SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

Cloud-Native Modernization As AI Readiness Infrastructure: An Architectural Framework From Multi-Industry Case Studies

Maitray Modi

Independent Researcher, USA.

Abstract
Enterprise cloud modernization programs are typically measured by two of their most common objectives: lowering the total cost of operations and eliminating technical debt. This framing systematically undervalues the most strategically valuable and complex result of these programs: the creation and operation of data platform infrastructure for ML, real-time analytics and generative AI at enterprise scale. The paper reports a multiple-case study of three enterprise cloud-native modernizations of proven business value in the telecoms, retail, and supply chain verticals. A cross-case analysis led to the formulation of an ARCMF. The AI-Ready Cloud Migration Framework (ARCMF) comprises four enabling patterns. They are the data contract, the event-driven integration backbone, the tiered data Lakehouse architecture, and the versioned inference API. Adopting Yin's multiple-case study method, the analysis revealed that organizations embedding these patterns right from the program's beginning can deploy production ML capabilities months ahead of those that treat AI enablement as a post-migration task, while achieving orders of magnitude lower marginal costs. Achievements include $12M+ per year in software licensing savings for telecommunications transformation, re-platforming a retail system comprising 85% of the organization's revenue, and eliminating $1.5M per month in supply chain operational losses with machine learning-based routing intelligence. This paper builds a multi-industry synthesis, a practitioner's four-pattern architectural building blocks for AI readiness, and seven foundational design principles for AI-ready cloud modernization. Finally, the paper discusses implications for enterprise architects, cloud platform engineers, and technology leaders; limitations in the study; and future research.

Keywords: cloud-native modernization, AI readiness, machine learning infrastructure, event-driven architecture, data Lakehouse; microservices, enterprise digital transformation, MLOps.

1. Introduction

The shift from legacy, monolithic enterprise systems to cloud-native architectures has become one of the defining capital programs of the current technological decade. Global enterprise cloud infrastructure spending is already over $500 billion in 2023 and is continuing to grow rapidly as organizations have come to realize that the differentiation that cloud-native platforms enable extends well beyond the infrastructure cost economics that originally drove adoption. However, a gap persists between the actual experiences of practitioners delivering the most successful enterprise cloud modernization programs and the dominant framing of the cloud migration problem in the academic literature on cloud migration. Research has focused on the micro-level of migration execution, including categorizing workloads, sequencing migration activities, minimizing transition risks and quantifying the cost of infrastructure [2, 3]. The planned perception that has characterized the most impactful modernization programs in practice, which is that the primary purpose of cloud-native migration is to create the event-driven, data-rich, API-consistent platform foundation that machine learning and real-time analytics require, has not received sufficient attention in research.

The framing gap has implications for design decisions. Organizations that use cost optimization framing as the primary design criterion in late planning stages for modernizing systems often make architecture decisions that turn out to be expensive workarounds when later deploying AI and machine learning capabilities. For

example, cloud storage cost optimization formats break the ACID transactional guarantees used in Lakehouse architectures to ensure reproducible model training. Language bindings of data consumption patterns for migrating legacy data systems require tight coupling between machine learning feature stores and data lakes. This prevents real-time data signals from being consumed. For instance, API designs that do not consider the requirements of batch- and stream-based machine learning inference endpoints can create version coupling that ties the retrain and redeploy cadence of production ML models to the release cadence of the transaction processing systems they are designed to augment [4]. Such architectural mismatch incurs a cost after the closure of the modernization program, outside the program's success metrics, and hence remains systematically invisible to the modernization program's governance and investment frameworks.

This article seeks to address the gap with a comparative case study of three enterprise cloud-native modernization programs in different industry verticals for which the authors have assessed outcomes. The authors' principal research question is: what architectural patterns, when implemented in a cloud modernization program from the outset, create the infrastructure on which production machine learning capabilities can be deployed most rapidly and at the lowest marginal cost?" "The contributions of this article are a multi-industry synthesis of AI readiness outputs of documented cloud modernization programs. (ii) the AI-Ready Cloud Migration Framework (ARCMF), a sequenced framework of four design patterns from the synthesis, and (iii) a set of seven design principles to apply ARCMF for practicing cloud architects.

The other sections of the article survey the literature on cloud migration patterns, event-driven integration, data Lakehouse architecture, and machine learning operations, in that order. We explain the research design in Section 3, and we review each case study in Section 4. Section 5 summarizes the cross-case findings of the building block phases, and Section 6 describes seven design principles. Section 7 provides implications and limitations of the ARCMF, including its compatibility with existing tasks. Section 8 concludes.

2 Background and Related Work

2.1 Cloud Migration Patterns and Modernization Strategies

The taxonomy of cloud migration strategies has been systematically developed in both academic and practitioner literature. Fehling et al. [2] provide a foundational pattern catalog establishing the vocabulary for reasoning about migration trade-offs. The widely adopted six Rs migration classification—Rehost, Replatform, Repurchase, Refactor, Retire, Retain—frames migration decisions as choices along a spectrum from minimal transformation to full architectural refactoring [5]. Empirical analysis of migration programs consistently finds that rehosting delivers the fastest time-to-cloud at lowest risk, while refactoring delivers the highest long-term operational and capability value at substantially higher execution cost [3, 6].

