International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 11 | Nov 2026 www.irjet.net p-ISSN: 2395-0072

INTELLIGENT CI/CD FAILURE DIAGNOSIS SYSTEM

Prof. Madhu N Hiremath[1], Rahul R Kattimani[2], Sandhya Belavi[2], Sanjana N[2], Yashasvi Kote[2]

[1]Assistant Professor, Department of Computer Science and Engineering, Bapuji Institute of Engineering and Technology, Davanagere, Karnataka, India.
[2]Student, Department of Computer Science and Engineering, Bapuji Institute of Engineering and Technology, Davanagere, Karnataka, India.

Abstract - *A Continuous Integration and Continuous Deployment (CI/CD) pipelines play an important role in modern software development by automating code integration, building, testing, and deployment. However, pipeline failures caused by compilation errors, dependency conflicts, test failures, permission issues, and configuration problems can interrupt software delivery. Identifying the cause of these failures is difficult because CI/CD tools generate lengthy and unstructured log files that developers must often inspect manually. This paper presents an Intelligent CI/CD Failure Diagnosis System that assists developers in analysing pipeline logs and identifying probable failure categories. The system accepts manually submitted logs and supports retrieval of failed workflow logs through GitHub Actions integration. The collected logs undergo preprocessing and cleaning to remove irrelevant information and identify important error messages. A hybrid diagnosis approach combines predefined pattern matching using regular expressions with Term Frequency-Inverse Document Frequency (TF-IDF) feature extraction and a Logistic Regression classifier. The system categorises common failures and displays the probable cause along with general troubleshooting recommendations. A web-based interface developed using Python and Flask presents the diagnosis results, while SQLite maintains the history of previous analyses. Functional testing is performed to verify log submission, failure classification, workflow-log retrieval, and history storage. The proposed system aims to simplify failure analysis, reduce manual log inspection, and provide developers with an accessible tool for understanding CI/CD pipeline failures.*

Key Words: CI/CD, DevOps, Machine Learning, Natural Language Processing, Log Analysis.

1. INTRODUCTION

Modern software development increasingly depends on automation to deliver applications quickly and reliably. Continuous Integration and Continuous Deployment (CI/CD) are important practices in DevOps that automate the process of integrating code changes, executing tests, building applications, and deploying software. Platforms such as Jenkins, GitHub Actions, and GitLab CI/CD are widely used to manage these workflows.

A typical CI/CD pipeline consists of several stages, including source code checkout, dependency installation, compilation, testing, packaging, and deployment. Each stage depends on the successful execution of the previous stages. Even a small issue, such as a syntax error, missing dependency, failed test case, incorrect configuration, or insufficient permission, can interrupt the entire pipeline.

When a pipeline fails, the CI/CD platform generates logs containing execution details, warnings, error messages, timestamps, stack traces, and other system information. Although these logs provide useful information about the failure, they are often lengthy and difficult to understand. Developers must manually inspect the logs to locate the important error messages and determine the probable cause. This process can be time-consuming, especially when the logs contain repeated messages or multiple errors.

Machine Learning and Natural Language Processing techniques provide useful methods for analysing textual information and identifying patterns in log data. Previous research, such as DeepLog, has demonstrated the use of learning-based techniques for detecting abnormal patterns in system logs [1]. Text classification techniques can also be used to represent log messages numerically and classify them into predefined categories.

The proposed Intelligent CI/CD Failure Diagnosis System combines predefined error-pattern matching with a lightweight text-classification approach. It analyses submitted or retrieved pipeline logs, identifies common failure categories, and presents probable causes and troubleshooting recommendations through a web-based dashboard. The system also maintains diagnosis history to help developers review previous failures.

The main contributions of this work are:

- Collection of CI/CD logs through manual submission and GitHub Actions integration.
- Preprocessing and cleaning of unstructured log data.
- Hybrid failure classification using predefined patterns and TF-IDF with Logistic Regression.
- Presentation of failure categories and troubleshooting recommendations.
- Storage and retrieval of previous diagnosis records using SQLite.

