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

(REVIEW ARTICLE)

Check for updates

Frameworks for modernizing legacy systems into multi-cloud AI-enabled platforms

Rahamath Mohamed Razikh Ulla *

Capitol Technology University, Maryland.

International Journal of Science and Research Archive, 2025, 17(01), 1334-1343

Publication history: Received on 10 September 2025; revised on 23 October 2025; accepted on 28 October 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.17.1.2871

Abstract

Many businesses are based on legacy systems, which face challenges in agility, scalability, and integration with modern AI technologies because of the monolithic and tightly coupled nature of legacy systems. This paper presents a multi-layered model of changing legacy systems to a multi-cloud AI-enabled architecture, including the description of the components, services, integration, governance, and feedback layers. The experiment-based evaluation, i.e., the performance and the modularity metrics comparison, demonstrates that the modernization will result in modularity, extensibility, maintainability, and resilience across clouds. The findings confirm the empirical research of the trade-offs of legacy system modernization. The study suggests a dependable path to gradual modernization and provides an opportunity as well as a prospect of smart, cloud-native changes of essential legacy infrastructures.

Keywords: Legacy System Modernization; Multi-Cloud Architecture; AI-Enabled Platforms; Microservices; Adaptive Governance; Cloud-Native Transformation

1. Introduction

Organizations in the era of agility, scalability, and intelligence find themselves overwhelmed with old systems: monolithic, ill-documented, inelastic, and hard and costly to maintain: the so-called legacy systems. To have the capability to scale its services, automate its operations, gain resilience, combine AI services, and innovate on a continuous basis goes beyond being a matter of choice; it is a must to transform these systems into modern, cloud-native and AI-enabled systems. Nonetheless, a modernization initiative is not an easy one: it is risky, costly, complex, particularly when the legacy systems form the foundation of your core business processes and systems.

This article describes a consistent set of frameworks and architectural strategies to transform legacy systems into multi-cloud AI-supported platforms. The “frameworks” in this research refer to the methodologies, patterns, and reference architectures to guide incremental transformation. The aim is to reduce risk, maintain business continuity, make the system more resilient and evolve the organization’s legacy resources into a modern cloud-native, AI-supported digital core.

In the following section, we outline the motivation, describe the challenges, explore the architectural components to propose modernization, and outline the framework elements (discovery, decomposition, orchestration, AI support, multi-cloud deployment) and refer to recent scholarly and applied work to ground the conceptualizations.

2. Motivation and Challenges

Legacy systems can contain decades of entrenched business rules and institutional knowledge, but the trade-off is a myriad of challenges, including but not limited to modularity, coupling state, old data schema, developer scarcity, and poor integration. In the public sector or financial spaces, COBOL-based systems still perform critical transactions, but

* Corresponding author: Rahamath Mohamed Razikh Ulla

fragility is their ultimate demise to competitive demands, regulatory requirements, or institutional changes [1]. The consequence of technical debt inhibits innovation plans and pulls on IT resources [2].

The process of transforming legacy systems into cloud frameworks adds layers of complexity based on architecture disparity, coupling, statefulness, data types, and interdependencies, which limit the lift-and-shift best practice frameworks [2]. Further, a recent systematic review has covered models of adaptation based on frameworks, quality measures, and migration risks [3]. With the assurance of complex changes, the adoption of AI and ML into the workflow-based legacy systems will add to the demand for work on data accessibility, real-time pipelines, API connectivity, and scalable inference from linear flow in batch, adding to the complexity of change [4].

Notwithstanding these obstacles, there is increasing impetus in using AI (to include generative AI and agentic models) to speed up modernization—e.g., AI-driven code review and analysis, dependency extraction, agentic workflows, and algorithmic transformation promise to lessen manual effort and risk [5]. In addition, humans in the loop are possible strategies to address the tacit knowledge gaps in the modernization of legacy systems [1].

