World Journal of Advanced Engineering Technology and Sciences
eISSN: 2582-8266
Cross Ref DOI: 10.30574/wjaets
Journal homepage: https://wjaets.com/

(REVIEW ARTICLE)

Check for updates

TELEMETRY INTELLIGENCE FOR SAP CLOUD ALM: AN OPENTELEMETRY-BASED FRAMEWORK FOR CORRELATED, LLM-AUGMENTED ROOT CAUSE ANALYSIS

Karthik Paramasivam *

PSG College of Technology, Peelamedu, Coimbatore, Tamil Nadu 641004, India.
* Corresponding Author

ORCID Details
Karthik Paramasivam: https://orcid.org/0009-0009-7437-1975

World Journal of Advanced Engineering Technology and Sciences, 2026, 21(02), 001–009

Publication history: Received on 14 June 2026; revised on 12 October 2026; accepted on 14 October 2026

Article DOI: https://doi.org/10.30574/wjaets.2026.21.2.0372

Copyright © 2026 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution License 4.0

ABSTRACT

In SAP landscapes, the indirect operational costs of delayed or incorrect root-cause analysis include increased mean time to resolution, increased risk of SLA breaches and overburdening expert analysts who have to connect the dots on multiple monitoring tools and correlate them before formulating a hypothesis. SAP Cloud ALM offers solutions on the data collection end of the problem with Health Monitoring, Integration and Exception Monitoring, status events and a documented API layer. The issue is not a lack of data; rather, evidence from four different monitoring applications must be synthesized manually before it can support action. This paper proposes a telemetry intelligence framework bounded to the four SAP Cloud ALM evidence domains, drawing on the Analytics API, Raw Metrics API, and Raw Logs API as extraction sources. OpenTelemetry normalizes diverse signals into a unified metrics, logs, and events schema. A multi-dimensional correlation engine then correlates related signals by time window, managed component, service, endpoint and message flow. With this correlated evidence, under strict evidence constraints, a large language model provides the root-cause textual narrative, the confidence level and the recommended action for each claim, together with the corresponding telemetry evidence on which the claim is based. The health of the extraction pipeline is treated as an additional reliability-control layer: the confidence score for each LLM explanation is lowered if the extraction pipeline's health is poor. The contribution is completed by a six-class predictive operations action model and telemetry intelligence maturity model, consisting of five levels. Preliminary validation was conducted in a real SAP Cloud ALM tenant using Health Monitoring and Integration and Exception Monitoring fields across more than five cases. This paper does not include a formal large-scale evaluation or a production-grade scoring implementation for BPM or RUM, which are addressed in companion papers.

Keywords: SAP Cloud ALM; OpenTelemetry; LLM-assisted root cause analysis; Telemetry Correlation; Predictive Operations; Anomaly Detection

1 INTRODUCTION

Delayed or incorrect root-cause analysis in SAP operations is not just a hindrance, but a tangible risk to the business. Each minute an incident is not resolved, SLA commitments are lost, dependent processes stall and automated recovery becomes more and more difficult. In large hybrid SAP landscapes, mean time to resolution is often prolonged not only by limited monitoring data but also by the need to correlate evidence across separate applications. This paper begins to tackle this manual assembly step directly.

Each signal type is already displayed in an application for monitoring in SAP Cloud ALM. Health Monitoring includes managed-component health [1] while Integration and Exception Monitoring provide end-to-end visibility of interface calls and message flows [2] and status events report downtime and degradation from Business Service Management [3]. The challenge is not visibility but correlation and in case of an active incident, it is the problem of determining which signals belong together across the four separate applications.

A structured approach is possible through the API layer. SAP documents an Analytics API, domain-specific Health and Exception Monitoring Analytics APIs, Raw Data Outbound Metrics API which has OpenTelemetry Metrics Exporter format support, and Raw Data Inbound Metrics and Logs APIs [4,5,6,7]. The SAP Cloud ALM APIs category in Integration and Exception Monitoring turns on API self-monitoring [8]. These constitute a fully documented extraction pipeline.

