The American Journal of Applied Sciences
ISSN 2689-0992 Volume 08 - 2026

Design and Deployment of a Cloud-Native Event-Driven Enterprise Banking Platform Using Java Microservices and Kafka-Based Real-Time Transaction Processing

Dr. Liang Wei
Department of Computer Science and Artificial Intelligence TAJAS-13-21-Design and Deployment
Tsinghua Institute of Advanced Computing
Beijing, China

Prof. Chen Xiaoyu
School of Cloud Computing and Intelligent Systems
Shanghai International University of Technology
Shanghai, China

Received: 01 May 2026 | Received Revised Version: 15 May 2026 | Accepted: 24 May 2026 | Published: 01 Jun 2026
Volume 08 Issue 06 2026 |

ABSTRACT

The modernization of enterprise banking systems has become a strategic requirement as financial institutions increasingly depend on high-availability, low-latency, and secure digital services. Traditional monolithic banking applications often struggle to support real-time transaction processing, elastic scalability, fault isolation, and rapid feature delivery. This research paper proposes a cloud-native, event-driven enterprise banking platform designed using Java microservices, Kafka-based asynchronous communication, secure API gateways, automated DevOps pipelines, and infrastructure-as-code practices. The study develops a conceptual and technical architecture for transforming legacy banking systems into distributed real-time ecosystems capable of handling payments, account services, fraud monitoring, notification workflows, audit logging, and regulatory reporting.

The proposed model is grounded in prior research on serverless and cloud-native computing, function composition, platform performance, fraud detection, anomaly identification, and distributed enterprise security. Existing literature highlights the advantages of cloud programming abstraction, scalable service deployment, and event-based processing, while also revealing unresolved challenges related to latency, observability, false positives in financial monitoring, and operational complexity (Jonas et al., 2019; Baldini et al., 2017; Castro et al., 2019; Wang et al., 2018). In response, this paper presents an integrated architecture that combines domain-driven microservice decomposition, Kafka event streams, Java-based service orchestration, CI/CD automation, containerized deployment, and real-time analytics pipelines.

The findings indicate that event-driven microservices can improve transaction responsiveness, reduce dependency coupling, and enhance system resilience when compared with tightly integrated legacy architectures. Kafka enables durable, scalable, and replayable event processing, while automated deployment pipelines reduce release friction and support continuous modernization. However, the model also introduces governance challenges involving event schema management, distributed tracing, data consistency, security enforcement, and operational monitoring. The study concludes that cloud-native event-driven banking platforms offer significant potential for mission-critical financial modernization, provided that performance engineering, fraud controls, DevSecOps, and compliance-by-design principles are embedded into the platform architecture.

Keywords: Cloud-native banking, Java microservices, Kafka, event-driven architecture, real-time transaction processing, DevOps, legacy modernization, financial fraud detection, distributed systems, secure APIs.

Cite This Article: Dr. Liang Wei, & Prof. Chen Xiaoyu. (2026). Design and Deployment of a Cloud-Native Event-Driven Enterprise Banking Platform Using Java Microservices and Kafka-Based Real-Time Transaction Processing. The American Journal of Applied Sciences, 8(06), 13–21. Retrieved from https://www.theamericanjournals.com/index.php/tajas/article/view/8034

1. Introduction

The Banking institutions operate in one of the most demanding enterprise technology environments. Core banking platforms must process high volumes of transactions, maintain strict accuracy, protect sensitive financial information, support regulatory auditability, and remain available even during periods of extreme load. Historically, many banks have depended on large monolithic systems designed around centralized databases, batch processing, and tightly coupled business modules. Although these systems provide stability for traditional operations, they often limit digital transformation because changes in one component can affect the entire application, deployment cycles are slow, and scaling is usually coarse-grained rather than service-specific.

The rise of cloud-native computing has changed expectations for enterprise software design. Modern digital banking requires real-time payment confirmation, instant fraud detection, personalized customer notifications, secure API-based ecosystem integration, and continuous availability across mobile, web, branch, and partner channels. Cloud-native approaches provide a foundation for these requirements by promoting distributed service design, automated deployment, elastic infrastructure, and observability. Research on serverless and cloud programming has shown that cloud platforms can simplify deployment, reduce infrastructure management overhead, and enable more flexible resource consumption (Jonas et al., 2019; Baldini et al., 2017). However, banking systems cannot rely only on abstraction; they must also satisfy strict consistency, latency, security, and compliance constraints.

