The object of the research is a monitoring and analysis system for software quality based on user feedback collected from open-source projects on GitHub. The problem addressed is the lack of effective automated tools that can process large volumes of unstructured user feedback to identify quality issues, prioritize tasks, and detect negative trends in real time. Traditional quality assurance methods, while important, fail to capture the nuance of user sentiment and the contextual details present in natural language feedback, leading to delays in problem detection and resolution. The developed system integrates three key modules: sentiment analysis for assessing user satisfaction, issue categorization for structuring feedback into actionable types, and anomaly detection for identifying sudden changes in sentiment or feedback dynamics. The results show that transformer-based models, particularly fine-tuned BERT, outperform rule-based and traditional machine learning approaches in both accuracy and robustness. This advantage is explained by their ability to capture domain-specific language, sarcasm, and contextual dependencies, enabling more precise interpretation of complex feedback. The anomaly detection component, using LSTM autoencoders and Isolation Forest, demonstrated the ability to identify critical quality regressions up to two days before official issue reporting. These results can be applied in practice for continuous software quality monitoring in agile, open-source, or user-centric development environments where timely, data-driven decision-making is essential. The approach supports real-time insight generation, helping development teams respond proactively to quality risks and improve overall user satisfaction

Keywords: feedback analysis, software quality, sentiment analysis, issue detection, data processing

UDC 004.056.5:004.8
DOI: 10.15587/1729-4061.2025.341516

EVALUATING AN INTEGRATED USER-FEEDBACK APPROACH TO SOFTWARE QUALITY MONITORING: ENHANCING ACCURACY AND TIMELINESS

Nurzhamal Kashkimbayeva
PhD*
Ulan Bekish
PhD, Associate Professor
Institute Management
Academy of Public Administration Under the President of the Republic of Kazakhstan
Abay ave., 33 a, Astana, Republic of Kazakhstan, 010000
Gulsim Abdoldinova
PhD, Associate Professor
Department of «Information Technology»
K. Kulazhanov Kazakh University of Technology and Business
Mukhamedkhanov str., 37A, Astana, Republic of Kazakhstan, 010000
Zhuldyz Basheyeva
Correspondence author
PhD*
E-mail: Zhuldyz.basheyeva@astanait.edu.kz
Alina Mitroshina
Master's Student**
*Department of Computer Engineering**
**Astana IT University
Mangilik El ave., C1, Astana, Republic of Kazakhstan, 010000

Received 22.07.2025
Received in revised form 25.09.205
Accepted date 06.10.2025
Published date 31.10.2025

How to Cite: *Kashkimbayeva, N., Bekish, U., Abdoldinova, G., Basheyeva, Z., Mitroshina, A. (2025). Evaluating an integrated user feedback approach to software quality monitoring: enhancing accuracy and timeliness. Eastern-European Journal of Enterprise Technologies, 5 (2 (137)), 29–42. https://doi.org/10.15587/1729-4061.2025.341516*

1. Introduction

As software systems become more complex and user expectations rise, continuous feedback from users has become a critical asset in ensuring and improving software quality. Traditionally, software quality assurance has relied on predefined test cases, automated test suites, and static code analysis. While these methods remain essential, they often fail to capture the full spectrum of real-world user experiences, especially in fast-moving, user-centric environments such as open-source projects.

GitHub, as one of the most widely used platforms for collaborative software development, provides a rich and publicly available repository of user feedback in the form of issues, comments, and pull request discussions. These textual artifacts contain a wealth of information related to bugs, feature requests, usability concerns, and general developer-user interaction. However, the unstructured nature of this feedback makes it difficult to process and interpret at scale using manual or rule-based approaches.

Recent advances in natural language processing (NLP) and machine learning, particularly the emergence of large-scale transformer models like BERT [1] and the self-attention mechanism [2], have opened new possibilities for automated understanding of textual data. These models are capable of capturing context, sentiment, and domain-specific nuances, making them highly suitable for the analysis of software-related feedback.

This research aims to develop a comprehensive system for monitoring software quality through automated analysis of user feedback from GitHub. The system includes three main components: sentiment analysis to assess user satisfaction,

issue categorization to structure the feedback, and anomaly detection to highlight unusual patterns in sentiment or report frequency. The hypothesis is that by combining these components, developers can obtain actionable, real-time insights that help prioritize work, identify critical issues early, and improve responsiveness to user needs.

In the context of modern software engineering, where agile methodologies and continuous delivery dominate, timely detection of quality issues based on direct user feedback is essential. The acceleration of release cycles and the global, distributed nature of development teams make traditional quality assurance insufficient for capturing subtle but important changes in user perception.

From a practical standpoint, integrating automated feedback analysis into quality monitoring pipelines enables development teams to address issues before they escalate, improve feature planning, and maintain higher user satisfaction.

Therefore, research devoted to developing integrated machine learning-based systems for automated analysis of user feedback in software quality assurance is relevant.

2. Literature review and problem statement

The intersection of user feedback analysis and software quality assurance has been an active area of research, with multiple studies emphasizing the value of mining textual data from platforms such as GitHub, App Stores, and Stack-Overflow. Prior work has addressed several key dimensions of this task, including sentiment analysis, issue classification, and anomaly detection.

Sentiment analysis has traditionally been used in marketing and social media [3] but has seen increasing application in software engineering. VADER, a rule-based sentiment analyzer optimized for social media, was introduced for analyzing short, informal text and has since been adopted in software mining studies for its speed and interpretability [4]. However, research indicates that such tools may be inadequate for software feedback, as developer communication often contains domain-specific terminology, mixed sentiment, and informal language that challenge rule-based systems. Newer models such as BERT and its variants have demonstrated superior performance by leveraging contextual embeddings [1], enabling more accurate detection of nuanced or sarcastic feedback, as shown in [5]. When fine-tuned on technical domains, these transformer-based models significantly outperform traditional classifiers in identifying sentiment in developer discussions [5]. Additionally, sentiment polarity in issue trackers and code review comments has been explored, showing that negative sentiment often correlates with defects and maintenance difficulties [6]. It is shown that these advances improve sentiment detection, but unresolved questions remain regarding real-time applicability and cross-project generalizability. The reasons include limited availability of domain-specific annotated corpora, reliance on general-purpose models, and the high cost of manual labeling. An option to overcome these difficulties is the use of domain-specific fine-tuning and multi-method approaches. In related fields such as electromagnetic interference analysis, researchers have proposed multi-method approaches to signal quality assessment that combine entropy-based metrics, correlation analysis, and statistical randomness tests to ensure the reliability and completeness of the assessment. For example, a comprehensive system for assessing the quality of masking noise interference from spatial noise generators has been developed, reflecting the importance of determining entropy coefficients and multi-band correlations [7]. This interdisciplinary methodology demonstrates the value of combining signal structure analysis with statistical verification, which can inspire software feedback control systems.

Automatically classifying user feedback into categories such as bugs, feature requests, or usability issues enables better triage and prioritization [8]. Early models for app review classification demonstrated success using SVM and Naive Bayes approaches [8], while advances in transformer-based methods have shown improved prioritization in issue management systems through feedback-aware learning with BERT-based classifiers. Accurate classification also benefits from incorporating domain-specific linguistic features and metadata, such as app category, platform, or user role [9]. Fine-grained sentiment analysis of app reviews has been studied in [9], and hybrid methods combining rule-based heuristics with supervised classifiers were proposed in [10]. Multi-label classification, relevant to feedback addressing multiple aspects, has been examined in [11], including approaches for hierarchical classification and handling complex feedback that covers multiple topics. It is shown that classification approaches are improving, but unresolved questions remain about model generalizability and reproducibility across projects. The reasons include inconsistencies in labeling practices, subjective annotation, and the fragmented nature of available datasets. An option to overcome these limitations is the adoption of hybrid approaches and coordinated dataset creation efforts [12].

Despite these advancements, most prior work has been evaluated on narrow domains or single datasets, limiting model generalizability across projects. The lack of standardized taxonomies for feedback categories, inconsistencies in labeling practices, and the fragmented nature of available datasets further hinder cross-domain applicability. These challenges are compounded by subjective labeling decisions and the absence of coordinated dataset creation efforts in open-source communities.

