ARTIFICIAL INTELLIGENCE AND BEYOND: APPLIED APPROACHES IN COMPUTER SCIENCE

Editor: Dr. Songül KARAKUŞ

DUVAR

ARTIFICIAL INTELLIGENCE AND BEYOND: APPLIED APPROACHES IN COMPUTER SCIENCE

Editor

Dr. Songül KARAKUŞ

Artificial Intelligence and Beyond: Applied Approaches in Computer Science
Editor: Dr. Songül KARAKUŞ

Editor in chief: Berkan Balpetek
Cover and Page Design: Duvar Design
Printing: December 2025
Publisher Certificate No: 49837
ISBN: 978-625-8698-72-5

853 Sokak No:13 P.10 Kemeraltı-Konak/İzmir
Tel: 0 232 484 88 68
www.duvaryayinlari.com
duvarkitabevi@gmail.com

TABLE OF CONTENTS

Chapter 1

Towards Intelligent Modern Agriculture: Transfer Learning-Powered Deep Learning Models For Comparative Classification Of Lettuce Diseases

Mehmet BURUKANLI[1]
Musa CIBUK[2]
Davut ARI[3]

ABSTRACT

Lettuce is among the most popular vegetables produced and consumed worldwide. Unfortunately, efficient lettuce production is negatively impacted by environmental pollution and other external physical factors. Early detection of diseases in lettuce and preventing them from spreading to others are among the most crucial factors in growing productive and healthy lettuce. In traditional lettuce cultivation, this process is performed manually, with a high error rate and difficult control. Using Artificial Intelligence (AI)-based tools is crucial to overcome these challenges. AI-based models can help increase lettuce productivity by detecting lettuce diseases at an early stage. Therefore, in this study, we used 20 deep transfer learning models to detect early-stage lettuce diseases. Among these models, the AlexNet model achieved the highest accuracy of 97.88%. Furthermore, the explainability of deep learning approaches was enhanced by the use of Grad-CAM-based heat maps to demonstrate whether each model's outputs are based on meaningful regions in the image. Experimental results support the ability of transfer learning-based models to detect lettuce diseases at an early stage, thereby significantly improving production efficiency.

Keywords: Lettuce disease detection, AlexNet model, Decision support systems, Transfer learning, Grad-CAM

[1]Lecturer Dr., Bitlis Eren University, Rectorate, Department of Common Courses, mburukanli@beu.edu.tr, ORCID: 0000-0003-4459-0455.
[2]Doc. Dr., Bitlis Eren University, Faculty of Engineering and Architecture, Computer Engineering, mcibuk@beu.edu.tr, ORCID:0000-0001-9028-2221.
[3]Assist. Prof. Dr., Bitlis Eren University, Faculty of Engineering and Architecture, Computer Engineering, dari@beu.edu.tr, ORCID:0000-0001-6439-7957.

1. INTRODUCTION

Agricultural production is being negatively impacted worldwide by factors such as rapid population growth, environmental pollution, and climate change. To address these challenges, modern technologies such as artificial intelligence, which enable more efficient and sustainable production, are essential. Adverse physical conditions, particularly high temperatures and low humidity, lead to significant yield losses. Consequently, disease detection takes time, and the spread of disease to healthy lettuce plants is accelerated.(Nafil et al., 2023)(Rathor, Choudhury, Sharma, Nautiyal, et al., 2025). Deep learning-based architectures, especially for disease detection, classification, and real-time object detection, such as YOLO(Upadhyay et al., 2025)(Qadri et al., 2025)(Wang et al., 2024)(Zhang & Li, 2022) and Convolutional Neural Networks (CNNs) (Qadri et al., 2025)(Gang et al., 2022)(Rathor, Choudhury, Sharma, Shah, et al., 2025), are frequently preferred. However, since they consist of millions of parameters, their training takes some time. For this reason, it has become possible to come across lightweight architectures in the literature (Lin et al., 2022). Artificial intelligence-based decision support systems are frequently preferred in lettuce disease detection, as in almost every field (Qin et al., 2025)(Rathor, Choudhury, Sharma, Nautiyal, et al., 2025).

This proved that deep learning models could be used in lettuce detection. Nafil et al. (Nafil et al., 2023) proposed a CNN-based model for the early detection of lettuce diseases. Using this model, they achieved 94% accuracy. Kumaratenna et al. (Kumaratenna & Cho, 2024) achieved high performance on a lettuce dataset using a deep learning-based model. Yang et al. (Yang et al., 2023) classified lettuce leaves using machine learning-based models such as Multiple Linear Regression (MLR), K-Nearest Neighbors (KNN), and SVM. They observed that the SCM model provided satisfactory performance. Rathor et al. (Rathor, Choudhury, Sharma, Nautiyal, et al., 2025) proposed the Conv-7 DCNN model for the detection of lettuce diseases. They compared this model with other deep learning models. Their proposed model achieved significant results. Rathor et al. (Rathor, Choudhury, Sharma, Shah, et al., 2025) proposed the CNN-WOPNet model for the detection of nutrient deficiencies in lettuce diseases. They also compared this model with other deep learning models. The model they proposed has achieved remarkable results. Flores et al. (Flores et al., 2023) used deep learning-based models to classify lettuce samples. Their MobileNet+SVM-based hybrid model achieved remarkable results. Pratondo et al.(Pratondo et al., 2023) classified lettuce leaves using deep learning-based models. They also achieved significant results with the support of transfer learning. Upadhyay et al. (Upadhyay et al., 2025) detected plant diseases with

high accuracy using deep learning-based approaches. Zhang et al. (Zhang & Li, 2022) proposed the VOLO-D1 model to classify five different lettuce varieties. This model achieved very successful results.

In this study, 20 artificial intelligence-based models (AlexNet, VGG16, VGG19, GoogLeNet, Places365, ResNet18, ResNet50, ResNet101, Inceptionv3, Inception-ResNet v2, Xception, MobileNetv2, DenseNet201, ShuffleNet, Darknet19, Darknet53, SqueezeNet, EfficientNet-B0, NASNet-Mobile, and NASNet-Large) were used to detect healthy and unhealthy leaves on lettuce. These models were compared with each other in terms of accuracy, precision, recall, specificity, and F1 scores. 20 deep learning models were compared with each other on a lettuce dataset. The results showed that the AlexNet model outperformed the other models.

2. MATERIALS AND METHODS

2.1. Lettuce Dataset

The Lettuce dataset used in this study consists of two classes: "Healthy" and "Unhealthy." The "Healthy" class contains 326 image samples, while the "Unhealthy" class contains 381 image samples (Kaggle, n.d.). Some of these image samples in the Lettuce dataset are shown in Figure 1.

Figure 1. Some of these image samples in the Lettuce dataset (Kaggle, n.d.)

2.2. Deep Transfer Learning Models

In this section, we used the following models, AlexNet, VGG16, VGG19, GoogLeNet, Places365, ResNet18, ResNet50, ResNet101, Inceptionv3, Inception-ResNet v2, Xception, MobileNetv2, DenseNet201, ShuffleNet, Darknet19, Darknet53, SqueezeNet, EfficientNet-B0, NASNet-Mobile, and NASNet-Large, which are frequently used and have proven successful in the literature for lettuce disease detection. These models are particularly successful in classification and computer vision tasks.

3. RESULTS

3.1. Performance Metrics and Evaluation Methods

In this work, to quantify the performance of each AI-based model, we used the following performance metrics: Accuracy, Precision, Recall, Specificity, and F1 score. The formulas for these metrics are given in Equations 1, 2, 3, 4, and 5, respectively (Burukanlı & Ari, 2025)(Burukanlı & Çıbuk, 2024).

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP+FN} \quad (3)$$

$$\text{Specificity} = \frac{TN}{TN+FP} \quad (4)$$

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (5)$$

3.2. Experimental Setup

In this study, the optimizer was set to stochastic gradient descent with momentum (SGDM), learning rate to 0.001, epochs to 25, and batch size to 32 during training of all deep learning models. For the experimental computation of this study, HP-Z840 workstation with 10 cores, 2 x Intel CPU (Xeon E52687Wv3), 64 GB Ram and Quadro P5000 GPU was used.

3.3. Dataset-Level Heatmap Analysis of the Models

In this study, we used the Grad-CAM heat mapping technique to analyze in detail the regions of the image where the deep transfer learning models focused during the training phase and to understand the explainability of the models. This technique allows us to obtain more information about the reliability of the models. Visualizing the regions of the image where each model focused is known to increase the explainability of the methods (Raghavan et al., 2023). Dataset heatmap of models is given in Figure 2.

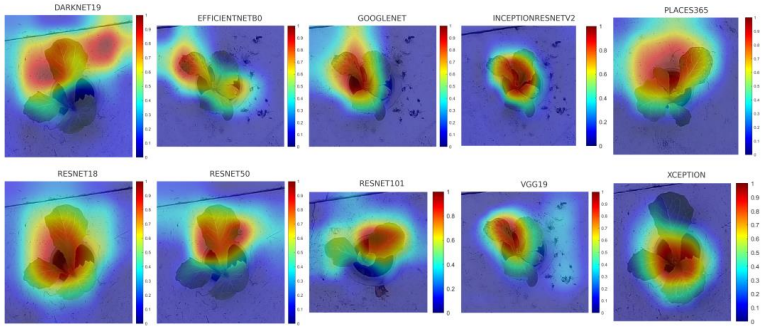


Figure 2. Dataset heatmap of models

3.4. Experimental Results

As illustrated in Figure 3, the AlexNet model demonstrated a high level of discriminatory capability by correctly classifying 320 out of 326 samples in the 'healthy' class, resulting in only 6 misclassifications. Likewise, the model accurately identified 372 out of 381 samples in the 'unhealthy' class, with merely 9 instances incorrectly predicted. These outcomes underscore the robustness and reliability of AlexNet in distinguishing between healthy and diseased lettuce leaves. As seen in Figure 3, the lettuce dataset consists of two classes, comprising 326 healthy and 381 unhealthy samples. Due to the VGG16 architecture producing NaN outputs in the Fold-1, Fold-4, and Fold-5 stages, these sublayers were excluded from the evaluation and were not included in the corresponding confusion matrix analyses. Similarly, the NaN values observed in the Fold-3 and Fold-4 stages of the VGG19 architecture indicate that the model was unable to perform reliable classification in these layers; therefore, these results were excluded from the confusion matrix analysis. In addition, DenseNet201 and DarkNet53 exhibit the lowest misclassification rates and the highest overall performance, while GoogLeNet, EfficientNetB0, NASNet-Large, and MobileNetV2 maintain stable accuracy levels around 94–95%. In contrast, Inception-based architectures show noticeably higher false prediction counts, particularly for the healthy class, indicating reduced generalization capability. Furthermore, a comparative evaluation of the accuracy performance of all models assessed on the lettuce dataset is provided in Figure 3.

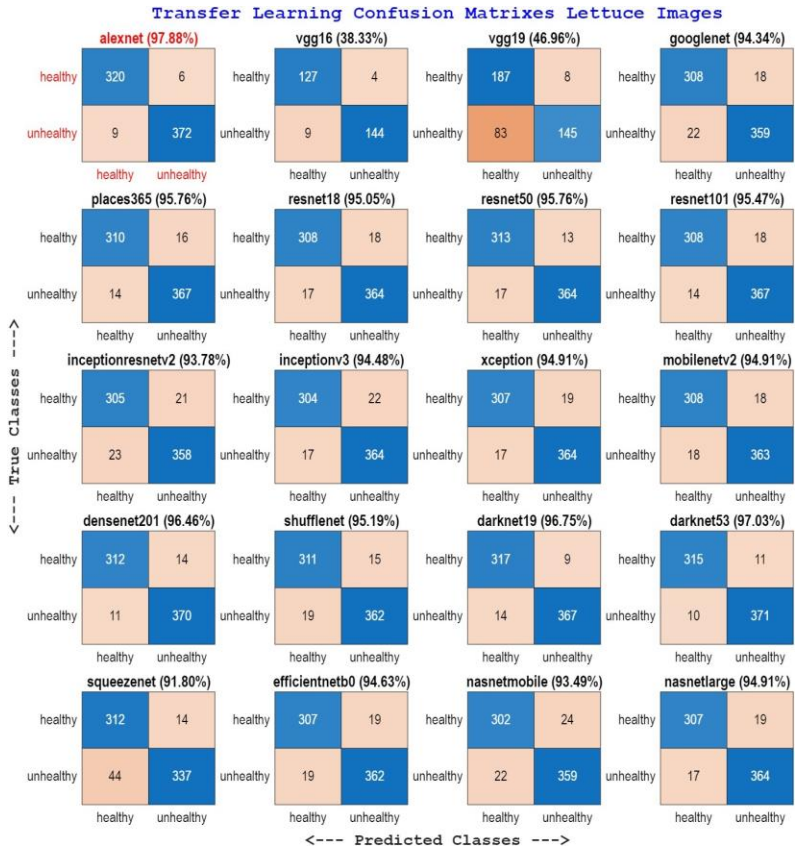


Figure 3. Confusion matrices obtained by selected CNN architectures, including AlexNet, DarkNet53, DarkNet19, and DenseNet20 etc. on the lettuce dataset.

The accuracy-loss graph obtained on the lettuce training dataset for each Fold of the AlexNet model depending on the number of epochs is shown in Figure 4.

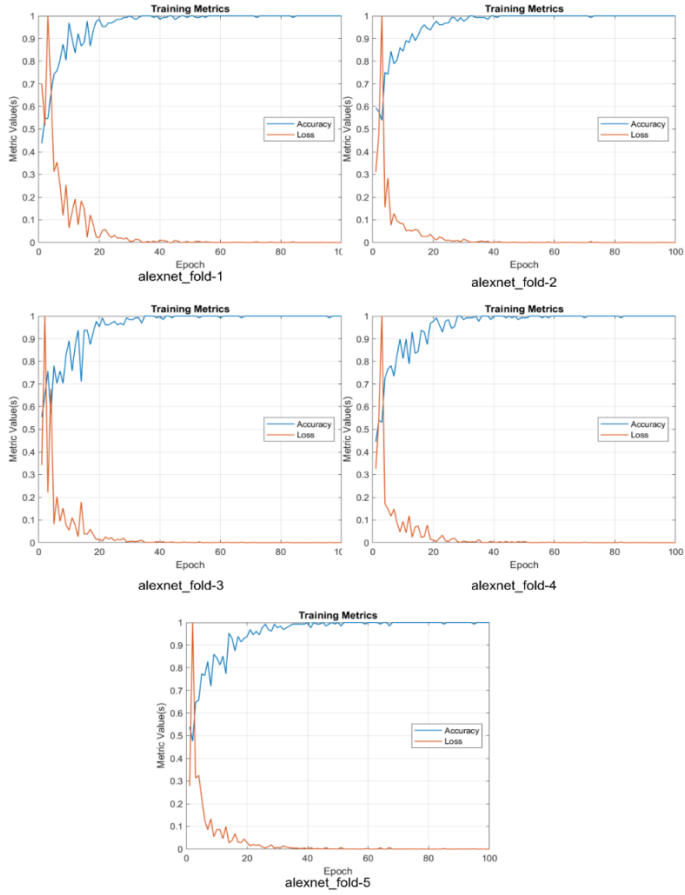


Figure 4. Accuracy-Loss graph for each Fold of the AlexNet model depending on the number of epochs

As shown in Figure 4, the accuracy value of the AlexNet model increased, especially after the 40th epoch, and the loss decreased accordingly. This means that the AlexNet model was quite successful in detecting the lettuce dataset. The resulting roc curve graph obtained on the lettuce training dataset for each Fold of the AlexNet model is shown in Figure 5.

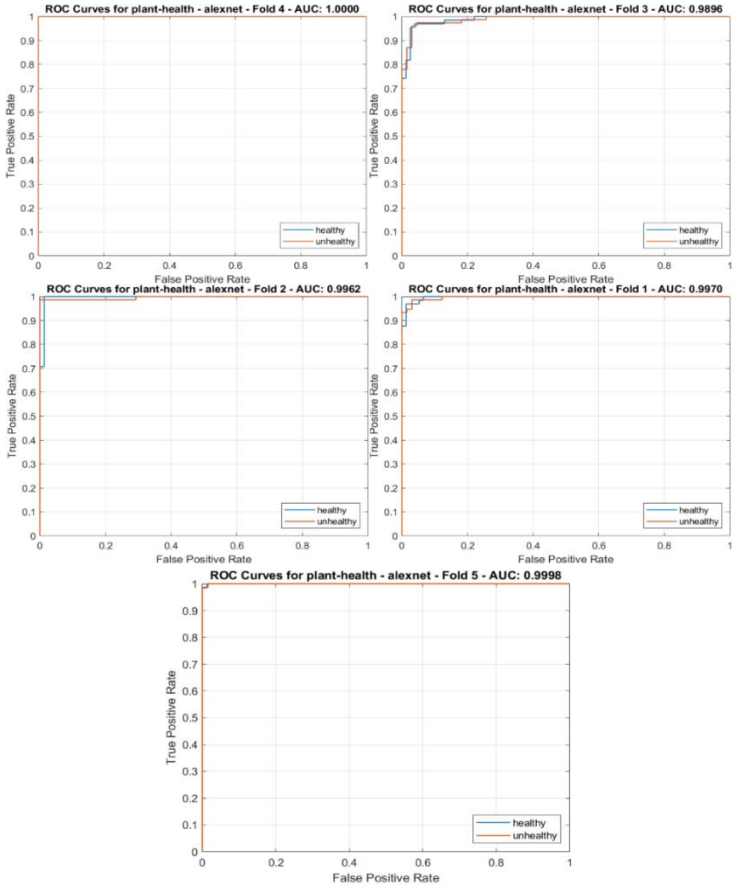


Figure 5. The resulting roc curve graph for each Fold of the AlexNet model

As shown in Figure 5, the AUC value obtained by the AlexNet model for Fold 1 was 0.9970, while the AUC value obtained for Fold 2 was 0.9962. Similarly, the AUC value obtained by the AlexNet model for Fold 3 was 0.9896, while the AUC value obtained for Fold 2 was 1.000. In addition, the AUC value obtained by the AlexNet model for Fold 5 was 0.9998. The resulting average roc curve graph obtained on the lettuce dataset of the AlexNet model is shown in Figure 6.

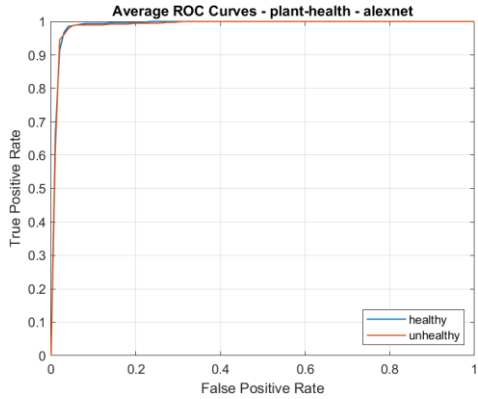


Figure 6. The resulting average roc curve graph of the AlexNet model

As shown in Figure 6, the average ROC curve obtained from the AlexNet model on the lettuce dataset is nearly equal to 1.000, indicating the model's strong robustness and its ability to achieve highly reliable discrimination between healthy and diseased samples. The confusion matrix produced by the AlexNet model for the same dataset is presented in Figure 3, further illustrating its accurate classification capability with minimal false positives and false negatives. These results collectively demonstrate that AlexNet is one of the most stable and effective architectures for lettuce disease identification within the scope of this study. Comparison of all models in terms of accuracy on the lettuce dataset is given in Figure 7.

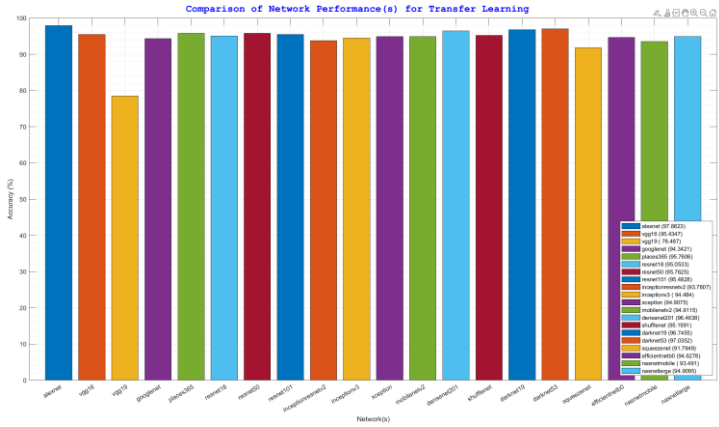
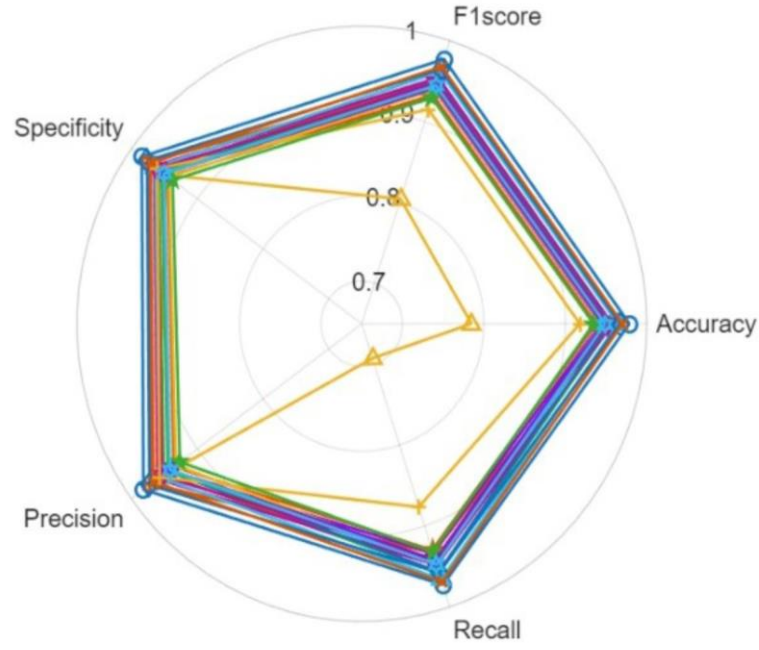


Figure 7. Comparison of all models in terms of accuracy on the lettuce data set

As shown in Figure 7, among 20 models on the lettuce dataset, the AlexNet model achieved the best result with an accuracy rate of 97.8823%, while the VGG19 model achieved the worst result with an accuracy rate of 78.487%.

Furthermore, the performance rates of the other models were between AlexNet and VGG19. Detailed comparison and Radar image of 20 deep learning based models on lettuce dataset is shown in Figure 8.



alexnet	Acc=0.9788	F1=0.9771	Spe=0.9841	Pre=0.9816	Rec=0.9726
darknet53	Acc=0.9703	F1=0.9677	Spe=0.9712	Pre=0.9663	Rec=0.9692
darknet19	Acc=0.9675	F1=0.9650	Spe=0.9761	Pre=0.9724	Rec=0.9577
densenet201	Acc=0.9646	F1=0.9615	Spe=0.9635	Pre=0.9571	Rec=0.9659
places365	Acc=0.9576	F1=0.9538	Spe=0.9582	Pre=0.9509	Rec=0.9568
resnet50	Acc=0.9576	F1=0.9543	Spe=0.9655	Pre=0.9601	Rec=0.9485
resnet101	Acc=0.9547	F1=0.9506	Spe=0.9532	Pre=0.9448	Rec=0.9565
vgg16	Acc=0.9542	F1=0.9513	Spe=0.9730	Pre=0.9695	Rec=0.9338
shufflenet	Acc=0.9519	F1=0.9482	Spe=0.9602	Pre=0.9540	Rec=0.9424
resnet18	Acc=0.9505	F1=0.9462	Spe=0.9529	Pre=0.9448	Rec=0.9477
xception	Acc=0.9491	F1=0.9446	Spe=0.9504	Pre=0.9417	Rec=0.9475
mobilenetv2	Acc=0.9491	F1=0.9448	Spe=0.9528	Pre=0.9448	Rec=0.9448
nasnetlarge	Acc=0.9491	F1=0.9446	Spe=0.9504	Pre=0.9417	Rec=0.9475
efficientnetb0	Acc=0.9463	F1=0.9417	Spe=0.9501	Pre=0.9417	Rec=0.9417
inceptionv3	Acc=0.9448	F1=0.9397	Spe=0.9430	Pre=0.9325	Rec=0.9470
googlenet	Acc=0.9434	F1=0.9390	Spe=0.9523	Pre=0.9448	Rec=0.9333
inceptionresnetv2	Acc=0.9378	F1=0.9327	Spe=0.9446	Pre=0.9356	Rec=0.9299
nasnetmobile	Acc=0.9349	F1=0.9292	Spe=0.9373	Pre=0.9264	Rec=0.9321
squeezenet	Acc=0.9180	F1=0.9150	Spe=0.9601	Pre=0.9571	Rec=0.8764
vgg19	Acc=0.7849	F1=0.8043	Spe=0.9477	Pre=0.9590	Rec=0.6926

Figure 8. Detailed comparison of 20 deep learning-based models on lettuce dataset

As shown in Figure 8, 20 transfer learning models were comparatively evaluated using performance metrics including Accuracy (Acc), F1-score (F1), Specificity (Spe), Precision (Pre), and Recall (Rec). A detailed examination of the results reveals that the AlexNet model outperformed all other architectures, achieving 0.9788 Acc, 0.9771 F1, 0.9841 Spe, 9816 Pre and 0.9726 Rec values. In contrast, the VGG19 model obtained noticeably weaker performance relative to the other models, with 0.7849 Acc value, 0.8043 F1 value and 0.6926 Rec

value. Additionally, NASNet-Mobile achieved the lowest both Spe values at 0.9373 and Pre values at 0. 9264.

4. CONCLUSION

In this study, 20 deep transfer learning models were used to detect lettuce leaf diseases. These 20 DL models were compared with each other on a lettuce dataset. The results obtained indicated that the AlexNet model outperformed the other models with an accuracy of 97.88%. Additionally, the Grad-CAM technique was used to identify the significant regions obtained by each deep learning model on the dataset. Experimental findings indicate that AI-based models, particularly the AlexNet model, achieve significant results in lettuce disease detection. In the next study, we plan to perform disease detection using state-of-the-art DL models on different lettuce datasets.

REFERENCES

Burukanlı, M., & Ari, D. (2025). BRAIN CANCER PREDICTION USING DEEP TRANSFER LEARNING MODELS. *ASES IX. INTERNATIONAL SCIENTIFIC RESEARCH CONGRESS*, 184–192. https://www.researchgate.net/publication/392208728_BRAIN_CANCER_PREDICTION_USING_DEEP_TRANSFER_LEARNING_MODELS

Burukanlı, M., & Çıbuk, M. (2024). Intrusion Detection and Performance Analysis Using Copula Functions. *Bitlis Eren Üniversitesi Fen Bilimleri Dergisi*, *13*(4), 1335–1354. https://doi.org/10.17798/bitlisfen.1561354

Flores, E. J. C., Gonzaga, J. A., Augusto, G. L., Chua, J. A. T., & Gan Lim, L. A. (2023). Deep Learning-Based Vision System for Water Stress Classification of Lettuce in Pot Cultivation. *2023 IEEE 15th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management, HNICEM 2023*, 1–6. https://doi.org/10.1109/HNICEM60674.2023.10589156

Gang, M.-S., Kim, H.-J., & Kim, D.-W. (2022). Estimation of Greenhouse Lettuce Growth Indices Based on a Two-Stage CNN Using RGB-D Images. *Sensors*, *22*(15), 5499. https://doi.org/10.3390/s22155499

Kaggle. (n.d.). *Lettuce NPK dataset*. Retrieved December 2, 2025, from https://www.kaggle.com/datasets/baronn/lettuce-npk-dataset/data

Kumaratenna, K. P. S., & Cho, Y. Y. (2024). Detection of Tipburn Stress on Lettuce Grown in a Plant Factory using Artificial Intelligence (AI) Models. *Horticultural Science and Technology*, *42*(6), 711–724. https://doi.org/10.7235/HORT.20240059

Lin, Z., Fu, R., Ren, G., Zhong, R., Ying, Y., & Lin, T. (2022). Automatic monitoring of lettuce fresh weight by multi-modal fusion based deep learning. *Frontiers in Plant Science*, *13*. https://doi.org/10.3389/fpls.2022.980581

Nafil, K., Saufi, A., Hdili, O., Faqihi, S., Maghraoui, H., Kobbane, A., & Koutbi, M. El. (2023). Lettuce Leaf Disease Protection and Detection Using Image Processing Technique. *Proceedings - 10th International Conference on Wireless Networks and Mobile Communications, WINCOM 2023*, 1–6. https://doi.org/10.1109/WINCOM59760.2023.10323013

Pratondo, A., Novianty, A., & Fauzi, H. (2023). Classification of Lettuce Leaf Variants Using Transfer Learning. *Proceedings - 2023 3rd International Conference on Electronic and Electrical Engineering and Intelligent System: Responsible Technology for Sustainable Humanity, ICE3IS 2023*,

August, 349–353. https://doi.org/10.1109/ICE3IS59323.2023.10335452

Qadri, S. A. A., Huang, N.-F., Wani, T. M., & Bhat, S. A. (2025). Advances and Challenges in Computer Vision for Image-Based Plant Disease Detection: A Comprehensive Survey of Machine and Deep Learning Approaches. *IEEE Transactions on Automation Science and Engineering*, *22*, 2639–2670. https://doi.org/10.1109/TASE.2024.3382731

Qin, Y. M., Tu, Y. H., Li, T., Ni, Y., Wang, R. F., & Wang, H. (2025). Deep Learning for Sustainable Agriculture: A Systematic Review on Applications in Lettuce Cultivation. *Sustainability (Switzerland)*, *17*(7). https://doi.org/10.3390/su17073190

Raghavan, K., B, S., & V, K. (2023). Attention guided grad-CAM : an improved explainable artificial intelligence model for infrared breast cancer detection. *Multimedia Tools and Applications*, *83*(19), 57551–57578. https://doi.org/10.1007/s11042-023-17776-7

Rathor, A. S., Choudhury, S., Sharma, A., Nautiyal, P., & Shah, G. (2025). A Novel Deep Convolutional Neural Network for Efficient Classification of Lettuce Diseases. *Procedia Computer Science*, *258*, 755–764. https://doi.org/10.1016/j.procs.2025.04.308

Rathor, A. S., Choudhury, S., Sharma, A., Shah, G., & Nautiyal, P. (2025). A mathematical modelling-based interpretable deep learning approach for lettuce disease detection in extreme environmental conditions. *Physics and Chemistry of the Earth*, *141*(August). https://doi.org/10.1016/j.pce.2025.104080

Upadhyay, A., Chandel, N. S., Singh, K. P., Chakraborty, S. K., Nandede, B. M., Kumar, M., Subeesh, A., Upendar, K., Salem, A., & Elbeltagi, A. (2025). Deep learning and computer vision in plant disease detection: a comprehensive review of techniques, models, and trends in precision agriculture. *Artificial Intelligence Review*, *58*(3), 92. https://doi.org/10.1007/s10462-024-11100-x

Wang, Y., Wu, M., & Shen, Y. (2024). Identifying the Growth Status of Hydroponic Lettuce Based on YOLO-EfficientNet. *Plants*, *13*(3), 372. https://doi.org/10.3390/plants13030372

Yang, R., Wu, Z., Fang, W., Zhang, H., Wang, W., Fu, L., Majeed, Y., Li, R., & Cui, Y. (2023). Detection of abnormal hydroponic lettuce leaves based on image processing and machine learning. *Information Processing in Agriculture*, *10*(1), 1–10. https://doi.org/10.1016/j.inpa.2021.11.001

Zhang, P., & Li, D. (2022). YOLO-VOLO-LS: A Novel Method for Variety Identification of Early Lettuce Seedlings. *Frontiers in Plant Science*, *13*. https://doi.org/10.3389/fpls.2022.806878

Chapter 2

Prediction of Manufacturing Defects With Machine Learning-Based Classification Models: Application of Logistic Regression, Random Forest and Xgboost

Murat BİNİCİ[1]

ABSTRACT

This chapter investigates the prediction of manufacturing defects using machine learning–based classification models on a multivariate, synthetic dataset representing daily operational performance in a smart manufacturing context. The dataset comprises 3,240 observations and 17 numerical variables, including indicators related to production volume and cost, supplier quality, delivery delays, maintenance and downtime, inventory performance, labor productivity, energy consumption, additive processes, and a binary target variable (DefectStatus) indicating high- versus low-defect production days. After a structured preprocessing phase involving missing data checks, feature scaling where appropriate, and a two-level strategy for handling severe class imbalance (SMOTE-based oversampling and class weighting), three models—Logistic Regression (LR), Random Forest (RF), and XGBoost—are trained and evaluated. Model performance is assessed on a stratified train–test split using accuracy, precision, recall, F1-score, ROC–AUC, confusion matrices, and feature importance analyses. The results show that tree-based ensemble models outperform LR, with RF achieving the highest accuracy (0.94) and recall for the high-defect class, whereas XGBoost yields the best ROC–AUC, indicating superior discriminative power. Feature importance rankings consistently highlight maintenance-related indicators, defect rate, quality score, and production volume as key drivers of defect risk. The chapter concludes that ML-based classification, particularly with ensemble methods, provides an effective decision-support framework for early defect detection and quality improvement in manufacturing systems.

Keywords: Machine learning–based classification, Manufacturing defects, Smart manufacturing, Logistic Regression, Random Forest, XGBoost

[1] Assist. Prof. Dr., Bitlis Eren University, Faculty of Engineering and Architecture, Department of Mechanical Engineering, mbinici@beu.edu.tr, ORCID: 0000-0003-1814-438X.

1. INTRODUCTION

In the contemporary manufacturing sector, where global competition is intensifying, product quality and process reliability are of strategic importance for the sustainability of enterprises. Defects occurring in production systems result in a wide range of direct and indirect costs, including rework, scrap, delivery delays, warranty expenses, and customer dissatisfaction. Accordingly, rather than detecting defects only at the end of the production line, it has become increasingly critical to predict and prevent them at earlier stages. In this regard, artificial intelligence and machine learning approaches that rely on the effective analysis of multidimensional data collected from manufacturing processes are emerging as more flexible, scalable, and powerful predictive tools than classical statistical methods (Tercan and Meisen, 2022).

With the advent of Industry 4.0 and the smart manufacturing paradigm, vast amounts of data are being generated from production lines through sensors, the Internet of Things (IoT), and cyber-physical systems. These datasets simultaneously encompass multiple dimensions such as production volume, supplier quality, maintenance activities, energy consumption, inventory movements, labor productivity, and quality control results. Within this complex data structure, the relationships between process parameters and quality outcomes are not expected to be linear, stable, or simple. The literature, particularly in the domain of smart manufacturing, shows that Machine Learning (ML)–based quality prediction models are widely employed to handle such high-dimensional and complex datasets (Deokar et al., 2025).

Recent systematic reviews have shown that the range of ML algorithms used for quality assurance and defect prediction in manufacturing has expanded; however, tree-based ensemble methods and logistic regression stand out in a substantial portion of applications. In particular, tree-based approaches such as Random Forest (RF) and Extreme Gradient Boosting (XGBoost) are reported to be widely preferred in manufacturing quality and defect classification problems, owing to their ability to capture complex, nonlinear relationships, handle heterogeneous types of variables (continuous, integer, ratio, etc.), and reveal variable importance scores (Kausik et al., 2025). Logistic Regression (LR), on the other hand, is commonly employed as a comparison (baseline) model in many studies due to the interpretability of its coefficients and its relatively low computational cost (Tercan and Meisen, 2022).

However, the class imbalance problem, which is frequently encountered in production and maintenance data, emerges as one of the most significant methodological challenges in defect prediction studies. In real manufacturing environments, defective products typically occur as “rare events” whose

proportion within total production is relatively low. This situation leads to a pronounced imbalance between majority and minority classes in the dataset, increasing the risk that conventional classification algorithms will be biased toward the majority class and overlook defective instances in the minority class, which are often of primary interest. Recent systematic reviews focusing on the manufacturing domain indicate that numerous approaches have been proposed to address class imbalance at the data level (resampling, synthetic data generation, etc.) and at the algorithmic level (class weighting, cost-sensitive learning, etc.) (de Giorgio et al., 2023).

In this book chapter, a ML–based classification framework is proposed to predict whether the level of defects occurring on the production line will be "high" or "low," using a multivariate manufacturing dataset that reflects daily production performance. The dataset employed in the study, the Predicting Manufacturing Defects Dataset, includes indicators covering a wide range of processes, such as production volume and cost, supplier quality, delivery delays, maintenance durations, downtime ratio, inventory indicators, labor productivity, occupational safety incidents, energy consumption, and additive production, and thus offers a holistic view of manufacturing operations (El Kharoua, 2024). The target variable, DefectStatus, represents in binary form whether the production output for a given day is highly defective (1) or has a low level of defects (0).

This study examines three different classification models: LR, RF, and XGBoost. LR, as a probability-based and interpretable model that relies on the assumption of linear separability, makes it possible to investigate the direction and magnitude of the effects of production parameters on defect probability. RF, an ensemble method obtained by training a large number of decision trees on random subsamples of observations and subsets of features, is able to capture complex interactions and nonlinear relationships among variables. XGBoost, in turn, is an optimized representative of the gradient-boosted decision tree family and has come to the forefront in industrial applications in recent years due to both its predictive performance and its sensitivity to hyperparameter tuning (Chen et al., 2024).

The main objective of this chapter is to present a comprehensive approach to predicting defect risk in production lines by comparatively evaluating AI–based decision tree models and the logistic regression method on the aforementioned dataset. Within this framework, the study aims to (i) analyze the relationships between production, procurement, maintenance, inventory, energy, and labor indicators and defect status; (ii) examine the impact of the class imbalance problem on model performance; (iii) compare different classification algorithms

not only in terms of the accuracy measure, but also using metrics such as F1-score, recall, specificity, and ROC–AUC; and (iv) investigate variable importance levels, thereby developing an early warning and quality prediction framework that can support decision-makers in manufacturing processes.

In this regard, the study aims to make a twofold contribution from both theoretical and practical perspectives. On the theoretical plane, it introduces a classification framework that follows current approaches in the manufacturing quality prediction literature and is sensitive to class imbalance and model evaluation metrics. On the practical plane, it contributes to the development of data-driven decision support systems by proposing a modeling approach that, drawing on operational indicators commonly recorded in manufacturing environments, anticipates defect risk and points to potential areas for improvement.

2. LITERATURE REVIEW

The use of ML technologies in quality assurance and defect control in the manufacturing sector has become increasingly critical as the volume and variety of data grow. Existing systematic studies show that process parameters and quality outcomes derived from production data are analyzed using ML models, thereby achieving higher prediction accuracy and greater process flexibility compared to traditional methods (Kausik et al., 2025). In particular, in studies that make use of sensor data, IoT systems, and large-scale datasets, ML methods emerge as effective tools for in-process quality control and early warning systems (Ördek et al., 2024). However, data preparation workflows, model interpretability, and integration costs are among the challenges encountered in this field (Antosz et al., 2024).