The central problem addressed in this paper is how mission-critical legacy banking systems can be transformed into cloud-native, event-driven platforms without compromising reliability, transaction integrity, or operational governance. Traditional modernization projects often focus on migrating applications to cloud infrastructure without redesigning the underlying architecture. Such “lift-and-shift” strategies may reduce infrastructure burden but rarely solve deeper issues such as service coupling, batch dependency, deployment rigidity, and limited real-time visibility. Therefore, modernization must be architectural rather than merely infrastructural.

Event-driven architecture provides a suitable model for banking modernization because many banking operations naturally occur as events. Examples include account creation, fund transfer initiation, payment authorization, card usage, fraud alert generation, loan status change, and compliance report submission. When these events are published to a distributed messaging platform such as Kafka, downstream services can consume and process them asynchronously. This reduces direct dependency between services and allows different business capabilities to evolve independently. Prior work on serverless function composition and distributed cloud platforms highlights the importance of scalable event handling, service coordination, and performance-aware design (Castro et al., 2019; McGrath and Brenner, 2017; Wang et al., 2018).

Java microservices remain highly relevant in enterprise banking because of their maturity, performance, ecosystem support, and suitability for large-scale transactional systems. A Java-based microservice architecture can encapsulate banking domains such as accounts, payments, cards, customers, fraud monitoring, notifications, and audit services. Each service can own its data model, expose secure APIs, and publish domain events to Kafka. This model supports gradual legacy transformation because individual modules can be extracted from the monolith and rebuilt as independent services without requiring full system replacement.

Security and fraud monitoring are equally central to banking modernization. Studies on financial fraud detection demonstrate the growing importance of machine learning, anomaly detection, evidence fusion, feature selection, and process mining in identifying suspicious transactions (Darwish, 2020; Eshghi and Kargari, 2019; Ravisankar et al., 2011; Baader and Krcmar, 2018). These studies show that fraud control must be integrated into operational workflows rather than added as an external afterthought. In an event-driven banking platform, fraud detection services can consume transaction events in near real time, evaluate risk, and publish fraud decision events to downstream services.

The objective of this research is to design and analyze a cloud-native enterprise banking platform based on Java microservices, Kafka event streams, secure APIs, DevOps automation, and performance engineering. The paper contributes a structured architecture for real-time transaction processing, explains its functional components, evaluates expected benefits, and identifies implementation limitations. The scope of the study is conceptual and architectural, focusing on platform design rather than empirical deployment in a specific bank. Its significance lies in providing a research-driven technical framework for financial institutions seeking to modernize legacy banking systems into scalable, secure, and real-time digital ecosystems.

2. Literature Review

The literature on cloud-native and serverless computing provides a strong foundation for understanding distributed platform modernization. Baldini et al. (2017) discuss serverless computing as an evolving model that abstracts infrastructure management and allows developers to focus on application logic. Although serverless platforms are not always directly suitable for every core banking workload, the principles of elasticity, automation, and function-level scalability are highly relevant to cloud-native banking. Jonas et al. (2019) similarly argue that cloud programming can simplify application development by shifting infrastructure concerns to managed platforms. For banking modernization, this suggests that platform engineering should reduce operational complexity while still preserving control over security, performance, and compliance.

McGrath and Brenner (2017) examine the design, implementation, and performance aspects of serverless computing, highlighting that performance variability remains an important concern. This is particularly relevant for banking because payment authorization, fraud evaluation, and customer account operations require predictable response times. Wang et al. (2018) further show that serverless platforms may hide internal performance behaviors, making observability and benchmarking critical. These findings indicate that cloud-native banking systems must not depend blindly on platform abstraction; instead, they require explicit performance engineering, monitoring, and workload profiling.

Function composition is another important issue in distributed architectures. Castro et al. (2019) introduce the serverless trilemma, showing that composition among functions can create trade-offs between performance, cost, and programming simplicity. In enterprise banking, similar trade-offs appear when payment services, account services, fraud engines, notification modules, and audit systems interact through distributed workflows. A poorly designed service chain may increase latency and failure propagation. Therefore, event-driven architecture must be designed with clear boundaries, compensating workflows, and observable event flows.

