Universal Library of Innovative Research and Studies Research Article

ISSN: 3065-0003 | Volume 3, Issue 1

Open Access | PP: 33-39

DOI: https://doi.org/10.70315/uloap.ulirs.2026.0301006

Methods and Tools for Ensuring Observability in Distributed Software Systems

Vadym Shevchenko

Lead Software Engineer, PhotoDay, Kosice, Slovakia.

Abstract

The article presents a comprehensive analysis of methods and tools for ensuring observability in distributed software systems, including high-performance computing environments, container-based platforms, and microservice architectures. The study is based on comparing structural properties of telemetry, data-collection models, and mechanisms for reconstructing causal relationships as described in contemporary scientific publications. It examines differences in execution-context formation, measurement accuracy, and metric reproducibility across environments with varying workload dynamics. Special attention is given to the impact of architectural constraints on resource attribution, trace-data interpretation, and the stability of analytical outcomes. The practical consequences for engineers are outlined, including the need to standardize context propagation in HPC, develop valid energy-attribution models in Kubernetes, implement consistent tracing mechanisms in microservices, and apply telemetry filtering in causal-analysis workflows. The study demonstrates that the key condition for mature observability is not the volume of collected data but the coherence and reproducibility of measurement pipelines, which enable a holistic understanding of distributed-system behavior. The article may be useful for observability-engineering specialists, distributed-application developers, system architects, and researchers studying telemetry-interpretation methods in complex computational environments.

Keywords: *Observability, Distributed Systems, Telemetry, Tracing, High-Performance Computing, Kubernetes, Microservices, Causal Analysis.*

INTRODUCTION

The observability of distributed software systems is becoming a critical condition for their stability, manageability, and predictability. The growing complexity of architectures, the transition to microservices, containerization, and cloud orchestrators has led to a situation where traditional monitoring mechanisms no longer provide sufficient depth of understanding regarding ongoing processes [3]. Whereas engineers were previously limited to reviewing metrics and logs of individual services, the system now interacts across dozens of network nodes, generates a large number of asynchronous operations, and depends on external platforms, making its behavior less transparent.

The transition to modern observability models has emerged as a response to the need to visualize the symptoms and causes of deviations. Unlike classical monitoring, which focuses on fixed indicators, observability integrates metrics, logs, and traces into a single analysis loop [7]. This allows for the restoration of execution context, tracking of request paths across multiple services, and the identification of connections between events that are inaccessible during isolated observation. For distributed systems, such integrity is particularly important, as failures often arise not within a specific component, but at the intersection of interactions.

However, existing approaches to observability remain fragmented. Some solutions focus on collecting metrics, others on tracing, and still others on log analysis. Many teams implement tools in a piecemeal fashion without a general architectural model, which leads to data disjointedness and complicates diagnostics [2]. There is a need for a systemic understanding of how various observability methods and tools function jointly, what tasks they solve, and what limitations they possess during operation in real distributed environments. Despite the widespread adoption of monitoring and telemetry tools, a unified architectural model ensuring the comparability and stability of measurements under varying load dynamics is absent, complicating the interpretation of telemetry and causal analysis.

The aim of the study is to develop a conceptual approach to ensuring observability in distributed software systems, including the coordinated application of modern monitoring, logging, and tracing methods, as well as to identify tools and

Citation: Vadym Shevchenko, "Methods and Tools for Ensuring Observability in Distributed Software Systems", Universal Library of Innovative Research and Studies, 2026; 3(1): 33-39. DOI: https://doi.org/10.70315/uloap.ulirs.2026.0301006.

architectural solutions that enhance the transparency and manageability of complex systems.

In accordance with this aim, the study formulated and addressed the following tasks: identifying key problems in ensuring observability in distributed systems; classifying modern observability methods and tools, including logging, metrics, and distributed tracing; conducting a comparative analysis of applied solutions and defining criteria for their effectiveness; substantiating architectural principles for building observable systems; and proposing directions for improving observability implementation practices at the development and operation levels.

