Science, Engineering & Education, 9, (1), 2024, 65-74

Engine Fault Detection System (Autoguard)

Timon Trifonov*

PPMG „Akademik Ivan Tsenov", 18 Demokracia Blvd., Vratsa, Bulgaria

Received 23 September 2024, Accepted 26 October 2024
DOI: 10.59957/see.v9.i1.2024.9

ABSTRACT

This report presents the development of a mobile application named AutoGuard, which serves as a comprehensive anomaly detection system specifically designed for the automotive industry. AutoGuard utilizes advanced machine learning techniques, including statistical modeling, to analyze data collected from various sensors in a vehicle's engine. The application not only detects potential faults but also provides real-time diagnostics by processing and visualizing the information in a user-friendly format. Extensive testing and analysis demonstrate the system's ability to enhance vehicle safety and maintenance efficiency, making it a valuable tool for both drivers and automotive professionals.

Keywords: Vehicle diagnostics, Fault detection system, Sensor data analysis, Machine learning, Principal component analysis, Real-time monitoring.

INTRODUCTION

As vehicles become increasingly complex and their systems more interconnected, the need for robust anomaly detection solutions has never been more critical. Modern vehicles rely on a multitude of sensors and electronic control units (ECUs) that communicate continuously to ensure optimal performance and safety. This complexity, however, introduces new challenges in monitoring and maintaining vehicle health, necessitating advanced solutions capable of identifying anomalies before they escalate into serious issues.

Various algorithmic approaches have been explored to address this challenge, with machine learning and statistical methods at the forefront. These methods leverage vast amounts of data generated by vehicle systems to detect patterns and deviations indicative of potential problems. Among these, an anomaly detection model based on Principal Component Analysis (PCA) method proposed by Mei-Ling Shyu et al. has shown significant promise [1]. By analysing data extracted from the car's engine and other critical components, this system can detect anomalies in real or near-real time, enabling proactive maintenance, error prevention, and improved safety measures. The implementation of such a system presents numerous benefits. Proactive maintenance can significantly reduce downtime and repair costs by addressing issues before they lead to major failures. Additionally, early detection of anomalies contributes to error prevention, ensuring that vehicles operate within

**Correspondence to: Timon Trifonov, PPMG „Akademik Ivan Tsenov", 18 Demokracia Blvd., Vratsa, Bulgaria, E-mail: timonaki6566@gmail.com*

safe parameters and reducing the likelihood of accidents caused by mechanical faults. This, in turn, enhances overall vehicle safety, providing peace of mind to drivers and passengers alike. Auto Guard is an innovative application designed to democratize access to these advanced anomaly detection capabilities. It allows users, even those without a basic understanding of machine learning, to train and apply the anomaly detection model in real time.

The user-friendly interface and intuitive design make it accessible to a broad audience, empowering vehicle owners and fleet managers to take control of vehicle health monitoring without requiring specialized technical knowledge. By making cutting-edge technology accessible to everyday users, Auto Guard bridges the gap between complex machine learning algorithms and practical, real-world applications, ensuring the vehicle's safety and reliability.

EXPERIMENTAL

Vehicle connection

The integration of the anomaly detection system with the vehicle is facilitated through an On-Board Diagnostics (OBD) device, a standardized interface that provides access to a wealth of real-time data from the car's various sensors and electronic control units. The OBD device plugs into the vehicle's OBD-II port, typically located under the dashboard, and serves as a bridge between the car's internal systems and the anomaly detection application. By continuously streaming data such as engine performance metrics, fuel efficiency, emission levels, and diagnostic trouble codes, the OBD device enables comprehensive monitoring of the vehicle's health. This seamless connection allows the anomaly detection model to analyze the incoming data in real or near-real time, identifying any deviations from normal patterns that might indicate potential issues.

To connect with the attached OBD device, the application uses the flutter_bluetooth_serial module, which allows easy communication even with devices supporting only older versions of Bluetooth, such as v2, v2.1 and v3 [2].

Communication between devices

The communication rules between the device and the application are defined by the SAE J1979 standard [3]. SAE J1979 is a unified protocol that defines the methods of obtaining diagnostic data and the list of standard parameters that can be required by the device. Each parameter is addressed by a unique parameter ID (PID) defined in the SAE J1979 protocol. To obtain information about a specific parameter, the application sends the corresponding identification numbers to the OBD device. This is the so-called PID request. The device then accepts the request and sends back the desired information.