This paper presents a telemetry intelligence framework which practically utilizes this pipeline. The first is an OpenTelemetry-based normalization layer which transforms the collected evidence into a uniform schema, the second is a correlation engine that correlates the signals, and the final is a large language model (LLM) running under strict evidence constraints to transform the correlated evidence into an actionable explanation that an operator can act upon. The primary research question is: How to normalize and correlate SAP Cloud ALM Health Monitoring, SAP Cloud ALM Integration and Exception Monitoring, SAP Cloud ALM status events and SAP Cloud ALM API-extracted raw telemetry data for evidence-driven, LLM-powered predictive operations.

2 SAP CLOUD ALM TELEMETRY ARCHITECTURE

This section provides the basis of the framework in terms of the specific evidence domains in SAP Cloud ALM. Health Monitoring services monitor the health of managed cloud services and technical systems based on the metrics collected and then use them to calculate the overall health of managed components [1]. Business Context Attributes, like the order number or employee ID, are used to track messages at the message level with Integration and Exception Monitoring [2]. Exception evidence, including family, frequency, and recurrence, is surfaced in the same application. The correlation analysis is anchored by a known disruption window, even in the presence of noisy underlying metrics through status events from Business Service Management [3].

Table 1: Evidence domains and SAP sources

Evidence Domain	Representative Fields	Role in Framework
Health Monitoring	Availability, connection health, CPU, memory, service/endpoint status	Earliest technical degradation signal
Integration Monitoring	Message throughput, failed messages, retry counts, queue depth	Interface and message-flow evidence
Exception Monitoring	Exception family, frequency, recurrence	Failure pattern and clustering evidence
Status Events	Downtime, degradation, maintenance windows	Disruption-window anchor
Analytics / Raw APIs	Structured queries; OpenMetrics/OTel metric streams; external logs	Extraction and cross-tool interoperability
API Self-Monitoring	API health, failure rate, latency, availability	Telemetry pipeline reliability control

The Analytics API reads the SAP Cloud ALM measurements filtered on the analytic dimensions, and domain-specific variants are available for Health Monitoring and Exception Monitoring [4,5]. Cloud ALM metrics can be pulled by the Raw Data Outbound Metrics API in OpenMetrics or OpenTelemetry format, and Cloud ALM can push telemetry through the Inbound Metrics and Logs APIs [6,7]. However, API self-monitoring must be explicitly requested or enabled and

serves as an important control mechanism: it makes it possible to see the condition of the extraction pipeline before conclusions can be drawn downstream [8].

3 OPENTELEMETRY-BASED TELEMETRY FABRIC

The framework does not provide dedicated integration for each evidence domain. Instead, these evidence domains are translated into a standard OpenTelemetry-style representation (metrics, logs and events) with a managed-component identifier, a time indication, dimensions and a severity indicator. This approach is consistent with SAP managed connectivity for cloud services, which leverages OpenTelemetry and SAP's next-generation data-collection infrastructure. The Raw Data Outbound Metrics API explicitly supports OpenTelemetry Metrics Exporter format [6,9].

Telemetry fabric can be used to normalize across four source families: SAP cloud services, the Integration Suite, on-premises or RISE-connected managed components and external systems via the Raw Logs API. It all leads into its local monitoring application, from which it then flows into the same OpenTelemetry normalization layer, which means that all the signal families can speak the same language before reaching the correlation engine. The end-to-end architecture is shown in Figure 1.