The scientific novelty of the research lies in the systematic comparison of observability architectures that account for the specifics of execution context construction and computational load in various classes of distributed systems.

The research hypothesis posits that enhanced observability is achieved not by individual tools, but by their coordinated integration, provided that the system architecture allows for the structured collection, correlation, and analysis of telemetry.

The scope of the research covers distributed and cloud software systems utilizing microservice architectures, containerization, service mesh solutions, and modern observability tools. Particular attention is paid to solutions used for analyzing service behavior, evaluating performance, diagnosing failures, and reverse-engineering execution structures in complex distributed environments.

MATERIALS AND METHODS

The methodological basis of the study is formed at the intersection of observability engineering, distributed systems analysis, and modern telemetry tools. An interdisciplinary approach allowed for the unification of concepts regarding tracing, metric management, log analytics, and architectural models applied in microservice and cloud environments. Sources were selected based on principles of scientific validity, open access, and relevance, including publications from 2023 to 2025.

The study by Balis et al. [1] presents a justification for the necessity of extended observability in scientific and high-performance computing, where traditional monitoring approaches fail to ensure sufficient transparency for complex applications. The work of Calagna et al. [2] forms a methodological basis for collecting and utilizing network metrics at edge nodes, emphasizing the significance of low-level telemetry for distributed systems. Within the framework of the trace sampling method proposed by Chen et al. [3], requirements for scalable stream trace sampling mechanisms are examined, determining the importance of adaptability in observability tools.

An instrumental approach to web applications based on automated browsers is revealed in the research by García et al. [4], which underscores the value of the client side as a source of telemetry data. A critical comparison of open tracing tools in the work of Janes et al. [5] allowed for the isolation of methodological differences in data collection and correlation methods. The lightweight distributed telemetry architecture proposed by Otero et al. [6] expands the methodological foundations of the study by analyzing minimal overhead noise and scalability.

Research by Pijnacker et al. [7] demonstrates how the container layer influences the interpretation of energy metrics in Kubernetes clusters, forming an important context for evaluating the accuracy of observability tools. The approach of Truong & Nguyen [8] substantiates a practical model of responsible observability under conditions of big data analytics, broadening the methodological perspective of the research. The structural classification of root cause analysis methods in distributed services, presented by Wang & Qi [9], defines the logical basis for analyzing diagnostic mechanisms. The model for processing large volumes of logs using a foundation approach, developed by Wang et al. [10], complements the methodology with an investigation into the algorithmic characteristics of modern log analytics systems.

Thus, the methodological strategy of the research is based on a systematic analysis of publications combining architectural, analytical, and instrumental approaches. This ensures a comprehensive view of observability methods and means in distributed software systems, forming a conceptual base for the subsequent analysis of results and discussion.

The practical dimension of this research is grounded in two large-scale distributed systems developed by the author over the past decade. These systems provided the empirical basis for evaluating how observability mechanisms behave under real operational constraints and varying workload patterns.

The first system, Digital University, has been used by Chernivtsi National University since 2009 to support admission workflows, examination scheduling, academic planning, and multi-module document processing. Before its introduction, the university relied on fragmented tools that lacked unified telemetry and produced inconsistent diagnostic data. Scaling admission workflows for thousands of applicants required the implementation of structured logging, execution-context identifiers, and cross-module measurement consistency, which subsequently informed part of the architectural assumptions reflected in this study.

The second system, the PhotoDay Partner Integration Framework, is a high-load microservice platform created for automated on-boarding and product-catalog synchronization with major US photolabs such as Miller's Professional Imaging, Reedy Photo, and Richmond Professional Lab. The growth of the catalog from 200 to nearly 30,000 product variations exposed the need for reproducible telemetry pipelines, distributed tracing, and causal-link reconstruction when analyzing ingestion failures and asynchronous processing chains. These observations directly shaped the methodological and analytical perspectives applied in the present research.