After information is sent to the application, it is decoded from the hexadecimal number system and a certain formula is applied to the resulting result, which converts the resulting number into the desired unit of measurement (Fig. 1). On initial contact with a connected OBD device, the application resets the OBD settings to default and sends a request with PID of "00". The request asks the device to send back all the available PIDs supported. The application then checks the available PIDs and maps them to the corresponding engine parameters.

In Fig. 1, an example is shown: We want to know the current value of the engine temperature. We send the ID of the parameter, which in this case is "05." The "01" header before the ID number indicates that this is a request (Fig. 1a). The "4105" header indicates that this is a response to a request with ID "05." "C9" is the hexadecimal number corresponding to the engine temperature (Fig. 1b). In the decimal system, this is the number 201. We apply the appropriate formula from the SAE J1979 protocol. Here, for conversion to degrees Celsius, it is A-40. We calculate 201 – 40 and find a temperature of 161°C (Fig. 1c).

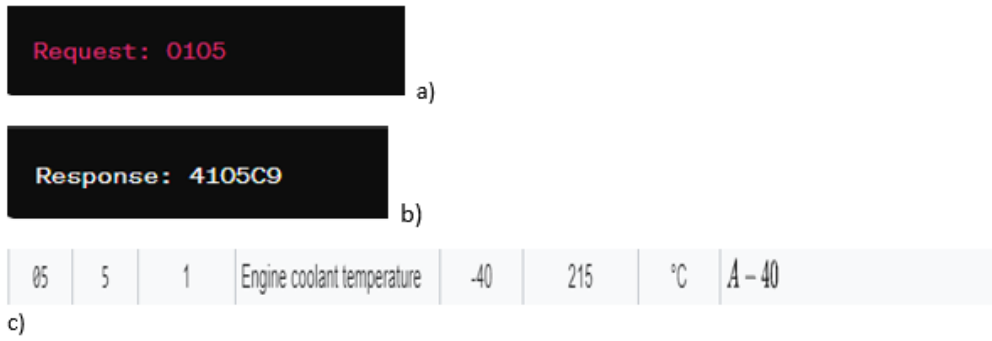


Fig. 1. Converts the resulting number into the desired unit of measurement.

Anomaly detection

Every single car is unique, with variations in design, performance, and usage that create a vast array of parameter values. This inherent diversity means there are no universally accepted reference values for all cars. Each parameter - whether it's engine performance, fuel efficiency, or sensor readings - differs from one model to another. As a result, the common supervised machine learning (ML) approach, which relies on large, labeled datasets for training, is often impractical for anomaly detection in the automotive industry. The industry struggles with a scarcity of such comprehensive datasets, making supervised methods less effective.

In contrast, unsupervised machine learning offers a promising alternative. Unlike supervised ML, unsupervised ML does not require labeled datasets, allowing it to adapt to the unique characteristics and behaviors of individual vehicles. This approach provides more flexibility and is better suited to handle the individualism inherent in automotive data. By leveraging unsupervised ML, we can develop models that detect anomalies based on the specific operational patterns of each vehicle, leading to more accurate and reliable diagnostics and maintenance recommendations.

For this reason, AutoGuard relies on Principal Component Analysis (PCA) for anomaly detection - a long-proven unsupervised ML algorithm.

Principal Component Analysis

PCA is a powerful statistical technique used for dimensionality reduction, feature extraction, and data visualization. It transforms the data into a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (the first principal component), the second greatest variance on the second coordinate, and so on. The principal components are obtained by calculating the eigenvalues and eigenvectors of the covariance matrix. The eigenvectors determine the direction of the principal components, while the eigenvalues determine their magnitude. This method is particularly useful for anomaly detection because it helps in identifying the principal structure of the data and highlights variations or outliers that do not conform to this structure.

Steps in performing PCA for anomaly detection

1. Data standardization: PCA is sensitive to the scale of the data. Therefore, it is crucial to standardize the data. Given a dataset $X=\{x_1,x_2,\ldots, x_n\}$ where each x_i is an observation with p features, the standardization process transforms the data so that each feature has a mean of 0 and a standard deviation of 1. This is done by

subtracting the mean and dividing by the standard deviation feature-wise.