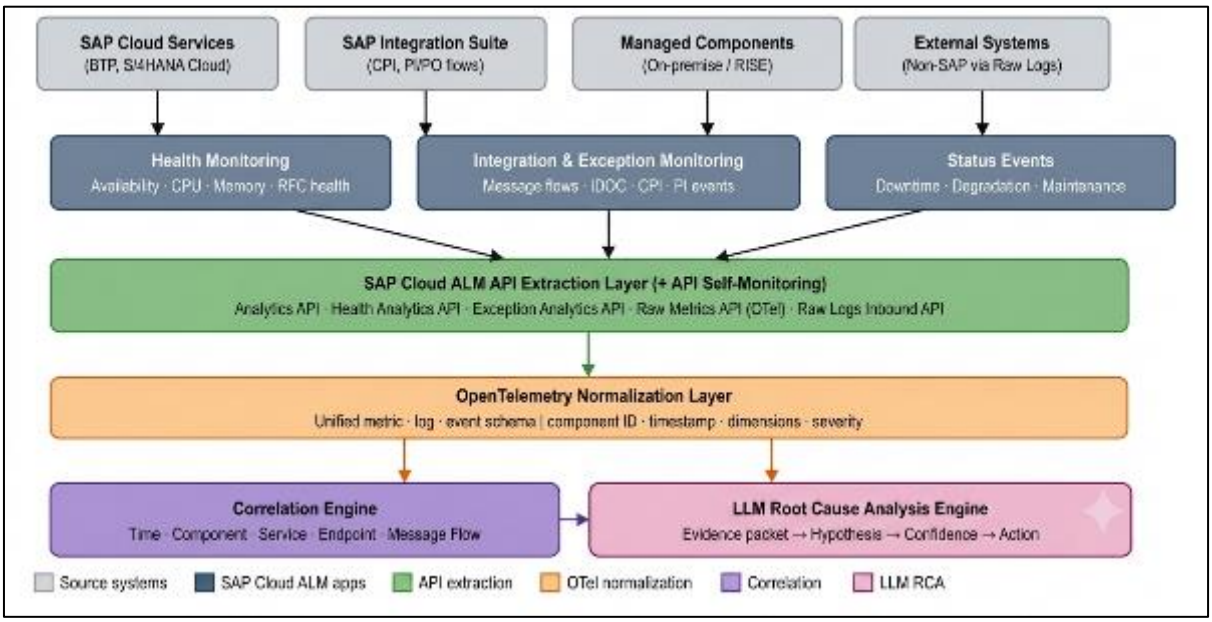


Figure 1: OpenTelemetry-Based Telemetry Fabric Architecture

Table 2: Telemetry fabric layers and responsibilities

Layer	Key Inputs	Output
Source systems	SAP cloud services, Integration Suite, managed components, external systems	Raw telemetry
Monitoring apps	Health Monitoring, Integration & Exception Monitoring, status events	Cloud ALM evidence store
OpenTelemetry normalization	Analytics, Raw Metrics and Raw Logs APIs	Normalized telemetry stream
Correlation engine	Feature packets and history	Incident candidates with scores
LLM RCA engine	Correlated, scored evidence packets	Root-cause narrative and confidence
Operations dashboard	LLM explanation and evidence	Closed-loop analyst workflow

4 TELEMETRY CORRELATION MODEL

Normalization is used to deal with the representation problem, whereas correlation is used to tackle the grouping problem (which signal belongs to which incident narrative). The correlation engine codifies the five dimensions that an analyst normally would have to apply manually: time, managed component, service, endpoint, and message flow. Temporal alignment ensures consistent comparisons; all signals must be within a common window, which is programmable. Managed-component and service identities further reduce the range of candidates to provide evidence to support a plausible root cause. Endpoint and message-flow identities provide finer granularity in incidents that contain an integration element, where the fault is in a particular call path. Failure pattern and disruption window evidence complete the picture with exception family and exception status-event evidence [10].

Correlation strength is rated as low, moderate or high. Weak correlations are left to identify trends and not escalate. Moderate correlations flag the enrichment of evidence, but do not lead to a conclusion. An LLM root cause packet will only be created for strong correlations, when multiple dimensions align and evidence is internally consistent. By using a staged approach, the LLM layer is not called on weak or ambiguous evidence. The entire correlation routing is included in Figure 2.