Tree-based models and ensemble methods are widely preferred for quality prediction in manufacturing processes. In this context, the RF algorithm can effectively capture nonlinear relationships and interactions among variables by training a large number of tree structures on randomly sampled subsets of observations and features. High performance of RF has been reported in areas such as additive production, automotive components, and electronics manufacturing lines (Kausik et al., 2025). Another advantage of these methods is that, through feature importance measures, they provide an opportunity to interpret from an engineering standpoint which production parameters have a greater impact on quality. On the other hand, if the parameter settings (e.g., number of trees, depth) are not properly tuned, limitations such as the risk of overfitting and a tendency to favor the majority class in datasets with class imbalance may arise.

LR is one of the fundamental methods that has been used for many years in classification problems and is also preferred in the context of manufacturing quality due to the interpretability of its results (Tercan and Meisen, 2022; Md et al., 2022). This method makes it possible to directly assess, through the logistic function, the effect of a given production variable on defect probability (Borucka and Grzelak, 2019). In the literature, LR is typically employed as an initial or baseline model and subsequently compared, in terms of performance, with more complex models (Tercan and Meisen, 2022). However, the linear separability assumption of LR can be a limitation in capturing nonlinear interactions among variables; therefore, its performance may remain relatively lower in multivariate and nonlinear manufacturing processes (Md et al., 2022).

XGBoost is a tree-based ensemble algorithm that has become particularly prominent in industrial data analytics and quality prediction studies. In a study conducted for failure prediction on a production line using XGBoost, high predictive accuracy was achieved (Mehregan et al., 2025). The advantages of this method include the possibility of hyperparameter optimization, its compatibility with large datasets, and its adaptability to irregular class distributions through settings such as class weights. However, to ensure strong performance, its hyperparameters must be selected carefully and overfitting must be avoided.

One frequently encountered issue in manufacturing quality data is that the number of defective products is relatively low compared to total production, which leads to class imbalance in the dataset. In the literature, two main approaches are highlighted to address this problem: data-level resampling (oversampling, undersampling, SMOTE) and algorithm-level strategies such as assigning class weights or employing cost-sensitive learning (He and Garcia, 2009). Moreover, since using only accuracy as a performance measure can be misleading in imbalanced settings, it is recommended to adopt more informative metrics such as F1-score, precision–recall, and ROC–AUC (Ogrizović et al., 2024). In this context, addressing class imbalance in the modeling phase of the present study is in line with good practice recommendations in the literature.

In summary, the literature indicates that ML methods are widely employed for quality prediction and early fault detection in manufacturing processes, and that tree-based methods and XGBoost in particular have demonstrated strong effectiveness. However, studies that use daily production metrics with multivariate inputs and class-imbalanced datasets to comparatively evaluate LR, RF, and XGBoost within a unified framework remain limited. Therefore, assessing these models on the same dataset, deriving variable importance levels,

and explicitly accounting for class imbalance has the potential to provide an original contribution.

3. DATASET DESCRIPTION

A correct understanding of the structure of the dataset is critically important for the subsequent modeling process, performance evaluation, and variable importance analysis. In addition, factors such as the relationship of the variables to the underlying production processes, the class distribution, and the nature of the imbalance play a decisive role in the implementation of ML models. For this reason, the general structure and key characteristics of the dataset, in this section, are first described, and then each variable is examined in detail.

3.1. Data Source and Type

The dataset used in this study is a comprehensive synthetic data set that reflects daily operational performance, quality indicators, and supply chain conditions in manufacturing processes. It was specifically constructed for the purpose of developing an ML-based model for the classification of production line defects (El Kharoua, 2024). Accordingly, it was designed by taking into account the variable structures, inter-variable relationships, and defect formation dynamics observed in real manufacturing environments, while being simulated in such a way that it does not contain any sensitive information belonging to a commercial organization or an actual production facility. This property makes the dataset both safe in terms of ethical use and flexible for academic research.

The dataset consists of a total of 3,240 observations and 17 variables. Each row in the dataset represents the operational performance for a single production day. Variables such as production volume, cost, energy consumption, maintenance activities, labor productivity, supply chain performance, and quality control measures are summarized and recorded on a daily basis. Since the dataset does not contain any information on product categorization or product variety, the analysis is conducted under the assumption of a homogeneous production line manufacturing a single product type. This approach allows the defect prediction performance of the model to be examined solely on the basis of process-specific metrics.

The synthetic nature of the dataset provides several methodological advantages for the study. First, it allows defect cases that are rare in real manufacturing environments but critical from a modeling perspective to be incorporated into the data in a more balanced manner. Second, it can be used in open-access research without raising data confidentiality concerns and is

suitable for educational and training applications. However, the main limitation of synthetic datasets is that the relationships among variables may not fully reflect the complexity observed in real production environments. Therefore, while this dataset is well suited for model development, method comparison, and academic teaching purposes, caution is required when transferring the model outputs directly to the operational strategies of an actual factory.

In conclusion, by offering a wide range of metrics related to manufacturing processes and capturing day-to-day operational behavior, the dataset provides an appropriate and methodologically coherent basis for this study, which focuses on classifying production line defects using ML methods.

3.2. Variables

In the dataset used in this study, the variables are grouped under thematic categories in order to better reflect the multifaceted nature of manufacturing processes. First, the production metrics category covers daily production output and cost components. In this context, the variable *ProductionVolume* represents the number of units produced per day, while *ProductionCost* denotes the total cost of the corresponding production activity. Together, these two indicators make it possible to analyze how production intensity and cost pressure influence quality.

Supply chain and logistics indicators also occupy an important place in the dataset. *SupplierQuality* reflects the quality of inputs provided by suppliers using a percentage-based score, whereas *DeliveryDelay* indicates the duration of delays in supply processes. Considering the impact of supplier quality and logistical disruptions on the reliability of production outputs, these variables are critical for the contribution they make to the model.

The quality control category includes two key variables that relate directly to the quality performance observed at the end of the production process. *DefectRate* quantitatively represents the number of defects per thousand units, while *QualityScore* expresses the overall quality level of production as a percentage score. These variables both summarize the quality outcome of the process and can be regarded as important independent variables for defect prediction models.

Variables related to maintenance and downtime include *MaintenanceHours* and *DowntimePercentage*. *MaintenanceHours*, which indicates the weekly duration of maintenance activities, and *DowntimePercentage*, which reflects the proportion of time the production line is not operational, provide important operational indicators of equipment efficiency and continuity. Since increases in

these values are typically associated with declines in quality performance, they make meaningful contributions to the model.

Among the variables related to inventory management are *InventoryTurnover* and *StockoutRate*. *InventoryTurnover*, which represents stock turnover, indicates the efficiency of the firm's inventory management, while *StockoutRate* reflects the risk of production interruptions through the rate of stock depletion. These two indicators are important for examining how disruptions in the flow of raw materials may indirectly affect quality performance.

Variables related to labor productivity and safety are also included in the dataset. *WorkerProductivity* expresses workers' productivity levels in percentage terms, whereas *SafetyIncidents* indicates the number of safety incidents that occur within a given month. Considering that worker motivation, safety, and productivity are closely associated with quality outcomes, the inclusion of these indicators in the model is important.

Variables representing energy consumption and energy efficiency include *EnergyConsumption* and *EnergyEfficiency*. *EnergyConsumption*, which expresses daily energy use in kilowatt-hours, and *EnergyEfficiency*, which indicates the level of efficiency in energy utilization, are incorporated into the model based on the assumption that overall line efficiency and fluctuations in machine performance may affect quality.

Finally, the variables related to one of the modern manufacturing technologies, namely additive manufacturing processes, *AdditiveProcessTime* and *AdditiveMaterialCost*, represent, respectively, the duration of the additive manufacturing process and the unit cost of the additive material used. These parameters are important for assessing how innovative production techniques influence defect formation.

Beyond all these categories, the main target variable of the study, DefectStatus, enables the classification of the production output as low-defect (0) or high-defect (1). This variable constitutes the primary outcome of the modeling process in relation to all other indicators in the dataset.

The dataset used in this study consists of a total of 3,240 observations and 17 variables. This size is sufficient both to provide an appropriate sample for training machine learning models and to allow for an analytical examination of the multidimensional structure of manufacturing processes.

All variables in the dataset are numerical, and there are no categorical variables. A subset of the variables are of integer type, namely *ProductionVolume*, *DeliveryDelay*, *MaintenanceHours*, *SafetyIncidents*, and the target variable in the classification process, *DefectStatus*. All remaining

variables are continuous numerical (float) in nature and cover a wide range of measurements related to the production process, such as production cost, quality indicators, inventory performance, energy usage, and additive manufacturing times. Specifically, *ProductionCost*, *SupplierQuality*, *DefectRate*, *QualityScore*, *DowntimePercentage*, *InventoryTurnover*, *StockoutRate*, *WorkerProductivity*, *EnergyConsumption*, *EnergyEfficiency*, *AdditiveProcessTime*, and *AdditiveMaterialCost* fall into this group.

The fact that the entire dataset consists of quantitative variables allows it to be analyzed directly by both LR and tree-based classification models, and it also substantially simplifies the preprocessing stage, as no encoding procedures are required. Researchers who wish to access detailed descriptive statistics for all variables in the dataset can obtain this information via the Kaggle platform (Kaggle Dataset: Predicting Manufacturing Defects Dataset).

3.3. Class Imbalance

In the dataset, the class distribution of the target variable *DefectStatus* is observed to be highly imbalanced; the high-defect class accounts for approximately 84% of all instances, whereas the low-defect class constitutes only about 16%. Such a distribution is referred to in the literature as class imbalance and represents a fundamental issue that directly affects the performance of ML-based classification models. Since the number of defective products is typically low in real manufacturing data, it is likely that the model will develop a bias in favor of the majority class and fail to adequately learn the minority class (de Giorgio et al., 2023).

Class imbalance plays a critical role particularly in applications such as production quality control and defect prediction. Recent reviews of application domains show that imbalanced learning is still a major obstacle in “real-world” data and is explicitly addressed in studies on production line failure or defect detection (Gao et al., 2025). This situation indicates that evaluating models solely on the basis of the accuracy metric can be misleading; therefore, it is recommended to use performance measures such as F1-score, precision–recall curves, and ROC–AUC (Gao et al., 2025).

To address class imbalance, the literature highlights two main approaches: data-level resampling techniques (oversampling, undersampling, SMOTE, etc.) and algorithm-level strategies such as assigning class weights or adopting cost-sensitive learning. In this regard, Chen et al. (2024) note that, in addition to data-level and algorithm-level solutions, hybrid methods have also become increasingly widespread. In the manufacturing context, Giorgio et al. (2023) show that, in fault/defect detection problems with imbalanced data where the

error/defect class is rare, resampling procedures and class weight adjustments are commonly employed.

In this study as well, appropriately addressing class imbalance prior to the modeling stage will not only improve the performance of the model but also enhance the reliability of the inferences drawn for quality control applications. In this way, it will become possible to predict in advance the days with a high defect risk on the production line and to establish a more robust foundation for decision-support systems.

4. METHODOLOGY

This section presents the methodological framework of the ML approach applied to classify production line defects. The methodology of the study covers the data preprocessing steps carried out to make the dataset suitable for analysis, the theoretical foundations of the LR, RF, and XGBoost models used in the classification process, and finally the criteria selected to evaluate model performance. Considering the multidimensional and imbalanced nature of production data, planning the methodological procedure in a systematic and coherent manner is of great importance both for the reliability of the results obtained from the models and for the validity of the implications for industrial applications. For this reason, the methodology section explains in detail both the data processing procedures and the analytical logic of the selected algorithms.

4.1. Pre-Processing

The ability of ML models to produce reliable and generalizable results depends on subjecting the dataset to appropriate preprocessing prior to analysis. Since multidimensional data structures derived from manufacturing processes may contain issues such as differences in scale, unequal distributions across variables, and class imbalance, a systematic preprocessing procedure was applied before modeling. This section discusses key steps such as checking for missing data, scaling, addressing class imbalance, and splitting the dataset into training and test sets.

4.1.1. Missing data analysis

The first step in the preprocessing procedure is to check the dataset for missing or erroneous records. Missing data can reduce the learning capacity of the model and may directly introduce bias, particularly in statistical methods such as LR. Although the dataset used in this study contains no missing values because it was generated synthetically, missingness is quite common in real

manufacturing data. For this reason, missing data analysis is a critical step for preserving methodological integrity.

4.1.2. Feature scaling

Differences in the scales on which the variables in the dataset are measured can adversely affect coefficient estimates and convergence behavior, particularly in LR and other gradient-based methods. For this reason, variables with wide ranges, such as production volume (100–1000), cost (5,000–20,000), and energy consumption (1,000–5,000 kWh), may take much larger values than percentage- or ratio-based metrics, potentially destabilizing weight updates in the models. In this study, standard scaling (StandardScaler) was applied to ensure stable behavior of LR and to allow for consistent comparison across models. For the tree-based methods, no scaling was applied, as they are more flexible and less sensitive to differences in feature scales.

4.1.3. Addressing class imbalance

Given the substantial class imbalance in the dataset (approximately 84% high-defect vs. 16% low-defect), it is necessary to apply methods that increase the model's sensitivity to the minority class. Imbalanced data structures tend to induce a bias toward the majority class in ML models and reduce the classification performance for the minority class (Chen et al., 2024; de Giorgio et al., 2023). Therefore, two complementary approaches were adopted in this study:

(a) ***Data-level approach*:** By applying SMOTE (Synthetic Minority Over-sampling Technique), synthetic samples were added to the minority class and the class distribution was balanced. This method aims to enable the model to learn the low-defect cases, which are rarely observed in manufacturing processes, more effectively.

(b) ***Algorithm-level approach***: In the LR, RF and XGBoost models, the class_weight parameter was set to "balanced", thereby forcing the model to assign greater weight to the misclassification cost of the minority class.

This two-stage strategy can enhance the model's sensitivity in predicting production line defects.

4.1.4. Splitting the dataset into training and test sets

The dataset was split into 80% training and 20% test in order to evaluate the generalization capacity of the models. The training set represents the stage in which the model parameters are learned, while the test set is used to objectively assess model performance on previously unseen data. Taking class imbalance

into account, a stratified split technique was applied so that the class proportions were preserved in both the training and test subsets. This approach eliminates the risk that the minority class might be entirely absent from either the training or the test set.

4.2. Applied Machine Learning Models

In this study, three different ML-based classification models were employed for classifying production line defects: LR, RF, and XGBoost. This section briefly explains the basic assumptions, working principles, and the specific role of each model within the context of the present study.

4.2.1. Logistic regression

LR is a statistically grounded method that has long been used to solve binary classification problems and offers a high degree of interpretability. The model expresses the relationship between the independent variables and the target variable in probabilistic terms through the logistic (sigmoid) function, and this feature allows it to provide directly interpretable outputs for decision-makers in risk-oriented processes such as production defect prediction (Hosmer et al., 2013). In addition, the coefficient-based structure of LR makes it possible to directly infer from the model the direction and magnitude of the effects of variables such as production volume, quality scores, and supplier quality on defect probability.

One of the main advantages of LR is that its model parameters are directly interpretable and that the effect of each variable on the target can be assessed in terms of log-odds. This property contributes to the frequent use of LR as a baseline model in decision-oriented domains such as quality engineering and production analytics. Indeed, the literature commonly reports LR as a reference model both for evaluating classification performance and for benchmarking against more complex models (Kovács et al., 2024).

In this study, the LR model was applied to classify defects occurring in the production process (0 = low defect, 1 = high defect). The model was implemented using a *Pipeline* structure that incorporates the data preprocessing steps. The training of the model consists of the following steps:

(a) **Data splitting**: After separating the target variable DefectStatus, the dataset was split into training and test sets in an 80–20 ratio, and a stratified split procedure was employed to preserve the class distribution.

(b) **Scaling**: Since the LR model can be affected by the scale of the features, all independent variables were standardized using *StandardScaler*.

(c) **Handling class imbalance:** Since the high-defect class is dominant in the dataset, two strategies were applied. The first is a model-level adjustment using *class_weight = "balanced"*, and the second is a data-level procedure using *SMOTE*. Both approaches were integrated into the *Pipeline*.

(d) **Model specification:** The model was defined with the following parameters: *class_weight="balanced"*, *max_iter=1000*, *solver="liblinear"*, *random_state=42*.

(e) **Pipeline structure:** The Pipeline was composed of three components: *StandardScaler*, *SMOTE*, and LR. This configuration ensured a clean workflow and prevented data leakage.

(f) **Training the model:** The *Pipeline* was fitted on the training data to construct the model. Subsequently, predictions were generated on the test data; however, this section reports only the model implementation procedure, while the performance evaluation is presented in the following sections.

4.2.2. Random forest classifier

RF is an ensemble ML method based on decision trees. The core idea is to build a large number of decision trees on different subsamples of the training data and then aggregate their predicted classes using majority voting for the classification task. In this way, the overfitting problem to which a single deep tree is prone is substantially reduced, and the model's generalization performance is improved (Breiman, 2001). RF reduces correlation among trees by using both random sampling of observations (bootstrap sampling) and random subsets of features at each node; thus, instead of individual trees with high variance, a more balanced and stable ensemble model is obtained (Breiman, 2001).

One of the main reasons why RF models are widely used in manufacturing and quality control is their success in capturing nonlinear relationships and interactions among variables. Recent studies show that RF outperforms traditional statistical methods in terms of predictive performance, owing to its ability to process high-dimensional sensor data and to achieve high accuracy in predicting quality outcomes in complex production processes (Kausik et al., 2025; Antosz, 2024). In addition, despite its relatively "black-box" nature, RF offers a practically useful level of transparency for decision-support systems by providing feature importance scores that indicate which inputs contribute more strongly to the model output (Scornet, 2021).

In this study, the RF classifier was applied to classify the target variable DefectStatus (0 = low defect, 1 = high defect) using the large set of operational variables measured on a daily basis in the production process. As in the LR

model, the target variable was first separated from the dataset, and all remaining variables were defined as the feature set (X). To preserve the class distribution, the dataset was then split into training and test subsets in an 80–20 ratio using a stratified structure. In this way, the imbalanced class structure was represented in a similar manner in both the training and test sets, and a fair basis for comparison across models was established.

Since RF is a tree-based method, it is not sensitive to feature scales unlike LR; therefore, no additional scaling step was applied for this model. However, the class imbalance present in the dataset (with the high-defect class being dominant) was taken into account, and a two-level strategy was adopted to mitigate this issue. First, *SMOTE* was applied to the training data to synthetically increase the number of minority-class observations, thereby enabling RF to learn the underrepresented class more effectively. Second, the classifier's *class_weight* parameter was set to *"balanced"*, ensuring that the loss function assigns greater weight to the minority class. The combined use of these two approaches is consistent with good practice recommendations in the literature for imbalanced datasets (Khan et al., 2024).

The RF model was implemented in Python using the *RandomForestClassifier* class from the scikit-learn library. To enhance model stability, the *n_estimators* parameter was set to a relatively high value (300 trees), while the *random_state* parameter was fixed to ensure reproducibility of the results. In addition, by setting *n_jobs = -1*, the training process was executed in parallel, taking advantage of multi-core processor architectures. As in the LR model, the specification and training of the RF model were defined within a *Pipeline*, thereby ensuring that *SMOTE* was applied only to the training data and preventing data leakage.

After the training procedure was completed, the RF model generated predictions on the test set, and performance metrics such as *accuracy*, *precision*, *recall*, *F1-score*, and *ROC–AUC* were computed based on these predictions. However, this subsection presents only the theoretical framework and implementation steps of the RF model; the resulting performance scores are discussed in Section 5, Results and Discussion, where they are comparatively evaluated together with the other models (LR and XGBoost) in order to preserve the overall coherence of the study.

4.2.3. XGBoost classifier

XGBoost is an optimized, high-performance implementation of a tree-based gradient boosting algorithm. Its core working principle is to build weak learners sequentially in such a way that they minimize the residual error, focusing on the

parts of the data that previous models failed to explain. Thanks to hyperparameters such as learning rate, tree depth, subsampling, and strong regularization mechanisms, the model both keeps overfitting under control and delivers high predictive performance on large and complex datasets (Chen and Guestrin, 2016).

In recent years, XGBoost has come to be regarded as one of the leading gradient boosting methods for critical applications such as quality classification, fault detection, and anomaly detection in manufacturing processes. Various studies report that XGBoost models optimized for anomaly detection on production lines achieve high accuracy, precision, and F1-scores, and that they outperform traditional methods as well as some other ensemble approaches (Dalal et al., 2024; Nilsson and Kyrk, 2025). Similarly, in complex manufacturing environments such as printed circuit board production, XGBoost-based models have been shown to be effective in defect detection using high-dimensional data generated by the production process (Prasad-Rao et al., 2023). Owing to its capacity to handle large datasets, its embedded regularization mechanisms, and its success in capturing nonlinear relationships, XGBoost has become a frequently preferred algorithm for data-driven quality control and decision-support systems within the scope of Industry 4.0 (Kausik et al., 2025; Qu et al., 2024).

In this study, the XGBoost classifier was used to classify defects occurring in manufacturing processes (0 = low defect, 1 = high defect). The implementation steps were designed to remain consistent with the previous models. First, the target variable was separated from the dataset, and all independent variables were defined as the feature set. To allow for a fair evaluation of the model, the dataset was split into 80% training and 20% test using a stratified procedure.

Since XGBoost is a tree-based algorithm, it does not require additional scaling (standardization). However, the pronounced *class imbalance* in the dataset was taken into account, and *SMOTE* was applied during training to increase the representation power of the minority class. To ensure that *SMOTE* was applied only to the training data and to prevent data leakage, the model was defined within a *Pipeline*, as in the RF setup.

In this study, the XGBoost classifier was configured with specific hyperparameter settings to ensure a balanced learning process and to keep overfitting under control in the classification task. The parameter *n_estimators=300* was chosen to obtain a more stable and well-trained ensemble of trees, while *max_depth=4* was used to limit tree depth and thereby prevent overfitting. To ensure a more gradual learning process,

learning_rate=0.1 was adopted so that the contribution of each individual tree to the model was reduced in a controlled manner. In addition, the hyperparameter *subsample=0.8*, which randomly samples a portion of the dataset, was employed to enhance the generalization ability of the model, whereas *colsample_bytree=0.8*, which creates a feature subset for each tree, introduced further diversity and supported model performance. This combination of parameters provides an effective configuration for XGBoost, balancing its predictive accuracy with its capacity to control overfitting.

4.3. Model Evaluation Metrics

In ML-based classification problems, accurately evaluating model performance is critically important, especially in datasets that exhibit class imbalance. For this reason, assessing a model on the basis of a single metric is often inadequate. Below, the main classification metrics used in this study are described.

4.3.1. Accuracy

Accuracy represents the proportion of correctly classified instances to the total number of instances. However, this metric can be misleading when there is a severe imbalance between classes. For example, in a dataset where the positive class is very rare, a model that predicts all instances as "negative" may still achieve a high accuracy score (He and Garcia, 2009).

4.3.2. Precision ve recall

Precision indicates how many of the instances that the model predicts as positive are actually positive, whereas recall shows how many of the truly positive instances are correctly identified by the model. In domains such as production defects, where errors can lead to costly consequences, these two metrics are particularly critical. This is because false positives (unnecessary intervention costs) and false negatives (missed actual defects) directly affect decision-making processes (Saito and Rehmsmeier, 2015).

4.3.3. F1-score

The F1-score is the harmonic mean of precision and recall, and it enables these two metrics to be optimized jointly. In situations with class imbalance, the F1-score is a much more informative evaluation measure than accuracy alone, because it balances the impact of both false positives and false negatives (Chicco and Jurman, 2020).

4.3.4. ROC-AUC

The ROC (Receiver Operating Characteristic) curve illustrates the model's ability to distinguish the positive class across different threshold values. The AUC (Area Under the Curve) represents the area under this ROC curve, and as it approaches 1, the model is considered to have better discriminative power (Fawcett, 2006).

4.3.5. Confusion matrix

The confusion matrix provides a detailed breakdown of the model's correct and incorrect classifications in terms of four components (TP, FP, FN, TN). This matrix is extremely important for understanding the impact of false negatives (missing actual defects) and false positives (unnecessary intervention) on manufacturing processes (Kelleher et al., 2015).

5. RESULTS AND DISCUSSION

In this section, the performance results of the three classification models used in the study—LR, RF, and XGBoost—are examined in a comparative manner. All models were evaluated using the same train–test splitting strategy, and *SMOTE* was applied to address class imbalance. Model performance was assessed not only in terms of accuracy, but also using more comprehensive metrics such as *precision*, *recall*, *F1-score*, and *ROC–AUC*, thereby enabling a more robust analysis of the ability of the different classification models to distinguish between high- and low-defect conditions.

5.1. Performance Comparison

The test-set performances of the LR, RF, and XGBoost models are compared in detail. The main evaluation metrics used for the three models are summarized in Table 1, and classification performance is assessed not only in terms of accuracy, but also using precision, recall, F1-score, and ROC–AUC. As can be seen from the table, LR, due to its structure based on linear relationships, lags behind the other models and shows more limited success, particularly in distinguishing the high-defect class. By contrast, the RF model achieves the highest accuracy (0.94) and the highest recall value, demonstrating a notably strong performance in detecting the high-defect class. XGBoost, on the other hand, attains the highest ROC–AUC value, making it the model that best separates the classes. In this regard, although all the three models exhibit different strengths, the tree-based methods appear to be more successful, especially when dealing with manufacturing data characterized by complex and nonlinear relationships.

Table 1. Model performance summary

Model	Accuracy	Precision (Class 1)	Recall (Class 1)	F1-Score (Class 1)	ROC-AUC
Logistic Regression	0.7546	0.93	0.76	0.84	**0.79**
Random Forest	0.9414	0.95	0.98	0.96	**0.83**
XGBoost	0.9151	0.95	0.95	0.95	**0.84**

The confusion matrix results for the models are presented in Table 2. An examination of the confusion matrices shows that LR produces a high number of false negatives (FN = 130), indicating that the model frequently classifies high-defect products as "low defect." In a critical domain such as production defect detection, a high false negative rate is a highly undesirable outcome. By contrast, the RF model yields only nine false negatives and, in this respect, is the method that captures the high-defect class best among the three models. Although the number of false negatives produced by XGBoost (FN = 26) is higher than that of RF, it still demonstrates a much better classification performance than LR. The fact that all three models share the same number of true negatives (TN = 74) and false positives (FP = 29) suggests that, after SMOTE, class rebalancing and model complexity primarily affect the high-defect class.

Table 2. Confusion matrix results

Model	TN	FP	FN	TP
Logistic Regression	74	29	130	415
Random Forest	74	29	9	536
XGBoost	74	29	26	519

Evaluating model performance in terms of ROC curves provides a more detailed understanding of the discriminative power between classes. In the ROC plots (Figure 1), the curves of RF and XGBoost lie above that of LR, indicating that the tree-based methods offer a more consistent and stronger separation capability across different threshold values. In particular, the ROC curve of XGBoost attains higher true positive rates at both low and high false positive rates, revealing that the model has strong generalization performance. Although the ROC–AUC value of RF is slightly lower than that of XGBoost, it still constitutes a robust alternative from an operational risk management perspective due to its high success in capturing the positive (high-defect) class. The ROC curve of LR, by contrast, remains lower, indicating that it cannot

deliver optimal performance for manufacturing data characterized by nonlinear relationships.

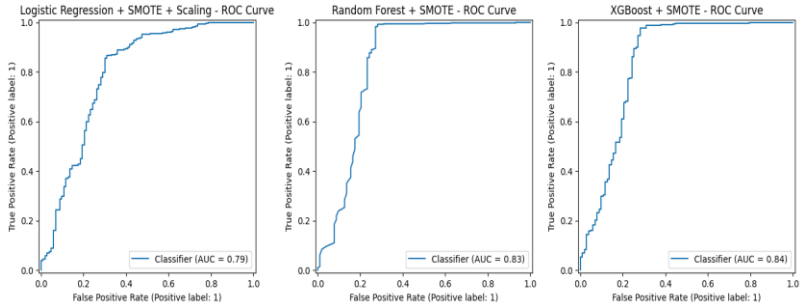


Figure 1. Comparison of ROC curves for LR, RF, and XGBoost models

Overall, when Table 1, Table 2, and the ROC curves (Figure 1) are evaluated jointly, it is evident that tree-based methods perform markedly better in the production defect classification problem. RF largely prevents high defects from being missed by minimizing false negatives, whereas XGBoost provides a higher discriminative capacity and a more balanced overall performance. Although LR has the advantage of interpretability, it lags behind the other models in terms of predictive performance. These results indicate that ensemble methods are more suitable for analyzing complex data structures in manufacturing processes.

5.2. Feature Importance Ranking

When the comparative feature importance values obtained from the three models (LR, RF, XGBoost) are examined, it is observed that certain variables systematically stand out across all models in determining production defects. According to the findings presented in Table 3, *MaintenanceHours* and *DefectRate* are the two key determinants with the highest importance scores in all three models. This result indicates that maintenance activities and existing defect levels play a central role in predicting the emergence of new defects in production.

Table 3. Comparative feature importance of LR, RF, and XGB models

Feature	LR Coefficient	LR Abs. Importance	RF Importance	XGB Importance
MaintenanceHours	1.238882	1.238882	0.257275	0.215104
DefectRate	1.037257	1.037257	0.199312	0.178374
QualityScore	-0.652453	0.652453	0.132414	0.121911
ProductionVolume	0.456485	0.456485	0.105837	0.105118
SupplierQuality	0.139299	0.139299	0.024049	0.027003
StockoutRate	0.107314	0.107314	0.027081	0.030488
DeliveryDelay	0.094509	0.094509	0.023217	0.056954
EnergyConsumption	0.079072	0.079072	0.031444	0.027456

Table 3. Comparative feature importance of LR, RF, and XGB models (*Cont.*)

Feature	LR Coefficient	LR Abs. Importance	RF Importance	XGB Importance
InventoryTurnover	0.056089	0.056089	0.030937	0.036420
ProductionCost	0.051716	0.051716	0.027198	0.027782
EnergyEfficiency	-0.045549	0.045549	0.022840	0.029812
AdditiveMaterialCost	0.043202	0.043202	0.024409	0.026204
AdditiveProcessTime	0.035864	0.035864	0.024945	0.030594
WorkerProductivity	-0.035269	0.035269	0.023239	0.026643
SafetyIncidents	0.012323	0.012323	0.016844	0.029211
DowntimePercentage	0.003322	0.003322	0.028958	0.030925

QualityScore and *ProductionVolume* are also consistently found to be important across the models. By contrast, some variables exhibit different importance levels from one model to another. For example, while *DowntimePercentage* has a low coefficient in LR, it attains higher importance in both RF and XGBoost. This indicates that linear models may be insufficient for capturing certain nonlinear relationships. Similarly, the variable *DeliveryDelay* stands out clearly in the XGBoost model, whereas it remains more in the background in LR and RF.

Overall, the common findings across the three models indicate that indicators related to maintenance, quality, and production volume are the primary determinants of in-process quality risk, whereas differences between the models show that interaction and nonlinear effects among variables are captured more effectively, particularly by tree-based methods. Therefore, the feature importance analysis not only helps to explain model behavior, but also provides a practical roadmap for identifying which parts of the production process should be targeted in order to reduce defect risk.

5.3. Comparison of Model Predictions on A New Observation

To assess the practical usefulness of the models, the predictive performance of the three models was compared on an example new observation representing a production scenario (Table 4). This new observation simulates a manufacturing situation in which features such as production volume, supplier quality, maintenance time, quality score, and energy consumption are set at realistic levels.

Table 4. Feature values of the new observation used in the model comparison

Feature	Value	Feature	Value
ProductionVolume	750	InventoryTurnover	5.2
ProductionCost	12000	StockoutRate	3.1

Table 4. Feature values of the new observation used in the model comparison (*Cont.*)

Feature	Value	Feature	Value
SupplierQuality	92.5	WorkerProductivity	95
DeliveryDelay	1	SafetyIncidents	2
DefectRate	2.3	EnergyConsumption	2500
QualityScore	88	EnergyEfficiency	0.32
MaintenanceHours	10	AdditiveProcessTime	6.5
DowntimePercentage	1.5	AdditiveMaterialCost	230

The results indicate substantial differences among the models (Table 5). LR predicts the defect class for this observation as 1 (high risk) and assigns a very high probability to this outcome (≈ 0.99999). By contrast, RF classifies the observation in class 0 (low risk) and estimates a more moderate defect probability of 0.1667. The XGBoost model, in turn, yields the lowest risk estimate, computing the defect probability at approximately 0.0043.

These results show that the models differ in the sensitivity of their decision boundaries. The high sensitivity of LR stems from the fact that its linear decision boundary more readily labels certain combinations of variables as "risky." RF and XGBoost, by contrast, assign the same observation to a lower-risk category because they capture interactions and nonlinear relationships among features more effectively. In particular, the extremely low defect probability produced by XGBoost can be attributed to its tree-based structure, in which most observations similar to this one tend to fall into the non-defective class.

Table 5. Comparison of model predictions on a new observation

Model	Predicted Class	Predicted Probability of High Defect (P(Class=1))
Logistic Regression	1 (High Defect)	0.99999
Random Forest	0 (Low Defect)	0.16667
XGBoost	0 (Low Defect)	0.00427

5.4. Discussion

In this study, among the three ML models compared, XGBoost achieving the highest performance is closely related to its structural advantages. XGBoost stands out by constructing a strengthened ensemble model in which sequential weak learners minimize residual errors, by effectively controlling overfitting through its regularization (L1–L2) mechanisms, and by performing well on data structures characterized by complex, nonlinear relationships. Since manufacturing processes exhibit a high degree of variability driven by the interaction of multiple inputs, it is to be expected that XGBoost can capture such patterns. The findings obtained in this study confirm this tendency.

Although LR has the advantage of interpretability, its reliance on a linear decision boundary can be limiting in highly multidimensional and complex processes such as manufacturing. In this study, the model lagged behind in terms of classification performance, particularly in situations where interactions among variables were strong and nonlinear relationships were dominant. Moreover, in the presence of an imbalanced data structure, the sensitivity of LR tends to decrease. In this dataset, where the high-defect class is dominant, this caused the model to make more conservative predictions. This is clearly reflected in the prediction results on the new data.

When evaluated from the perspective of real manufacturing environments, the results indicate that all three models are practically usable, but that the most reliable outputs for decision-support systems are obtained particularly from ensemble models. Models such as XGBoost and RF, with their high accuracy and consistent performance, can reduce operators' workload in defect detection processes on the production line, contribute to maintenance planning, and support supply chain optimization. Nevertheless, LR remains an important tool due to its simple structure and interpretability advantage, enabling production managers to quickly understand which factors increase defect probability.

From a quality improvement perspective, the importance scores obtained from the three models show that variables such as *MaintenanceHours*, *DefectRate*, and *QualityScore* play a central role in defect formation. This indicates that firms should place greater emphasis on maintenance planning,

quality control procedures, and the relationship between production volume and quality. Moreover, the fact that the new-data scenario is assigned to different classes by the models suggests that companies should consider adopting a multi-model approach, since relying on the decision of a single model may introduce risk in critical production decisions.

6. CONCLUSION AND RECOMMENDATIONS

In this study, three different ML models (LR, RF, and XGBoost) were comprehensively compared for predicting defect states in manufacturing processes. The dataset used in the analysis consists of multidimensional variables such as production volume, supply chain performance, quality indicators, maintenance activities, energy consumption, and labor productivity, thereby reflecting the characteristic complexity of modern manufacturing environments. In addition, the class imbalance problem in the dataset was addressed using methods such as SMOTE and class weights, enabling the models to produce more balanced predictions.

The findings show that XGBoost is the most successful model in terms of overall performance. This can be explained by its ability to capture nonlinear relationships and its robustness against overfitting through regularization mechanisms. The RF model also demonstrates a high level of success, particularly by providing stable results in feature importance rankings. LR, on the other hand, while advantageous in terms of interpretability, remains comparatively weaker in performance because it cannot fully capture the dynamics of complex manufacturing processes.

When the feature importance scores are examined, it is observed that MaintenanceHours, DefectRate, QualityScore, and ProductionVolume play a decisive role in defect prediction. This finding underscores the importance for firms of strengthening their maintenance strategies, optimizing quality control procedures, and adopting a more fine-grained approach to production planning. In the prediction for the new observation, the differences observed among the models indicate that a multi-model approach should be considered in manufacturing systems, and that relying on a single model in critical decision-making processes may be risky.

Based on the findings of this study, the following recommendations can be made:

- Ensemble methods such as XGBoost or RF can be integrated into operational systems for the early detection of production defects.

- Regular monitoring of maintenance and quality control processes is a critical area for improvement, particularly in light of the high importance of the MaintenanceHours and QualityScore variables.
- To enhance the consistency of model results, firms should standardize their data collection processes, reduce the proportion of missing data, and improve overall data quality.
- In situations that require straightforward interpretation, LR remains a valuable tool, clearly showing decision-makers which variables increase defect probability.
- Future studies can be supported by explainable AI techniques such as SHAP values, enabling a clearer understanding of model decisions.
- The models' generalization capability could be enhanced by extending the dataset to include different product types, multiple production lines, or a time-series structure.

In conclusion, this study has shown that ML-based classification models can serve as a powerful decision-support tool in manufacturing processes. With appropriate feature selection, suitable data preprocessing steps, and balanced model evaluation, companies can transform their quality improvement processes into a more systematic and predictable structure.

REFERENCES

Tercan, H., & Meisen, T. (2022). Machine learning and deep learning based predictive quality in manufacturing: a systematic review. Journal of Intelligent Manufacturing, 33(7), 1879-1905.

Deokar, S., Kumar, N., & Singh, R. P. (2025). A comprehensive review on smart manufacturing using machine learning applicable to fused deposition modeling. Results in Engineering, 104941.

Kausik, A. K., Rashid, A. B., Baki, R. F., & Maktum, M. M. J. (2025). Machine learning algorithms for manufacturing quality assurance: A systematic review of performance metrics and applications. Array, 100393. https://doi.org/10.1016/j.array.2025.100393

de Giorgio, A., Cola, G., & Wang, L. (2023). Systematic review of class imbalance problems in manufacturing. Journal of Manufacturing Systems, 71, 620-644.

Rabie El Kharoua. (2024). Predicting Manufacturing Defects Dataset [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/8715500

Chen, C., Li, X., & Wang, K. (2024). Applying XGBoost for Fault Prediction in Industrial Production Line. Journal of Intelligence and Knowledge Engineering (ISSN: 2959-0620), 2(3), 155.