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

Here, z_{ij} represents the standardized value of the j-th feature for the i-th observation and μ_j, σ_j are the mean and standard deviation of the same feature across all observation.

2. Covariance matrix: Something that reflects the relationships between different features is required. For that reason, the covariance matrix Σ of all standardized observations Z is computed:

$$\Sigma = \frac{1}{n} Z^T Z$$

3. Eigen decomposition: Decompose the covariance matrix Σ to find eigenvalues and eigenvectors. Each eigenvector $\vec{v}$ represents a direction in the feature space and each eigenvalue λ indicates the variance captured by its corresponding eigenvector:

$$\Sigma \vec{v} = \lambda \vec{v}$$

4. Eigenvalues and eigenvectors: Arrange the eigenvalues λ and their corresponding eigenvectors $\vec{v}$ in descending order of λ.

5. Principal components: Choose the top k eigenvectors based on the largest eigenvalues to form a matrix V_k with dimensions $p \times k$. Each column of V_k is a principal component that is a linear combination of the original features.

6. Data transformation: Project the standardized data Z onto the new feature space defined by the selected principal components:

$$Y = ZV_k$$

Here, Y is the transformed dataset with reduced dimensions k. Each new feature in Y is a principal component, which is a linear combination of the original standardized features.

7. Data reconstruction: Reconstruct the data from the lower-dimensional representation Y. This involves projecting Y back to the original feature space using the transpose of the principal components matrix V_k^T:

$$\hat{Z} = Y V_k^T$$

8. Revert standardization: Convert $\hat{Z}$ back to the original scale of X:

$$\hat{X}_{ij} = \hat{Z}_{ij}\sigma_j - \mu_j$$

This gives the reconstructed data $\hat{X}$ in the original feature space.

9. Reconstruction error: The reconstruction error is the difference between the original data X and the reconstructed data $\hat{X}$:

$$Reconstruction\ Error = X - \hat{X}$$

In practice, you often compute the norm of this error (such as the Euclidean distance) to get a single value that represents the magnitude of the error for each observation:

$$Reconstruction\ Error = ||X - \hat{X}||$$

The reconstruction error provides a measure of how well the lower-dimensional subspace captures the original data. Data points with high reconstruction errors are considered anomalies, as they do not fit well within the subspace defined by the principal components. When reconstruction error exceeds given threshold the observation is flagged as anomaly.

Fig. 2 shows example case with artificially generated data where the red points are anomalies because they do not lie in the same plane as the green ones after sequentially applying PCA for dimensionality reduction and reconstruction. This is because they cannot be explained by only 2 principal components i.e. in two-dimensional space, unlike the set of green dots.

Database

Sensor data and model weights need to be stored efficiently and securely for further processing and visualization. The strategy involves saving the model weights both locally on mobile devices and remotely on a server, while sensor data is exclusively stored in a remote database. This approach ensures that trained models and data are accessible by multiple devices simultaneously and remain available even after app reinstallation.

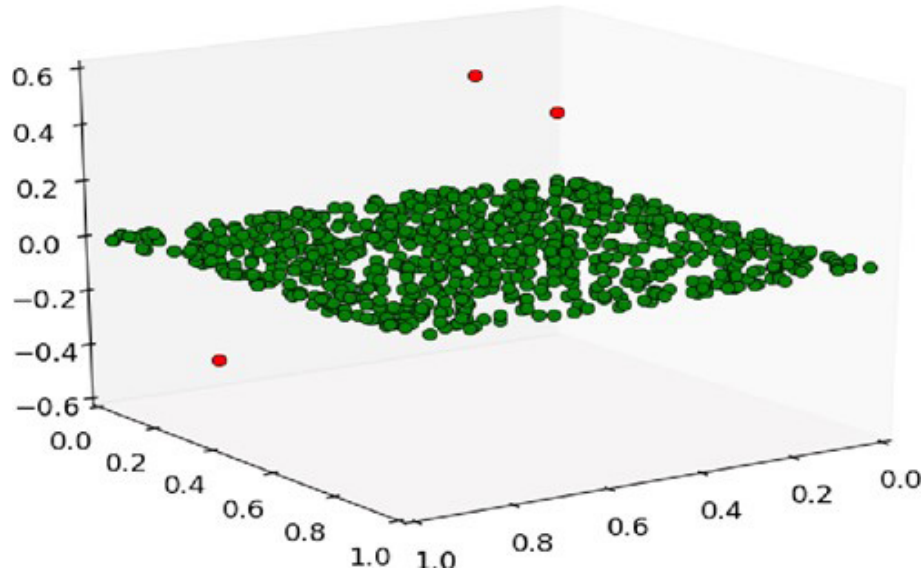


Fig. 2. Identification of Anomalies Using PCA - Red Points Deviate from the Green Points Due to Insufficient Explanation by Two Principal Components [7].