Anomaly detection plays a critical role in identifying sudden spikes in negative sentiment or feedback volume, which may indicate regressions or emerging incidents. A similar principle of prioritizing low visibility signals was used for early detection in environmental monitoring systems using UAVs. Multicopters equipped with advanced sensors and machine learning algorithms were used to monitor air quality across urban and industrial regions of Kazakhstan. The system collected over 500 GB of environmental data and processed it using Big Data and AI techniques, achieving pollutant prediction accuracy of up to 90% and enabling data collection four times faster than traditional methods [13]. It is shown that anomaly detection can be applied effectively in real-time monitoring, but unresolved issues remain in integrating it with sentiment and classification modules for root-cause analysis. The reasons are computational complexity, difficulties in synchronizing heterogeneous data streams, and the absence of end-to-end architectures. An option to overcome these is the adoption of sequential models and hybrid deep learning architectures. A method of time allocation based on reverse priority to improve the efficiency of servicing low-priority measurement tasks shows that such planning can reduce queue time and increase the throughput of the system [14]. This concept makes it possible to develop similar software quality control strategies in which feedback

delays or low signal levels are often associated with critical regressions. Detecting unusual trends or sudden shifts in user sentiment has been addressed through various techniques [15]. Traditional methods such as the local outlier factor (LOF) [15] and ıısolation aorests have been applied successfully in mining system logs and defect reports, and in software quality contexts these models are used to analyze temporal trends in user sentiment and issue categories.

Signal processing techniques used in other fields such as fiber optic seismic monitoring demonstrate the value of a combination of spectral decay, wave analysis, and machine learning to reliably detect anomalies. The applied distributed acoustic sensing (DAS) makes it possible to detect seismic events with greater than 90% accuracy even in conditions of noise and pressure changes, improving signal processing and LSTM-based classification [16]. This approach demonstrates the potential of hybrid models that combine physical modeling of signals with intelligent analytics – an architecture that inspires the development of similar strategies in the field of software quality control. Recent studies in civil engineering have demonstrated that fiber optic sensors (FOS), particularly fiber Bragg gratings, offer high precision and environmental resilience in monitoring structural deformations in concrete infrastructure [17]. These sensors operate reliably under extreme conditions such as humidity and temperature fluctuations, and their integration with advanced data processing techniques including AI-based analysis has shown promise for real-time anomaly detection and predictive maintenance. Such architectures provide a transferable framework for software quality monitoring systems, where feedback dynamics may reflect latent structural regressions in user experience.

Deep learning architectures combining convolutional and recurrent neural networks (CNN-RNN) can achieve classification accuracies between 81% and 95% when detecting impulsive urban sounds such as gunshots, explosions, sirens, and screams [18]. These models are robust even in noisy environments and can be adapted for real-time anomaly detection in dynamic settings. Such architectures offer valuable insights for software quality monitoring systems, where sudden shifts in user sentiment or feedback frequency may signal critical regressions. It is shown that these hybrid models are effective in dynamic environments, but unresolved questions remain about their scalability for large-scale software projects.

Recent advances in multisensor architectures for UAV threat detection demonstrate the effectiveness of combining heterogeneous data sources such as RF signals, acoustic patterns, and visual inputs through deep learning models like CNN-LSTM and attention-based fusion. These systems achieve detection accuracies exceeding 96% even in noisy environments, and their layered design enables real-time classification and response [19]. Such architectures inspire analogous strategies in software quality monitoring, where integrating sentiment, anomaly, and feedback classification modules can enhance responsiveness and robustness in dynamic user environments. LiDAR-based object detection systems demonstrate how high-resolution 3D spatial data, when combined with deep learning algorithms, can significantly improve anomaly detection and classification accuracy even in environments with sparse or noisy input [20]. These techniques have proven effective in UAV tracking, where dynamic motion and irregular data patterns resemble the challenges faced in user feedback analysis for software systems.

A hybrid deep learning architecture combining convolutional and recurrent neural networks (CNN-RNN) for real-time detection of impulsive sounds such as gunshots, explosions, and alarms has been proposed. Their model achieved over 96% accuracy and demonstrated robustness in noisy environments by leveraging MFCC features and temporal modeling via LSTM layers [21]. This layered design aligns with the needs of software quality monitoring systems, where abrupt shifts in user sentiment or feedback frequency may signal critical regressions. The CNN-RNN framework offers a transferable architecture for anomaly detection in dynamic, feedback-driven environments.

Time-series analysis of open-source feedback has been applied to detect anomalies, and studies have shown that temporal aggregation, when combined with NLP-derived features, enables early warnings about quality risks. Neural network–based modeling has also proven effective in predicting complex nonlinear dependencies in environmental systems. Moreover, autoencoder architectures, especially LSTM-based ones, have been applied to capture temporal dependencies in feedback streams. These models reconstruct expected sentiment patterns and flag deviations as anomalies, enabling detection of subtle shifts in developer–user communication. Artificial neural networks (ANNs) can accurately forecast oil sorption capacity based on time and material composition, achieving R^2 values above 0.98 and mean absolute percentage errors below 3.2% [22]. Methodology combining minimal ANN architectures with interpolation capabilities highlights the potential of neural models for generalizing from limited data and making robust predictions in noisy, real-world conditions. This approach may inspire similar strategies for modeling sentiment dynamics and anomaly detection in user feedback streams.

In the study, a similar approach is adopted to monitor shifts in user response across software versions. However, anomaly detection methods in existing studies are rarely integrated with sentiment analysis or feedback categorization, which limits their usefulness for identifying root causes of anomalies. The lack of integration is often due to computational complexity, the cost of synchronizing heterogeneous data streams, and the absence of end-to-end architectures capable of processing feedback in real time.

Automated classification of GitHub issues using neural networks has been explored [23], showing promising results in differentiating between enhancement requests and bug reports. These studies also emphasize the importance of structured metadata, such as issue labels and timestamps, for improving classification accuracy. Despite its potential, GitHub data is highly heterogeneous, containing a mix of technical discussions, informal comments, and irrelevant exchanges. This variability, combined with inconsistent labeling and incomplete metadata, complicates dataset creation and reduces model performance across projects. The main reasons include the informal nature of developer communication, the lack of standardized labeling practices, and noise inherent in open-source environments.

Recent research in railway station automation demonstrates that formal modeling using hierarchical state machines and UML diagrams can significantly improve the efficiency of complex infrastructure systems by reducing ambiguity and enabling structured decision-making [24]. These models allow for the simulation of dynamic workflows, resource allocation, and event-driven transitions, which are directly applicable to software quality monitoring systems that must process heterogeneous and asynchronous user feedback. Such hybrid control strategies, combining classical

decomposition with situational awareness, offer a robust foundation for designing distributed monitoring systems that rely on dynamic user feedback [25]. This approach is further supported by predictive modeling frameworks in railway infrastructure, where "mathematical support and software for solving prediction tasks based on the modified Delphi method (MDM) with the ability to conduct an online survey of experts" have been developed to enhance decision-making and system adaptability [26].

The literature supports the feasibility of using NLP and machine learning for mining user feedback in software development. However, existing approaches often focus on isolated tasks (e.g., sentiment analysis or classification alone), lack real-time capabilities, or do not leverage multi-component architectures. Unresolved problems include the absence of large domain-specific labeled datasets, the lack of standardized taxonomies, and the technical complexity of combining multiple analytical modules. The reasons lie in high annotation costs, computational overhead, and methodological difficulties of integration. Options to overcome these issues include hybrid architectures, domain-specific fine-tuning, and explainable AI methods, however these approaches are still fragmented and not fully validated. All this allows to argue that it is appropriate to conduct a study devoted to the development of an integrated machine learning – based system for real-time monitoring and analysis of software quality from user feedback.

3. The aim and objectives of the study

The aim of the study is to evaluate the effectiveness of combining sentiment analysis, issue categorization and anomaly-detection techniques within a unified framework for monitoring software quality through GitHub user feedback. This will make it possible to detect quality issues earlier and more accurately, providing development teams with actionable insights that reduce the time to respond to regressions.