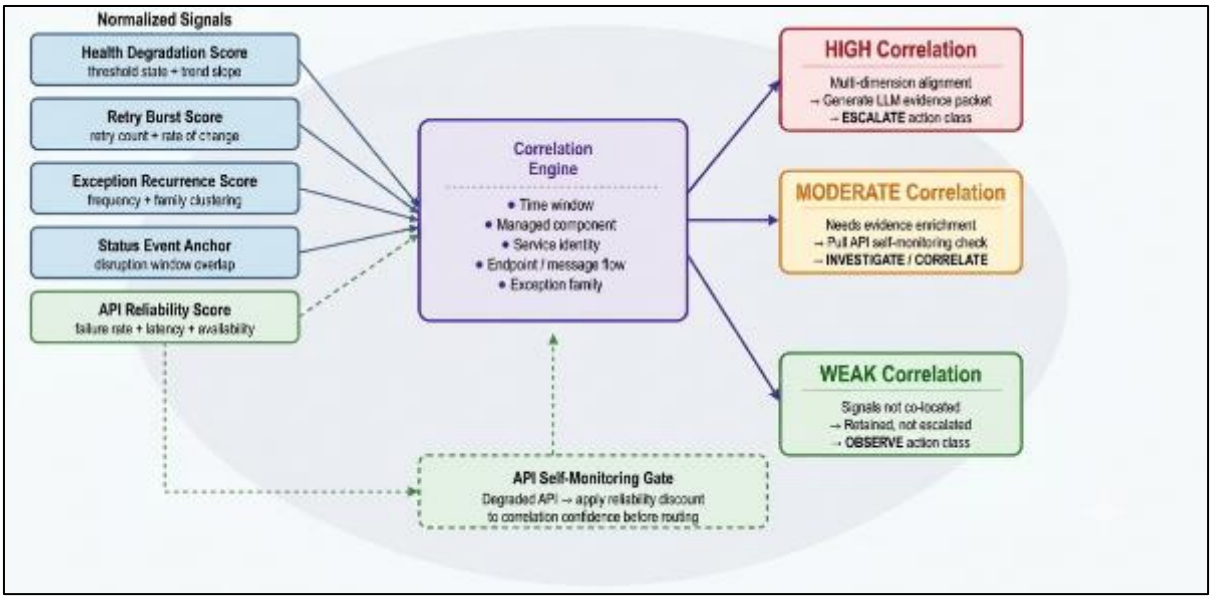


Figure 2: Telemetry Correlation Model and Routing Logic

Table 3: Correlation dimensions

Dimension	Purpose	Example
Time	Anchors signal eligibility	30-minute rolling window around incident start
Managed component	Groups evidence by affected system or service instance	Health and exception evidence sharing one component ID
Service	Groups at a coarser, business-relevant level	Multiple components belonging to one logical service
Endpoint	Narrows integration evidence to a specific call path	A single API endpoint with rising retries
Message flow	Tracks an integration scenario end-to-end	An interface scenario with business context attributes
Exception family	Groups exceptions by failure type	Repeated timeout exceptions across several calls
Status event	Anchors a known disruption window	Maintenance or degradation event overlapping incident window

5 FEATURE ENGINEERING AND LLM ROOT CAUSE ANALYSIS

Regardless of normalization, raw telemetry needs to be transformed into features to be used by a correlation engine or LLM. A health degradation score is a combination of threshold state and trend slope. A retry burst score is the combination of retry count and rate of change. An exception recurrence score combines frequency and exception-family clustering. The API reliability score, which is based on the API's self-monitoring evidence, takes into account failure rate, latency, and availability. A service impact score is a weighted score for the duration and severity of an identified disruption window. The formulas are intentionally kept simple and reportable to support operator trust and auditability [8, 10, 12].

The LLM layer is only given a structured evidence packet that is created in the correlation and feature-engineering steps. It never gets raw logs, unfiltered telemetry, or direct API access. A tightly packed evidence packet lowers the chances that a viable, but unsupported hypothesis will emerge. A guarded prompt template will ask the model to make sure it distinguishes between observation and inference, provide entries for supporting evidence and missing evidence, and reject hypothesis generation. Each output field is related to a particular entry in the evidence packet. The overall LLM governance flow is depicted in Figure 3.