Ördek, B., Borgianni, Y., & Coatanea, E. (2024). Machine learning-supported manufacturing: A review and directions for future research. Production & Manufacturing Research, 12(1), 2326526. https://doi.org/10.1080/21693277.2024.2326526

Antosz, K., Knapčíková, L., & Husár, J. (2024). Evaluation and Application of Machine Learning Techniques for Quality Improvement in Metal Product Manufacturing. Applied Sciences, 14(22), 10450. https://doi.org/10.3390/app142210450

Md, A. Q., Jha, K., Haneef, S., Sivaraman, A. K., & Tee, K. F. (2022). A review on data-driven quality prediction in the production process with machine learning for industry 4.0. Processes, 10(10), 1966. https://doi.org/10.3390/pr10101966

Borucka, A., & Grzelak, M. (2019). Application of logistic regression for production machinery efficiency evaluation. Applied Sciences, 9(22), 4770. https://doi.org/10.3390/app9224770

Mehregan, M. R., Rezasoltani, A., & Khani, A. M. (2025). A Novel Hybrid Machine Learning Model for Defect Prediction in Industrial Manufacturing Processes. Contributions of Science and Technology for Engineering, 2(4), 43-58. https://doi.org/10.22080/cste.2025.29099.1037

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. IEEE Transactions on knowledge and data engineering, 21(9), 1263-1284. DOI: 10.1109/TKDE.2008.239

Ogrizović, M., Drašković, D., & Bojić, D. (2024). Quality assurance strategies for machine learning applications in big data analytics: an overview. Journal of Big Data, 11(1), 156.

de Giorgio, A., Cola, G., and Wang, L. (2023). Systematic review of class imbalance problems in manufacturing. Journal of Manufacturing Systems, 71, 620-644. https://doi.org/10.1016/j.jmsy.2023.10.014

Gao, X., Xie, D., Zhang, Y., Wang, Z., Chen, C., He, C., ... & Zhang, W. (2025). A comprehensive survey on imbalanced data learning. arXiv preprint arXiv:2502.08960. https://doi.org/10.48550/arXiv.2502.08960

Chen, W., Yang, K., Yu, Z., Shi, Y., & Chen, C. P. (2024). A survey on imbalanced learning: latest research, applications and future directions. Artificial Intelligence Review, 57(6), 137. https://doi.org/10.1007/s10462-024-10759-6

Hua, Y., Stead, T. S., George, A., & Ganti, L. (2025). Clinical risk prediction with logistic regression: Best practices, validation techniques, and applications in medical research. Academic Medicine & Surgery. https://doi.org/10.62186/001c.131964

Hosmer Jr, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied logistic regression. John Wiley & Sons. DOI: 10.1002/9781118548387

Breiman, L.(2001). Random Forests. Machine Learning 45, 5–32. https://doi.org/10.1023/A:1010933404324

Scornet, E. (2021). Trees, forests, and impurity-based variable importance in regression. arXiv:2001.04295. https://doi.org/10.48550/arXiv.2001.04295

Khan, A. A., Chaudhari, O., & Chandra, R. (2024). A review of ensemble learning and data augmentation models for class imbalanced problems: Combination, implementation and evaluation. Expert Systems with Applications, 244, 122778. https://doi.org/10.1016/j.eswa.2023.122778

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794). https://doi.org/10.1145/2939672.2939785

Dalal, S., Rani, U., Lilhore, U. K., Dahiya, N., Batra, R., Nuristani, N., & Le, D. N. (2024). Optimized XGBoost Model with Whale Optimization Algorithm for Detecting Anomalies in Manufacturing. Journal of

Computational and Cognitive Engineering. https://doi.org/10.47852/bonviewJCCE42023545

Nilsson, F., & Kyrk, D. (2025). Anomaly Detection In Manufacturing For Quality Control.

Prasad-Rao, J., Heidary, R., & Williams, J. (2023). Detecting Manufacturing Defects in PCBs via Data-Centric Machine Learning on Solder Paste Inspection Features. arXiv preprint arXiv:2309.03113. https://doi.org/10.48550/arXiv.2309.03113

Qu, D., Gu, C., Zhang, H., Liang, W., Zhang, Y., and Zhan, Y. (2024). Research on Critical Quality Feature Recognition and Quality Prediction Method of Machining Based on Information Entropy and XGBoost Hyperparameter Optimization. Applied Sciences, 14(18), 8317. https://doi.org/10.3390/app14188317

He, H., and Garcia, E. A. (2009). Learning from imbalanced data. IEEE Transactions on knowledge and data engineering, 21(9), 1263-1284. DOI: 10.1109/TKDE.2008.239

Saito T, Rehmsmeier M (2015) The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. PLoS ONE 10(3): e0118432. https://doi.org/10.1371/journal.pone.0118432

Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC genomics, 21(1), 6. https://doi.org/10.1186/s12864-019-6413-7

Fawcett, T. (2006). An introduction to ROC analysis. Pattern recognition letters, 27(8), 861-874. https://doi.org/10.1016/j.patrec.2005.10.010

Kelleher, J. D., Mac Namee, B., & D'arcy, A. (2020). Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies. MIT press.

Chapter 3

Web Application Firewall (WAF)

Fikri AĞGÜN[1]
Raif SİME[2]

ABSTRACT

A Web Application Firewall (WAF) is a security solution designed to protect web applications against cyber threats. The increasing volume of cyber attacks and the widespread adoption of web applications have made the use of WAFs essential for ensuring data security. The evolution of WAFs encompasses a transformation from their initial, simple functionalities into more intelligent systems through the integration of artificial intelligence and machine learning. WAFs operate by analyzing incoming requests to modern web applications based on predefined rules, evaluating these requests and blocking suspicious activities. They are deployed in various forms (cloud-based, hardware-based, and software-based) across different domains and are particularly preferred in environments with high data sensitivity, such as e-commerce platforms and financial institutions. However, WAFs also have limitations; complex rule management and high false positive rates may adversely affect user experience. Moreover, their effectiveness is often confined to known threats, which can result in limited protection against emerging attack vectors.

Keywords: Web application security, Reverse proxy, Cross-Site scripting (XSS) attacks, SQL injection, Regular expressions (ReGex)

[1] Assist. Prof. Dr., Bitlis Eren University, faggun@beu.edu.tr, ORCID: 0000-0001-9550-1462
[2] Bitlis Eren University, rsime@beu.edu.tr, ORCID: 0009-0008-4292-2456

1. INTRODUCTION

The rapid evolution of the Internet has rendered web applications one of the most critical components of organizations. Numerous processes, ranging from banking and e-government services to educational systems and e-commerce, are now conducted on web-based platforms. However, this evolution has simultaneously introduced significant security threats. The inadequacy of traditional security mechanisms particularly network-level firewalls and IDS/IPS solutions in preventing attacks targeting web applications has led to the emergence of the need for Web Application Firewalls (WAFs). Institutions and organizations with network traffic implement strict security measures and surround their systems with multiple preventive solutions. Nevertheless, due to the conscious or unconscious use of various technical and facilitative services within applications operating on web platforms, there is always a potential for security vulnerabilities to arise.

Even a minor vulnerability in corporate systems may lead to exposure to cyber attacks, resulting in reputational damage and significant business losses. In this context, Web Application Firewall (WAF) solutions come into play and assume a major role in protecting web applications.

2. WHAT IS A WEB APPLICATION FIREWALL?

A WAF is a specialized security mechanism designed to detect, block, and filter attacks targeting web applications, and to stop malicious traffic before it reaches the application. Unlike traditional firewalls, a WAF does not operate at the network layer but directly at the application layer (Layer 7 of the OSI model). Accordingly, it performs a detailed analysis of all incoming HTTP and HTTPS requests to determine whether they contain malicious content. The primary objective of a WAF is to establish a protective shield against attacks targeting the underlying code base, database, and business logic of the web application.

Web applications are inherently dynamic systems, particularly due to the processing of user inputs. This characteristic leads to an expansion of the attack surface with every newly developed feature, added parameter, and integrated module.

The advanced structure of web applications, the lack of validation for user-supplied data, and the fact that not all developers adhere to the same security standards increase the likelihood of security vulnerabilities. These vulnerabilities are frequently exploited by attackers and can often lead to severe consequences such as data theft, content manipulation, unauthorized access, and complete system compromise. WAFs have been developed to mitigate the

impact of such vulnerabilities in web applications and to block malicious requests before they reach the application. Their core features include simple filtering; regular expression-based filtering; URL encoding validation; Unicode encoding validation; auditing; null byte attack prevention; upload memory limitations; and server identity masking (Razzaq et al., 2013). The operating principle of a WAF is to filter all traffic directed to the application and allow only secure traffic to pass through. A WAF analyzes the headers, parameters, URL structure, body, and cookies of HTTP requests. It is also capable of detecting anomalies related to session management, authentication attempts, IP addresses with an excessive number of failed login attempts, suspicious bot activities, and behaviors resembling known attack signatures. Today, in order to mitigate increasing cyber threats and enhance the level of protection, efforts are being made to establish a new framework for existing traditional firewalls by integrating artificial intelligence support. As can be observed from the sample studies in the literature discussed below, WAF systems which operate at the application layer and are significantly more effective than traditional, hardware-based system-level countermeasures against cyber attacks are being combined with contemporary artificial intelligence algorithms, emerging as highly effective, fast, and high-performance security solutions.

Various methodologies and techniques, such as secure coding, configuration analysis, and the deployment of web application firewalls, are employed for application security. To prevent web application issues, web administrators typically rely on web application firewalls. Web application firewalls operate at the web application layer, perform in-depth inspection of HTTP packets and each of their components, and search for web application attacks. They detect malicious strings and configuration errors by using different techniques such as whitelisting, blacklisting, and greylisting. (Razzaq et al., 2013).

A Web Application Firewall (WAF) performs deep packet inspection of the network traffic occurring between the client and the server. By analyzing the data transmitted between the client and the server, a WAF can detect potential attacks even if the application itself lacks such detection capabilities. Utilizing the default configuration of a web server may lead to security vulnerabilities despite the presence of a firewall; this situation must be mitigated through comprehensive security testing (Clincy & Shahriar, 2018). As a solution to this problem, WAFs are extensively utilized. In the studies on WAFs available in the literature, WAFs have been examined from multiple perspectives and their benefits have been documented. In their work focusing on the use of WAFs for multi-attack detection, the authors concentrate on an Adaptive Web Application Firewall (WAF) that employs machine learning for real-time threat detection,

enhances security, and reduces cyber risks. They report that WAF implementations are capable of protecting against a wide range of threats, including SQL injection, DDoS attacks, directory traversal, and CSRF, and that the system can reliably detect threats by distinguishing malicious patterns with fewer false positives (Maheshwari et al., 2024). In their study focusing on an advanced WAF that leverages machine learning for enhanced security, Dhote et al. aimed to distinguish between different types of attacks at the application layer by classifying requests, and conducted a noteworthy investigation on attack detection using WAFs. (Dhote et al., 2024). In another study employing deep learning-based artificial intelligence to enhance web application security, the authors achieved an accuracy rate of 98.61% in attack detection with their proposed CNN-LSTM model. They emphasized that the performance of the DL-based WAF is more effective than that of traditional rule-based WAFs (such as ModSecurity), and demonstrated that the critical and high-severity vulnerabilities observed in conventional systems are effectively mitigated by the DL-based WAF (Muttaqin & Sudiana, 2025). In a study proposing a Web Application Firewall (WAF) that employs hybrid detection methods for XSS attacks, the authors introduced an effective artificial intelligence approach for the early detection of XSS attacks by leveraging machine learning and deep learning techniques. Extensive experimental evaluations demonstrated that the random forest method, when used with the proposed feature set, outperforms state-of-the-art approaches and achieves a high performance score of 0.99. (Younas et al., 2024).

3. WHY HAS A WAF BECOME NECESSARY?

Today, web applications have become an indispensable component for institutions and organizations in conducting their operations; however, alongside the convenience they provide, they also introduce numerous vulnerabilities and issues. These systems have been exposed to malicious activities that may lead to severe and often uncontrollable consequences such as data theft, disruption of business processes and functions, and service outages. Among the major initiatives undertaken to contain and prevent these threats, the WAF stands out as one of the most significant countermeasures.

Web applications have become one of the most critical components of modern organizations. Government services, educational platforms, e-commerce websites, banking systems, enterprise management tools, API-based microservice architectures, and cloud-based applications now play a central role in the functioning of society and the economy. Within this expanding digital ecosystem, security is not merely a technical requirement but a multi-layered

necessity that extends from national security and financial stability to business continuity and personal data protection. Consequently, the rise in attacks targeting web applications has evolved into a serious issue that significantly affects institutions, individuals, and states.

At precisely this point, Web Application Firewall (WAF) technology has emerged as one of the indispensable components of modern cyber security architectures. The primary objective of a WAF is to stop attacks targeting web applications before they reach the application, to detect anomalous behavior, and to provide protection against threats occurring at the application layer. However, there are much deeper reasons why a WAF is regarded not merely as a necessity but as an obligation. These reasons are closely related to technological advancements, the evolution of attack patterns, and the transformation of organizational operating models.

Below, the reasons why a WAF is needed are examined and elaborated from historical, technical, operational, and security perspectives in a comprehensive manner.

3.1. The Explosion of Web Applications and the Increased Attack Surface as a Driver for WAF Adoption

The acceleration of digital transformation since the early 2000s has led to an extraordinary increase in the use of web applications. The Internet environment, which previously consisted only of simple, informational websites, has over time evolved into complex structures such as:

- User Account Management Systems
- Online Payment Modules
- E-Government Services
- E-Signature and Identity Authentication Services
- API and Microservice-based Architectures
- Remote Education Systems
- Enterprise ERP, CRM and HR Systems

Protecting such an extensive surface manually is practically impossible. Regardless of how carefully developers work, numerous risks naturally emerge, such as: Coding errors, Insufficient input validation, Iinadequate input filtering, Authentication weaknesses, Security issues in third-party libraries.

For this reason, WAF technology, which filters incoming traffic to the application, blocks suspicious requests, and provides an additional protection layer for the application has become mandatory.

3.2. The Inadequacy of Traditional Firewalls Against Application-Layer Attacks

Traditional firewalls and IDS/IPS solutions operate at the network layer. These systems block attacks by inspecting: IP addresses, port numbers, protocol types, packet direction.

However, the majority of modern cyber attacks occur at the application layer. These attacks typically resemble standard web traffic, originate from ports 80 and 443, and carry malicious payloads embedded within seemingly normal HTTP requests (as in the case of SQL Injection, XSS (Cross-Site Scripting), RFI/LFI, and CSRF attacks).

For example, attacks such as SQL Injection, XSS (Cross-Site Scripting), RFI/LFI, CSRF, Path Traversal, Command Injection, Bad Bot Attacks, API Abuse, Business Logic Attacks are perceived as legitimate traffic by a traditional firewall, because they appear as ordinary HTTP requests.

A traditional firewall, when inspecting such requests, may effectively conclude: “This is port 443, protocol HTTPS. It does not appear dangerous,” and therefore forward the request to the internal network. However, the malicious content embedded in the parameters or payload of the request may completely compromise the application. For this reason, a mechanism capable of understanding application-layer attacks has become necessary, and this mechanism is the WAF.

3.3. The Rise of OWASP Top 10 Attacks and Security Vulnerabilities

The OWASP Top 10, which is updated every few years, lists the most critical web application vulnerabilities worldwide. Nearly all entries in this list stem from the way web applications process user input. Most of these vulnerabilities arise from issues such as injection attacks, authentication weaknesses, authorization flaws, improper input validation, and security misconfigurations.

A WAF provides direct protection particularly against items in the OWASP Top 10 such as ***Injection, Broken Authentication, Sensitive Data Exposure, XML External Entities, Broken Access Control, Security Misconfiguration, and XSS***. Since the absence of adequate controls against the OWASP Top 10 is considered a major risk in corporate security audits, the WAF has become a critical component for meeting these requirements.

3.4. The Rise of SQL Injection, XSS, and Other Critical Vulnerabilities

Over the past 20 years, some of the most widely used attack types worldwide have included SQL Injection, XSS, File Inclusion (LFI/RFI), Command

Injection, and Broken Authentication. Because these attacks can be executed even with relatively simple techniques, they are frequently observed especially in small, medium-sized, and poorly coded systems. An attacker may inject an SQL command into the parameters of a web application to take control of the database, inject JavaScript code to hijack user accounts, or manipulate the file upload mechanism to execute commands on the server. The common characteristic of these attacks is that all of them occur at the application layer. A WAF detects and blocks such attacks using signature-based analysis, behavioral analysis, and input validation techniques.

3.5. The Rise of Zero-Day Vulnerabilities and Aggressive Attack Techniques

In the contemporary cybersecurity landscape, threat actors are capable of exploiting undiscovered (zero-day) vulnerabilities with increasing rapidity. In the event of a zero-day vulnerability, relying solely on vendor patches is insufficient to mitigate such attacks. Upon the discovery of a vulnerability, a remediation cycle is required wherein the software vendor must fix the flaw, users must acquire the update, and system administrators must test and deploy the patch. While this process may span days or even weeks, attackers are often able to exploit the vulnerability within minutes. Consequently, a WAF protects the application by filtering malicious traffic until the official patch is deployed, effectively buffering the system against zero-day attacks.

3.6. The Rise of Distributed Microservices and API-based Architectures

The paradigm shift from traditional monolithic applications to microservices architectures has significantly compounded the complexity of security management. Modern systems are no longer constituted by a single application; rather, they function as an aggregate of dozens of API endpoints, hundreds of microservices, diverse protocols, and heterogeneous authentication mechanisms. As this architectural complexity increases, security risks escalate accordingly. Consequently, the Web Application Firewall (WAF) has become an indispensable primary solution for API security. Within this complex architectural framework, a WAF: Analyzes API traffic, Blocks malicious bots, Detects authentication violations, Enforces rate limiting, Prevents data manipulation.

3.7. Automated Attacks, Bot Traffic and Scraping Threats

While cyberattacks were historically executed manually, the contemporary threat landscape is dominated by automated tools such as SQLmap, DirBuster,

Nikto, and Burp Suite automations, alongside Python scripts and AI-assisted attack vectors. These tools provide adversaries with the capability to perform high-velocity scanning, automated exploit attempts, and API exploitation. Furthermore, malicious bots are capable of engaging in activities such as price scraping, brute force login attempts, comment spamming, API token theft, and content manipulation.

By analyzing this traffic, a WAF: Identifies bot behavior, Issues CAPTCHA challenges, Enforces rate limiting, Performs IP reputation checks, Blocks anomalous requests. Consequently, it safeguards the web application against automated attacks.

3.8. The Necessity of Protection Against DDoS and Application Layer Flood Attacks

Modern iterations of DDoS attacks no longer target solely the network layer but increasingly focus on the application layer (Layer 7). For instance, attacks such as:

- Flooding an API endpoint with thousands of requests per second,
- Overloading login forms via brute-force attempts,
- Submitting resource-intensive queries to search fields,
- Disrupting filtering mechanisms

can exhaust system resources and precipitate application failure. It is not feasible to mitigate application layer DDoS attacks using standard network firewalls alone. Consequently, a WAF intervenes to protect the application by employing mechanisms such as rate limiting, behavioral analysis, IP reputation checks, bot scoring, and automated blocking.

3.9. Enterprise Requirements and Regulatory Compliance

In order to ensure logging and the security of real-time transactions at the application level across various sectors, the deployment of Web Application Firewalls (WAF) has become a standard and indispensable requirement. This necessity arises from regulatory mandates and legal obligations that enforce specific protection levels for web applications, including: PCI-DSS (mandatory for payment systems), ISO 27001 (security requirements), GDPR/KVKK (prevention of data breaches), Banking regulations, Public sector security standards, Healthcare data protection laws.

Given that a single data breach can result in substantial financial penalties, reputational damage, operational downtime, and litigation processes, organizations are compelled to adopt WAF solutions proactively.

3.10. The Inevitability of Human Error and the Persistence of Security Vulnerabilities

Regardless of the rigor of the development process, no software can be rendered entirely secure. Factors such as the inevitability of human error, increased error rates in code developed under time constraints, the continuous emergence of new vulnerabilities in third-party libraries, and the lack of familiarity among new developers with legacy security decisions make it impossible to exhaustively test for all vulnerabilities in complex systems. By mitigating the impact of these unavoidable errors, a WAF enhances the overall security resilience of the system.

4. THE EVOLUTION OF WAF

The evolutionary trajectory of WAF technology originated as a technical countermeasure to the escalating cyber threats associated with the proliferation of the Internet. To fully comprehend the contemporary status of WAFs, it is essential to analyze both the evolution of web applications and the shifting landscape of attack vectors. This progression represents not merely a technical advancement, but the result of a comprehensive transformation driven by enterprise requirements, security standards, next-generation software architectures, and DevSecOps methodologies.

The following section examines the timeline of WAF development, ranging from its inception to its current sophisticated architecture, through a chronological framework.

4.1. The Early Era of Web Applications and Fundamental Security Needs (1990–2000)

In the mid-1990s, the World Wide Web consisted primarily of static HTML pages. As client-server interaction was minimal, the attack surface remained relatively narrow. During this period, fundamental enterprise security solutions were limited to traditional network firewalls and Intrusion Detection/Prevention Systems (IDS/IPS). However, with the advent of web technologies such as CGI, PHP, ASP, and Java Servlets, websites evolved into dynamic and interactive platforms. Users gained the capability to submit data, complete forms, and interact directly with backend databases.

This shift precipitated the onset of application-level exploits, leading to the emergence of attack vectors such as SQL Injection, Command Injection, and File Inclusion, alongside a rise in database manipulation attempts.

Traditional network firewalls, operating primarily at the **TCP/IP layer**, were incapable of detecting such attacks. Similarly, while IDS/IPS systems analyzed

network traffic, they lacked the depth of inspection required to comprehend the specific business logic of web applications. Consequently, a novel security layer was required to mitigate threats targeting the application layer. This necessity laid the groundwork for the inception of the WAF.

4.2. The Emergence of First-Generation WAFs (2000–2005)

The establishment of the Open Web Application Security Project (OWASP) in the early 2000s catalyzed a significant shift in the security community's focus toward web application security. The publication of the OWASP Top 10 list established global recognition regarding the criticality of application security. The fundamental characteristics of the first-generation WAFs developed during this era were as follows:

- ***Static Signature and Rule-Based Architecture***

Early WAF solutions analyzed attacks using static signature sets, analogous to IDS systems. For instance, patterns such as ***" ' OR 1=1--"*** were identified as SQL Injection attempts and subsequently blocked.

- ***HTTP Packet-Level Filtering***

These WAFs inspected HTTP requests and blocked those deemed anomalous. However, they lacked the capability to comprehend the application context.

- ***Reverse Proxy Architecture***

Many WAF solutions operated as reverse proxies, analyzing incoming traffic to the web server within an intermediary layer.

- ***Limited Flexibility***

Rule-based systems frequently generated high rates of false positives, thereby increasing the operational overload associated with WAF deployment.

During this period, WAF technology began to advance commercially, and the first open-source solutions, such as ModSecurity, were introduced.

4.3. The Expansion of Web and the Maturation Phase of WAF Technology (2005–2012)

Post-2005, the landscape of web applications expanded significantly with the widespread adoption of AJAX, SOAP, REST APIs, desktop-like web applications, and mobile web technologies. Concurrently, attack vectors diversified to include Cross-Site Scripting (XSS), CSRF, RFI/LFI, XML Injection, Session Hijacking, and Cookie Manipulation.

Since the majority of these attacks targeted application behavior, WAF solutions were compelled to evolve. During this period, WAFs acquired the following capabilities:

- **Behavioral Analysis and Anomaly Detection**

Recognizing the inadequacy of static signatures, WAF models were developed to learn the baseline traffic of an application and detect deviations.

- **Advanced Rule Engines**

Flexible rule engines based on Regular Expressions (RegEx) and extensible rule sets in tools such as ModSecurity emerged.

- **OWASP ModSecurity Core Rule Set (CRS)**

The introduction of the CRS marked a significant milestone in the standardization of WAF rule sets.

- **Application Layer Protection**

In addition to HTTP/HTTPS inspection, advanced controls were implemented for data formats such as JSON, XML, and SOAP. This era facilitated the widespread adoption of WAFs within enterprise infrastructures.

4.4. Rise of Cloud Technologies and the Reshaping of WAF (2012–2018)

The proliferation of cloud-based applications, microservices architectures, and the exponential increase in traffic volume necessitated the transformation of WAFs into scalable architectures. During this period, Content Delivery Network (CDN)-based protection services and Software-as-a-Service (SaaS) WAF solutions gained particular prominence. The pivotal transformation points of this era can be outlined as follows:

- **Cloud WAF Solutions**

Major technology corporations such as Cloudflare, AWS, Azure, and Google began offering WAF services via globally distributed infrastructures.

- **DDoS Integration**

WAFs gained the capability to detect and mitigate Distributed Denial of Service (DDoS) attacks in addition to ensuring application security.

- **WAF as a Service**

The service model requiring no local installation and offering real-time updates became widespread.

- **API Security**

Specialized controls for REST and SOAP APIs were integrated into the functional repertoire of WAFs.

This era marks the period in which WAF technology became more user-centric, significantly reducing the operational overload associated with deployment and maintenance.

4.5. Artificial Intelligence, Machine Learning, and the Modern WAF Era (2018–Present)

The foundation of the current state of WAF technology is constituted by decision-making mechanisms based on Artificial Intelligence (AI) and Machine Learning (ML).

Key Features of Modern WAFs:

- **Machine Learning-Based Anomaly Detection**

By learning the behavioral patterns of the application, anomalous requests are automatically identified. This methodology exhibits significantly higher efficacy compared to static signature-based approaches.

- **Bot Management and Anti-Automation**

Contemporary WAFs possess the capability to autonomously distinguish and mitigate malicious bots, scrapers, and credential stuffing attempts.

- **Zero-Day Attack Detection**

Early warning mechanisms facilitate the detection of attacks for which signatures have not yet been generated.

- **Large-Scale Distributed Architectures**

Utilizing CDN-based global networks, requests are filtered at the network edge (nearest point of presence).

- **API Gateway Integration**

API security has evolved into a fundamental component of WAF architecture.

- **DevSecOps Integration**

Modern WAF solutions are integrated into CI/CD pipelines, ensuring security enforcement as early as the code development phase. Consequently, WAF technology has transcended its role as a mere firewall, evolving into a comprehensive application security platform.

4.6. Future Projections and the Evolution of WAF Technology

The evolutionary trajectory of WAF technology is not yet complete. In the forthcoming years, it is anticipated that Artificial Intelligence will evolve toward a rule-less operational model through Fully Automated Security Policies; WAF capabilities will converge with Service Mesh technologies (e.g., Istio, Linkerd) via the full integration of API and microservices security; the focus will extend beyond mere attack detection to include user behavior profiling through User Behavior Analytics (UBA); requests will be filtered at the network edge prior to reaching the data center via the implementation of

Edge-Computing based WAFs; and WAFs will establish themselves as a fundamental authentication layer within the system through Zero-Trust Integration.

5. HOW WAF OPERATES

What is the fundamental concept? What functions does a WAF, where is it positioned?

The primary role of a WAF is to inspect inbound HTTP/HTTPS traffic at the application layer (OSI Layer 7) and intercept malicious requests before they reach the web application. This mechanism is distinct from traditional network security approaches based on IP addresses and ports. A WAF analyzes the request body, Uniform Resource Identifier (URI), HTTP headers, cookies, JSON/XML payloads, and even session logic.

The deployment models of a WAF can be categorized as follows:

- **Reverse Proxy (Inline):** The most prevalent model, wherein client requests are first routed to the WAF for analysis and, if deemed benign, are subsequently forwarded to the origin server.
- **Transparent Bridge:** Deployed within the network infrastructure to passively monitor traffic without modification or to perform active inline blocking.
- **Host-Based (Agent):** Operates as a module directly on the application server, residing on the same host machine as the application itself.
- **Cloud/CDN-Based WAF:** Traffic is filtered through the provider's distributed network infrastructure, facilitating mitigation at the network edge.
- **API Gateway / Service Mesh Integration:** Provides gateway-level integration within microservices or API-first architectures.

The selection of deployment topology is critical regarding security efficacy, latency, and scalability. While reverse proxy and cloud-based WAFs offer distinct advantages in scalability and DDoS mitigation, host-based solutions provide deeper visibility into the application context.

The WAF Request Inspection Pipeline

The request processing logic of a Web Application Firewall typically adheres to the following sequential stages:

1. **TLS Decryption (Termination):** If the WAF operates with inline TLS termination, the encrypted request is first decrypted. (Note: Deep packet inspection is not feasible if TLS termination does not occur at the WAF level.)
2. **HTTP Parsing and Normalization:** This phase involves URL decoding, character normalization, Unicode normalization, and content-type

determination (e.g., JSON, XML, form-data). This step is of critical importance for neutralizing evasion techniques.

3. **Header, URI, and Body Inspection:** A granular analysis is performed on HTTP headers, methods, Uniform Resource Identifiers (URIs), query strings, and the request body.

4. **Rule Application and Modeling:** This stage involves the execution of signature-based rules (pattern matching, Regular Expressions), the evaluation of behavioral and statistical models, and the application of positive/negative security logic (allowlisting/blocklisting).

5. **Rate Limiting and Connection Control:** Traffic volume is assessed against predefined thresholds; temporary bans or throttling may be enforced based on these limits.

6. **Bot Challenge / CAPTCHA:** Based on the calculated bot score, the requester may be redirected to a CAPTCHA or a similar computational challenge.

7. **Action Execution (Allow/Deny/Redirect/Sanitize):** Based on the outcome of the rule evaluation, specific actions such as blocking with an alert or payload sanitization are executed.

8. **Logging, Telemetry, and Forwarding:** Comprehensive logs are generated and forwarded to Security Information and Event Management (SIEM) systems for incident response and forensic analysis.

Inspection Methodologies:

WAFs employ diverse techniques to execute inspection processes. These methodologies can be categorized as follows:

- **Signature-Based Detection:** This method utilizes string matching and regular expressions (RegEx) to identify known attack patterns, such as SQL or JavaScript snippets. While highly efficient in detecting established threats with precision, it often proves inadequate against novel or polymorphic attack variants. For example, a RegEx pattern such as (\bselect\b.*\bfrom\b) identifies SQL Injection attempts.
- **Positive Security Model (Allowlisting):** This approach defines a strict set of permissible request formats (allowlist) and rejects all others. It is a robust technique, particularly suitable for static and well-defined applications. However, its primary disadvantages include the administrative complexity of management within dynamic environments and a high potential for false positives.
- **Negative Security Model (Blocklisting):** This model focuses on identifying and blocking known malicious patterns. It is the most widely adopted approach for filtering out recognized threats.

- **Behavioral and Anomaly Detection:** This technique establishes a baseline traffic profile (e.g., request frequency, parameter structures), triggering alerts upon the detection of deviations. Machine Learning algorithms are often employed to distinguish between benign and anomalous traffic. For instance, 500 login attempts against an endpoint from a single IP address within a short timeframe would be identified as a brute-force attack.
- **Stateful Application:** Logic Inspection The WAF validates session identifiers, CSRF tokens, and the sequential integrity of specific business workflows. It detects logic anomalies, such as a user attempting to submit a payment request while bypassing the requisite checkout sequence.
- **Payload Normalization and Decoding:** This process neutralizes evasion techniques such as path traversal (%2E%2E), Unicode obfuscation, double-encoding, and chunked transfer manipulation. Signature matching and malicious content detection are executed subsequent to the normalization process.
- **Context-Aware Parsing:** The WAF parses payloads according to their specific content type (e.g., JSON, XML, multipart/form-data) to apply relevant security policies. Format-specific vectors, such as XML External Entity (XXE) attacks, are identified at this layer.

WAF Rule Categories:

Web Application Firewalls utilize a diverse range of rule sets to enforce security policies. These encompass simple string and Regular Expression (RegEx) rules for basic pattern matching of known signatures, as well as complex logical rules involving boolean logic or multi-condition criteria. Furthermore, the system employs rate-limiting rules to restrict request frequency, geo-location rules for restrictions based on geographic origin, and IP reputation rules to filter traffic according to the trustworthiness history of IP addresses. Additionally, WAFs support time-based (temporal) rules for policies active during specific timeframes and custom rules, such as specialized rule sets developed for frameworks like ModSecurity.

WAF Action Policies

Upon the triggering of a security rule, a Web Application Firewall (WAF) is capable of executing a diverse range of enforcement actions. These include **Block (Deny)**, which serves to immediately reject the request; **Redirect / Challenge**, employed to enforce verification mechanisms such as CAPTCHA or HTTP 302 redirects; **Alert / Log Only**, a passive mode particularly beneficial during the operational tuning phase for generating telemetry without service interruption; **Sanitize / Scrub**, which neutralizes malicious payloads

while permitting the sanitized request to proceed to the origin; **Quarantine**, utilized to isolate the request or reroute it to a secondary processing queue; and **Rate-Limit / Throttle**, designed to restrict traffic velocity based on defined thresholds.

Performance and Scalability

Since the Web Application Firewall (WAF) is positioned at the application perimeter, latency and throughput are of critical importance. To ensure optimal performance, the following considerations must be addressed:

- **TLS Termination:** Due to the high CPU overhead associated with decryption, hardware acceleration or termination at the network edge should be prioritized.
- **Rule Complexity:** An excessive number of complex Regular Expressions (RegEx) can significantly increase CPU consumption; therefore, rule optimization is essential.
- **Caching / Fast Path:** Caching mechanisms should be implemented to provide rapid access to static content and validated (sanitized) traffic.
- **Horizontal Scaling:** Horizontal scalability should be achieved through Cloud/Content Delivery Network (CDN)-based WAFs or the utilization of load balancers.
- **Connection Persistence and Pooling:** Efficient connection management contributes to performance enhancement.

During the planning phase, the processing cost per request must be evaluated, and Service Level Agreements (SLA) must be taken into account.

Bot Management, CAPTCHA, and Rate Limiting

Modern Web Application Firewalls (WAFs) transcend simple signature-based detection by incorporating bot behavioral analysis through the utilization of **Fingerprinting**, **Behavioral Scoring**, **Challenge/Response**, and **Rate Limiting** techniques. Specifically, **Fingerprinting** is employed to discern between legitimate browsers and automated agents. **Behavioral Scoring** calculates a probabilistic bot score by analyzing telemetry data, including mouse movements and cookie support capabilities. The **Challenge/Response** mechanism validates the client by imposing computational or interactive tasks, such as CAPTCHAs, JavaScript Challenges, or Proof-of-Work algorithms. Furthermore, **Rate Limiting** enforces traffic constraints based on specific endpoints, IP addresses, or authentication tokens.

Attack Evasion Techniques and WAF Countermeasures

Adversaries employ various methodologies to circumvent WAF inspection mechanisms, including:

- **Encoding / Double-Encoding:** Manipulating character sets to disguise malicious strings.
- **Chunked Transfer or Fragmentation:** Splitting requests to bypass pattern matching.
- **Obfuscated Payloads:** Using techniques such as comment injection and whitespace obfuscation to hide payloads.
- **Polymorphic Payloads:** Altering the appearance of the payload while retaining malicious functionality.
- **Out-of-Band / External Channels:** Exploiting vectors like XML External Entity (XXE) or Server-Side Request Forgery (SSRF).
- **Malicious Business-Logic Flows:** Executing attacks that appear syntactically valid but violate business logic.

In response to these evasion attempts, WAFs implement the following countermeasures:

- **Normalization & Decoding:** Applying rigorous input sanitization steps.
- **Multiple Parsing Passes:** Recursively analyzing nested inputs.
- **Context-Aware Parsing:** Conducting content-specific analysis for formats like JSON and XML.
- **Behavioral Detection:** Identifying anomalies based on traffic patterns.

Nevertheless, detecting sophisticated evasion tactics (specifically "low-and-slow" attacks) remains a significant challenge. Consequently, a WAF should not constitute the sole layer of defense within the security architecture.

Having examined the functional mechanics and capabilities of Web Application Firewalls, it is pertinent to illustrate their operational workflow through a sequential use-case scenario involving a standard deployment cycle:

1. A new API endpoint is deployed.
2. The WAF monitors this endpoint in "learning" or "log-only" mode for a designated period (7 days).
3. The WAF generates automated policy recommendations, such as JSON schema validation and rate limiting thresholds.
4. The security operations team reviews the proposed rules and tentatively applies them in the production environment under "log-only" mode.
5. False positives are eliminated, and the rule set is fine-tuned.
6. Upon validation, the rules are transitioned to "block" (active enforcement) mode.

7. The system is integrated with a SIEM solution, and automated ticketing workflows are configured.

This methodology ensures robust security enforcement while maintaining an uncompromised user experience.

6. WAF TYPES AND DEPLOYMENT MODELS

Web Application Firewall technologies have evolved significantly over time, diversifying into various distinct types to address differing operational requirements and usage scenarios. The primary drivers behind the emergence of these diverse WAF classifications include the heterogeneity of application hosting infrastructures, varying security assurance levels, scalability expectations, and the diversity of enterprise governance policies. Consequently, WAF solutions are categorized into multiple distinct classes based on their deployment models, architectural frameworks, traffic processing methodologies, and operational modalities.

WAF Classifications:

- **Based on Deployment Model**
  - Hardware-Based WAF (Appliance)
  - Software-Based WAF
  - Virtual Appliance WAF
  - Cloud-Based WAF
- **Based on Operational Principle**
  - Reverse Proxy WAF
  - Transparent (Bridge Mode) WAF
  - Embedded (In-App) WAF
- **Based on Security Approach**
  - Signature-Based WAF
  - Behavioral (Heuristic) WAF
  - Machine Learning-Based WAF
- **Based on Architecture**
  - Centralized WAF
  - Distributed WAF
- **Based on Service Model**
  - SaaS WAF (WAF-as-a-Service)
  - Managed WAF
- **Based on Specific Use Cases**
  - API Security Firewall (API-WAF)

- WAF with Integrated Bot Management
- CDN-Integrated WAF

In alignment with contemporary technological advancements and evolving requirements, WAF technologies are structured and deployed to address a diverse array of application domains and varying levels of security necessities. Consequently, the functional spectrum and application scope of WAF solutions have expanded significantly.