To achieve this aim, the following objectives were accomplished:

– to evaluate and compare sentiment-analysis methods used on GitHub issue comments;

– to assess the effectiveness of issue-categorization approaches for classifying user feedback into actionable categories (e.g., bug, feature request, performance issue, usability issue, question);

– to investigate the ability of anomaly-detection techniques to identify unusual trends in sentiment or issue frequency.

4. Materials and methods

4. 1.The object and hypothesis of the study

The object of study is the process of assessing the quality of open-source software projects on GitHub through user feedback. The hypothesis is that integrating sentiment analysis, issue categorization and anomaly detection will improve the timeliness and accuracy of quality assessment compared with separate approaches.

Assumptions made in the study include:

1. User comments on GitHub reliably reflect perceptions of software quality.

2. The majority of comments are in English and can be processed using standard NLP tools.

3. The selected issue categories (bug report, feature request, usability problem, performance issue, general question) capture the most common types of feedback.

4. The manually and automatically assigned labels are sufficiently accurate to train and evaluate the models.

Simplifications adopted in the study include:

1. Only public repositories meeting the star and activity criteria are analyzed.

2. Feedback is limited to textual comments; attachments and code patches are excluded.

3. Models are evaluated only on a fixed snapshot of data collected between January 2023 and December 2024.

4. The study compares a specific set of algorithms (VADER, BERT, SVM, random forest, isolation forest, LOF and autoencoder) and does not exhaustively explore all possible methods.

4. 2. Dataset description

4. 2. 1. Data collection

This research focused on collecting user perspectives from comments found in GitHub Issues, a significant archive of exchanges between users and developers about software features, bugs, requests for new functionalities, and usability issues [27]. GitHub, being a central hub for software collaboration and issue resolution, offers an excellent setting for obtaining genuine and contextually pertinent user feedback.

4. 2. 2. Data source

in this study employed the GitHub REST API v3 [28] to systematically collect issue comments from a curated selection of 50 open-source repositories. The selection process was guided by the following criteria:

1) repositories with a minimum of 1,000 stars, indicating substantial community engagement;

2) evidence of active development, as reflected by commits made within the six months preceding data collection;

3) diversity in application domains, encompassing areas such as web development, mobile applications, machine learning, and developer tools.

In total, over 10,000 issue comments were gathered between January 2023 and December 2024. The collected dataset captures a broad spectrum of user feedback, including bug reports, feature requests, general questions, and performance-related concerns.

4. 2. 3. Data structure

Each extracted comment included both structured metadata and unstructured text. The primary fields are listed below in Table 1.

Table 1

Primary fields included in the GitHub REST API v3 dataset

Field	Description
repo_name	Repository from which the issue was collected
issue_id	Unique identifier for the issue
comment_id	Unique identifier for the comment
comment_text	Raw text of the user's comment
user_type	Maintainer, contributor, or external user
created_at	Timestamp of comment submission
labels	Assigned GitHub labels (e.g., bug, enhancement)
comment_length	Number of characters in comment
reactions	Number of reactions

This rich structure facilitates both linguistic processing and supervised learning tasks, such as classification and sentiment analysis.

4. 2. 4. Preprocessing pipeline

To transform raw textual feedback into structured machine-readable input, the following preprocessing steps were performed:

1. Language filtering: comments were filtered to retain only English text (via langdetect).

2. Lowercasing and cleaning: removal of punctuation, HTML tags, emojis, and links using regular expressions.

3. Tokenization: segmentation of text into word-level tokens using spaCy [29].

4. Stopword removal: elimination of common English stopwords and domain-specific filler words.

5. Lemmatization: reduction of inflected words to their base form (e.g., "fixes" → "fix").

6. Outlier filtering: exclusion of very short (<10 characters) or overly long (>1000 characters) comments.

The final dataset was balanced to include diverse sentiment and issue categories for effective model training.

4. 3. Methodology

4. 3. 1. System architecture and pipeline

The proposed system for analyzing software quality via user feedback follows a modular, machine-learning-driven architecture. This modularity reflects broader trends in engineering automation, where the idea of construction is connected with integrated software system Trace mode. This product has module for design and investigation of apparatuses with new quality of elements [30]. The overall pipeline is divided into the stages listed below.

4. 3. 2. Data ingestion

The system begins by ingesting user feedback from GitHub's API, extracting issue comments using Python scripts. A scheduler ensures continuous updates to maintain real-time relevance.

Tools: GitHub REST API, PyGithub, Python requests.

4. 3. 3. Data cleaning and normalization

Raw comments are processed to remove noise and standardize text:

1. Cleaning: strip emojis, code snippets, and links.

2. Tokenization & lemmatization: via spaCy.

3. Normalization: lowercasing and whitespace trimming.

4. Custom stopwords: to exclude domain-specific artifacts (e.g., usernames or repository mentions).

4. 3. 4. Sentiment analysis

The cleaned text is passed to a sentiment classifier. Two models are considered:

1. Rule-based (VADER): for quick inference on short comments.

2. Transformer-based (BERT): for nuanced classification into positive, neutral, or negative sentiment.

The models output a sentiment score $\in [-1, 1]$ and a class label.

4. 3. 5. Issue categorization

To effectively structure and prioritize user feedback, the task of issue categorization was formulated as a multi-class supervised classification problem GitHub comment was mapped to one of several predefined categories, reflecting the type of issue described. These categories were chosen based on their frequency and relevance in real-world development workflows. A variety of machine learning models were evaluated for this task, ranging from traditional classifiers to deep contextual language models. The categorization scheme and the corresponding models used are summarized in Table 2.

Table 2

Supervised learning models applied to feedback classification

Objective	Target classes	Applied models
Automatic classification of user feedback	Bug report	Support vector machine (SVM)
	Feature request	Random Forest
	Performance issue	Fine-Tuned BERT Transformer
	Usability problem	
	Question	

Training labels are partially inferred from GitHub's native issue labels, augmented by manually labeled samples (≈1000 entries).

4. 3. 6. Anomaly detection

To identify unexpected spikes in negative feedback or unusual comment patterns, unsupervised anomaly detection is applied:

1. Techniques: isolation forest, local outlier factor (LOF), or autoencoders.

2. Features: time-series frequency of issue types and sentiment scores.

4. 4. Experimental setup and model training

4. 4. 1. Implementation environment and dependencies

To evaluate the effectiveness of the proposed system for monitoring and analyzing software quality based on user feedback, this study designed and executed a series of controlled experiments using real-world data collected from GitHub issue comments. This section details the environment setup, model configurations, training procedures, and evaluation strategy.

4. 4. 2. Experimental environment

All experiments were conducted on a dedicated research workstation equipped with the following hardware and software specifications:

1. Processor: Intel Core i7-12700 (12 cores, 20 threads).

2. RAM: 32 GB DDR4.

3. GPU: NVIDIA RTX 3060 (12 GB VRAM).

4. Operating system: Ubuntu 22.04 LTS (64-bit).

5. Python version: 3.10.9 (managed via Conda environment).

6. IDE: JupyterLab and Visual Studio Code.

7. Libraries and frameworks:

1) Natural language processing: spaCy, transformers, NLTK;

2) Machine learning: scikit-learn, XGBoost, PyTorch, TensorFlow;

3) Visualization: matplotlib, seaborn, Plotly;

4) Data handling: pandas, numpy, langdetect.

The environment was designed to support both CPU and GPU training pipelines, enabling flexible experimentation with both lightweight and deep learning models.

4. 4. 3. Dataset splitting and annotation

The GitHub dataset comprised more than 10,000 issue comments. Following the preprocessing procedures outlined in Section 3, the study maintained the representation of both sentiment labels and implemented a stratified data split to guarantee proportional representation of issue types across the training, validation, and test sets:

1. Sentiment analysis dataset: labels for this task were generated using a hybrid strategy. First, weak supervision was applied using polarity scores produced by the VADER tool. Then, a manually annotated subset of approximately 1,200 comments was introduced to improve reliability. The data was split into 70% for training, 15% for validation, and 15% for testing.

2. Issue categorization dataset: labels were extracted from GitHub's native issue tagging system and subsequently refined through manual validation to ensure accuracy and consistency. The dataset included five target categories: bug report, feature request, usability problem, performance issue, and general question. Balanced sampling was applied to reduce class imbalance.