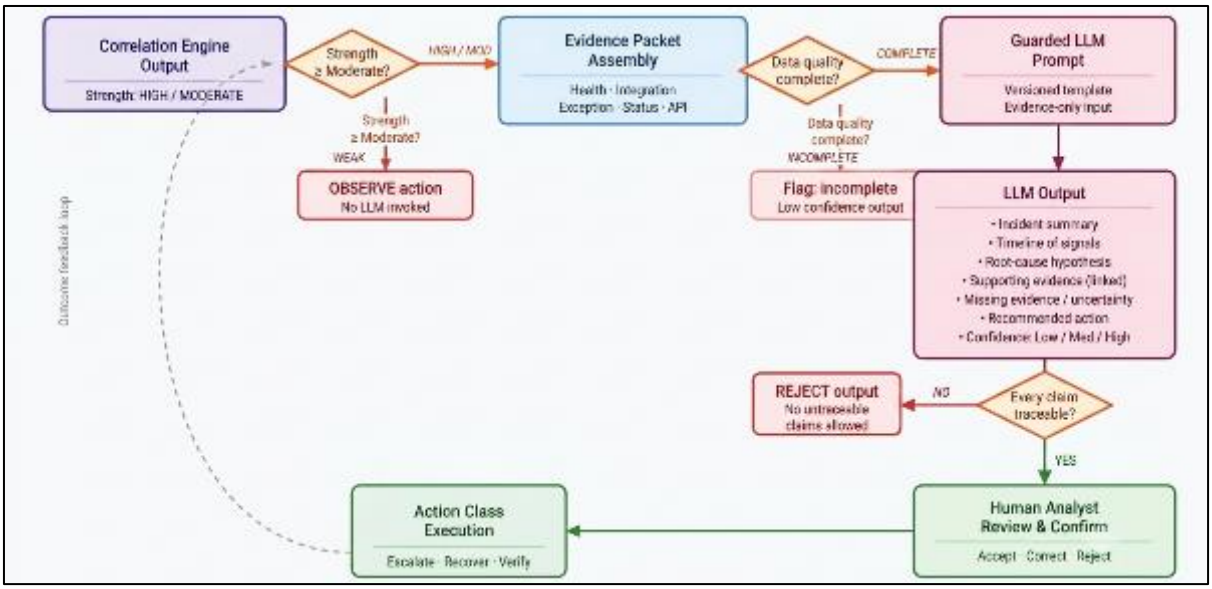


Figure 3: LLM Root Cause Analysis Governance Flow

Table 4: Feature engineering by evidence source

Source	Feature	Root-Cause Value
Health	Health Degradation Score (threshold state + trend slope)	Shows whether a component was declining before the incident
Integration	Retry Burst Score (retry count + rate of change)	Shows whether interface instability is building or isolated
Exception	Exception Recurrence Score (frequency + family clustering)	Shows whether a failure pattern is repeating rather than incidental
API Self-Monitoring	API Reliability Score (failure rate, latency, availability)	Controls confidence in every downstream conclusion
Status Event	Service Impact Score (duration + severity of disruption window)	Anchors incident to a known event rather than inferred timing

6 PREDICTIVE OPERATIONS AND GOVERNANCE

A correlated incident covers only half of an operations workflow—deciding what to do next is the other half. There are six classes of actions defined in the framework which correspond directly with outputs of correlation strength and

confidence scoring. Observe: When there is a weak correlation and low severity, evidence is stored to be used for trend visibility but does not cause an action. Investigate: If there is some correlation, but it is not large enough, then it is worth looking into the matter first before any further automation. Correlate is used when evidence is incomplete, and then the system enriches the evidence. Escalate is used when there is high confidence and a strong correlation and leads to the incident being assigned to a named owner and the workflow being opened. Recover applies during remediation, with continued monitoring until the affected signals return to baseline. Verify is when remediation is thought to be complete and must be confirmed by stabilization before closure.

There are four governance controls that span all levels. Each LLM prompt template is generated to produce a corresponding explanation which is saved together with the prompt and model versions that created it. API governance also considers the extraction pipeline as a dependency to monitor and adds API self-monitoring evidence to the confidence score directly, so that when the API is degraded, it does not give a false sense of confidence by giving a false explanation. Evidence traceability must ensure that each LLM claim is able to be traced to a specific evidence packet entry. Human validation: The LLM does not close an incident, set an owner, or take action to remediate without analyst validation [13, 14].

