International Journal of Science and Research Archive
eISSN: 2582-8185
Cross Ref DOI: 10.30574/ijsra
Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

# Integrating Jenkins with AIOps Platforms for Hybrid SAP Basis Automation: A reference architecture

Anand Singh *

*Sr. Manager / Architect – SAP Basis, Abbvie Inc. USA.*

International Journal of Science and Research Archive, 2026, 19(01), 1128-1140

Publication history: Received on 18 March 2026; revised on 24 April 2026; accepted on 27 April 2026

Article DOI: https://doi.org/10.30574/ijsra.2026.19.1.0898

## Abstract

The increasing complexity of SAP enterprise landscapes — spanning on-premises NetWeaver stacks, cloud-deployed SAP S/4HANA systems, and Business Technology Platform (BTP) environments — has created an urgent operational challenge for SAP Basis administrators. Managing runtime services such as Apache Tomcat embedded within SAP's Java Application Server requires constant vigilance against memory exhaustion, thread pool saturation, and classloader failures. Conventional operational approaches rely on either manual monitoring or rigid, rule-based automation pipelines that lack contextual intelligence. This paper proposes and rigorously evaluates a novel hybrid reference architecture that combines Jenkins CI/CD pipeline automation with AIOps (Artificial Intelligence for IT Operations) platforms to create an intelligent, self-healing SAP Basis operational framework. The architecture introduces a five-layer model spanning SAP runtime telemetry ingestion, observability normalization, AIOps-driven anomaly detection, Jenkins orchestration, and autonomous remediation execution. Using Apache Tomcat log monitoring and automated restart orchestration as the primary demonstration use case, the proposed system is evaluated against Jenkins-only and AIOps-only baselines across key operational metrics. Experimental results demonstrate that the hybrid architecture reduces Mean Time to Detect (MTTD) by 89%, reduces Mean Time to Restore (MTTR) by 77%, and reduces the false-positive remediation rate by 77% compared to Jenkins-only automation. The architecture achieves 94% automation coverage while maintaining full ITIL compliance through Jenkins' native audit trail capability. This research contributes a reproducible integration pattern, a confidence-threshold calibration model for multi-environment SAP deployments, and a Tomcat error taxonomy that bridges IT operations research with SAP Basis engineering practice.

**Keywords:** SAP Basis Administration, AIOps; Jenkins Automation; Apache Tomcat Monitoring; Hybrid IT Operations; Self-Healing Systems; JVM Anomaly Detection; Natural Language Processing; Root Cause Analysis; ITIL Compliance; DevOps; MLOps; SAP NetWeaver Java Stack

## 1. Introduction

Enterprise resource planning (ERP) systems built on SAP technology underpin mission-critical business processes for thousands of organizations globally, spanning manufacturing, financial services, healthcare, and public-sector administration. The operational continuity of these environments depends substantially on the expertise and responsiveness of SAP Basis administrators — the specialized infrastructure engineers responsible for system configuration, performance tuning, transport management, and runtime service management. Among the most complex responsibilities within the SAP Basis domain is the management of the Java Application Server (AS Java), which hosts the SAP NetWeaver Java stack and relies on Apache Tomcat as its embedded servlet container. Tomcat's stability is central to the availability of critical SAP services including the SAP Enterprise Portal, SAP Process Integration (PI), and Web Dynpro applications.

* Corresponding author: Anand Singh

The operational challenges associated with Tomcat management in SAP environments are significant and multi-dimensional. At the runtime level, the Java Virtual Machine (JVM) that hosts Tomcat is subject to memory pressure from heap exhaustion, Metaspace (or PermGen in older JVMs) overflow, garbage collection (GC) storms, and thread pool starvation — all of which can cause Tomcat to become unresponsive or crash entirely. At the monitoring level, the catalina.out log — Tomcat's primary diagnostic file — is a high-volume, unstructured text stream that requires semantic interpretation to distinguish actionable critical errors from routine informational entries. At the operational level, the mean time between a Tomcat failure and human-initiated remediation has historically ranged from 15 to 45 minutes in organizations relying on manual monitoring, resulting in measurable service-level agreement (SLA) violations and downstream business impact.

Two automation paradigms have emerged to address these challenges. The first, pipeline-driven automation using tools such as Jenkins, provides deterministic execution of pre-authored remediation playbooks triggered by threshold-based monitoring alerts or scheduled health checks. Jenkins pipelines offer strong governance characteristics — reproducible builds, parameterized execution, native integration with ticketing systems, and comprehensive audit trails — but are fundamentally reactive. They cannot identify novel failure patterns not encoded in pre-existing rules, and they lack the ability to distinguish between superficially similar log entries that require different responses.

The second paradigm, AIOps (Artificial Intelligence for IT Operations), applies machine learning, statistical anomaly detection, and event correlation algorithms to telemetry streams, enabling proactive identification of emerging failure conditions before they escalate to outages. AIOps platforms excel at pattern recognition across heterogeneous, high-volume data but are operationally incomplete without a reliable execution mechanism for translating detected anomalies into remediation actions. When AIOps platforms act autonomously, they often bypass enterprise change management processes, creating compliance risks in regulated industries.

This paper resolves the fundamental tension between these two paradigms by proposing a hybrid reference architecture in which AIOps provides cognitive detection and Jenkins provides governed execution. The architecture uses a well-defined API integration contract to pass anomaly intelligence from the AIOps layer to the Jenkins orchestration layer, enabling autonomous remediation that is simultaneously intelligent, auditable, and ITIL-compliant. The primary use case evaluated throughout this paper is SAP Tomcat log monitoring and automated restart orchestration — a scenario chosen for its high operational frequency, measurable impact on SAP system availability, and the availability of rich, structured telemetry against which detection models can be evaluated.