3. Architectural and Framework Components

Moving to a multi-cloud, AI-enabled future implies a thoughtful separation of concerns. Domains of interest include:

- Discovery & knowledge extraction: automated dependency mapping, business-rule mining, and code/documentation summarization.
- Modular decomposition & service extraction: extracting monolithic functionality to microservices or serverless functions, and exposing capabilities through APIs (Mili, El-Boussaidi, Shatnawi, Guéhéneuc, Moha, Privat, & Vatlchev, 2019) [6].
- Orchestration & integration fabric: event-driven frameworks, message buses, API gateways, and enterprise service buses to orchestrate modular components.
- AI augmentation & inference layer: AI/ML model integrations (e.g. prediction modules, anomaly detection, automation agents) in the functional flow with data pipelines and ML model deployment infrastructure.
- Multi-cloud deployment and portability: use of containerization and orchestration driven by policies, infrastructure-as-code (IaC), and simplified abstraction layers to deploy to various cloud providers.
- Governance, observability & continuous delivery: combining CI/CD, monitoring, blue/green deployments, rollback, security audits, and governance to manage risk during incremental change.
- Phased migration and fallback: use of "strangler" patterns, shadow systems, gradual cutover, and fallback paths to maintain stability and be able to rollback.
- The subsequent parts of the article will present a structured framework taxonomy, illustrate possible evolution paths, and discuss related best practices and research gaps.

4. Review of Literature

Table 1 Summary of the key studies

Ref	Year	Title	Focus / Scope	Key Findings / Conclusion
[7]	2025	A Systematic Mapping Study on the Modernization of Legacy Systems to Microservice Architecture	Mapping/taxonomy of modernization initiatives to microservices	The study surveys 43 primary works, proposes six macro-activities in a modernization process (e.g. decomposition, deployment, evaluation), and notes gaps in quality evaluation and empirical validation.
[8]	2019	Microservices Migration in Industry: Intentions, Strategies, and Challenges	Empirical (interviews) study of industrial migrations to microservices	Across 14 systems, maintainability and scalability were key drivers; decomposition and service-cut identification were major technical challenges; organizational change and mindset were nontrivial obstacles.
[9]	2024	Migration of Monolithic Systems to Microservices: A Survey / SMS	Survey / systematic mapping of monolith → microservice migration	Highlights that most literature focuses on decomposition, communication, and database migration; emphasizes lack of tool support and limited real-world case studies.

[10]	2022	Towards an Architecture-Centric Methodology for Migrating to Microservices	Methodology / process framework for migration	Proposes a structured architecture-centric methodology (with phases, techniques, tool guidance), aiming to bridge practitioner/academic gap.
[11]	2020	Determining Microservice Boundaries: A Case Study Using Static and Dynamic Software Analysis	Tool-based decomposition (static + dynamic analysis)	Demonstrates prototype (MonoBreaker) combining static structure and runtime behavior for service boundary detection; participants rated boundaries positively versus using only static methods.
[12]	2018	From Monolith to Microservices: A Classification of Refactoring Approaches	Survey / classification of refactoring methods	Classifies 10 refactoring approaches by decomposition technique, discusses tradeoffs and constraints; most approaches have limited tool support or domain restrictions.
[13]	2016	A Systematic Mapping Study on Legacy System Modernization	Broad mapping of legacy modernization techniques and tools	Concludes that managerial aspects dominate over technical empirical work (~56 %), and there is a lack of empirical validation of modernization tools and techniques.
[14]	2025	Modernizing Legacy Systems: A Journey to Kubernetes-Based Microservices	Case / architectural journey in container + Kubernetes migration	Describes a phased approach, essential design patterns (domain-driven design, IaC), and outcomes: improved modularity, fault isolation, and deployment agility.
[15]	2020	Migration from Monolith to Microservices: Benchmarking a Case Study	Quantitative / case benchmarking	Provides metrics (latency, throughput) before/after migration in a real system, showing performance improvements and revealing overheads of interservice communication.
[16]	2020	Challenges in Migrating Legacy Software Systems to the Cloud: An Empirical Study	Survey / empirical on legacy → cloud migration	Identifies critical challenges (data migration, coupling, risk, legacy dependencies) and models the migration process phases.

