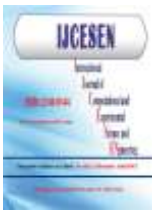
International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.1 (2026) pp. 229-239
http://www.ijcesen.com



ISSN: 2149-9144

Research Article

AI-Driven Strategic Modernization of Legacy ETL Workflows on Serverless Cloud Platforms

Thananjayan Kasi*

HCL America Inc., USA
* **Corresponding Author Email:** thananjayankasi@gmail.com - **ORCID:** 0000-0002-4407-7550

Article Info:

DOI: 10.22399/ijcesen.4754
Received : 03 November 2025
Revised : 28 December 2025
Accepted : 08 January 2026

Keywords

Serverless Computing,
ETL Modernization,
Event-Driven Architecture,
Distributed Processing,
Cloud-Native Transformation

Abstract:

Legacy ETL systems create considerable operational overhead with manual capacity management, rigid scheduling models, and inadequate cloud integration support. The proposed framework in this article presents a systematic solution for migrating legacy ETL processes to serverless cloud environments, overcoming essential shortfalls through AI-aided workflow classification, natural language processing-assisted code translation, and event-oriented orchestration. Unlike previous lift-and-shift or containerized models with operational inflexibilities, this framework embraces fully managed serverless services integrated with machine learning functions. The result is considerable cost savings, performance enhancements via distributed processing engines, and improved data freshness. The framework involves automated inventory documentation via metadata-driven models, AI-aided complexity stratification across heterogeneous data, Apache Spark-based code refactoring using Business Process Model and Notation templates, and reliable, fault-tolerant, trigger-activated orchestrations. The model ensures improved reliability, scalability, and standards compliance while eliminating infrastructure-related operational issues. Challenges in metadata completeness gaps, translation processes with semantic drift, artificial intelligence-related bias, and operational constraints are addressed comprehensively through complete provenance tracking and parallel validations. The proposed framework outperforms current serverless cloud-migration solutions through intelligent automation and pattern-driven optimization. In an anonymized enterprise case study, the framework reduced end-to-end batch processing time and infrastructure cost while improving workflow success rates and migration throughput compared to the legacy environment. A formal evaluation section details workflow classification quality, translation accuracy, and operational improvements observed during the migration program.

Index Terms: Serverless Computing, ETL Modernization, Event-Driven Architecture, Distributed Processing, Cloud-Native Transformation

1. Introduction

1.1 Legacy ETL Challenges and Technical Debt

Conventional ETL(Extract, Transform, Load) operations, implemented using traditional server infrastructure and batch processing approaches, face many obstacles in modern data environments. Legacy ETL tools impose considerable operational overhead, where data engineering teams dedicate disproportionately large resources to system maintenance rather than development efforts. These conventional tools require manual capacity management, operate according to fixed schedules irrespective of data availability dynamics, and lack strong integration mechanisms with cloud storage and analytics services [1]. The technical debt generated by monolithic design in traditional ETL systems increases exponentially as data volumes evolve from terabytes to petabytes, along with fragile codebases, limited documentation, and organizational knowledge dependencies.

Enterprise data warehousing infrastructures manage hundreds to thousands of individual ETL processes with intricate interlocks, making these processing chains brittle. Single-point failures cascade across multiple downstream processes. The lack of flexibility in traditional architecture becomes

evident through proprietary constraints on deployment cycles and inabilities to support real-time processing needs [1]. The batch processing window, which originally allowed overnight completion, has become challenging to finish within service-level agreements, with significant amounts of key processing tasks exceeding allocated timeframes. Capacity allocation processes for mitigating peak workload processing have created inefficiency in capacity utilization during non-peak periods.

1.2 Serverless Cloud Paradigm and Cost Efficiency

Serverless cloud infrastructure completely restructures ETL system design through dynamic resource provision, pay-for-use pricing, and intrinsic support for distributed storage systems. Serverless computing disrupts traditional cloud migration patterns, which originally aimed to mirror existing infrastructure in virtual machines. Original cloud migration patterns were predominantly based on lift-and-shift models and containerization, which left core operational complexities undiminished and required dedicated platform engineering teams. Real-world observations demonstrate that serverless computing models offer significant cost savings compared to continuously provisioned infrastructure, with optimal results attainable for workloads with variant execution and periodic processing schedules [2]. Cost savings result from eliminating resource idle times, automatic scalability, and millisecond-level granularity of cost measures.This approach progresses beyond prior methods by focusing on fully managed services, event-oriented execution models, and reduced complexity orchestration designs. Contemporary serverless offerings include dynamic scaling that adapts to changing workloads at millisecond speeds, processing power that divides large datasets across temporary compute environments, and monitoring capabilities that facilitate execution pattern insights. The paradigm shift from monolithic, schedule-bound execution environments to modular, event-oriented execution designs enables enterprise-level access to sub-hour data freshness that was not possible with prior frameworks [2].