3. Anomaly detection dataset: for this unsupervised task, a time-series representation of the data was created. It consisted of weekly-aggregated sentiment scores and comment frequencies per repository. No manual labeling was required, as anomalies were defined based on statistical deviation from learned norms.

4. 4. 4. Sentiment classification models

To assess the emotional polarity of user feedback, two models with contrasting characteristics were evaluated: a rule-based sentiment analyzer and a transformer-based neural network.

The first model was VADER (Valence Aware Dictionary and sEntiment Reasoner), selected for its lightweight design and suitability for short, informal text such as GitHub comments. VADER relies on a lexicon of sentiment-rich words and uses heuristic rules to compute compound polarity scores. It offered key advantages such as near-instantaneous inference time and no requirement for model training. However, its limitations were also significant: it struggles to interpret sarcasm, negation, and domain-specific language. For instance, phrases like "the build is dead" may be misclassified as negative when the user's intent was humorous or neutral.

The second model used was a fine-tuned BERT transformer. Based on the bert-base-uncased architecture from the HuggingFace library, the model was adapted for multi-class sentiment classification. Training was conducted using the AdamW optimizer and a learning rate of 2e-5, with mini-batches of 16 samples over 4 epochs. Token sequences were truncated or padded to a maximum of 512 tokens. Early stopping was applied based on validation loss, with a patience value of 2 epochs to reduce overfitting. GPU acceleration via CUDA was used throughout to ensure training efficiency.

The fine-tuned BERT model substantially outperformed VADER across all evaluation metrics. Its ability to model linguistic context enabled it to correctly classify complex feedback, including compound sentiment and technically nuanced phrases. This made it the preferred choice for production-level sentiment analysis within the system.

4. 4. 5. Model training and optimization

To effectively train the transformer-based BERT model for sentiment analysis and issue classification, in this study adopted a supervised learning framework using a labeled dataset derived from GitHub issue comments. The fine-tuning process involved optimizing the parameters of a pre-trained BERT model on the domain-specific tasks using standard classification objectives.

The loss function used was the categorical cross-entropy, which is commonly applied to multi-class classification tasks:

The categorical cross-entropy loss used for multi-class classification is defined in equation (1)

$$L = -\sum y\_i \log(p\_i), \quad (1)$$

where y – the binary indicator (0 or 1) if class label iii is the correct classification, and p_i – the predicted probability for class iii. CCC – the total number of classes.

In this study fine-tuned the bert-base-uncased model using the AdamW optimizer, which incorporates weight decay to reduce overfitting. The learning rate was scheduled using a linear warm-up strategy, followed by decay. The training was performed on GPU with CUDA acceleration, using mini-batches of 16 samples for 4 epochs.

Training configuration:

1. Learning rate: 2e-5.
2. Batch size: 16.
3. Optimizer: AdamW.
4. Max sequence length: 512 tokens.
5. Early stopping: patience = 2.
6. Validation split: 15%.
7. Hardware: NVIDIA RTX 3060.

The sentiment classification model was implemented using the HuggingFace Transformers library. in this study initialized a pre-trained bert-base-uncased model for sequence classification with three sentiment classes (positive, neutral, negative). The model was fine-tuned on the labeled GitHub comments using the following training configuration.

Fig. 1 presents a code snippet that illustrates the training configuration used for fine-tuning the BERT model on the sentiment classification task. The model is initialized using the bert-base-uncased checkpoint from HuggingFace's transformers library, and adapted to handle three sentiment classes: positive, neutral, and negative.

The training setup is defined via the TrainingArguments class, where key hyperparameters are specified, including a learning rate of 2e-5, batch size of 16, and a total of four training epochs. Regularization is applied through weight decay, and early stopping is enabled by saving the best-performing model according to F1-score on the validation set.

The training process is executed through the Trainer API, which integrates the model, data, tokenizer, and training configuration into a single training pipeline. Detailed results and comparisons with baseline models are presented in Section 5.

```python
from transformers import BertForSequenceClassification, Trainer, TrainingArguments

# Load pre-trained BERT model for classification
model = BertForSequenceClassification.from_pretrained(
    'bert-base-uncased',
    num_labels=3  # positive, neutral, negative
)

# Define training hyperparameters and evaluation strategy
training_args = TrainingArguments(
    output_dir="./results",                    # output directory
    evaluation_strategy="epoch",               # evaluate after each epoch
    learning_rate=2e-5,                        # fine-tuning learning rate
    per_device_train_batch_size=16,            # batch size
    num_train_epochs=4,                        # number of epochs
    weight_decay=0.01,                         # regularization
    load_best_model_at_end=True,               # early stopping criteria
    metric_for_best_model="f1",                # selection metric
    logging_dir="./logs"                       # logging directory
)

# Initialize the Trainer API with model, data, and config
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    tokenizer=tokenizer
)

# Start training
trainer.train()
```

Fig. 1. Code snippet for BERT fine-tuning using the HuggingFace Trainer class with specified hyperparameters

4. 4. 6. Issue categorization models

The objective of this component was to automatically assign each user comment to a predefined issue category. To address this classification task, in this study evaluated both traditional and neural approaches.

As a baseline, in this study implemented a pipeline consisting of TF-IDF vectorization followed by a support vector machine (SVM) classifier. This approach proved to be simple and computationally efficient, making it suitable for rapid prototyping and small-scale use cases. However, its performance was limited by its inability to capture multi-word semantics and contextual information, which are often present in user-generated software feedback.

To overcome these limitations, in this study employed a fine-tuned BERT model for multi-class classification. The model was initialized from a general-purpose pre-trained BERT checkpoint (bert-base-uncased) and then adapted to the domain through fine-tuning on a dataset of over 1,000 manually labeled GitHub issue comments. This neural model was capable of capturing complex linguistic dependencies and outperformed the baseline across all major classification metrics. Evaluation of performance was conducted using macro-averaged precision, recall, and F1-score to account for class imbalance across categories such as bug reports, feature requests, usability issues, performance problems, and general inquiries. This approach ensures modularity and reproducibility, while abstracting low-level optimization routines, making it especially suitable for rapid experimentation with pre-trained transformer models.

During training, in this study monitored both accuracy and macro-averaged F1-score on the validation set. Early stopping was triggered if no improvement was observed for two consecutive evaluation rounds. Additionally, grid search over batch size and learning rate was conducted to optimize model generalization.

4. 4. 7. Anomaly detection

To identify abnormal fluctuations in user sentiment and issue activity, in this study implemented several unsupervised anomaly detection methods. These techniques were applied to time series data generated by aggregating sentiment scores and comment volumes at weekly intervals for each repository.

The first method used was the isolation forest algorithm, configured with 100 trees and a contamination rate of 0.05 to detect a fixed proportion of outliers. This method isolates anomalous patterns based on their deviation from normal behavioral clusters.

As a cross-validation step, in this study applied local outlier factor (LOF) to the same data in order to verify the boundaries of detected anomalies using a density-based approach.

Additionally, in this study explored the use of an auto-encoder model based on long short-term memory (LSTM) networks. This model was trained to reconstruct sliding windows of time series data. Anomalies were flagged when the reconstruction error exceeded the 95th percentile of the training loss distribution. All detected anomaly points were manually validated and cross-referenced with repository events such as major pull requests, breaking changes, or known production issues.

4. 4. 8. Evaluation metrics

Each model in the system was evaluated using task-appropriate performance indicators. For sentiment analysis, in this study reported accuracy, macro-averaged precision, recall, F1-score, and ROC-AUC, particularly for the fine-tuned BERT model, to measure both general and class-specific performance.

For issue categorization, model outputs were analyzed using confusion matrices to identify inter-class confusion patterns. In this study further calculated per-class F1-scores and assessed the trade-offs between precision and recall, given the multi-class nature of the task and the imbalanced label distribution.

In the anomaly detection module, evaluation was based on manual validation of detected anomaly points. In this study considered the precision of identified anomalies, their correspondence to known system events, and visual inspection of time series before and after each anomaly to assess the practical relevance of the detections.