Table 5: Predictive operations action classes

Action Class	Trigger Condition	Operator Expectation
Observe	Weak correlation, low severity	No action; retained for trend visibility
Investigate	Moderate correlation, unclear cause	Analyst reviews before further action
Correlate	Evidence incomplete or ambiguous	System enriches evidence before concluding
Escalate	Strong correlation, high confidence	Named owner assigned; incident workflow opened
Recover	Remediation in progress	Monitor for return to baseline across affected signals
Verify	Remediation believed complete	Confirm stabilization before closing the incident

7 TELEMETRY INTELLIGENCE MATURITY MODEL

It is possible for two organizations to both use SAP Cloud ALM for operations, but with different levels of operational maturity. The maturity model helps organizations assess their current operational maturity and make informed decisions about the next stage of telemetry-intelligence adoption. Having the model configured is not enough; it needs to be governed to be taken to the next level – baselines, API reliability, correlation feature quality and human validation practices should be put in place before the next level becomes useful [14,15].

Table 6: Telemetry intelligence maturity model

Level	Characteristic State	Key Dependency
Level 1 - Manual Monitoring	Health, Integration, and Exception apps reviewed independently, screen by screen	Individual analyst expertise
Level 2 - Standardized Monitoring	Consistent monitoring baseline; API self-monitoring active	Monitoring configuration discipline
Level 3 - API-Enabled Analytics	Analytics and Raw Data APIs feed external reporting	API reliability and access governance
Level 4 - Telemetry Correlation	Health, integration, exception, and API signals correlated by time and component	Correlation feature quality
Level 5 - Predictive Telemetry Intelligence	Grounded LLM RCA and predictive action with closed-loop feedback	Governance, trust, and evidence traceability

8 VALIDATION

The manual prototype of the evidence-packet-to-LLM-explanation workflow was tested using the following fields extracted from a connected SAP Cloud ALM tenant using the Analytics and Raw Metrics and Logs APIs: Health Monitoring and Integration and Exception Monitoring. The exercise is not automated: an analyst has to pull the KPI

values for the window of an incident and put them together in the evidence packet structure described in Section 5 and submit it to an LLM under a guarded prompt.

The Health Monitoring fields were selected from the Communication area of the Health Monitoring application: tRFC entries in status CPICERR, SYSFAIL, and SYSLOAD; bgRFC TIN number of entries and errors; qRFC IN and OUT entries, errors and age. These fields describe the state of the ABAP transactional and queued RFC communication layer and are good indicators of degradation in the integration layer before it is detected as a confirmed exception.

There were six types of Integration and Exception Monitoring evidence: Erroneous PI Messages Detected (Single); IDOC Messages Detected with High Processing Time; Payroll IDOC Issue Detected; Anomalies Detected in the Number of IDOC Messages Processed; Erroneous Integration Artifact; and Erroneous CPI Message Detected. These include classic ABAP-side integration evidence and cloud integration evidence, on both sides of a hybrid integration path. In a small number of manually reviewed cases the root-cause hypothesis generated by the LLM was verified against the actual root cause independently established [15, 16].

Table 7: Health Monitoring fields used in validation

Field	What It Measures
tRFC entries in status CPICERR	Transactional RFC entries that failed at the CPI/communication layer
tRFC entries in status SYSFAIL	Transactional RFC entries that failed due to a system-side error
tRFC entries in status SYSLOAD	Transactional RFC entries queued due to system load
bgRFC TIN: number of entries / errors	Volume and error count for background RFC inbound processing
qRFC IN: entries / errors / age	Volume, errors, and oldest-entry age for inbound queued RFC
qRFC OUT: entries / errors / age	Volume, errors, and oldest-entry age for outbound queued RFC