The strangler fig pattern, formalized by Fowler [7], provides the practical resolution to the operational continuity challenge that makes pure refactoring impractical for revenue-bearing systems: by incrementally building new functionality alongside legacy components and gradually routing traffic from old to new, the transformation can be executed without taking the system offline. Newman [8] extends this to microservices decomposition specifically, providing guidance on identifying bounded contexts that can be independently migrated and validated. Dragoni et al. [9] provide a comprehensive academic treatment of microservices architecture principles that contextualizes the practitioner migration literature within formal distributed systems theory, characterizing the design properties—independent deployability, decentralized data management, and infrastructure automation—that distinguish microservices from prior service-oriented architectures. Richardson [10] extends the practitioner pattern language to specific implementation challenges—saga patterns for distributed transaction management, API gateway patterns for client aggregation, and circuit breaker patterns for resilience—directly relevant to the programs analyzed in this study.

2.2 Event-Driven Integration Architectures

The transition from synchronous point-to-point integration to event-driven architecture is one of the most consequential and least reversible decisions in a cloud modernization program, with implications extending from the integration layer into the feasibility of real-time analytics and machine learning at scale. Hohpe and Woolf [12] established the canonical pattern language for enterprise integration—event channels, event processors, message routers, and correlation identifiers—on which contemporary streaming platform implementations build. Kleppmann [4] provides the most comprehensive contemporary treatment of event-driven data systems, explicitly connecting the operational characteristics of event log architectures—

immutability, append-only writes, consumer group independence, and configurable retention—to the requirements of machine learning feature pipelines.

Apache Kafka, the distributed event streaming platform underlying the majority of enterprise event-driven architectures at scale, has been extensively analyzed in both operational and academic contexts [13]. Its core properties—partition-based parallelism, consumer group isolation, configurable retention, and exactly-once delivery semantics—make it particularly suitable for the AI readiness use case: machine learning feature extraction pipelines can be added as independent consumer groups to an existing event streaming backbone without modifying the producer services generating the events, enabling AI development to proceed independently of the transaction system roadmap [14]. Stopford [14] provides a practitioner treatment of event streaming architecture that specifically addresses schema design, stream processing topologies, and consumer group strategies with ML pipeline integration in mind.

2.3 Data Lakehouse Architecture and Machine Learning Platform Readiness

The data Lakehouse architecture — combining the ACID transaction guarantees and query performance of data warehouses with the storage flexibility and cost economics of data lakes—was formally described by Armbrust et al. [15] and subsequently implemented as the open table format layer across Apache Iceberg, Delta Lake, and Apache Hudi. The Bronze-Silver-Gold tier structure provides natural integration points for ML feature engineering at each transformation stage: raw events in Bronze, cleansed canonical entities in Silver, and business-ready feature aggregates in Gold. This staged transformation matches the structure of ML feature engineering pipelines and enables feature logic to be implemented as declarative Lakehouse transformations rather than bespoke pipeline code.

ACID transaction guarantees address a critical ML training data reliability requirement that raw data lake architectures cannot meet: concurrent writers to raw object storage can produce partial writes, inconsistent snapshots, and read-modify-write conflicts that corrupt training datasets without surfacing obvious errors [15]. Delta Lake's time travel capability — querying table state at any historical timestamp — enables training runs to be reproduced against the exact historical data snapshot used for any prior run, supporting systematic debugging of model performance changes [16]. Zaharia et al. [16] describe the Delta Lake architecture in detail, characterizing its performance characteristics relative to both data warehouse and raw data lake alternatives.

2.4 Machine Learning Operations in Enterprise Environments

The MLOps literature addresses the engineering discipline of deploying, monitoring, and governing machine learning models in production at enterprise scale. Kreuzberger et al. [17] provide a comprehensive review synthesizing the MLOps research landscape into a seven-phase lifecycle—data ingestion, feature engineering, model training, evaluation, deployment, monitoring, and governance—and characterizing the architectural infrastructure requirements at each phase. Two MLOps concepts are directly relevant to the AI readiness framing of this article: training-serving skew and the feature store.

Training-serving skew—the performance degradation when feature distributions in production differ from those in training data—arises most commonly when feature engineering transformations at training and inference time are implemented separately [17]. The architectural solution is a shared feature store: a platform layer maintaining canonical feature transformation logic and serving the same feature values to both training pipelines and production inference endpoints. This requirement connects directly to the Lakehouse architecture decisions made during modernization because the Lakehouse's Gold tier is the natural implementation substrate for the feature store that MLOps practice requires. Kim et al. [18] situate MLOps within the broader DevOps literature, characterizing the organizational and technical changes required for the deployment frequency and reliability that production ML demands.

3 Research Design

3.1 Case Study Methodology

This study employs a structured multiple-case study design following Yin's [19] methodology. Case study research is appropriate when a research question asks "how" or "why" about a contemporary phenomenon within its real-world context and when the researcher cannot control the variables of interest [19]. The central research question — how specific architectural patterns embedded in cloud modernization programs create the infrastructure enabling production machine learning deployment — satisfies both criteria: it is a how question about observable organizational and technical outcomes, and the architectural decisions and organizational dynamics cannot be experimentally controlled.