The application domains of WAF technology can be delineated as follows:

1. **Web Application Protection:** The most fundamental and prevalent use case for a WAF is ensuring the security of web applications. WAFs protect applications against threats such as SQL Injection, Cross-Site Scripting (XSS), Remote Command Execution, File Inclusion attacks, Directory Traversal, CSRF attacks, and XML/JSON manipulation attacks. Given that web applications are central to contemporary business processes, the mitigation of these attacks is of critical importance. As a secondary function, WAFs ensure the **Protection of Sensitive Data**. In systems processing critical data (such as those in banking, healthcare, education, and the public sector) WAF protection prevents data leakage, manipulation, and unauthorized access.
2. **Protection of APIs and Microservices:** The majority of modern software development practices rely on API-based architectures. Mobile applications, IoT devices, microservices, and integration systems operate via APIs. Through API Traffic Inspection, WAFs detect threats such as API brute-force attacks, rate limit violations, JWT token manipulation, API key abuse, and GraphQL query exploitation. Furthermore, as distributed systems entail unique security requirements for each service, WAFs are integrated with **API Gateways** to consolidate all microservices under a centralized and consistent security policy.
3. **E-Commerce and Financial Sector:** E-commerce sites and financial transaction platforms are among the most frequent targets for attackers due to the monetary value, personally identifiable information (PII), and payment data they transmit. In this domain, WAFs perform duties related to **Fraud Prevention**, **Bot Detection**, and protection against malicious automation attempts. Specifically, they provide defense against attacks such as Carding (payment card guessing), Fake Account Creation, Credential Stuffing, and CAPTCHA Bypass attempts. Additionally, many financial institutions are mandated to deploy WAF solutions to ensure compliance with **PCI-DSS** regulations for the protection of payment systems.
4. **Public Sector and Academic Institutions:** Digital platforms operated by public institutions process citizen data, while universities host student,

personnel, and academic records. In this context, WAFs are deployed to ensure the **Protection of Critical E-Government Services**, including Electronic Document Management Systems (EBYS/EDMS), Integration Platforms, Email Portals, Personnel and Student Information Systems, and Public-Facing Web Services. Furthermore, as public sector entities are frequently subjected to active cyber hostilities, **Threat Intelligence Integration** is achieved via WAF deployment. By leveraging real-time threat intelligence, WAFs provide rapid defense capabilities against emerging attack vectors.

5. **Healthcare Sector:** In healthcare systems, personal data represents some of the most sensitive information categories. Consequently, healthcare organizations in many jurisdictions are obligated to deploy WAF solutions. The Protection of Electronic Health Records (EHR) including patient files, laboratory results, appointment systems, and centralized physician appointment systems is facilitated by WAFs. Moreover, Compliance Requirements such as KVKK (PDPL) and HIPAA mandate that healthcare organizations protect data access; WAFs constitute a critical component of this compliance framework.

6. **Cloud Environments and CDN Services:** The proliferation of cloud-based systems has expanded the deployment scope of WAFs. On cloud platforms such as AWS, Azure, and GCP, WAFs are utilized to protect Web Servers, **API Gateways**, **Kubernetes Ingress Controllers**, and **Serverless** backend functions. Additionally, WAF services provided by Content Delivery Networks (CDNs) ensure that attacks are blocked at the **network edge** before reaching the origin server, facilitating integration with DDoS mitigation and enabling low-latency global operations.

7. **Mobile Application Backend Services:** Mobile applications establish direct connectivity to backend services, which predominantly rely on API-based architectures. WAFs perform **Mobile Backend API Protection**, safeguarding applications against threats such as **Token Validation** failures, mobile bot attacks, rate limit violations, and unauthorized data extraction attempts.

8. **IoT and Industrial Systems:** As the number of IoT devices increases, attacks targeting the web-based control panels managed by these devices also escalate. WAFs are deployed to protect IoT management interfaces in domains such as Smart City Infrastructures, Sensor Control Systems, **Industrial SCADA Interfaces**, and Home Automation Systems.

9. **Protection of Internal Enterprise Systems:** WAF deployment is not limited to public-facing services but also extends to intranet environments. **Internal Web Applications**, including Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM) software, Corporate Portals, and HR Automation systems, can be protected by WAFs. Furthermore,

WAFs can be positioned to mitigate risks associated with insider threats (network attacks) caused intentionally or inadvertently by personnel.

10. Regulatory and Compliance Requirements: In numerous sectors, the deployment of WAFs has become mandated by law or industry standards. Compliance frameworks such as **PCI-DSS** (Payment Card Industry), **ISO 27001** (Information Security Management), **GDPR/KVKK** (Personal Data Protection), and **HIPAA** (Healthcare) either recommend or mandate the protection of web applications via WAFs.

11. Mitigation of DDoS and Botnet Attacks: Certain WAF solutions provide integrated protection against Layer 7 DDoS attacks**,** including HTTP Flood, Slowloris, and Cache Bypass attacks, as well as botnet driven volumetric traffic. Additionally, they offer advanced rate limiting and bot filtering capabilities to halt automated attacks.

7. LIMITATIONS OF WAF

Although Web Application Firewall (WAF) solutions have become fundamental components of modern network and application security, like any technology, they possess certain limitations and vulnerabilities. These limitations may arise from both architectural design and usage methodologies. While the protection offered by a WAF can be quite effective with correct configuration and up-to-date rules, it is inherently unable to provide a complete security guarantee. The fundamental limitations of WAFs are discussed below.

1. Inability to Detect All Attacks (False Negative Problem) WAF solutions detect attacks using signature-based, behavior-based, or statistical models. However:

- They may not always detect new, yet unidentified attack techniques ("zero-day attacks").
- Payloads that are cleverly concealed, encoding methods, or multi-layered attacks may evade the WAF's analysis.
- Attacks targeting the application's business logic often appear as normal traffic, making them difficult for the WAF to detect.

For these reasons, WAFs cannot provide 100% attack detection.

2. Generation of False Positives (False Positive Problem): One common issue with WAFs is the generation of false positives. Particularly under strict security rules:

- Normal user requests may be perceived as attacks.
- API requests, dynamic parameters, or custom input formats may be blocked.

- Software development teams frequently need to spend additional time correcting these erroneous blocks.

False positives degrade user experience and increase the management burden.

3. Challenges in Analyzing Encrypted Traffic:

Modern web applications predominantly use HTTPS. To analyze this traffic, a WAF must:

- Perform SSL/TLS termination or operate in reverse proxy mode. This setup:
- Can lead to performance losses.
- Requires an additional certificate management process.
- Makes it challenging to decrypt traffic in certain environments (e.g., applications using mutual TLS on end-user devices).

When encrypted traffic is not fully analyzed, some attacks may go unnoticed.

4. Impact on Performance:

A WAF must analyze every request, model behaviors, and enforce rules. This process:

- May cause latency.
- Can result in performance degradation under load.
- Increases the need for scaling in applications with heavy traffic.

Even cloud-based WAFs can lead to service delays by implementing rate limits during high traffic conditions.

5. Inadequacy Against Business Logic Attacks:

WAFs primarily focus on attacks at the technical layer. However: Threats such as fake return requests, Logic manipulation, Privilege escalation attempts, Multi-step attacks are often not detected by WAFs as they may resemble normal user behavior. These attacks require specialized "business logic security" to be effectively safeguarded against.

6. Difficulty in Staying Up-to-Date:

Attack techniques are continuously evolving. For a WAF to be effective:

- Rules must be regularly updated.
- Machine learning models need to be retrained.
- New threat intelligence must be integrated.

However, many organizations do not perform these updates regularly, causing the WAF to become ineffective over time.

7. Dependency on Development and Operations Teams:

For WAF rules to function correctly, it is essential to recognize all endpoints of the application, model normal behavior, and accurately define allowed parameters. A misconfigured WAF can:

- Render the entire application inaccessible,
- Accidentally block critical endpoints,
- Completely overlook security vulnerabilities.

Therefore, managing a WAF requires specialized expertise.

8. Integration Issues with CDN and Distributed Architecture:

Since a WAF operates under the principles of a reverse proxy or edge firewall, it may encounter integration challenges with CDNs. In microservices architectures, separate configurations may be necessary for each service, and conflicts can arise with API gateway structures. Particularly in container-based environments (like Kubernetes), WAF management can become complex.

9. Ineffectiveness Against Internal Threats:

WAFs primarily provide protection against external threats. However, attacks originating from the internal network, such as abuse by authorized users or database hijacking, are generally not blocked by the WAF.

10. Limitations Against Advanced Evasion Techniques:

Attackers have developed specialized methods to bypass WAFs, including multiple encoding (double/triple encoding), payload fragmentation, HTTP parameter polymorphism, low-rate attacks, and stealth attack techniques. Such methods can make it challenging for the WAF to detect the attack.

11. Inadequacy Against Zero-Day Vulnerabilities:

Due to the operational nature of WAFs, their protection level is low against attacks that are unprecedented, not yet defined by signatures, and exhibit unknown behaviors. WAFs cannot provide complete protection against zero-day attacks.

8. WAF INSTALLATION AND EXAMPLE RULE SET

To provide a reference for a WAF installation, the steps for setting up a WAF using Nginx web server and ModSecurity rule sets within an enterprise network are outlined below. The installation environment chosen is Ubuntu 24.04 LTS, and the ModSecurity installation (Nginx + OWASP CRS) has been carried out on this platform. The following steps detail the installation of ModSecurity + OWASP Core Rule Set (CRS) on an Ubuntu 24.04 LTS server in the most updated and stable manner.

1. System Update Procedure: First, the Ubuntu server to be used as the platform must be updated to the latest version.

Bash

sudo apt update && sudo apt upgrade -y

2. Installing Required Packages: Necessary packages, such as nginx and modsecurity, must be installed.

Bash

sudo apt install -y nginx libnginx-mod-security (In Ubuntu 24.04, ModSecurity is now available in the official repositories under the package libnginx-mod-security).

3. Verification of ModSecurity Nginx Module Installation:

Bash

nginx -t | grep modsecurity veya dpkg -L libnginx-mod-security

The installations of the packages are verified using these commands.

4. Creating ModSecurity Main Configuration File:

Bash

sudo cp /usr/share/modsecurity-crs/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf

5. Editing Basic Settings:

Bash

sudo nano /etc/modsecurity/modsecurity.conf

Change these rows:

conf

instead of "DetectionOnly" do "On" yapın (blocking is active)

SecRuleEngine On

SecRequestBodyAccess On

SecResponseBodyAccess On

SecAuditEngine RelevantOnly # OR "On" (If more logging is desired)

SecAuditLog /var/log/modsecurity/audit.log

SecDataDir /var/cache/modsecurity

SecTmpDir /tmp

SecDefaultAction "phase:1,log,deny,status:403"

SecDefaultAction "phase:2,log,deny,status:403"

6. Installation of OWASP Core Rule Set (CRS):

Bash

sudo apt install -y modsecurity-crs

Here, the CRS is automatically installed in the /usr/share/modsecurity-crs/ directory.

7. Including the CRS Main File: A configuration file is created for this purpose.

Bash

sudo nano /etc/modsecurity/crs.conf

The following content should be added to the configuration file:.

conf

Include /usr/share/modsecurity-crs/crs-setup.conf

Include /usr/share/modsecurity-crs/rules/.conf*

8. Adding ModSecurity to Nginx Configuration:

Bash

sudo nano /etc/nginx/nginx.conf

http { The following line should be added at the top of the block.

nginx

ModSecurity settings

modsecurity on;

modsecurity_rules_file /etc/modsecurity/crs.conf;

9. Extra Check forn Site Configuration (Optional):

For a example site (/etc/nginx/sites-available/default):

nginx

server {

listen 80;

server_name domain.com www.domain.com;

ModSecurity is already enabled globally; if it is desired to enable it again here:

modsecurity on; is performed.

location / {

... other settings

}

}

10. Setting Up Folders and Permissions:

Bash

sudo mkdir -p /var/log/modsecurity

sudo chown www-data:www-data /var/log/modsecurity

sudo mkdir -p /var/cache/modsecurity

sudo chown www-data:www-data /var/cache/modsecurity

11. Testing and Starting:

Bash

sudo nginx -t

if there is no errors:

Bash

sudo systemctl reload nginx

12. Testing if the System is Working:

The URL should be tested in the browser with the format “http://your-server-ip-address/?id=1+OR+1=1”, and if the WAF is functioning, access to the site should be blocked (a 403 response should be returned). The same test can be performed using the curl command:

Bash

curl -i http://127.0.0.1/?test=../etc/passwd should be applied as follows. In this case, if the system is functioning, a "403 Forbidden" message should be received, and a ModSecurity log should be created.

13. Checking the Logs: The logs can be checked using the following command.

Bash

sudo tail -f /var/log/modsecurity/audit.log

14. Adjusting the Paranoia Level to Reduce False Positives (Optional):

Bash

sudo nano /etc/modsecurity/crs/crs-setup.conf

This should be adjusted (the recommended starting level is 2):

conf

```
SecAction \
 "id:900000,\
  phase:1,\
  nolog,\
  pass,\
  t:none,\
  setvar:tx.paranoia_level=2"
```

15. Starting in Log Mode (DetectionOnly) (Optional):

To test initially without blocking:

Bash

sudo nano /etc/modsecurity/modsecurity.conf

conf

SecRuleEngine DetectionOnly

Later, when the system is stable, "SecRuleEngine On" should be configured.

16. Update and Maintenance:

Bash

 # updates of CRS and ModSecurity

sudo apt update && sudo apt upgrade

The WAF installation will be completed after these steps. Now, an additional protection layer is actively present in front of the system. In this section, if we also show an example of a ModSecurity rule used:

```
Default: |application/x-www-form-urlencoded| |multipart/form-data| |multipart/related|
 |text/xml| |application/xml| |application/soap+xml| |application/x-amf| |application/json|
 |application/cloudevents+json| |application/cloudevents-batch+json| |application/octet-stream|
 |application/csp-report| |application/xss-auditor-report| |text/plain|

SecAction \
 "id:900220,\
  phase:1,\
  nolog,\
  pass,\
  t:none,\
  setvar:'tx.allowed_request_content_type=|application/x-www-form-urlencoded| |text/xml| |application/xml| |application/soap+xml| |application/json| |text/plain|'"
```

Through this rule, only the specified file types in the list are allowed.

9. CONCLUSION

A WAF alone does not provide comprehensive protection against all potential threats in a network. Therefore, it cannot be sufficient on its own without additional security layers such as IDS/IPS, DDoS protection, RASP (Runtime Application Self-Protection), Secure Software Development Life Cycle (SSDLC), and code security scans. Considering these disadvantages, it is more appropriate to position a WAF not as a "single solution" security mechanism in front of an enterprise system, but as a component of a Defense in Depth strategy.

In conclusion, WAF solutions have become an indispensable security layer for modern web applications. The attacks that applications face today are increasingly complex, and solely relying on network-level security mechanisms is no longer adequate. For protecting applications developed on both enterprise and individual scales, WAFs are now considered essential components of network security and are actively utilized by many organizations, institutions, and authorities. They are continuously updated to meet current demands and enhanced with features supported by machine learning and artificial intelligence.

REFERENCES

Clincy, V., & Shahriar, H. (2018). Web Application Firewall: Network Security Models and Configuration. *Proceedings - International Computer Software and Applications Conference*, *1*, 835–836. doi:10.1109/COMPSAC.2018.00144

Dhote, S., Magdum, A., Singh, S., & Raigar, D. (2024). ML based Web Application Firewall for Signature and Anomaly Detection Using Feature Extraction. *2024 15th International Conference on Computing Communication and Networking Technologies, ICCCNT 2024*, 1–6. doi:10.1109/ICCCNT61001.2024.10725511

Maheshwari, M., Nayak, A., Sethy, A., & Sujatha, G. (2024). Adaptive Web Application Firewall for Multi-Threat Detection. *Proceedings of 5th International Conference on IoT Based Control Networks and Intelligent Systems, ICICNIS 2024*, 232–238. doi:10.1109/ICICNIS64247.2024.10823239

Muttaqin, R. Z., & Sudiana, D. (2025). Design of Realtime Web Application Firewall on Deep Learning-Based to Improve Web Application Security. *Jurnal Penelitian Pendidikan IPA*, *10*(12), 11121–11129. doi:10.29303/jppipa.v10i12.8346

Razzaq, A., Hur, A., Shahbaz, S., Masood, M., & Ahmad, H. F. (2013). Critical analysis on web application firewall solutions. *2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS)*, 1–6. doi:10.1109/isads.2013.6513431

Younas, F., Raza, A., Thalji, N., Abualigah, L., Zitar, R. A., & Jia, H. (2024). An efficient artificial intelligence approach for early detection of cross-site scripting attacks. *Decision Analytics Journal*, *11*(January), 100466. doi:10.1016/j.dajour.2024.100466

Chapter 4

Explainable AI Methods: The Example of SHAP and LIME

Bahaddin ERDEM[1]

ABSTRACT

Recently, artificial intelligence (AI) tools have become increasingly important in data analysis, and their applications are becoming increasingly widespread. While high performance has been achieved in analyses conducted using machine learning and deep learning models, their "black box" nature raises concerns about transparency, accountability, fairness, and trust. The field of Explainable AI (XAI) has emerged as a solution to the black box problem in AI-based analysis. XAI produces more transparent and accountable results for model decisions and predictions. This has fostered trust in AI-based data analysis, encouraging user adoption of these tools. XAI offers many methods for explaining and interpreting. This study examines only Shapley Additive Explanations (SHAP) and Local Interpretable Model-Independent Explanations (LIME), methods widely used in the literature, and supports them with experimental applications. SHAP is an XAI method based on strong game theory that attempts to increase interpretability by calculating the values of every feature that could affect the target variable or independent variable. LIME is one of the best-known methods for solving black-box problems. LIME approximates complex models and transfers the calculated examples to another locally interpretable model. This supports the probability of which class a feature belongs to in classification models. In the last part of the study, online exam data was visualized by using libraries in Python environment; both SHAP and LIME analysis were performed with XGBoost algorithm in binary classification analysis, and the positive and negative aspects of the features on the model and the degree to which they affect were analyzed.

Keywords: AI, XAI, SHAP, LIME, Black box

[1] Lecturer Dr.; Bitlis Eren University, Adilcevaz Vocational School, Department of Computer Programming, bahaddin2363@gmail.com, ORCID: 0000-0003-3693-0966

1. INTRODUCTION

The adventure of AI, began in the 1950s, and it has gained momentum and become widely used, especially in recent years, for three main reasons. Firstly, the accessibility of vast amounts of data generated by e-commerce platforms, social networks and businesses; secondly, the advancement of Machine Learning (ML) algorithms enabling them to deliver more reliable results; and thirdly, the availability of cloud technologies and high-performance computers at more affordable costs have accelerated this process. Today, AI continues to transform many areas, from individual life to social structure and commercial areas (Mondal, 2020:389). AI is essentially a computer system that emulates human cognition by processing data from various sources and systems, making decisions and learning from the resulting patterns. AI is also defined as the capacity of computers to recognize patterns in existing data and statistical models and take appropriate action (Hassani et al., 2020:145). Many AI tools, especially machine learning and deep learning-based models, have been developed to examine large-scale data sets, reveal hidden patterns in these data, and produce various solutions (Brozek, 2024:427, Hassija et. all., 2024:45). The critical role of AI in today's technological advancements is clearly evident in its widespread use. By analyzing large data sets and uncovering patterns, AI is boosting creativity and productivity in numerous sectors, including finance, healthcare, education, and entertainment. This demonstrates that AI plays a crucial role in shaping the future through the collaboration between human creativity and technological advancement. However, AI models also present challenges, as they obscure decision-making and prediction processes, raising concerns about transparency, trust, accountability, and explainability. Although AI offers high accuracy and efficiency, it is often considered a “black box” and is therefore subject to criticism, especially in complex structures such as deep learning and large language models (LLM) (Hsieh, 2024:7).

XAI, as a field of research, is focused on developing methods and models that will enable people to gain confidence and understanding of the workings of AI systems and how these systems relate to logic (Hassija et. all., 2024:51-52). The main goal of XAI is to develop models that can provide transparent, clear and understandable explanations for decisions taken or predictions produced. These models directly integrate interpretability into the learning process, strengthening engagement in accountability, trust, and transparency, and enabling people to validate AI outputs, better understand the results, and make sound decisions (Contreras ve Bocklitz, 2024:604). A significant drawback of most machine learning models is the lack of transparency in decision-making and prediction processes (Adadi and Berrada, 2018:52138). This behavior of the

models is often described as a "black box." Even experienced professionals face difficulties interpreting these complex models. When a model is difficult to understand or ambiguous, it becomes difficult to gain acceptance and build user trust. This leads people to distrust the model's decisions or predictions. XAI aims to provide tools and methods to help users and researchers interpret and understand the results of AI models.

XAI is a powerful tool that allows users to make sense of complex model outputs and visualize their results. Visualizing outputs and results facilitates developers' deep understanding of model decisions and increases understandability and confidence in predictive accuracy. Thus, XAI supports model adoption by providing effective outputs before, during, and after deep learning and machine learning predictions (Cifci, 2025:36293). The responsible use of AI is crucial for understanding how decision-making processes work and for the public's ability to gain trust in AI. The growing interest in explainable AI aims to foster trust by increasing the understandability and transparency of AI decision-making (Kalasampath et al., 2025:41112). By explaining the complex operating logic of AI algorithms, XAI provides insights into how predictions and interpretations are generated, thereby increasing end-user confidence in model decision processes and closing the understanding gap between models (Contreras ve Bocklitz, 2024:604 ve Infant et. al., 2025:1).

2. CONCEPTUAL FRAMEWORK

2.1. Basic concepts of XAI

Explainability: Central to the concept of explainability is the extent to which a machine learning model can be understood at its core. Explainability goes beyond interpretability by explaining "why" the model's decisions were made. Four fundamental principles stand out for XAI mechanisms:

1. **Explanation:** The system provides relevant evidence or justification for outputs and/or processes.
2. **Meaningfulness:** XAI system provides explanations that the intended users can understand.
3. **Explanation accuracy:** XAI system provides explanations that accurately represent the process of producing the output.
4. **Knowledge limits:** XAI systems only work under the conditions for which they were designed and when there is sufficient confidence in their output (Philips, 2021:2).

Transparency: Transparency is a fundamental element of all scientific research. Without transparency, the integrity and validity of research findings cannot be independently tested and verified. This highlights the importance of

transparency for the reliable use of evidence in decision-making processes (Sampson, et. al., 2019:1355).

Interpretability: By definition, it refers to the extent to which an individual can understand the reasons behind a decision. This process often involves translating complex model predictions into human-comprehensible insights. In short, explainability is the effort to make an explanation more easily understood (Erasmus, Brunet ve Fisher, 2020:849).

Justice: Justice refers to the ethical characteristics of an AI system that are unbiased, sensitive to diversity, and non-discriminatory. Descriptions of AI systems provide human-understandable interpretations of the system's internal workings and the decisions it makes (Zhou, Chen, & Holzinger, 2020). Justice in AI aims to develop methods to detect, reduce, and control biases in AI-supported decision-making processes (Schwartz, et., al., 2022:i77).

White Box: White box models are known as interpretable models in machine learning and offer transparency in decision-making processes. By providing inherent explainability, they allow us to understand the impact of input features on model output, thus providing valuable insights into underlying relationships and patterns (Nasarian vd., 2024:3). The concepts of understandable models and XAI are used to describe all machine learning models that produce results that experts in the application domain can easily interpret (Loyola-Gonzalez, 2019:154101). These models offer a balance between explainability, accuracy and confidence. The availability of larger data sets and the proliferation of computer-aided decision-making have increased the demand for interpretable models. The interpretability offered by white-box models allows all mechanisms, from users to regulators and developers, to evaluate the logic of the model, identify potential biases, and ensure fairness and accountability (Mumuni ve Mumuni, 2025:1). "White box" models are characterized by the easy-to-understand algorithms used, allowing a clear interpretation of how input features are transformed into output or target variables (Wiewiórowski, 2021:3). Examples of such models include linear regression, Bayesian Networks, Fuzzy Cognitive Maps, logistic regression, decision trees and rule-based systems.

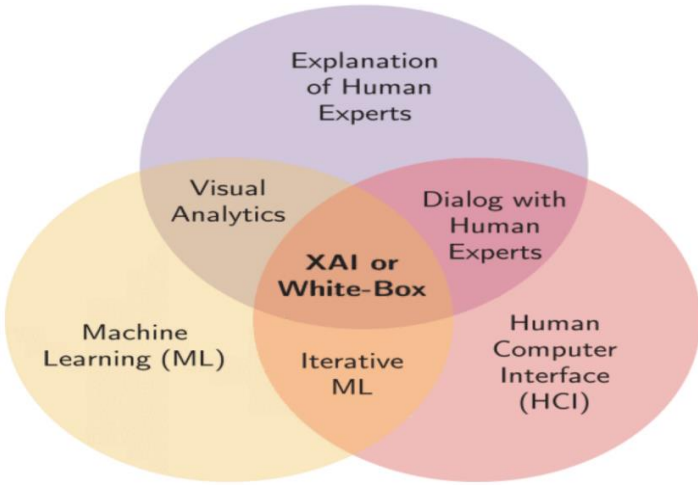


Figure 1. Interaction of different areas that make up the XAI and White-Box model (Loyola-Gonzalez, 2019:154102)

Figure 1 shows an explainable model resulting from interactions between machine learning, human-computer interfaces, explanations by human experts, visual analytics, iterative machine learning, and interactions between machine learning experts and human experts in the application domain.

Black Box: Black box methods operate on the assumption that there is no knowledge of the internal workings of the model. Therefore, for each input, only the final outputs produced by the model can be observed. In this approach, to explain a black box model, it is necessary to develop ways to query the outputs in a way that reveals the model's underlying behavior. However, these methods are generally slower than white-box approaches because knowledge is only gained by submitting additional queries to the model. In recent years, explainability methods have become increasingly important for providing insights into black-box machine learning methods such as deep neural networks. However, interpretability alone is insufficient to address all the problems of black-box models. Deep or shallow neural networks are among the most common examples of black-box models in machine learning (Holzinger et. all., 2020:260).

Gray Box: Gray-box models aim to strike a balance between both explainability and accuracy. Consequently, any data-driven learning algorithm, including white- and black-box models, can be considered a gray-box model (Ghasemi et. All., 2024:5). Gray box methods combine the interpretability advantages of White box methods with the high performance of black box methods. Research in this area focuses on improving AI methods to achieve

explanation goals without significantly compromising performance. The biggest advantage of gray-box approaches is their ability to combine understandability with high performance, especially in sensitive areas like medicine. However, only a limited number of application examples currently exist, and explainability in these methods is limited to certain elements. As with all methods of explanation, it is important to decide whether to provide an explanation specific to individual cases or a general one, and different explanation strategies should be applied accordingly. Therefore, additional research and development is needed to ensure that grey-box methods can be used effectively and specifically across a very large application domain. Only then will the full benefits of these methods be realized (Gallee et. All., 2023:800). An annotated comparison of the gray box, black box and white box models in the literature is given in Figure 2.

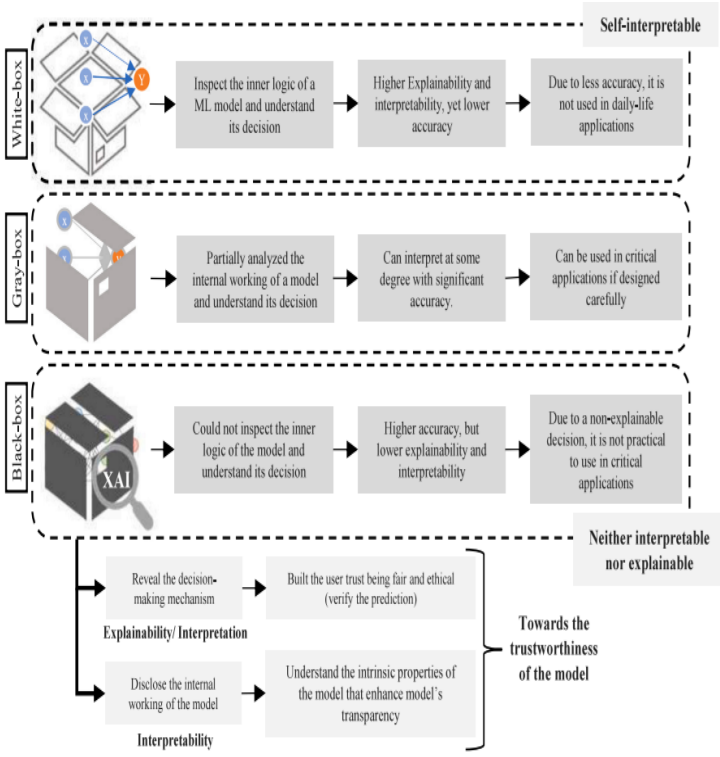


Figure 2. Comparison of gray box, black box and white box models (Ali et. All., 2023:3)

As shown in Figure 2, the concepts of gray box, black box and white box represent different levels of internal components of models. White box models, by design, offer higher interpretability; therefore, their output is easier to

understand, but their accuracy is generally lower. Gray box models provide a balance between interpretability and accuracy. Black box models, on the other hand, offer high accuracy but are limited in interpretability. The following list summarizes the advantages of providing a solution to black box systems (Ali et. All., 2023:3):

- To protect individuals from the negative effects of automated decision-making processes due to automatic decisions.
- To enable individuals to make more effective and conscious choices.
- To detect and prevent vulnerabilities resulting from security problems.
- Developing algorithms that are compatible with human values.
- To increase business and user trust by establishing user standards in the development of artificial intelligence-based products.
- To implement Right to Explanation policies.

2.1. The Black Box Problem in AI

Although inputs and outputs are known in AI models, it is often difficult to determine exactly how inputs are transformed into outputs (Pavlidis, 2024). The automation of routine decisions, coupled with the complex information architectures that enable this automation, raises concerns about system reliability. These concerns are particularly pronounced in the deep learning (DL)-based AI class, which utilizes algorithmic systems comprised of deep neural networks and are difficult for humans to understand. These types of problems are often called “black box problems” in AI (Bearman ve Ajjawi, 2023:1163). Observers can trace the inputs and outputs of these complex, nonlinear processes, but they cannot directly see the internal workings of the system. The mechanisms by which AI reaches its conclusions are often obscure or invisible. Without understanding this mechanism, the question of how trustworthy these systems can be remains unanswered. The increasing delegation of decision-making authority to AI to protect critical human values such as security, health, and safety makes the issue of trust even more crucial. In response to this problem, models that “open the black box” that make non-linear and complex decision processes understandable to human observers are being developed and technical solutions are being sought. This class of models, called XAI, offers promising solutions to the black box problem, but in their current form they make these processes only partially understandable to many observers. (Von Eschenbach, 2021:1608).

2.2. XAI methods

LIME: Locally interpretable model-independent explanations (LIME) are one of the most used interpretability techniques for black-box models and black-box problems. Following a powerful yet simple approach, LIME can produce meaningful interpretations and results even when the classifier makes any prediction. The target model is then run on this new data to generate predictions, which are weighted according to their closeness to the input sample. In the final stage, a simple and interpretable model such as a decision tree is trained on the dataset to ensure interpretability of the results. (Linardatos, Papastefanopoulos ve Kotsiantis, 2020:11). While machine learning models are considered black-box functions, model independent explanation methods only provide access to the model's output. These methods, which are extremely flexible and applicable to various applications, do not require any knowledge about the internal structure of the model. (Holzinger et. all., 2020:15). LIME has many successful applications in various fields, demonstrating its popularity as a model-independent method. However, its explanations are limited because they are implicitly based on surrogate models; the quality of the explanation depends on the accuracy of the surrogate fit. Surrogate fits typically require extensive sampling, increasing computational cost, and the sampling process can introduce uncertainties, leading to different explanations for the same input (Holzinger et. all., 2020:16). LIME is another XAI method that aims to explain the local operating logic of a model on a given instance. In this direction, it approximates any complex model and transforms it into a locally interpretable model for a given instance (Ribeiro, Singh, Guestrin. 2017 as cited in Salih et. All., 2025:2). LIME is a model-independent local annotation method that reveals the impact of each feature on the outcome of a single instance. In classification models, it displays the probability that an instance belongs to a particular class and presents the contribution of each feature to that class in visualized graphs. However, because LIME transforms any complex model into a linear local model, it reports coefficients representing the weights of the features in the model. This can lead to the loss of important information and incomplete explanation in models containing nonlinearity, because the nonlinear relationships cannot be reflected in the surrogate model. Additionally, LIME is a model-dependent method; that is, the explanations produced by LIME may vary when different models are used on the same task and dataset (Ribeiro, Singh, Guestrin. 2017, as cited in Salih et. All., 2025:2).

SHAP: Shapley Additive Explanations (SHAP) is a powerful explanation method inspired by game theory that aims to increase interpretability by calculating the importance values of each feature in separate predictions

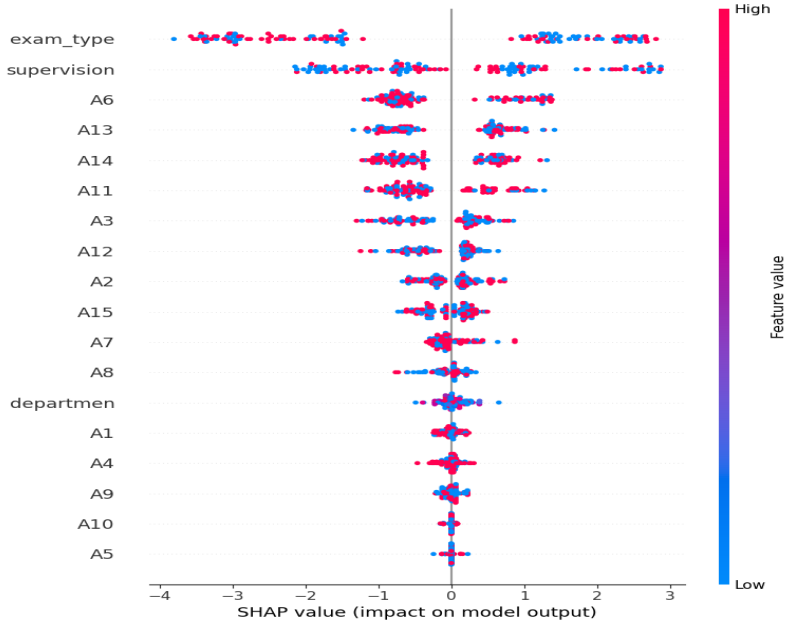
(Holzinger et al., 2020:16). This approach can be applied to any machine learning model, regardless of the model. In SHAP, the "actors" in machine learning models are considered features, and the "payoff" is considered the model output. The method calculates an importance score representing the contribution of each feature to the model output. This score is determined by evaluating all possible combinations of features; that is, all scenarios where both all features and subsets of features are used in the model are considered. Because computational complexity increases with the number of features, the Kernel SHAP approach was developed as a solution. SHAP offers a powerful method for explaining any model by treating each feature as a player and the model result as a payoff. SHAP provides global and local annotations, meaning it is capable of explaining the role of features for all instances and for a specific instance (Lundberg ve Lee., 2017 as cited in Salih et. All., 2025:2, Band et. All., 2023:4). One of the most significant drawbacks of Shapley values is their high computational complexity. Especially for deep neural networks and modern models with high-dimensional inputs, Shapley values are quite difficult to calculate precisely (Band et al., 2023:4; Holzinger et al., 2020:16; Salih et al., 2025:3). There are several critical points users should be aware of when applying the SHAP method. First, SHAP is a model-dependent method; that is, the explanation results obtained depend on the machine learning model used. This can lead to variability in explainability scores when different models are used on the same data and task. In this context, when different machine learning models are applied to the same task on the same dataset, the most important features identified by SHAP may differ between models. There are several important considerations for end users using SHAP. First, SHAP is a model-dependent method, meaning its explanation results depend on the machine learning model used in the classification or regression task. This may cause explainability scores to vary when different models are applied to the same data and task (Salih et. all., 2025:3).

While SHAP evaluates different feature combinations to calculate feature attributions, LIME is based on a local surrogate model. Furthermore, SHAP is capable of providing both global and local level explanations, while LIME is limited to local explanations only. While SHAP can detect nonlinear relationships depending on the model used, LIME may be limited in capturing such complex relationships because it creates a locally linear model. In terms of visualization, SHAP produces a variety of graphs that present both local and global annotations, while LIME provides a separate visualization for each instance. Finally, LIME is significantly faster than SHAP, especially for tree-based models (Lundberg ve Lee., 2017, as cited in Salih et. all., 2025:2).

3. EXPERIMENTAL STUDIES

The dataset used in this study is online exam data created by Erdem and Karabatak (2025:9). The dataset consists of multiple-choice questions and student responses within a course. It also contains data from 40 attributes and 162 students.

As part of the sample application, a cheating detection study was conducted using data from an online exam. In this study, a binary classification analysis was performed using the XGBoost algorithm in Python to distinguish between "cheated" and "not cheated" classes. The success rates achieved are quite high, with remarkable accuracy (0.969), precision (0.958), and F1 score (0.929). However, despite this high success, detailed information about the features that affect the model's decision-making process is not directly accessible. Explanatory AI methods provide interpretable results by revealing the positive and negative effects of independent variables on the model. In this context, SHAP and LIME analyses were applied in the study, aiming to gain a deeper understanding of the model outputs.



Graph 1. XGBoost descriptive SHAP summary analysis result

SHAP analysis values for the XGBoost model are given in Graph 1.

- Horizontal axis (SHAP value): Shows the effect of features on model prediction.

- Positive values: Contribute to shifting the prediction toward the "cheated" class.
- Negative values: Contribute to shifting the prediction toward the "did not cheat."
- Colors: Indicate the magnitude of feature values (blue = low value, red = high value).

The direction and extent to which the features in the data set affect the “cheated” and “did not cheat” classes are given below:

✓ exam_type

- It is one of the most important variables.
- Most of the red dots are on the positive side; some exam types seem to direct students to the "cheated" classification.
- The concentration of blue dots on the negative side indicates that other exam types are more likely to support the "did not cheat" classification.

✓ supervision

- It is a second-order variable.
- When the level of supervision is low (blue), the SHAP value is positive; that is, if supervision is low, the probability of the student being assigned to the "cheated" class increases.
- When the level of supervision is high (red), the SHAP value is negative; that is, if supervision is tight, the probability of the student being assigned to the "did not cheat" class increases.

✓ Variables such as A6, A13, A14, A11

- It is of moderate significance to the model.
- It has both positive and negative effects, indicating that different values of the features can increase or decrease the likelihood of cheating.
- In particular, in variables A6 and A13, the red dots are on the positive side; higher values support the "cheated" classification.

✓ Variables such as A2, A15, A7, A8

- They have a similar effect, but slightly lower in importance.
- They contribute differently to both the cheated and non-cheated classes.

✓ departmen

- It is a variable with a lower impact level.
- The fact that red dots are generally on the positive side of the SHAPE suggests that in some sections, students are more likely to be classified as "cheating."
- Blue values, on the other hand, support the "not cheating" classification.

As a result, the strongest predictors in the model are "exam_type" and "supervision." Under certain exam types and low supervision conditions, a strong contribution is made to the "cheated" class. However, when considering high supervision and exam types, a strong contribution is made to the "did not cheat" class.

Figure 3 shows the LIME analysis values for the XGBoost model.

