# End-to-End Solution for Software System Event Log Analysis

Events go in → Process insights come out

## Input:

Raw system logs generated by diverse software components, such as web servers, databases, and middleware systems. These logs may include:  
• Timestamped events  
• Log levels (INFO, WARN, ERROR)  
• Component or service identifiers  
• User/session information (if available)  
• Event messages or status codes

## Processing (Black Box):

Standard log analysis workflows usually involve manual filtering or basic keyword searches. However, such methods fail to handle heterogeneous log formats and lack structure for process analysis.

Typical limitations:  
• High noise due to irrelevant entries  
• Lack of consistent identifiers (case/session IDs)  
• Incomplete correlation between related events

## Output:

Basic log statistics (e.g., error counts, request frequencies) with limited interpretability.

## Actionable Insights (Current Limitations):

For Developers: Manual and time-consuming debugging process.  
For System Engineers: Difficult to detect bottlenecks or deviations in distributed systems.  
For Researchers: Lack of standardized event structures reduces reproducibility.

# AI-Enhanced End-to-End Solution

Input Layer → AI/Mining Processing Layer → Output Layer → Actionable Insights

## 1. Log Preprocessing and Normalization

Input:  
Raw heterogeneous logs from distributed components (e.g., Apache, Linux syslog, custom application logs).  
  
AI Processing:  
• Parse and filter relevant entries.  
• Normalize timestamps to ISO 8601.  
• Map component names and error types to unified categories.  
• Remove debug-level noise and redundant lines.  
  
Output:  
Structured log entries suitable for event correlation.

## 2. Smart Event Correlation and Trace Construction

Input:  
Preprocessed logs with standardized timestamps and component labels.  
  
AI Processing:  
• Detect correlation attributes (session ID, request ID, user ID).  
• When missing, use clustering and time-window heuristics to group related events.  
• Apply semantic similarity and temporal proximity for trace construction.  
  
Output:  
Traces representing individual process executions (e.g., user sessions, job executions).

## 3. Process Discovery and Model Generation

Input:  
Structured event logs with case ID, activity, and timestamp attributes.  
  
AI Processing:  
• Apply process discovery algorithms (Inductive Miner, Heuristic Miner).  
• Visualize event flow as BPMN or Petri net model.  
• Evaluate models using metrics like fitness, precision, and generalization.  
  
Output:  
Process model showing real execution flows and deviations from expected behavior.

## 4. Anomaly Detection and Conformance Checking

Input:  
Discovered process model and event logs.  
  
AI Processing:  
• Replay logs against the process model to identify deviations.  
• Detect anomalies, latency spikes, or missing transitions.  
• Use metrics like throughput time and performance heatmaps for visualization.  
  
Output:  
Alerts such as “Unexpected sequence detected in microservice A” or “High latency in component B”.

## 5. Performance Evaluation and Optimization

Input:  
Process models and performance logs.  
  
AI Processing:  
• Calculate average response times and throughput for each process path.  
• Identify bottlenecks or delays across services.  
• Recommend optimization strategies (e.g., load balancing, retry mechanisms).  
  
Output:  
Performance dashboards highlighting system efficiency and reliability trends.

## Final Output Layer

Process Models (BPMN / Petri Nets)  
Anomaly Detection Reports  
Performance Dashboards  
Trace Quality Summaries

## Actionable Insights:

For Developers: Automatic identification of inefficient or failing process segments.  
For System Engineers: Real-time performance visualization and anomaly alerts.  
For Researchers: Reproducible and scalable framework for event log analysis.