The multiple-case comparative design, rather than single-case analysis, is motivated by Yin's replication logic [19]: each additional case either confirms patterns observed in prior cases (literal replication), strengthening confidence in the framework's external validity, or reveals boundary conditions that refine its scope (theoretical replication). The three cases were selected to provide theoretical replication across industry verticals, legacy platform archetypes, and migration strategy types—maximizing the range of organizational and technical variation across which the framework's patterns can be evaluated.

3.2 Case Selection and Data Sources

Case selection applied four purposive criteria: (1) the program involved substantial cloud-native modernization of a revenue-bearing or operationally critical enterprise system; (2) verified financial and operational outcomes are available for the post-deployment period; (3) the program included explicit architectural decisions regarding AI and machine learning readiness; and (4) sufficient architectural documentation is available for detailed comparative analysis. All three selected cases satisfy these criteria. Data sources include architectural design documentation produced during program execution, operational outcome metrics from post-deployment monitoring, and practitioner retrospective analysis by the program's lead enterprise architect.

3.3 Analytical Approach

Cross-case analysis followed a pattern-matching logic [19]: architectural decisions and outcomes observed in each case were compared against an a priori theoretical framework derived from the cloud architecture and MLOps literature, and the fit between observed patterns and theoretical predictions was systematically assessed. Discrepancies between prior theory and observed practice were treated as opportunities to extend the framework. The ARCMF presented in Section 5 represents the output of this iterative pattern-matching process and reflects both confirmation of prior theoretical predictions and novel synthesis arising from the cross-case comparison.

4 Case Study Analysis

4.1 Case 1: Telecom Customer Data Platform — Strangler-Fig Migration

A large American telecommunications provider operated a customer data platform built around a commercial CRM suite and an enterprise Oracle database accumulated over fifteen years of incremental customization. At program initiation, the platform managed identity, subscription, billing, and interaction history data for tens of millions of subscribers. Technical debt assessment revealed that approximately 60% of the supporting virtual machine fleet operated at less than 20% average CPU utilization, and that the customer entity was defined inconsistently across seven distinct source systems — a data quality condition that directly precluded reliable machine learning model training, because any model trained on this data would learn the inconsistency along with the signal.

The program established formal data contracts — versioned specifications of customer entity definitions, validation rules, and canonical field ownership — as the first deliverable, before any migration execution began. The target architecture replaced the legacy CRM and Oracle database with cloud-native microservices organized around six bounded customer data domains, integrated through a managed Apache Kafka-compatible event streaming backbone providing a durable, replayable audit trail of every customer data change event. The data platform was simultaneously modernized into a Bronze-Silver-Gold Lakehouse architecture on cloud object storage with an open table format supporting ACID transactions and time travel. Migration of more than 450 virtual machines was executed using the strangler-fig pattern over 36 months, with each domain migrated and validated independently. Outcomes included $12M+ in annual software licensing savings, decommissioning of 450+ virtual machines, and a Lakehouse and event streaming foundation on which the organization's data science teams deployed production personalization, churn prediction, and propensity models within months of platform go-live.

4.2 Case 2: Retail Point-of-Sale Platform — Event-Driven Re-Platforming

A major telecommunications retailer operated a point-of-sale platform across more than 10,000 stores, responsible for approximately 85% of organizational retail revenue. The platform processed millions of daily transactions — device activations, plan upgrades, trade-in processing, accessory sales, and payment tendering — and was operationally non-negotiable throughout the modernization. The existing hybrid architecture combining on-premises services, cloud-hosted microservices, and legacy Windows Forms desktop components could not support the real-time ML inference capabilities — personalized offers, real-time inventory recommendations — the business required for competitive differentiation.

The re-platforming program was structured across eight cross-functional Agile teams with a combined footprint of over 100 practitioners, coordinated under a SAFe operating model. The target architecture comprised a React-based progressive web application, a Spring Boot microservices layer organized around eight independently deployable service domains, and a real-time event streaming backbone. The architecturally critical decision was the explicit design of the event streaming backbone to serve as the real-time signal channel for ML inference from the first day of architectural specification — event schema design, partition key selection, consumer group isolation, and retention policy were all specified with ML recommendation pipeline requirements in mind. The microservices API layer was designed with versioned contracts enabling ML inference endpoint deployments to be decoupled from transaction service release cycles. The re-platforming delivered measurable improvements in transaction latency, peak-period scaling, and deployment velocity. The event streaming backbone and versioned API contracts were consumed by data science teams to deploy recommendation and propensity models in production within months of platform go-live.

4.3 Case 3: Supply Chain - Mainframe to Cloud-Native with ML Routing Intelligence

A megasystem supporting the distribution of telecommunications devices manages the physical lifecycle of millions of mobile devices per year through a commercial ERP system. A batch processing architecture supports hours of latency between the occurrence of physical device lifecycle events and usable data being distributed to the distribution centers, causing employees to work with stale information when allocating and routing devices. Systematic errors in device disposition (unnecessary liquidations, missed maintenance opportunities and mis-shipments) cost close to $1.5M a month.

As part of the modernization, the legacy ERP integration has been replaced with a cloud-native event-driven approach that ingests real-time device lifecycle events and uses ML routing intelligence to determine the optimal disposition path for each device based on its condition telemetry, market demand signals, and operational capacity constraints. Data contracts that define the canonical device entity have been created as the first architectural deliverable. The ML models were hosted as managed inference endpoints in a real-time event processing pipeline using historical disposition data extracted from the legacy system. The pipeline reduced the time between an event being created and recorded from hours to seconds. The ML routing intelligence delivered the expected $1.5M/month in savings by the end of the quarter in which it was first deployed fully into production operation. It was the clearest case of AI readiness architecture in action of the three case studies.