2. LITERATURE SURVEY

Log analysis has become an important research area because system logs contain information about application execution, errors, and operational behaviour. Researchers have explored statistical methods, rule-based techniques, Machine Learning algorithms, and deep-learning approaches to identify abnormal events and diagnose system failures.

Du et al. proposed DeepLog, a deep-learning-based approach for detecting anomalies and diagnosing system logs. The method learns patterns from sequences of log events and identifies deviations from expected behaviour [1]. Such approaches demonstrate that log sequences can provide useful information for failure analysis. However, advanced learning-based methods may require suitable training data and additional computational resources.

Traditional Machine Learning methods are also used for log classification and failure prediction. Algorithms such as Support Vector Machine, Random Forest, Logistic Regression, and XGBoost can learn patterns from previously labelled data. These approaches are useful when the input logs can be converted into structured numerical features. However, their performance depends on the quality and representation of the training data.

Natural Language Processing techniques help convert unstructured textual logs into machine-readable representations. TF-IDF is a commonly used feature-extraction technique that assigns importance to words based on their frequency within a document and across a collection of documents. These numerical features can then be used by classification algorithms such as Logistic Regression.

Existing approaches provide useful capabilities for anomaly detection, failure prediction, and log classification. However, several practical limitations remain:

1. Many approaches focus mainly on detecting failures rather than presenting an integrated diagnosis workflow.
2. Manual inspection of lengthy logs is still required in many development environments.
3. Some methods depend on large labelled datasets for effective training.
4. Advanced models may require higher computational resources.
5. Integration with different CI/CD platforms can be challenging.
6. Developers may receive a failure status without sufficiently organised troubleshooting information.

The proposed system focuses on integrating log collection, preprocessing, failure classification, troubleshooting guidance, and diagnosis history within a lightweight web application.

2.1 Research Gap

The literature survey identifies a need for a practical system that combines log analysis and failure classification with an accessible interface. Existing methods often address individual tasks such as anomaly detection or prediction. The proposed system brings together predefined error-pattern matching, text classification, diagnosis presentation, and history storage to assist developers in understanding common CI/CD failures.

3. PROPOSED SYSTEM

The proposed Intelligent CI/CD Failure Diagnosis System is a web-based application designed to analyse CI/CD pipeline logs and provide useful diagnostic information. It accepts logs through manual input and supports retrieval of failed workflow logs through GitHub Actions integration.

The system follows a modular architecture consisting of log collection, preprocessing, feature extraction, failure classification, diagnosis presentation, and history management.

Initially, the user submits a CI/CD log or selects an available failed workflow. The system processes the input to remove unnecessary information and identify relevant error messages. Regular-expression-based pattern matching is applied to identify recognised failure signatures. If no strong pattern is identified, the system uses TF-IDF feature extraction and a Logistic Regression classifier as a fallback classification method.

Based on the identified category, the application displays a probable cause and general troubleshooting recommendations. The diagnosis result is presented through the web interface, and the analysis record is stored in an SQLite database.

3.1 System Architecture

The architecture contains the following modules:

1. Log Collection Module: Accepts manually submitted CI/CD logs and retrieves available workflow information and logs through the GitHub REST API when valid access permissions are configured.
2. Log Preprocessing Module: Cleans the input by removing unnecessary characters, normalising text, and identifying relevant error messages. It also applies pattern-based masking to common sensitive values where supported.
3. Feature Extraction Module: Converts the cleaned textual log into numerical features using TF-IDF. This representation helps the classifier process textual log information.

4. Failure Classification Module: Uses predefined regular-expression patterns to identify common errors. When no strong rule matches, the TF-IDF features are passed to the Logistic Regression classifier.
5. Diagnosis and Recommendation Module: Maps the identified failure category to a probable explanation and general troubleshooting recommendations. These recommendations are intended to assist the developer and should be verified against the original log.
6. Dashboard and History Module: Displays the failure category, diagnosis details, and recommendations. SQLite is used to store and retrieve previous diagnosis records.

