Preprocess these logs to extract relevant data points. After the EC2 instances are up, Figure 2 shows the zookeeper [21] and Kafka broker running, which is used to capture the logs for further processing.

Data integration: The preprocessed data from GC and broker logs are integrated into a unified dataset. The dataset captures the system's state and performance metrics over time, providing a comprehensive view necessary for model training.

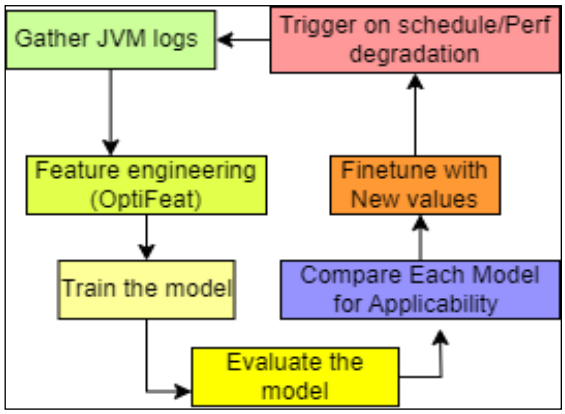

Figure 3. Optimized feature selection

3.2 Feature extraction

Metric selection: As shown in Figure 3, identified key performance metrics that influence the throughput and stability of the system. For GC logs, important metrics include GC pause times, heap usage, and frequency of garbage collection events [35]. For broker logs, the primary focus is on metrics such as message throughput, request latency, and error rates [10].

3.3 Model training

Regression models: Regression techniques are utilized to model the relationship between the extracted features and the performance metrics. Various regression models are considered, including linear regression, ridge regression, and more complex models like Random Forests and Gradient Boosting Machines [32]. For feature selection, we have adopted the OptiFeat approach, as detailed in prior work [36]. OptiFeat combines subject matter expertise with Recursive Feature Elimination (RFE) to ensure optimal feature selection, enhancing model interpretability and performance. The feature engineering process aligns with the methodology outlined in OptiFeat, providing only the most relevant features are retained for model training.

Training and validation: The dataset is split into training and validation sets (70:30). The training set is used to train the regression models. In contrast, the validation set is used to evaluate their performance [37]. Key steps include:

- Hyperparameter tuning using cross-validation to optimize model performance.
- Evaluating model accuracy using metrics such as R^2, MAE, and RMSE.
- Selecting the best-performing model based on validation results. The complete process is depicted in Figure 4.

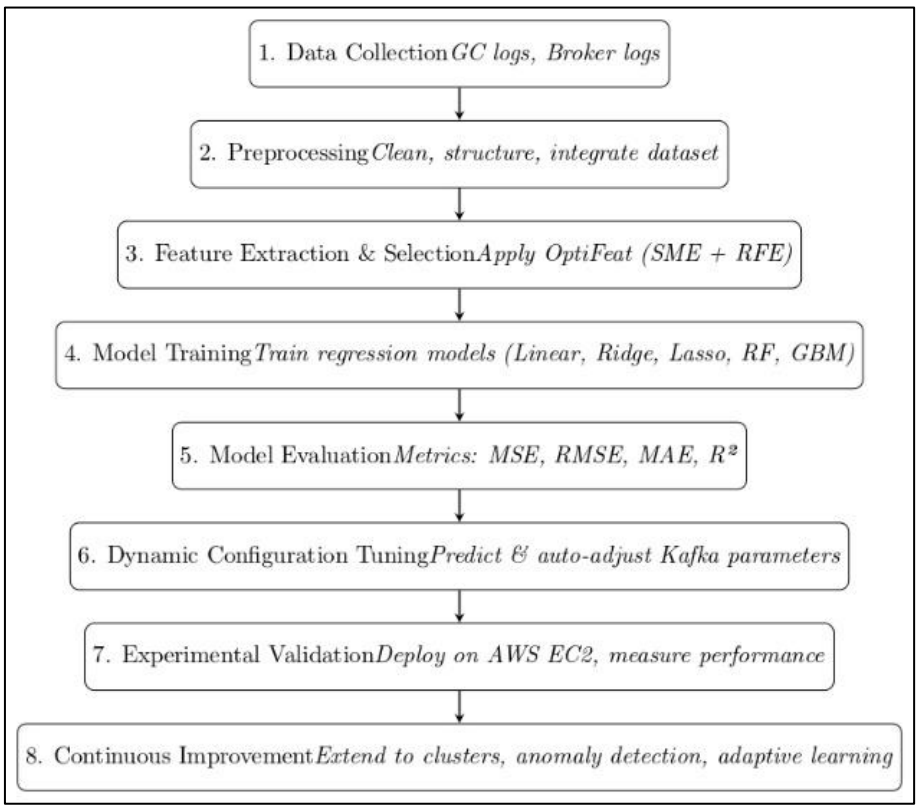

Figure 4. Procedure chart

3.4 Dynamic configuration tuning

Predictive analysis: The trained regression models are used to predict the impact of different configuration settings on system performance. By simulating various configurations, we identified settings that maximize throughput and minimize latency [9].