**The specific contributions of this paper are:**

- A five-layer hybrid reference architecture that integrates AIOps with Jenkins for SAP Basis automation.
- A Tomcat critical error taxonomy that provides a reproducible classification framework for log-based incident detection.
- A confidence-threshold calibration model that enables environment-aware (development, QA, production) governance of autonomous actions.
- A comparative experimental evaluation demonstrating quantitative superiority of the hybrid approach over Jenkins-only and AIOps-only baselines.
- A discussion of integration patterns, limitations, and future research directions including reinforcement learning for dynamic threshold optimization.

## 2. Background and Related Work

### 2.1. AIOps: Foundations and Enterprise Adoption

The term AIOps was introduced by Gartner in 2017 to describe the application of artificial intelligence and machine learning to enhance and partially replace core IT operations processes including availability and performance monitoring, event correlation, root cause analysis, and service desk automation. The functional architecture of AIOps platforms rests on three foundational capabilities: big data ingestion from heterogeneous IT telemetry sources (logs, metrics, events, traces, topology), algorithm-driven analysis encompassing anomaly detection, pattern recognition, and causal inference, and action orchestration that translates analytical outputs into operational interventions.

Research in AIOps has evolved substantially since the early work on log analysis by Fu and colleagues, who demonstrated that structured feature extraction from unstructured log text could enable effective runtime anomaly detection in distributed systems. Subsequent work by Du and colleagues introduced DeepLog, a deep learning model

trained on system log sequences that learns normal execution patterns and identifies deviations with high precision. More recent contributions have applied transformer-based architectures including BERT and its domain-specific variants to log classification tasks, achieving state-of-the-art performance on benchmark datasets derived from production system logs.

In the SAP domain specifically, the application of AIOps techniques remains nascent relative to general-purpose IT operations research. Existing SAP monitoring solutions — including SAP Solution Manager's Computer Center Management System (CCMS) and the newer SAP Cloud ALM platform — provide alert-based monitoring with threshold-driven notifications but do not incorporate machine learning for adaptive anomaly detection. This gap represents a significant research opportunity, as SAP environments generate rich, structured telemetry from the ABAP runtime, ICM, and Java stack layers that is well-suited to machine learning analysis.

### 2.2. Jenkins in IT Operations and SAP Automation

Jenkins, originally developed as Hudson at Sun Microsystems and later forked and renamed, has evolved from a continuous integration server into a general-purpose automation orchestration platform. Its Pipeline-as-Code model, introduced with Jenkins 2.0, enables infrastructure and operations teams to encode complex workflows as version-controlled Groovy scripts, providing the same software engineering discipline to operational automation that CI/CD brought to software delivery. In the SAP Basis domain, Jenkins has seen adoption for system refresh automation, transport management, backup orchestration, and health check scheduling.

Prior work on Jenkins in IT operations has focused primarily on its role in continuous delivery pipelines rather than operational remediation. However, a growing body of practitioner literature documents Jenkins' use as an orchestration engine for infrastructure automation, particularly in combination with Ansible for configuration management and execution. The Jenkins REST API, introduced in version 1.5xx, enables programmatic triggering of parameterized builds from external systems, making Jenkins a suitable execution backend for AIOps platforms that need a governed, auditable action mechanism.

### 2.3. Log-Based Anomaly Detection Methods

Log-based anomaly detection has been an active research area since the early 2000s, with approaches ranging from rule-based pattern matching to sophisticated deep learning models. Principal research contributions include the Drain log parsing algorithm, which constructs parse trees to efficiently extract structured templates from unstructured log messages; the LogCluster algorithm, which groups similar log events using hashing techniques; and sequence-based models that treat log streams as time-ordered sequences and apply recurrent neural networks (RNNs) or transformers to model normal execution patterns.

For JVM-specific log analysis, the challenges include the high volume of GC log entries, the semantic ambiguity of Tomcat WARNING versus SEVERE log levels, and the need to correlate individual log entries with concurrent JVM metric readings (heap utilization, GC pause duration, thread count) to accurately characterize failure conditions. This multi-modal analysis requirement — combining log text with numeric time-series data — distinguishes JVM anomaly detection from simpler log classification tasks and motivates the multi-component architecture proposed in this paper.

### 2.4. Hybrid Automation: The Research Gap

Despite the complementary strengths of AIOps and pipeline automation, no prior scholarly work has proposed a formal integration architecture that defines the interface between these two paradigms for SAP Basis operations. Practitioner blog posts and vendor whitepapers have described ad-hoc integrations, but without the formal specification of the API contract, confidence scoring model, or governance framework that enterprise adoption requires. This paper addresses that gap by providing a reproducible, formally specified reference architecture grounded in enterprise requirements analysis and validated through controlled experimentation.

## 3. Problem Statement and Research Objectives

The operational problem addressed in this paper is formally defined as follows: Given a production SAP NetWeaver Java landscape in which Apache Tomcat serves as the embedded servlet container, design an automation architecture that: (a) continuously ingests and semantically interprets Tomcat runtime telemetry without manual intervention, (b) accurately distinguishes critical error conditions requiring immediate restart from transient anomalies that self-resolve, (c) executes remediation actions with sufficient speed to minimize end-user impact, (d) maintains a complete, ITIL-compliant audit trail of all automated actions, and (e) continuously improves detection accuracy through closed-loop feedback from remediation outcomes.

This problem is non-trivial because the four requirements (a)-(d) are partially in tension. Requirement (a) favors continuous streaming analysis; requirement (b) favors deliberate, multi-source corroboration before action; requirement (c) favors speed that can conflict with deliberation; and requirement (d) favors routing actions through governed change management processes that introduce latency. The fifth requirement (e) introduces additional complexity by requiring the detection model to update based on operational experience, which demands a feedback loop that most operational automation architectures do not implement.

The specific research objectives pursued in this paper are: RO1 — to design a five-layer reference architecture that structurally separates the concerns of telemetry ingestion, anomaly detection, decision governance, execution orchestration, and feedback; RO2 — to specify the API integration contract between AIOps and Jenkins with sufficient precision for enterprise implementation; RO3 — to develop a Tomcat error taxonomy that enables consistent severity classification across SAP landscapes; RO4 — to define a confidence threshold calibration model that enforces environment-appropriate governance; and RO5 — to empirically evaluate the hybrid architecture against established baselines using operationally relevant performance metrics.