Research by Spillner, Mateos, and Monge (2017) emphasizes the potential of serverless computing for scientific workloads, while Spillner and Sbarski (2017) discuss staging environments for serverless applications. These studies are relevant because they show the need for controlled testing and deployment environments before cloud applications are released into production. In banking, staging environments are essential for validating transaction workflows, API contracts, event schemas, fraud rules, and infrastructure changes. Leitner et al. (2019) provide an industrial perspective on function-as-a-service development and show that practical adoption involves organizational, tooling, and operational challenges. This supports the argument that banking modernization must include DevOps maturity, not only architectural redesign.

The fraud detection literature contributes another important dimension. Ravisankar et al. (2011) demonstrate the use of data mining and feature selection for financial statement fraud detection, showing that financial risk identification depends heavily on selecting meaningful indicators. Seeja and Zareapoor (2014) propose a credit card fraud detection model based on frequent itemset mining, while Kiran et al. (2018) compare Naïve Bayes and KNN-based credit card fraud detection. Darwish (2020) introduces semantic fusion of classifiers, suggesting that combining multiple decision models may improve fraud detection accuracy. Eshghi and Kargari (2019) similarly focus on fusion of fraud evidence under uncertainty. These studies support the need for real-time analytical pipelines in banking platforms.

False positives are a major concern in fraud systems. Baader and Krcmar (2018) show that combining red flag approaches with process mining can reduce false positives. This is highly relevant to event-driven banking because excessive false positives may block legitimate transactions and degrade customer trust. Gonzalez-

Carrasco et al. (2019) investigate automatic detection of relationships between banking operations using machine learning, indicating that fraud and risk detection should consider transactional relationships rather than isolated events. Zamini and Hasheminejad (2019) provide a broader survey of anomaly detection across banking, healthcare, social networks, and wireless sensor networks, reinforcing the value of anomaly detection in complex distributed environments.

Security literature also informs the proposed model. Babu and Radhika (2023) study DDoS attack detection using swarm-optimized random forest classification, while Vullam et al. (2023) and Vellela et al. (2024) explore ensemble-based intrusion detection approaches. These works indicate that enterprise platforms must integrate intelligent security monitoring into their architecture. Since cloud-native banking platforms expose APIs, event streams, and distributed services, they require continuous threat detection and automated response mechanisms.

The literature reveals three key gaps. First, cloud-native and serverless studies often discuss scalability and abstraction but do not fully address banking-specific transaction integrity and compliance needs. Second, fraud detection studies focus on models and classifiers but rarely integrate them into end-to-end event-driven banking architecture. Third, security studies address intrusion and anomaly detection but often remain separate from enterprise modernization frameworks. This paper positions itself at the intersection of these areas by proposing a Java microservices and Kafka-based platform that integrates transaction processing, DevOps automation, fraud detection, security monitoring, and legacy modernization.

3. Methodology

This research follows a design-oriented architectural methodology. Instead of conducting a controlled laboratory experiment, it develops a conceptual enterprise platform model based on synthesis of the provided literature and practical requirements of mission-critical banking systems. The method consists of four major stages: requirement analysis, architectural decomposition, event-driven workflow design, and operational governance modeling.

The first stage identifies functional and non-functional requirements. Functional requirements include account management, payment processing, card transaction handling, customer notification, fraud scoring, audit logging, and reporting. Non-functional requirements include scalability, low latency, high availability, fault isolation, secure communication, observability, automated deployment, and regulatory traceability. These requirements are derived from the nature of banking operations and supported by literature on cloud platforms, performance challenges, and fraud detection (McGrath and Brenner, 2017; Wang et al., 2018; Darwish, 2020).

The second stage decomposes the enterprise banking system into domain-based Java microservices. A proposed platform includes an Account Service, Payment Service, Customer Service, Card Service, Fraud Detection Service, Notification Service, Audit Service, API Gateway, Identity Service, and Reporting Service. Each service owns its bounded context and data responsibility. For example, the Payment Service handles transaction initiation and status updates, while the Account Service manages balances and account state. The Fraud Detection Service does not directly control transactions but consumes transaction events and publishes risk decisions. This design reduces tight coupling and supports independent development.