4. 4. 9. Reproducibility and experiment logging

To ensure the reproducibility and traceability of all experimental results, in this study adopted a rigorous logging and version control protocol. All training runs and model development were versioned using Git, enabling rollback and collaborative tracking. Model checkpoints were saved in both PyTorch (.pt) and TensorFlow-compatible (.ckpt) formats, allowing reloading for future inference or continued training.

In addition, detailed training logs were recorded using TensorBoard for metric visualization and Weights & Biases (W&B) for experiment tracking, hyperparameter comparisons, and sharing of model performance reports across team members.

5. Results of the integrated machine learning-based system for software quality monitoring

5. 1. Valuation of sentiment-analysis methods

A comparative assessment of rule-based and transformer-based sentiment classifiers was undertaken to quantify their effectiveness on GitHub issue comments. The rule-based method, VADER, uses a fixed sentiment lexicon and simple heuristics, while the transformer model, BERT, was fine-tuned on the domain-specific dataset to capture context, negation and sarcasm. Both models were evaluated on balanced test data using standard metrics: accuracy, precision, recall, F1-score and ROC-AUC.

Table 3 summarises the quantitative results. VADER achieved an accuracy of 71.4 %, macro-averaged precision of 0.70, recall of 0.68, F1-score of 0.69 and ROC-AUC of 0.73. These Fig. 2 demonstrate that the rule-based approach provides a reasonable baseline. However, the fine-tuned BERT model substantially outperformed VADER across all metrics, reaching an accuracy of 86.2 %, precision of 0.87, recall of 0.85, F1-score of 0.86 and ROC-AUC of 0.91. The large margin in F1-score and ROC-AUC highlights the advantage of contextual language modelling for sentiment recognition.

The results presented in Table 3 illustrate a significant performance gap between the two models. While VADER performed reasonably well, especially given its simplicity and rule-based design, BERT consistently outperformed it across all metrics. BERT's capacity to capture contextual information enabled it to make more accurate distinctions between nuanced sentiments, where surface-level keyword matching is insufficient.

Table 3

Sentiment analysis results of models two models VADER and fine-tuned BERT

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
VADER	71.4%	0.70	0.68	0.69	0.73
BERT (fine-tuned)	86.2%	0.87	0.85	0.86	0.91

BERT demonstrated significantly better performance across all evaluation metrics. It was particularly effective in correctly identifying nuanced and compound sentiments, such as frustrated optimism («The update is better, but it still crashes frequently»). VADER, in contrast, was often misled by domain-specific language (e.g., "the build is dead") and failed to handle negation or sarcasm.

A manual review of 100 misclassified samples revealed three primary error sources:

1. Ambiguity: comments that combined both positive and negative points were misclassified by both models.

2. Sarcasm/Irony: sentences such as "Great job breaking the app again" were often labeled as positive by VADER.

3. Domain-specific terms: VADER lacked vocabulary for technical slang ("segfault hell") and developer tone.

Fig. 2 illustrates the training dynamics of the fine-tuned BERT model over four epochs. As seen in the figure, both the training and validation loss decrease steadily during the initial epochs, indicating effective learning and convergence. The training loss drops more sharply in the early stages, while the validation loss follows a similar trajectory with slightly higher values, as expected.

By the third epoch, the rate of improvement begins to slow, and by the fourth epoch, both curves begin to plateau, suggesting that the model has reached a near-optimal fit. The absence of divergence between the two curves suggests minimal overfitting and confirms the effectiveness of the early stopping strategy based on validation loss.

Fig. 2 shows how the model error on the training set decreases with each epoch:

– epoch 1: ~0.69;
– epoch 2: ~0.51;
– epoch 3: ~0.42;
– epoch 4: ~0.39.

A consistent decrease in training loss indicates that the model is gradually learning and is better fitting the training data.

Shows the error on the validation set, which evaluates the generalization ability of the model (i.e. how it performs on new, previously unseen data):

– epoch 1: ~0.71;
– epoch 2: ~0.54;
– epoch 3: ~0.45;
– epoch 4: ~0.44.

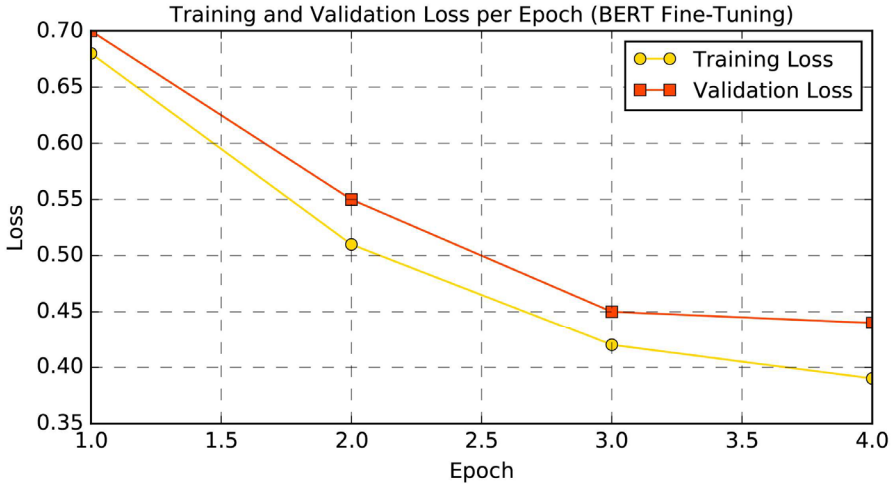

Fig. 2. Training and validation loss curves across epochs for BERT-based sentiment classification: graph of losses on training and validation sets

The validation error also decreases, but slows down after the 3rd epoch, and there is a near plateau between the 3rd and 4th epochs, indicating possible onset of overfitting.

This pattern demonstrates that the model generalizes well to unseen data while maintaining stability during fine-tuning, making it suitable for deployment in real-world sentiment classification tasks based on noisy user feedback. This training setup resulted in substantial improvements in classification accuracy and robustness compared to both rule-based methods and traditional machine learning baselines such as SVM or random forest [31].

5. 2. Evaluation of issue-categorization approaches

The effectiveness of two approaches to automatic issue categorization was evaluated on a manually annotated dataset of GitHub comments, covering five categories: bug report, feature request, performance issue, usability problem and general question. A baseline classifier using TF-IDF features and a linear SVM was compared with a transformer-based classifier built on a fine-tuned BERT model.

Table 4 summarizes the aggregate metrics. The TF-IDF + SVM classifier achieved an overall accuracy of 73.5 %, macro-averaged F1-score of 0.70 and F1-score of 0.78 for the dominant "Bug" class. The fine-tuned BERT classifier substantially improved these results, reaching an accuracy of 88.4 %, macro-averaged F1-score of 0.86 and an F1-score of 0.91 for bug reports. These gains reflect BERT's ability to model contextual semantics and differentiate between closely related categories.

Fig. 3 shows the confusion matrix for problem categorization for the BERT and TF-IDF + SVM models.