## 4. Theoretical Framework

### 4.1. The OODA Loop as an Architectural Metaphor

The hybrid architecture proposed in this paper is grounded in the Observe-Orient-Decide-Act (OODA) loop, originally formulated by military strategist John Boyd as a model for competitive decision-making under uncertainty. The OODA loop has been adopted in IT operations research as a conceptual framework for autonomous systems that must sense environmental conditions, interpret those conditions in context, make decisions about appropriate responses, and execute actions — then observe the results of those actions as new inputs to the cycle. Each layer of the proposed five-layer architecture corresponds to a phase of the OODA loop: Layers 1 and 2 fulfill the Observe function, Layer 3 fulfills the Orient function, the confidence scoring sub-component of Layer 3 fulfills the Decide function, and Layers 4 and 5 fulfill the Act function.

The closed-loop feedback mechanism that returns remediation outcomes to the AIOps model corresponds to the loop's iterative nature: each action produces new observational data that refines the system's orientation in subsequent cycles. This theoretical grounding ensures that the architecture is not merely an ad-hoc collection of technology integrations but reflects a principled model of autonomous operational intelligence.

### 4.2. The Confidence-Action Spectrum

A key theoretical contribution of this paper is the formalization of the confidence-action spectrum, which maps AIOps anomaly confidence scores to operational responses along a continuum from passive observation to fully autonomous remediation. The spectrum is defined over the interval [0, 1], where 0 represents complete uncertainty and 1 represents absolute certainty of a critical failure condition. Four response zones are defined within this interval: the Monitor Zone (score < 0.50), in which detected anomalies are logged and trend-tracked but no alert is generated; the Alert Zone (0.50 ≤ score < 0.70), in which on-call personnel receive a notification with diagnostic context but no automated action is taken; the Supervised Automation Zone (0.70 ≤ score < 0.90), in which automated action is initiated and simultaneously notified to on-call personnel for post-action review; and the Autonomous Action Zone (score ≥ 0.90), in which automated action is executed without advance human notification, relying on post-action audit trail for compliance.

Zone boundaries are parameterized rather than fixed, enabling organizations to shift the confidence-action mapping based on environmental risk tolerance, regulatory requirements, and the operational maturity of their AIOps model. A newly deployed model with limited training data should apply conservative zone boundaries; a mature model with

extensive production experience and high precision metrics can safely apply permissive boundaries. The calibration model for determining appropriate zone boundaries based on model performance metrics is defined in Section 6.3.

### 4.3. Separation of Cognitive and Executive Concerns

The theoretical foundation for the hybrid architecture's division of responsibility between AIOps and Jenkins rests on the principle of separation of concerns, applied at the level of operational system architecture. Cognitive concerns — understanding the meaning of observed telemetry, recognizing patterns, inferring causal relationships, and estimating the probability of critical failure — are the natural domain of machine learning systems trained on large telemetry datasets. Executive concerns — reliable, idempotent, rollback-safe execution of operational procedures with complete audit trails — are the natural domain of pipeline orchestration systems with mature execution guarantees.

Conflating these concerns, as both pure AIOps and pure Jenkins approaches implicitly do, produces systems that are either cognitively limited (Jenkins-only) or executionally unreliable (AIOps-only without a governed execution layer). The hybrid architecture maintains a clean boundary between these domains, mediated by a formally specified API contract that carries sufficient context for Jenkins to execute the correct remediation procedure without needing to understand the AIOps detection rationale.

## 5. Reference Architecture Design

The reference architecture comprises five hierarchical layers, each with precisely defined responsibilities, inter-layer communication protocols, and technology component options. The layers are designed to be technology-agnostic at their boundaries, enabling organizations to substitute specific products within each layer without altering the overall architectural pattern.

### 5.1. Layer 1: SAP Runtime Telemetry

The foundation of the architecture is the SAP runtime environment, which generates four categories of operational telemetry. Tomcat log data, originating from the catalina.out log file, constitutes the primary signal source for the use case evaluated in this paper. This log file receives output from the Tomcat container including application server startup and shutdown events, deployed web application lifecycle messages, HTTP request processing errors, and JVM-level exception stack traces. In production SAP NetWeaver environments with multiple Java instances, catalina.out files are generated per-instance, requiring collection infrastructure capable of aggregating multi-source log streams.

JVM metrics constitute the second telemetry category. The Java Management Extensions (JMX) framework exposes real-time JVM operational statistics including heap memory utilization across Young and Old generation spaces, GC collection frequency and pause duration, active thread count and thread pool utilization, classloader statistics, and CPU utilization attributable to JVM processes. These metrics provide the quantitative context that transforms qualitative log error messages into statistically characterizable failure precursors. ICM and work process logs comprise the third telemetry category, capturing SAP-specific infrastructure events at the dispatcher and work process level. The fourth category, HANA DB events, captures connection pool saturation, statement timeout, and memory allocation failures that frequently contribute to Java stack failures through cascade effects.

### 5.2. Layer 2: Observability Pipeline

Layer 2 normalizes the diverse telemetry formats emitted by Layer 1 into standardized representations consumable by the Layer 3 AIOps engine. Log data normalization is performed by Fluentd or Logstash agents deployed on each SAP application server host, which tail the catalina.out and ICM log files, apply parsing rules to extract structured fields (timestamp, log level, component, message body), and forward parsed events to a centralized log aggregation platform such as Elasticsearch or Splunk. The Fluentd deployment model is preferred in Kubernetes-based SAP environments where sidecar container patterns enable log collection without modification to the SAP application configuration.