The third stage defines the Kafka-based event architecture. Kafka topics are organized around domain events such as payment.initiated, payment.validated, account.debited, account.credited, fraud.alert.generated, notification.sent, and audit.record.created. Each event contains metadata such as event identifier, timestamp, source service, correlation ID, customer reference, transaction amount, risk score, and processing status. Kafka provides durable event storage and replay capability, which is important for audit reconstruction and failure recovery. Event replay allows services to rebuild state or reprocess transactions when downstream systems fail.

A hypothetical transaction flow illustrates the model. When a customer initiates a fund transfer through mobile banking, the API Gateway authenticates the request and forwards it to the Payment Service. The Payment Service validates basic transaction rules and publishes a payment.initiated event. The Account Service consumes the event and checks available balance. The Fraud Detection Service consumes the same event and evaluates risk using rule-based and machine-learning models. If the fraud score is acceptable, the Payment Service completes the transaction after receiving necessary status events. The Notification Service then

sends confirmation, while the Audit Service records the full event trail. This asynchronous model allows parallel processing and reduces direct service dependency.

The fourth stage develops the DevOps and deployment model. Each Java microservice is containerized and deployed through automated CI/CD pipelines. The pipeline includes code validation, unit testing, integration testing, static security scanning, container image building, deployment to staging, contract testing, and production release. Infrastructure-as-code defines compute clusters, network policies, Kafka brokers, database instances, secrets, observability tools, and API gateway rules. This aligns with prior findings that cloud-native adoption requires strong tooling, staging environments, and industrial development practices (Spillner and Sbarski, 2017; Leitner et al., 2019).

Security is embedded through multiple layers. The API Gateway enforces authentication, authorization, throttling, and request validation. Internal service-to-service communication uses encrypted channels and token-based access control. Kafka topics are protected through access control lists, encryption, and schema validation. Audit logs are immutable and linked with transaction correlation IDs. Security monitoring consumes access logs, network events, and abnormal service behavior to detect possible attacks. Ensemble-based intrusion detection research supports the value of intelligent monitoring for distributed platforms (Vullam et al., 2023; Vellela et al., 2024).

Performance engineering is treated as a core methodological component. The platform is designed to scale services independently according to workload. Payment and account services may require low-latency synchronous validation for critical steps, while notification and reporting services can operate asynchronously. Kafka partitions allow horizontal scaling of consumers, but partitioning must be designed carefully to preserve transaction ordering where required. For example, events related to the same account should be routed consistently to maintain ordered processing.

Fraud detection is integrated as an event consumer rather than an external batch system. The Fraud Detection Service uses transactional attributes, behavioral patterns, historical risk indicators, and relationship-based features. Studies on classifier fusion, frequent itemset mining, and anomaly detection support the need for multi-model fraud evaluation rather than single-rule detection (Seeja and Zareapoor, 2014; Eshghi and Kargari, 2019; Zamini and Hasheminejad, 2019). The model also accounts for false positives by allowing fraud decisions to include confidence scores, manual review flags, and process-mining-based contextual validation (Baader and Krcmar, 2018).

The methodology includes governance for event schemas and data consistency. Since microservices publish and consume events independently, schema evolution must be controlled. A schema registry can enforce compatibility rules so that changes in one service do not break downstream consumers. Data consistency is managed through eventual consistency and compensating transactions. For example, if debit succeeds but credit fails, a reversal event can be issued. This approach avoids distributed locking but requires careful workflow design.

Observability is included through centralized logging, metrics, tracing, and business-level monitoring. Technical metrics include request latency, Kafka lag, error rates, CPU usage, memory consumption, and database response time. Business metrics include transaction success rate, failed payment ratio, fraud alert rate, average processing time, and manual review volume. These metrics help detect both technical failures and business anomalies.

The final methodological output is an integrated reference architecture for a cloud-native banking platform. It combines Java microservices, Kafka event streaming, secure API access, DevOps automation, fraud analytics, intrusion monitoring, observability, and governance mechanisms. The model is intended for gradual migration from legacy systems, where new services can be introduced around existing core banking platforms before full decomposition is completed.