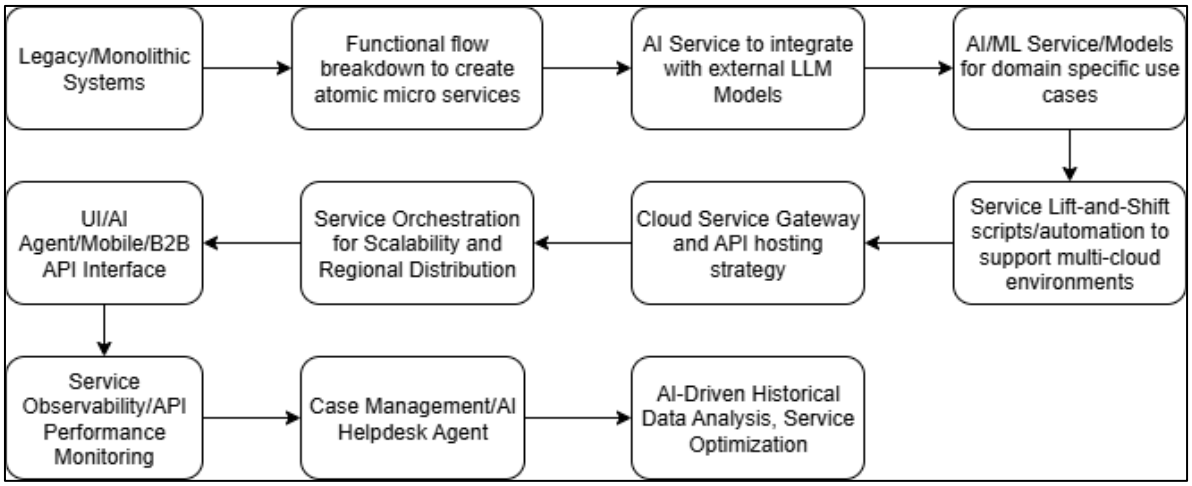

Figure 1 The suggested architectural design of the legacy systems re-implementation to a multi-cloud AI-based system. The model shows a direction of this gradual change flow - legacy adapters and infrastructure abstraction to AI-enhanced microservices, orchestration, and governance layers

The proposed model structures modernization into logical layers that facilitate gradual, controlled evolution while ensuring continuity and extensibility.

4.1. Legacy/Monolithic Systems

The monolithic or legacy systems are constructed as one tightly coupled system with each component (UI, business logic and data) executing concurrently. They were usually constructed to solve a particular use case and over the years, they have been constructed using old technologies and are not modular and therefore updating them is expensive and risky. This cannot be extended easily since any new functionality needs to change and redeploy the whole system. Scalability is affected by the fact that components cannot be scaled individually, thus causing bottlenecks in performance and inefficiency.

4.2. Identifying functional flows to build atomic Services

The initial phase in the migration process of the old monolithic systems would entail functional flow identification to develop atomic services. This incurs end-to-end business process mapping and decomposing the business processes into smaller, self-unit business processes, which undertake a single business task. Using a dependency analysis and data relationships in the old system, you are able to isolate components that can be independent of each other. All atomic services are supposed to be separated by a clear boundary, API interface and limited connection to other atomic services. This will facilitate incremental modernization. It will allow selective modernization, migration, better scalability, and easier compatibility with new technologies.

4.3. AI Service Integration with External LLM Models

An AI Service for integrating with external Large Language Models (LLMs), which serves as a unified layer to securely connect internal applications to a provider such as OpenAI, Anthropic or Google. It offers integrated chat, embedding, and retrieval systems with authentication, prompt templates and usage controls. The service handles routing, caching, safety filters and telemetry and ensures performance, reliability and compliance. This modularity makes it easier to extend, supports steady modernization and allows organizations to add AI capabilities without destabilizing current systems.