JVM metrics are collected by Prometheus using a JMX exporter configured as a Java agent within the SAP Java startup parameters, exposing metrics on a configurable HTTP endpoint for Prometheus scraping. Distributed trace data, where available through OpenTelemetry instrumentation of custom SAP Java applications, is buffered in Apache Kafka topics to provide the Layer 3 engine with asynchronous access to trace data without introducing Kafka latency into the log and metrics collection paths. The observability pipeline must be designed for high availability with no single points of failure, as its unavailability directly impairs the AIOps engine's detection capability.

### 5.3. Layer 3: AIOps Intelligence Engine

The AIOps intelligence engine is the cognitive core of the architecture, comprising four specialized analytical components that operate concurrently on the telemetry streams delivered by Layer 2. The anomaly detection component applies a Long Short-Term Memory (LSTM) neural network to the JVM metric time series, learning the normal temporal patterns of heap utilization, GC behavior, and thread counts across different periods of the business day and week. The LSTM architecture is chosen for its demonstrated effectiveness on multivariate time-series anomaly detection tasks and its ability to capture long-range temporal dependencies that are characteristic of gradually developing JVM memory leaks.

The NLP log classification component applies a fine-tuned BERT-based model to the parsed message bodies from the Tomcat log stream, classifying each log entry into one of five severity tiers: Critical (immediate restart warranted), High (thread dump capture and investigation warranted), Medium (monitoring escalation warranted), Low (record and trend-track), and Informational (discard). The model is fine-tuned on a labeled dataset of SAP Tomcat log entries curated from production incident post-mortems, supplemented with synthetic examples generated by adversarial data augmentation to improve robustness to novel error message formats introduced by SAP patches and custom Java developments.

The event correlation component constructs a causal dependency graph from the topology information provided by the CMDB integration and uses graph-based propagation algorithms to identify root causes when multiple concurrent anomalies are detected across different components. This capability is critical for avoiding incorrect remediation actions when a Tomcat memory error is actually caused by a HANA DB connection pool exhaustion event rather than a JVM configuration issue. Without causal correlation, a Jenkins-triggered Tomcat restart in such a scenario would be ineffective, as the root cause resides in the HANA DB layer. The confidence scoring component aggregates the outputs of the three analytical components using a weighted ensemble model to produce a single normalized confidence score in the interval [0, 1] that characterizes the probability that an autonomous Tomcat restart is the appropriate remediation action.

### 5.4. Layer 4: Jenkins Orchestration Plane

When the confidence scoring component determines that the confidence score exceeds the configured autonomous action threshold for the target environment, it invokes the Jenkins REST API with a parameterized build request. The Jenkins instance receives the request, validates the parameters against a pre-configured allowlist of authorized SAP system IDs, and selects the appropriate remediation pipeline from a library of Jenkinsfiles maintained in a version-controlled repository. The use of a parameter-driven pipeline selector rather than a single monolithic pipeline enables the architecture to handle multiple remediation scenarios (Tomcat restart, JVM heap analysis, ICM reset, HANA DB connection pool flush) with a single API integration point.

Before executing the remediation action, the Jenkins pipeline performs a pre-flight validation phase: it queries the CMDB to verify that the target SAP system is not in a scheduled maintenance window, confirms that the SAP system's availability monitoring indicates an active failure (rather than a precautionary restart during normal operation), and creates a ServiceNow incident record to establish the audit trail. The ServiceNow integration is implemented through the Jenkins ServiceNow plugin, which enables Jenkins pipeline steps to create, update, and close incident records using the ServiceNow REST API. This integration ensures that every automated action appears in the organization's ITSM system with complete context, satisfying ITIL change management requirements without requiring human intervention in the change creation process.

### 5.5. Layer 5: Remediation Actions and Feedback

The remediation execution layer implements the specific technical actions prescribed by the Jenkins pipeline. For the Tomcat restart use case, execution proceeds in a defined sequence: capture of a JVM thread dump and heap histogram for diagnostic preservation, graceful shutdown of the Tomcat instance using the SAP-specific control script (sapcontrol -nr {instance} -function Stop), a configurable wait period to allow in-flight requests to complete, verification that the Tomcat process has terminated, restart of the Tomcat instance using sapcontrol Start, and a configurable post-restart health verification phase that polls the SAP ICM health endpoint at 30-second intervals. The health verification phase continues for up to five minutes. A failed health check triggers an escalation workflow that pages the on-call SAP Basis administrator with a pre-populated incident summary and the pre-restart diagnostic artifacts (thread dump, heap histogram, log snippet).

The feedback mechanism constitutes the final and architecturally unique element of Layer 5. Upon completion of the remediation action (whether successful, failed, or escalated), the Jenkins pipeline publishes a structured outcome event to the AIOps platform's webhook endpoint. The outcome event includes the anomaly incident ID, the actions taken, the post-action system health state, and a human-reviewable flag indicating whether the automated action was appropriate in retrospect. The AIOps platform incorporates this labeled outcome data into its model retraining pipeline, enabling the detection models to continuously improve their classification accuracy based on production experience.

## 6. Tomcat Monitoring and NLP Classification Model

### 6.1. Tomcat Error Taxonomy

A standardized error taxonomy is a prerequisite for consistent AIOps classification across diverse SAP Tomcat deployments. The taxonomy developed in this paper is derived from analysis of 2,847 Tomcat incident records drawn from 14 SAP production environments spanning manufacturing, retail, and financial services industries. The taxonomy classifies Tomcat error conditions into five tiers based on their operational impact and the required response time and type:

**Table 1** SAP Tomcat Error Taxonomy — Severity Tiers, Response Times, and Prescribed Actions

| Error Pattern | Severity Tier | Req. Response | Prescribed Action |
|---|---|---|---|
| OutOfMemoryError: Java heap space | CRITICAL (Tier 1) | < 2 minutes | Immediate restart + heap histogram |
| GC overhead limit exceeded | CRITICAL (Tier 1) | < 2 minutes | JVM heap flush + Tomcat restart |
| OutOfMemoryError: Metaspace | CRITICAL (Tier 1) | < 5 minutes | Restart + JVM Metaspace tuning |
| StackOverflowError | HIGH (Tier 2) | < 10 minutes | Thread dump + investigation |
| SEVERE: Error listenerStart | HIGH (Tier 2) | < 10 minutes | Application restart |
| Connection refused (HANA DB) | MEDIUM (Tier 3) | < 30 minutes | Escalate to DBA; no Tomcat restart |
| WARNING: SetContextPropertiesRule | LOW (Tier 4) | Next business day | Log and review in next change window |
| INFO: Server startup in [N] ms | INFORMATIONAL (Tier 5) | No action | Discard after telemetry capture |

### 6.2. BERT-Based Log Classification Architecture

The NLP classifier is implemented as a fine-tuned BERT model (base architecture: bert-base-uncased, 110M parameters) with a classification head appended to the [CLS] token representation. The input to the model is a 512-token window centered on the log message body, with special handling for Java exception stack traces that may extend across multiple catalina.out lines. A log grouping pre-processing step uses the Drain algorithm to identify consecutive log lines belonging to the same exception event and concatenates them into a single classification input.

The fine-tuning dataset comprises 18,450 labeled Tomcat log entries distributed across the five severity tiers defined in the taxonomy. Class imbalance — Tier 5 entries constitute approximately 73% of all Tomcat log output in production environments — is addressed through a combination of stratified sampling during training batch construction and a class-weighted cross-entropy loss function that assigns higher loss weight to minority classes. The model is fine-tuned for 10 epochs using the AdamW optimizer with a learning rate of 2e-5 and a linear warmup schedule covering the first 10% of training steps.

### 6.3. Confidence Threshold Calibration

The confidence score produced by the AIOps ensemble model must be calibrated against the operational risk profile of the target SAP environment. The calibration procedure involves: evaluation of model precision and recall at each candidate threshold on a held-out validation set; estimation of the cost ratio between false negatives (missed critical errors that result in extended outages) and false positives (unnecessary Tomcat restarts that briefly interrupt service); and selection of the threshold that minimizes expected operational cost for the environment's specific risk tolerance.

For production SAP environments in regulated industries, the cost ratio is typically 10:1 (a missed critical error is 10 times more costly than an unnecessary restart), which drives threshold selection toward higher recall at the cost of some precision. For development landscapes, the cost ratio approaches 1:1, enabling lower thresholds and more aggressive automation.

## 7. Jenkins Pipeline Specification

### 7.1. API Integration Contract

The integration between the AIOps platform and Jenkins is mediated by a formally specified REST API contract. The AIOps platform acts as the API client, invoking the Jenkins buildWithParameters endpoint with a structured JSON payload that carries sufficient context for Jenkins to execute the correct remediation procedure without querying the AIOps platform for additional information. The API contract specifies the following mandatory parameters:

- SAP_SID: The three-character System ID of the target SAP system (e.g., PRD, QAS, DEV).
- INSTANCE_NUMBER: The two-digit SAP instance number of the affected Java instance.
- ACTION: A controlled vocabulary identifier for the remediation action (TOMCAT_RESTART, JVM_HEAP_ANALYSIS, ICM_RESET).
- SEVERITY: The error taxonomy severity tier detected by the AIOps NLP classifier.
- AIOPS_INCIDENT_ID: A unique identifier for the AIOps incident record, used for cross-system traceability.
- CONFIDENCE_SCORE: The ensemble model confidence score in [0, 1], logged for post-action analysis.
- ROOT_CAUSE: The causal correlation component's determination of the failure's origin layer.
- PRE_ACTION_THREAD_DUMP: Boolean flag indicating whether a thread dump should be captured before action.

### 7.2. Jenkinsfile Pipeline Structure

The Tomcat remediation Jenkinsfile implements a seven-stage pipeline that enforces pre-action validation, diagnostic capture, remediation execution, post-action verification, and feedback publication as discrete, independently auditable stages. Each stage emits structured log output that is archived as a Jenkins build artifact, providing the ITIL-required change record. The stages are: (1) Parameter Validation, which verifies that the incoming SAP_SID is in the authorized system allowlist and that the system is not in a scheduled maintenance window; (2) CMDB Verification, which queries the ServiceNow CMDB API to confirm the SAP system's current operational state; (3) Incident Creation, which creates a ServiceNow incident record with the AIOps diagnostic context; (4) Pre-Action Diagnostics, which optionally captures a JVM thread dump and heap histogram; (5) Remediation Execution, which runs the Ansible playbook for the specified ACTION; (6) Health Verification, which polls the SAP ICM endpoint until the service is confirmed healthy or the timeout elapses; and (7) Feedback Publication, which posts the structured outcome event to the AIOps webhook endpoint and updates the ServiceNow incident to Resolved or In Progress.

### 7.3. Ansible Playbook Design for Tomcat Restart

The Ansible playbook invoked by the Jenkins pipeline implements the Tomcat restart sequence as an ordered set of idempotent tasks that can be safely retried if interrupted. Idempotency is achieved by checking the current state of the Tomcat process before each action: if Tomcat is already stopped when the stop task executes, the task reports success without attempting a second stop. The restart sequence is parameterized to accept the SAP instance number, enabling the same playbook to target any instance in the landscape. Post-restart validation is implemented as a wait_for module task that polls the ICM health URL with an exponential backoff strategy, providing more aggressive retry behavior in the first 60 seconds after restart and progressively longer intervals thereafter.

## 8. Experimental Evaluation

### 8.1. Experimental Setup

The experimental evaluation was conducted on a dedicated SAP NetWeaver 7.5 SP20 landscape deployed on VMware virtualized infrastructure with four Java instances across two application servers. The landscape was instrumented with the full observability stack described in Layer 2 of the reference architecture: Fluentd 1.16 for log collection, Prometheus 2.45 with JMX exporter 0.19.0 for metric collection, and Kafka 3.5 for event buffering. The AIOps intelligence engine was implemented using a combination of PyTorch 2.1 for LSTM model development, the Hugging

Face Transformers library (version 4.35) for BERT-based log classification, and a custom Python event correlation service. Jenkins 2.426 was deployed on a dedicated server with the Ansible, ServiceNow, and Pipeline plugins installed.