3.2 Failure Categories

The system considers common CI/CD failure categories, including:

- Dependency Error
- Build or Compilation Error
- Test Failure
- Permission Error
- Network or Service Error
- Resource Exhaustion
- Authentication or Secret Error
- Timeout or Cancellation
- Unknown Error

These categories help organise different types of pipeline failures and provide a starting point for troubleshooting.

3.3 Proposed Methodology

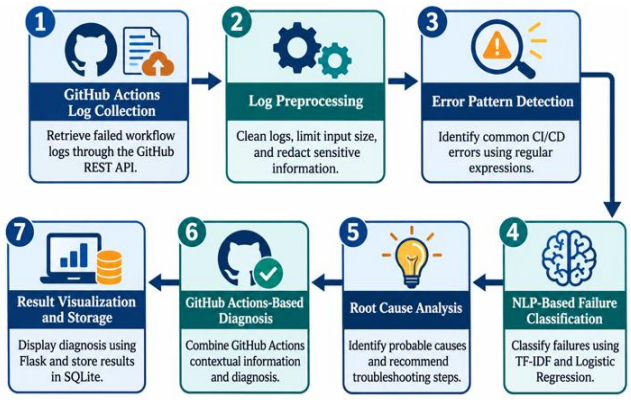

Fig.1. Methodology Diagram

The proposed methodology follows six major stages:

1. **Collect CI/CD Logs:** Obtain logs through manual submission or available GitHub Actions workflow integration.
2. **Clean and Organise Logs:** Remove unnecessary information and extract relevant error messages.
3. **Failure Identification and Classification:** Apply pattern matching and the TF-IDF with Logistic Regression classification approach.
4. **Generate Troubleshooting Suggestions:** Map the identified failure category to probable causes and general corrective steps.
5. **Display Diagnosis Results:** Present the failure type, probable cause, and recommendations through the dashboard.
6. **Test Using Sample Logs:** Verify the system using different failure-log examples and functional test cases.

4. IMPLEMENTATION

The system is implemented using a lightweight software stack that combines web application development, text processing, Machine Learning, API integration, and database management.

4.1 Technologies Used

Technology	Purpose
Python	Main programming language
Flask	Backend web framework and application routing
HTML and CSS	User interface and page styling
scikit-learn	Text vectorisation and classification
TF-IDF	Numerical representation of log text
Logistic Regression	Failure classification fallback
Regular Expressions	Identification of predefined error patterns and basic masking
GitHub REST API	Retrieval of workflow information and logs
SQLite	Local storage of diagnosis history

4.2 Log Collection

The system provides two ways to obtain CI/CD logs. The first method allows users to paste or submit logs manually through the web interface. This method is useful when the user already has a log file or copied error output.

The second method uses GitHub Actions integration. When the required repository access and permissions are configured, the application retrieves workflow information and available logs using the GitHub REST API [2]. The retrieved log is then forwarded to the preprocessing module.

4.3 Log Preprocessing

Raw CI/CD logs may contain timestamps, repeated status messages, unnecessary whitespace, and other information that is not directly useful for identifying a failure. The preprocessing module prepares the input for further analysis.

The main operations include:

- Removing unnecessary whitespace and control characters.
- Normalising textual information.
- Filtering irrelevant or repeated messages where applicable.
- Identifying important error messages and failure-related keywords.
- Applying pattern-based masking to common sensitive values.

The purpose of preprocessing is to reduce unnecessary information and retain the relevant parts of the log for classification.

4.4 Feature Extraction Using TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) is used to convert textual log information into numerical feature vectors. It considers how frequently a word appears in a log and how important that word is across the available collection of logs.

The TF-IDF representation can be expressed as:

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)$$

Where:

- t represents a term or word.
- d represents a document or log.
- TF represents the frequency of the term in the document.
- IDF represents the importance of the term across the document collection.

