# Proposed Idea for Software Group Project

Anomaly Detection in System Logs using Machine Learning

## 1. Definition

Anomaly detection in system logs refers to the process of identifying unexpected or abnormal patterns within log data generated by software applications, servers, or distributed systems. These anomalies may indicate system failures, potential cyber threats, misconfigurations, or performance bottlenecks. The proposed system parses unstructured log entries, extracts structured data, and applies statistical and machine learning techniques to automatically detect such anomalies without requiring human intervention.

## 2. Project Overview

This project aims to build a **log analysis and anomaly detection engine** that works entirely from the backend. It will involve developing a pipeline to ingest unstructured logs, convert them into structured format, and then apply both statistical and unsupervised machine learning models to flag unusual behavior. The solution will run through a command-line interface (CLI), eliminating the need for frontend development while focusing deeply on backend and ML components.

The pipeline consists of:

- **Log Parser**: Reads raw logs and extracts structured features (timestamp, log level, component, etc.).

- **Feature Extractor**: Aggregates logs into time or count-based windows and computes numerical features.

- **Anomaly Detection Engine**: Applies statistical thresholds and ML models (like Isolation Forest) to detect anomalous behavior.

- **Report Generator**: Outputs a detailed report highlighting anomalous windows and key contributing features.

### 3. Problem Statement

Modern software systems generate vast amounts of logs during their operation. Manually reviewing these logs to detect errors, failures, or malicious activity is extremely inefficient and prone to human error. The core problems this project addresses include:

- Difficulty in identifying hidden or complex anomalies in massive log files.
- Lack of automation in analyzing patterns that deviate from normal system behavior.
- Time-consuming manual diagnosis during outages or system malfunctions.
- Noisy log data which hides valuable signals within millions of entries.

**This project automates the log analysis process** using machine learning, allowing system administrators and developers to detect issues earlier, faster, and more reliably.

| Component | Tools/Technologies Used |
|---|---|
| Programming Language | Python |
| Data Parsing & Analysis | `pandas`, `re` (for regex-based parsing), `datetime` |
| ML Library | `scikit-learn` (for Isolation Forest, preprocessing) |
| Statistical Analysis | `numpy`, `scipy` |
| Log Dataset | Public datasets like **HDFS Logs** or **BGL Logs** |

| | |
|---|---|
| CLI Interface | Native Python CLI (`argparse`) or shell-based usage |
| Optional Storage | CSV files for intermediate parsed/feature data and output reports |
| Model Persistence | `joblib` or `pickle` (to save/load trained models) |
| Version Control | Git (for code management and reproducibility) |
| Deployment Container | Docker (optional, for standardizing project environment) |

**5. Workflow Breakdown**

1. **Log Collection**
   → Use sample datasets like HDFS or BGL log files.

2. **Log Parsing**
   → Extract timestamp, component name, log level, and message using custom regex-based scripts.

3. **Feature Engineering**
   → Create structured windows of logs and compute features such as error frequency, component activity, average time gaps.

4. **Anomaly Detection**
   → Apply:

   - Statistical thresholds (e.g., outliers based on frequency, z-score).
   - Isolation Forest ML model to learn and flag anomalous patterns.

5. **Result Output**
   → Flag anomalous windows with explanation. Store output in CSV or present via CLI.

**6. End Result (Final Outcome)**

Upon successful execution of the project for Semester 5, the system will be able to:

- Automatically parse raw log files into a structured format.
- Convert logs into feature vectors and analyze them.
- Train an unsupervised machine learning model (e.g., Isolation Forest) to detect patterns of anomalies.
- Flag specific time windows or log sequences as anomalous, backed by numerical scores.
- Output a structured CSV report containing:
    - Timestamp range of anomalous window
    - Severity score
    - Components involved
    - Description of what made that window anomalous (e.g., spike in errors, unseen component behavior).