Tomcat incident scenarios were synthesized by injecting failure conditions at the JVM level using custom Java agents that simulate heap pressure (by allocating large arrays at controlled rates), thread pool exhaustion (by launching blocking threads that do not release pool slots), and classloader leaks (by repeatedly loading and abandoning classloader instances without explicit garbage collection hints). A total of 120 incident scenarios were generated across four failure categories over a 90-day evaluation window, with scenarios distributed across different times of day and business cycles to ensure representative coverage of JVM telemetry patterns.

### 8.2. Evaluation Metrics

The evaluation uses five primary metrics: Mean Time to Detect (MTTD), measured from the moment a synthetic failure condition is injected to the moment the monitoring system produces an actionable alert; Mean Time to Restore (MTTR), measured from failure injection to confirmed system health restoration; False Positive Rate (FPR), defined as the proportion of automated remediation actions triggered when the SAP system was not experiencing a genuine failure condition; Automation Coverage (AC), defined as the proportion of injected failure scenarios that were resolved without human intervention; and Model Precision and Recall, measured against the labeled test set held out from the NLP classifier training process.

### 8.3. Baseline Configurations

Three system configurations are compared. Configuration A (Jenkins-Only Baseline) implements a threshold-based Jenkins monitoring pipeline that polls JVM metrics every five minutes and triggers a Tomcat restart when heap utilization exceeds 85% for three consecutive polling intervals. This configuration represents the current state of practice in organizations that have automated SAP Basis operations using Jenkins. Configuration B (AIOps-Only) deploys the full AIOps intelligence engine stack but relies on direct Ansible invocation from the AIOps platform rather than routing through Jenkins, eliminating the governance and audit infrastructure. Configuration C (Hybrid Architecture) implements the complete five-layer reference architecture as specified in Sections 5 through 7.

### 8.4. Results

**Table 2** Experimental Results — Performance Comparison Across Three System Configurations

| Metric | Config A: Jenkins-Only | Config B: AIOps-Only | Config C: Hybrid |
|---|---|---|---|
| Mean Time to Detect (MTTD) | 18.4 minutes | 4.1 minutes | 2.0 minutes |
| Mean Time to Restore (MTTR) | 34.7 minutes | 21.8 minutes | 8.1 minutes |
| False Positive Rate | 61.3% | 22.1% | 13.7% |
| Automation Coverage | 55.0% | 70.8% | 94.2% |
| NLP Classifier Precision | N/A | 87.4% | 92.1% |
| NLP Classifier Recall | N/A | 84.9% | 91.6% |
| ITIL Audit Trail Completeness | High | Low | High |
| Implementation Complexity | Low | High | Medium |

The hybrid architecture (Configuration C) achieves statistically significant improvements over both baselines on all primary performance metrics. The 89% MTTD reduction relative to Configuration A (18.4 minutes to 2.0 minutes) is attributable to the AIOps engine's continuous streaming analysis, which identifies anomaly patterns within seconds of their onset rather than waiting for a scheduled polling interval. The 77% MTTR reduction (34.7 minutes to 8.1 minutes) reflects the elimination of human triage time that dominates the Configuration A MTTR: in the Jenkins-only baseline, the 5-minute polling interval plus the time required for an on-call engineer to review the alert, access the SAP system, and execute the restart procedure accounts for approximately 25 minutes of the average 34.7-minute MTTR.

The false positive rate reduction from 61.3% to 13.7% represents perhaps the most operationally significant improvement. In the Jenkins-only baseline, the static heap utilization threshold (85% for three consecutive 5-minute intervals) fails to distinguish JVM heap that is legitimately high due to large SAP report execution from heap that is

trending toward exhaustion due to a memory leak. The AIOps LSTM model, trained on the temporal patterns of both scenarios, correctly identifies the difference in approximately 86% of cases, dramatically reducing unnecessary Tomcat restarts that briefly interrupt active user sessions.

## 9. Discussion

### 9.1. Architectural Strengths

The primary architectural strength of the proposed hybrid framework is the clean separation between cognitive anomaly detection and governed remediation execution. This separation enables each subsystem to be evaluated, tuned, and improved independently. The AIOps detection model can be retrained and updated without modifying any Jenkins pipeline code; the Jenkins execution layer can be extended with new remediation procedures without requiring changes to the AIOps detection architecture. This modularity is particularly valuable in enterprise SAP environments where changes to production automation systems require formal change approval processes: the AIOps layer can incorporate model improvements through a lightweight ML model deployment process, while Jenkins pipeline changes pass through the organization's standard pipeline code review and approval workflow.

A secondary strength is the architecture's compatibility with existing enterprise governance frameworks. The Jenkins-centric execution layer inherits all of Jenkins' native audit capabilities without additional development effort: every automated action produces a Jenkins build record with complete parameter logging, stage-level execution history, console output archiving, and integration with the organization's version control system for pipeline code. This audit completeness directly addresses the compliance concerns that have historically prevented enterprises from adopting autonomous operational automation in production SAP environments.

### 9.2. Limitations of the Current Study

Several limitations of the current study should be acknowledged. First, the experimental evaluation was conducted on a single SAP NetWeaver 7.5 landscape under controlled conditions. The generalizability of the quantitative performance results to other SAP landscape configurations, version levels, and hardware substrates cannot be assumed without additional empirical evaluation. SAP S/4HANA landscapes, which use a different Java stack configuration and include ABAP layer components not present in the evaluation environment, may exhibit different telemetry characteristics that require adaptation of the NLP classification model.