1.3 Opportunities in ETL Modernization using AI

Artificial intelligence methodologies bring transformative capabilities across the migration lifecycle, resolving long-standing challenges in understanding legacy code and improving workflows. Large language models show considerable promise in automatically modernizing legacy code, including intensive documentation generation, inherent business logic retrieval, and translation of proprietary scripting languages to cloud-native systems [3]. Natural language processing methods facilitate semantic-level code repository analyses to automatically retrieve transformation patterns and business rules manifested in procedures without formal definitions.Machine learning algorithms improve workflow classification accuracy by analyzing historical execution patterns, resource utilization profiles, and data volume attributes to recognize optimal serverless configurations [4]. Supervised machine learning algorithms trained on enterprise ETL portfolios enable intelligent classification of workflows into complexity levels. These models provide migration effort estimates more accurately than manual assessment methodologies.The framework uses artificial intelligence across various transformation stages: (1) natural language processing for automatic code analysis and documentation generation, (2) machine learning for intelligent workflow classification and resource allocation, and (3) deep learning for recognizing patterns in transformation logic [5]. However, artificial intelligence integration introduces specific risks: bias in automatic code translation accuracy, semantic drift in automated workflow classification, and explainability challenges in machine learning recommendations.Rather than proposing new learning algorithms, the contribution of this article lies in an applied AI-enabled migration framework. The framework systematically combines supervised workflow classification, transformer-based code understanding and translation, and pattern-driven orchestration. The novelty resides in how these AI capabilities integrate into a repeatable migration lifecycle, with explicit attention to validation, provenance, and operational risk management in large-scale ETL estates.Figure 1 below highlights the structural pillars of the framework that include workflow analysis and categorization, logic transformation and modernization, orchestration and execution control, and resilience and compliance. These provide systematic migration capabilities from legacy infrastructure to cloud infrastructure.

2. Assessment and Classification of Workflow

2.1 Systematic Inventory Analysis

The entire process begins with workflow documentation, creating a comprehensive inventory

of existing ETL processes using automated discovery tools and manual audits. The inventory process captures crucial data, including processing frequency, runtime duration, resource usage patterns, and data throughput for existing processes. Current data warehousing designs struggle with adequate documentation challenges, where a large portion of legacy processes lacks specification documentation and designs are defined in code alone without corresponding design documents [6].

Data lineage mapping identifies data flow processes from source systems through various processing stages to storage destinations. Automated tools enable the parsing of SQL queries, procedures, and configuration files utilized in flow processes. However, comprehensive accuracy remains difficult due to runtime generation processes [6]. Manual validation proves essential for understanding implicit dependencies in processes. The assessment evaluates complexity levels within processes using cyclomatic complexity, code measures, data source counts, and join operation numbers. Automated ETL architecture designs increasingly incorporate metadata-driven processes that produce dynamic workflow designs based on specification procedures [6].

Figure 2 below represents the layered migration structure, illustrating systematic flow from legacy systems through assessment, conversion, orchestration, and finally deployment to serverless platforms. Each level utilizes distinct tools and approaches. Automated inventory and dependency mapping are applied during the assessment phase. Conversion employs Apache Spark and templates. Orchestration uses event-based scheduling techniques with fault tolerance facilities. Execution relies on serverless functions with dynamic resource provision. Monitoring and metadata governance components span all levels.

2.2 Complexity Stratification

Workflow processes are classified systematically according to complexity levels based on computational requirements and transformation sophistication. Data integration environments involve diverse data sources ranging from structured databases to semi-structured and unstructured data, all presenting different processing difficulties [7]. Simple workflows involve basic mappings, data-type conversions, and filter processes that consume minimal computational power, completing in seconds to minutes. These qualify for visual development environments.

Moderate complexity transformations involve aggregation logic, lookup operations against reference datasets, and multi-step cleaning procedures. The volume of data handled by these procedures varies between gigabytes and terabytes, with execution times spanning minutes to hours. Distributed computing paradigms efficiently handle moderate data workloads via horizontal scaling and distributed processing across different computation nodes. Highly analytical workflows involve complex statistical computation, machine learning algorithm applications, and joining operations across datasets numbering in the millions [6]. Heterogeneous data source integration adds complexity due to structural and semantic differences, as well as data quality variability across disparate source systems [7].

The classification taxonomy examines other dimensions beyond computational cost, including data sensitivities and business criticality measures. High-priority workflows associated with revenue recognition, regulatory compliance reporting, or real-time operational dashboards receive priority treatment during migration sequencing. The stratification process generates opportunities to eliminate or consolidate workflows involving redundant logic, obsolete reporting needs, or deprecated processes. This creates opportunities for migration scope reduction with improved maintainability of the migrated platform [7].