Automated tuning: Developed an automated tuning mechanism that adjusts the system's configuration parameters based on model predictions. The mechanism continuously monitors system performance and logs, dynamically updating configurations to maintain optimal performance. The steps involved are:

- Monitoring: Continuously collect and preprocess GC and broker logs.
- Prediction: Use regression models to predict performance under current settings.
- Adjustment: Automatically adjust configuration parameters based on predictions.
- Feedback Loop: Implement a feedback loop where the effects of configuration changes are monitored, and models are retrained periodically to adapt to evolving workloads.

Clear criteria, such as performance deterioration thresholds, prediction confidence ratings, and historical trend analysis, control the auto-tuning process. The model is guided by these measurements to determine the best time and way to make configuration changes, guaranteeing the system performance stays stable and adaptable to shifting workloads.

Implementation: The auto-tuning mechanism is implemented as a modular system that can be integrated with existing data streaming platforms. It includes components for log collection, feature extraction, model inference, and configuration management. The relatively poor performance of SVR and Decision Tree models can be attributed to both the characteristics of the dataset and model sensitivity. Since the underlying relationship between configuration parameters and throughput is largely linear, simpler regression models are better suited. In contrast, non-linear methods such as SVR and Decision Trees underperform without extensive hyperparameter tuning. This reinforces the importance of the proposed plug-and-play framework, which allows such models to be tested in baseline form while enabling future integration of optimized or alternative regression techniques.

4. EXPERIMENTAL SETUP

To validate the methodology, set up a series of experiments using a representative real-time data streaming environment. As the AWS t2.micro EC2 instance comes with 1 GB of RAM, we have to reduce the memory size of brokers.

Adopted methodology leverages machine learning techniques to provide an automated solution for tuning real-time data streaming platforms. By analyzing GC and broker logs and employing regression models, which can dynamically adjust configuration settings to optimize performance [38]. The proposed approach demonstrates significant improvements in throughput and stability, reducing the need for manual tuning and paving the way for more intelligent and autonomous data streaming

- Monitoring: Continuously collect and preprocess GC and broker logs.
- Prediction: Regression models to predict performance under current settings.
- Adjustment: Automatically adjust configuration parameters based on predictions.
- Feedback Loop: Implement a feedback loop where the effects of configuration changes are monitored, and models are retrained periodically to adapt to evolving workloads.

5. RESULT ANALYSIS

Have chosen a variety of regression techniques to provide a comprehensive comparison of both linear and non-linear models, as shown in Table 2, ensuring all potential relationships in the data are considered. Linear Regression serves as a baseline, offering simplicity and interpretability. Ridge and Lasso Regression [39] introduce regularization to handle multicollinearity and feature selection, potentially improving model performance. Non-linear models like SVR [40], Decision Tree Regression, Random Forest Regression, and Gradient Boosting Regression [32] were included to capture complex and non-linear relationships. The diverse selection allows us to benchmark performance across different approaches, revealing that Linear and Ridge Regression perform exceptionally well, indicating a strong linear relationship in the dataset. Non-linear models like Random Forest also showed good performance, suggesting they capture some additional patterns. The comprehensive approach helps in selecting the best model based on empirical results, ensuring robust and accurate predictions.