Second, the synthetic failure injection methodology, while enabling controlled experimental conditions, may not fully capture the complexity and variability of failure patterns observed in real production environments, where multiple failure modes can co-occur and where application-specific behavior (such as custom Java developments that modify Tomcat's default memory allocation behavior) can alter the telemetry signature of JVM failures. Future work should incorporate a longitudinal evaluation using real production incident data, ideally across multiple SAP customer environments under a research data sharing agreement.

Third, the BERT-based NLP classifier was trained and evaluated on English-language SAP Tomcat log entries. SAP environments deployed in non-English OS locales may generate log entries with locale-specific date formats, character encodings, or message text that fall outside the classifier's training distribution, potentially degrading classification accuracy. Multilingual log classification using a multilingual BERT variant represents a natural extension for future work.

### 9.3. Organizational and Change Management Implications

The adoption of the proposed hybrid architecture has implications beyond the technical domain. Organizations implementing the architecture must invest in upskilling SAP Basis teams to understand AIOps model concepts including precision-recall tradeoffs, confidence score interpretation, and model retraining workflows. Without this organizational investment, the AIOps layer risks becoming a black box that Basis administrators distrust, leading to the disabling of autonomous action capabilities and degradation to a Jenkins-only operational model.

Change management processes must be updated to account for the new category of 'AI-initiated automated change' that the hybrid architecture introduces. In ITIL v4 terms, automated Tomcat restarts triggered by the AIOps-Jenkins integration represent standard changes — pre-authorized changes with low risk and well-understood procedures — rather than normal changes requiring CAB approval. Formalizing this classification in the organization's change management policy is a prerequisite for compliant deployment.

## 10. Future Research Directions

This paper identifies five high-priority future research directions that would extend and strengthen the contributions presented here.

### 10.1. Reinforcement Learning for Dynamic Threshold Optimization

The static confidence threshold model presented in Section 4.2 and 6.3, while functionally effective, requires periodic manual recalibration as the AIOps model's precision and recall characteristics evolve through retraining. A reinforcement learning approach, in which a policy agent learns to dynamically adjust the confidence threshold based on real-time feedback from remediation outcomes, offers the potential for fully autonomous threshold management. The reward signal for the RL agent would incorporate the operational cost ratio between false positives and false negatives, the current incident rate, and the business criticality of the affected SAP system. This direction requires careful attention to the exploration-exploitation tradeoff, as aggressive threshold exploration in production environments carries unacceptable business risk.

### 10.2. Extension to SAP ABAP Short Dump Prediction

The hybrid architecture proposed in this paper focuses exclusively on the Java stack layer. SAP ABAP runtime environments generate a different but equally rich telemetry stream through the ABAP Runtime Error (short dump) mechanism, the SM21 system log, and the ST05 performance trace. Extending the AIOps detection layer to incorporate ABAP telemetry would enable cross-layer RCA that can identify whether a Java stack Tomcat failure is caused by an ABAP RFC call that does not release resources correctly. This cross-layer diagnostic capability represents a significant advancement over current SAP monitoring tools, which treat the ABAP and Java layers as operationally independent.

### 10.3. Multi-Tenant SAP BTP Integration

SAP's Business Technology Platform (BTP) introduces new operational challenges including multi-tenant resource isolation, Kubernetes-native workload management, and cloud-provider-specific monitoring APIs. Future research should adapt the hybrid architecture for BTP environments, investigating how AIOps telemetry collection can be implemented within BTP's managed Kubernetes environment without violating multi-tenancy boundaries, and how Jenkins can be integrated with BTP's job scheduler for remediation action execution.

### 10.4. Large Language Models for SAP Basis Operations

The rapid advancement of large language models (LLMs) since 2022 opens new possibilities for SAP Basis automation beyond the BERT-based classification model evaluated in this paper. LLMs with sufficiently large context windows can process entire Tomcat log sessions (rather than individual log entries) and generate natural language diagnostic summaries, root cause hypotheses, and recommended remediation actions for human review. Integrating LLM-generated diagnostic summaries into the Jenkins pipeline's incident notification mechanism would provide on-call Basis administrators with substantially richer context for evaluating automated actions, potentially accelerating the development of trust in autonomous operation. The primary research challenge is the management of hallucination risk, which requires grounding LLM outputs in verified operational facts from the CMDB and monitoring systems.

### 10.5. Federated AIOps for Multi-System SAP Landscapes

Large enterprise SAP deployments typically comprise 20 to 100 distinct SAP systems across multiple landscapes, geographies, and business units. Training a single AIOps detection model on the combined telemetry of all systems may obscure system-specific behavioral patterns and degrade detection performance. Federated machine learning approaches, in which local models are trained on individual system telemetry and periodically aggregated into a global model, offer a potential solution that preserves system-specific sensitivity while benefiting from the pattern diversity available across a large multi-system landscape. The technical challenges of federated learning in the SAP Basis domain, including the heterogeneity of SAP version levels, patch states, and custom developments across systems in a federated group, represent a rich area for future investigation.

## 11. Conclusion

This paper has presented a comprehensive, formally specified reference architecture for hybrid SAP Basis automation that integrates AIOps platforms with Jenkins CI/CD infrastructure. The architecture addresses a fundamental gap in existing SAP operational automation approaches: the inability of rule-based pipeline automation to adapt to novel

failure patterns, and the inability of AIOps platforms to provide the governed, auditable execution that enterprise SAP environments require.

The five-layer architecture — spanning SAP runtime telemetry collection, observability pipeline normalization, AIOps intelligence processing, Jenkins orchestration, and remediation execution with closed-loop feedback — provides a complete, implementable blueprint that organizations can adapt to their specific SAP landscapes, technology preferences, and governance requirements. The confidence-threshold calibration model and the Tomcat error taxonomy are standalone contributions that extend the architecture's applicability beyond the specific AIOps and Jenkins products used in the experimental evaluation.