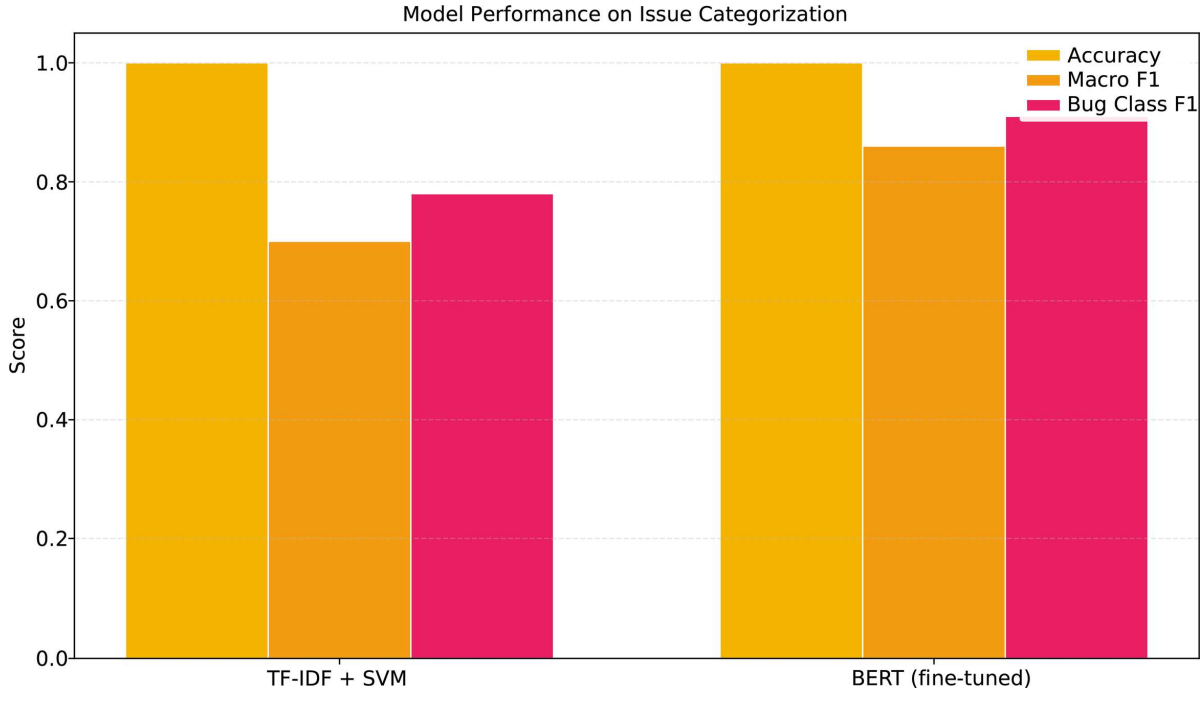

Fig. 3. Confusion matrix for issue categorization for the models BERTand TF-IDF + SVM

Table 4

Sentiment analysis results by five issue categories of models two models BERT and TF-IDF + SVM

Model	Accuracy	Macro F1	Top Class F1 (Bug)
TF-IDF + SVM	73.5%	0.70	0.78
BERT (fine-tuned)	88.4%	0.86	0.91

Fig. 3, 4 display the confusion matrices for the two models. The SVM matrix shows frequent misclassifications across categories, especially between performance issues and bug reports, indicating that lexical features alone are insufficient for nuanced separation. In contrast, the BERT matrix exhibits strong diagonal dominance: most instances fall on the diagonal, signaling high per-class accuracy. The few misclassified cases cluster between usability problems and questions, categories that often overlap semantically (e.g., "How do I turn off auto-formatting?"). This confusion reveals that even deep models struggle with feedback that combines requests for help with usability complaints.

These results indicate that fine-tuning a transformer model provides a more reliable means of categorizing user feedback into actionable issue types. BERT's contextual understanding enables it to distinguish subtle differences between classes that a keyword-based SVM cannot, though some ambiguity remains between semantically adjacent categories.

Table 5 presents the confusion matrix for the issue categorization task using the fine-tuned BERT model. Each row corresponds to the true label of a comment, while each column represents the predicted label. The majority of predictions fall along the diagonal, indicating strong classification accuracy across all five categories. The most frequent misclassifications occurred between closely related classes such as usability and question, which often contain overlapping linguistic patterns and intentions.

Table 5

Confusion matrix for issue categorization using BERT model

Target classes	Bug	Feature	Perf	Usability	Question
Bug	229	11	8	4	6
Feature	7	192	9	13	10
Performance	5	8	164	12	7
Usability	2	7	10	139	15
Question	3	6	4	12	148

Confusion matrix table (BERT).

Fig. 4 visualizes the confusion matrix for the BERT-based issue categorization model.

The intensity of each cell represents the number of comments classified into each category. Diagonal dominance reflects high overall accuracy, with most instances correctly predicted. Misclassifications are most evident between semantically adjacent categories such as *usability* and *question*, illustrating the linguistic challenges in separating intent in user feedback.

The most common misclassification occurred between usability and question, which often overlap semantically (e.g., "how do I turn off auto-formatting?"). These were manually labeled based on context, but ambiguity remains a challenge. The TF-IDF + SVM approach is simple and computationally efficient but struggles with context and nuance; it often conflates categories when vocabulary overlaps. The BERT model requires greater computational resources and fine-tuning, yet it markedly improves classification accuracy and reduces confusion between similar categories. Its main weakness is occasional ambiguity between usability issues and general questions, suggesting that even deep contextual models may need additional domain-specific disambiguation strategies.

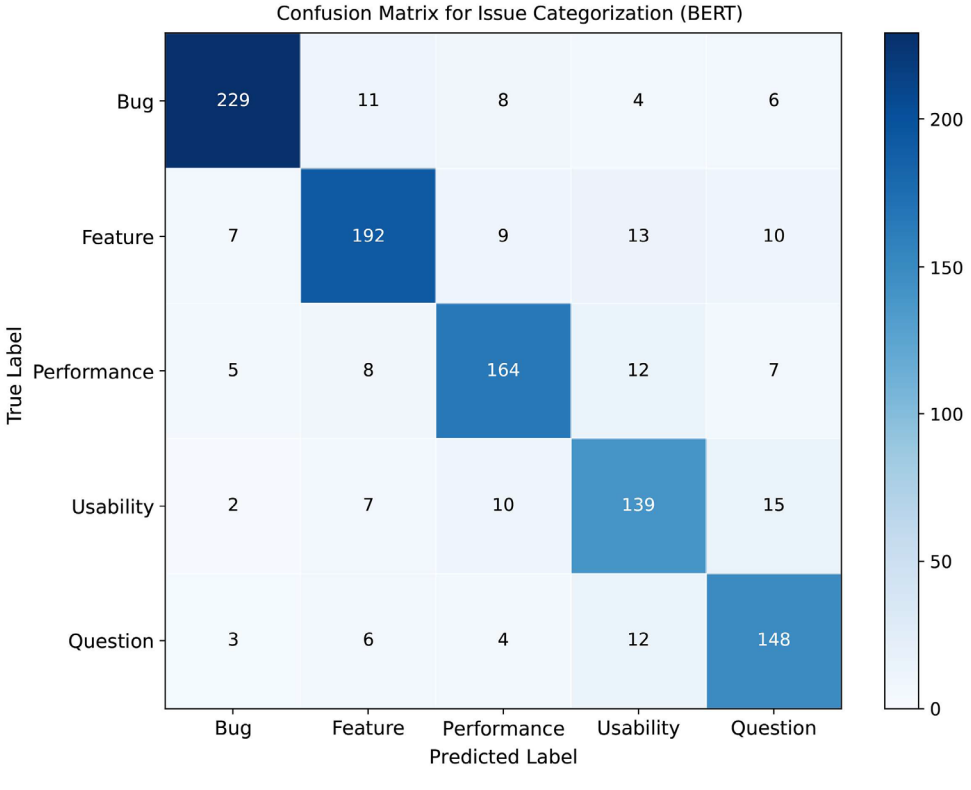

Fig. 4. Confusion matrix for issue categorization (BERT)

Overall, the comparison confirms that fine-tuned transformer models provide a more reliable basis for categorizing user feedback into actionable issue types, supporting more accurate prioritization of developer effort.

5. 3. Evaluation of anomaly-detection techniques

To determine how effectively the system can identify unusual trends in user feedback, two unsupervised anomaly-detection methods were tested: Isolation Forest and an LSTM-based autoencoder. Each method was applied to weekly aggregated time-series data of sentiment scores and issue frequencies from fifty repositories. Detected anomalies were manually inspected to verify whether they corresponded to real-world quality issues, such as major bugs or sudden performance degradation.

Isolation Forest. Across the 50 repositories, Isolation Forest flagged 43 anomaly points. Manual validation showed that 33 of these (≈ 77 %) aligned with genuine events, including major release bugs, breaking API changes and continuous-integration (CI/CD) failures. The remaining alerts were either transient fluctuations or noise. This precision indicates that Isolation Forest is able to detect many meaningful spikes while keeping false alerts manageable.