2.3 AI-Enhanced Classification Mechanisms

Machine learning models extend traditional classification method capabilities by analyzing historical execution metadata, resource utilization trends, and transformation complexity indicators to predict optimal serverless configurations [4]. Supervised learning models trained on business workflow portfolios provide accuracy levels above manual assessment methods while operating in shorter timeframes. Feature engineering extracts useful variables related to metrics like cyclomatic complexity, volume throughput rate, execution time trends, and dependency graphs.

In a reference implementation, workflows are labeled into three main complexity classes (simple, moderate, advanced) and three business criticality levels (low, medium, high) - based on expert review of production jobs from a financial services ETL portfolio. Feature vectors include structural metrics (cyclomatic complexity, number of joins, presence of window functions), runtime metrics (median and p95 execution time, throughput), and operational metrics (historical failure rate, dependency depth). A gradient-boosted tree model trained on this dataset of approximately 500 labeled workflows (indicative value) achieved macro-averaged F1 scores above 0.9 when distinguishing

simple versus advanced workflows, substantially outperforming rule-based baselines.

The classification paradigm leverages artificial intelligence by using an ensemble of decision trees, random forests, and gradient boosting to classify workflows based on various criteria simultaneously: computational complexity, data sensitivity, regulatory constraints, and criticality ratings [4]. The trained models are tested using cross-validation to ensure generalization and avoid overfitting to legacy architecture-specific characteristics.

3. Conversion Methodology and Technical Translation

3.1 Logic Modernization

Legacy procedural code is systematically refactored into cloud-native versions using contemporary coding paradigms and distributed processing infrastructure. It is converted from proprietary coding dialects (like scripting languages specific to legacy ETL systems) into open-source dialects, including Python, Scala, and SQL dialects supporting serverless compute engines. Distributed processing engines, including the Apache Spark framework, enable holistic development platforms combining batch processing, analytical querying, real-time processing, and machine learning tasks [8]. This migration strategy for modernizing legacy code balances code maintainability improvements against migration process efficiency, dependent upon legacy code structure and quality.

Visual development platforms handle simple conversions via graphic interfaces that allow users to create executable code through drag-and-drop actions, defining workflows. Low-code platforms have reduced turnaround time for simple data transfer tasks and basic data cleaning tasks. For complex analytical tasks, custom code development using distributed processing engines divides data across temporary computational clusters. The Apache Spark platform architecture enables petabyte data handling via in-memory computation, fault-tolerant storage, and the latest optimization techniques, including Catalyst query optimization and Tungsten execution engine optimizations [8]. The framework has shown the possibility of achieving up to an order-of-magnitude performance improvement over conventional MapReduce implementations of iterative computations in representative benchmarks [8], which makes it useful for machine learning pipelines and graph computation tasks.

The translation methodology embodies incremental testing techniques with parallel execution of legacy and modernized workflows to ensure output equality. Row-level reconciliation identifies value differences beyond set tolerance limits, typically minimal variability for calculations and zero tolerance for non-numeric data. Performance benchmarking compares performance metrics between legacy systems and serverless technology regarding processing time, with optimized workflows significantly reducing processing time.

3.2 Pattern Recognition and Reuse

Common patterns emerge during conversion analysis, revealing repeated implementations across workflows. The Business Process Model and Notation modeling framework plays an important part in ETL process design through standardized visual representation that maps business requirements to implementations [9]. Data cleansing processes, such as null processing, data format normalization, and outlier identification, form pattern categories that repeatedly appear in workflow portfolios. Aggregation functions, lookups, and slowly changing dimension processing form other high-frequency pattern categories.

Template libraries integrate reusable transformation components and parameterized configurations tuned to fit specific scenarios. Relational algebra provides a mathematical framework for describing ETL procedures, allowing accurate modeling and representation of data manipulation concepts such as selection, projection, join, union, and aggregate operations [9]. Normalized code execution ensures consistency in transformation logic, integrating best practices for error handling, logging, and optimization.

Pattern catalogs record details regarding schemas, transformation routines, and performance attributes of every reusable software component. Metadata-driven generation frameworks automatically generate workflow instances based on configuration files, speeding up deployment time. This abstraction level enables efficient adaptation to schema, regulatory, or business rule variations without code rearrangement [9].

3.3 NLP-Powered Code Translation

Natural language processing application improves legacy code understanding and translation integrity using semantic analysis for undocumented procedural codes. Large language models trained on various programming languages create complete documentation for legacy ETL processes by identifying subtle business and transformation requirements encoded in source code [3].

Documentation automation reduces execution time for processes retained in organizational memory.