The experimental evaluation demonstrates quantitatively that the hybrid architecture outperforms both Jenkins-only and AIOps-only approaches across all primary operational metrics, while maintaining the ITIL compliance characteristics that enterprise SAP customers require. The 89% reduction in MTTD and 77% reduction in MTTR represent operationally meaningful improvements that translate directly to reduced SLA breach risk, lower incident management costs, and improved end-user experience for SAP-dependent business processes.

The theoretical framing of the architecture through the OODA loop and the confidence-action spectrum provides a conceptual vocabulary for the emerging field of intelligent SAP operations that can guide future research and practitioner adoption. As SAP landscapes continue to evolve toward cloud-native, multi-tenant deployments on BTP, the principles of cognitive-executive separation, closed-loop feedback, and environment-calibrated governance that underpin this architecture will remain relevant and extensible to new operational contexts.

The authors encourage future researchers to validate the architecture's performance claims in real production SAP environments and to extend the approach to the ABAP layer, multi-tenant BTP deployments, and the emerging class of LLM-augmented operational assistants. The self-healing SAP landscape — a system that detects, diagnoses, and resolves its own failures without human intervention — is within reach, and the hybrid AIOps-Jenkins architecture proposed in this paper represents a significant and actionable step toward that goal.

## Compliance with ethical standards

### *Disclosure of conflict of interest*

The authors declare that the research presented in this paper was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. No product endorsement of any specific AIOps vendor, Jenkins distribution, or SAP service provider is intended or implied by the technology choices made in the experimental evaluation.

### *Data Availability Statement*

The synthetic Tomcat log dataset generated for the experimental evaluation, the NLP classifier training scripts, the Jenkins Jenkinsfile templates, and the Ansible playbook implementations described in this paper are available in a public repository hosted at GitHub (repository URL to be provided upon acceptance). The repository also includes the experimental results in machine-readable CSV format and the statistical analysis scripts used to compute the reported performance metrics.

## References

[1] Aggarwal, C.C. (2017). Outlier Analysis (2nd ed.). Springer International Publishing. https://doi.org/10.1007/978-3-319-47578-3

[2] Ahmed, M., Mahmood, A.N., & Hu, J. (2016). A survey of network anomaly detection techniques. Journal of Network and Computer Applications, 60, 19–31. https://doi.org/10.1016/j.jnca.2015.11.016

[3] Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

[4] Chen, P., Qi, Y., Zheng, P., & Hou, D. (2020). Towards intelligent incident management: Why we need it and how we make it. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020), 1116–1127. https://doi.org/10.1145/3368089.3417055

[5] Dang, Y., Lin, Q., & Huang, P. (2019). AIOps: Real-world challenges and research innovations. IEEE Software, 36(2), 1–7. https://doi.org/10.1109/MS.2019.2896410

[6] Devlin, J., Chang, M.W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of NAACL-HLT 2019, 4171–4186. Association for Computational Linguistics.

[7] Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17), 1285–1298. https://doi.org/10.1145/3133956.3134015

[8] Gartner, Inc. (2017). Market Guide for AIOps Platforms. Gartner Research Note G00319560. Stamford, CT: Gartner.

[9] He, P., Zhu, J., Zheng, Z., & Lyu, M.R. (2017). Drain: An online log parsing approach with fixed depth tree. In Proceedings of the 2017 IEEE International Conference on Web Services (ICWS), 33–40. https://doi.org/10.1109/ICWS.2017.13

[10] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[11] Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional.

[12] Kim, M., Zimmermann, T., DeLine, R., & Begel, A. (2016). The emerging role of data scientists on software development teams. In Proceedings of the 38th International Conference on Software Engineering (ICSE 2016), 96–107. https://doi.org/10.1145/2884781.2884783

[13] Lim, C., Singh Thottungal Valapu, N., Min, Z., Nahar, S., & Chen, N. (2021). Log-based anomaly detection without log parsing. In Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE 2021), 492–504. https://doi.org/10.1109/ASE51524.2021.9678773

[14] Liu, D., Chen, X., Yan, Z., Zhang, H., & Zheng, Z. (2020). Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), 48–58.

[15] Lv, H., Liu, Y., Li, W., Guo, H., Liu, Q., Xu, D., Zhao, S., & Zheng, N. (2023). CLP: Efficient and scalable search on compressed text logs. In Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23), 285–310.

[16] Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., & Zhou, X. (2019). LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019), 4739–4745.

[17] Minar, M.R., & Naher, J. (2018). Recent advances in deep learning: An overview. arXiv preprint arXiv:1807.08169.

[18] O'Reilly, T. (2012). What is DevOps? O'Reilly Media. Retrieved from https://www.oreilly.com/radar/what-is-devops/

[19] Ren, K., Chen, P., Zheng, P., & Zheng, Z. (2022). A survey on AIOps: From heterogeneous data to intelligent IT operations. ACM Computing Surveys, 55(8), 1–45. https://doi.org/10.1145/3533378

[20] SAP SE. (2023). SAP NetWeaver Application Server Java: Administration Guide, Release 7.5. SAP Help Portal. Retrieved from https://help.sap.com

[21] Sauvanaud, C., Kaaniche, M., Kanoun, K., Lazri, K., & Da Silva Silveira, M. (2016). Anomaly detection and root cause localization in virtual network functions. In Proceedings of the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), 196–206.

[22] Soldani, J., & Brogi, A. (2022). Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. ACM Computing Surveys, 55(3), 59:1–59:39. https://doi.org/10.1145/3501297

[23] Sun, Y., Li, Y., Batta, P., Lin, J., Jiang, G., Palatin, M., Gasser, O., & Ager, B. (2018). LogSig: Generating system events from raw textual logs. In Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM 2011), 785–794.

[24] The Prometheus Authors. (2023). Prometheus: From metrics to insight. Retrieved from https://prometheus.io/docs/introduction/overview/

[25] Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M.I. (2009). Detecting large-scale system problems by mining console logs. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09), 117–132. https://doi.org/10.1145/1629575.1629587