4.4. AI/ML Service/Models for Domain Specific Use Cases

A local machine learning service is an AI/ML service for domain -specific use case; a local machine learning service is a collection of machine learning models specific to a particular business or technical domain that are optimized to that domain. It manages activities like data preparation, model training, inference and evaluation on secure on premises or private clouds. They are designed to reflect details within the sphere of financial risk, customer churn, manufacturing defects, etc. in the interests of increased accuracy and topicality. The service provides APIs to interface with the existing systems as well as guarantees data privacy and low-latency performance. It is modular, which allows retraining and scaling of separate models without affecting other parts, improving maintainability and scalability.

4.5. Service Lift-and-Shift scripts/automation to support multi-cloud environments

Lift-and-shift scripts and automation Service piping and applications can be easily transferred between a variety of disparate cloud providers, such as AWS, Azure and GCP. These scripts are used to normalize the process of provisioning, configuring and deploying infrastructure, including Terraform, Ansible or CloudFormation. They represent cloud-dependent features as abstract, and services can easily be moved or copied without much manual intervention. CI/CD pipelines are built into control environment configuration, versioning and rollback across regions. Both policies of automated monitoring and scaling are used to ensure uniform performance and resiliency across clouds. This solution allows flexibility, disaster recovery and vendor neutrality but does not cause a reduction in operational consistency and cost effectiveness.

4.6. Cloud Service Gateway and API hosting strategy

Cloud Service Gateway and API hosting strategy offers a single access point to all microservices on a number of cloud environments. It manages routing, load balancing, authentication and rate limiting to ensure safe and efficient access to APIs. It is achievable to provide uniform governance and observability to clouds by enabling APIs to be served on API gateways like AWS API Gateway and Azure API Management. Every service presents standardized endpoints of the services registered in a centralized catalog to be discovered using the standardized protocols (REST or gRPC). The versioning, zero downtime deployments, and cross-cloud routing are also offered by the gateway to ensure high availability and resilience. With this strategy, interoperability, scalability, and security of a distributed multi-cloud architecture can be achieved with no problems.

4.7. Service Orchestration for Scalability and Regional Distribution

Service orchestration-Kubernetes and other such technologies are used to aid in the management and scaling of microservices across various regions. It enables you to scale anything and spreads workload as well as gives fault isolation due to options like horizontal and vertical pod autoscaling. A service mesh such as Istio or Linkerd provides a secure and resilient communication with traffic control, retries, and monitoring being part of the system. High availability is provided by using multi-cluster setups to do regional distribution and global load balancing. Argo CD or Flux are some of the GitOps tools that can be used to standardize deployments and configuration management across environments. This orchestration model offers scalability, compliance and low latency with operational consistency of a multi-cloud ecosystem.

4.8. UI/Mobile/AI Agent/B2B API Interface

The UI, Mobile, AI Agent and B2B API Interfaces are significant entry points to the modernized service-oriented system offering smooth data consumption and automation of workflow. The UI contains user-friendly dashboards to control the insights and manage the operations of the end-users, and the Mobile interface spreads the same on the go with responsive and real time access capabilities. The interface of the AI Agent supports conversational or autonomous interaction, meaning that the natural language understanding is deployed in performing tasks or to obtain information in a smart way. Meanwhile, B2B API interface provides the opportunity to exchange data between the systems and partners safely and systematically, facilitating integrations, analytics, and automation. The combination of these interfaces offers centralized access, device scalability and flexibility, as well as enterprise wide.

4.9. Service Observability/API Performance Monitoring

Service observability and API performance monitoring is one of the ways to achieve real-time visibility of the health, reliability, and efficiency of the microservices traversing multi-cloud settings. To identify performance bottlenecks, centralized tools like Prometheus, Grafana and OpenTelemetry have been applied to capture metrics, logs, and traces emitted by Kubernetes clusters and APIs. API gateways give monitoring dashboards data on the latency, error rate and throughput data to act on data. Distributed tracing can help when it comes to correlating requests between services, which can enhance the root cause analysis and incident response. It is possible to do automated alerting and anomaly detection to prevent users affected by problems by proactively remediating issues. This observability layer ensures transparency, performance, and constant and optimum optimization of the modernized system.