The database is a relational database implemented using PostgreSQL (Fig. 3). It consists of four interconnected tables, designed to efficiently store user information, sensor data collected from cars, model information, and anomalies detected by the models. The integration of TimescaleDB, a PostgreSQL extension specialized in time-series data, enhances the performance of the application by optimizing the storage and querying of sensor data [4]. By leveraging PostgreSQL and TimescaleDB, along with the postgres.dart connector for application-database communication, this setup ensures efficient, scalable, and reliable management of user information, sensor data, models, and anomalies [5].

Workflow Overview

This application comprises several key components, each performing distinct functions to ensure seamless interaction with the OBD (On-Board Diagnostics) device, data management, and anomaly detection in a car's engine. The primary parts of the application are as follows:

1. OBD Device Connection and Communication: This component establishes a secure connection between the application and the OBD device. It authenticates the OBD device to ensure its legitimacy, encodes information when sending data to the OBD device, and decodes information received from the OBD device for further processing.

2. Data Presentation and Storage: This section involves the collection of sensor data from the OBD device. It displays the sensor information to the user through interactive tables and charts, stores the collected data in a database for future reference and analysis, and prepares the stored data for potential use in training machine learning models.

3. Anomaly Detection Model: This component utilizes the collected sensor data to train a machine learning model. The trained model is then implemented to detect anomalies in the car's engine in real-time. Continuous updates are made to the model to improve its accuracy and reliability.

4. Anomaly Recording and User Notification: This section records each detected anomaly along with relevant information such as timestamp, sensor readings, and possible causes. It compiles