The resulting feature vectors are provided to the classification module. The TF-IDF vectoriser is available through the scikit-learn library [3].

4.5 Failure Classification Using Logistic Regression

Logistic Regression is used as the fallback classification algorithm when predefined error-pattern rules do not identify a strong match. The classifier learns relationships between textual features and their corresponding failure categories from labelled training examples. During diagnosis, the processed log is transformed using the fitted TF-IDF vectoriser and passed to the classifier. The classifier returns a predicted category based on the learned patterns. This category is then mapped to a readable failure label and troubleshooting information. Logistic Regression is suitable for a lightweight text-classification implementation because it can work with numerical feature vectors and is supported by scikit-learn [4].

4.6 Rule-Based Error Identification

Before applying the fallback classifier, the system checks the processed log against predefined regular-expression patterns. These patterns are designed to identify recognisable error messages such as missing dependencies, compilation errors, failed tests, permission-denied messages, network errors, authentication failures, and timeout messages. When a matching pattern is identified, the corresponding failure category and general troubleshooting guidance are selected. This hybrid approach combines direct pattern identification with statistical text classification.

4.7 Diagnosis and Recommendation Display

After classification, the system maps the predicted category to a probable explanation and general troubleshooting steps. The result is displayed through the web interface along with the relevant diagnosis information.

The original log remains available to the developer so that the suggested cause can be checked against the actual error message. The application provides decision-support information and does not automatically modify source code or deployment configurations.

4.8 Diagnosis History

The system uses SQLite to store previous diagnosis records locally. The history module allows users to review earlier analyses and examine recurring failure categories. Each stored record can include the diagnosis date, failure category, and related diagnostic information, depending on the available application fields.

5. SYSTEM TESTING AND RESULTS

Functional testing is performed to verify the main features of the Intelligent CI/CD Failure Diagnosis System. The test cases cover dashboard loading, log submission, classification, GitHub Actions integration, and diagnosis-history storage.

5.1 Results and Discussion

The developed system provides a web interface for submitting CI/CD logs, examining failure categories, and reviewing troubleshooting guidance. It also supports the retrieval of available workflow logs through GitHub Actions integration when the required permissions are configured.

The diagnosis engine combines predefined error-pattern matching with a TF-IDF and Logistic Regression fallback. Recognisable error messages can be identified through predefined patterns, while the classifier provides an additional method of categorising logs that do not match a strong predefined rule. The dashboard presents the diagnosis information in an organised manner. The SQLite database maintains previous diagnosis records, allowing users to review earlier analyses. Functional testing documented in the project report verifies the main application features. However, a separate labelled dataset and controlled classification evaluation are required before making quantitative claims about accuracy, precision, recall, F1-score, or response-time improvement.

5.2. Limitations

The system depends on the quality and coverage of predefined error patterns and the labelled examples used to train the classifier. Unfamiliar error messages or logs containing multiple interacting failures may result in incomplete or incorrect classifications. The current GitHub Actions integration also depends on repository permissions and API availability. Further evaluation using a larger and more diverse collection of CI/CD logs is necessary to establish classification performance.