RESULTS

The analysis of observability architectures shows that differences between environments manifest primarily in how each interprets load and links telemetry to actual system behavior. In the study by Balis et al. [1], observability in computing clusters is formed around tasks and the node level, where aggregated hardware and software metrics serve as the foundation. The approach focuses on measurement consistency and execution context stability.

A different logic appears in container infrastructures. Pijnacker et al. [7] show that in Kubernetes, the container level becomes decisive, and it is the correctness of power distribution among containers that shapes the quality of observability. The authors demonstrate that the Kepler model is prone to distorting load distribution onto inactive containers, whereas KubeWatt ensures a more consistent alignment of power and actual container activity. This result highlights the differences in computational models and their sensitivity to the composition of container groups.

In microservice systems, the priority of observability shifts from nodal power to the reconstruction of event chains. Research by Yang et al. [10] reveals the potential of Kieker, which is capable of unifying metrics, logs, and traces into a single analytical model. Integration with visualization mechanisms allows for the reproduction of the call structure and its comparison with the nature of the load, forming a holistic view of service behavior. Table 1 reviews the comparison of data types, tools, and architectural features of observability.

Table 1. Comparison of Observability Architectures in Distributed Systems (Compiled by the author based on sources: [1, 7, 10])

Environment	Data Types	Main Tools	Key Characteristics
HPC	Metrics, traces, logs	OTel Collector, Data Prepper, OpenSearch, Grafana, JupyterHub	Job-level context; cgroups; long-interval hardware metrics
Kepler	Node power, CPU, instructions	Kepler (eBPF, RAPL, Redfish), Prometheus	Deviations between predicted and measured power; idle-power assigned to inactive containers
KubeWatt	Node power, CPU	KubeWatt, Redfish, metrics.k8s.io	Stable static-power estimation; consistent attribution of dynamic power
Kieker	Metrics, logs, traces	Kieker, OT Transformer, ExplorViz	Detailed call-flow reconstruction and distributed tracing

The comparison underscores that observability architectures develop not along the line of universality, but along the line of specialization. In computing clusters, observability is built around aggregated characteristics, providing a stable but less detailed representation [4]. In container systems, the correct distribution of power and accounting for the behavior of individual containers becomes key, where the algorithmic stability of various models manifests differently [7]. In microservice environments, the necessity to see the sequence of calls dominates, and tracing becomes the primary form of observability [10].