5 Cross-Case Synthesis: The AI-Ready Cloud Migration Framework

5.1 The Four Enabling Architectural Patterns

Cross-case comparisons for all three programs identify four architectural patterns that ease fast, low-marginal-cost deployment of AI capabilities across the enterprise, leading to the AI-Ready Cloud Migration Framework (ARCMF).

Pattern 1: Data Contract Establishment. Formalizing the definitions, ownership rules, validation criteria, and canonical representations of all the core business entities that will be managed by the system being modernized before the migration execution begins. Data contracts reduce the cost of implicit reverse engineering of legacy data semantics, enabling faster migration, while also providing the data quality guarantee needed for a trustworthy machine learning training dataset. Data contracts are developed as the first program deliverable in Case 1 and as the activity to standardize data in the initiation phase of Cases 2 and 3. The data management literature is likewise supportive of this approach, with DAMA International [20] citing data contract definition as a prerequisite of any data integration program.

Pattern 2 - Event-Driven Integration Backbone: Replace point-to-point synchronous integration with a durable event streaming backbone in which the event producer is decoupled from the event consumer and events are immutable, replayable, and consumer-independent. All three are present, allowing ML feature pipelines to be added as new consumers of enterprise transaction events without modifying the upstream systems, decoupling the AI system development velocity from the transaction system release cycles [4, 14]. .

Pattern 3 - Tiered Data Lakehouse Architecture: Data from the analytics platform is organized according to Bronze-Silver-Gold tiers in a cloud object store with an open table format providing ACID transactions, schema enforcement, and time travel. This kind of design not only achieves ACID guarantees for data consistency during training (training data) but also enables feature engineering to create the tier structure where ML-specific transformations naturally occur for the intermediate models. Without this, data science teams create point-to-

point data pipelines that introduce the training-serving skew and integration dependencies that affect both teams' architecture [17].

Pattern 4 - Versioned Inference API Integration. Designing the microservices API layer with explicit support for ML inference endpoint invocation and versioned API contracts that decouple ML model update cadences from transaction service release cycles. This can be most clearly seen in Case 2, where the streaming backbone for events acted as a real-time channel for recommending model inference from the first design phase. In Case 3, the inference endpoint was defined as a first-class member of the event processing pipeline from its beginning.

5.2 Sequencing dependencies between patterns

The four ARCMF patterns share explicit sequencing dependencies to guarantee the critical path for establishing the AI readiness infrastructure. Data contracts (Pattern 1) should materialize before the rest of the patterns. Since authoritative definitions of entities do not exist, the design of event schema and Lakehouse tier logic for transforming features into ML training data can be ambiguous. The event-driven backbone (Pattern 2) should be implemented before the Lakehouse (Pattern 3) starts receiving real-time change events from transaction systems and ML inference endpoints start receiving real-time prediction request signals (Pattern 4). It implies that a Lakehouse (Pattern 3) must have been created before running feature engineering pipelines, which should be completed before production-ready models are trained and deployed on inference endpoints (Pattern 4).

These four patterns comprise the base (critical path) for building AI infrastructure: Contracts, Backbone, Lakehouse, and Inference API. Any program that skips or shortcuts some of these four patterns will encounter the same issues: data inconsistencies in training data (skipping Pattern 1), lack of real production signaling during feature pipeline execution (skipping Pattern 2), training-serving skew due to feature transformations divergence (skipping Pattern 3), and model-to-application release coupling (skipping Pattern 4).

5.3 Comparative Outcomes Across Cases

Table 1 summarizes the comparative dimensions and outcomes across all three cases.

Dimension	Case 1: Telecom	Case 2: Retail	Case 3: Supply Chain
Industry vertical	Telecommunications	Telecommunications retail	Device distribution
Legacy archetype	Commercial CRM + Oracle DB	Hybrid on-prem / cloud POS	Commercial ERP (batch)
Migration pattern	Strangler fig (36 months)	Event-driven re-platform (~24 months)	Full cloud-native replacement (~18 months)
Pattern 1 — Data Contracts	First program deliverable	Entity standardization at initiation	Device entity contracts at initiation
Pattern 2 — Event Backbone	Managed Kafka backbone	Real-time streaming for all domains	Real-time device lifecycle events
Pattern 3 — Lakehouse	Bronze-Silver-Gold on object storage	Unified data platform built in parallel	Lakehouse for historical training data
Pattern 4 — Inference API	Versioned APIs for DS teams	Designed for ML inference from day one	ML endpoints in event processing pipeline
Primary financial outcome	$12M+ annual licensing savings	85% of org revenue re-platformed	$1.5M/month operational loss eliminated
ML deployment velocity	Production models within months of go-live	Production models within months of go-live	Production routing intelligence in Q1