4. Results

The proposed architecture demonstrates that cloud-native event-driven design can significantly improve the structural flexibility of enterprise banking systems. The primary finding is that Kafka-based asynchronous processing reduces direct dependency among banking services. Instead of requiring every service to call another service synchronously, business events are published once and consumed by multiple downstream services. This improves extensibility because new capabilities such as fraud analytics, customer notification, compliance monitoring, or reporting can be

added without changing the original transaction service.

A second finding is that Java microservices provide a practical modernization path for legacy banking systems. Rather than replacing an entire monolithic platform at once, banks can incrementally extract domains such as payments, customer profiles, cards, or alerts into independently deployable services. This supports controlled transformation and reduces migration risk. The approach is consistent with cloud-native principles of modularity, elasticity, and service independence discussed in prior cloud computing studies (Baldini et al., 2017; Jonas et al., 2019).

A third finding concerns real-time fraud detection. Event streams allow fraud services to evaluate transaction behavior as soon as a payment is initiated. This enables near real-time risk scoring compared with batch-oriented fraud analysis. Literature on classifier fusion, anomaly detection, and financial fraud mining indicates that fraud detection improves when multiple indicators and models are combined (Darwish, 2020; Eshghi and Kargari, 2019; Ravisankar et al., 2011). The proposed architecture supports this by allowing fraud engines to consume transaction, account, customer, and behavioral events.

The architecture also improves auditability. Since each business action produces an event with metadata, correlation ID, timestamp, and source service, the complete transaction journey can be reconstructed. This is valuable for dispute resolution, compliance investigation, and operational debugging. Kafka's replay capability further supports recovery and forensic analysis.

However, the findings also show important limitations. Event-driven systems increase architectural complexity. Developers must manage eventual consistency, duplicate events, schema evolution, message ordering, and failure handling. Performance is not automatically guaranteed by cloud adoption; latency must be continuously measured, especially for payment workflows where customer experience depends on immediate confirmation (McGrath and Brenner, 2017; Wang et al., 2018). The platform therefore requires mature DevOps, observability, and governance practices.

Overall, the results indicate that a cloud-native event-driven banking platform can improve scalability, resilience, and real-time processing, but only when supported by disciplined engineering controls.

5. Discussion

The proposed platform reflects a shift from application-centric banking architecture to event-centric enterprise architecture. In traditional systems, business processes are embedded inside tightly coupled application modules. In the proposed model, business state changes are represented as events that can be processed, audited, analyzed, and extended. This transformation has theoretical significance because it treats banking operations as distributed event flows rather than isolated database transactions.

The main practical implication is improved adaptability. Banks frequently need to introduce new payment channels, fraud rules, compliance checks, and customer communication methods. In a monolithic system, each change may require modifications across multiple modules. In an event-driven microservice platform, new consumers can subscribe to relevant Kafka topics and add functionality with lower disruption. This aligns with the broader cloud-native argument that modular services support faster innovation and operational flexibility (Baldini et al., 2017; Leitner et al., 2019).

The platform also strengthens fraud and security integration. Prior studies show that fraud detection benefits from anomaly detection, evidence fusion, process mining, and relationship analysis (Baader and Krcmar, 2018; Gonzalez-Carrasco et al., 2019; Zamini and Hasheminejad, 2019). By embedding fraud detection into event streams, the architecture enables continuous monitoring rather than delayed batch evaluation. Similarly, intrusion detection methods can be connected to service logs and network events to identify abnormal behavior (Babu and Radhika, 2023; Vullam et al., 2023).

A key trade-off is between responsiveness and consistency. Banking systems require accurate balances and transaction integrity, but distributed microservices often rely on eventual consistency. The architecture must therefore distinguish between operations that require immediate synchronous confirmation and operations that can be processed asynchronously. For example, balance validation may require strong consistency, while notifications and reporting can tolerate delay. This hybrid design is more realistic than a purely asynchronous model.

Another limitation is operational complexity. Kafka clusters, schema registries, service meshes, CI/CD pipelines, monitoring tools, and security policies require specialized engineering maturity. Without governance, microservices can become fragmented and difficult to

manage. This concern reflects the serverless and cloud literature, which shows that abstraction can simplify deployment but also hide performance and operational issues (Wang et al., 2018; Castro et al., 2019).