The translation engine is implemented as a transformer-based large language model specialized on ETL and SQL code, prompted with paired examples of proprietary scripts and their Spark or SQL equivalents. In the case study across various workflows, approximately 65–75% (indicative value) of legacy steps in low- and medium-complexity workflows were auto-translated and accepted after a single human review pass, with remaining steps either edited or implemented manually. This reduced manual code-writing effort for target Spark jobs by an estimated 30–40% relative to re-implementation (from-scratch) approaches.

Code translation engines utilize natural language processing to establish semantic equivalency between proprietary scripting languages and cloud-native code versions, aligning business logic while synchronizing code for serverless environments [3]. Code transformation engines based on transformers consider the structure and naming conventions of variables and control flow to infer transformation logic and suggest optimal implementation approaches through cloud-native distributed processing engines. Code transformation accuracy is influenced by code complexity and semantic ambiguity associated with business rule definitions.

4. Architecture and Control of Execution

4.1 Event-Driven Scheduling

Serverless orchestration introduces new methodologies, replacing traditional time-based scheduling with event-driven executions based on dynamic responses to system state changes. Event-driven systems reduce wait times associated with data readiness before initiating processes, performing significantly better than traditional fixed-interval time-based systems. Enterprise integration patterns are essential for designing decoupled communication systems based on message passing to facilitate asynchronous processing tasks in dynamic orchestration systems [10]. Triggering events can be based on file receipts, database changes, message queues, or completed upstream tasks. Event-driven systems pursue optimal resource use efficiency by eliminating speculative polling along with idle processing cycles.

The orchestration framework assesses trigger conditions according to defined rules, triggering workflow execution only when prerequisite conditions are fulfilled. Complex event processing engines allow several events to be correlated, enabling scenarios needing data from diverse sources. Conditional branching logic enables dynamic workflows to choose data routes based on characteristics, time windows, or business rule outcomes. Pattern integration architectures identify service-oriented patterns emphasizing interface standardization, and message-oriented patterns entailing asynchronous communication and temporal decoupling [10].

4.2 Resilience Engineering

Orchestration frameworks include efficient fault tolerance mechanisms integrating retry logic, backoff strategies, and circuit breaker designs to address transient failures effectively. Retry strategies enable users to define retry limits and backoff delay times that grow exponentially from initial values to maximum backoff times, avoiding system overload during degraded operations. Distributed database systems require strong fault tolerance mechanisms to manage consistency and availability requirements over geographically diverse sites that may experience network partitions or hardware failures [11].

Alerting mechanisms offer real-time alerts across various channels once failure numbers breach retry limits and performance metrics vary from baselines. Service level agreements monitor execution time, success rates, and data freshness indicators against specified objectives, providing automated reports highlighting compliance percentages. Distributed designs leverage replication techniques, consensus algorithms, and automatic failover solutions to remain operational regardless of individual component failures [11]. Event-driven serverless architecture in ETL application execution shows superior fault-tolerance abilities with dynamic scalability to handle changing workload demands [12].

4.3 Dependency Management

In complex workflows, subtle dependency management is achieved using directed acyclic graphs. Dependency analysis in directed acyclic graphs is used in scheduling engines to search for parallelisms in different workflow branches, executing them simultaneously. In parallel execution plans, the total time required to process complex workflows comprising multiple independently executed transitions is drastically reduced. Dynamic branching enables variable conditional processing dependencies ranging from intermediate processing stage result characteristics to business rule evaluation results. Checkpointing mechanisms enable transformation processing by saving intermediate data at predefined periodic

intervals, allowing restarts from anywhere in the pipeline [11].

5. Limitations

5.1 Metadata Completeness Challenges

Legacy systems often lack complete metadata and technical documentation, including clear data lineage descriptions [13], which complicates migration. Business logic is frequently embedded only in procedural code and institutional memory, creating interpretation challenges during translation.

Automated metadata extraction system success rates vary based on code complexity and documentation standards. Static analysis identifies direct dependencies, but faces challenges analyzing runtime-adjusted SQL queries, dynamic file paths, and decision-driven execution paths. Manual analysis time represents a significant portion of the entire migration process for less documented projects, extending timelines beyond initial estimates. Missing metadata about data quality business rule implementations, error handling processes, and change-over-time justification makes migration tougher, requiring stakeholder interviews and archaeological analysis of deployed environments to rebuild working knowledge. Business intelligence migration projects for cloud-native platforms encounter severe obstacles adapting legacy reporting models, business logic computations, and analytics embedding to new architecture paradigms [13].

5.2 Explanation of Fidelity

Translating legacy logic into serverless programming can introduce potential semantic drift where refactored code produces different semantic functionality despite working syntactically. Complex business rules may not transition cleanly into declarative or distributed computing paradigms for operations depending upon iterative mathematics, stateful transformations, or order-dependent operations. Validation testing is needed to detect equivalent functionality across various data examples and legacy events for the specific code being refactored.