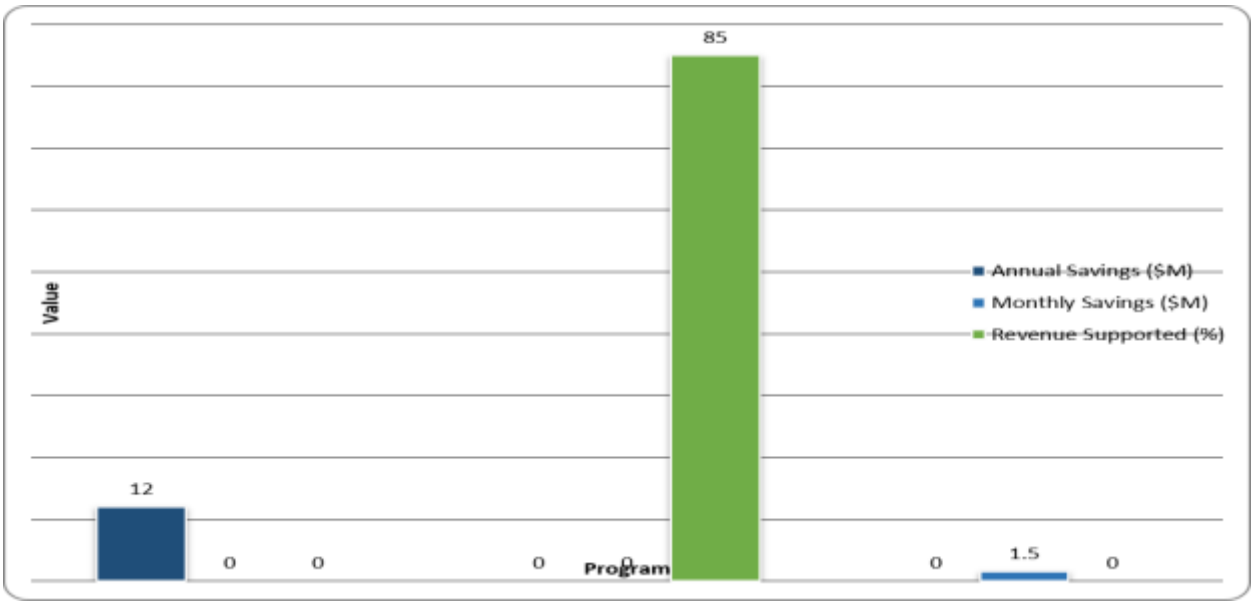

Figure 1: Financial Outcomes by Modernization Program

The uniformity of ARCMF pattern presence across cases with substantially different industry contexts, legacy platform archetypes, program scales, and migration strategies provides consistent cross-case evidence that the four-pattern framework captures a portable, context-independent set of enabling conditions for AI readiness in cloud modernization programs.

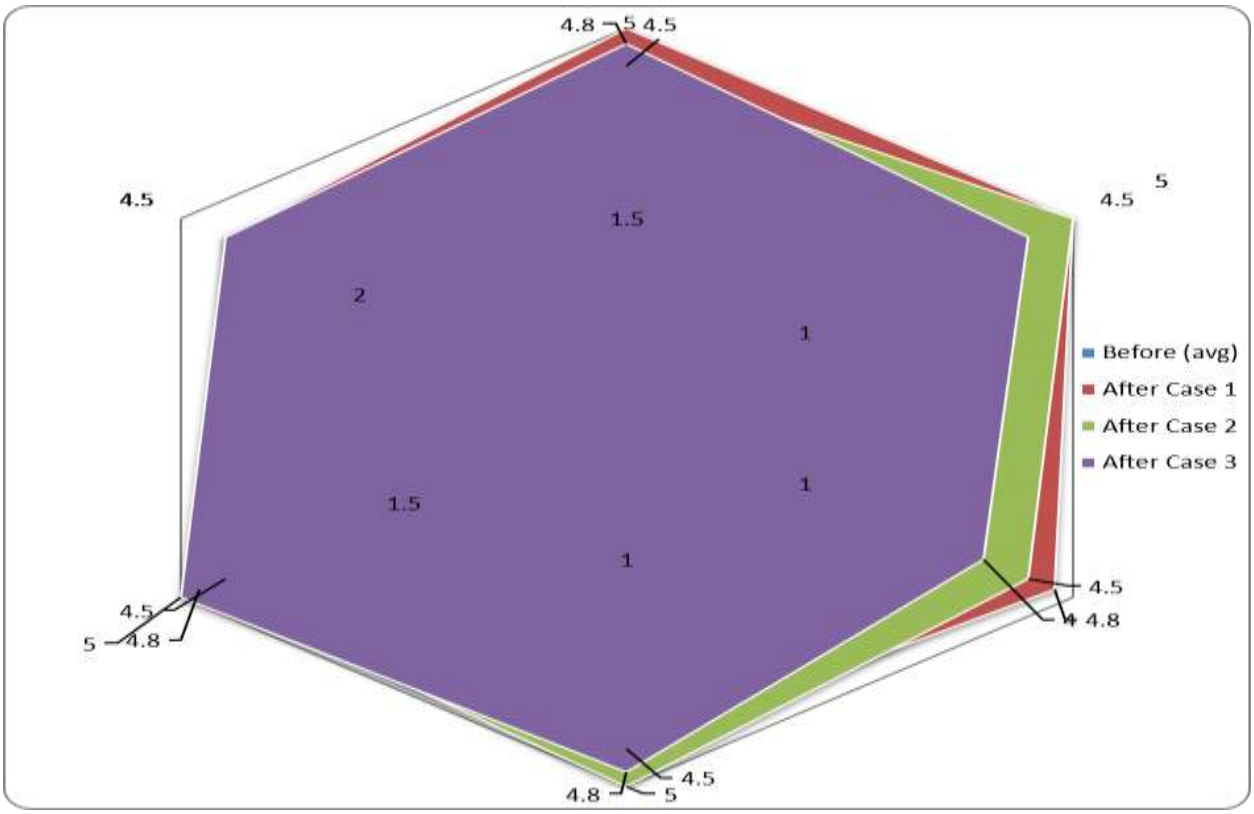

Figure 2: Governance Maturity Before vs After Modernization

6 Design Principles for AI-Ready Modernization

The cross-case synthesis resulted in the identification of seven design principles, which are supported by at least two of the three cases and by the cloud architecture and MLOps literature.