Table 8: Integration and Exception Monitoring event types used in validation

Event Type	Integration Path
Erroneous PI Messages Detected (Single)	Cloud-to-cloud or cloud-to-on-premise PI integration
IDOC Messages Detected with High Processing Time	Classic ABAP-side IDOC processing
Payroll IDOC Issue Detected	ABAP-side IDOC processing, payroll-specific detection rule
Anomalies Detected in the Number of IDOC Messages Processed	Classic ABAP-side IDOC volume monitoring
Erroneous Integration Artifact	General integration artifact monitoring
Erroneous CPI Message Detected	SAP Integration Suite / Cloud Integration

It is important to clarify what the validation does and does not demonstrate. The validation did not include a large number of cases, the evidence assembly process is not automated, and no formal precision or recall figure is reported, because the sample size is insufficient for a reliable assessment. The validation confirmed that only using real fields from an already connected tenant in this evidence-packet structure (as outlined in Section 5), the LLM was able to generate explanations that an analyst found consistent with the confirmed actual cause in several different cases. It does not demonstrate statistical accuracy at scale, generalizability across landscape configurations, or the performance of an automated pipeline.

These would be needed to improve validation: (1) A much greater number of cases from multiple tenants and landscape configurations; (2) Automation of evidence extraction and correlation (eliminating the manual evidence assembly step); (3) A formal evaluation protocol to measure precision and recall of the evidence correlation, time to triage compared to manual evidence, and the rate of acceptance of LLM-generated explanations by analysts without correction; (4) A methodology based on a rolling window of time to prevent data leakage.

9 EVALUATION FRAMEWORK

To measure such a framework, one needs to consider whether operators' actions change for the better – not just whether the anomaly scores are statistically correct. A model capable of identifying all anomalies with high accuracy but that gives explanations that operators do not trust, has not solved the problem that the paper tackles.

Quantitative metrics involve correlation accuracy and recall regarding verified incidents, time taken to triage versus manual cross-application review, accuracy of API self-monitoring as a confidence input and the rate of LLM explanations accepted without correction. Qualitative measures are the analyst's rated clarity of explanation and trust, as well as the matching of the recommended action class with an independent choice by the experienced operator. Due to the sequential nature of operations data, a rolling time-window validation approach is suitable.

Table 9: Research propositions

Prop.	Statement
P1	Correlating health, integration, and exception signals will reduce manual triage time compared with reviewing each monitoring application independently
P2	LLM explanations grounded in correlated, scored evidence will be rated more useful by analysts than reviewing the same evidence as a raw alert list
P3	Treating API self-monitoring as a confidence input will reduce overconfident explanations during periods of telemetry pipeline degradation
P4	Multi-dimension correlation will identify more operationally meaningful incidents than single-signal threshold alerting
P5	Evidence traceability requirements will increase analyst trust in LLM-generated explanations compared with unconstrained narrative output
P6	Automating the manual evidence-assembly process validated in Section 8 will preserve accuracy while reducing time-to-evidence-packet from manual minutes to near-real-time

10 DISCUSSION

The problem is not a lack of data but a lack of structure for interpreting four evidence domains as a unified operational picture as four separate dashboards. For most operating situations such incidents are already covered in health evidence, integration evidence, exception evidence and status-event evidence. They did not share a schema, a discipline that used to relate the entities, and a bounded method of transforming the result into an operator's language in a way that doesn't require re-processing.

Two design decisions must be highlighted. It's important to consider API self-monitoring as a governance issue, rather than an operational one: If a correlation and explanation pipeline does not validate the health of its data source, the pipeline can provide a confident-sounding but incorrect answer which can be worse than nothing. This is a deliberate choice as it is a less general model than one that would be able to search logs directly, but a safer model as it cannot generate evidence-grounded explanations from information that was not provided.

Two related domains are not covered in this paper but are addressed in companion work using the same evidence-grounded approach. One is Business Process Monitoring, which uses evidence as Business Process KPI drift, and not system telemetry. The other concerns Real User Monitoring and evidence is user experience, instead of backend health. A logical next step after such a model is one that is cross-domain, with business-impact signals as strong evidence from those two domains as additional context for the technical correlation described here.