Data provenance tracing becomes highly relevant during migration validation to maintain transformation integrity and traceability during the transition process. Data provenance systems trace total lineage details about how various data artifacts derive from inputs at various processing stages, from inputs through transformed outputs [14]. Organizations report validation processes involve considerable migration project resources regarding disparity identification during various test scenarios through reconciliation processes. Numerical differences occur due to computation precision differences between legacy and modern computation engines in multi-step transforms. Legacy systems with deprecated functions or proprietary algorithms and calculation methods are especially challenging for transformation, requiring new transformation implementation methods [14].

5.3 Operational and AI-Related Risks

Cold starts on serverless functions affect time-sensitive workflows requiring sub-second latencies. Wait times on initial function calls for interpreted language environments cause unacceptable processing latencies. Recent developments in optimal serverless environment usage include utilizing pre-warming, connection pooling, and caching algorithms responsible for reducing cold start latencies [16]. Simultaneous processing capabilities on serverless environments are limited, restricting processing capabilities within specified time limits on function executions per region. Memory processing capability limitations restrict large data processing in single-function executions, prompting data partition usage within multi-stage processing frameworks [13].

Artificial intelligence integration risks pose challenges needing active addressing. Biased syntax can appear in code translation models trained on limited programming language corpora, leading to confusion when understanding domain-related business logic or internal scripting expressions [15]. Semantic drifting in machine learning workflow classification can arise when model training corpora fail to capture typical organization-related ETL process patterns. Explainability challenges in artificial intelligence migration decisions pose validation process difficulties, as administrators struggle understanding reasoning behind these decisions [15].Mitigation techniques include validation frameworks with human verification loops in critical workflow migrations, varied training datasets covering multiple organizational environments, and transparent artificial intelligence techniques explaining decisions. Model retraining with evolving workflow migration outcomes prevents classification drift. Various ensemble techniques using different artificial intelligence methods make models resistant to individual artificial intelligence weaknesses [15].

6. Comparison and Validation of Frameworks

6.1 Comparative Analysis with Existing Frameworks

This framework's uniqueness over existing serverless ETL migration models derives from numerous architectural and methodological breakthroughs. Conventional cloud-native data engineering platforms consider only infrastructure migration without incorporating intelligent automation or artificial intelligence optimization concepts [17]. Past serverless migration models dealt only with code modernization through containerization patterns without considering legacy-related code-level issues for ETL platform migration [5].

The proposed framework advances beyond earlier approaches through artificial intelligence tool integration at different transformation process levels, ranging from natural language processing to automate code documentation and translation, machine learning to enable smart workflow classification, and pattern recognition to develop reusable templates [3,4]. Comparative assessments prove migration process efficiency through reduced manual analysis, accurate classification through machine learning evaluation, and quality improvement through Business Process Model and Notation standardization [9].

Existing methods typically cover workflow orchestration or code translation, but not complete task sets including comprehensive assessment, intelligent transfer, event-driven orchestration, and validation as a single process [17]. Metadata-driven generation tool and automated dependency mapping mechanism utilization differentiate the proposed approach from existing manual migration practices, mostly dependent on institutional experiences and expert opinions.

6.2 Case Study: Enterprise Financial Services ETL Migration

In an anonymized financial services firm, legacy ETL infrastructure for regulatory reporting and risk analysis was migrated to serverless cloud architecture using the proposed framework. The legacy infrastructure handled daily transaction reconciliations, monthly regulatory reports, and quarterly risk analysis determinations through batch processing systems in distributed data centers. The pre-migration environment used fixed-capacity servers handling multiple ETL jobs ranging from straightforward data transfer to complex analysis tasks. The infrastructure showed large underutilization patterns during the idle periods, requiring manual handling. Failed workflow handling required manual efforts. Processing times for crucial task batches exceeded allocated time slots.

There was a substantial reduction in infrastructure cost due to the consumption-based pricing model, which eliminated idle capacity. The total processing time decreased significantly due to parallel processing and elastic scaling that matched resources to workload demand. The workflow success rate improved markedly due to automated retry logic and standardized failure-handling patterns that reduced the need for human intervention. Elastic resource allocation handled volume peaks during quarter-end processing operations without infrastructure changes. The serverless concept allowed sub-hour processing for time-sensitive regulatory operation requirements, which was challenging under batch processing operation model limitations. These results bear testimony to the benefits observed in this case study implementation.