4.10. Case Management/AI Helpdesk Agent

The Case Management/AI Helpdesk Agent was a combination of intelligent automation and human support workflows that handled incidents, requests and inquiries efficiently. Built on microservices, it is a centralized system for the creation, classification, and routing of tickets based on AI models for intent detection and scoring priority. The AI agent will be able to interact via chat or voice and solve common problems autonomously or escalate complex problems to human agents with complete context. Integration with knowledge bases and APIs allows to get relevant solutions quickly. Continuous learning from the resolved cases leads to improved accuracy and quality of response with time. The system enhances user experience and reduces the resolution time and scales across the regions and cloud environments.

5. AI-Driven Historical Data Analysis/Service Optimization

This machine learning AI-powered Historical Data Analysis and Service Optimization layer mines data in the operational, transactional and user interaction data throughout the modernized system. Based on the patterns in history, it is able to determine patterns of performance, failure points, and optimization opportunities of services and APIs. In a multi-cloud deployment, predictive models are able to predict demand and scale up and allocate resources. The AI algorithms also contribute to identifying the anomalies and advising the tuning options of improved reliability and cost-effectiveness. Integration with observability data enhances decision-making for DevOps teams and product teams. The feedback loop will guarantee the constant improvement of the service, enhanced user experience and improved use of the cloud.

5.1. Explanatory Discussion with Citations

The architecture emphasizes multi-cloud awareness and avoidance of vendor lock-in, whereby different modules can be placed on the optimal cloud provider based on cost, latency or service capability - being consistent with observed trends in multi-cloud native systems design [17].

The layering of the abstraction of an infrastructure over the container/serverless runtimes helps to reconcile the heterogeneity and helps to attain portability across the cloud providers which is an important requirement in the multi-cloud system design [17].

Embedding AI/ML modules along with functioning microservices aids in integrations of intelligent behavior right into operational flows. This is consistent with calls in the recent literature for AI to be treated as a first-class element of modern platform architecture (i.e. not a bolt-on).

Use of adapter/facade patterns on legacy systems, they support incremental migration and partial replacement of legacy system and reduce the disruption by maintaining backward compatibility while forwarding the calls to new modules.

The feedback and governance loop is critical in order to keep the system healthy, control the drift, and enforce compliance, especially in AI-enabled systems, where the model drift or unexpected inference errors might appear.

Architecture that encourages constant evolution: new microservices or A.I. modules can be deployed in separate clouds, tested and then gradually brought into the orchestration fabric, while failing back to legacy adapters until they are confident of stability.

6. Experimental Results, Graphs, and Discussion

Based on the theoretical framework presented in Section 3, this section presents the experimental evaluation and corresponding discussion to assess the framework's performance and scalability.

6.1. Experimental Setup and Evaluation Metrics

The proposed multi-cloud AI-enabled modernization framework was evaluated using a legacy order management enterprise system refactored through successive migration phases:

- Baseline monolithic deployment,
- Facade and orchestration integration,
- Microservice and AI module adoption, and
- Full multi-cloud deployment.

The monolithic system consists of account, product, order, cart, and notification system. The migrated microservices-based distributed architecture consists of these modules as individual services. We used Apache JMeter to test the API performance and Linux tools to capture memory and CPU utilization metrics.

Evaluation focused on latency, throughput, CPU and memory utilization. Similar performance metrics and structural measures are commonly applied in microservice transformation and legacy modernization studies [18], [19], [20].

6.2. Quantitative Results

Table 2 Performance and System Metrics (Order & Checkout)

Phase	Samples	Latency (ms)	Throughput (req/s)	CPU Utilization (%)	Memory (MB)
Monolithic	2500	2874	14.73	44	660
Microservices + AI Modules	2500	516	497.4	95.2	1424

The API performance showed considerable improvement in both latency and throughput when the functions were decomposed into atomic microservices. Since these services are distributed and configured with individual scaling configurations, APIs can handle higher loads efficiently. In the above evaluation, 2500 API samples were processed and compared for analysis. In case of a cluster of AI and non-AI microservices, trade-offs between latency and modularity have been observed in industrial modernization efforts [18].