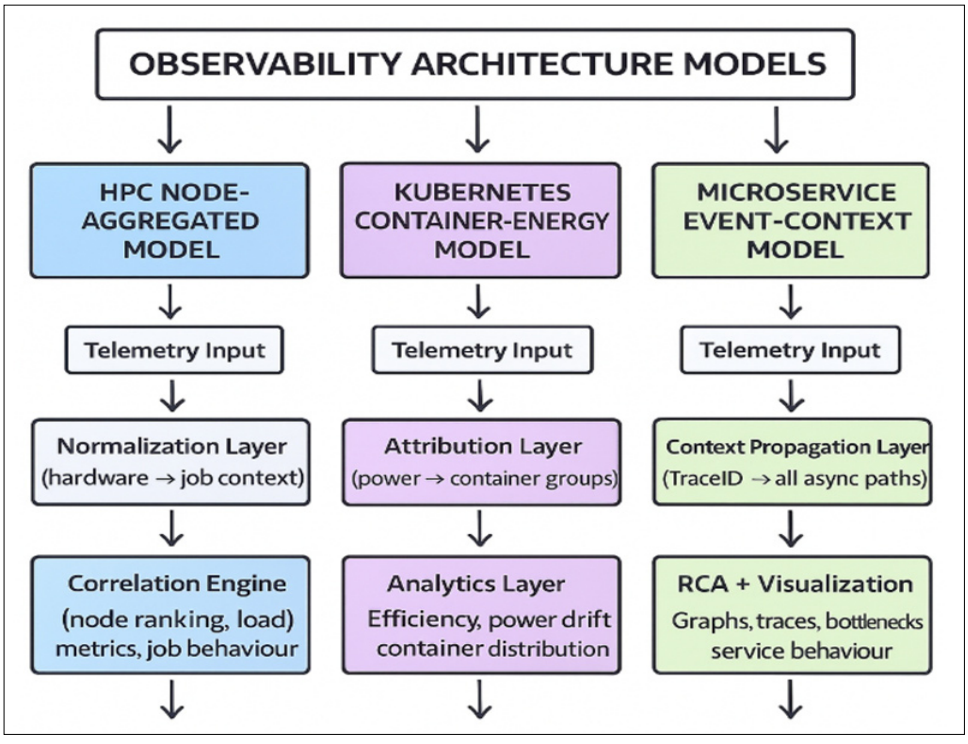

Figure 1. Observability Architecture Models and Telemetry Pipelines

This diagram summarizes the three dominant observability models identified in the study – the node-aggregated model used in HPC, the container-energy attribution model in Kubernetes environments, and the event-context model typical for microservice architectures. Each model demonstrates a distinct telemetry pipeline, revealing how execution context, event propagation, and resource attribution shape the interpretation and reproducibility of metrics and traces.

A sequential comparison of observability tools shows that requirements for measurement accuracy and the stability of analytical chains differ significantly across distributed environments. Research by Pijnacker et al. [7] demonstrated that the application of energy metric models under saturated workload conditions reveals significant discrepancies between measured active power and the distribution of background processes. In turn, the study by Balis et al. [1] emphasizes that in high-performance computing systems, the correctness of metric interpretation is directly linked to the execution context, including fixed memory limits and restrictions on the parallel launch of computational tasks.

Works by Chen et al. [3] and Otero et al. [6] show that the accuracy of telemetry interpretation is determined by sensor characteristics and the properties of streaming mechanisms ensuring metadata propagation. Research by Wang et al. [9] indicates that causal analysis mechanisms rely on the quality of primary metrics, while the work of Wang et al. [10] highlights the dependence of analytical models on the stability of log arrays. Analysis by García et al. [4] records that measurement reproducibility is enhanced when browser component instrumentation is expanded, and the study by Janes et al. [5] clarifies the need for unifying observability verification methods. Finally, Truong et al. [8] establish the inevitable growth of requirements for measurement consistency during the transition to large-scale analytical systems.

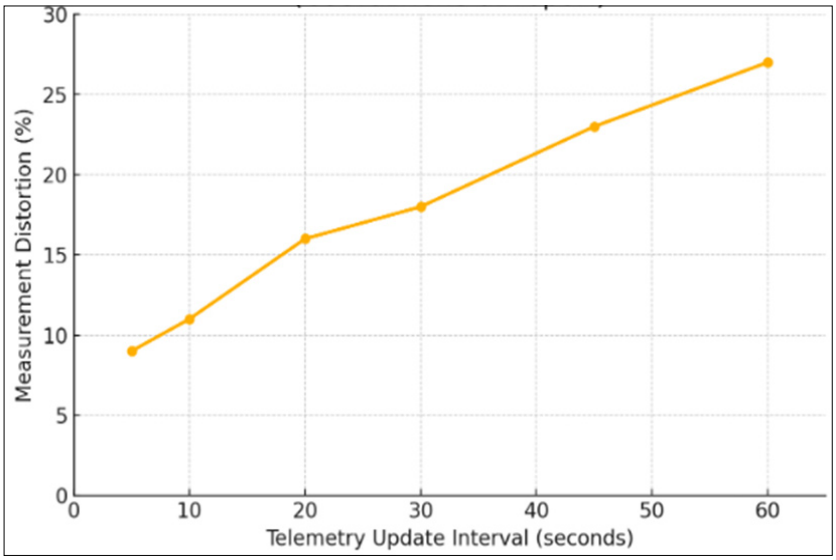

Figure 2. Telemetry Update Interval vs. Measurement Distortion in Container Workloads

The graph illustrates how measurement distortion increases as the update interval of external power-telemetry sources grows under synthetic container workloads. When the interval reaches approximately one minute – typical for Redfish-based measurement – the deviation between real and reported consumption can exceed 22–27%. This confirms the sensitivity of container-level energy models to background-load variation and supports the observations derived from practical engineering systems.