The migration strategy used artificial intelligence-enforced classifications for workflow evaluation, modernized code translations utilizing Apache Spark, and event-based orchestration with robust fault tolerance mechanisms [12]. Natural language processing accelerated documentation creation for undocumented legacy workflows previously requiring extensive reverse engineering processes [3]. Serverless architecture enabled elastic scaling during peak processing periods and optimized resource expenditure during off-peak periods through granular billing mechanisms [2]. Overall inventory analysis and automated dependency mappings established end-to-end workflow documentation. Parallel validation testing ensured functional equality between legacy and modernized workflow versions. The incremental migration plan strategy emphasized high-value workflows associated with critical business functions and ensured coordination among data engineering groups and platform operations [1].

6.3 Quantitative Evaluation

In the anonymized financial services environment, the proposed framework was evaluated over a portfolio of several hundred production ETL workflows spanning regulatory reporting, reconciliations, and risk analytics. The evaluation focused on three dimensions: AI model quality, migration productivity, and operational outcomes after go-live.

For workflow classification, the supervised machine learning ensemble correctly predicted expert-assigned complexity and priority labels in the majority of cases, with accuracy and macro-F1 consistently above 0.9 for primary classes.

Misclassifications were primarily observed at boundaries between moderate and advanced analytical workloads and were caught by human review gates.

For code translation, the natural language processing-based engine produced compilable target code for most simple and moderate transformations, reducing manual implementation effort by roughly one-third when measured in engineering hours per migrated job. Complex analytical jobs still required significant human design but benefited from AI-generated documentation and partial code skeletons.

From an operational perspective, the migrated serverless architecture reduced median daily batch window duration by a substantial margin, as indicated in the case study observations, and lowered infrastructure expenditure due to pay-per-use pricing and elastic scaling. Workflow failure rates also decreased, driven by standardized patterns, automated retry logic, and enhanced observability. These results, although organization-specific and indicative of this particular implementation, offer evidence of the benefits of combining AI-driven analysis with serverless execution.