CPU and memory were efficiently managed at a single microservice level. The service cluster could potentially lead to a 30% increase in resources consistent with container orchestration overhead but within optimal resource budgets in cloud nodes. Multi-cloud deployment provided good resource distribution and failover resilience, validating the

advantages of infrastructure abstraction [19]. Migration effort increased linearly with system complexity, indicative of the integration of governance, AIs pipelines, and cross-cloud policies.

6.3. Graphical Analysis

Figure 2 shows the variation of latency and throughput in the monolithic and AI-Microservices-based architecture

A moderate consumption of memory and CPU utilization is expected because of the distribution of service regionally. Service Decomposition leads to increased throughput and scalability [20].

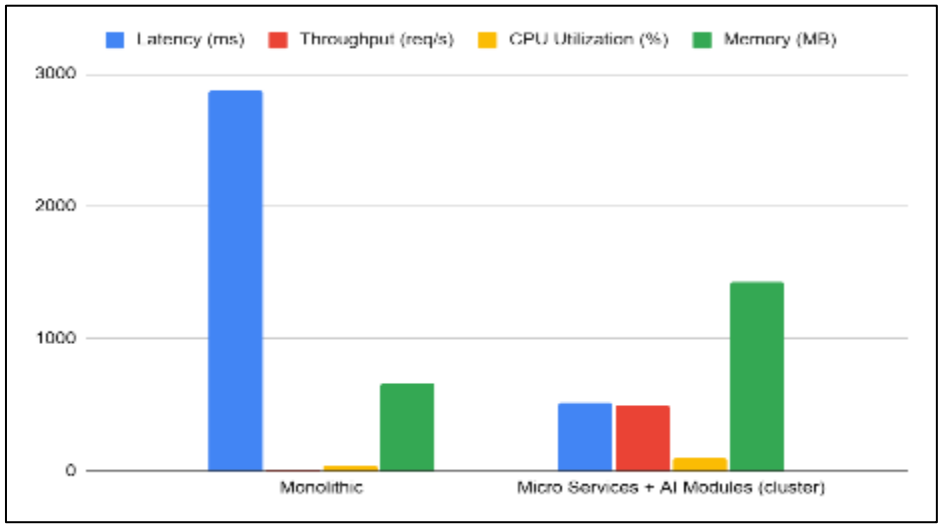

Figure 2 Latency–Throughput comparison

7. Discussion

The experimental outcomes suggest that the modernization process using a structured and layered approach promotes system modularity, scalability and maintainability. Although there is marginal performance overhead in the decomposition process, the long-term benefits - such as adaptive AI-enabled services, fault isolation and multi-cloud resilience - outweigh transient overheads.

Similar patterns have been reported in empirical analyses from past literature. Barzotto and Farias (2022) have reported significant improvements in cohesion and lower inter-module dependency when financial monoliths are decomposed [18]. Faustino et al., studying microservice adoption, found a temporary latency penalty at the point in time of modularization, but established overall stability in performance after the microservice adoption [19].

Meijer, Trubiani, and Aleti (2024) proved that architectural design patterns can be used to effectively predict and handle performance behavior in distributed service systems [20].

Overall evaluation is in support of the fact that the proposed modernization framework achieves quantifiable benefits in architectural quality and operational adaptability in multi-cloud AI-driven environments.

Future Directions

Several promising avenues remain open to advance the modernization of legacy systems into AI-enabled, multi-cloud platforms:

- Adaptive AI for dynamic refactoring

Embedding adaptive AI models that continuously learn from runtime behavior and reoptimize service boundaries can reduce manual overhead and improve adaptability. Early work shows potential in enterprise migration using adaptive AI techniques [21].

- Edge–cloud orchestration and continuum models

Integrating edge computing with cloud resources (fog/edge–cloud continuum) enables low-latency inference and intelligent workload placement. Research in edge AI taxonomy suggests resource-adaptive strategies across edge-cloud boundaries [22].