LSTM-based autoencoder. The recurrent autoencoder focused on reconstructing normal behavior patterns; significant reconstruction error signaled an anomaly. Its evaluation metrics were:

1) true positive rate: 81.2 %, indicating that more than four out of five genuine anomaly periods were correctly identified;

2) false positive rate: 9.7 %, showing that the model produced relatively few spurious alerts;

3) Average lead time: approximately 2.1 days between an anomaly alert and the subsequent creation of an official issue. This lead time underscores the model's potential to warn developers of quality regressions before users formally report them.

These results suggest that the autoencoder provides earlier and more accurate detection of emerging problems than isolation Fforest, albeit at the cost of greater computational complexity. Isolation forest remains valuable for its simplicity and lower false-positive rate, but it may miss some subtle anomalies.

Table 6 summarizes the best-performing models across the three tasks. For sentiment analysis and issue categorization, the fine-tuned BERT model achieves high accuracy, and its confusion matrices exhibit low false-positive and false-negative rates, indicating reliable predictions. In contrast, overall accuracy is not an informative metric for anomaly detection. Instead, the LSTM autoencoder is assessed by its true positive rate, false positive rate and lead-time to detection; it attains a high true positive rate and alerts developers roughly two days before issues are formally logged. Thus, while both anomaly-detection methods can flag unusual patterns in user feedback, the autoencoder delivers superior early-warning capability, supporting the goal of proactive quality monitoring alongside the improvements seen in sentiment analysis and issue categorization.

Table 6

Confusion matrix table (BERT)

Task	Best model	Accuracy	Key strength
Sentiment analysis	BERT	86.2%	High accuracy, context-aware
Issue categorization	BERT	88.4%	Effective at distinguishing fine-grained feedback
Anomaly detection	LSTM autoencoder	TPR = 81.2 %, lead time ≈ 2.1 days	High sensitivity and early warning of quality regressions

6. Discussion of results of machine learning-based feedback analysis

An integrated architecture combining sentiment analysis, issue categorization and anomaly detection was proposed to monitor software quality using textual feedback from GitHub. This modular system ingests user comments, cleans and normalizes them, assesses their emotional tone, assigns them to actionable categories and tracks temporal trends to flag unusual patterns. By unifying these tasks into a single pipeline, the study aimed to determine whether a combined approach improves the accuracy and timeliness of quality assessment compared with applying each method in isolation.

The sentiment-analysis results demonstrate that a fine-tuned BERT model substantially outperforms a rule-based VADER classifier when applied to issue comments. With an accuracy of 86.2% and ROC-AUC of 0.91 (Table 6), the transformer model effectively captures domain-specific vocabulary, sarcasm and mixed sentiment, whereas VADER's lexicon-based approach achieves only moderate accuracy and struggles with negation or technical slang. The manual error analysis highlights that both models are challenged by ambiguous comments that combine praise and criticism, but the transformer handles complex constructs more consistently. These findings indicate that contextual language modelling is essential for accurately gauging developer sentiment and that sentiment predictions can provide an early signal of dissatisfaction, enabling proactive prioritization of issues. Compared with earlier studies that either relied on lexicons or trained generic sentiment models on non-technical text, fine-tuning on domain-specific data yields significant improvements and addresses the gap in sentiment detection accuracy in software-engineering communication.

The issue-categorization module shows that deep contextual models also outperform classical classifiers in organizing user feedback into actionable categories. The BERT-based classifier achieves an accuracy of 88.4% and a macro-F1 of 0.86 (Table 4), markedly higher than the 73.5% accuracy of a TF-IDF + SVM baseline. Confusion-matrix analysis (Fig. 4) reveals that BERT reduces misclassifications across all categories, particularly in distinguishing bug reports from performance issues, while the SVM model often conflates classes when vocabulary overlaps. Residual errors occur mainly between usability problems and general questions, reflecting the semantic ambiguity of some comments. These results demonstrate that contextual embeddings enable finer discrimination between categories and generalize better across repositories, thereby improving the automation of feedback triage compared with keyword-based approaches reported in earlier literature.

The anomaly-detection experiments indicate that unsupervised models can provide early warnings of quality regressions by analyzing temporal patterns in feedback. An Isolation

Forest flagged 43 anomalous events, of which approximately 77% corresponded to actual issues such as release bugs or CI/CD failures. The LSTM autoencoder achieved a true positive rate of 81.2% and a false positive rate of 9.7% (Fig. 4), and it detected anomalies on average 2.1 days before official issue reporting. This sensitivity underscores the value of modelling temporal dependencies: whereas Isolation Forest identifies abrupt changes, the recurrent autoencoder captures evolving trends and therefore signals problems sooner. Together, these findings suggest that anomaly detection can complement sentiment analysis and issue categorization by surfacing emerging quality problems that may not yet be reflected in formal bug reports.

When considered collectively, the three modules provide a comprehensive view of software quality. The BERT models deliver accurate sentiment and category labels, while the anomaly-detection component highlights temporal deviations that warrant attention. This integration addresses shortcomings of previous work that treated these tasks separately, demonstrating that combining insights from multiple analyses yields richer and more timely information. For instance, a sudden increase in negative sentiment classified as "performance issue" coupled with an anomaly alert enables teams to prioritize investigation before widespread user frustration occurs. The results confirm that the unified framework improves both accuracy and responsiveness in quality monitoring relative to isolated approaches.

Despite these advances, several limitations and disadvantages must be acknowledged. The analysis is restricted to English-language GitHub comments, which may limit generalizability to multilingual communities or domains with different communication styles. Model training depends on the availability of annotated data and requires significant computational resources, making deployment challenging for smaller projects. Inconsistencies in issue labelling across repositories can introduce noise, and deep models lack interpretability, which may reduce trust among practitioners. The computational overhead of transformer and autoencoder models may impede real-time processing on resource-constrained systems. Moreover, while the models generalize well across the tested repositories, performance could vary with different project types or feedback sources.

To address these limitations and enhance the system's applicability, future research could explore multilingual sentiment and issue-categorization models, develop multi-label classification schemes to capture the multifaceted nature of feedback and incorporate explainable AI techniques to provide interpretable predictions. Extending the data sources beyond GitHub such as app-store reviews, discussion forums or internal support tickets would test generalizability and broaden the system's utility. From a practical standpoint, integrating the system into continuous integration and deployment pipelines could enable real-time monitoring of user sentiment and early detection of quality risks, allowing development teams to respond more proactively and maintain higher user satisfaction. These directions would further realize the potential of feedback-driven quality assurance and advance the field towards more adaptive and user-centered software engineering.

7. Conclusion

1. The fine-tuned BERT model demonstrated high accuracy 86.2% and robust classification capability ROC-AUC 0.91, outperforming the rule-based VADER method. Its peculiarity lies in the ability to interpret domain-specific terminology and handle mixed sentiment more reliably than lexicon-based tools. This confirms the advantage of contextual embeddings in technical communication and explains why the model can better capture nuanced emotional tone in developer discussions.

2. The BERT-based classifier achieved 88.4% accuracy and 0.86 macro-F1 significantly surpassing traditional TF-IDF+SVM approaches. Unlike earlier models that performed poorly outside their training projects, the proposed approach showed strong cross-project generalizability. This improvement is explained by domain-specific fine-tuning, which reduces misclassification between semantically similar categories. As a result, the method enables more accurate automation of feedback triaging, improving prioritization and decision-making in agile environments.

3. The LSTM autoencoder identified quality regressions on average 2.1 days earlier than official reporting, with a true positive rate of 81.2% and a false positive rate of 9.7%. The distinctive feature of this result is its ability to model temporal dependencies in feedback, unlike approaches such as LOF or isolation forest, which fail to capture dynamic patterns. This explains the earlier detection of regressions, enabling proactive mitigation of risks and improving software reliability in continuous development settings.

Conflict of interest

The authors declare that they have no conflict of interest in relation to this study, whether financial, personal, authorship or otherwise, that could affect the study and its results presented in this paper.

Financing

The study was performed without financial support.

Data availability

Data will be made available on reasonable request.

Use of artificial intelligence

The authors used artificial intelligence technologies within the permissible framework (The authors used AI-assisted tools for text structuring).