Principle 1: Treat data quality as integral to the data contract. The data contract is the foundational deliverable in any cloud modernization project to become AI-ready. Thus, the data contract definition in Case 1 when starting the program was time-consuming but did not incur serious losses in integration performance and ML platform enablement. Similar aspects were recognized in the field of data management [20] and in MLOps-based ML training data quality requirements [17].

Principle 2: Create event schemas for ML consumers, not just for transaction producers. The event schemas for use cases focused on transaction processing may lack contextual signals such as event timestamps, entity state at the time of the event, and causal identifiers, which are important for ML feature pipelines. Including machine learning consumer features in the event schema specification is free and can be evolved at low cost when machine learning is enabled [4, 14].

Principle 3: Prefer open table formats for all analysis data. The ACID transaction guarantees, time travel, and schema evolution guarantees of open table formats are not optional conveniences. They are requirements for reproducible ML training and reliable feature stores. If these formats are chosen during the modernisation process, the cost of data re-ingestion and pipeline rebuilding exceeds the difference in storage costs [15, 16].

Principle 4: Build the Lakehouse before building the models. Data science teams who don't have a Lakehouse in place when building models will build point-to-point data pipelines, lack data quality and monitoring guarantees, create training-serving skew, and create integration dependencies that limit design choices later on [17].

Principle 5: Version inference API contracts from day one. ML models have their own cadence of improvement, decoupled from transaction systems. Versioned API contracts on the caller and inference endpoints are the minimal architectural primitives that enable independent deployment of callers and inference endpoints. If version numbers are not specified on API contracts for consuming ML inference endpoints, the costs of subsequent API versioning are higher, and its risk of regression is greater than otherwise [8, 10].

Principle 6: Use the strangler fig pattern within the AI-bearing transaction systems. Transaction systems that must remain online during modernization must be migrated using the strangler fig pattern, and each bounded domain must be migrated and validated independently before the legacy component can be decommissioned. This makes API contract redesign and event streaming, which are essential to AI readiness possible, and minimizes the impact that this change has on operational service [7, 8].

Principle 7: Measure AI deployment velocity as a first-class modernization success metric. Programs that measure the readiness of their platforms for AI as a metric, e.g., time from platform go-live to first production ML model deployed, invest differently in enabling infrastructure compared to programs focused only on monetary cost and technical debt. In all cases, measuring AI deployment velocity at the program's start impacted architecture because it provided accountability for AI readiness outcomes not captured in cost or infrastructure.