To evaluate the accuracy and reliability of regression models for performance prediction, several statistical metrics are employed. The most widely used is the MSE, depicted in Eq. (3):

$$MSE = \left(\frac{1}{n}\right) \Sigma_{\{i=1\}}^{n} (y_i - \hat{y}_i)^2 \tag{3}$$

As defined, it measures the average squared difference between the actual values y_i and the predicted values $\hat{y}_i$. Building on this, as shown in Eq. (4), the RMSE provides the error in the same units as the target variable by taking the square root of the MSE:

$$RMSE = \sqrt{\left(\frac{1}{n}\right) \Sigma_{\{i=1\}}^{n} (y_i - \hat{y}_i)^2} \tag{4}$$

Another useful metric is the MAE, depicted in Eq. (5), which captures the average magnitude of prediction errors without squaring them, making it less sensitive to outliers:

$$MAE = \left(\frac{1}{n}\right) \Sigma_{\{i=1\}}^{n} |y_i - \hat{y}_i| \tag{5}$$

Finally, the Coefficient of Determination (R^2) assesses how well the model explains the variance in the data and is expressed as in Eq. (6):

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} \left(y_i - \overline{y}_i\right)^2} \tag{6}$$

- y_i = actual observed value for the i^{th} data point $\hat{y}_i$ = predicted value for the i^{th} data point (from the regression model), $\overline{y}_i$ = mean of all observed values y_i.

- n = total number of observations.
- $\sum_{i=1}^{n}$ = Summation over all data points from i = 1 to n.

Among these, MSE and R^2 are particularly well-suited for system performance prediction tasks. MSE heavily penalizes large deviations, which is crucial for throughput predictions where significant errors can distort optimization decisions. R^2, on the other hand, provides an interpretable measure of the proportion of variance explained by the model, enabling straightforward comparison across different regression approaches. Together, they offer both absolute and relative perspectives on model performance, ensuring accurate and reliable assessment for auto-tuning distributed systems.

Table 2. Comparing different regression models

Model	MSE	R^2	RMSE	MAE	Performance Analysis
Linear Regression	4.84E-06	0.9999	0.0022	0.0014	Excellent fit; captures the relationship very well.
Ridge Regression	4.84E-06	0.9999	0.0022	0.0014	Excellent fit; similar to Linear Regression, indicates minimal regularization needed.
Lasso Regression	1.26E-05	0.9999	0.0035	0.0023	Perfect fit; slightly higher MSE, useful for feature selection.
Support Vector Regression	35.7065	0.6423	5.9755	3.8041	Poor performance; not suitable for the dataset.
Decision Tree Regression	4.8861	0.7752	2.2105	1.4072	Moderate performance; likely overfits, does not generalize well.
Random Forest Regression	0.3241	0.9850	0.5693	0.3624	Good performance; captures relationships well, less precise than linear models.
Gradient Boosting	1.6797	0.9227	1.2960	0.8251	Good performance; better than Decision Tree, but not as good as linear models.

The performance of various models is compared in Table 2. MSE and R^2 were chosen for their balance of simplicity, interpretability, and comprehensive insights into model performance [23]. MAE is less sensitive to outliers and provides a simple average error. MSE helps penalize significant errors. RMSE offers a more interpretable error in the original unit of measurement, and R^2 shows how well the model explains variance in the data, with values nearer 1 denoting a better fit. They provide a clear and concise way to evaluate and compare the regression models, ensuring robust and accurate analysis. From the comparative results, it is evident that Linear and Ridge Regression outperform other techniques with near-perfect accuracy and stability, making them the natural choice for this study. Lasso Regression, while slightly less accurate, provides added value for feature selection, whereas non-linear models such as Random Forest and Gradient Boosting captured additional patterns but did not surpass the precision of the linear approaches. Support Vector Regression and Decision Trees, on the other hand, showed weaker generalization and higher error rates, rendering them unsuitable for this dataset. Although Linear and Ridge Regression were ultimately selected for their superior performance and interpretability, the framework is intentionally designed to be algorithm-agnostic, allowing future researchers to plug in alternative regression or advanced learning models that may yield better results under different data distributions or workload conditions. The results of this study provide a framework for automated configuration tuning that has been proven to improve the performance of data streaming systems while reducing the need for manual intervention.

6. CONCLUSION

This study demonstrates the potential of machine learning–based approaches for auto-tuning distributed systems, with a focus on real-time data streaming platforms such as Apache Kafka. By establishing a plug-and-play framework for regression techniques, the work highlights how system performance metrics like throughput, latency, and stability can be systematically modeled and optimized. Although the current study concentrates on single-node deployments, the findings provide a strong foundation for extending the approach to more complex, distributed setups. The ability to generalize beyond Kafka to other performance-critical domains, including distributed databases, container orchestration platforms, and cloud resource management frameworks, further emphasizes the broader impact of this methodology.