The discussion therefore suggests that cloud-native banking modernization is not merely a technology migration. It is an organizational and architectural transformation requiring domain modeling, platform engineering, DevSecOps, observability, and risk governance. The proposed model is most effective when implemented incrementally with strong controls over event design, security, and performance.

6. Conclusion

This research paper presented a cloud-native event-driven enterprise banking platform using Java microservices and Kafka-based real-time transaction processing. The study addressed the modernization challenge faced by legacy banking systems that struggle with scalability, agility, real-time responsiveness, and operational resilience. By decomposing banking functions into domain-aligned Java microservices and connecting them through Kafka event streams, the proposed architecture enables asynchronous processing, independent service scaling, improved auditability, and real-time fraud monitoring.

The paper's main contribution is an integrated architectural model that combines cloud-native deployment, event-driven communication, DevOps automation, secure API design, fraud analytics, intrusion monitoring, and observability. Unlike modernization strategies that focus only on cloud migration, this model emphasizes architectural transformation. It shows how payments, accounts, fraud detection, notifications, audit logging, and reporting can function as interoperable services within a distributed financial ecosystem.

The study also highlights that cloud-native banking modernization involves significant limitations. Event-driven systems require careful handling of consistency, schema evolution, duplicate events, message ordering, latency, and monitoring. Security must be embedded at every layer, from API gateways and service communication to Kafka topics and deployment pipelines. Fraud detection must balance sensitivity with false-positive reduction to avoid disrupting legitimate transactions.

Future research may extend this work through empirical benchmarking of Kafka-based banking workloads, simulation of transaction failure scenarios, comparison of synchronous and asynchronous payment workflows, and evaluation of machine-learning-based fraud detection within live event streams. Further studies may also examine regulatory compliance automation, privacy-preserving analytics, and AI-assisted observability in cloud-native banking ecosystems. Overall, the proposed architecture provides a strong foundation for transforming mission-critical banking systems into scalable, secure, and real-time enterprise platforms.

References

1. G. Baader and H. Krcmar, "Reducing false positives in fraud detection: Combining the red flag approach with process mining," International Journal of Accounting Information Systems, 2018.
2. I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski et al., "Serverless computing: Current trends and open problems," in Research advances in cloud computing. Springer, 2017, pp. 1–20.
3. R. S. Babu and K. Radhika, "DDoS Attack Detection using Swarm Optimized Random Forest Classification," International Journal of Intelligent Systems and Applications in Engineering, vol. 12, pp. 231–238, 2023.
4. P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The serverless trilemma: Function composition for serverless computing," in Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. ACM, 2019, pp. 276–289.
5. S. M. Darwish, "An intelligent credit card fraud detection approach based on semantic fusion of two classifiers," Soft Computing, vol. 24, no. 2, pp. 1243–1253, Jan. 2020.
6. A. Eshghi and M. Kargari, "Introducing a new method for the fusion of fraud evidence in banking transactions with regards touncertainty," Expert Systems with Applications, vol. 121, pp. 382–392, May 2019.
7. Gonzalez-Carrasco, J. L. Jimenez-Marquez, J. L. Lopez-Cuadrado, and B. Ruiz-Mezcua, "Automatic detection of relationships between banking operations using machine learning," Information Sciences, vol. 485, pp. 319–346, Jun. 2019.
8. S. Hossain, A. Abtahee, I. Kashem, M. M. Hoque, and I. H. Sarker, "Crime Prediction Using Spatio-

Temporal Data," in Computing Science, Communication and Security, Singapore : Springer, 2020, pp. 277–289.