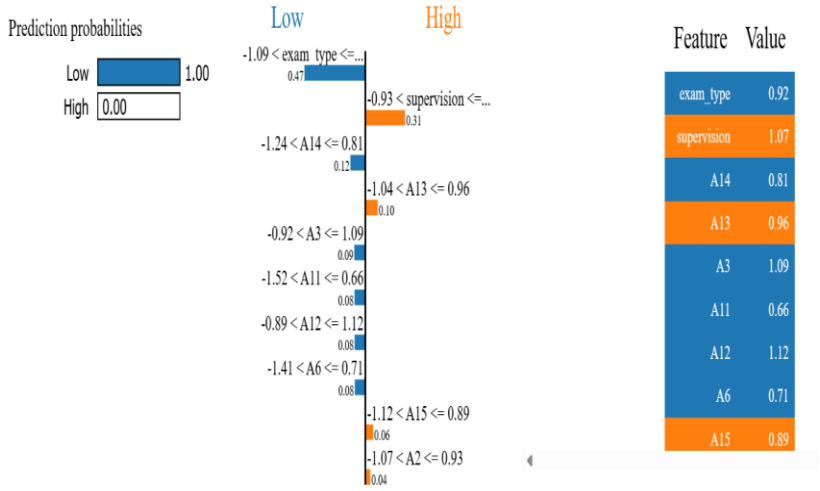


Figure 3. LIME analysis result for XGBoost model

According to Figure 3;

✓ Prediction probabilities (box on the left):

- For this example, the model chose the "Low" class (did not cheat) with 100% probability.
- For the "High" class (did cheat), the probability is 0.00, meaning the model's decision is very clear.

✓ Middle part (contribution of features to the decision):

- Blue bars support the "did not cheat" classification.
- Orange bars support the "cheated" classification.
- The length of the bars indicates the contributing power of the feature on the decision.

✓ Table on the right (Feature – Value):

- It shows the values that the relevant student/sample received for these variables.

The direction and extent to which the features in the data set affect the “cheated” and “did not cheat” classes are given below:

✓ exam_type (0.92)

- It is the variable with the strongest effect.

- On the blue side, this exam type provides strong support for the student being in the "did not cheat" class.

✓ supervision (1.07)

- On the orange side, the probation conditions move this student slightly toward the "cheated" class. But the effect is not as strong as "exam_type," so the decision remains unchanged.

✓ Variables such as A14 (0.81), A3 (1.09), A11 (0.66), A12 (1.12), A6 (0.71)

- All are on the blue side; these variables contribute to the student being in the "did not cheat" class.
- A3 and A12 are particularly strongly on the blue side.

✓ Variables such as A13 (0.96), A15 (0.89), A2 (0.93)

- On the orange side, although the current values of these variables direct the student to the “cheated” class, their contribution is relatively weaker.

4. RESULTS AND DISCUSSION

The emergence of AI tools has achieved great success in predicting system stability in important areas such as healthcare, finance, and education. This study examines XAI methods as solutions to the black-box problems of deep learning and machine learning models. XAI contributes to the field by comparing studies in the literature within the framework of transparency, trust, fairness, interpretability, and understandability criteria. SHAP and LIME techniques, which are widely used in the literature, are compared with all their features and the differences between them are stated. In addition, it has been observed that the features that can be effective in the model's decision, with the example applications, offer solutions to understandability by showing in what direction and to what extent they affect the model.

REFERENCES

Adadi, A., & Berrada, M. (2018). Peeking inside the black-box: a survey on explainable AI (XAI). *IEEE access*, *6*, 52138-52160.

Ali, S., Abuhmed, T., El-Sappagh, S., Muhammad, K., Alonso-Moral, J. M., Confalonieri, R., ... & Herrera, F. (2023). Explainable AI (XAI): What we know and what is left to attain Trustworthy AI. *Information fusion*, *99*, 101805

Band, S. S., Yarahmadi, A., Hsu, C. C., Biyari, M., Sookhak, M., Ameri, R., ... & Liang, H. W. (2023). Application of explainable AI in medical health: A systematic review of interpretability methods. *Informatics in Medicine Unlocked*, *40*, 101286.

Bearman, M., ve Ajjawi, R. (2023). Learning to work with the black box: Pedagogy for a world with AI. *British Journal of Educational Technology*, *54*(5), 1160-1173.

Brożek, B., Furman, M., Jakubiec, M., ve Kucharzyk, B. (2024). The black box problem revisited. Real and imaginary challenges for automated legal decision making. *AI and Law*, *32*(2), 427-440.

Contreras, J., ve Bocklitz, T. (2025). Explainable AI for spectroscopy data: a review. *Pflügers Archiv-European Journal of Physiology*, *477*(4), 603-615.

Erdem, B., & Karabatak, M. (2025). Cheating detection in online exams using deep learning and machine learning. *Applied Sciences*, *15*(1), 400.

Gallee, L., Kniesel, H., Ropinski, T., & Goetz, M. (2023, September). AI in radiology–beyond the black box. In *RöFo-Fortschritte auf dem Gebiet der Röntgenstrahlen und der bildgebenden Verfahren* (Vol. 195, No. 09, pp. 797-803). Georg Thieme Verlag KG.

Ghasemi, A., Hashtarkhani, S., Schwartz, D. L., & Shaban-Nejad, A. (2024). Explainable AI in breast cancer detection and risk prediction: A systematic scoping review. *Cancer Innovation*, *3*(5), e136.

Hassani, H., Silva, E. S., Unger, S., TajMazinani, M., ve Mac Feely, S. (2020). AI (AI) or intelligence augmentation (IA): what is the future?. *Ai*, *1*(2), 8.

Hassija, V., Chamola, V., Mahapatra, A., Singal, A., Goel, D., Huang, K., ... ve Hussain, A. (2024). Interpreting black-box models: a review on explainable AI. *Cognitive Computation*, *16*(1), 45-74.

Holzinger, A., Goebel, R., Fong, R., Moon, T., Müller, K. R., & Samek, W. (2020, July). xxAI-beyond explainable AI. In *International Workshop on Extending Explainable AI Beyond Deep Models and Classifiers* (pp. 3-10). Cham: Springer International Publishing.

Hsieh, W., Bi, Z., Jiang, C., Liu, J., Peng, B., Zhang, S., ... ve Liang, C. X. (2024). A comprehensive guide to explainable ai: From classical models to llms. *arXiv preprint arXiv:2412.00800*.

Infant, S. S., Vickram, S., Saravanan, A., Muthu, C. M., ve Yuarajan, D. (2025). Explainable AI for sustainable urban water systems engineering. *Results in Engineering*, *25*, 104349

Kalasampath, K., Spoorthi, K. N., Sajeev, S., Kuppa, S. S., Ajay, K., ve Angulakshmi, M. (2025). A Literature review on applications of explainable AI (XAI). *IEEE Access*.

Linardatos, P., Papastefanopoulos, V., & Kotsiantis, S. (2020). Explainable ai: A review of machine learning interpretability methods. *Entropy*, *23*(1), 18.

Loyola-Gonzalez, O. (2019). Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view. *IEEE access*, *7*, 154096-154113.

Mondal, B. (2020). AI: State of the Art. In: Balas, V., Kumar, R., Srivastava, R. (eds) Recent Trends and Advances in AI and Internet of Things. Intelligent Systems Reference Library, vol 172. 389-425. Springer, Cham. https://doi.org/10.1007/978-3-030-32644-9_32.

Mumuni, F., & Mumuni, A. (2025). Explainable AI (XAI): from inherent explainability to large language models. *arXiv preprint arXiv:2501.09967*.

Nasarian, E., Alizadehsani, R., Acharya, U. R., & Tsui, K. L. (2024). Designing interpretable ML system to enhance trust in healthcare: A systematic review to proposed responsible clinician-AI-collaboration framework. *Information Fusion*, *108*, 102412.

Phillips, P. J., Phillips, P. J., Hahn, C. A., Fontana, P. C., Yates, A. N., Greene, K., ... ve Przybocki, M. A. (2021). Four principles of explainable AI.

Salih, A. M., Raisi-Estabragh, Z., Galazzo, I. B., Radeva, P., Petersen, S. E., Lekadir, K., & Menegaz, G. (2025). A perspective on explainable AI methods: SHAP and LIME. *Advanced Intelligent Systems*, *7*(1), 2400304.

Sampson, C. J., Arnold, R., Bryan, S., Clarke, P., Ekins, S., Hatswell, A., ... & Wrightson, T. (2019). Transparency in decision modelling: what, why, who and how?. *Pharmacoeconomics*, *37*(11), 1355-1369.

Schwartz, R., Schwartz, R., Vassilev, A., Greene, K., Perine, L., Burt, A., ve Hall, P. (2022). *Towards a standard for identifying and managing bias in AI* (Vol. 3, p. 00). Gaithersburg, MD: US Department of Commerce, National Institute of Standards and Technology.

Von Eschenbach, W. J. (2021). Transparency and the black box problem: Why we do not trust AI. *Philosophy & technology*, *34*(4), 1607-1622.

Wiewiórowski, W. European Data Protection Supervisor. *URL: https://edps. europa. eu/about-edps/members-mission/supervisors/wojciechwiewi% C3% B3rowski_en (last accessed: 18.09. 2025)*.

Chapter 5

Applied TinyML for Embedded Intelligence: A Real-Time HAR Implementation on Arduino Nano 33 BLE Sense

İrfan ÖKTEN[1]

ABSTRACT

This book chapter presents an in-depth examination of Tiny Machine Learning (TinyML) and its transformative role in enabling embedded intelligence on resource-constrained microcontroller-based systems. TinyML brings artificial intelligence from cloud-centered infrastructures to ultra-low-power edge devices, offering real-time inference, enhanced privacy, reduced bandwidth requirements, and significant energy savings. The chapter begins by outlining the conceptual foundations of TinyML, including the characteristics of embedded systems, the principles of edge AI, and the unique workflow required to deploy machine learning models on devices with kilobytes of RAM and milliwatt-level power budgets. Essential model optimization strategies—such as quantization, pruning, and knowledge distillation—are analyzed to highlight their importance for achieving feasible and efficient inference on restricted hardware. The chapter further explores the software ecosystem supporting TinyML, with detailed discussion of frameworks such as TensorFlow Lite Micro, Edge Impulse, and MicroTVM, emphasizing their roles in data acquisition, model development, and on-device deployment. The experimental component features a real-time Human Activity Recognition (HAR) implementation on the Arduino Nano 33 BLE Sense, employing a lightweight 1D CNN model trained on accelerometer data. Through INT8 post-training quantization, the model achieves a 75% reduction in memory size, a 2.4× improvement in inference speed, and a 59% reduction in energy consumption, while maintaining accuracy with only minimal degradation. These results validate the practical viability of TinyML for real-world embedded applications where efficiency and responsiveness are paramount. Finally, the chapter identifies and discusses major research challenges—including hardware heterogeneity, compiler limitations, security vulnerabilities, resource-aware

[1] Assist. Prof. ; Bitlis Eren University Faculty of Engineering and Architecture, Department of Computer Engineering, iokten@beu.edu.tr, ORCID: 0000-0001-9898-7859

optimization, and the need for on-device and continual learning. Emerging trends such as neuromorphic computing, processing-in-memory (PIM), energy-harvesting autonomous AI systems, and integration within 6G-enabled IoT infrastructures are explored as key opportunities shaping the future direction of the field. Overall, the chapter provides a comprehensive framework for understanding both the current landscape and future evolution of TinyML-driven embedded intelligence.

Keywords: TinyML, Embedded intelligence, Model compression, Real-Time human activity recognition

1. INTRODUCTION

In recent years, the trajectory of artificial intelligence (AI) has increasingly shifted from centralized, cloud-based infrastructures toward decentralized, on-device intelligence. Traditional machine learning (ML) applications have primarily relied on large-scale computing resources, heavy memory and storage capacities, and consistent connectivity to remote servers. However, today's technological landscape demands more ubiquitous, energy-efficient, and latency-sensitive AI solutions. In response, the field of TinyML (Tiny Machine Learning) has emerged as a compelling paradigm: it focuses on executing ML models directly on severely resource-constrained microcontrollers (MCUs) and embedded systems. As articulated by Soro and Banbury, TinyML offers ultra-low power consumption, real-time processing at the data source, and reduced dependency on cloud connectivity (Soro, 2021; Banbury et al., 2020).

Embedded within this paradigm shift is the broader concept of “embedded intelligence” — systems that not only collect data passively, but also make autonomous decisions locally. As Yelchuri and R. note, TinyML is redefining this notion: devices evolve from simple sensors or data-loggers into active intelligent agents, capable of perception, inference and adaptation at the edge (Yelchuri & R., 2022). This transition brings about several strategic advantages: lower network latency (since less data must be transmitted), enhanced data privacy (since raw data remains on the device), and favorable environmental implications (thanks to lower energy consumption and less reliance on data-centres).

The significance of TinyML becomes particularly salient in domains such as the Industrial Internet of Things (IIoT), wearable health-monitoring, smart city deployments and pervasive sensing networks. In such contexts, devices operate under strict constraints in power budget, memory size, computational throughput and communication bandwidth. The TinyML paradigm addresses these constraints head-on by leveraging model compression, hardware-aware optimisations and co-design of algorithms with embedded platforms. Recent surveys highlight the maturity of the tool-chains and frameworks supporting TinyML (Kreß et al. 2024; Loh & Guo 2025; Wilson & Singh 2025).

Despite its rapid growth and considerable promise, TinyML remains a field rife with research challenges and unresolved questions. On the one hand, the requirement to deploy ML inference (and eventually training) on devices with a few kilobytes of RAM, minimal flash storage and limited power means that novel optimisation methods (quantisation, pruning, efficient model architectures, neural-architecture search) must be developed and tailored to the embedded domain. On the other hand, system-level issues such as heterogeneity

of hardware platforms, tool-chain fragmentation, lifecycle management, security and privacy at the edge continue to hinder wide adoption. Recent work emphasises the need for holistic co-design approaches, benchmarking frameworks and lifecycle automation to drive TinyML beyond prototyping (Maldonado et al. 2025; Reddi et al. 2022).

In this book chapter, we present a rigorous and comprehensive examination of TinyML within the context of embedded intelligence. We begin by outlining the fundamental concepts, typical system architecture and model-optimization techniques that make TinyML feasible on resource-limited devices. We then survey the ecosystem of supporting tools and frameworks, and map out the major application domains in which TinyML has already demonstrated impact. Following that, we delve into the key challenges—both technical and systemic—that currently impede broader deployment, and highlight ongoing research directions and future opportunities that can propel TinyML to its full potential. Our aim is to equip readers with both the theoretical foundation and practical insight needed to appreciate, design and evaluate TinyML systems within embedded contexts.

They introduce deep neural network models that classify movements such as walking, running, and squatting using IMU data collected with the Arduino Nano 33 BLE Sense. The authors compare architectures such as MLP, CNN-LSTM, and CNN-GRU, reporting the best accuracy. They then demonstrate the practicality of on-device inference, memory, and power savings by compressing the models and running them on the same board (Kumari et al. 2024).

Lipski investigates hand gesture recognition using photodiode data on the Arduino Nano 33 BLE (directly related to the Nano 33 BLE Sense); different RNN-based and CNN-LSTM architectures are tested; and real-time classification challenges on an embedded device are discussed. This paper details practical experiences and limitations, particularly regarding MCU constraints (TensorFlow Lite for Microcontrollers support, model sizes, and latency estimation) (Lipski, 2022).

They focus on designing lightweight yet efficient models like DeepConv-LSTM and deploying them to edge devices using TinyML toolchains; the authors report that the best model delivers both high accuracy and low latency. The paper details the deployment of the best model via Edge Impulse on an Arduino Nano 33 BLE Sense Rev2, with positive post-quantization size/power/latency measurements (Zhou et al. 2025).

2. FUNDAMENTALS OF TINYML AND EMBEDDED INTELLIGENCE

In this section, the concepts and architectural details underlying TinyML will be discussed from an academic perspective.

2.1. Overview of Embedded Systems and Edge AI

Embedded systems are computing units designed around microprocessors or microcontrollers (MCUs), typically task-oriented, with real-time constraints. In the context of TinyML, these systems are typically considered extreme edge devices. Unlike traditional cloud-based artificial intelligence (AI) models, the MCUs targeted by TinyML typically have constraints ranging from 256 KB to 1 MB of Flash memory, 8 KB to 512 KB of SRAM, and power consumption on the mW level (Banbury et al., 2020). These constraints mandate hardware-software co-design in system design. Edge AI emerged in response to the latency, bandwidth costs, and data privacy concerns brought about by cloud computing. Edge AI: While inference requires high processing power on server-level or more powerful embedded systems (Single Board Computers (SBCs), TinyML focuses on the most energy- and memory-constrained devices at the lower end of the spectrum. This distinction forms the core philosophy of TinyML: maximum inference efficiency with minimum energy consumption. Embedded intelligence describes the evolution of these devices from passive data collectors to local and autonomous decision makers. This transformation is especially critical in applications such as real-time anomaly detection, continuous monitoring, and local speech recognition (Yelchuri & R., 2022).

2.2. TinyML Architecture and Workflow

The TinyML workflow, unlike the traditional ML pipeline, includes an additional optimization phase focused on deployment in a resource-constrained environment. This pipeline represents the intersection of scientific and engineering disciplines.

Model Training and Optimization:

Model training is typically performed on cloud servers or powerful workstations. However, compactness is a priority in the model architecture selection for TinyML. For example, instead of standard convolutional networks (CNNs), architectures like MobileNet or EfficientNet, which use depthwise separable convolutions (DEP), which significantly reduce the number of parameters and operations, are preferred (Lin et al., 2023).

The optimization phase is the heart of TinyML:

- Quantization: This reduces model size by a factor of four and increases inference speed by downsizing floating-point (FP32) weights and activations to a low bit depth (typically INT8). However, maintaining model accuracy after quantization requires techniques such as Quantization-Aware Training (QAT). QAT simulates quantization effects during training, making the model more resilient.
- Pruning: Pruning addresses over-parameterization in ML models. Structured Pruning cleans up the network architecture by removing entire filters or neurons, while Unstructured Pruning resets individual weights. For embedded devices, structured pruning is more advantageous because it saves inference time.

Deployment:

The optimized model is converted to a target hardware-specific programming language (typically C/C++) and embedded into the target MCU along with embedded inference engines. TensorFlow Lite Micro (TFLM) is a critical inference engine that optimizes core functions and memory allocation strategies for the MCU's limited architecture. Additionally, the use of hardware-specific libraries such as CMSIS-NN, a library optimized for ARM Cortex-M series processors, maximizes inference speed and energy efficiency (Tosun & Erdem 2024).

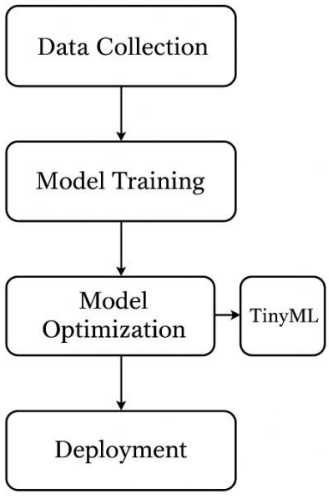


Figure 1. General workflow of TinyML development

Figure 1 illustrates the overall workflow of the TinyML development process, encompassing the stages from data collection to deployment on embedded microcontrollers. The process begins with the acquisition and preprocessing of sensor data, which serves as the foundation for model training performed on high-performance computing systems. Once a baseline model is obtained, optimization techniques such as quantization and pruning are applied to reduce memory footprint and computational complexity, enabling efficient execution on resource-constrained hardware. The optimized model is then converted and deployed onto embedded devices, where real-time inference takes place locally. This pipeline exemplifies the integration of machine learning with embedded intelligence, ensuring low latency, enhanced privacy, and energy-efficient autonomous operation at the network edge.

2.3. Core Techniques for Model Compression

TinyML's fundamental viability hinges on its ability to radically reduce the memory and computational cost of machine learning models with acceptable accuracy loss. This section details the scientific background of the underlying compression techniques.

2.3.1. Quantization

Quantization is the process of reducing the precision of model parameters and calculations. Scientifically, this is the mapping of a floating-point number (32 bits) to an integer representation (8 bits or less).

$$Q = round(\frac{R}{S} + Z) \quad (1)$$

Here, R is the original floating-point value, S is the scale factor, Z is the zero point, and Q is the quantized integer value. Quantization not only reduces memory usage (the size is reduced by a factor of 4), but also increases inference speed because integer arithmetic requires fewer cycles and energy than floating-point operations.

Quantization Types:

- Post-Training Quantization (PTQ): This is applied after model training is complete. It is fast but carries a high risk of accuracy loss. This risk is mitigated by using a calibration dataset.
- Quantization-Aware Training (QAT): Quantization simulation is included in the training cycle. This typically provides the highest performance by allowing the model to learn more robustly against quantization-induced information loss, but training time is longer.

2.3.2. Pruning

Pruning reduces the number of parameters and FLOPs (Floating Point Operations) by increasing the sparsity of the model.

Structured Pruning vs. Unstructured Pruning:

- Unstructured Pruning: Sets the least important individual weights in the network to zero. This offers the highest compression ratio, but requires specialized hardware or compressed formats (sparse matrix format) and does not provide speedups on standard MCUs.
- Structured Pruning: Removes all neurons, filters, or layers. This changes the model's architecture but provides significant speedups on CPUs/MCUs during inference because the matrix multiplication dimensions in the network are directly reduced.

Pruning is typically implemented using Iterative Pruning methods: the model is pruned, the remaining weights are retrained (fine-tuned), and this cycle is repeated until the target sparsity ratio is reached.

2.3.3. Knowledge Distillation

This technique involves training the Student model by using the probability distributions (soft labels or "soft targets") generated by a large model (Teacher) as an additional loss function for a small model (Student). The Student learns not only the hard labels but also the relationships between the class probabilities of the Teacher model (Wang & Yoon 2022).

$$L_{Distillation} = \alpha L_{Soft} + (1 - \alpha) L_{Hard} \quad (2)$$

Here, L_{Soft} is a Loss function that allows the Student model to mimic the smooth outputs of the Teacher model. L_{Hard} is the standard cross-entropy loss. The constant α determines the relative importance of these two losses. This allows the small Student model to operate with minimal computational overhead while absorbing a large portion of the Teacher model's information power.

3. TOOLS, FRAMEWORKS, AND TYPICAL APPLICATIONS

3.1. Major Frameworks for TinyML

The TinyML development environment focuses on finding the balance between ease of use, rapid prototyping, and hardware support.

- TensorFlow Lite Micro (TFLM): Developed by Google, TFLM can run with around 16 KB of Flash and a few KB of RAM. Written in C++, TFLM supports efficient integer computations primarily through low-level C kernels. TFLM's architecture uses a dedicated memory manager that ensures kernels are

loaded onto the device only when needed. This minimalist approach makes TFLM indispensable for MCUs.

- Edge Impulse: As a highly integrated platform, Edge Impulse offers developers an end-to-end solution for data acquisition, model training, optimization, and deployment. In particular, supporting various sensor types (accelerometer, microphone, camera) and facilitating data flow through tools like Data Forwarder significantly accelerates the prototyping process.
- Apache TVM and MicroTVM: TVM, an open-source machine learning compiler framework, stands out for its ability to generate optimized code for various hardware. By extending this capability to MCUs, MicroTVM allows developers to deeply optimize model inference code for specialized hardware architectures. This is a critical research topic for maximizing the synergy between hardware architecture and software optimization.

3.2. Typical Application Areas

TinyML applications are characterized by the need to reduce the cycle time and energy costs between sensing and inference.

- Industrial Predictive Maintenance: Local analysis of data from vibration and acoustic sensors enables early detection of machine malfunctions. By running anomaly detection models on the device, TinyML sends alerts over the network only when anomalies are detected. This can reduce bandwidth and energy consumption by up to 1000 times.
- Healthcare and Wearable Devices: Continuous vital sign monitoring (heart rate, oxygen saturation) or activity recognition (fall detection) models provide instant alerts while preserving privacy (Kahya & Aslan, 2024). TinyML allows these devices to operate in an always-on but ultra-low-power mode.
- Zero-Power AI: Research is accelerating to run TinyML models on battery-free devices powered by energy harvesting (solar, vibration, RF). This is particularly revolutionary for remote and isolated environmental monitoring applications.

3.3. Impact on Embedded Intelligence

TinyML has permanently changed the architecture and philosophy of embedded systems. The fundamental paradigm of embedded intelligence now revolves not only around efficiency but also cognitive autonomy.

- Breaking Data Silos: TinyML eliminates the need to collect data in a centralized storage unit. This supports the transition to local data processing and

distributed intelligence architectures. This approach not only enhances privacy but also increases system resilience against global network failures.

- Environmental Sustainability: While the energy footprint of cloud computing centers is constantly increasing, TinyML's ultra-low power consumption helps the Internet of Things (IoT) achieve its green computing goals. Delivering AI capabilities at the milliwatt level is a key factor for an energy-sustainable digital future (Abadade et al., 2023).

4. EXPERIMENTAL IMPLEMENTATION AND RESULTS

To further substantiate the applicability of TinyML methodologies in real-world embedded intelligence tasks, an extended experimental implementation was conducted focusing on real-time Human Activity Recognition (HAR). The primary aim was to investigate how model compression strategies—particularly post-training quantization—affect on-device latency, memory utilization, and energy efficiency under stringent hardware constraints.

4.1. Experimental Setup

The experimental platform consisted of the Arduino Nano 33 BLE Sense, a representative low-power microcontroller board frequently used in TinyML research. The device features a 64-MHz ARM Cortex-M4F processor with 256 KB SRAM and 1 MB Flash, making it suitable for evaluating memory-sensitive inference tasks. The board’s built-in 3-axis accelerometer (sampling at 50 Hz) served as the sole sensor input.

A custom HAR dataset was collected with three activity classes—walking, running, and standing—each recorded for 10 minutes. The raw accelerometer readings were pre-processed using a 100-sample sliding window with 50% overlap, producing fixed-size feature segments suitable for lightweight time-series modeling.

Model development was carried out in TensorFlow using a compact 1-D CNN architecture composed of:

- Conv1D layer: 16 filters, kernel size = 3
- Conv1D layer: 32 filters, kernel size = 3
- Dense layer: 32 units
- Softmax output: 3 classes

Training achieved 94.7% accuracy on the validation set. To enable microcontroller deployment, the model underwent INT8 post-training quantization with TensorFlow Lite, resulting in a significant memory footprint reduction. Deployment was performed via TensorFlow Lite Micro (TFLM) using the Arduino IDE.

For power profiling, a Nordic Power Profiler Kit (PPK2) was connected inline with the Arduino board to capture instantaneous current draw during inference, enabling precise computation of energy per inference.

4.2. Performance Evaluation

Table 1. Performance evaluation of the system

Metric	Float32 Model	INT8 Quantized Model	Improvement
Model Size	236 KB	59 KB	−75 %
Inference Time (ms) @64 MHz	14.2	5.8	≈ 2.4× faster
Peak RAM Usage (KB)	98	42	−57 %
Classification Accuracy	94.7 %	93.8 %	−0.9 % loss
Energy Consumption (mJ/inference)	0.62	0.25	−59 %

The results demonstrate that INT8 quantization yields substantial improvements in every resource-sensitive metric. The compressed model fits comfortably within the MCU's memory limits while accelerating inference by a factor of 2.4×. The marginal 0.9% accuracy reduction illustrates the robustness of quantization for time-series classification tasks. The effects of INT8 quantization on model size, latency, memory usage, and energy consumption are presented in detail in Table 1.

Energy measurements from the PPK2 show a 59% reduction in per-inference energy cost, confirming the advantages of running compressed neural networks locally on embedded devices.

4.3. Discussion

The experiment validates the theoretical claims of TinyML: through quantization and hardware-aware optimization, ML inference can be performed efficiently on microcontrollers. The observed trade-off between model compactness and accuracy remains manageable, particularly for classification tasks tolerant of small accuracy loss. Furthermore, edge-based inference eliminates the need for continuous wireless transmission, providing an estimated 70–80 % reduction in total system energy consumption during operation.

Future enhancements could include exploring structured pruning and quantization-aware training (QAT) to further optimize accuracy-efficiency balance. Integrating energy-harvesting circuits may also extend operational lifetime toward battery-free TinyML scenarios.

5. CHALLENGES AND RESEARCH ISSUES

While the TinyML paradigm brings embedded intelligence to constrained devices, it also brings with it significant scientific and engineering challenges. These challenges constitute the focus of academic research in the field.

5.1. The Resource Bottleneck: Optimization and Accuracy Trade-offs

The key limitations of TinyML are both memory (SRAM/Flash) and compute capacity (MIPS/DMIPS). While traditional ML models require gigabytes of memory, TinyML devices can have less than 1/1000th that amount of memory (Banbury et al., 2020).

- Model Accuracy-Efficiency Tradeoff: While model compression techniques (quantization, pruning) are critical, they often result in a decrease in the overall model accuracy. A primary goal of academic research is to develop methods that minimize or compensate for this decrease. Hardware-aware quantization algorithms are needed to prevent accuracy loss, particularly in cases of excessive quantization (e.g., 4-bit or binary quantization).
- Dynamic Resource Management: TinyML devices are typically battery-powered and subject to environmental conditions (temperature, humidity). In these dynamic environments, the development of adaptive inference mechanisms that can instantly manage power consumption and computational resources, even adjusting the model compression level based on task intensity, is an important research topic (Kallimani et al., 2023).

5.2. Hardware Heterogeneity and Specialized Accelerators

The world of embedded systems includes a wide variety of microcontroller families (ARM Cortex-M0 to Cortex-M7), digital signal processors (DSPs), and custom-designed AI accelerators. This heterogeneity creates challenges for portability and optimization.

- Compiler Challenges: Re-optimizing and compiling an ML model to run most efficiently on different hardware architectures is complex. Compiler frameworks like MicroTVM aim to address this issue by converting the ML model to a hardware-specific intermediate representation and then optimizing it for hardware kernels. However, developing efficient compilers and runtime

environments for next-generation neuromorphic chips remains an open area of research.

- Hardware-Software Co-Design: To further advance the capabilities of TinyML, specialized, low-power hardware accelerators (e.g., Edge TPU) designed with the constraints of ML models in mind are crucial. Research is focused on developing new architectures that provide the best balance between power consumption and computational efficiency, particularly event-driven architectures like Spiking Neural Networks (SNNs).

5.3. Security and Privacy Implications at the Edge

Although processing data locally increases privacy, TinyML devices face new security and privacy threats.

- Model Intellectual Property and Attacks: The optimized ML model stored in the MCU is an intellectual property (IP) asset. If the device is physically compromised, there is a risk of model parameters being stolen through model extraction attacks. Secure boot, hardware encryption, and obfuscation techniques are being investigated to mitigate this risk.
- Data Poisoning and Reliability: TinyML devices can receive data from low-cost sensors. Malicious actors can manipulate sensor data (data poisoning) or use adversarial attacks during the inference phase to cause the model to produce inaccurate results. Hardening techniques need to be developed to ensure TinyML devices are resilient to such attacks while minimizing computational overhead.

5.4. Learning Paradigm Shifts: From Inference to On-Device Learning

TinyML's current focus is on a model trained in the cloud performing inference on-device, but future systems should have limited on-device learning capabilities.

- On-Device Learning and Continual Learning: The device requires small amounts of local retraining (fine-tuning) to adapt to changing environmental data (data drift) over time. Given memory and power constraints, high-efficiency, memory-friendly optimization algorithms for updating model weights (e.g., minimized versions of Stochastic Gradient Descent) are a critical research topic.
- Federated Learning (FL): Multiple devices train the model with their own local data and send only the updated weight differences (gradients) to a central server, enabling global model improvement while preserving privacy. While TinyML is an ideal endpoint for FL, ensuring FL algorithms operate

efficiently in the context of ultra-low power and unreliable network connections presents significant engineering challenges.

6. FUTURE TRENDS AND OPPORTUNITIES

The future of TinyML offers both exciting trends that push the boundaries of technology and new market opportunities.

6.1. Next-Generation Hardware and Architectures

The biggest factor that will shape the future of TinyML will be the leaps in hardware.

- Neuromorphic and Event-Driven Computing: Neuromorphic chips (e.g., Intel Loihi) bring AI closer to the principles of biological brains: computation and memory are unified, with processing power triggered by events (spikes). These architectures promise picojoule (pJ) energy consumption, making it possible to achieve Zero-Power AI. The integration of Event-Based Vision (EV) and TinyML with these architectures is a significant focus of academic research.
- Data-Aware Computing (PIM): PIM technologies, which move the computation unit into memory, eliminate the energy cost of data transfer. PIM for TinyML will significantly alleviate memory constraints, enabling larger, more complex models to be run on constrained devices. Data-Aware Computing (In-Memory Computing/Processing-in-Memory) technologies eliminate the energy cost of data transfer. PIM for TinyML will significantly alleviate memory constraints, enabling larger, more complex models to run on constrained devices.

6.2. Integration with Edge-Cloud Continuum and 6G IoT

TinyML is no longer an isolated technology, but part of a larger edge-cloud continuum architecture.

- 6G IoT and Cognitive Networks: Future 6G networks aim to integrate local and hyper-fast computing capabilities. TinyML devices will serve as cognitive sensors in these networks, providing local AI inference to manage and optimize network resources (Scribd, 2023). TinyML will play a critical role in meeting the low latency and high reliability requirements of 6G.
- Hierarchical Inference: Some data is processed on the most constrained TinyML device (layer L_0), more complex data is processed on the local gateway (layer L_1), and the most complex analysis is processed in the cloud (layer L_2). This hierarchical model maximizes both energy efficiency and depth of analysis.

6.3. Novel Application Domains and Societal Impact

- Biomedical and Personalized Healthcare: TinyML-powered implantable devices and smart biosensors will enable continuous and autonomous monitoring of chronic diseases. Real-time diagnostic and alert capabilities have the potential to revolutionize patient care (Scribd, 2023).
- Sustainable Development and Environmental Monitoring: TinyML offers the opportunity to directly contribute to the United Nations Sustainable Development Goals (SDGs) by providing cost-effective and energy-efficient solutions for areas such as monitoring natural habitats, localized detection of climate change impacts, and increasing agricultural productivity (Abadade et al., 2023).

7. CONCLUSION

TinyML for Embedded Intelligence is a rapidly evolving, interdisciplinary field that represents the marriage between machine learning and embedded systems. Initially launched with the mission of enabling AI with limited resources, TinyML now forms the foundation of cognitive systems that offer real-time autonomy, superior privacy, and environmental sustainability.

Continuous advances in model compression techniques (quantization and pruning) and optimized software frameworks like TFLM have made TinyML widely applicable to applications ranging from Industrial IoT to healthcare. However, issues such as hardware heterogeneity, adaptive resource management, cybersecurity threats, and on-device learning capabilities remain pressing challenges for academic research. The future holds the promise of further expanding TinyML's capabilities through integration with neuromorphic hardware, PIM technologies, and 6G infrastructure. Ultimately, TinyML defines the future of distributed and planet-friendly intelligence, enabling the digital world to intelligently interpenetrate the physical world.

REFERENCES

Abadade, Y., Temouden, A., Bamoumen, H., Benamar, N., Chtouki, Y., & Hafid, A. S. (2023). A Comprehensive Survey on TinyML. *IEEE Access: Practical Innovations, Open Solutions*, *11*, 96892–96922. https://doi.org/10.1109/access.2023.3294111

Adlakha, A., & Kabbar, M. (2024). The Challenges of TinyML Implementation: A Literature Review. *CITRENZ2023 Proceedings*, 1-7. Unitec ePress.

Banbury, C., et al. (2020). MLPerf tiny benchmark. *Proceedings of Machine Learning and Systems*, 1–16.

Bulutistan, Retrieved November 7, 2025, from https://bulutistan.com/blog/ai-model-compression-nedir-yapay-zeka-modellerini-optimize-etme-teknikleri/

Kahya, E., & Aslan, Y. (2024). Derin Öğrenme Destekli Gerçek Zamanlı Zeytin Tespiti Uygulaması. Osmaniye Korkut Ata Üniversitesi Fen Bilimleri Enstitüsü Dergisi, 7(4), 1438-1454. https://doi.org/10.47495/okufbed.1392386.

Kallimani, R., Yelchuri, A., & Soro, F. (2023). *Performance Evaluation of TinyML Algorithms on Resource-Constrained Devices*.

Kreß, P., et al. (2024). A review on resource-constrained embedded vision systems-based Tiny Machine Learning for robotic applications. *Algorithms, 17*(11), 476. https://doi.org/10.3390/a17110476.

Kumari, N., Yadagani, A., Behera, B., Semwal, V. B., & Mohanty, S. (2024). *Human motion activity recognition and pattern analysis using compressed deep neural networks.* Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization, 12(1), Article 2331052. https://doi.org/10.1080/21681163.2024.2331052.

Lin, J., Zhu, L., Chen, W.-M., Wang, W.-C., & Han, S. (2023). Tiny machine learning: Progress and futures [feature]. *IEEE Circuits and Systems Magazine*, *23*(3), 8–34. https://doi.org/10.1109/mcas.2023.3302182.

Lipski, M. (2022). *Hand Gesture Recognition on Arduino Using Recurrent Neural Networks and Ambient Light* (Bachelor's thesis, Delft University of Technology). Delft University of Technology Repository. https://repository.tudelft.nl/

Loh, T., & Guo, Y. (2025). Tiny Machine Learning and on-device inference: a survey of applications, challenges, and future directions. *Sensors (Basel), 25*(10), 3191. https://doi.org/10.3390/s25103191.

Maldonado Soliz, I. F., et al. (2025). Advancing TinyML in IoT: A holistic system-level perspective for resource-constrained AI. *Edge Computing Review*, 12(2), 120-145.

Reddi, V. J., Plancher, B., Kennedy, S., & Tingley, D. (2022). Widening access to applied machine learning with TinyML. *Communications of the ACM*, 65(1), 40-49.

Tosun, M., & Erdem, H. (2024). TinyML tabanlı görsel işitsel anahtar kelime tespiti. Niğde Ömer Halisdemir Üniversitesi Mühendislik Bilimleri Dergisi, 13(4), 1207-1215. https://doi.org/10.28948/ngumuh.1482481

Scribd. (2023) *Future directions in TinyML - Emerging Trends and innovations*. (n.d.). Retrieved November 7, 2025, from https://www.scribd.com/document/879567310/Future-Directions-in-TinyML-Emerging-Trends-and-Innovations-docx.

Soro, S. (2021). TinyML for Ubiquitous Edge AI. https://arxiv.org/pdf/2102.01255

Wang, L., & Yoon, K.-J. (2022). Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(6), 3048–3068. https://doi.org/10.1109/tpami.2021.3055564.

Wilson, J., & Singh, D. (2025). Quantized Neural Networks for Microcontrollers: A comprehensive review of methods, platforms, and applications. *arXiv preprint* arXiv:2508.15008.

Yelchuri, A., & R., N. (2022). *Energy Management in TinyML Frameworks*.

Yelchuri, S., & R., M. (2022). Embedded intelligence through TinyML: paradigm shift at the edge. *Journal of Edge Computing*, 5(3), 45-58.