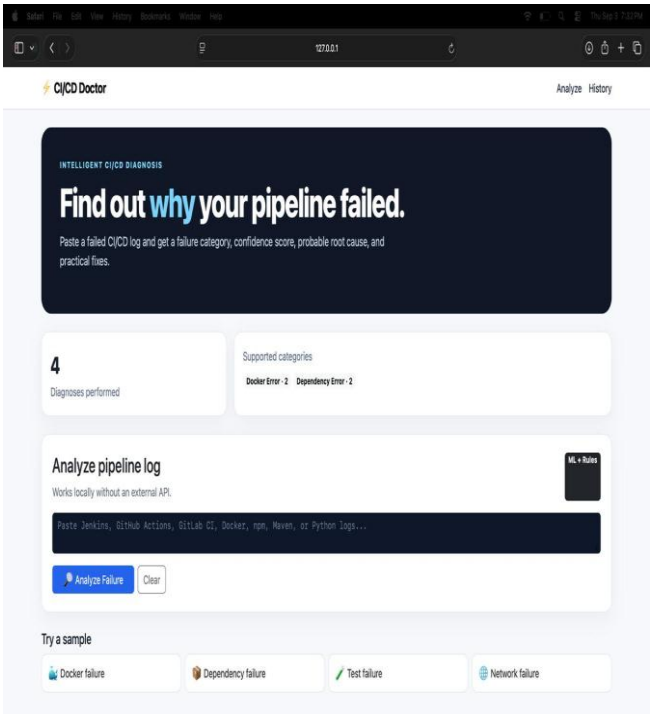

Fig.2. Main dashboard of the Intelligent CI/CD Failure Diagnosis System.

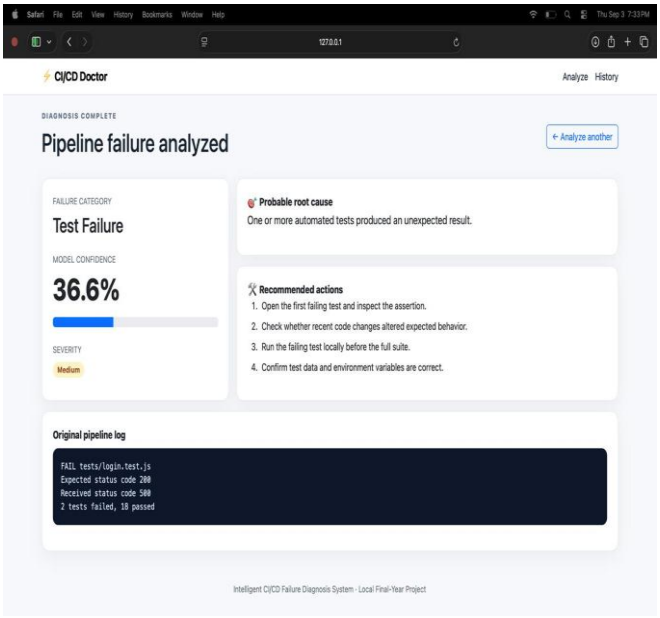

Fig.3. Example log submission and diagnosis result.

6. CONCLUSION AND FUTURE SCOPE

This paper presents an Intelligent CI/CD Failure Diagnosis System designed to assist developers in analysing pipeline failures. The system combines log collection, preprocessing, rule-based error identification, TF-IDF feature extraction, and Logistic Regression classification within a web-based application.

The implementation supports manual log submission and retrieval of available GitHub Actions workflow logs. It identifies common failure categories and displays probable causes with general troubleshooting recommendations. The dashboard provides an organised view of diagnosis results, while SQLite maintains the history of previous analyses.

Functional testing verifies the main application features, including log submission, failure classification, workflow-log retrieval, and history storage. The proposed system provides a lightweight approach to organising CI/CD failure information and reducing dependence on manual log inspection. However, the quality of its diagnosis depends on the available error patterns and training examples, and the recommendations must be verified against the original logs. Future improvements may include expanding support to additional CI/CD platforms such as Jenkins and GitLab CI/CD, developing a larger labelled dataset, improving classification for unseen failure patterns, supporting multiple failure categories in a single pipeline, enhancing sensitive-data masking, and evaluating the system using accuracy, precision, recall, F1-score, and response time. More detailed evidence presentation and developer feedback can also be incorporated to improve the usefulness of the diagnosis process.

REFERENCES

[1] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), 2017, pp. 1285–1298, doi: 10.1145/3133956.3134015.

[2] GitHub, "REST API Endpoints for Workflow Runs," GitHub Documentation. Available: https://docs.github.com/en/rest/actions/workflow-runs

[3] Scikit-learn Developers, "TfidfVectorizer," Scikit-learn Documentation. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[4] Scikit-learn Developers, "LogisticRegression," Scikit-learn Documentation. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html