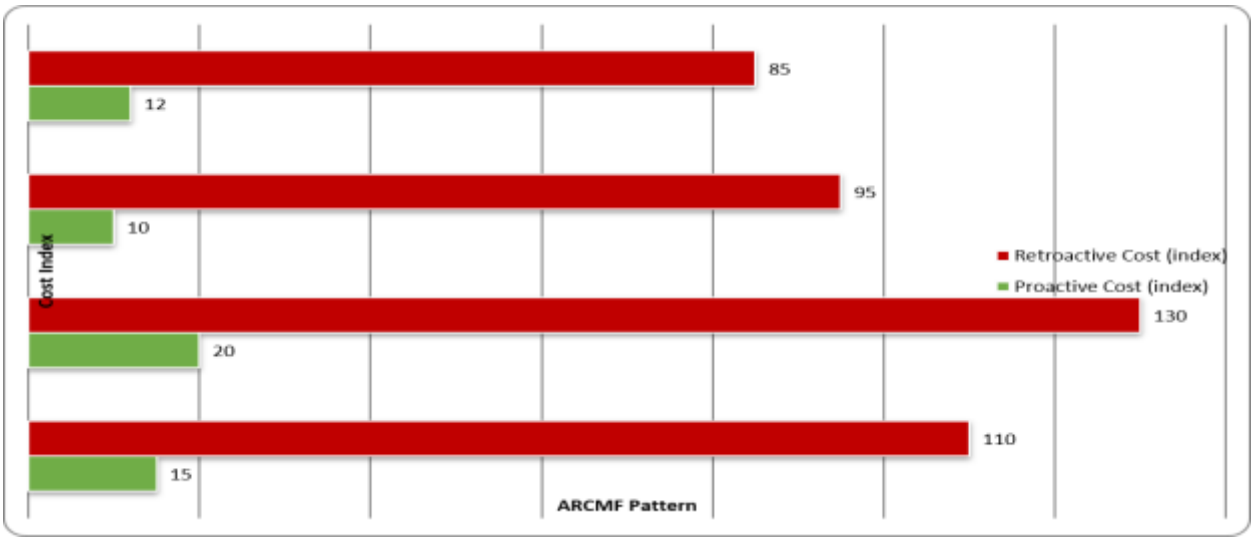

Figure 3: Proactive vs Retroactive AI Readiness Cost Index

7 Discussion

7.1 Implications for Enterprise Architecture Practice

The ARCMF provides a structured design tool for a practitioner community largely reliant on vendor-specific migration guidance and case-by-case experience sharing [22]. Its four-pattern structure and explicit sequencing dependencies address a fundamental organizational challenge in modernization programs: the architectural decisions that most determine AI readiness outcomes are made in early program phases, when AI enablement may not be on the near-term roadmap and the case for AI-readiness investment is weakest. The framework gives architects a principled basis for advocating these design investments during program initiation—when they are cheapest—rather than discovering their absence during subsequent AI development phases—when they are most expensive to retrofit [23].

The cross-industry validity of the framework across telecommunications, retail, and supply chain verticals, with legacy archetypes ranging from commercial CRM and Oracle databases to mainframe-era ERP batch systems, suggests robustness to variation in the most common sources of contextual heterogeneity between enterprise modernization programs. The boundary conditions of this applicability require further investigation, particularly for regulated industries—financial services, healthcare, and defense—where event streaming backbone design must accommodate regulatory audit requirements and data contract obligations may be legally mandated [24].

7.2 Limitations and Threats to Validity

Four primary limitations constrain this study. First, cases were selected purposively on theoretical replication criteria rather than statistical representativeness; the ARCMF cannot be generalized probabilistically to the population of enterprise cloud modernization programs. Second, all three cases were led by the same architect, introducing a potential confound between framework efficacy and individual practitioner expertise [25]. Third, the study presents each case without a counterfactual—the cost of equivalent AI readiness without applying the ARCMF patterns cannot be directly observed. Fourth, the study includes no contrast cases where ARCMF patterns were not applied, which would strengthen empirical claims by demonstrating AI readiness failures attributable to pattern absence [26].

These limitations suggest a clear agenda for future research: studies involving different practitioners, contrast cases where AI readiness was not treated as a first-class modernization objective, longitudinal analysis tracking AI enablement cost as a function of ARCMF pattern presence at program initiation, and investigation of the framework's applicability in mid-market organizations with smaller engineering teams and tighter investment constraints [27].

7.3 Relationship to Existing Cloud Migration Frameworks

The ARCMF is complementary to, rather than competitive with, existing cloud migration execution frameworks such as the AWS Migration Acceleration Program [21] and Microsoft Azure Cloud Adoption Framework [22]. These frameworks address migration governance methodology—how to plan, sequence, and govern migration execution—while the ARCMF addresses architectural design decisions—what patterns to build and in what

sequence. A modernization program using an existing execution framework for governance and the ARCMF for AI-readiness design is more likely to achieve both operational and capability outcomes than one using either framework alone . The ARCMF is similarly complementary to MLOps frameworks [17]: MLOps describes how to build machine learning systems on an existing platform; the ARCMF describes how to design the platform itself to make MLOps practices most productive [28]. The organizational sequence—ARCMF during modernization planning and execution and MLOps frameworks during subsequent ML system development—represents the implementation order through which the AI-ready outcomes documented in the three cases were achieved [29, 30, 31].

8 Conclusion

This article has presented a structured comparative case study of three enterprise cloud-native modernization programs across telecommunications, retail, and supply chain verticals, deriving the AI-Ready Cloud Migration Framework (ARCMF) through cross-case analysis and demonstrating its connection to verified financial and AI enablement outcomes. The central argument is that the strategic value of cloud-native modernization lies primarily in the AI readiness infrastructure it creates — and that organizations which design explicitly for AI readiness from program initiation, by embedding the four ARCMF patterns before migration execution begins, achieve substantially faster and lower-cost AI capability deployment than those treating AI enablement as a subsequent phase.

The framework's empirical grounding in three programs with documented outcomes — $12M+ in annual savings, re-platforming of a system supporting 85% of organizational revenue, and $1.5M per month in eliminated operational losses through production ML routing intelligence — provides an evidentiary foundation for its architectural prescriptions that purely theoretical analysis cannot offer. Its cross-industry validity across three substantially different legacy archetypes and migration patterns suggests a generalizability extending beyond the specific contexts studied, while the study's methodological limitations define the research agenda that subsequent work must address to establish the framework on a fully robust empirical basis.

Enterprise architects planning cloud modernization programs have in the ARCMF a framework for connecting the architectural decisions made during program planning to the AI deployment outcomes that materialize months or years later. The seven design principles distilled from the synthesis provide operationally specific guidance for applying the framework under the resource and timeline constraints of real modernization programs. The field of cloud computing research has much to gain from more systematic comparative analysis of AI readiness outcomes; this article is a contribution to establishing the methodology and theoretical vocabulary through which that research agenda can advance.