The contributions of this work are threefold: (a) it provides empirical evidence that machine learning models can enhance Kafka's performance tuning, (b) it introduces a flexible plug-and-play framework that accommodates both linear and non-linear regression techniques, and (c) it demonstrates improvements in throughput prediction accuracy and system stability assessment. These findings contribute to the literature on automated system optimization and hold practical relevance for operators managing real-time streaming platforms.

7. FUTURE WORK

7.1 Extensions to streaming systems

Future research should expand the framework to multi-node Kafka clusters, where challenges such as leader election, partitioning, and cross-broker coordination add complexity to tuning. Exploring workload-aware auto-tuning under dynamic conditions and scaling in multi-tenant cloud environments will also be essential to ensure robustness and consistency in real-world deployments.

7.2 Cross-domain applications

The methodology can be extended beyond Kafka to other performance-critical domains, such as distributed databases, container orchestration platforms, and cloud resource management frameworks. Adapting the proposed approach to these systems can broaden its applicability and impact across diverse distributed infrastructures.

7.3 Methodological advances

On the methodological side, future work should investigate non-linear and ensemble models (e.g., Random Forests, Gradient Boosting) to capture more intricate relationships between configuration and performance. Incorporating advanced anomaly detection and adaptive optimization algorithms can enhance resilience under varying workloads. Furthermore, multi-objective optimization techniques will be critical to balance competing performance metrics such as throughput, latency, and resource utilization.

Recent studies confirm the importance of these directions: anomaly detection and adaptive learning have been widely explored in distributed systems optimization [41], while multi-objective optimization approaches are emerging as promising strategies for balancing performance trade-offs [42]. Finally, integrating these techniques into a fully automated CI/CD pipeline would enable real-time JVM parameter tuning, creating a scalable framework for continuous optimization in streaming environments.

REFERENCES

[1] Calderon, G., del Campo, G., Saavedra, E., Santamaría, A. (2024). Monitoring framework for the performance evaluation of an IoT platform with Elasticsearch and Apache Kafka. Information Systems Frontiers, 26(6): 2373-2389. https://doi.org/10.1007/s10796-023-10409-2

[2] Kroth, B., Matusevych, S., Zhu, Y. (2025). Autotuning systems: Techniques, challenges, and opportunities. In Companion of the 2025 International Conference on Management of Data, pp. 821-828. https://doi.org/10.1145/3722212.3725638

[3] Deva, S. (2025). Optimizing Apache Kafka for efficient data ingestion. World Journal of Advanced Engineering Technology and Sciences, 15(2): 1081-1091. https://doi.org/10.30574/wjaets.2025.15.2.0566

[4] Arega, K.L., Bagwari, A., Tune, K.K., Beyene, A.M., Rodriguez, C., Lezama, P., Salau, A.O. (2025). A deep learning-based approach for detecting Afan Oromo fake news on social media. Mathematical Modelling of Engineering Problems, 12(9): 3278-3288. https://doi.org/10.18280/mmep.120930

[5] Patil, Y., Fathima, R., Sundarajan, S., Sridevi Ponmalar, P., Ramachandran, H. (2024). Impact of feature selection on wheat yield prediction using machine learning. International Journal of Design & Nature and Ecodynamics, 19(6): 1909-1917. https://doi.org/10.18280/ijdne.190607

[6] Toderean, L., Daian, M., Cioara, T., Anghel, I., Michalakopoulos, V., Sarantinopoulos, E., Sarmas, E. (2025). Heuristic based federated learning with adaptive hyperparameter tuning for households energy prediction. Scientific Reports, 15(1): 12564. https://doi.org/10.1038/s41598-025-96443-3

[7] Van Aken, D., Pavlo, A., Gordon, G.J., Zhang, B. (2017). Automatic database management system tuning through large-scale machine learning. In Proceedings of the 2017 ACM International Conference on Management of Data, pp. 1009-1024. https://doi.org/10.1145/3035918.3064029

[8] Xue, W., Roy, C.J. (2023). Machine learning-driven autotuning of graphics processing unit accelerated computational fluid dynamics for enhanced performance. arXiv preprint arXiv:2306.14011. https://doi.org/10.48550/arXiv.2306.14011

[9] Eldor, E. (2023). RAM allocation in Kafka clusters: Performance, stability, and optimization strategies. In Kafka Troubleshooting in Production: Stabilizing Kafka Clusters in the Cloud and On-premises, pp. 63-84. https://doi.org/10.1007/978-1-4842-9490-1_6.