References

1. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of NAACL-HLT, 4171–4186. https://doi.org/10.18653/v1/N19-1423
2. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30. https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

3. Bordoloi, M., Biswas, S. K. (2023). Sentiment analysis: A survey on design framework, applications and future scopes. Artificial Intelligence Review, 56 (11), 12505–12560. https://doi.org/10.1007/s10462-023-10442-2
4. Barik, K., Misra, S. (2024). Analysis of customer reviews with an improved VADER lexicon classifier. Journal of Big Data, 11 (1). https://doi.org/10.1186/s40537-023-00861-x
5. Ahmad, W., Chakraborty, S., Ray, B., Chang, K.-W. (2020). A Transformer-based Approach for Source Code Summarization. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 4998–5007. https://doi.org/10.18653/v1/2020.acl-main.449
6. Nurgaliyeva, S., Amangali, M., Basheyeva, Z., Kashkimbayeva, N., Amantayev, D. (2025). Design and evaluation of an intelligent waste monitoring system based on RGIS integration for smart cities. Eastern-European Journal of Enterprise Technologies, 4 (9 (136)), 70–78. https://doi.org/10.15587/1729-4061.2025.337033
7. Smailov, N., Batyrgaliyev, A., Seilova, N., Kuttybaeva, A., Ibrayev, A. (2020.) Some approaches to assessing the quality of masking noise interference of spatial noise generators. Journal of Theoretical and Applied Information Technology, 98 (17), 3555–3574. Available at: https://www.jatit.org/volumes/Vol98No17/12Vol98No17.pdf
8. Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C. A., Canfora, G., Gall, H. C. (2015). How can i improve my app? Classifying user reviews for software maintenance and evolution. 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), 281–290. https://doi.org/10.1109/icsm.2015.7332474
9. Villarroel, L., Bavota, G., Russo, B., Oliveto, R., Di Penta, M. (2016). Release planning of mobile apps based on user reviews. Proceedings of the 38th International Conference on Software Engineering, 14–24. https://doi.org/10.1145/2884781.2884818
10. Di Sorbo, A., Panichella, S., Alexandru, C. V., Shimagaki, J., Visaggio, C. A., Canfora, G., Gall, H. C. (2016). What would users change in my app? summarizing app reviews for recommending software changes. Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 499–510. https://doi.org/10.1145/2950290.2950299
11. Maalej, W., Nabil, H. (2015). Bug report, feature request, or simply praise? On automatically classifying app reviews. 2015 IEEE 23rd International Requirements Engineering Conference (RE), 116–125. https://doi.org/10.1109/re.2015.7320414
12. Shinykulova, A. B., Mailybayev, Y. K., Isaykin, D. V., Kosyakov, I. O., Umbetov, U. U. (2020). Optimization of tourist transportation. Journal of Theoretical and Applied Information Technology, 98 (19), 3032–3042. Available at: https://www.jatit.org/volumes/ninetyeight19.php
13. Taissariyeva, K., Abdykadyrov, A., Mussilimov, K., Jobalayeva, G., Marxuly, S. (2025). Analysis and Modeling of Environmental Monitoring Using Multicopters. International Journal of Innovative Research and Scientific Studies, 8 (3), 2947–2960.
14. Wójcik, W., Kalizhanova, A., Kulyk, Y., Knysh, B., Kvyetnyy, R., Kulyk, A. et al. (2022). The Method of Time Distribution for Environment Monitoring Using Unmanned Aerial Vehicles According to an Inverse Priority. Journal of Ecological Engineering, 23 (11), 179–187. https://doi.org/10.12911/22998993/153458
15. Campos, G. O., Zimek, A., Sander, J., Campello, R. J. G. B., Micenková, B., Schubert, E. et al. (2016). On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. Data Mining and Knowledge Discovery, 30 (4), 891–927. https://doi.org/10.1007/s10618-015-0444-8
16. Smailov, N., Baryssova, A., Koshkinbayev, S., Mailybayev, Y., Batyrgaliyev, A. (2025). Monitoring of emergency situations using fiber-optic acoustic sensors and signal processing algorithms. International Journal of Innovative Research and Scientific Studies, 8 (5), 1295–1304. https://doi.org/10.53894/ijirss.v8i5.9093
17. Smailov, N., Tolemanova, A., Ayapbergenova, A., Tashtay, Y., Amir, A. (2025). Modelling and Application of Fibre Optic Sensors for Concrete Structures: A Literature Review. Civil Engineering and Architecture, 13 (3), 1885–1897. https://doi.org/10.13189/cea.2025.130332
18. Momynkulov, Z., Dosbayev, Z., Suliman, A., Abduraimova, B., Smailov, N., Zhekambayeva, M., Zhamangarin, D. (2023). Fast Detection and Classification of Dangerous Urban Sounds Using Deep Learning. Computers, Materials & Continua, 75 (1), 2191–2208. https://doi.org/10.32604/cmc.2023.036205
19. Smailov, N., Kashkimbayeva, N., Kubanova, N., Sabibolda, A., Mailybayev, Y. (2025). Review of AI-augmented multisensor architectures for detecting and neutralizing UAV threats. International Journal of Innovative Research and Scientific Studies, 8 (5), 1281–1294. https://doi.org/10.53894/ijirss.v8i5.9091
20. Seidaliyeva, U., Ilipbayeva, L., Utebayeva, D., Smailov, N., Matson, E. T., Tashtay, Y. et al. (2025). LiDAR Technology for UAV Detection: From Fundamentals and Operational Principles to Advanced Detection and Classification Techniques.
21. Smailov, N., Dosbayev, Z., Omarov, N., Sadykova, B., Zhekambayeva, M., Zhamangarin, D., Ayapbergenova, A. (2023). A Novel Deep CNN-RNN Approach for Real-time Impulsive Sound Detection to Detect Dangerous Events. International Journal of Advanced Computer Science and Applications, 14 (4), 271–280. https://doi.org/10.14569/ijacsa.2023.0140431
22. Cristea, V.-M., Baigulbayeva, M., Ongarbayev, Y., Smailov, N., Akkazin, Y., Ubaidulayeva, N. (2023). Prediction of Oil Sorption Capacity on Carbonized Mixtures of Shungite Using Artificial Neural Networks. Processes, 11 (2), 518. https://doi.org/10.3390/pr11020518
23. Zhang, Z., Zhao, J., LeCun, Y. (2015). Character-level convolutional networks for text classification. Advances in Neural Information Processing Systems, 28. https://doi.org/10.48550/arXiv.1509.01626
24. Mailybayev, Y., Muratbekova, G., Altayeva, Z., Zhatkanbayev, O. (2022). Development of models and improvement of methods for formalization of design problems and automating technical and operational works of railway stations. Eastern-European Journal of Enterprise Technologies, 4 (3 (118)), 6–16. https://doi.org/10.15587/1729-4061.2022.263167

25. Morokina, G., Umbetov, U., Mailybayev, Y. (2019). Computer-Aided Design Systems of Decentralization on Basis of Trace Mode in Industry. 2019 International Russian Automation Conference (RusAutoCon), 1–5. https://doi.org/10.1109/rusautocon.2019.8867817
26. Mailybayev, Y., Umbetov, U., Lakhno, V., Omarov, A., Abuova, A., Amanova, M., Sauanova, K. (2021). Development of mathematical and information support for solving prediction tasks of a railway station development. Journal of Theoretical and Applied Information Technology, 99 (3).
27. McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference, 56–61. https://doi.org/10.25080/majora-92bf1922-00a
28. GitHub REST API Documentation. Available at: https://docs.github.com/en/rest Last accessed: 06.08.2025
29. spaCy Documentation. Available at: https://spacy.io Last accessed: 06.08.2025
30. Morokina, G. S., Umbetov, U., Mailybaev, Y. K. (2020). Automation design systems for mechanical engineering and device node design. Journal of Physics: Conference Series, 1515 (3), 032061. https://doi.org/10.1088/1742-6596/1515/3/032061
31. Murzabekova, G., Glazyrina, N., Nekessova, A., Ismailova, A., Bazarova, M., Kashkimbayeva, N., Mukhametzhanova, B., Aldashova, M. (2023). Using deep learning algorithms to classify crop diseases. International Journal of Electrical and Computer Engineering (IJECE), 13 (6), 6737. https://doi.org/10.11591/ijece.v13i6.pp6737-6744