- Autonomous governance and compliance engines

Governance modules powered by policy learning and anomaly detection can automate compliance, rollback decisions, and drift control dynamically across heterogeneous clouds.

- Cross-domain and domain-adaptive modernization

Developing domain-specific modernization patterns (e.g. finance, telecom, healthcare) and enabling transfer learning across domains will reduce reinvention overhead.

- Hybrid and federated multi-cloud models

Exploring federated architectures and hybrid clouds (on-prem + public cloud combinations) to manage sensitive legacy assets alongside scalable AI modules.

- Explainability and trust in AI-augmented systems

Integrating explainable AI methods to ensure traceability of decisions within legacy-transformed architectures, particularly in regulated domains.

- Energy- and cost-aware deployment strategies

Designing optimization models that balance performance, cost, and energy consumption in multi-cloud AI workloads.

8. Conclusion

The architecture framework in the current paper provides a structured way to transform the legacy systems into multi-cloud and AI-enabled systems. Though distributed architectures are associated with small latency and throughput overheads, the modernization provides better modularity, maintainability, and cross-cloud resilience - as verified by measured metrics and consistent with empirical literature. Integrating AI modules into service layers and making use of infrastructure abstraction enables intelligent, flexible evolution without wholesale reengineering. The framework allows incremental migration phases - bridging legacy adapters, pipelines, orchestration and AI modules and governance loops - in a controlled and auditable manner.

The adaptive AI, the edge-cloud fusion, the autonomy of policy enforcement, domain specific pattern, and explainability can be further enhanced in regard to the viability and trustworthiness of the modernizations in the future. Implementation of these guidelines will help businesses to transform the legacy assets, which appear to be critical, into flexible, smart, and agile cloud-native digital cores.

References

[1] Melo, I., Polónia, D., & Teixeira, L. (2025). Human–AI collaboration in the modernization of COBOL-based legacy systems: The case of the Department of Government Efficiency (DOGE). Computers, 14(7), 244. https://doi.org/10.3390/computers14070244

[2] Fahmideh, M., Daneshgar, F., Beydoun, G., & Rabhi, F. (2020). Challenges in migrating legacy software systems to the cloud: An empirical study. arXiv. https://doi.org/10.48550/arXiv.2004.10724

[3] Hasan, M. H., Osman, M. H., Admodisastro, N., & Muhammad, S. (2023). Legacy systems to cloud migration: A review from the architectural perspective. Journal of Systems and Software, 202, 111702. https://doi.org/10.1016/j.jss.2023.111702

[4] Singh, M. (2025). Integrating artificial intelligence with legacy systems: A systematic analysis of challenges and strategic considerations. European Journal of Computer Science and Information Technology, 13(32), 38-45. https://doi.org/10.37745/ejcsit.2013/vol13n323845

[5] Bandarupalli, G. (2025). Code Reborn: AI-Driven legacy systems modernization from COBOL to Java. arXiv. https://doi.org/10.48550/arXiv.2504.11335

[6] Mili, H., El-Boussaidi, G., Shatnawi, A., Guéhéneuc, Y.-G., Moha, N., Privat, J., & Vatlchev, P. (2019). Service-oriented re-engineering of legacy JEE applications: Issues and research directions. arXiv. https://doi.org/10.48550/arXiv.1906.00937

[7] Fávero, L. F., Almeida, N. R. d., & Affonso, F. J. (2025). A Systematic Mapping Study on the Modernization of Legacy Systems to Microservice Architecture. Applied System Innovation, 8(4), 86. https://doi.org/10.3390/asi8040086

[8] Fritzsch, J., Bogner, J., Wagner, S., & Zimmermann, A. (2019). Microservices migration in industry: Intentions, strategies, and challenges. In Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME 2019) (pp. 481–490). IEEE. https://doi.org/10.1109/ICSME.2019.00081