[10] Eldor, E. (2023). Kafka Troubleshooting in Production. Springer Books. https://doi.org/10.1007/978-1-4842-9490-1

[11] Vyas, S., Tyagi, R.K., Jain, C., Sahu, S. (2021). Literature review: A comparative study of real time streaming technologies and Apache Kafka. In 2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT), Sonepat, India, pp. 146-153. https://doi.org/10.1109/CCICT53244.2021.00038

[12] Choudhary, A., Govil, M.C., Singh, G., Awasthi, L.K., Pilli, E.S., Kapil, D. (2017). A critical survey of live virtual machine migration techniques. Journal of Cloud Computing, 6(1): 23. https://doi.org/10.1186/s13677-017-0092-1

[13] JVM logs. https://www.kaggle.com/datasets/vijayakumargundappa/jvm-logs, accessed on May 20, 2024.

[14] Nanda, S.K., Chaudhary, D.K. (2024). Machine Learning: Principles, Algorithms, and Tools. Addition Publishing House.

[15] Mehta, S., Patnaik, K.S. (2021). Improved prediction of software defects using ensemble machine learning techniques. Neural Computing and Applications, 33(16): 10551-10562. https://doi.org/10.1007/s00521-021-05811-3

[16] Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaugh, M., Poulosky, P. (2016). Benchmarking streaming computation engines: Storm, Flink and spark streaming. In 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, IL, USA, pp. 1789-1792. https://doi.org/10.1109/IPDPSW.2016.138

[17] Bagla, P., Kumar, K. (2023). A rule-based fuzzy ant colony improvement (ACI) approach for automated disease diagnoses. Multimedia Tools and Applications, 82(24): 37709-37729. https://doi.org/10.1007/s11042-023-15115-4

[18] Austin, A.M., Ramkumar, N., Gladders, B., Barnes, J.A., Eid, M.A., Moore, K.O., Goodney, P.P. (2022). Using a cohort study of diabetes and peripheral artery disease to compare logistic regression and machine learning via random forest modeling. BMC Medical Research Methodology, 22(1): 300. https://doi.org/10.1186/s12874-022-01774-8

[19] Balasubramanian, S., Ghosal, D., Sharath, K.N.B., Pouyoul, E., Sim, A., Wu, K., Tierney, B. (2018). Auto-tuned publisher in a pub/sub system: Design and performance evaluation. In 2018 IEEE International Conference on Autonomic Computing (ICAC), Trento, Italy, pp. 21-30. https://doi.org/10.1109/ICAC.2018.00012

[20] Mao, H., Alizadeh, M., Menache, I., Kandula, S. (2016). Resource management with deep reinforcement learning. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks, pp. 50-56.