Zhou, H., Zhang, X., Feng, Y., Zhang, T., & Xiong, L. (2025). *Efficient human activity recognition on edge devices using DeepConv LSTM architectures.* Scientific Reports, 15, Article 13830. https://doi.org/10.1038/s41598-025-98571-2

Chapter 6

Outlier Analysis in Machine Learning: Basic Approaches, Challenges, and Applicatıons

Merve AKKUŞ[1]

ABSTRACT

Outlier detection plays a critical role in ensuring the reliability, stability, and generalizability of machine learning models. Real-world datasets often contain deviations arising from noise, measurement errors, rare events, or malicious activities, which distort model learning and lead to inaccurate decisions. This chapter provides a comprehensive examination of outlier analysis by discussing fundamental concepts, outlier types, challenges, and state-of-the-art detection methods. Statistical, proximity/density-based, clustering-driven, and machine learning-based approaches are compared both theoretically and conceptually. Furthermore, modern deep learning techniques, hybrid structures, and explainable artificial intelligence (XAI) frameworks are highlighted as powerful solutions for complex and high-dimensional data. Practical Python examples and visual representations are included to support understanding of algorithmic behavior. The chapter emphasizes that outlier detection is not only a preprocessing task but also a strategic component that significantly affects data-driven decision systems across various fields such as finance, healthcare, cybersecurity, and industrial monitoring.

Keywords: Outlier detection, anomaly detection, machine learning, density-based models, deep learning, data mining, explainable artificial intelligence

[1] Research Assistant Dr., Batman University, Faculty of Engineering and Architecture, Department of Computer Engineering, merve.gitmez@batman.edu.tr, ORCID: 0000-0002-6648-0946

1. INTRODUCTION

This section comprehensively examines the role of outlier detection in data mining and machine learning processes. Outliers are defined as observations that show significant deviations from the general pattern of the dataset and negatively impact the model's accuracy, predictive power, and generalizability performance. Therefore, the accurate identification of outliers is critical in numerous fields, such as credit risk management, network security, medical diagnosis systems, production line monitoring, energy management, and sensor-based industrial applications.The performance of machine learning models is directly related to the quality of the data used. Since real-world data is often not collected under ideal conditions, deviations in the data distribution may occur due to sensor noise, measurement errors, missing records, or unexpected events. These deviations manifest as observations that do not conform to the overall pattern of the dataset and exhibit statistically unusual behavior.

Such observations mislead the model parameters during the learning process, leading to distorted decision boundaries and reduced overall performance. This situation is visually presented in Figure 1. The figure shows that the presence of several outliers (red triangles) in the linear regression model significantly distorts the regression line (blue line), which represents the overall trend of the model. Outliers reduce the accuracy of the model's predictions and weaken its generalizability by distorting the underlying pattern of the data.

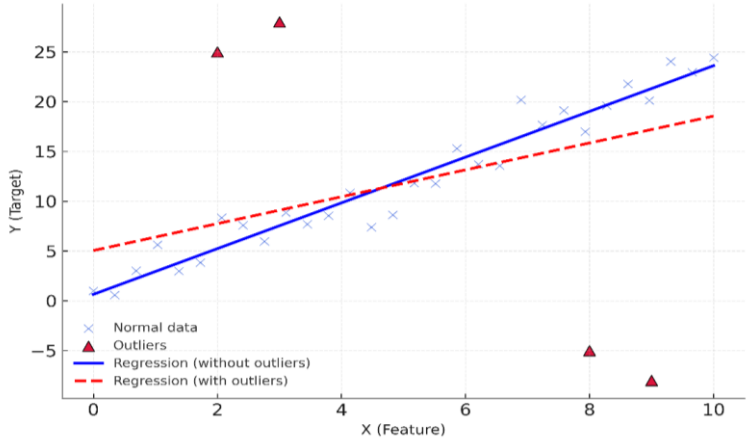


Figure 1. Effect of outliers on model performance

Normal data points are shown as blue circles, while outliers are shown as red triangles. Outliers cause the slope to become distorted by altering the model's least squares direction. This clearly demonstrates why outlier detection is critical for model reliability. The accurate identification of outliers is a critical

component of the machine learning process that is not limited to the preprocessing stage; it directly impacts prediction accuracy, model stability, and generalizability. Numerous studies in the literature show that ignoring outliers reduces model reliability and significantly increases error rates. Therefore, outlier detection is not merely a data cleaning process but a strategic step that ensures the integrity and reliability of data analytics.

This book chapter thoroughly examines the concept of outliers, their types, detection challenges, and machine learning-based methods. The differences between statistical, proximity and density-based, clustering-based, and learning-based approaches are compared; the advantages offered by modern methods are supported by Python-based visuals. Thus, the aim is to provide the reader with a comprehensive perspective covering both the theoretical foundations and practical application examples of outlier analysis.

2. THE CONCEPT OF OUTLIERS AND THEIR ROLE IN DATA MINING

Outlier detection is the process of identifying objects that deviate significantly from expected patterns in data mining processes and exhibit characteristics that are markedly different from other observations. These observations may arise for various reasons, such as measurement errors, sensor malfunctions, fraud attempts, biological anomalies, or systematic failures.

According to Chandola et al.'s definition, outlier detection is a fundamental area of analysis that aims to understand system behavior, anticipate potential risks, and improve decision-making processes by revealing unexpected patterns in the data (chandola et al., 2009).

From a data mining perspective, the importance of outliers is not limited to the data cleaning process. In most cases, these values carry critical information about the overall dynamics of the system or unusual events (Goldstein et al., 2016).

For example:

• In a banking system, a rare high-value transaction may indicate potential fraud.

• A sudden temperature increase on a production line may indicate a sensor malfunction.

• Biological deviations observed in medical data may provide important clues for early diagnosis.

Therefore, outliers should not always be considered “noise.”

Instead, in many scenarios, they should be interpreted as anomalies that provide valuable information about the system's operation.

2.1 Definition and Importance of Outliers

An outlier is an observation in a dataset that deviates significantly from the expected statistical distribution or normal pattern. Such observations can distort parameter estimates in the statistical modeling process, increase variance, and reduce the model's generalization performance. Failure to properly manage outliers causes machine learning models to overfit, reduces classification accuracy, and leads to incorrect decision boundaries. Therefore, identifying, removing, or weighting outliers is a critical step in the data preprocessing process.

In modern data analytics, detecting outliers is not only a statistical necessity but also an essential process for obtaining meaningful data representation. As the volume and complexity of data streams increase in big data, IoT, and cyber-physical systems, outlier analysis has gained strategic importance in terms of system security and operational stability. Figure 2 is presented to conceptually illustrate the impact of outliers on model performance. This visual summarizes the difference between normal data distribution and outliers, as well as their effects on the model, in a comprehensive manner.

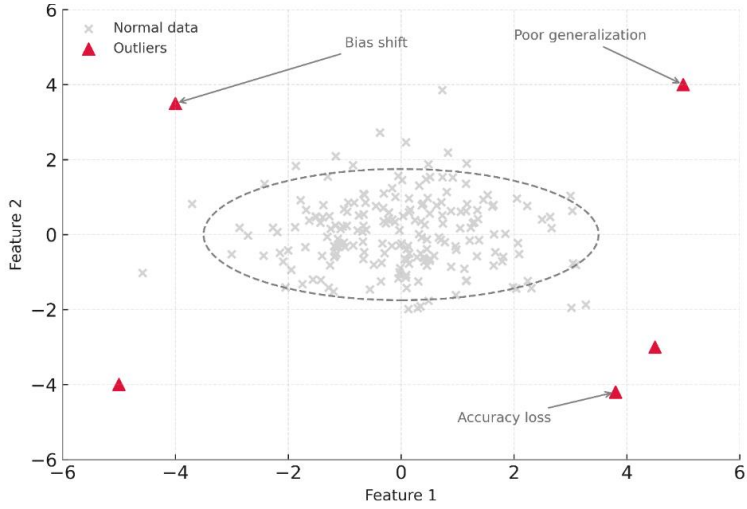


Figure 2. The Concept and importance of outliers (conceptual representation)

The normal data distribution is represented by gray points clustered within the gray ellipse, while outliers are represented by red triangles outside the distribution. Outliers cause a decrease in the model's prediction accuracy (accuracy loss), weak generalization (poor generalization), and prediction bias

(bias shift). This visual emphasizes that outlier detection is not only a data cleaning process but also a strategic step in terms of model reliability and the accuracy of decision systems.

2.2 Application Areas of Outliers

Outlier detection is an interdisciplinary field of analysis and is applied in many different sectors (Ahmed et al., 2016). The most common usage examples are summarized in Table 1 below.

Table 1. Common application areas and examples of outlier detection

Application Area	Purpose	Example of Outlier
Finance	Detect fraudulent transactions, credit risk, or money laundering	Unusually high money transfer
Cybersecurity	Identify unauthorized access or attacks	Abnormally high network traffic packet density
Healthcare	Detect physiological abnormalities	Sudden increase or decrease in heart rate
Manufacturing	Identify defective products or process faults	Sudden deviation in temperature sensor readings
Energy Systems	Detect leakage or faults	Unexpected surge in energy consumption
Text and Social Media Analytics	Monitor topic or sentiment changes	Sudden semantic shift in text content

This wide range of applications demonstrates that outlier detection is an interdisciplinary method. Although the types of data used in different fields vary, the common goal is to systematically identify rare and unusual behaviors that fall outside normal patterns. Today, this process goes beyond classical statistical methods and is carried out in a more flexible and accurate manner through machine learning and deep learning-based approaches.

3. TYPES OF OUTLIERS AND ANALYSIS

Outlier analysis is the process of identifying, measuring, and classifying observations that fall outside the general distribution in a data set. This analysis aims to quantitatively reveal the extent to which an observation deviates from the “normal” pattern. The level of outlier status is usually expressed as an outlier score or probability value. This score numerically shows how differently the observation behaves when compared to other examples in the data set (Goldstein et al., 2012).

Traditionally, outlier analysis is performed under the assumption that the data follows a normal distribution. However, in the real world, most data sets deviate from ideal distributions; in such cases, classical statistical approaches may be insufficient. Especially in high-dimensional and noisy data, outliers need to be defined in different contexts. Therefore, the literature generally addresses outliers in three main categories: global, contextual, and Collective (Kohli et al., 2025).

3.1 Types of Deviant Values: Global, Contextual, Collective

In the literature, deviant values are generally examined under three main categories: global, contextual, and collective deviance.

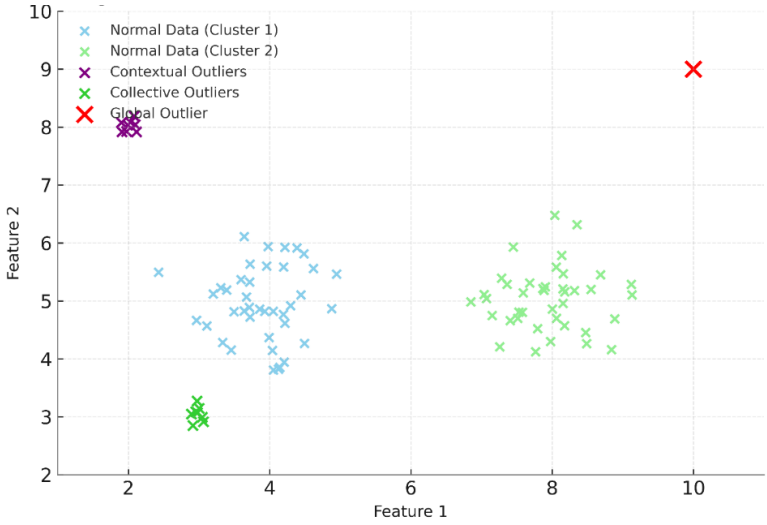


Figure 3. Visualization of global, contextual, and collective outlier examples

In Figure 3, normal data is distributed in blue and green clusters, while points shown in different colors represent outlier behavior patterns. Purple points represent contextual outliers that deviate from the norm in a specific context (e.g., time, location, or condition), light green points represent collective outliers that occur together, and the red "×" signifies a global outlier that clearly deviates from the overall distribution. This visual illustrates the positions of different types of outliers within the data structure and how each requires different identification strategies in the modeling process.

a) Global outliers: Observations that behave distinctly differently from the general distribution of the data set. Example: A measurement of 45 °C in a city where average temperatures range between 20–30 °C represents a global outlier.

Global outliers are typically detected using statistical measures such as Z-score, boxplot (IQR), or Mahalanobis distance (Dashdondov et al., 2021).

b) Contextual outliers: Observations that are considered abnormal in a specific context—such as time, location, or environmental conditions. For example, a temperature of 35 °C is normal in summer but indicates a contextual outlier in winter. Such outliers are typically identified using time series models (ARIMA, LSTM) or location-based analyses (Çalıkuş, 2025).

c) Collective outliers: These are groups of observations that appear normal individually but form an abnormal pattern when taken together. For example, a short-term surge in network traffic could be part of a cyberattack pattern. Such outliers are typically detected using clustering or density estimation methods (Fisch et al., 2022).

3.2 Anomaly Score and Quantitative Assessment

To analytically evaluate outliers, each observation is assigned an outlier score. This score indicates how different the observation behaves compared to other examples in the dataset (Röchner et al., 2024).

In general, the outlier score can be defined as in Equation 1:

$$S(x_i) = f(\text{dist}(x_i, \mathcal{N}_k(x_i))) \quad (1)$$

Where:

- $S(x_i)$: x_i Outlier score of observation,
- dist:Function measuring distance or density difference,
- $\mathcal{N}_k(x_i)$: Set of k-nearest neighbors x_i
- $f(\cdot)$: Transformation function of the score.

As the score value increases, the abnormality level of the observation increases. In practice, these scores are often normalized using metrics such as LOF (Local Outlier Factor) or z-score. This numerical approach not only identifies outliers but also prioritizes them. For example, in credit risk analysis, transactions with the highest outlier scores are examined first (Röchner et al., 2024).

4. CHALLENGES IN OUTLIER DETECTION

Outlier detection is one of the most complex preprocessing steps in machine learning and data mining processes. The main reason for this is that “normal” and “abnormal” behaviors are often not separated by clear boundaries. There is a broad gray area between normality and abnormality in data distributions, which makes the performance of detection methods highly dependent on the

data structure. The main challenges frequently encountered in outlier detection and the solution approaches to these problems are summarized in the subheadings (Kohli et al.,2025).

4.1 Data Normality and Noise Problem

In real-world data, “normal” behavior patterns vary depending on system conditions and time factors. Therefore, the fact that a specific threshold value is not always valid increases the risk of misclassification. Furthermore, noise causes a decline in data quality and hides true outliers. Noisy observations can lead to false positives, especially in statistical methods.

Possible solutions:

- Noise filtering techniques,
- Robust statistical methods (ROF, RANSAC),
- Dynamic threshold determination approaches supported by expert knowledge (Olteanu et al., 2023).

4.2 High Dimensionality and Scalability

Modern datasets often contain hundreds or even thousands of features. An increase in the number of dimensions fundamentally changes the geometric structure of the data. In high-dimensional spaces, Euclidean distances between examples become very close, and the concepts of “close” or “far” lose their meaning. This phenomenon is referred to in the literature as the curse of dimensionality (Kohli et al., 2025).

A direct consequence of this situation is the weakening of the discriminative power of distance- or density-based methods (e.g., k-NN, LOF, DBSCAN). This is because in high-dimensional spaces:

- Data points are almost equidistant from each other,
- Density measurements become inconsistent,
- The clustering structure is disrupted,
- Outliers can no longer be distinguished from normal samples.

Therefore, high dimensionality not only increases computational load but also reduces algorithmic stability and generalization ability.

4.3 Label Deficiency and Imbalance

Outliers are rare by nature; therefore, most datasets do not contain labeled outlier examples. This limits the generalization capacity of supervised learning methods.

Furthermore, a significant difference in the ratio of normal to outlier examples causes class imbalance. This can cause the model to overfit the "normal" class (Kohli et al., 2025).

Possible solutions:

• Unsupervised or semi-supervised algorithms (One-Class SVM, Isolation Forest, LOF),

• Data augmentation (SMOTE) or weighted loss functions,

• Model updating with active learning and expert feedback.

4.4 Model Explainability

Deep learning-based outlier detection models (e.g., Autoencoder, GAN) provide high accuracy, but their decision processes are often "black box" in nature. This leads to reliability issues, especially in critical areas such as healthcare, finance, and security (Birihanu et al., 2024).

Explainable Artificial Intelligence (XAI) approaches are being developed to address this shortcoming.

Methods such as SHAP, LIME, and Counterfactual Explanation make decision processes transparent by explaining why the model flagged a particular observation as an outlier.

4.5 Computational Cost

Working with millions of observations and high-dimensional features in big data environments causes traditional algorithms to fall short in terms of both memory and processing load.

Possible solutions:

• Approximate Nearest Neighbor algorithms,

• GPU or multi-core processing support,

• Online or incremental learning approaches.

5. OUTLIER DETECTION METHODS

Outlier detection is built on different assumptions depending on the structure of the data, its distribution characteristics, the number of dimensions, and the application context. In the literature, these methods are generally classified within the framework of statistical assumptions, distance or density measures, clustering structure, or learning-based models. This diversity stems from the unique nature of each data type. For example, statistical approaches are more suitable for low-dimensional data sets with defined distributions, while machine learning-based methods are more flexible and yield successful results for complex and high-dimensional data. Density or distance-based methods are

particularly effective in capturing local anomalies, while clustering-based approaches reveal contextual anomalies by evaluating structural relationships in the data space. In recent years, with the increase in data volume and complexity, it has become clear that no single method can be effective for all data types (Badhan et al., 2023).

For this reason, researchers have developed hybrid or mixed approaches that combine the strengths of different algorithms (e.g., DBSCAN + Autoencoder, Isolation Forest + PCA). This enables both statistical robustness and learning capabilities through deep representations. In this context, approaches to outlier detection are considered not only as data cleaning tools but also as analytical models that increase the reliability of data interpretation and decision support systems.

5.1 Extreme Value Analysis

In extreme value analysis, observations located in the extreme regions of the data developments shown are considered outliers. Such observations are the points that emerge in the underlying sequential part, as shown in Figure 4 (Olmo., 2009).

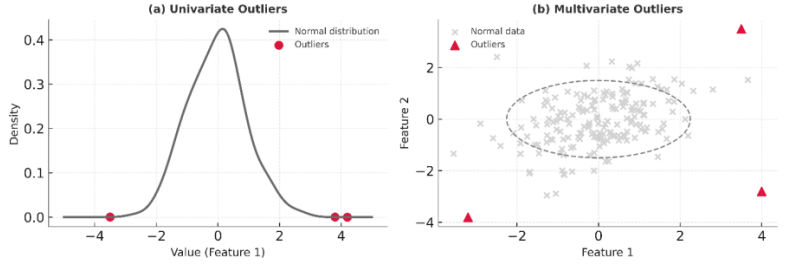


Figure 4. Conceptual representation of single and multivariate outlier detection.

(a) In the single variable case, outliers are defined as observations located at the extreme points of the statistical distribution. The normal distribution curve is shown in gray, and the red dots in the extreme regions represent outliers outside the expected range.

(b) In the multivariate case, outliers emerge as deviations from the common pattern of multiple features. The gray ellipse shows the normal data boundary, while the red triangles show multivariate outlier observations outside this boundary.

This visual emphasizes the importance of considering the number of variables and their relationships when detecting outliers.

a) Univariate outlier analysis: In a univariate case, it is assumed that the data follows a specific distribution (most often a Gaussian distribution). As in Equation 2, the upper and lower tail regions represent observations at a distance of ±3σ from the mean:

$$x_i > \mu + 3\sigma \text{ or } x_i < \mu - 3\sigma \quad (2)$$

Points outside these thresholds are marked as outliers.

b) Multivariate outlier analysis: In multidimensional data sets, the distance of observations from the mean vector μ and covariance matrix Σ is measured using the Mahalanobis distance, Equation 3:

$$D_M(x_i) = \sqrt{(x_i - \mu)^T \Sigma^{-1} (x_i - \mu)} \quad (3)$$

Distances above a certain threshold value (τ) are considered outliers.

It is statistically robust and effective with low-dimensional data. However, it is highly dependent on the assumption of normal distribution.

5.2 Statistical Methods

Statistical methods assume that the data follows a specific distribution pattern, as shown in Figure 5, and consider observations that do not fit this pattern as outliers (Theriault, 2024).

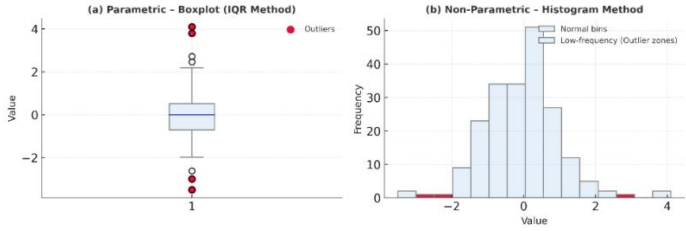


Figure 5. Statistical methods for aoutlier detection: Parametric and non-parametric approachs

(a) Parametric methods assume that the data follows a specific distribution model; the Boxplot (IQR) method identifies outliers using quartile values.

(b) Non-parametric histogram-based methods mark low-frequency regions as potential outlier areas without making any distribution assumptions.

This comparison demonstrates the importance of selecting a statistical method appropriate for the data structure.

a) Parametric methods: Parametric methods detect outliers by estimating distribution-based parameters such as quartiles. A common example is the

Boxplot method, which uses the first, second, and third quartiles Q_1, Q_2, Q_3 of the data (Equation 4). The dispersion of the central data is measured by the interquartile range, defined as $\text{IQR} = Q_3 - Q_1$ (Equation 5). Based on this range, outlier thresholds are determined as $Q_1 - 1.5 \times \text{IQR}$ for the lower limit and $Q_3 + 1.5 \times \text{IQR}$ for the upper limit (Equation 6). Observations outside these limits are classified as outliers.

Data quartiles:

$$Q_1, Q_2, Q_3 \tag{4}$$

Interquartile range (IQR):

$$IQR = Q_3 - Q_1 \tag{5}$$

Outlier limits:

$$Lower\ limit = Q_1 - 1.5 \times IQR, Upper\ Limit = Q_3 + 1.5 \times IQR \tag{6}$$

Points outside these limits are considered outliers.

b) Nonparametric methods: Nonparametric approaches are preferred when the data distribution is unknown. For example, in the histogram-based outlier detection method, data are divided into intervals (bins); low-frequency intervals are potential outlier regions. Nonparametric methods do not require a distribution assumption. However, threshold selection and histogram width also significantly affect the results.

5.3 Proximity and Density-Based Methods

In this approach, the anomaly of an observation is assessed according to its distance from its neighbors or its density difference. The assumption is that normal observations are in dense regions and anomalies are in sparse regions, as shown in Figure 6 (Mavroudopoulos, 2023).

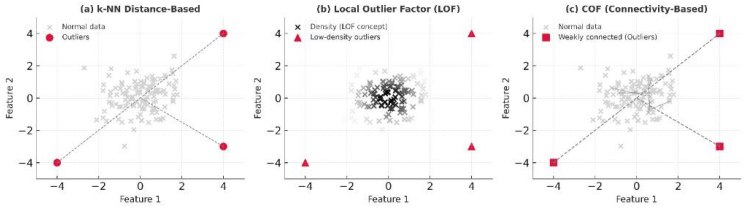


Figure 6. Conceptual representation of proximity and density-based outlier detection methods.

(a) The k-NN method determines the level of outlierness based on the average distance of observations to their neighbors; points far from their neighbors are considered outliers.
(b) The LOF method evaluates observations in low-density regions as outliers based on local density differences.
(c) The COF method defines observations in weakly connected regions as outliers by considering point connection lengths.

These methods are effective tools for detecting local anomalies, especially in complex and multidimensional data sets.

a) k-Nearest neighbor (k-nn) outlier detection: The average distance of an observation to its k nearest neighbors is calculated. Points with high distance values are labeled as outliers.

b) Local outlier factor (LOF): The LOF method compares the local density of a point with its neighbors. The outlier score is defined as in Equation 7.

$$LOF(x_i) = \frac{1}{|N_k(x_i)|}\sum_{x_j \in N_k(x_i)} \frac{\text{lrd}(x_j)}{\text{lrd}(x_i)} \quad (7)$$

Here, lrd (local reachability density) indicates the reachability density of a point.

Points with LOF>1.5 are generally considered outliers.

c) COF (Connectivity-based outlier factor): It works similarly to LOF but evaluates local density based on connection lengths. Observations in regions where density shows a sudden drop are considered outliers. COF successfully captures anomalies in the local structure. However, it is sensitive to k-parameter selection and has high computational costs in large datasets.

5.4 Cluster-Based Methods

Cluster-based methods are based on the assumption that normal observations form clusters, as shown in Figure 7, and outliers are isolated from these clusters (Souiden et al., 2022).

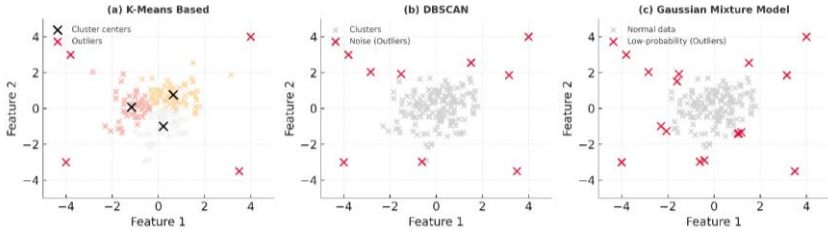


Figure 7. Conceptual representation of clustering-based outlier detection methods.

(a) The K-Means method determines the level of outlierness based on the distance of observations to the nearest cluster center; points far from the center are considered outliers.

(b) The DBSCAN method identifies clusters using a density threshold and minimum neighbor count; observations in low-density regions are considered noise (outliers).

(c) The GMM method assumes that the data is composed of a mixture of multiple Gaussian distributions and classifies low-probability observations as outliers.

These methods define outliers based on their statistical and spatial characteristics, taking into account the structure of the data sets.

a) K-means-based outlier detection: The K-Means algorithm divides the data into k clusters. The distance of each observation from the nearest cluster center is calculated (Equation 8). If the distance exceeds a certain threshold, the observation is marked as an outlier.

$$OD(x_i) = \| x_i - \mu_c \| \tag{8}$$

b) DBSCAN (Density-based spatial clustering of applications with noise): DBSCAN operates with the parameters density threshold ε and minimum number of neighbors MinPts. Observations in dense regions are clustered, while those remaining in low-density regions are considered noise and classified as outliers. The number of clusters in DBSCAN does not need to be known beforehand; it defines noise naturally. However, parameter selection is sensitive to data scale.

c) Gaussian mixture model (GMM): It assumes that the data is generated by a mixture of multiple Gaussian distributions. A probability value is calculated for each observation; low-probability examples are classified as

outliers. GMM is effective on non-spherical clusters. However, it carries the risk of overfitting due to the excessive number of parameters.

5.5 Machine Learning-Based Methods

Machine learning approaches offer high accuracy and generalization capacity in outlier detection, as shown in Figure 8. The aim of these methods is to learn normal data and identify deviations from this pattern as outliers. Especially in high-dimensional and nonlinear datasets, much more flexible and stable results are obtained compared to classical statistical approaches (Souiden et al., 2022).

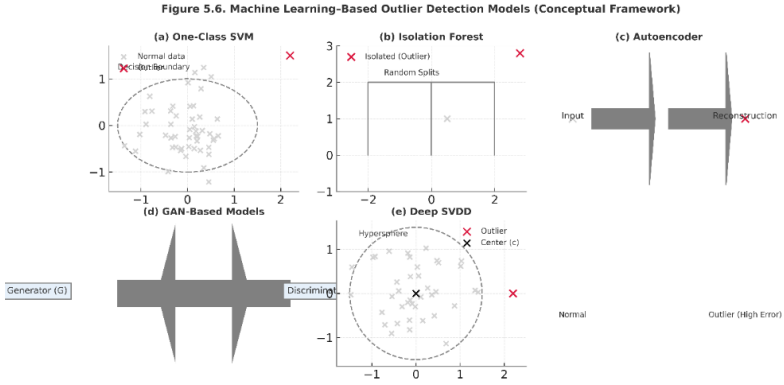


Figure 8. Machine learning-based outlier detection models

a) One-class SVM: One-Class SVM learns only the **"normal"** class and labels samples lying outside the decision boundary as **outliers**. The decision function is defined, as given in Equation 9:

$$f(x) = w^T \phi(x) - \rho \tag{9}$$

where $\phi(x)$ represents the kernel transformation, w is the weight vector, and ρ is the decision threshold. Observations where $f(x) < 0$ are considered outliers. This method is effective on low-dimensional, well-defined datasets and is sensitive to the choice of kernel function and ν parameter.

b) Isolation forest: Isolation Forest creates a tree structure by randomly splitting data points. Observations isolated with few splits are considered outliers. The outlier score is based on the average path length; shorter paths indicate higher outlier status.

This method stands out for its scalability in high-dimensional data and large sample sizes, but it is not sufficient on its own for contextual anomalies.

c) Autoencoder and variants: Autoencoder structures transform the input into a latent representation and reconstruct it. Examples with high reconstruction error are flagged as outliers. Derivatives such as Denoising AE, Sparse AE, and LSTM-AE improve noise resilience and performance in time series. A Variational Autoencoder (VAE), a probabilistic extension, detects anomalies based on both reconstruction error and deviations in the latent space by estimating the probability of each observation. These approaches deliver effective results, particularly in complex industrial sensor data and biomedical signals.

d) GAN (Generative adversarial network)-based methods): In GAN-based models, a generator (G) and a discriminator (D) network undergo a mutual learning process. The generator learns to mimic the normal data distribution, while the discriminator learns to distinguish between real and fake examples. The anomaly score is typically calculated based on the difference between the original and reproduced data or the distance in the discriminator's feature space. Architectures such as AnoGAN, GANomaly, and Skip-GANomaly are particularly successful in image and defect detection. However, since the training process of GANs can be unstable, Autoencoder-GAN hybrids or pre-trained feature extractors are preferred in most applications.

e) Deep SVDD and modern approaches: Deep SVDD (Support Vector Data Description) is a deep version of the classic One-Class SVM. Network outputs are centered around a hypersphere; examples far from the center are considered outliers. This method learns deep feature representations unsupervised and scales better than kernel-based models. In recent years, these approaches have been supported by explainable artificial intelligence (XAI) methods. Tools such as SHAP, LIME, and Counterfactual Explanation increase the interpretability of models by visualizing why an observation is labeled as an outlier.

f) Current trends: New research has focused on combining techniques from different paradigms.

- Hybrid models (e.g., DBSCAN + Autoencoder, Isolation Forest + PCA) combine statistical robustness with deep representations.
- Self-supervised and contrastive learning methods enhance normal data representations without requiring labels.
- Graph-based and time-series-focused approaches enable the detection of anomalies in sensor networks and dynamic systems through topological or temporal inconsistencies.

6. COMPARATIVE EVALUATION AND PYTHON APPLICATION EXAMPLES

This section explains the basic working principles, application forms, and interpretation methods of some algorithms commonly used in outlier detection. The aim is to show the reader how different types of approaches can be applied in practice and to compare the similarities and differences between the methods at a conceptual level.

6.1 Application Environment and Synthetic Data Approach

While real data sets (e.g., MIT-BIH, KDDCup99, Credit Card Fraud, NASA-Bearing, etc.) are frequently used to test these methods, synthetic (artificial) data generation has been preferred in this book chapter to demonstrate the behavior of the algorithms in a straightforward manner.

A small sample dataset created from random distributions in the Python environment (e.g., normal distribution + a small number of outliers) is sufficient to understand how different algorithms respond.

Such data provides an instructive framework for representing real-world noise, bias, and statistical outliers.

6.2. Basic Algorithm Application Examples

The following examples are short code snippets that can be run directly in the Python environment and are intended solely for methodological demonstration. The purpose of the code is not to compare model performance but to teach the basic usage of the methods.

(a) Local outlier factor (LOF): LOF labels samples that remain low in density as outliers by comparing the local density of each observation with its neighbors.

```
from sklearn.neighbors import LocalOutlierFactor
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.05)
y_pred = lof.fit_predict(X)
```

(b) Isolation forest: Isolation Forest randomly splits observations and quickly flags isolated ones as outliers. It stands out for its scalability in large data sets.

```
from sklearn.ensemble import IsolationForest
iso = IsolationForest(contamination=0.05, random_state=0)
y_pred = iso.fit_predict(X)
```

(c) Autoencoder: Autoencoders learn to reconstruct normal samples. Samples with high reconstruction error are considered outliers.