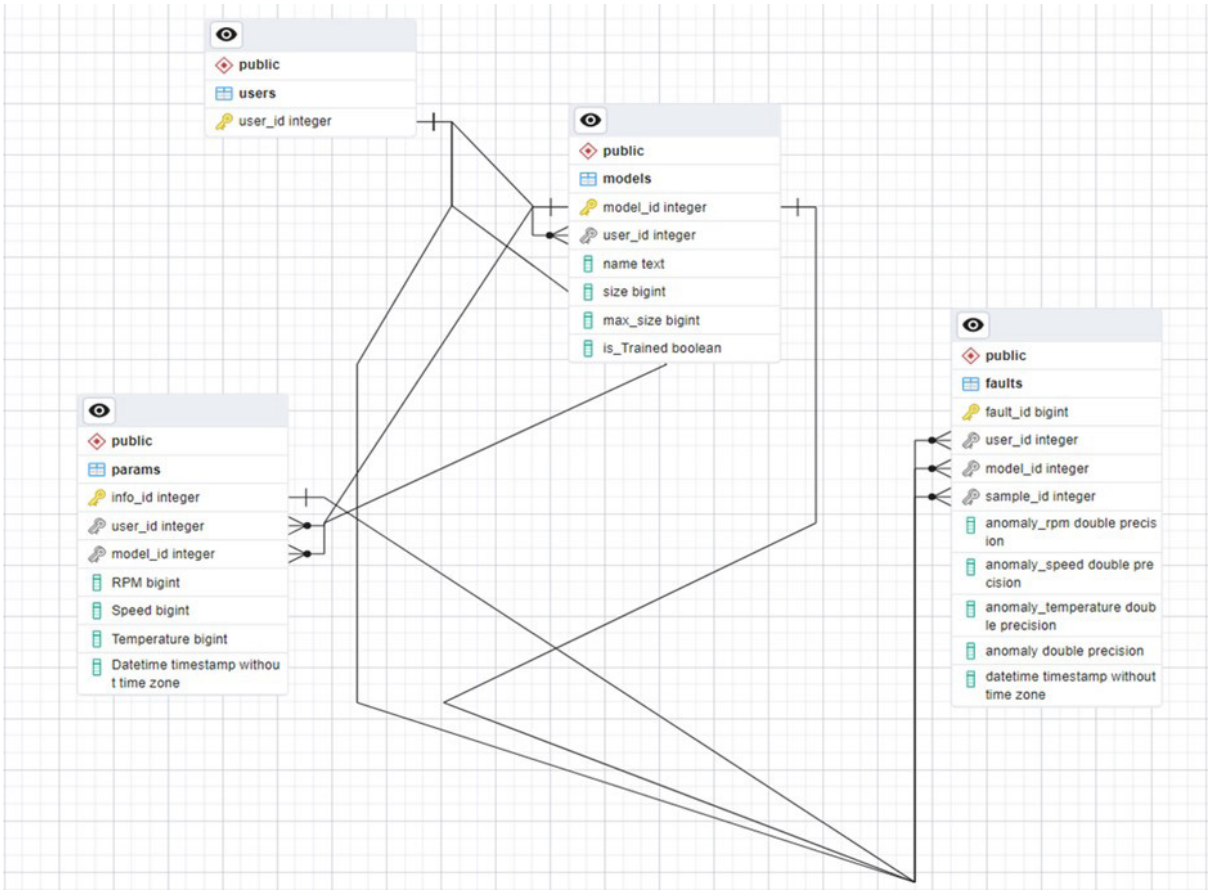


Fig. 3. Simplified view of the PostgreSQL database.

a detailed list of all registered anomalies and provides the user with access to this list, offering insights and recommendations for further action.

By integrating these components, the application ensures efficient communication with the OBD device, effective data storage, robust anomaly detection, and comprehensive reporting of engine issues.