9. E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar et al., "Cloud programming simplified: A berkeley view on serverless computing," arXiv preprint arXiv: 1902.03383, 2019.
10. S. Kiran, J. Guru, R. Kumar, N. Kumar, D. Katariya, and M. Sharma, "Credit card fraud detection using Naïve Bayes model based and KNN classifier," International Journal of Advance Research, Ideas and Innovations in Technology, vol. 4, pp. 44–47, 2018.
11. Krushnasamy VS, Vellela SS, Kadu RK, Reddy CSP, Vinay TV, Vahalingam R. A Machine Learning-Based Prediction Model for Postoperative Delirium in Cardiac Valve Surgery Using Electronic Health Records. J Rare Cardiovasc Dis. 2025 ; 3 (S1): 84–102.
12. P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of function-as-a-service software development in industrial practice," Journal of Systems and Software, vol. 149, pp. 340–359, 2019.
13. G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). IEEE, 2017, pp. 405–410.
14. M. Pohoretskyi, D. Serhieieva, and Z. Toporetska, "The proof of the event of a financial resources fraud in the banking sector: problematic issues," Financial and Credit Activity-Problems of Theory and Practice, vol. 1, no. 28, pp. 36–45, 2019.
15. Polasi, P. K., Vellela, S. S., Narayana, J. L., Simon, J., Kapileswar, N., Prabu, R. T., & Rashed, A. N. Z. (2024). Data rates transmission, operation performance speed and figure of merit signature for various quadurature light sources under spectral and thermal effects. Journal of Optics, 1–11.
16. Ram, C. S., Vellela, S. S., Sravanthi Javvadi, D. V., Rashid, S. Z., & Madhumathi, S. M. (2025). Integrated Robotic–Imaging Platforms in Endovascular Surgery: Current Capabilities and Future Directions. Vascular and Endovascular Review, 8 (16s), 285–298.
17. P. Ravisankar, V. Ravi, G. Raghava Rao, and I. Bose, "Detection of financial statement fraud and feature selection using data mining techniques," Decision Support Systems, vol. 50, no. 2, pp. 491–500, 2011.
18. R. S. B. Rakki, "AI in Financial Fraud Detection: Machine Learning Techniques," Journal Publication of International Research for Engineering & Management, vol. 5, no. 4, pp. 1–6, 2025.
19. K. Seeja and M. Zareapoor, "FraudMiner: A Novel Credit Card Fraud Detection Model Based on Frequent Itemset Mining," The Scientific World Journal, pp. 1–10, 2014.
20. J. Spillner, C. Mateos, and D. A. Monge, "Faaster, better, cheaper: The prospect of serverless scientific computing and hpc," in High Performance Computing: 4th Latin American Conference, CARLA 2017. Springer, 2017, pp. 154–168.
21. J. Spillner and P. Sbarski, "Towards a staging environment for serverless applications," in Proceedings of the Workshop on Serverless Computing. ACM, 2017, pp. 1–6.
22. S. Phani Praveen, Sai Srinivas Vellela, Kanhaiya Sharma, Lavanya Dalavai, "Quantitative Evaluation of Smart Textile Adoption in Rural Weaving Communities using Machine Learning ", Journal of the Textile Association, 86 / 3 (Sept Oct ' 2025), 277–284, https://doi.org/10.5281/zenodo.17597818
23. C. Tyagi, P. Parwekar, P. Singh, and K. Natla, "Analysis of Credit Card Fraud Detection Techniques," Solid State Technology, vol. 63, no. 6, pp. 18057–18069, 2020.
24. Vellela, S. S., Purimetla, N. R., Rao, P. V., Daniel, V. A. A., Koppolu, H. K. R., & Janani, B. (2025). AI-Enabled Wearable Hemodynamic Monitoring System for Early Identification of Thrombotic Events. Vascular and Endovascular Review, 8 (16s), 321–336.
25. Vellela, S. S., Vullum, N. R., Thommandru, R., Rao, T. S., Sowjanya, C., & Kumar, K. K. (2024, May). Improving Network Security Using Intelligent Ensemble Techniques: An Integrated System for Detecting and Managing Intrusions in Computer Networks. In 2024 International Conference on Advances in Modern Age Technologies for Health and Engineering Science (AMATHE) (pp. 1–7). IEEE.
26. Vullam, N., Roja, D., Rao, N., Vellela, S. S., Vuyyuru, L. R., & Kumar, K. K. (2023, December). An Enhancing Network Security: A Stacked Ensemble Intrusion Detection System for Effective Threat Mitigation. In 2023 3rd International

Conference on Innovative Mechanisms for Industry Applications (ICIMIA) (pp. 1314–1321). IEEE.

27. L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in 2018 USENIX Annual Technical Conference (USENIX ATC 18), 2018, pp. 133–146.

28. M. Zamini and S. M. H. Hasheminejad, "A comprehensive survey of anomaly detection in banking, wireless sensor networks, social networks, and healthcare," Intelligent Decision Technologies-Netherlands, vol. 13, no. 2, pp. 229–270, 2019.