https://doi.org/10.1145/3005745.3005750
[21] Pithode, K., Patheja, P.S. (2023). Analyzing system logs of big data distributed environment: A review. In International Conference on Data Science and Big Data Analysis, pp. 433-446. https://doi.org/10.1007/978-981-99-9179-2_34
[22] Singh, P., Adebanjo, A., Shafiq, N., Razak, S.N.A., Kumar, V., Farhan, S.A., Sergeevna, M.T. (2024). Development of performance-based models for green concrete using multiple linear regression and artificial neural network. International Journal on Interactive Design and Manufacturing (IJIDeM), 18(5): 2945-2956. https://doi.org/10.1007/s12008-023-01386-6
[23] Hodson, T.O. (2022). Root mean square error (RMSE) or mean absolute error (MAE): When to use them or not. Geoscientific Model Development Discussions, 15(14): 5481-5487. https://doi.org/10.5194/gmd-15-5481-2022
[24] Nourbakhsh, Z., Habibi, N. (2023). Combining LSTM and CNN methods and fundamental analysis for stock price trend prediction. Multimedia Tools and Applications, 82(12): 17769-17799. https://doi.org/10.1007/s11042-022-13963-0
[25] Ozili, P.K. (2023). The acceptable R-square in empirical modelling for social science research. In Social research Methodology and Publishing Results: A Guide to Non-Native English Speakers, pp. 134-143. https://doi.org/10.4018/978-1-6684-6859-3.ch009
[26] Driscoll, L., de la Torre, S., Gomez-Ruiz, J.A. (2022). Feature-based lithium-ion battery state of health estimation with artificial neural networks. Journal of Energy Storage, 50: 104584. https://doi.org/10.1016/j.est.2022.104584
[27] William, P., Paithankar, D.N., Yawalkar, P.M., Korde, S.K., Rajendra, A., Rakshe, D.S. (2023). Divination of air quality assessment using ensembling machine learning approach. In 2023 International Conference on Artificial Intelligence and Knowledge Discovery in Concurrent Engineering (ICECONF), Chennai, India, pp. 1-10. https://doi.org/10.1109/ICECONF57129.2023.10083751
[28] Deng, A. (2023). Database task processing optimization based on performance evaluation and machine learning algorithm. Soft Computing-A Fusion of Foundations, Methodologies & Applications, 27(10): 6811-6821. https://doi.org/10.1007/s00500-023-08111-1
[29] Nabi, S., Ahmad, M., Ibrahim, M., Hamam, H. (2022). AdPSO: Adaptive PSO-based task scheduling approach for cloud computing. Sensors, 22(3), 920. https://doi.org/10.3390/s22030920
[30] Vijayakumar, G., Bharathi, R.K. (2022). Predicting JVM parameters for performance tuning using different regression algorithms. In 2022 Fourth International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT), Mandya, India, pp. 1-8. https://doi.org/10.1109/ICERECT56837.2022.10060788
[31] Vijayakumar, G., Bharathi, R.K. (2023). Streaming big data with open-source: A comparative study and architectural recommendations. In 2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS), Erode, India, pp. 1420-1425. https://doi.org/10.1109/ICSCDS56580.2023.10105025
[32] Tuniya, N., Parihar, M., Patil, S., Lawand, K., Nawale, H. (2022). Comparative analysis of regressor models on non-invasive blood glucose dataset. In Proceedings of International Conference on Computing and Communication Networks: ICCCN 2021, pp. 209-217. https://doi.org/10.1007/978-981-19-0604-6_19
[33] Raptis, T.P., Passarella, A. (2023). A survey on networked data streaming with Apache Kafka. IEEE Access, 11: 85333-85350. https://doi.org/10.1109/ACCESS.2023.3303810
[34] Mishra, P. (2023). Advanced AWS services. In Cloud Computing with AWS: Everything You Need to Know to be an AWS Cloud Practitioner, pp. 247-277. https://doi.org/10.1007/978-1-4842-9172-6_9
[35] Beronić, D., Novosel, N., Mihaljević, B., Radovan, A. (2022). Assessing contemporary automated memory management in Java–garbage first, Shenandoah, and Z garbage collectors comparison. In 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, pp. 1495-1500. https://doi.org/10.23919/MIPRO55190.2022.9803445
[36] Vijayakumar, G., Bharathi, R.K. (2024). OptiFeat: Enhancing feature selection, a hybrid approach combining subject matter expertise and recursive feature elimination method. Discover Computing, 27(1): 44. https://doi.org/10.1007/s10791-024-09483-0
[37] Douiba, M., Benkirane, S., Guezzaz, A., Azrour, M. (2023). Anomaly detection model based on gradient boosting and decision tree for IoT environments security. Journal of Reliable Intelligent Environments, 9(4): 421-432. https://doi.org/10.1007/S40860-022-00184-3/METRICS
[38] Gupta, A., Jain, S. (2022). Optimizing performance of Real-Time Big Data stateful streaming applications on Cloud. In 2022 IEEE International Conference on Big Data and Smart Computing (BigComp), Daegu, Korea, pp. 1-4. https://doi.org/10.1109/BigComp54360.2022.00010
[39] Zhang, Y., Politis, D.N. (2023). Debiased and thresholded ridge regression for linear models with heteroskedastic and correlated errors. Journal of the Royal Statistical Society Series B: Statistical Methodology, 85(2): 327-355. https://doi.org/10.1093/jrsssb/qkad006
[40] Dash, R.K., Nguyen, T.N., Cengiz, K., Sharma, A. (2023). Fine-tuned support vector regression model for stock predictions. Neural Computing and Applications, 35(32): 23295-23309. https://doi.org/10.1007/s00521-021-05842-w
[41] Moriano, P., Hespeler, S.C., Li, M., Mahbub, M. (2025). Adaptive anomaly detection for identifying attacks in cyber-physical systems: A systematic literature review. Artificial Intelligence Review, 58(9): 283. https://doi.org/10.1007/s10462-025-11292-w
[42] Harkare, V., Mangrulkar, R., Thorat, O., Jain, S.R. (2024). Evolutionary approaches for multi-objective optimization and pareto-optimal solution selection in data analytics. In Applied Multi-Objective Optimization, pp. 67-94. https://doi.org/10.1007/978-981-97-0353-1_4