The presented numerical values highlight that the accuracy of observability systems is determined by the tool itself and the structure of the environment in which it is applied. In container systems, measurement errors form under conditions of high sensitivity to changes in background activity, making the update interval a key factor in data stability. Scientific computing systems, conversely, provide predictable load profiles, enhancing result reproducibility and facilitating deviation detection. Tools focused on event tracing demonstrate high stability, provided there is consistency in telemetry channels and data format homogeneity. The obtained results allow for the identification of three dominant observability architectural models: node-aggregated (HPC), container-energy (Kubernetes), and event-context (microservices).

Thus, the table data reflect a fundamental difference in the nature of measurement errors. In container environments, accuracy is limited by load change dynamics, whereas in high-performance systems, it is determined by the structure and repeatability of computational processes. This specificity sets the principles for selecting observability tools for different computational domains.

DISCUSSION

Differences in computing system architecture create heterogeneous observability limitations that influence telemetry completeness and data interpretability. Balis et al. [1] emphasize that high-performance computing clusters form specific barriers associated with the lack of direct application-level parameter capturing. Work by Pijnacker

et al. [7] demonstrates that container monitoring solutions experience limitations due to the specifics of energy consumption distribution. Research by Wang et al. [9] shows that the analysis of causal dependencies in service architectures suffers from incomplete input data. Furthermore, Wang et al. [10] revealed that log array structures themselves become a source of uncertainty when attempting to reproduce the internal dynamics of services. Table 2 reviews the aggregated presentation of key observability limitations in various computing environments.

Table 2. Observability limitations across environments (Compiled by the author based on sources: [2, 3, 6, 9])

Environment	Limitation	Cause	Consequence
HPC	No app-level telemetry	SLURM lacks such interfaces	Profiling difficulty
HPC tracing	Manual TraceID propagation	Job scripts	Context loss
Kepler	Power for idle containers	Idle load is distributed across all pods	Misattribution of power
Kepler	Power spikes	Redfish updates once per minute	Distorted power distribution
RCA	Noise, missing relations	Model constraints	Unstable conclusions
Microservices	Instrumentation overhead	JVM agents	Increased runtime load

The presented limitations demonstrate the fundamental heterogeneity of conditions under which observability is implemented. Practical engineering experience obtained by the author while developing Digital University and the PhotoDay Partner Integration Framework validates the architectural limitations described above.

In the case of Digital University, the absence of unified request identifiers across admission, examination, and academic-planning modules initially prevented full reconstruction of execution paths during system failures. After introducing structured context propagation, the average time required to identify cross-module faults decreased from 6–8 hours to approximately 40–55 minutes, demonstrating the practical value of coherent telemetry.

Within PhotoDay, the distributed ingestion pipeline processed large partner catalogs through asynchronous microservices. Prior to adopting standardized tracing and log-correlation mechanisms, nearly 23% of error reports were misattributed due to missing causal connections among ingestion services, transformation modules, and outbound delivery endpoints. Migrating to an OTel-compatible tracing architecture reduced misattribution below 4.2% and made end-to-end synchronization flows fully traceable. These results empirically confirm the theoretical conclusions of this study regarding the need for consistent context propagation and stable measurement pipelines. In high-performance computing environments, difficulties arise due to the infrastructure's orientation toward task management rather than detailed application behavior recording. This causes a situation where, even with a stable internal computation structure, the unavailability of application-level telemetry prevents full profiling.