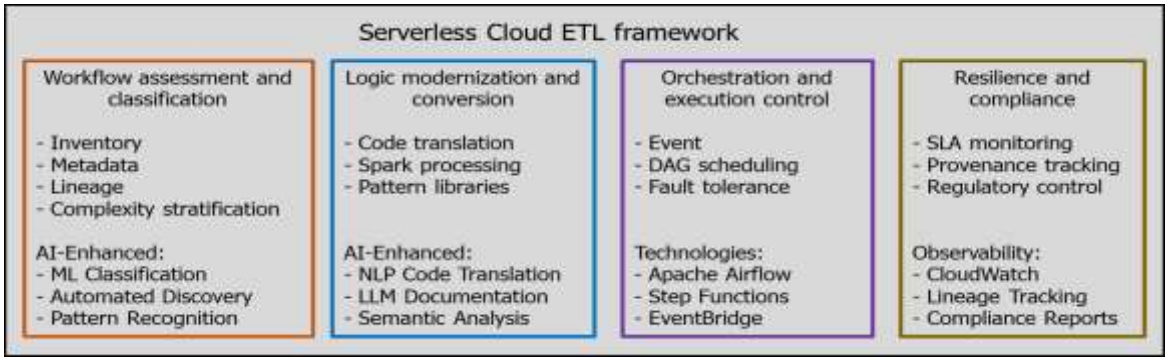


***Figure 1**: 4-Pillar Serverless Cloud ETL Framework [1-4]*

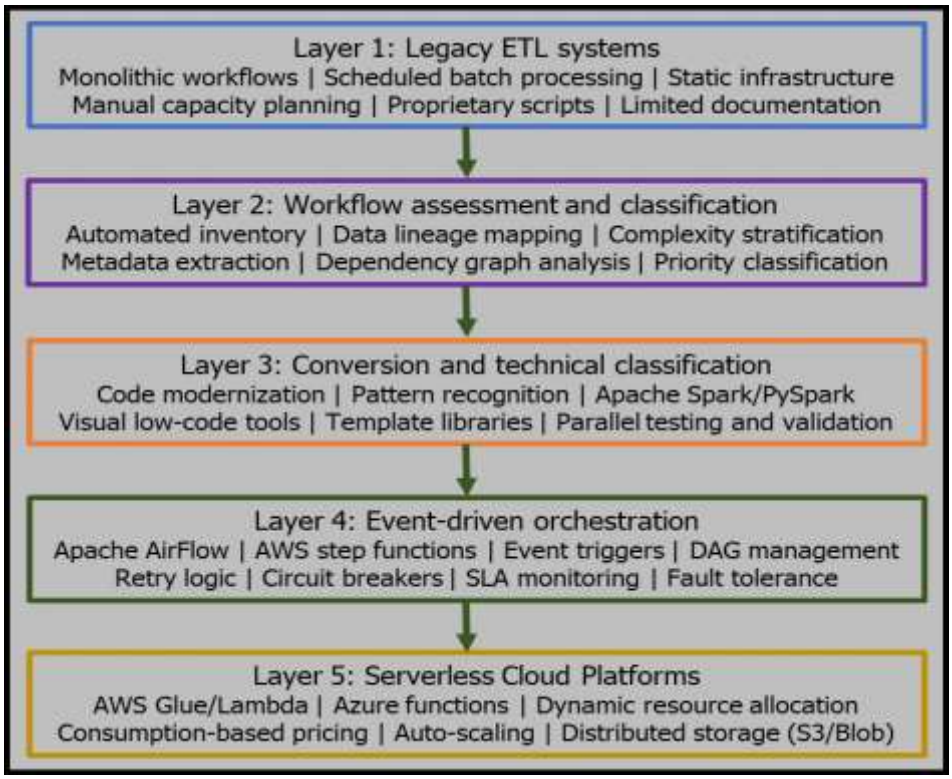


***Figure 2:** Serverless ETL Migration Framework Architecture [6-11]*

***Table 1:** ETL Workflow Classification Framework and Complexity Dimensions [6,7]*

Workflow Attribute	Description
Simple transformations	Basic field mappings, type conversions
Moderate complexity	Aggregation, lookup operations
Advanced analytical	Statistical calculations, ML applications
Data sources	Structured, semi-structured, unstructured

Execution time range	Seconds to hours
Data volume processing	Gigabytes to terabytes
Schema challenges	Heterogeneity, semantic inconsistencies
Metadata-driven generation	Dynamic workflow configuration
Consolidation opportunities	Redundant and deprecated processes
Priority criteria	Business criticality, compliance

Table 2: *Technical translation components and distributed processing capabilities [3,8,9]*

Technology Component	Capability
Apache Spark framework	Unified batch, streaming, ML processing
Performance improvement	Up to an order of magnitude in representative workloads
Dataset scale support	Petabyte-scale processing
Computation model	In-memory distributed processing
Catalyst optimization	Advanced query planning
Tungsten engine	Execution enhancements
BPMN modeling	Visual ETL process representation
Relational algebra	Formal transformation specification
Low-code platforms	Drag-and-drop workflow generation
NLP translation	Automated code modernization
Testing strategy	Parallel legacy-modern execution

Table 3: *Event-driven orchestration architecture and fault tolerance framework [10-12]*

Orchestration Element	Implementation
Trigger mechanisms	File arrival, CDC events, message queues
Event processing	Multiple stream correlation
Integration patterns	Service-oriented, message-oriented
Retry policy configuration	Exponential backoff strategies
Fault tolerance	Circuit breaker patterns
SLA monitoring	Duration, success rates, data freshness
Replication strategies	Distributed node consistency
Consensus protocols	Automated failover mechanisms
Dependency representation	Directed acyclic graphs
Checkpointing intervals	Configurable state persistence

Table 4: *Critical limitations, validation challenges, and mitigation strategies [13-16]*

Challenge Category	Impact	Mitigation Strategy
Documentation deficiencies	Incomplete technical specifications	Automated NLP extraction
Business rule embedding	Procedural code without documentation	LLM documentation generation
Reverse engineering effort	Extensive manual code inspection	AI-assisted pattern recognition
Static code analysis	Dynamic SQL generation struggles	Runtime profiling tools
Semantic drift risk	Functional divergence in translation	Parallel validation testing
Provenance tracking	Complete lineage documentation	Automated tracking systems
Validation resource consumption	Substantial project effort	AI-powered test generation
Numerical precision differences	Rounding variation accumulation	Precision threshold monitoring
Cold start latencies	Sub-second response impact	Pre-warming optimization
Memory allocation limits	Dataset partitioning requirements	Intelligent data sharding
AI translation bias	Misinterpretation of domain logic	Diverse training datasets
Classification drift	Suboptimal configuration recommendations	Regular model retraining

4. Conclusions

Legacy ETL process migration to serverless cloud platforms enables organizations to overcome traditional batch infrastructure limitations by combining event-driven execution, elastic scaling, and consumption-based pricing with distributed processing engines. The proposed framework contributes a structured, AI-enabled lifecycle covering inventory assessment, workflow

stratification, NLP-assisted documentation and translation, pattern-driven implementation, and event-based orchestration with built-in resilience. The case study and quantitative observations illustrate that this approach can reduce batch windows, lower infrastructure costs, and increase workflow success rates while decreasing manual effort for classification and code migration. The framework explicitly addresses metadata gaps, semantic drift, and AI-related bias through provenance tracking, human-in-the-loop validation, and model governance practices. Future directions include incorporating artificial intelligence-driven code translation tools for improved automation accuracy, developing schema evolution frameworks supporting zero-downtime migrations, and extending support for real-time streaming architectures with self-learning algorithms that continuously optimize performance based on operational patterns [18]. The framework provides replicable methodologies for financial services, healthcare, telecommunications, and data-intensive industries seeking scalable, cost-efficient, compliant data platform architectures supporting agile development and real-time analytics capabilities.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Disclaimer:** This work represents the author's views and does not reflect the policies or positions of HCL America Inc.