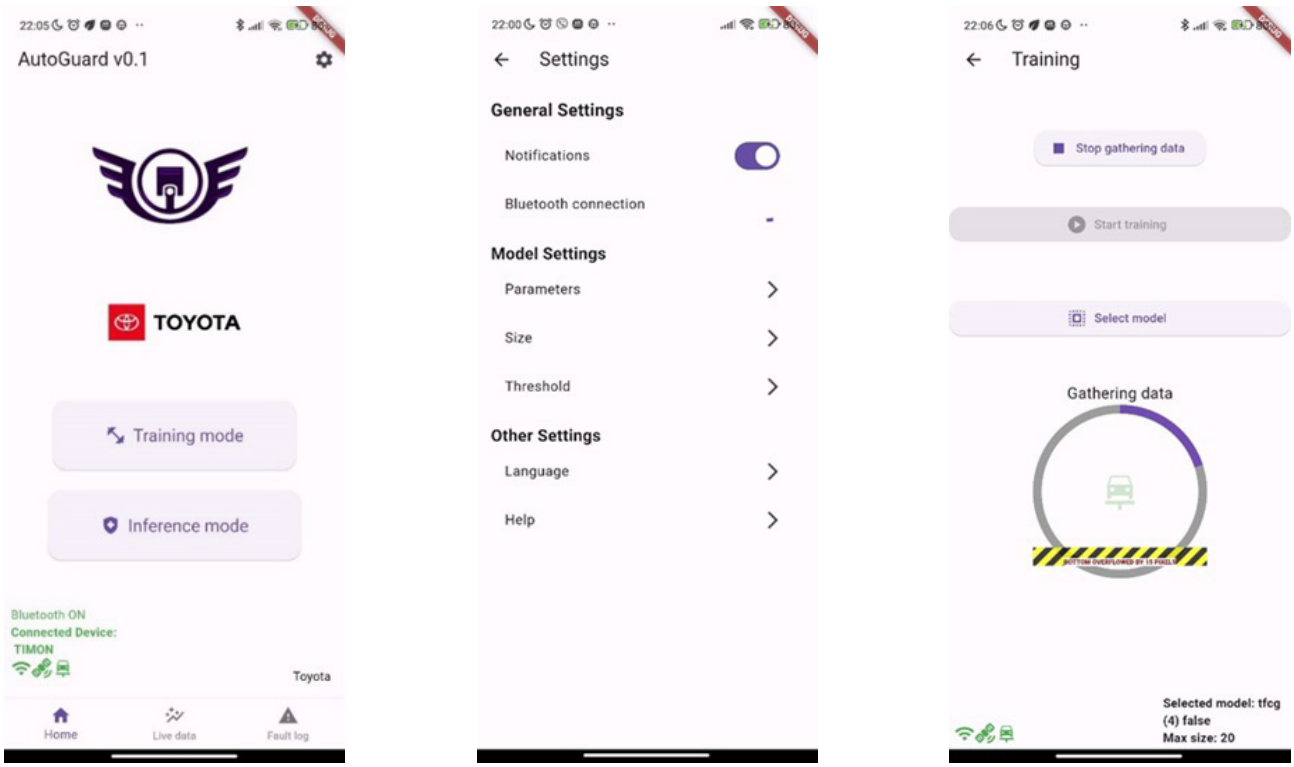
Interface

The mobile application interface is structured into three primary sections: the Home Page, the Sensor Data Page, and the Anomalies Page. This layout ensures intuitive navigation and efficient access to the application's core features.

Home page

The Home page contains three main buttons, each capturing essential functionalities of the application.

- *Settings Button*: Located at the top right corner, this button opens the settings menu. The settings menu provides various options, including:
  - Connecting to a Bluetooth device
  - Managing notifications
  - Changing the language
  - Accessing help resources
  - Customizing parameters that the model will monitor, including model size and threshold values for anomaly detection



a) b) c)

Fig. 4. The design of the AutoGuard app: (a) Home Page; (b) Settings, (c) Training menu.

The primary function of the settings menu is to offer users comprehensive control over the application's configuration and behavior.

- *Data Collection and Model Training Button*: Positioned centrally, this button directs users to the menu for collecting data, as well as creating and training machine learning models.
- *Inference Mode Button*: Also centrally located, this button initiates the Inference mode, which employs a pre-trained model to perform real-time anomaly detection.

The design of the Home Page, that is shown on Fig. 4, ensures that users can quickly access core unctionalities and customize their experience according to their needs.

Live Data page

The live data page (Fig. 5) consists of table displaying information about the current read engine parameters and plot showing recent anomaly scores. The plot illustrates the current deviation from the training data, expressed as a percentage of anomalies. If the graph exceeds a certain limit (indicated by the red dashed line in the plot), an anomaly will be triggered..

Fault Log page

The Fault Log page (Fig.6) provides detailed information about recently detected anomalies, offering insights into the overall anomaly score and the variance explained by each parameter. The variance for each parameter is calculated using the explained variance ratio, defined by the following formula:

$$Explained\ Variance\ Ratio_i = \frac{\lambda_i}{\sum_j \lambda_j}$$

where λ_j is the j-th eigenvalue of the covariance matrix, corresponding to j-th parameter.

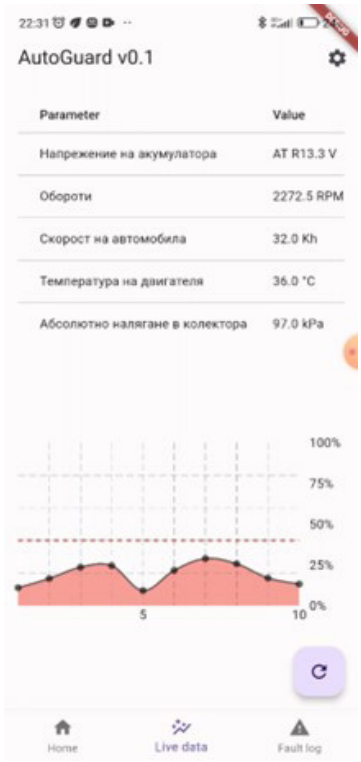


Fig. 5. Live data page.

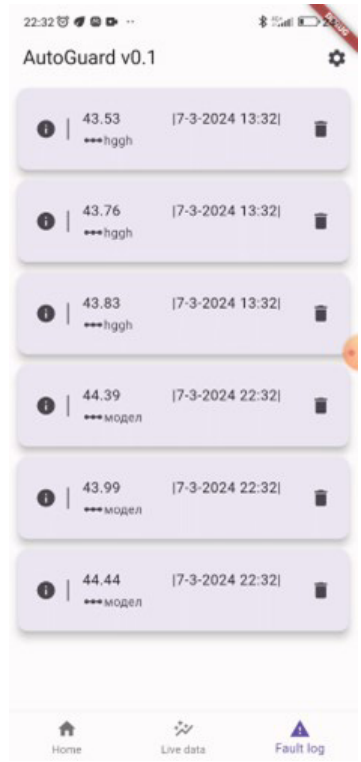


Fig. 6. Fault log page.