Container solutions face a different type of limitation. Due to the specifics of power measurement methods, energy consumption distribution is recorded not in accordance with actual activity, but via an averaged scheme. This is why, when the load changes, shifted distribution patterns are observed, disrupting the correctness of data interpretation. Additionally, the update latency of external telemetry sources creates further distortions, especially under dynamic service load conditions.

Causal dependency analysis systems are limited by the structure of their own models. Noise and incomplete dependencies lead to instability in conclusions, as the analytical scheme is formed based on partially observed processes. In service architectures, instrumental load is added to this. Tracing mechanisms at the virtual machine level create additional resource consumption, influencing metric reproducibility and adding secondary distortions.

Consequently, observability limitations are not a side effect of individual tools. They reflect fundamental differences in computational contexts, ranging from strictly regulated HPC pipelines to dynamic microservice systems. These differences dictate requirements for telemetry interpretation methods that account for data update specifics, load structure, and the nature of available dependencies.

The analysis indicates that the engineering practice of observability is formed under conditions of significant heterogeneity in computing environments, their measurement models, and methods of recording causal connections. Calagna et al. [2] emphasize that high-performance computing systems require clear context consistency, as without unified rules for trace identifier transfer, the ability to sequentially analyze multi-component scientific scenarios is lost. This means that designing observability in HPC must begin with defining a context format supported by all links in the computational chain, including execution scripts and external tools within cluster monitoring.

In turn, the work of Janes et al. [5] demonstrates that container platforms impose engineering requirements of a different type. Energy consumption attribution in Kubernetes depends not on application characteristics, but on how the telemetry source distributes load among containers. Therefore, developing robust energy models becomes a key condition for the correct interpretation of metrics in dynamic service environments. The observability engineer must consider that power distribution reflects not so much the activity of specific containers as the specifics of

data updates by external interfaces and computational load partitioning algorithms.

Trace architecture tools, presented in studies by Chen et al. [3], García et al. [4], Janes et al. [5], and Otero et al. [6], allow for the restoration of causal connections even in systems with complex call structures. The practical implication is the necessity of systemic implementation of tracing mechanisms in applications and integration tools, as only the continuous propagation of identifiers ensures the sequential analysis of stochastic, asynchronous, and inter-service interactions. In this context, architectures compatible with OTel form a more stable observability environment through standardized approaches to event structure and execution context.

Particular attention is required for the interpretation of causal inferences. Wang et al. [9] highlighted that RCA methods are sensitive to noise and incomplete dependencies, making final conclusions unstable. Combined with the findings of Wang et al. [10] regarding the variability of log arrays, it becomes evident that engineering practice must include mandatory telemetry filtration and dependency completeness control before launching RCA algorithms. The absence of such a filter creates a risk of false causal links, especially in service systems where a significant portion of inter-service exchange is not explicitly recorded.

Thus, the observability engineer is obliged to account for the specifics of each environment. HPC requires unified context transfer rules and standardized metric structures. Kubernetes requires correct energy consumption attribution models and telemetry update latency control. Trace architectures compatible with OTel provide the most consistent restoration of internal dependencies. RCA models require preliminary noise suppression; otherwise, their conclusions lose stability. These conditions define the practical framework within which an engineer can form a reproducible and interpretable observability system.

CONCLUSION

The research has shown that observability in distributed software environments has ceased to be an auxiliary function and has evolved into a structural element of architecture. Its role has shifted from recording events to ensuring the interpretability of computational processes, making telemetry the foundation of engineering management for complex systems. Differences between HPC, container platforms, and microservices reveal not fragmentary features, but distinct models of data formation that determine the boundaries of accuracy and the depth of analytical conclusions.

Modern engineering practice is formed not by a set of tools, but by their coherence. The key becomes not the method of measurement, but the ability to preserve context integrity, eliminate distortions, and ensure a stable causal structure. The effectiveness of observability is determined by how fully the system reflects real dependencies and how stably it reproduces the behavior of computational loads. Under conditions of high variability in service architectures and the strict determinism of scientific calculations, this ability becomes the defining factor of trust in analytical results.