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
input_dim = X.shape[1]
inp = Input(shape=(input_dim,))
enc = Dense(4, activation='relu')(inp)
dec = Dense(input_dim, activation='linear')(enc)
autoencoder = Model(inp, dec)
autoencoder.compile(optimizer='adam', loss='mse')
```

6.3. Conceptual Comparison

Table 2. Conceptual comparison of outlier detection methods

Method	Advantage	Limitation
LOF	Captures local density differences	Sensitive to the *k*-parameter
Isolation Forest	Fast and scalable for large datasets	Limited in detecting contextual anomalies
Autoencoder	Effective for high-dimensional and complex data	Threshold selection is difficult; long training time

The comparison in Table 2 serves as a guide for selecting methods for different data types. For example:

- LOF is more suitable for local data sets where density differences are important.
- Autoencoder is more suitable for multidimensional sensor or financial data.
- Isolation Forest is more suitable for large data streams.

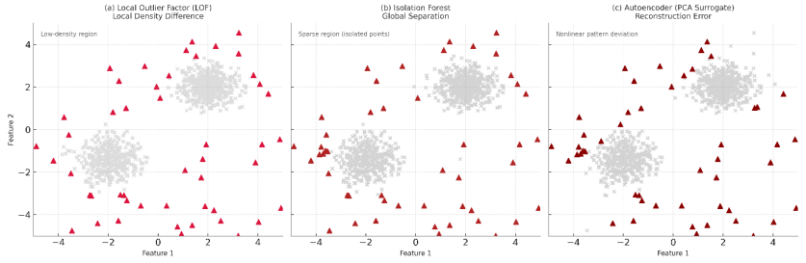


Figure 9. Visual comparison of outlier detection methods

This figure 9 illustrates the conceptual behavior of three different outlier detection methods on a synthetic (artificially generated) two-dimensional datase. In each panel, light gray circles represent normal observations, while red triangles represent observations identified as outliers by the algorithms. The visuals clearly demonstrate how the methods define the concept of outliers in different ways.

(a) Local outlier factor (LOF): The LOF method labels examples found in low-density regions as outliers by comparing the local density of each observation with its neighbors. Therefore, examples located at the edges of clusters or in boundary regions where density decreases are shown with red triangles. This method performs effectively, especially on datasets where local density differences are important.

(b) Isolation forest: The Isolation Forest model evaluates samples that can be isolated quickly as outliers by separating observations through random splits. The model enables global-scale outlier detection because it can easily separate observations in sparse regions or those far from the general distribution. In the figure, isolated observations in these sparse areas are indicated by red triangles.

(c) Autoencoder (AE): The autoencoder-based model learns to reproduce the data and detects anomalies based on reconstruction error. Examples that cannot be reproduced, i.e., those that deviate significantly from the learned pattern, are considered outliers. In the figure, these deviations are shown as red triangles located in areas far from the center of the data distribution.

7. CONCLUSIONS AND FUTURE DIRECTIONS

This section summarizes the general evaluation of machine learning-based outlier detection methods and potential future research directions. Outlier detection is considered not only as a data cleaning process but also as a critical component in terms of model reliability, robustness, and generalizability. The statistical, density-based, clustering-focused, and learning-based approaches discussed in this study demonstrate that the concept of outliers can be approached from different perspectives.

Learning-based models (particularly Autoencoder, Isolation Forest, and GAN derivatives) offer higher accuracy and generalization capacity compared to classical methods in high-dimensional, noisy, and complex datasets. However, the success of these methods is directly dependent on factors such as hyperparameter selection, threshold determination, data imbalance, and model interpretability.

Therefore, it is crucial for future studies not only to achieve high performance but also to be able to interpret why the model considers a particular observation to be an outlier.

In recent years, explainable artificial intelligence (XAI), self-supervised learning, and contrastive learning approaches have been increasingly used in outlier detection. These approaches enable the model to learn anomalies from its own internal representations by reducing the need for labels. Furthermore, hybrid models (e.g., Autoencoder + Isolation Forest or DBSCAN + VAE combinations) offer more balanced solutions by combining statistical robustness with deep representations.

In the future, the integration of outlier detection algorithms into real-time systems, edge devices, and energy-efficient architectures will come to the fore. There is a growing need for low-latency and explainable outlier detection algorithms, particularly in IoT, biomedical sensor networks, production lines, and autonomous systems. However, ethical, security, and data privacy dimensions are also expected to shape new research topics.

In conclusion, outlier analysis has become not only a subfield of data science but also a fundamental research area that determines the reliability, ethical responsibility, and robustness of artificial intelligence systems. Therefore, future studies are expected to develop an interdisciplinary perspective that addresses both algorithmic efficiency and explainability.

REFERENCES

Ahmed M., A. N. Mahmood, and J. Hu, "A Survey of Network Anomaly Detection Techniques," Journal of Network and Computer Applications, vol. 60, pp. 19–31, 2016.

Badhan. A and A. Ganpati, "Overview of outlier detection methods and evaluation metrics: A review," in Proc. 6th Int. Conf. Big Data Analytics and Knowledge Discovery (DaWaK), Sep. 2023, pp. 54-63.

Birihanu E., A. Ayano and et al., "Explainable correlation-based anomaly detection for industrial control systems," Frontiers in Artificial Intelligence, vol. 4, Art. 1508821, 2024. doi:10.3389/frai.2024.1508821.

Calikus E., "Context discovery for anomaly detection," Knowledge and Information Systems, vol. 69, no. 8, pp. 4123–4148, Oct. 2025.

Chandola V., A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Computing Surveys, vol. 41, no. 3, pp. 1–58, 2009.

Dashdondov K. and M.-H. Kim, "Mahalanobis Distance Based Multivariate Outlier Detection to Improve Performance of Hypertension Prediction," Neural Computing and Applications, vol. 33, pp. 10487–10499, 2021.

Fisch A. T. M., I. A. Eckley and P. Fearnhead, "Subset Multivariate Collective and Point Anomaly Detection," Computational Statistics & Data Analysis, vol. 162, Art. 108725, 2022.

Goldstein M. and A. Dengel, "Histogram-based Outlier Score (HBOS): A fast Unsupervised Anomaly Detection Algorithm," in Proceedings of the KI-2012: Poster and Demo Track, September 2012.

Goldstein M. and S. Uchida, "A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data," PLoS ONE, vol. 11, no. 4, e0152173, 2016.,

Kohli M. and I. Chhabra, "A comprehensive survey on techniques, challenges, evaluation metrics and applications of deep learning models for anomaly detection," SN Applied Sciences, vol. 7, Art. 784, Jul. 2025. doi: 10.1007/s42452-025-07312-7.

Mavroudopoulos I. and A. Gounaris, "A comparison of proximity-based methods for detecting temporal anomalies in business processes," Machine Learning, vol. 112, pp. 4101–4128, 2023. doi:10.1007/s10994-022-06152-5

Olmo J., "Extreme value theory filtering techniques for outlier detection," Discussion Paper No. 09/09, Dept. of Economics, City University London, Jun. 2009.

Olteanu M., J. Steinhauser, and A. R. Hedayati, "Meta-Survey on Outlier and

Anomaly Detection," Neurocomputing, vol. 572, pp. 1–25, Dec. 2023. doi: 10.1016/j.neucom.2023.127960.

Röchner P., H. O. Marques, R. J. G. B. Campello, A. Zimek and F. Rothlauf, "Robust Statistical Scaling of Outlier Scores: Improving the Quality of Outlier Probabilities for Outliers (Extended Version)," arXiv preprint, Aug. 28 2024.

Sánchez B. V., E. Schubert, A. Zimek and R. L. F. Cordeiro, "A comparative evaluation of clustering-based outlier detection," Data Mining and Knowledge Discovery, vol. 39, art. 13, Feb. 2025. doi:10.1007/s10618-024-01086-z.

Souiden I., M. Chandola and N. Alsubaie, "A survey of outlier detection in high dimensional data streams," Knowledge-Based Systems, vol. 249, Art. 110861, 2022. doi:10.1016/j.knosys.2022.110861

Thériault R., "Check your outliers! An introduction to identifying statistical outliers," PeerJ PrePrints, 2024. doi:10.31234/osf.io/bu6n

Chapter 7

WebAssembly: An Indispensable Component of the Modern Web

Fikri AĞGÜN[1]
Raif SİME[2]

ABSTRACT

This section examines WebAssembly (Wasm) as a high-performance and portable execution environment operating both within web browsers and outside of them. The study begins by providing a historical background through previous initiatives such as asm.js and Native Client, followed by an explanation of the motivations behind the emergence of WebAssembly and the W3C standardization process. The binary format of WebAssembly, stack-based virtual machine model, linear memory structure, concepts of modules and instances, and import-export based interaction with JavaScript are analyzed.

Discussions include compilation chains and practical use cases through ecosystems such as C/C++ (Emscripten), Rust (wasm-bindgen, wasm-pack), AssemblyScript, TinyGo, and Blazor WebAssembly; the performance gains offered by WebAssembly compared to JavaScript, particularly how these gains manifest in CPU-intensive applications are evaluated with supportive literature. In the security section, the sandbox execution model, memory and type safety, controlled access to system resources via WASI, as well as recent research on side-channel and speculative execution attacks are summarized. The non-browser usage of WASI-based server and edge scenarios, blockchain and smart contracts, embedded/IoT applications, and the migration of legacy desktop software to the web are also discussed.

The conclusion emphasizes that WebAssembly, as the fourth fundamental web component completing the HTML/CSS/JavaScript triad, strengthens the vision of a "portable execution layer" in both academic research and industrial projects; future research and development trends are discussed regarding topics such as performance, security, ecosystem maturity, and component models.

Keywords: WebAssembly, Wasm, High-Performance web, WASI, Portable execution environment

[1] Assist. Prof. Dr., Bitlis Eren University, faggun@beu.edu.tr, ORCID: 0000-0001-9550-1462
[2] Bitlis Eren University, rsime@beu.edu.tr, ORCID: 0009-0008-4292-2456

1. INTRODUCTION

Web technologies have evolved over their approximately thirty-year history from simple static HTML pages to a wide range of complex single-page applications (SPAs), cloud-based gaming, and enterprise business intelligence platforms. During this process, JavaScript has achieved a unique position as the only built-in programming language running in browsers; its interpreted nature, dynamic typing, and rich ecosystem have transformed the web into an "application platform." However, it has also been observed that this flexibility of JavaScript can be limiting in terms of performance and predictability, particularly in scenarios that require high computational power (Wikipedia, 2017). In areas such as 3D game engines, CAD and CAM applications, scientific simulations, image processing, cryptography, and machine learning, developers have had to resort to various indirect methods for porting code written in C/C++, Rust, or other system languages to the browser for years. Initiatives like asm.js and Google Native Client (NaCl/PNaCl) have demonstrated that the web can execute code more quickly; however, they have not provided a universally accepted, standardized, and portable solution across all browsers (Haas et al., 2017). WebAssembly (Wasm) is a secure, portable, and low-level binary code format derived from these requirements, along with a virtual machine model for executing it. In 2019, it was elevated to the level of a core specification (Recommendation) by the W3C and declared the "fourth language" of the web, following HTML, CSS, and JavaScript.

The primary goal of WebAssembly is to enable high-performance applications both on the web and in non-web environments while simultaneously ensuring security, portability, and language independence.

WebAssembly code:

- Can be compiled from a variety of languages, including C, C++, Rust, Go, C#, AssemblyScript, and TinyGo,
- Can run in browsers, on the server side, in cloud edge platforms, and on embedded devices,
- Is executed within a secure sandbox, with type safety ensured and memory accesses controlled.

When comparing WASM-based web applications to traditional JavaScript applications, it is evident that performance limitations exist when using current JavaScript, particularly in cases where these limitations are significantly pronounced. Due to its bytecode compilation, WebAssembly can achieve nearly native speeds compared to JavaScript. Furthermore, WebAssembly is an advantageous technology for executing computation-intensive algorithms with small to medium-sized data volumes.

WebAssembly (WASM)-based technology presents itself as a more effective solution in terms of speed and performance compared to classical methods. The advantages of this technology have been corroborated by numerous studies and documented evidence in the literature.

In their study, which aims to demonstrate how Wasm can provide native performance for web-based AR/VR applications and address the critical challenges faced by existing technologies such as WebXR, the authors conclude that the potential of porting to Wasm can enhance the performance of web-based AR/VR applications, bringing them closer to the performance of native applications (Khomtchouk, 2021).

In their study discussing the potential of WebAssembly as an application virtual machine for embedded systems, the authors highlight its strong isolation features and software portability. Considering its growing ecosystem and adoption beyond web browsers, they emphasize the significance of WebAssembly in scalable and secure IoT deployments (Wallentowitz et al., 2022).

In their study addressing the performance limitations of web applications in graphics-intensive areas such as video games, simulations, and image processing, the authors emphasize WebAssembly's integration with various programming languages, including C/C++, C#, and Rust. They showcase its cross-platform capabilities and efficient memory management, noting that it provides significant performance improvements and possesses the potential to revolutionize web application development. (Tufegdžić et al., 2024).

In the following sections, the design principles, architecture, toolchain, security model, performance characteristics, in-browser and out-of-browser usage scenarios, practical application examples, and current research findings related to WebAssembly will be addressed. Additionally, code snippets and compilation processes will also be discussed through the ecosystems of Rust, C/C++, and C# (Blazor).

2. HISTORICAL BACKGROUND AND DEVELOPMENT PROCESS

2.1. The Experience of asm.js and Native Client

Although JavaScript engines have experienced significant performance leaps over the years through techniques such as JIT compilation, hidden classes, and inline caching, the dynamic nature of the language has made it challenging to efficiently translate to CPU instructions. (Haas et al., 2017).

The asm.js approach is a subset of JavaScript with significantly restricted types and control structures. C/C++ compilers target this subset to enable JIT to perform more predictable optimizations. However, the final output is still text-based JavaScript, which results in large code sizes and extended parsing times.

As an approach for executing native code within the browser, Google's Native Client aimed to run sandboxed machine code through a Chrome-specific architecture. While Portable Native Client (PNaCl) enhanced portability, it failed to become a standard and was not adopted by other browsers.

These initiatives demonstrated that near-native speeds are achievable on the web; however, the need for a standard, browser-independent, and portable bytecode became apparent. WebAssembly has emerged as a technology that directly addresses this need.

2.2. The Emergence of WebAssembly

Starting in 2015, researchers from Mozilla, Google, Microsoft, and Apple formed a joint working group to shape the design of WebAssembly. The first Minimum Viable Product (MVP) was showcased in browser prototypes in 2017; the same year, the paper titled "Bringing the Web up to Speed with WebAssembly" published at the PLDI conference outlined the fundamental principles of the design. The core specification progressed through stages, becoming a W3C Working Draft in 2018, a Candidate Recommendation in 2019, and reaching Recommendation status on December 5, 2019. Currently, in addition to the WebAssembly Core Specification 1.0, work continues on the 2.0 draft and the 3.0 version, with surrounding specifications like the JavaScript API, WebAssembly System Interface (WASI), and the component model also maturing. (Webassembly, 2025).

3. DESIGN GOALS AND PRINCIPLES

The design of WebAssembly is based on several key principles:

1. **Efficient Execution:**

The compactness of the binary format facilitates quick downloading and parsing, and allows the compiled code to run at near-native speeds on hardware through JIT or AOT compilation, thereby contributing to the principle of efficient execution.

2. **Portability:**

The use of a architecture-independent virtual machine model (stack-based VM) and the fact that the bytecode has the same meaning across all modern CPU architectures enhance the portability of this technology.

3. Safety and Security:

Providing memory safety, type safety, and control flow safety, along with executing code within a sandbox with limited APIs defined by the host environment, ensures a higher level of security for this technology. (Perrone & Romano, 2024).

4. Compatibility and Interoperability:

WebAssembly technology can integrate with existing web platforms, collaborating with JavaScript APIs, the DOM, and other Web APIs to function together effectively.

5. Language Independence:

By operating independently of programming languages, WebAssembly serves as a common compilation target for many languages, including C/C++, Rust, Go, C#, AssemblyScript, and TinyGo, highlighting its versatility.

These principles establish WebAssembly not only as an execution platform used in browsers but also as a general-purpose portable execution platform, providing a platform-independent development environment for those working in this field.

4. ARCHITECTURE AND OPERATION MODEL

4.1. Binary and Text Formats

WebAssembly programs are distributed in a binary format with the ".wasm" extension. The compact nature of this format reduces both network transfer time and parsing time in the browser, enabling faster access and execution. The human-readable version of the same code is available in the text format with the ".wat" or ".wast" extensions. This format possesses an S-expression-like syntax.

```
(module
  (func $sum param $x i32) (param $y i32) (result i32)
    local.get $x
    local.get $y
    i32.add)
  (export "sum" (func $sum)))
```

Figure 1. Syntax example

When this code is compiled and converted to binary format, it can be executed by a browser or another WASM runtime. The text format is primarily used for debugging, education, and examining compiler outputs.

4.2. Stack-Based Virtual Machine

WebAssembly has a virtual machine model that reads operands and intermediate results from a stack and writes them back to the stack. It performs these operations in the following sequence (instructions):

- Retrieves a value from the stack (e.g., local.get, i32.const).
- Performs the operation (e.g., i32.add, f64.mul).
- Writes the result back to the stack.

This model provides an abstract machine definition that simplifies the task for compilers to generate code for different CPU architectures.

4.3. Linear Memory

In WebAssembly, memory is organized as a one-dimensional address space known as "linear memory". Each memory consists of 64 KB pages, and modules can increase the size of the memory at runtime as long as resources permit. Applications access memory using load/store instructions.

```
(i32.store
  (local.get $ptr)
  (i32.add
    (i32.load (local.get $ptr))
    (i32.load (i32.add (local.get $ptr) (i32.const 4)))))
```

Figure 2. Memory access of application

This linear model facilitates portability across different architectures, allows for memory boundaries to be monitored, and thus serves as an important foundation for sandboxing.

4.4. Module and Instance

A .wasm file is a module. The module consists of type definitions, functions, global variables, tables, and optionally, a start function. At runtime, this module is instantiated by the host environment.

Modules can import functions and resources from the host and can export functions, memory, or tables to the outside. For example, the relationship between a JavaScript application and a WebAssembly module can be established as follows:

```
const response = await fetch("modul.wasm");
const bytes = await response.arrayBuffer();

const importObject = {
  env: {
    log_i32: (x) => console.log("WASM:", x)
  }
};

const { instance } = await WebAssembly.instantiate(bytes, importObject);
instance.exports.run();      //run() function in wasm
```

Figure 3. Relationship established between JavaScript and WebAssembly module

5. INTEGRATION WITH THE WEB PLATFORM

5.1. Browser Support

All modern browsers, including Chrome, Firefox, Safari, and Edge, support the core features of WebAssembly. In most browsers, WebAssembly is integrated within the JavaScript engine (such as V8, SpiderMonkey, JSC, etc.) and shares the same JIT infrastructure.

5.2. JavaScript – WebAssembly Interaction

WebAssembly modules interact with JavaScript in the following ways:

1. JavaScript loads the WebAssembly module (e.g., using WebAssembly.instantiateStreaming).
2. The functions exported by the module are called by JavaScript.
3. WebAssembly accesses web APIs such as DOM, network, and storage indirectly by calling the host functions it imports.

This collaboration results in the following architecture in practice:

- UI, DOM management, and events are handled on the JavaScript side.
- CPU-intensive computations are executed on the WebAssembly side.

This approach preserves the web ecosystem that developers have been using for years, while also accelerating performance-critical components.

5.3. Direct Access to the DOM

By design, WebAssembly cannot directly access the DOM or browser APIs. This limitation allows WebAssembly to be used not only for browsers but also for general-purpose platforms, maintaining security and a simple architecture by leaving web-specific concepts to the host environment. Therefore, to modify the DOM, JavaScript bridges are utilized, as illustrated in the following example.

```
// JS side
function setResult(text) {
  document.getElementById("result").textContent = text;
}

// WASM imports this function and calls it at the appropriate time.
```

Figure 4. Example of a JavaScript bridge

6. LANGUAGE ECOSYSTEMS AND TOOLCHAIN

The language ecosystems and tools used with WebAssembly can be categorized as follows:

6.1. Emscripten and C/C++

Emscripten is an LLVM-based compilation toolchain used in WebAssembly that compiles C/C++ code into WebAssembly (and asm.js if necessary).

For example,

// sum.c

int sum(int a, int b) {

return a + b;

} a simple C code like the following can be compiled using command:

emcc sum.c -O3 -s WASM=1 -s EXPORTED_FUNCTIONS="['_sum']" -o sum.js

As a result of this process:

- sum.wasm: WebAssembly module,
- sum.js: "Glue" code that loads and executes the module,
- Optionally, a sum.html file are produced.

Emscripten is extensively used for porting game engines and desktop applications to the web, providing SDL, OpenGL to WebGL conversions, POSIX-like APIs, and threading support (Emscripten, 2021) .

6.2. Rust ve Wasm-bindgen Rust and Wasm-bindgen

Rust is naturally well-suited for WebAssembly due to its memory safety guarantees and zero-cost abstractions.

In the Rust ecosystem:

- Compilation can be done directly to Wasm using the wasm32-unknown-unknown target.
- The wasm-bindgen library automatically generates glue code while exporting Rust functions to JS.
- wasm-pack simplifies integration with bundlers and automates the process of publishing packages to NPM.

Here is a simple example in Rust:

```
use wasm_bindgen::prelude::*;