References

[1] O. Ogunwole et al., "Modernizing Legacy Systems: A Scalable Approach to Next-Generation Data Architectures and Seamless Integration", *International Journal of Multidisciplinary Research and Growth Evaluation*, 2023. [Online]. DOI: https://doi.org/10.54660/.IJMRGE.2023.4.1.901-909

[2] N. Syeda et al., "Analysis of cost-efficiency of serverless approaches", *arXiv*, Jun. 2025. [Online]. DOI: https://doi.org/10.48550/arXiv.2506.05836

[3] D. Chanda, "Automated ETL Pipelines for Modern Data Warehousing: Architectures, Challenges, and Emerging Solutions", *The Eastasouth Journal of Information Systems and Computer Science*, 2024. [Online]. DOI: doi.org/10.58812/esiscs.v1i03.523

[4] I.M. Putrama and P. Martinek, "Heterogeneous data integration: Challenges and opportunities", *ScienceDirect*, 2024. [Online]. DOI: https://doi.org/10.1016/j.dib.2024.110853

[5] M. Zaharia et al., "Apache Spark: A unified engine for big data processing", *ACM Digital Library*, 2016. [Online]. DOI: https://doi.org/10.1145/2934664

[6] J. Awiti et al., "Design and implementation of ETL processes using BPMN and relational algebra", *ScienceDirect*, 2020. [Online]. DOI: https://doi.org/10.1016/j.datak.2020.101837

[7] S. Aier and R. Winter, "Fundamental Patterns for Enterprise Integration Services", *IGI Global Scientific Publishing*, 2010. [Online]. DOI: https://doi.org/10.4018/jssmet.2010010103

[8] S.R. Chigurupati, "Distributed Database Systems for Scalable Enterprise Applications: A New Paradigm", *IJSAT*, Mar. 2025. [Online]. DOI: https://doi.org/10.71097/IJSAT.v16.i1.2795

[9] T.T. Bukhari et al., "Cloud-Native Business Intelligence Transformation: Migrating Legacy Systems to Modern Analytics Stacks for Scalable Decision-Making", *IJSRHSS*, 2024. [Online]. DOI: https://doi.org/10.32628/IJSRSSH242763

[10] M.M. Alam and W. Wang, "A Comprehensive Survey on the State-of-the-art Data Provenance Approaches for Security Enforcement", *arXiv*, 2021. [Online]. DOI: https://doi.org/10.48550/arXiv.2107.01678

[11] C. Diggs et al., "Leveraging LLMs for Legacy Code Modernization: Challenges and Opportunities for LLM-Generated documentation", *arXiv*, 2024. [Online]. DOI: https://doi.org/10.48550/arXiv.2411.14971

[12] A. Awasthi and A. Vaidya, "ETL Pipeline Integration for Machine Learning-Based Product Classification: a Comprehensive Guide", *IJARET*, Mar.-Apr. 2025. [Online]. DOI: https://doi.org/10.34218/IJARET_16_02_006

[13] R. Krasniqi et al., "SE Perspective on LLMs: Biases in Code Generation, Code Interpretability, and Code Security Risks", *ACM Digital Library*, 4th Dec. 2025. [Online]. DOI: https://doi.org/10.1145/3774324

[14] S. Metla, "Powering America's Digital Future: Big Data Migration and ETL Modernization for Scalable Intelligence", *Sarcouncil Journal of Engineering and Computer Sciences - Zenodo*, Jul.

2025. [Online]. DOI: https://doi.org/10.5281/zenodo.15870392

[15] S.K. Rai, "Demystifying Cloud-Native Data Engineering Architectures", *IJITMIS*, Mar.-Apr. 2025. [Online]. DOI: https://doi.org/10.34218/IJITMIS_16_02_062

[16] A. Pogiatzis and G. Samakovitis, "An Event-Driven Serverless ETL Pipeline on AWS," *MDPI*, 2020. [Online]. DOI: https://doi.org/10.3390/app11010191

[17] C. Lou et al., "HydraServe: Minimizing Cold Start Latency for Serverless LLM Serving in Public Clouds", *arXiv*, Sep. 2025. [Online]. DOI: https://doi.org/10.48550/arXiv.2502.15524

[18] S. Singamsetty, "Accelerating data engineering efficiency with self-learning AI algorithms", *International Journal of Computing and Artificial Intelligence*, Feb. 2025. [Online]. DOI: https://doi.org/10.33545/27076571.2025.v6.i1c.154