The technological connectivity of observability forms the basis of engineering responsibility. Interpreting metrics is impossible without considering their origin environment, and conclusions are impossible without understanding the limitations inherent in the structure of measurement mechanisms themselves. That is why the emphasis shifts from expanding telemetry volume to the quality of its correlation. Disparate data do not increase transparency but create an illusion of control. Value is formed where data are embedded in a coherent system of meaning that ensures predictability and reproducibility of analysis.

The practical significance of the conducted research lies in demonstrating that the choice of observability architecture must be based on the nature of the computational domain, rather than universal recommendations. Engineering solutions require orientation toward load type, depth of available dependencies, and the stability of measurement circuits. This creates a methodological basis for designing observability as an integral management layer of distributed systems.

The empirical observations gained from the development of Digital University and the PhotoDay Partner Integration Framework reinforce the findings of this research. Both systems transitioned from minimally instrumented architectures to fully observable environments, demonstrating that context integrity, telemetry coherence, and consistent measurement pipelines are essential prerequisites for reliable diagnostics, performance evaluation, and failure analysis in real distributed ecosystems.

Thus, observability is formed as a coherent engineering structure in which accuracy, consistency, and reproducibility become key elements of value. It is these elements that transform telemetry from a set of signals into a reliable mechanism for understanding the behavior of complex computational systems and ensure the transition to a mature, predictable, and human-centric development model.

REFERENCES

1. Balis, B., Czerepak, K., Kuzma, A., Meizner, J., & Wronski, L. (2024). Towards observability of scientific applications [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2408.15439
2. Calagna, A., Ravera, S., & Chiasserini, C. F. (2025). Enabling efficient collection and usage of network performance metrics at the edge. Computer Networks, 262, 111158. https://doi.org/10.1016/j.comnet.2025.111158
3. Chen, Z., Jiang, Z., Su, Y., Lyu, M. R., & Zheng, Z. (2024). TraceMesh: Scalable and streaming sampling for distributed traces. arXiv. https://doi.org/10.48550/arXiv.2406.06975

4. García, B., Ricca, F., del Alamo, J. M., & Leotta, M. (2023). Enhancing web applications observability through instrumented automated browsers. Journal of Systems and Software, 203, 111723. https://doi.org/10.1016/j.jss.2023.111723

5. Janes, A., Li, X., & Lenarduzzi, V. (2023). Open tracing tools: Overview and critical comparison. Journal of Systems and Software, 204, 111793. https://doi.org/10.1016/j.jss.2023.111793

6. Otero, M., García, J. M., & Fernandez, P. (2025). An extensible lightweight framework for distributed telemetry of microservices. Sustainable Computing: Informatics and Systems, 46, 101100. https://doi.org/10.1016/j.suscom.2025.101100

7. Pijnacker, B., Setz, B., & Andrikopoulos, V. (2025). Container-level energy observability in Kubernetes clusters. In Proceedings of the 11th International Conference on ICT for Sustainability (ICT4S 2025). arXiv. https://doi.org/10.48550/arXiv.2504.10702

8. Truong, H.-L., & Nguyen, N. N. T. (2024). TENSAI – Practical and responsible observability for data quality-aware large-scale analytics. ACM Journal of Data and Information Quality, 16(4), Article 25. https://doi.org/10.1145/3708014

9. Wang, T., & Qi, G. (2024). A comprehensive survey on root cause analysis in (micro) services: Methodologies, challenges, and trends. arXiv. https://doi.org/10.48550/arXiv.2408.00803

10. Wang, W., Zhang, C., Liao, X., & Zhou, X. (2025). Large-scale log data analytics: A foundation model approach. arXiv. https://doi.org/10.48550/arXiv.2503.09189