11 CONCLUSION

All the building blocks for system-level telemetry intelligence are already provided in SAP Cloud ALM: Health Monitoring, Integration and Exception Monitoring, event status and a fully documented API layer, including API self-monitoring. This paper does not solve a data-availability problem; it proposes a way to normalize, correlate, and explain existing SAP Cloud ALM evidence as a unified operational narrative.

The system described herein is limited to that extent. Four core evidence domains are unified into a single schema in a fabric based on OpenTelemetry. The correlation engine groups signals by component, endpoint, service, message flow,

and time window. API self-monitoring is not assumed to be reliable by default; its status is explicitly incorporated into the confidence level for each conclusion. Remaining and correlated evidence is converted into a traceable explanation by a bounded LLM layer, and a six-class action model converts the explanation into an action that can be performed.

This evidence structure is worked out based on manual validation against a connected tenant, with real Health Monitoring and Integration and Exception Monitoring fields for over five cases; the logical next step is larger-scale, automated validation.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Disclosure of conflict of interest

The author declares no conflict of interest related to this conceptual framework paper.

REFERENCES

[1] Faseeha, U., Syed, H. J., Samad, F., Zehra, S., & Ahmed, H. (2025). Observability in microservices: An in-depth exploration of frameworks, challenges, and deployment paradigms. IEEE Access, 13, 72011–72039.

[2] Kosińska, J., Baliś, B., Konieczny, M., Malawski, M., & Zieliński, S. (2023). Toward the observability of cloud-native applications: The overview of the state-of-the-art. IEEE Access, 11, 73036–73052.

[3] Usman, M., Ferlin, S., Brunström, A., & Taheri, J. (2022). A survey on observability of distributed edge & container-based microservices. IEEE Access, 10, 86904–86919.

[4] Wang, X.-Z., & Tseng, C.-C. (2024). Building an observable system based on OpenTelemetry. In 2024 International Computer Symposium (ICS) (pp. 76–81). IEEE.

[5] Cinque, M., Della Corte, R., & Pecchia, A. (2022). Microservices monitoring with event logs and black box execution tracing. IEEE Transactions on Services Computing, 15(1), 294–307.

[6] He, S., He, P., Chen, Z., Yang, T., Su, Y., & Lyu, M. R. (2021). A survey on automated log analysis for reliability engineering. ACM Computing Surveys, 54(6), Article 130, 1–37.

[7] Notaro, P., Cardoso, J., Gerndt, M., & Andrikopoulos, V. (2021). A survey of AIOps methods for failure management. ACM Transactions on Intelligent Systems and Technology, 12(6), Article 81, 1–45.

[8] 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), Article 59, 1–39.

[9] Barata, L. M., Sequeira, S., Lopes, E. R. L., Inácio, P. R. M., & Freire, M. M. (2026). Anomaly detection and root-cause identification in microservices: A survey. Cluster Computing, 29, Article 309.

[10] Darban, Z. Z., Webb, G. I., Pan, S., Aggarwal, C. C., & Salehi, M. (2024). Deep learning for time series anomaly detection: A survey. ACM Computing Surveys, 57(1), Article 15.

[11] Zhang, L., Jia, T., Jia, M., Wu, Y., Liu, A., Yang, Y., Wu, Z., Hu, X., Yu, P. S., & Li, Y. (2025). A survey of AIOps in the era of large language models. ACM Computing Surveys, 58(2), 1–35.

[12] Han, Y., Du, Q., Huang, Y., Wu, J., Tian, F., & He, C. (2024). The potential of one-shot failure root cause analysis: Collaboration of the large language model and small classifier. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (pp. 931–943). ACM.

[13] Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information Fusion, 58, 82–115.

[14] Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature Machine Intelligence, 1(5), 206–215.

[15] Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (MLOps): Overview, definition, and architecture. IEEE Access, 11, 31866–31879.

[16] Birkstedt, T., Minkkinen, M., Tandon, A., & Mäntymäki, M. (2023). AI governance: Themes, knowledge gaps and future agendas. Internet Research, 33(7), 133–167.