RESULTS AND DISCUSSION

AutoGuard's anomaly detection system was evaluated using a dataset generated by the ELM327-emulator library [6]. This library simulates sensor data from real vehicles and transmits it to the application via Bluetooth, enabling comprehensive testing and validation of the system under controlled conditions. The dataset included normal operational data and data with induced faults to test the system's ability to detect anomalies. The following metrics were used to assess the model's performance:

- Precision: The percentage of true positive detections among all detections made by the system.
- Recall: The percentage of true positives detected out of all actual anomalies.
- F1-Score: The harmonic mean of precision and recall, providing a single metric that balances both.

AutoGuard demonstrated a precision of 95 %, a recall of 92 %, and an F1-score of 93.5 %. These results indicate that the system effectively identifies anomalies with high accuracy, minimizing false positives and ensuring reliable fault detection. Table 1 compares the performance of AutoGuard's PCA-based method with other common anomaly detection algorithms.

Real-World Testing

Tests were also conducted in real-world conditions using Opel Meriva B and Mazda CX-3 vehicles equipped with an ELM327 OBD2 device. Given the budget constraints of this project, inducing real-time engine faults in vehicles for testing was deemed too costly. As a result, tests with intentionally damaged cars were not performed.

Despite these limitations, the application performed as expected in most scenarios.

Table 1. Performance Comparison of AutoGuard's PCA-Based Method with Other Anomaly Detection Algorithms.

Algorithm	Precision, %	Recall, %	F1-Score, %
AutoGuard (PCA)	95	92	93.5
Isolation Forest	90	88	89
One-Class SVM	85	83	84
K-Means Clustering	80	78	79

However, two primary issues were identified:

1. Slower Data Collection: The data collection rate from the vehicles was occasionally slower than anticipated. This lag can affect the real-time diagnostic capability of the application, particularly in scenarios where immediate feedback is crucial.

2. Intermittent Submission of Invalid Data: There were instances where the vehicles' sensors transmitted invalid data. This intermittent issue required additional filtering and validation processes within the application to ensure accuracy in anomaly detection.

These findings underscore the importance of further refining the data collection mechanisms and enhancing the robustness of the data validation processes. Addressing these issues will be critical in optimizing the application's performance and reliability in real-world conditions.

User Experience

User feedback was collected through surveys and usability testing sessions. The following aspects were evaluated:

- Ease of Use: Users rated the application 4.7 out of 5 for ease of use, highlighting the intuitive design and clear instructions provided within the app.
- Interface Design: The user interface received a rating of 4.8 out of 5, with users appreciating the clear visualizations and logical layout.

Overall Satisfaction: Overall satisfaction with the application was rated at 4.9 out of 5, indicating high acceptance and approval from both individual drivers and fleet managers.

CONCLUSIONS

AutoGuard offers a robust, user-friendly solution for real-time vehicle anomaly detection and diagnostics. By leveraging advanced machine learning techniques and providing a user-friendly interface, the application enhances vehicle safety, reduces maintenance costs, and democratizes access to cutting-edge technology. Continued development and enhancement of AutoGuard hold the promise of even greater benefits, making it an invaluable tool for drivers and automotive professionals alike.

REFERENCES

1. M.L. Shyu, S.C. Chen, K. Sarinnapakorn, L. Chang, A Novel Anomaly Detection Scheme Based on Principal Component Classifier, Proceedings of the International Conference on Data Mining, 2003.
2. 3daysapp.com.br. flutter_bluetooth_serial. Version 0.2.2. Available at: https://pub.dev/packages/flutter_bluetooth_serial, Accessed on July 8, 2024.
3. OBD-II PIDs. Wikipedia. Available at: https://en.wikipedia.org/wiki/OBD-II_PIDs, Accessed on July 8, 2024.
4. Timescaledb. GitHub, Available at https://github.com/timescale/timescaledb, Accessed

on July 8, 2024.
5. agilord.com. postgres. Version 3.2.1. Available at: https://pub.dev/packages/postgres
6. ELM327-emulator. GitHub. Available at https://github.com/Ircama/ELM327-emulator?tab=readme-ov-file. Accessed on July 8, 2024.
7. Cyber Security Network Anomaly Detection and Visualization Major Qualifying Project, 2017.