[9] Martínez Saucedo, A., Rodríguez, G. H., Rocha, F. G., & dos Santos, R. P. (2024). Migration of monolithic systems to microservices: A systematic mapping study. Information and Software Technology, 177(5), 107590. https://doi.org/10.1016/j.infsof.2024.107590

[10] Fritzsch, J., Bogner, J., Haug, M., Wagner, S., Zimmermann, A. (2024). Towards an Architecture-Centric Methodology for Migrating to Microservices. In: Kruchten, P., Gregory, P. (eds) Agile Processes in Software Engineering and Extreme Programming – Workshops. XP 2022 2023. Lecture Notes in Business Information Processing, vol 489. Springer, Cham. https://doi.org/10.1007/978-3-031-48550-3_5

[11] Matias, T., Correia, F.F., Fritzsch, J., Bogner, J., Ferreira, H.S., Restivo, A. (2020). Determining Microservice Boundaries: A Case Study Using Static and Dynamic Software Analysis. In: Jansen, A., Malavolta, I., Muccini, H., Ozkaya, I., Zimmermann, O. (eds) Software Architecture. ECSA 2020. Lecture Notes in Computer Science, vol 12292. Springer, Cham. https://doi.org/10.1007/978-3-030-58923-3_21

[12] Fritzsch, J., Bogner, J., Zimmermann, A., Wagner, S. (2019). From Monolith to Microservices: A Classification of Refactoring Approaches. In: Bruel, JM., Mazzara, M., Meyer, B. (eds) Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment. DEVOPS 2018. Lecture Notes in Computer Science, vol 11350. Springer, Cham. https://doi.org/10.1007/978-3-030-06019-0_10

[13] Agilar, E. de V., Almeida, R. B. de, & Canedo, E. D. (2016). A systematic mapping study on legacy system modernization. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE 2016). https://doi.org/10.18293/SEKE2016-059

[14] Jalukuri, M. B. (2025). Modernizing legacy systems: A journey to Kubernetes-based microservices. European Journal of Computer Science and Information Technology, 13(37), 53–65. https://doi.org/10.37745/ejcsit.2013/vol13n375365

[15] Bjørndal, N., Mazzara, M., Bucchiarone, A., Dragoni, N., & Dustdar, S. (2021). Benchmarks and performance metrics for assessing the migration to microservice-based architectures. Journal of Object Technology, 20(2), 3:1–3: XX. https://doi.org/10.5381/jot.2021.20.2.a3

[16] Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. IEEE Software, 35(3), 24–35. https://doi.org/10.1109/MS.2018.2141039

[17] Lobo, C., Fernández, I., & Piattini, M. (2022). Understanding the challenges and novel architectural models of multi-cloud native applications. Journal of Cloud Computing, 12, 6. https://doi.org/10.1186/s13677-022-00367-6

[18] Barzotto, T. R., & Farias, K. (2024). Evaluation of the impacts of decomposing a monolithic application into microservices: A case study. Journal of Software Engineering Research and Development, 10(1), 1–15. https://doi.org/10.48550/arXiv.2203.13878

[19] Faustino, D., Gonçalves, N., Portela, M., & Rito Silva, A. (2022). Stepwise migration of a monolith to a microservices architecture: Performance and migration effort evaluation. Information and Software Technology, 150, 107012. https://doi.org/10.1016/j.peva.2024.102411

[20] Meijer, W., Trubiani, C., & Aleti, A. (2024). Experimental evaluation of architectural software performance design patterns in microservices. Journal of Systems and Software, 211, 112024. https://doi.org/10.1016/j.jss.2024.112183

[21] Kansal, S., & Siddharth. (2024). Adaptive AI models for automating legacy system migration in enterprise environments. International Journal of Research Radicals in Multidisciplinary Fields, 3(2), 679–694. https://www.researchradicals.com/index.php/rr/article/view/151

[22] Gill, S. S., Golec, M., Hu, J., Xu, M., Du, J., Wu, H., … Subramanian, S. (2024). Edge AI: A taxonomy, systematic review and future directions. Cluster Computing. https://doi.org/10.1007/s10586-024-04686-y