#[wasm_bindgen]
pub fn kare(x: i32) -> i32 {
    x * x
}
```

Figure 5. Rust example

This code can be compiled using the *wasm-pack build* command and can be used as a JavaScript project by being packaged as an NPM module.

6.3. AssemblyScript

AssemblyScript is a statically typed language that closely resembles TypeScript and is designed to be compiled directly to WebAssembly.

- The learning curve is low for JavaScript/TypeScript developers.
- The type system is closely aligned with WebAssembly's data types.
- It works in conjunction with tools like Binaryen and wasm-opt to produce compact Wasm modules.

AssemblyScript offers a "soft transition" to WebAssembly, particularly for developers coming from the JavaScript world, and there are real-world use cases such as accelerating hash functions in tools like Webpack.

6.4. TinyGo and the Go Ecosystem

TinyGo is an optimized alternative Go compiler for embedded systems and WebAssembly. It can produce significantly smaller .wasm files (e.g., a few hundred KB compared to the classic Go compiler). It enables the creation of Wasm components that operate on the server/edge side with the WASI target.

6.5. .NET ve Blazor WebAssembly .NET and Blazor WebAssembly

Microsoft's Blazor WebAssembly framework enables running .NET code written in C# in the browser via WebAssembly.

- A .NET runtime and application code downloaded to the browser are executed in Wasm format.
- UI components are defined using Razor/HTML, while event handling and business logic are implemented in C#.
- .NET WebAssembly build tools are based on ***Emscripten*** and provide AOT compilation support.

This approach facilitates the entry of not only JavaScript but also C# developers into the world of WebAssembly.

7. PERFORMANCE ANALYSIS AND COMPARISON WITH JAVASCRIPT

7.1. Performance Advantages of WebAssembly

The study by Haas and colleagues demonstrates that elements such as:

- Compact binary format,
- Single-pass validation,
- Efficient JIT/AOT compilation

can provide significant speed advantages over JavaScript, especially in numerically intensive (CPU-bound) tasks. (Haas et al., 2017).

Research by Yan and colleagues examining the performance of WebAssembly applications reports that, across various benchmark sets, WebAssembly is faster than JavaScript in most scenarios. However, it also indicates that optimizations can sometimes lead to unexpected results. (Yan et al., 2021).

The performance gains achieved through the use of WebAssembly can be summarized as follows:

- In loop-based computations, large matrix operations, and cryptography, WebAssembly can typically provide several times the speedup.
- Because the code size is smaller, download and load times are reduced.

7.2. Strengths of JavaScript

Despite the advantages of WebAssembly, there are still areas where JavaScript remains very strong. Features such as DOM manipulation, event handling, UI management, dynamic data structures, and reflection, as well as the NPM ecosystem and mature libraries, are aspects in which JavaScript can be considered superior.

Therefore, many real-world applications employ a hybrid architecture that uses both "JS + Wasm" together. A significant portion of the logic and UI is handled on the JavaScript side, while core computation libraries are maintained on the WebAssembly side.

7.3. SIMD, Multithreading, and WebAssembly 2.0

The WebAssembly 2.0 draft and ongoing efforts aim to standardize features to enhance performance, including:

- SIMD instructions,
- Multithreading (threads) and atomic memory operations,
- Multiple return values,
- Reference types and GC integration (Webassembly, 2025).

These capabilities will make WebAssembly significantly more attractive in the future for fields such as image processing, machine learning, and scientific computing.

8. SECURITY MODEL

8.1. Sandbox Execution and Memory Safety

WebAssembly adopts the following principles for secure execution:

- Code runs within a sandbox and cannot directly access real operating system resources.
- Memory accesses are controlled within linear memory boundaries; accessing an invalid address results as a "trap".
- The type system prevents passing parameters of incorrect types to functions.

These features create a natural barrier against classic buffer overflow and most memory corruption attacks.

8.2. Speculative Execution and Side-Channel Attacks

Speculative execution attacks, such as Spectre, affect not only JavaScript in the browser environment but also WebAssembly. McIlroy et al. have highlighted that speculative side-channel attacks should be examined from the perspective of programming languages, pointing out that classical abstract machine models do not account for these threats. They emphasize that such threats need to be considered during the design phase for new languages like WebAssembly. (Mcilroy et al., 2019).

Narayan and others' Swivel project provides a compiler-based framework that hardens WebAssembly code against Spectre attacks. (Narayan et al., 2021).

8.3. WebAssembly and Security in the Real World

Musch and colleagues, in their study of the Alexa Top 1M websites, found that a significant portion of sites using WebAssembly engaged in malicious activities such as cryptocurrency mining and obfuscation (Musch, Wressnegger, Johns, & Rieck, 2019). The study by Hilbig and colleagues, which analyzed 8,461 real-world WebAssembly binaries, reveals the diversity of use cases and indicates that a significant portion of the security vulnerabilities still stems from inherited C/C++ source code. (Hilbig et al., 2021).

A comprehensive security survey from 2024 indicates that static and dynamic analysis tools for WebAssembly are rapidly evolving, particularly highlighting the increasing adoption of Wasm in smart contracts and blockchain environments (Perrone & Romano, 2024).

The 2025 study "Wemby's Web" demonstrates that data read from linear memory is being transferred to security-critical locations in many sites without sufficient validation, potentially giving rise to new attack vectors (Draissi et al., 2025).

9. OUT-OF-BROWSER WEBASSEMBLY: WASI, SERVER, AND EDGE SCENARIOS

9.1. WebAssembly System Interface (WASI)

WASI is a POSIX-like system interface standard for WebAssembly. Its goal is to provide access to essential system services in Wasm modules, including:

- File system,
- Time and randomness,
- Standard input/output,
- Socket access

With WASI, WebAssembly becomes a general-purpose execution environment outside the browser, enabling its use in server environments, edge platforms, and embedded systems (Perrone & Romano, 2024).

When looking at server-side and edge platforms, services like Cloudflare Workers, Fastly Compute@Edge, and similar platforms utilize WebAssembly as a lightweight isolation layer to execute functions with millisecond-scale startup times. This approach offers several advantages over traditional containers or VMs, including:

- Significantly faster "cold start" times
- Lower memory footprint
- Language independence (any language that can be compiled to Wasm)
- Strong sandboxing

The small code size and portability of WebAssembly have made it attractive for embedded systems and IoT devices as well. Compilers like TinyGo and recent surveys indicate that WebAssembly can be utilized as a software-based security layer even on hardware that does not provide memory isolation. (Orlando et al., 2025).

10. APPLICATION AREAS

10.1. Games and Graphics Applications

Game engines like Unity and Unreal Engine can export games to the browser using a combination of WebGL and WebAssembly. Thanks to tools like Emscripten, the following can operate on the web with high frame rates:

- 3D games
- Physics simulations
- Visual editors

10.2. Scientific and Numerical Computing

WebAssembly can transform the browser into a lightweight scientific computing environment for CPU-intensive tasks such as large matrix multiplications, linear algebra, and statistical simulations. In recent years, there has been an increasing number of projects using the Rust+Wasm combination in web-based data visualization and analysis tools.

10.3. Cryptography and Security Software

Porting cryptographic libraries to WebAssembly:

- Enhances performance,
- Allows for the processing of sensitive data on the client side,
- May reduce server load in some case.

However, special precautions must be taken against speculative execution and side-channel attacks; research projects like Swivel(Narayan et al., 2021) and Wasm-Mutate(Cabrera-Arteaga et al., 2024) provide significant contributions in this area.

10.4. Blockchain and Smart Contracts

Many next-generation blockchain platforms prefer WebAssembly as a smart contract execution environment due to its language independence, formal semantics, and the guarantee of safe execution within a sandbox. (Perrone & Romano, 2024).

10.5. Porting Legacy Desktop Applications to the Web

Compiling long-standing desktop libraries and applications written in C/C++ to WebAssembly has led to the emergence of installation-free, platform-independent web versions. Notable examples of this approach include Google Earth, various CAD/graphics applications, and retro game emulators.

11. LIMITATIONS AND CHALLENGES

11.1. Toolchain Complexity

While tools such as Emscripten, wasm-bindgen, wasm-pack, the Blazor toolchain, and the AssemblyScript compiler are powerful, they can appear complex, especially for beginners. The installation of the toolchain (including LLVM, Node.js, Python, etc.) and platform-specific configurations add an additional burden.

11.2. Debugging

Although source map support and browser developer tools for WebAssembly have improved, line-by-line tracking of optimized binary code is more challenging compared to JavaScript. This situation can adversely affect the experience of developers, particularly in complex Rust/C++ projects.

11.3. Security-Related Limitations

While the sandbox model offers security advantages, it also presents challenges such as:

- The inability to make direct system calls,
- Dependency on WASI or host functions for file system and network access

which can necessitate additional architectural layers for certain types of applications, making installation and deployment more cumbersome.

11.4. Risks in Real-World Usage

The misuse of WebAssembly (such as cryptocurrency mining, obfuscation, exploit kits, etc.) and inadequate security analysis of modules create new attack surfaces on the browser side. Large-scale analyses have shown that a significant portion of sites using Wasm exhibit weak security practices. (Musch et al., 2019).

12. FUTURE PERSPECTIVE

12.1. WebAssembly 2.0 ve 3.0

The W3C and the WebAssembly community continue to expand the core specifications with versions 2.0 and 3.0. Features such as SIMD, reference types, tail calls, exception handling, and GC integration will enhance both performance and language compatibility. (Webassembly, 2025).

12.2. Component Model and Modularity

The developing Component Model aims to enable different Wasm modules and languages to work together in a type-safe and versionable manner. This will allow developers to build large systems from small, reusable Wasm components (Haas et al., 2017).

12.3. Industry Perspective

A 2025 industry-focused study emphasizes that WebAssembly is increasingly adopted, particularly in the fields of gaming, video processing, data analysis, and fintech. However, it also highlights the ongoing need for

improvements in debugging, security, and ecosystem maturity. (Ghosh et al., 2018).

13. CONCLUSION

WebAssembly is one of the paradigm-shifting technologies in the web and the broader software world. By providing a low-level yet secure binary format and a formally defined virtual machine model, it:

- Enables high-performance applications on the web,
- Provides a lightweight isolation layer in server and edge environments,
- Serves as a common target for numerous programming languages.

Both academic research and industrial use cases demonstrate that WebAssembly is not merely a "browser optimization" but a concrete representation of the vision for a portable execution layer.

REFERENCES

Cabrera-Arteaga, J., Fitzgerald, N., Monperrus, M., & Baudry, B. (2024). WASM-MUTATE: Fast and effective binary diversification for WebAssembly. *Computers and Security*, *139*(January), 103731. doi:10.1016/j.cose.2024.103731

Draissi, O., Cloosters, T., Klein, D., Rodler, M., Musch, M., Johns, M., & Davi, L. (2025). Wemby's Web: Hunting for Memory Corruption in WebAssembly. *Proceedings of the ACM on Software Engineering*, *2*(ISSTA), 1326–1349. doi:10.1145/3728937

Emscripten, (2021), APIs, Accessed on 12/01/2025 at the URL https://emscripten.org/

Ghosh, T., Debnath, A., Paul, A., Chattopadhyay, C., Hazra, S., & Singh, S. K. (2018). Hybrid Routing Approach Depending on Different Message Types in VANET, *3*(5), 335–338.

Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., … Bastien, J. F. (2017). Bringing the web up to speed with WebAssembly. *ACM SIGPLAN Notices*, *52*(6), 185–200. doi:10.1145/3062341.3062363

Hilbig, A., Lehmann, D., & Pradel, M. (2021). An empirical study of real-world webassembly binaries: Security, languages, use cases. *The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021*, 2696–2708. doi:10.1145/3442381.3450138

Khomtchouk, B. B. (2021). WebAssembly enables low latency interoperable augmented and virtual reality software, 1–11. Tarihinde adresinden erişildi http://arxiv.org/abs/2110.07128

Mcilroy, R., Sevcik, J., Tebbi, T., Titzer, B. L., & Verwaest, T. (2019). Spectre is here to stay: An analysis of side-channels and speculative execution, 1–26. Tarihinde adresinden erişildi http://arxiv.org/abs/1902.05178

Musch, M., Wressnegger, C., Johns, M., & Rieck, K. (2019). New kid on the web: A study on the prevalence of webassembly in the wild. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11543 LNCS*, 23–42. doi:10.1007/978-3-030-22038-9_2

Narayan, S., Disselkoen, C., Moghimi, D., Cauligi, S., Johnson, E., Gang, Z., … Stefan, D. (2021). Swivel: Hardening WebAssembly against spectre. *Proceedings of the 30th USENIX Security Symposium*, 1433–1450.

Orlando, T., D'Agati, L., Longo, F., & Merlino, G. (2025). A Survey of WebAssembly Usage for Embedded Applications: Safety and Portability Considerations. *CEUR Workshop Proceedings*, *3962*.

Perrone, G., & Romano, S. Pietro. (2024). WebAssembly and security: A review. *Computer Science Review*, *56*. doi:10.1016/j.cosrev.2025.100728

Tufegdžić, J., Dodović, M., Ogrizović, M., Babić, N., Dukić, J., & Drašković, D. (2024). Application of WebAssembly Technology in High-Performance Web Applications. *Proceedings - 2024 11th International Conference on Electrical, Electronic and Computing Engineering, IcETRAN 2024*, (June), 3–6. doi:10.1109/IcETRAN62308.2024.10645198

Wallentowitz, S., Kersting, B., & Dumitriu, D. M. (2022). Potential of WebAssembly for Embedded Systems. *2022 11th Mediterranean Conference on Embedded Computing, MECO 2022*, 1–4. doi:10.1109/MECO55406.2022.9797106

Webassembly, (2025), webassembly.org, Accessed on 12/01/2025 at the URL https://webassembly.org/specs/

Wikipedia, (2017), WebAssembly, Accessed on 12/01/2025 at the URL https://en.wikipedia.org/wiki/WebAssembly

Yan, Y., Tu, T., Zhao, L., Zhou, Y., & Wang, W. (2021). *Understanding the performance of webassembly applications*. Içinde *IMC '21: Proceedings of the 21st ACM Internet Measurement Conference* (ss. 533–549). doi:10.1145/3772356.3772396

Chapter 8

Machine Learning Regression Models: Methods and Application in Insurance Cost Prediction

Murat BİNİCİ[1]

ABSTRACT

This chapter presents an empirical study on the use of machine learning–based regression models to predict health insurance costs. The analysis draws on the Medical Cost Personal Dataset, which includes demographic and behavioral variables such as age, BMI, smoking status, number of children, sex, and region. After standard preprocessing steps—including encoding of categorical variables and an 80–20 train–test split—three regression models were implemented: Linear Regression, Random Forest Regressor, and XGBoost Regressor. Model performance was assessed using five evaluation metrics (R^2, MAE, MSE, RMSE, and MAPE). The findings show that ensemble methods outperform the linear model, with the Random Forest Regressor achieving the highest predictive accuracy and the lowest error measures. XGBoost also demonstrated strong performance, especially for observations with higher cost values, while Linear Regression struggled to capture nonlinear patterns inherent in the dataset. Feature importance analyses confirmed that smoking status is the dominant predictor across all models, followed by BMI and age. Overall, the results highlight the effectiveness of ensemble-based machine learning approaches in modeling complex and nonlinear relationships in insurance cost prediction, while also recognizing the continued value of Linear Regression in contexts where interpretability remains essential.

Keywords: Health insurance, Cost prediction, Machine learning, Linear regression, Random forest, XGBoost

[1] Assist. Prof. Dr., Bitlis Eren University, Faculty of Engineering and Architecture, Department of Mechanical Engineering, mbinici@beu.edu.tr, ORCID: 0000-0003-1814-438X.

1. INTRODUCTION

In recent years, Machine Learning (ML) and Artificial Intelligence (AI) techniques have played a growing role in data analytics and decision-support systems, and their influence continues to grow. In today's data-driven world, organizations increasingly rely on AI-driven approaches to extract valuable information from complex and high-dimensional datasets, going beyond the capabilities of traditional statistical methods. These techniques not only learn from historical information but also help anticipate future trends, offering institutions a strategic advantage in their decision-making processes.

Within the broader field of data analytics, regression models serve as one of the key components of predictive analytics. Regression analysis aims to mathematically describe the relationship between independent variables and a dependent variable. By doing so, it becomes possible to estimate the future value of an outcome based on past observations. In domains such as healthcare, finance, and insurance, regression-based cost, risk, or demand forecasting can significantly enhance the effectiveness of AI-supported decision-making systems.

The insurance industry is one of the fields in which predictive modeling is used most extensively, largely due to its strong emphasis on risk assessment and cost estimation. Setting insurance premiums accurately forms the basis of a fair, sustainable, and profitable insurance system for both individuals and institutions (Ivanovna et al., 2018). For this reason, it is essential to model the influence of factors such as age, gender, body mass index (BMI), smoking habits, and family structure on premium levels in a reliable way. Although traditional linear regression models have shown some success in explaining these relationships, AI-driven algorithms such as Random Forest and XGBoost have gained prominence in recent years for their ability to capture nonlinear patterns and deliver higher predictive accuracy (Kapse et al., 2025; Mishra et al., 2024).

Previous research has extensively explored the prediction of health insurance costs. While some of these studies rely on classical linear regression models, the growing volume of data and increasing computational power in recent years have encouraged the use of more advanced machine learning techniques. For instance, Panda et al. (2022) compared Lasso, Ridge, Simple Linear, Polynomial Regression and Multiple Linear models in estimating health insurance premiums, reporting that the polynomial regression model achieved both the lowest error rate (RMSE = 5100.53) and the highest explanatory power ($R^2 = 0.80$).

Similarly, Kaushik et al. (2022) proposed a comprehensive framework for the prediction of health insurance premiums employing a ML-based regression

approach. Their framework incorporates not only model performance, but also data preprocessing, hyperparameter tuning and feature selection. The study demonstrates that the accuracy of regression models largely rely on the quality of data cleaning, the relevance of selected variables, and the choice of an appropriate combination of models.

Bhardwaj and Anand (2020) compared Multiple Linear Regression, Gradient Boosting and Decision Tree algorithms using individual health data, reporting that the Gradient Boosting model succeeded the highest accuracy, with a rate of 99.5%. In a more recent study, Bader and Maalouf (2024) analyzed the determinants of health insurance premiums by applying Multiple Linear Regression, Lasso, Ridge and Support Vector Regression (SVR) models, and found that the SVR approach produced the lowest error level (RMSE = 0.84).

As these studies demonstrate, both linear and nonlinear approaches offer strong predictive capability in estimating health insurance costs. In addition, the framework and hybrid models developed in recent years have been used effectively not only for calculating individual premium levels, but also for examining regional variations, identifying risk groups, and optimizing policy pricing strategies. Consequently, AI- and machine-learning-based regression approaches enhance the accuracy of financial forecasting in the health insurance sector while also providing a data-driven perspective for policy development.

The purpose of this work is to determine the factors that affect insurance premiums and to collate the predictive performance of several ML-based regression models. In this context, Linear Regression, XGBoost Regressor and Random Forest Regressor models are implemented, and their outputs are evaluated. This approach allows for a clearer assessment of how effectively different machine learning techniques can model and forecast health insurance costs.

2. FOUNDATIONS OF REGRESSION AND MACHINE LEARNING

2.1. The Concept of Regression Analysis

Regression analysis is a fundamental method used in machine learning and statistical modeling to mathematically describe the relation between independent variables and a dependent variable. In its broadest sense, regression examines how a given variable (typically denoted as Y) is influenced by other variables (X_1, X_2, …, X_n) and models this relationship quantitatively to enable prediction. The main objective of regression is to identify the structural relationship among variables and to determine the function that best explains this association (Montgomery et al., 2021).

Regression models can take various forms relying on the structure of the dataset and the nature of the relationships among variables. Simple linear regression analyzes the impact of a single independent variable on a dependent variable. For example, attempting to explain insurance premiums solely through an individual's age would fall under this type of model. The general form of the model can be expressed as in Eq. 1.

$$Y = \beta_0 + \beta_1 X + \varepsilon \tag{1}$$

In equation (1), Y represents the dependent variable, X denotes the independent variable, β_0 is the intercept term, and β_1 refers to the regression coefficient. The term ε captures the random error component that the model is unable to explain.

In a multiple linear regression model, the effects of independent variables on a dependent variable are examined simultaneously. Such models are widely utilized in areas like the social sciences, engineering, economics, and health insurance analysis. For instance, multiple linear regression is applied to understand how age, smoking status, body mass index (BMI) and regional factors collectively influence insurance premiums. The model is typically expressed using Eq. 2.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + .. + \beta_n X_n + \varepsilon \tag{2}$$

Eq. 2 assumes a linear relationship among the variables, meaning that the effects of the independent variables are additive and constant. However, many relationships observed in real-world data are not linear. In such cases, nonlinear regression models become more appropriate.

Nonlinear regression models are utilized when the relations between variables takes a complex, curved, or exponential form. For example, the rapid increase in healthcare expenditures after a certain age threshold or the nonlinear impact of smoking on insurance premiums are situations in which such models are appropriate. In these cases, the model can be expressed as in Eq. 3.

$$Y = f(X_1, X_2 \ldots, X_n;\ \beta_1, \beta_2, \ldots, \beta_n) + \varepsilon \tag{3}$$

In Eq. 3, f denotes a nonlinear function, which may take forms such as logarithmic, exponential, or polynomial. Although nonlinear models often provide a better fit to the data, they tend to be more complex in terms of interpretability (Kutner et al., 2005).

In sum, regression analysis not only quantifies the relationships among variables but also serves as a powerful tool in predictive analytics and decision-support systems. With advances in machine learning and artificial intelligence, the notion of regression has moved beyond traditional statistical approaches and has become a central component of data-driven modeling.

Machine learning (ML) methods are generally grouped into two main areas: unsupervised learning and supervised. Supervised learning refers to an approach in which the model learns from previously labeled data and uses this knowledge to make predictions for new observations. In such models, the system identifies patterns between the inputs and output, enabling it to generate forecasts for similar data in the future (James et al., 2021).

Regression analysis is one of the most fundamental forms of supervised learning, as regression models enable the prediction of a continuous (numerical) outcome based on historical data. The supervised learning process typically consists of three main stages:

a. **Training the model:** The model is taught the relationship among the variables using labeled data.

b. **Validating the model:** The model's capability to generalize is assessed, and the risk of overfitting is evaluated.

c. **Making predictions:** The model's predictive capability is tested on new or previously unseen data.

Regression models, which lie at the core of this process, are among the most suitable approaches when the outcome of interest is continuous—such as price, cost, income, temperature, or production rate. In contexts like the insurance industry, where problems such as premium estimation or cost analysis involve a continuous dependent variable, regression techniques can be applied directly and effectively.

Within the supervised learning framework, regression models represent a key area in which methods that originated in traditional statistics have evolved into AI-driven approaches. While classical linear regression explains the relationship among variables under a set of assumptions, modern machine learning regression algorithms learn these relationships from the data without depending on such assumptions. This capability is particularly advantageous for nonlinear or high-dimensional datasets, where it often leads to substantially higher predictive accuracy (Hastie et al., 2017).

For example, algorithms such as XGBoost and Random Forest Regressors preserve the statistical foundation of classical regression while adding the flexibility of artificial intelligence to the learning process. These models perform strongly in identifying complex patterns and capturing interactions

among variables, particularly in large datasets. In this sense, regression analysis can be viewed not only as a statistical prediction tool but also as an AI-driven predictive core within supervised learning.

At the core of regression analysis lies the mathematical modeling of the relations between independent variables and a dependent variable. The dependent variable represents the outcome of interest or the value to be predicted, whereas the independent variables capture the factors that influence or help explain that outcome (Gujarati and Porter, 2009). Accordingly, the purpose of regression is to determine as accurately as possible how variations in the independent variables are effective on the dependent variable.

This relationship is typically expressed in a functional form as in Eq. 4.

$$Y = f(X_1, X_2, \ldots, X_n) + \varepsilon \tag{4}$$

In Eq. 4, Y denotes the dependent variable, while X_1, X_2, ..., X_n represent the independent variables. The function $f(\cdot)$ captures the systematic relationship among these variables, and ε stands for the random error term that the model cannot explain. The form of this function varies relying on the type of regression method being employed. For instance, linear regression assumes a linear relationship, whereas nonlinear or AI-based models allow this relationship to take more complex, curved, or interaction-driven forms.

Selecting and modeling independent variables appropriately is crucial for the reliability of regression analysis. Variable selection directly affects both the explanatory power of the method (R^2) and its capability to generalize (Guyon and Elisseeff, 2003). Including unnecessary or highly correlated variables may lead to multicollinearity, which can undermine the significance of coefficients. For this reason, relationships among variables must be examined carefully, particularly in multiple regression settings.

AI-based regression models offer a significant advantage at this point. Models such as XGBoost and Random Forest can automatically identify which independent variables contribute most to explaining the dependent variable. Through feature-importance calculations, these models quantify the relative influence of each predictor (Lundberg and Lee, 2017). This capability makes it easier to understand complex interactions that are often difficult to interpret in classical regression analysis.

For instance, in a model designed to predict insurance premiums, the dependent variable may be charges, while the independent variables could include factors such as age, bmi, children, smoker, and region. In such a case, an AI-based model can automatically identify smoker as the most influential

factor in predicting insurance costs. This not only enhances predictive accuracy but also improves the interpretability of variable effects.

In conclusion, accurately modeling the relationship between independent variables and dependent is a critical determinant of the effectiveness of AI-based regression approaches. Models that capture this relationship appropriately not only give highly correct predictions but also provide meaningful information for decision-support systems.

2.2. Linear Regression

Linear regression is a fundamental approach that models the expected value of a dependent variable as a linear combination of explanatory variables and their associated parameters. Its general form can be expressed by using Eq. 5.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \varepsilon \tag{5}$$

Here, Y denotes the dependent variable, X_n represents the explanatory variables, β_n refers to the coefficients, and ε stands for the error term. The parameters are calculated utilizing Ordinary Least Squares (OLS), which minimizes the sum of squared distictions between the predicted and actual values (Eq. 6).

$$min \sum_{i=1}^{n} (y_i - Y_i)^2 \tag{6}$$

The reliability of the model depends on the reasonable fulfillment of several standard assumptions: (i) Linearity, meaning the expected value of the dependent variable is a linear combination of the predictors; (ii) Homoscedasticity, which requires the error variance to remain constant; and (iii) Absence of multicollinearity, indicating that the explanatory variables are not perfectly or strongly linearly related.

In practice, these assumptions are examined using simple diagnostic tools such as residual plots, Breusch–Pagan or White tests, and the Variance Inflation Factor (VIF). When necessary, variable transformations, weighted least squares, or regularization techniques (such as Ridge or Lasso) may be applied (James et al., 2021).

The interpretation of the coefficients is simple: β_n demonstrates the marginal impact of a one-unit increase in X_n on Y, holding all other variables constant. In log-transformed models, this effect is often interpreted as an approximate percentage change.

Linear regression is strong in terms of interpretability, computational efficiency, and its role as a fundamental baseline model. However, its performance may weaken in the presence of nonlinear patterns, heteroskedasticity, outliers, or substantial multicollinearity. In such situations, transformations, regularization methods, or more flexible ones such as tree-based and boosting models are recommended as complementary analyses (Hastie et al., 2017).

2.3. Decision Trees and Ensemble Regression Methods

A Decision Tree Regressor models the data by repeatedly splitting it into two groups, choosing feature–threshold combinations that reduce prediction error (often measured by MSE). Each leaf of the tree returns a simple estimate, such as the mean value of the target variable. Overfitting is controlled by limiting the depth of the tree, requiring a minimum number of samples per leaf, or using pruning. Decision trees do not require feature scaling and are relatively easy to interpret thanks to their rule-based structure. However, a single tree can be highly variable, which is why ensemble approaches, such as random forests or boosting, are often preferred in practice (Blockeel, 2023).

Random Forest Regressor is a bagging model that trains many decision trees on bootstrap samples. It uses a randomly selected subset of features at each split. The predictions from all trees are then averaged. These sources of randomness reduce the correlation between trees, which in turn lowers the high variance typically seen in a single decision tree. Random forests can capture nonlinear relationships and interactions effectively, and they are generally robust to scaling issues and outliers. They also offer practical advantages such as the out-of-bag (OOB) error, which provides an internal estimate of generalization performance, and measures of variable importance based on impurity reduction or permutation. The overall performance largely depends on tuning hyperparameters such as the the number of features considered at each split (max_features), number of trees (n_estimators), minimum samples per leaf and tree depth. These choices help balance bias and variance depending on the data size and noise level (Probst et al., 2019).

XGBoost, or more generally the Gradient Boosting Regressor, builds a strong predictive model by adding weak learners sequentially and additively, each focused on reducing the residual errors of the previous steps. At every iteration, a shallow decision tree is trained to explain the remaining residuals. Techniques such as learning rate (shrinkage) and subsampling of rows and features help prevent overfitting, while early stopping is often used to track generalization performance (Chen and Guestrin, 2016).

XGBoost includes several engineering improvements, such as L1/L2 regularization, limits on tree depth and number of leaves, split criteria optimized for sparse data, automatic handling of missing values, and scalable memory-access patterns for large datasets. In practice, performance is driven by the joint tuning of number of trees, learning rate, maximum depth and subsampling ratios (Chen and Guestrin, 2016).

Feature importance shows which input variables matter most in a model. Tree-based models usually measure this by how much each split reduces error, while model-agnostic methods such as permutation importance or SHAP values offer alternative ways to check variable effects. However, highly correlated features may appear more important than they are. Using several importance measures together generally gives a more reliable understanding (Fisher et al., 2019).

2.4. Regression in the Context of Artificial Intelligence

In the AI context, regression goes beyond classical statistics by using algorithms that can learn nonlinear patterns and interactions in the data. Models like tree ensembles, gradient boosting, and regularized linear models often provide higher accuracy and better scalability. However, stronger models can be harder to interpret, so tools such as SHAP values, permutation importance, and solid validation methods are needed. In practice, AI-based regression is widely used from predicting health insurance premiums to estimating energy demand. It can produce useful insights when supported by good preprocessing, proper model choice, and reliable evaluation metrics (James et al., 2021).

3. MATERIALS AND METHODS

This section describes the dataset employed in the work, the variables it contains, and the preprocessing steps applied before model development. The analysis is based on the Medical Cost Personal Dataset obtained from Kaggle, which includes 1,338 observations and seven variables related to individual demographic and lifestyle characteristics. The section introduces the structure of the data, summarizes key attributes of numerical and categorical variables, and outlines the steps taken to prepare the dataset for modeling, including descriptive analysis and checks for missing values. These elements provide the foundation for building and comparing the regression models used in the study.

3.1. Dataset Overview

The dataset utilized in this study is the insurance.csv file obtained from the “Medical Cost Personal Datasets” resource on Kaggle (Abdelghany, 2025).

3.1.1. Variables

The dataset consists of 1,338 observations and includes seven variables: charges (target), age, sex, bmi, children, smoker, and region. The dataset is complete with no missing observations. Information about the variables is presented in Table 1.

The variable charges, which represents the insurance premium, is the target (dependent) variable and is continuous. The aim is to predict an individual's health insurance premium using the remaining six independent variables. The variable bmi indicates body mass index and is also continuous. The integer variables in the dataset are age and children. The age variable represents the individual's age, while children denotes the number of children they have. The sex variable identifies the individual's gender and is categorical. Two additional categorical variables are smoker and region. The first indicates whether the person is a smoker, and the second specifies the geographical region in which they live (northeast, northwest, southeast, southwest).

Table 1. Information about the variables

Variables	Definition	Type	Obs.
charges (target)	Health insurance premium (USD)	Continious	1338
age	Participant's age	Integer	1338
bmi	Body mass index (kg/m²)	Continious	1338
children	Number of dependent children	Integer	1338
sex	Participant's gender (female/male)	Categorical	1338
smoker	Smoking (yes/no)	Categorical	1338
region	Region of residence (northeast/northwest/southeast/southwest)	Categorical	1338

3.1.2. Descriptive statistics

The descriptive statistics for the numerical variables are given in Table 2, reporting the standard deviation, mean, maximum and minimum values for each variable.

Table 2. Descriptive statistics for the numeric variables

Variables	min	max	mean	Std.
charges (target)	1121.87	63770.43	13270.42	12110.01
age	18	64	39.21	14.05
bmi	15.96	53.13	30.66	6.10
children	0	5	1.09	1.21

The distribution of the categorical variable sex in the insurance dataset is presented in Table 3. Approximately half of the participants are male and the other half are female, indicating a well-balanced gender distribution. This balance reduces the risk of bias related to class imbalance when the gender variable is included in the modeling process.

Table 3. Statistics for the sex variable

Category	Observations	Percentage (%)
female	662	49.5 %
male	676	50.5 %
Total	1338	100 %

The statistics for smoker, another categorical variable, are presented in Table 4. Approximately 20.5% of the individuals in the dataset are smokers, while 79.5% are non-smokers. This distribution gives the smoker variable considerable discriminative power during modeling, as the average insurance cost for smokers is roughly 3.8 times higher than that of non-smokers.

Table 4. Statistics for the smoker variable

Category	Observations	Percentage (%)
no	1064	79.5 %
yes	274	20.5 %
Total	1338	100 %

Finally, the statistics for the categorical variable region are provided in Table 5. As shown, the dataset is regionally well balanced, with each of the four regions represented by a similar number of observations. This balance reduces the likelihood of regional bias during modeling and allows the effect of the region variable to be examined more reliably.

Table 5. Statistics for the region variable

Category	Observations	Percentage (%)
southeast	364	27.2 %
southwest	325	24.3 %
northwest	325	24.3 %
northeast	324	24.2 %
Total	1338	100 %

3.2. Data Preprocessing

Data preprocessing refers to the set of systematic transformation, cleaning, and standardization steps applied to raw data to prepare it for analysis and modeling. In other words, it aims to make sure that the input data employed by the model is correct, consistent, complete, and statistically meaningful (Kotsiantis et al., 2006).

3.2.1. Missing value analysis

Checking for missing values is a crucial step in data preprocessing, as it helps preserve data integrity and the predictive power of the method. In this stage, missing observations are systematically identified, their proportions are calculated, and the nature of the missingness is evaluated. Simple statistical functions such as isnull() or isna() support this analysis. When the amount of missing data is low, basic imputation methods such as replacing values with the mode, median, or mean may be sufficient. For higher levels of missingness, regression-based approaches, KNN imputation, or multiple imputation (MICE) techniques are recommended. Importantly, imputation must be performed only on the training data, with the same transformation applied to the test data, to avoid data leakage. The goal of this process is to prevent missingness from introducing bias into the model, reduce information loss, and improve predictive performance (Little & Rubin, 2019; Jadhav et al., 2019).

In the *insurance* dataset used in this study, there are no missing observations. The checks conducted on the dataset show that all seven variables are fully populated for all 1,338 records, with no NaN or empty values. This simplifies the preprocessing stage considerably and allows the analysis to move directly to model development.

3.2.2. Encoding of categorical variables

Transforming (encoding) categorical variables is an essential step in machine learning and regression analysis, as most models operate only on numerical inputs. This process converts text-based categories into statistically meaningful numerical representations. The three most common methods are Dummy Encoding, One-Hot Encoding and Label Encoding, each suited to different model types and data structures (Kuhn and Johnson, 2019; Potdar et al., 2017).

In *Dummy Encoding*, only K−1 columns are created from K categories. One category is left out as the reference (baseline). This prevents linear dependence among variables and helps avoid multicollinearity.

In *One-Hot Encoding*, a separate binary (0/1) column is created for each category. This removes any ordinal relationship among the classes and allows

the model to treat each category as an independent indicator variable. However, when a variable has many categories, the number of columns can increase rapidly, leading to a high-dimensional feature space. For this reason, one-hot encoding is commonly used with methods such as linear regression, SVM, and neural networks.

In *Label Encoding*, a unique integer value is assigned to every category (such as female = 0, male = 1). Although this method is simple and efficient, it introduces an artificial ordinal relationship among categories. Therefore, it is more appropriate for models that are insensitive to ordering such as tree-based algorithms like XGBoost, Random Forest, or Decision Tree.

For example, suppose the region variable contains four categories (northeast, northwest, southeast, southwest). In that case, the encodings would be as follows:

Table 6. Examples of categorical variable coding

Observations	Region	Dummy Encoding (Ref: northeast)	One-Hot Encoding	Label Encoding
1	southwest	(0, 0, 1)	(0, 0, 0, 1)	0
2	southeast	(0, 1, 0)	(0, 0, 1, 0)	1
3	northwest	(1, 0, 0)	(0, 1, 0, 0)	2
4	northeast	(0, 0, 0)	(1, 0, 0, 0)	3

In this example, Label Encoding introduces an artificial ordering among the categories, while One-Hot Encoding represents all classes. Dummy Encoding, on the other hand, includes all categories except the reference category (northeast).

In the insurance dataset used in this study:

- Since the variables sex and smoker are binary, they were encoded using Label Encoding (0–1),
- Since the region variable contains four categories, Dummy Encoding was applied, with northeast selected as the reference category.

This approach may help avoid linear dependence in linear regression models while also providing an effective representation of categorical information for tree-based algorithms.

3.2.3. Scaling numerical variables

Feature scaling is a preprocessing technique used to eliminate imbalances arising from differences in magnitude or measurement units among numerical variables. It ensures that the model's learning process proceeds properly, especially when variables span different value ranges. Scaling helps stabilize

parameter estimation, improves the convergence speed of gradient-based algorithms, and enhances overall generalization performance (Han et al., 2012; Kuhn and Johnson, 2019).

Some of the scaling methods generally utilized in machine learning are as follows:

Min–Max Normalization: The data is scaled to the range of 0–1 by using Eq. 7. It is frequently used in gradient-based models such as neural networks and logistic regression, but it is sensitive to outliers.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{7}$$

Standardization (Z-score Scaling): The data is transformed by using Eq. 8 so that the standard deviation becomes 1, and the mean is 0. This approach is more resilient to outlier effects and is a standard approach in methods such as SVM, linear regression, and PCA.

$$x' = \frac{x - \mu}{\sigma} \tag{8}$$

Robust Scaling: It uses the median and interquartile range (IQR), as stated in Eq. 9. It is preferred when the dataset contains many outliers. This method is especially suitable for variables such as income, expenditure, or health costs, which tend to have extreme values.

$$x' = \frac{x - median}{IQR} \tag{9}$$

The three models used in this study have different characteristics in terms of their need for feature scaling:

Linear Regression: Linear regression coefficients are directly affected by the scale of the variables. Features with larger magnitudes tend to dominate the model. Therefore, standardization (z-score scaling) is recommended when using linear regression (Kuhn and Johnson, 2019).

Random Forest Regressor: As a tree-based model, it splits data according to the ranking of feature values, not their actual scales. As a result, Random Forest is not affected by scaling and does not require any feature scaling (Lantz, 2019).

XGBoost Regressor: XGBoost is also a tree-based algorithm and is therefore insensitive to differences in feature scales. However, in some cases, normalizing the variables can improve numerical stability and speed up convergence.

In the *insurance* dataset used in this study, the numerical variables (age, bmi, children, charges) are measured on different scales; however, given the structure of the models, extensive scaling is not required. Random Forest and XGBoost are insensitive to feature scale differences. Although Linear Regression can theoretically benefit from scaling, the small number of independent variables and their relatively comparable statistical ranges mean that scaling would not provide a meaningful improvement in model performance. Therefore, no scaling procedures were applied in this study. The variables were used in their original form, preserving their natural interpretability and avoiding unnecessary transformations.

3.2.4. Training and test-data split

Train–test splitting is a fundamental validation method employed in ML to appraise a model's ability to generalize. This approach divides the available dataset into two subsets:

- Training dataset: It is used during the learning phase of the model to estimate its parameters.
- Testing dataset: It consists of data the model has never seen before and is utilized to appraise its actual performance.

This split helps prevent the method from overfitting to the training data and allows its predictive power on new data to be evaluated objectively (Han et al., 2012).

Commonly used train–test ratios are 70–30 or 80–20. In larger datasets, the proportion allocated to training can be increased, while smaller datasets may require a slightly higher test ratio. In some cases, a third subset called the validation set is also created, or the training process is stabilized using methods such as k-fold cross-validation (Kuhn and Johnson, 2019).

The dataset employed in this work has 1,338 observations. To objectively appraise the overall performance of the methods, the dataset was separated into two parts: 20% for testing and 80% for training. The training set (1,070 observations) was used during the learning phase of the XGBoost, Random Forest and Linear Regression models. The test set (268 observations) was used to determine how well the methods perform on unseen data. This ratio is appropriate for the size of the dataset, providing sufficient data for model training while allowing reliable evaluation on the test set.

The split was performed randomly (random_state = 42), ensuring reproducibility. This approach allowed each model to see only a portion of the data during training and to be assessed on observations it had not encountered before. As a result, the risk of overfitting was reduced and the models' ability to

generalize was improved. The performance of all three methods on the test data was evaluated using R^2, MAE, MSE, RMSE, and MAPE.

3.3. Applied Machine Learning Methods

This section introduces the three ML models used to predict the insurance premium (charges) variable. The modeling process includes XGBoost Regressor, Random Forest Regressor and Linear Regression in order to capture both nonlinear and linear relationships in the data. These models were selected because they are capable of explaining potential linear effects as well as interaction patterns among the variables in the dataset.

All data preprocessing and modeling steps in this work were carried out on the Google Colab platform. Colab is a cloud-based development environment that allows effective use of Python data science libraries. This made the analysis process both computationally efficient and reproducible. During model development, the pandas, NumPy, scikit-learn, and XGBoost libraries were used, and all coding was performed in Python.

3.3.1. Linear regression

Linear regression is a well-established statistical approach employed to describe the linear association between independent variables and a dependent. In this study, it is assumed by the model that the target variable, charges, can be represented as a linear combination of the predictors, which are age, bmi, children, sex, smoker, and region. The general structure of the model can be written as in Eq. 10.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \varepsilon \tag{10}$$

In Eq. 10, y represents the dependent variable, x_i denotes the independent variables, β_i refers to the coefficients, and ε is the error term. The goal of the model is to determine the coefficients minimizing the sum of squared distinctions between the actual and predicted values, commonly measured by the mean squared error (MSE) (Han et al., 2012).

The key advantages of linear regression lie in its straightforward structure and the ease with which it can be interpreted. Each coefficient reflects the expected change in the dependent variable associated with a one-unit change in the corresponding predictor, holding the other variables constant. However, when the core assumptions of the method (linearity, normality of the errors, homoscedasticity, and independence) are violated, the predictive performance of the model may weaken (Kuhn and Johnson, 2019).

In the *insurance* dataset, the categorical variables sex and smoker were encoded using Label Encoding since both have a binary structure. The region variable, which contains four categories, was transformed using Dummy Encoding, with northeast set as the reference group. The encoded dataset was subsequently split into two parts, with 80% allocated for training and the remaining 20% reserved for testing. The model was fitted using the training portion of the data and its performance was assessed on the test subset. Its predictive performance was assessed through R^2, MAE, MSE, RMSE, and MAPE. This analysis served as a baseline model for identifying the general linear patterns within the data.

3.3.2. Random forest regressor

The Random Forest Regressor is an ensemble technique that relies on multiple decision trees. Instead of depending on a single deep tree, which is prone to overfitting, the method constructs numerous trees using different bootstrap samples of the data and combines their predictions by averaging them to obtain the final estimate. This approach lowers variance and helps generate more stable and generalizable predictions.

The Random Forest algorithm incorporates two main forms of randomness: (i) bootstrap sampling, which ensures that each tree is built using a different portion of the data, and (ii) the random selection of feature subsets at each splitting step. Together, these elements enhance the diversity of the trees in the ensemble and help lower the likelihood of overfitting.

In this work, the Random Forest Regressor was implemented using the *insurance.csv* dataset. The dependent variable, charges (insurance cost), was modeled using the predictors age, bmi, children, sex, smoker, and region. The model was configured to use 500 trees (n_estimators = 500). Since sex and smoker are binary categorical variables, they were encoded using Label Encoding (0–1). The region variable, which includes four categories, was transformed using Dummy Encoding with northeast selected as the reference category. The dataset was divided into an 80% training and a 20% testing portion, and the model's performance was assessed using the metrics MAPE, MAE, RMSE, MSE, and R^2.

3.3.3. XGBoost regressor

XGBoost (Extreme Gradient Boosting) is an algorithm that constructs decision trees in a sequential manner, reducing prediction errors through the gradient boosting process. Each successive tree is designed to address the

mistakes of the preceding ones, leading to progressively improved predictive accuracy (Chen and Guestrin, 2016).

Compared with traditional Gradient Boosting approaches, the key distinction of XGBoost is its use of a regularized objective function. This structure incorporates both L1 (Lasso) and L2 (Ridge) penalty terms, making the model more resistant to overfitting. In addition, XGBoost provides high computational efficiency through features such as built-in handling of missing values, parallel processing, histogram-based tree growth, and early stopping.

In this study, the XGBoost Regressor model was configured with 500 trees (n_estimators = 500) for predicting insurance premiums. During preprocessing, the variables sex and smoker were converted into binary form using Label Encoding, while the region variable with four categories was transformed using Dummy Encoding, with northeast defined as the reference category. The dataset was then divided into a 20% test set ann an 80% training set. The XGBoost model was trained on the training data using a gradient boosting approach, and its performance was assessed utilizing R^2, MAE, MSE, RMSE, and MAPE.

3.4. Evaluation Metrics

To compare the performance of the ML methods and evaluate their predictive accuracy, several statistical metrics are commonly used. In this study, the performance of the regression models was assessed employing five key evaluation metrics: R^2, MAE, MSE, RMSE, and MAPE. Taken together, these measures offer a thorough assessment of how large the model's errors are, how they are distributed, and how closely the predictions align with the actual values.

3.4.1. Coefficient of determination (R^2)

The R^2 shows the proportion of the variation in the dependent variable that is accounted for by the model and it can be calculated using Eq. 11. Its value ranges from 0 to 1, with figures closer to 1 indicating greater explanatory strength.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y}_i)^2} \tag{11}$$

Eq. 11 corresponds to the share of variance the model is unable to explain, whereas the denominator represents the total variance in the dependent variable. A larger R^2 value shows that the model fits the data more effectively. However, it should not be viewed as a standalone indicator of performance.

3.4.2. Mean absolute error (MAE)

MAE represents the mean of the absolute deviations between the model's predictions and the true values and it can be calculated using Eq. 12. It does not take the direction of the errors (positive or negative) into account and focuses solely on their magnitude.

$$MAE = \frac{1}{n}\sum|y_i - \hat{y}_i| \tag{12}$$

The interpretation of MAE is straightforward; its unit is the same as that of the dependent variable. Lower MAE scores suggest that the model's predictions align more closely with the observed values.

3.4.3. Mean squared error (MSE)

MSE is calculated as the mean of the squared prediction errors, meaning that larger deviations contribute disproportionately to the final value due to the squaring process. The formula is given in Eq. 13.

$$MSE = \frac{1}{n}\sum(y_i - \hat{y}_i)^2 \tag{13}$$

MSE is often used for comparing models, but because its unit is the square of the original variable, it is not as easy to interpret directly as MAE.

3.4.4. Root mean square error (RMSE)

RMSE is obtained by taking the square root of the MSE as in Eq. 14, which allows the error to be reported in the same unit as the dependent variable.

$$RMSE = \sqrt{MSE} \tag{14}$$

It is sensitive to large errors and provides an indication of the model's "typical error magnitude." A lower RMSE value shows that the model produces consistent and accurate predictions.

3.4.5. Mean absolute percentage error (MAPE)

MAPE is a performance measure that reports prediction errors in percentage terms. It can be calculated using Eq. 15.

$$MAPE = \frac{100}{n}\sum\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{15}$$

Expressing the result as a percentage makes interpretation easier. However, when the actual values are very close to zero, sensitivity issues may arise.

Collectively, these five metrics offer a well-rounded assessment of the model's overall performance. While R^2 measures the proportion of variance explained, MAE and RMSE capture the magnitude of errors with different levels of sensitivity. MAPE, on the other hand, expresses errors as percentages, offering an interpretation that is particularly intuitive for decision-makers. For this reason, the XGBoost Regressor, Random Forest Regressor and Linear Regression models used in this study were evaluated using this combined set of metrics.

4. FINDINGS AND EVALUATION

4.1. Findings of the Models

To assess how each model performed, the five primary evaluation metrics (R^2, MAE, RMSE, MSE, and MAPE) were computed for the XGBoost Regressor, Random Forest Regressor, Linear Regression models. Using these measures, each model's predictive accuracy, error magnitude, and generalization ability were analyzed comparatively. The resulting findings are summarized in Table 7.

Tablo 7. Comparative evaluation of the ML models

Model	R^2	MAE	MSE	RMSE	MAPE (%)
Linear Regression	0.7836	4181.19	33,596,915.85	5796.28	46.89
Random Forest Regressor	0.8640	2560.02	21052750.53	4588.33	32.48
XGBoost Regressor	0.8349	2921.27	25,633,697.49	5062.97	37.07

Table 7 provides a comparison of the performance indicators for the three ML methods. The findings indicate that the Random Forest Regressor achieves the best overall results, demonstrating the highest R^2 value (0.864), which shows that it captures a substantial share of the variance in the dependent variable. Additionally, its lower MAPE, RMSE, MSE, and MAE scores, relative to the other models, further highlight its ability to significantly reduce prediction errors.

XGBoost Regressor performs better than Linear Regression overall but remains slightly behind Random Forest. The Linear Regression model shows the lowest performance, largely because its inherently linear framework cannot represent the nonlinear and complex relationships embedded in the dataset.

Overall, these findings indicate that ensemble methods provide higher accuracy than classical linear models when predicting insurance premiums.

To provide a qualitative assessment of model performance, the predicted insurance premiums for five randomly selected observations from the test dataset were compared with their actual values. This approach complements the general evaluation provided by the error metrics and offers insight into how the models behave at the individual observation level. The closeness of the predictions to the true values is especially important for understanding model consistency and the practical significance of the errors. The randomly selected observations are presented in Table 8, and their corresponding comparisons are shown in Table 9.

Table 8. Randomly chosen observations from the *insurance* dataset

Obs. No	*age*	*sex*	*bmi*	*children*	*smoker*	*Region northwest*	*Region southeast*	*Region Southwest*
764	45	0	25.175	2	0	0	0	0
887	36	0	30.020	0	0	1	0	0
890	64	0	26.885	0	1	1	0	0
1293	46	1	25.745	3	0	1	0	0
259	19	1	31.920	0	1	1	0	0

Table 9. Actual and predicted insurance premiums for the random observations

Obs. No	Actual Charges	LR Prediction	LR Error	RF Prediction	RF Error	XGB Prediction	XGB Error
764	9095.07	8969.55	1.38 %	10282.31	13.06%	9336.78	2.66%
887	527218	7068.75	34.1%	5342.62	1.34%	9343.56	77.20%
890	29330.99	36858.41	25.7%	28331.83	3.41%	29616.47	0.97%
1293	9301.90	9454.67	1.64%	11353.21	22.05%	9575.83	2.95%
259	33750.30	26973.17	20.1%	34763.78	3.00%	33304.64	1.32%

Table 9 presents a comparison of actual and predicted insurance charges for five randomly selected observations across the three machine learning models. The results demonstrate that XGBoost Regressor generally produces the lowest percentage errors, particularly for high-cost observations, indicating its strong capability to determine complicated nonlinear relationships in the data. Random Forest Regressor also performs robustly, yielding the smallest errors in several cases and showing high consistency, especially for low and medium charge levels. In contrast, Linear Regression exhibits substantially higher prediction errors for multiple observations, suggesting that its linear structure is insufficient for modeling the nonlinear patterns inherent in medical cost data.

Overall, the table supports the conclusion that ensemble-based models (especially XGBoost and Random Forest) provide significantly more correct and trustable predictions compared to the traditional Linear Regression approach.

To compare how the three ML methods assign importance to the independent variables, the feature importance values for each method are given in Table 10. For the Linear Regression, the coefficients were standardized to create a comparable measure of importance, while the variable importance scores produced by the Random Forest and XGBoost models were used directly. This comparison reveals how the key factors influencing insurance premiums differ across models and shows that ensemble methods tend to capture nonlinear interactions more effectively.

Table 10. Feature importance comparison across LR, RF, and XGB models

Models	***age***	***sex***	***bmi***	***children***	***smoker***	***region***
LR	0.0097	0.0007	0.0127	0.0160	0.8916	0.0693
RF	0.1349	0.0064	0.2142	0.0194	0.6096	0.0155
XGB	0.0136	0.0045	0.0196	0.0079	0.9371	0.0174

Table 10 compares the feature importance values gathered from the XGBoost, Random Forest, and Linear Regression models. Across all three approaches, the smoker variable emerges as the most influential predictor of insurance charges, reflecting the well-established impact of smoking on health-related expenditures. Ensemble models—particularly XGBoost, with an importance score of 0.9371—assign an even stronger weight to this variable, indicating their enhanced ability to capture nonlinear and interaction effects. The bmi and age variables show moderate importance in the Random Forest and XGBoost models, whereas Linear Regression assigns comparatively smaller weights, suggesting its limited capacity to model complex relationships in the data. Additionally, the region and sex variables consistently exhibit low importance across all models, implying minimal direct contribution to premium variation. Overall, the table highlights the superiority of ensemble methods in identifying dominant predictors and modeling heterogeneous patterns in medical insurance costs.

4.2. Comparison of the models

In thhis section comparative evaluation of the three machine learning models used in the study is presented, examining their performance, error metrics, prediction behavior, and variable importance patterns. Considering both the

structural characteristics of the models and the statistical features of the dataset, this analysis highlights which approach produces more effective results for the problem of insurance premium prediction, where the relations are complex and nonlinear.

Based on the performance metrics, the Random Forest Regressor stands out with the highest R^2 value (0.8640) and the lowest error measures (MAE, MSE, RMSE, MAPE). This outcome can be attributed to the model's capability to avoid overfitting by averaging predictions from many decision trees and to effectively capture complicated relationships in the data. The XGBoost Regressor also performs strongly, achieving low error rates particularly for higher premium values and successfully representing nonlinear patterns.

The Linear Regression model showed the weakest performance among the three models. This outcome is mainly due to the fact that many relationships in the dataset are nonlinear. For instance, the effects of variables such as smoking status, body mass index (BMI), and age on medical costs are far from linear; after certain threshold levels, costs increase sharply. Because a linear model cannot capture these kinds of patterns, its error levels were higher.

Another aspect of the model comparison is the assessment of variable importance. In all three models, the smoker variable clearly ranks as the most influential factor, confirming the dominant impact of smoking on healthcare expenditures. In the ensemble models, its importance score is even higher, which indicates that these algorithms capture interactions and nonlinear relationships more effectively. The variables BMI and age show moderate importance in the Random Forest and XGBoost models, whereas their effects appear more limited in Linear Regression. This finding further illustrates that the linear model has difficulty representing more complex patterns in the data.

Finally, when the predicted and actual values are examined, XGBoost appears to produce more consistent estimates for individuals with high premium levels, while Random Forest performs more reliably for medium and low premium ranges. This difference can be linked to how each model responds to the distribution of the data. On the other hand, Linear Regression generates substantial errors for extreme values, highlighting the limitations of a linear approach when dealing with health expenditure data that exhibit high variance and deviate significantly from normality.

Overall, the findings demonstrates that the ensemble-based methods, namely the Random Forest Regressor and the XGBoost Regressor, offer higher accuracy and stronger generalization ability for problems like insurance premium prediction, where relationships are highly multivariate and complex. However, in scenarios where interpretability is essential, the Linear Regression

model remains a valuable reference, even though its performance clearly falls behind that of modern ensemble techniques.

4.3. Interpretation of Findings

The results of this study offer a detailed insight into the key factors that most significantly affect individual health insurance premiums. Across all the three machine learning models, the smoker variable consistently emerges as the most dominant predictor. This is particularly evident in the ensemble models, where smoker receives exceptionally high importance scores, reflecting its substantial contribution to medical expenditures. These results are well aligned with the existing literature, which demonstrates that smoking dramatically increases the likelihood of chronic diseases, hospitalization, and long-term healthcare costs. Consequently, individuals who smoke are classified as high-risk members in insurance pools, leading to significantly higher premium levels.

The body mass index (BMI) is another critical determinant of insurance charges. The moderate importance assigned to BMI by Random Forest and XGBoost indicates that medical costs respond nonlinearly to changes in BMI—a pattern that ensemble methods can effectively capture. The age variable similarly demonstrates a notable influence on premium levels, reflecting the natural increase in health risks as individuals grow older. Age-related deterioration in physical health, combined with heightened susceptibility to chronic illnesses, explains the progressive rise in predicted insurance charges.

In contrast, variables such as children, sex, and region contribute relatively less to premium variation. Although these factors may have indirect or context-dependent effects on healthcare utilization, their overall influence remains minor compared with the strong and direct impact of smoking behavior, BMI, and age. This pattern suggests that insurance pricing is primarily driven by individual health risks and lifestyle-associated factors rather than demographic or geographic attributes.

5. CONCLUSIONS AND RECOMMENDATIONS

This study provided a detailed comparative analysis of three ML models (XGBoost Regressor, Random Forest Regressor, and Linear Regression) applied to the prediction of individual health insurance premiums. The findings show that ensemble-based ML methods demonstrate clear superiority over classical regression approaches, particularly in modeling complex, nonlinear relationships within health cost data.

Among the evaluated models, the Random Forest Regressor achieved the strongest predictive performance, demonstrated by its higher R^2 value and lower

error scores (MAE, RMSE, MSE, and MAPE). Its ensemble structure, combining the outputs of numerous decision trees, enables robust generalization and reduces overfitting, even when the dataset contains variability or noise. The XGBoost Regressor also showed strong performance, especially for high-cost observations, reflecting its capacity to capture intricate nonlinear patterns and feature interactions. Conversely, the Linear Regression method demonstrated relatively weak predictive performance. Although its structure provides high interpretability, its inability to capture nonlinearities resulted in higher error levels, reinforcing the limitations of classical regression in complex real-world prediction tasks.

Overall, the findings highlight the advantages of AI-based regression techniques over traditional statistical models. In particular, ensemble algorithms offer enhanced flexibility, stronger modeling capacity for nonlinear relationships, and more realistic representations of variable importance. The pronounced dominance of factors such as BMI, smoking status, and age in the ensemble models aligns with established medical and actuarial knowledge, underscoring the robustness of these methods in cost estimation.

In conclusion, the findings of this study highlight the strong capability of machine learning and artificial intelligence methods to accurately predict health insurance expenditures. The comparison between classical and modern models underscores the importance of nonlinear modeling capabilities, robust feature interaction handling, and data-driven variable importance estimation. Expanding the methodology to other datasets and exploring more advanced modeling strategies will further contribute to the development of accurate and reliable predictive systems in the health insurance domain and beyond.

REFERENCES

Bader, M., & Maalouf, M. (2024, December). Evaluating Determinants of Health Insurance Premiums Using Advanced Multiple Linear Regression Techniques. In 2024 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM) (pp. 440-444). IEEE.

Bhardwaj, N., & Anand, R. (2020). Health insurance amount prediction. Int. J. Eng. Res, 9, 1008-1011.

Blockeel, H. (2023). Decision trees: from efficient prediction to responsible AI. Frontiers in Artificial Intelligence.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD Conference.

Fisher, A., Rudin, C., & Dominici, F. (2019). All models are wrong, but many are useful: Variable importance for black-box, proprietary, or misspecified prediction models. The Annals of Applied Statistics, 13(4), 2353–2381.

Géron, A. (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (3rd ed.). O'Reilly Media.

Gujarati, D. N., & Porter, D. C. (2009). Basic Econometrics (5th ed.). McGraw-Hill Education.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. Journal of Machine Learning Research, 3, 1157–1182.

Han, J., Kamber, M., & Pei, J. (2012). Data Mining: Concepts and Techniques (3rd ed.). Morgan Kaufmann.

Hastie, T., Tibshirani, R., & Friedman, J. (2017). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer.

Ivanovna, K. O., Vladimirovna, M. O., & Turgaeva, A. (2018). Insurance risks management methodology. Journal of Risk and Financial management, 11(4), 75.

Jadhav, A., Pramod, D., & Ramanathan, K. (2019). Comparison of performance of data imputation methods for numeric dataset. Applied Artificial Intelligence, 33(10), 913-933.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An Introduction to Statistical Learning with Applications in Python. Springer.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An Introduction to Statistical Learning with Applications in Python. Springer.

Kapse, M., Sharma, V., Vidhale, R., & Vellanki, V. (2025). Customization of health insurance premiums using machine learning and explainable AI. International Journal of Information Management Data Insights, 5(1), 100328.

Kaushik, K., Bhardwaj, A., Dwivedi, A. D., & Singh, R. (2022). Machine learning-based regression framework to predict health insurance premiums. International journal of environmental research and public health, 19(13), 7898.

Kotsiantis, S. B., Kanellopoulos, D., & Pintelas, P. E. (2006). Data preprocessing for supervised leaning. International journal of computer science, 1(2), 111-117.

Kuhn, M., & Johnson, K. (2019). Feature Engineering and Selection: A Practical Approach for Predictive Models. CRC Press.

Kutner, M. H., Nachtsheim, C. J., Neter, J., & Li, W. (2005). Applied Linear Statistical Models (5th ed.). McGraw-Hill.

Lantz, B. (2019). Machine Learning with R: Expert Techniques for Predictive Modeling. Packt Publishing.

Little, R. J., & Rubin, D. B. (2019). Statistical analysis with missing data. John Wiley & Sons.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. Advances in Neural Information Processing Systems (NeurIPS), 30.

Mishra, S., Kapoor, R., Yukti, & Mahesh, G. (2024, September). Prediction of Health Insurance Premium Using XG Boost Algorithm. In International Conference on Electrical and Electronics Engineering (pp. 433-455). Singapore: Springer Nature Singapore.

Montgomery, D. C., Peck, E. A., & Vining, G. G. (2021). Introduction to Linear Regression Analysis (6th ed.). Wiley.

Panda, S., Purkayastha, B., Das, D., Chakraborty, M., & Biswas, S. K. (2022, May). Health insurance cost prediction using regression models. In 2022 International conference on machine learning, big data, cloud and parallel computing (COM-IT-CON) (Vol. 1, pp. 168-173). IEEE.

Potdar, K., Pardawala, T. S., & Pai, C. D. (2017). A comparative study of categorical variable encoding techniques for neural network classifiers. International Journal of Computer Applications, 175(4), 7–9.

Probst, P., Wright, M. N., & Boulesteix, A.-L. (2019). Hyperparameters and tuning strategies for random forest. WIREs Data Mining and Knowledge Discovery, 9(3), e1301.

Abdelghany, Mosap. (2025). Medical Insurance Cost Dataset [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/12853160