References

[1] Gartner (2023) Gartner forecasts worldwide public cloud end-user spending to reach nearly $600 billion in 2023. Gartner Inc, Stamford CT. https://www.gartner.com/en/newsroom/press-releases/2023-04-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023

[2] Fehling C, Leymann F, Retter R, Schupeck W, Arbitter P (2014) Cloud computing patterns: fundamentals to design, build, and manage cloud applications. Springer, Vienna

[3] Linthicum DS (2017) Cloud computing and SOA convergence in your enterprise: a step-by-step guide. Addison-Wesley, Boston, MA

[4] Kleppmann M (2017) Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems. O'Reilly Media, Sebastopol, CA

[5] Gartner (2020) 5 options to migrate your applications to the cloud. Gartner Research, Stamford, CT

[6] Toffetti G, Brunner S, Blöchlinger M, Dudouet F, Edmonds A (2015) An architecture for self-managing microservices. Proceedings of the 1st International Workshop on Automated Incident Management in Cloud. ACM, New York, pp 19–24

[7] Fowler M (2004) Strangler fig application. martinfowler.com. https://martinfowler.com/bliki/StranglerFigApplication.html

[8] Newman S (2019) Monolith to microservices: evolutionary patterns to transform your monolith. O'Reilly Media, Sebastopol CA

[9] Dragoni N, Giallorenzo S, Lafuente AL, Mazzara M, Montesi F, Mustafin R, Safina L (2017) Microservices: yesterday, today, and tomorrow. In: Present and ulterior software engineering. Springer, Cham, pp 195–216. https://doi.org/10.1007/978-3-319-67425-4_12
[10] Richardson, C. (2018). Microservices patterns: with examples in Java. Manning Publications, Shelter Island, NY
[11] Bass L, Clements P, Kazman R (2012) Software architecture in practice, 3rd edn. Addison-Wesley, Upper Saddle River, NJ
[12] Hohpe G, Woolf B (2004) Enterprise integration patterns: designing, building, and deploying messaging solutions. Addison-Wesley, Boston, MA
[13] Apache Software Foundation (2023) Apache Kafka documentation.
https://kafka.apache.org/documentation
[14] Stopford B (2018) Designing event-driven systems: concepts and patterns for streaming services with Apache Kafka. O'Reilly Media, Sebastopol, CA
[15] Armbrust M, Ghodsi A, Xin RS, Zaharia M (2021) Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. 11th Annual Conference on Innovative Data Systems Research (CIDR 2021). https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf
[16] Zaharia M, Armbrust M, Das T, Davidson E, Ghodsi A, Or A, Wendell P, Xin RS (2020) Delta lake: high-performance ACID table storage over cloud object stores. Proceedings of the VLDB Endowment 13(12):3411–3424. https://doi.org/10.14778/3415478.3415560
[17] Kreuzberger D, Kühl N, Hirschl S (2023) Machine learning operations (MLOps): overview, definition, and architecture. IEEE Access 11:31866–31879. https://doi.org/10.1109/ACCESS.2023.3262138
[18] Kim G, Humble J, Debois P, Willis J (2016) The DevOps handbook: how to create world-class agility, reliability, and security in technology organizations. IT Revolution Press, Portland OR
[19] Yin RK (2018) Case study research and applications: design and methods, 6th edn. SAGE Publications, Thousand Oaks CA
[20] DAMA International (2017) DAMA-DMBOK: data management body of knowledge, 2nd edn. Technics Publications, Basking Ridge NJ
[21] Amazon Web Services (2023) AWS migration acceleration program. https://aws.amazon.com/migration-acceleration-program
[22] Microsoft (2023) Microsoft Azure cloud adoption framework. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework
[23] Reddy SY, Babu MK, Unnikrishnan R (2025) Decoupling cloud security (DCS): a framework for data sovereignty and cross-border cloud compliance. J Inf Syst Eng Manag 10(4).
https://doi.org/10.52783/jisem.v10i4.9317
[24] Gollapudi R (2026) Autonomous multi-zone replication for zero-loss settlement systems. Int J Comput Exp Sci Eng 12(1). https://doi.org/10.22399/ijcesen.4817
[25] Suthari Y, Mohan P (2025) Cloud-driven machine learning-based framework for measuring customer experience in digital touch-points. In: Proceedings of the 2025 IEEE 11th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA), pp 328–333. IEEE.
https://doi.org/10.1109/ICSIMA66552.2025.11233566
[26] Gollapudi R (2022) Risk-controlled near-zero-downtime Oracle database migration using GoldenGate. Int J Comput Exp Sci Eng 8(3). https://doi.org/10.22399/ijcesen.5382
[27] Quintero FA (2024) Reducing production time without compromising quality: optimization strategies in high-end VFX simulations. Sarcouncil Journal of Engineering and Computer Sciences 3(8):1–8
[28] Kumar PS (2025) Index-state uncertainty for trustworthy enterprise retrieval-augmented generation (RAG). J Inf Syst Eng Manag 10(36s):1258–1271. https://doi.org/10.52783/jisem.v10i36s.15043
[29] Quintero FA (2023) Fluid dynamics-based VFX design: trade-offs between visual realism, computational cost, and production timelines. Journal of Computational Analysis and Applications (JoCAAA) 31(4):2722–2736. https://www.eudoxuspress.com/index.php/pub/article/view/5148
[30] Bajaj D, Shah V, Kanaparthi S (2024) A practical framework for migrating large manual test suites to automation pipelines: an industrial case study. J Inf Syst Eng Manag 9(3):1–8
[31] A-Clottey F (2023) Project oversight and client relationship management as drivers of sustainable business growth under strategic leadership frameworks. Int J Comput Exp Sci Eng 9(4).
https://doi.org/10.22399/ijcesen.5146