Problem Statement:

Develop an AI-based predictive maintenance system that analyzes system logs (CPU usage, memory, disk I/O) to predict potential failures. The model should provide alerts before failures occur.

Prediction accuracy (AUC-ROC, F1-score) Timeliness of failure prediction Visualization clarity (graphs of system health trends) Usability of generated alerts

Expected Output: Predictive model trained on log data Report showing predicted system failures Visualization dashboard (optional)

**Overall Approach**

1. **Data Collection**: Gather system logs containing CPU usage, memory consumption, disk I/O, and other relevant metrics.
2. **Data Preprocessing**: Clean the data, handle missing values, and normalize the features.
3. **Feature Engineering**: Extract meaningful features from logs (e.g., rolling averages, spikes, trend patterns).
4. **Model Selection**: Choose a machine learning model (e.g., Random Forest, XGBoost, LSTM for time series) to train on historical log data.
5. **Model Training & Evaluation**: Train the model, measure prediction accuracy using **AUC-ROC** and **F1-score**.
6. **Failure Prediction**: Generate alerts when the model detects potential failures before they occur.
7. **Visualization**: Build a dashboard to show system trends and predictions.
8. **Deployment**: Integrate the trained model into a live monitoring system.

**Data Collection (Windows Machine)**

**A screen shot of a computer screen

AI-generated content may be incorrect.**

**Get-EventLog -LogName System | Export-Csv -Path "C:\AI\_Project\logs\system\_logs.csv" -NoTypeInformation**

**1. Data Pipeline**

Start by defining how you'll collect and structure the log data:

* **Data Sources**: System logs (Windows Event Viewer, perfmon exports, syslog for Linux, etc.)
* **Preprocessing**: Clean and resample logs into a uniform time base.
* **Feature Engineering**: Include rolling stats (mean, std), rate of change, lagged values, error/event codes, disk queue length, etc. You’ve already done great work on this, so modularizing for reuse will pay off.

**2. Model Development**

* **Labels**: Identify failure points (shutdowns, errors) and create lead-time windows to mark data as “failure approaching.”
* **Model Choice**: XGBoost or LightGBM are strong for tabular time series, or you could explore LSTM/GRU if modeling sequences.
* **Evaluation**: AUC-ROC and F1-score for class balance, but also prioritize *lead time before failure* as a practical metric.

**3. Alerts & Usability**

* **Rules-Based Layer**: Overlay rules to avoid alert fatigue (e.g. suppress duplicates, throttle frequency).
* **Severity Grading**: Let models assign confidence scores and categorize alerts (Low/Medium/High).
* **Alert Output**: JSON for API use or write to Windows Event Log, email, or Slack.

**4. Visualization Dashboard (Optional)**

* Tools: Plotly Dash, Streamlit, or Power BI.
* Views:
  + **System Metrics Over Time** with shaded failure zones.
  + **Upcoming Predicted Failures** with lead time clock.
  + **Feature Importance** heatmap.

**5. Reporting**

* Weekly or monthly reports with:
  + Predicted vs. actual failure occurrences
  + False positives/negatives
  + Model health over time