International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(10s) 2026 pp. 467-481
https://doi.org/10.70917/ijcisim-2026-3629
Article

Real-Time Financial Data Validation Platform for Enterprise Accounting Systems

Vishram Singh

Independent Researcher, USA

Abstract: Enterprise accounting systems generate large volumes of financial transaction data, consisting of journal lines, subledger postings, and end-of-period adjustments, subject to accounting rules, dimensional hierarchies, and regulatory requirements. Classic enterprise resource planning (ERP) systems customarily perform financial validation synchronously within transaction processing workflows, leading to performance bottlenecks as transaction volumes grow. These bottlenecks delay transaction processing, reduce throughput, and increase business and operational risks during peak processing periods.

This article proposes a Real-Time Financial Data Validation Platform (RTFDVP) for use in enterprise settings that improves the performance, scalability, and reliability of validation processes. It leverages microservices and event-based messaging technology for asynchronous validation by decoupling validation logic from the core ERP system and sending validation jobs to a distributed validation engine. Features include account verification with rules, financial dimension and accounting calendar verification, threshold-based approval routing, and anomaly detection and alerting, all designed in a modular, horizontally scalable architecture. Evaluation with synthetically generated enterprise financial workloads shows RTFDVP improves validation throughput by roughly an order of magnitude (9.5x), reduces the ERP CPU utilization by 45%, and reduces the average user-perceived validation latency from 8 seconds to 1.5 seconds. RTFDVP provides a general, extensible architecture for modernizing financial validation infrastructure in large-scale enterprise environments. RTFDVP also provides a path towards smart anomaly detection and predictive validation.

Keywords: Financial Systems, Enterprise Resource Planning Data Validation, Distributed Systems, Event-Driven Architecture, Microservices, Accounting Automation Asynchronous Processing

1. Introduction

Enterprise accounting systems constitute the majority of enterprise systems. They allow companies to record, classify, and audit enterprise financial transactions in accordance with organizational, regulatory and statutory rules and standards. Modern companies, particularly multinational corporations, need enterprise resource planning (ERP) systems such as Microsoft Dynamics. 365 Finance, SAP S/4HANA, and Oracle Fusion Financials maintain the integrity of their financial records across many geographies, business entities, and currencies [1].

The volume and complexity of financial transactions have risen considerably in the last decade. The reasons for this include the digitalization of the procure-to-pay and order-to-cash supply chains, real-time networks for payment, a rise in regulatory requirements for financial reporting, and continuous accounting where journal entry activity is spread throughout the period as opposed to being concentrated at period-end [2]. This has resulted in ERP systems having to support transaction streams of high frequency and volume with various levels of constraints.

In an enterprise ERP system, the financial transactions subject to validation may include journal entries, accounts payable invoice postings, accounts receivable remittance applications, fixed asset acquisition and depreciation adjustments, intercompany eliminations, and period-close accruals. In a transacting ERP system, a validation pipeline is applied to each of these transactions, which may include account existence checks, dimensional combination checks, accounting period status checks, authorization and approval threshold checks, and custom, internally defined business rules [3].

In a customary ERP implementation, this validation pipeline is part of the synchronous ERP workflow. Each time a user submits a journal entry or a scheduled process uploads a batch of transactions, the ERP system must

complete all the validations before returning a response. The synchronous relationship between transaction submission and validation execution forms a fundamental scalability bottleneck, as the transaction volume directly impacts the validation latency, while the amount of database I/O and application server CPU requests from multiple simultaneous ERP users and automated batch runs determine the limited capacity of the system.

The negative consequence of this limitation is that the financial controllers may have to wait for a while to hear whether the iXBRL journal batches they uploaded were valid, holding up the period close. ERP systems that are operated by system administrators may have to deliberately provide spare capacity for verification loads at month-end and quarter-end periods. Automated integration pipelines that feed transactions from upstream payroll, procurement and treasury may experience unpredictable latencies at peak periods. Thus, inefficiencies can considerably impact enterprise financial operations [4].

Based on the aforementioned statement of problems, this article proposes a Real-Time Financial Data Validation Platform (RTFDVP) based on an architectural separation of concerns where the financial data validation pipeline is separated from the ERP transaction submission workflow and the validation task is routed to a distributed asynchronous processing layer [5]. To maintain semantic equivalence with ERP validation logic, the platform consumes reference data from the ERP system's reference data layer (e.g., chart of accounts, dimension hierarchies, accounting calendar states, and approval thresholds) in order to produce semantically equivalent decentralized validation results [6].

This article makes two contributions: a distributed architecture to decouple the business logic for validating financial transactions from the logic used to process calls from ERP application servers and a rule-based validation engine that encapsulates account, dimension, period and business rule checks into independent components [7]. Third, we develop an event-driven validation workflow to enable asynchronous processing and real-time reporting of the validation result without blocking the submission of the ERP transaction [8]. Finally, we present a performance evaluation on simulated enterprise workloads to highlight the throughput and latency benefits of the proposed architecture over customary synchronous ERP validation approaches.

In the remainder of this article, Section 2 provides a review of financial data validation in ERP systems and other similar approaches to distributed processing. The RTFDVP architecture is described in section 3, the RTFDVP financial validation process in section 4, the validation rule engine in Section 5, and the distributed processing engine in Section 6. Section 7 describes ERP integration. Section 8 describes the experimental evaluation. Section 9 describes the results. Section 10 discusses implications and limitations. Future work is discussed in Section 11.

2. Background and Related Work

2.1 Financial data validation in ERP systems

ERP systems most commonly use multi-tier quality validation pipelines activated at the point of transaction input (manual posting) and batch upload. These are implemented as part of the ERP application tier architecture and executed synchronously within the posting transaction workflow.

Document validation checks in SAP S/4HANA relate to the Financial Accounting (FI) module. The FI module verifies the substitution and document validation rules defined in the Customizing framework [1]. The checks are enforced based on the data contained in the document header or line item, such as account assignment combinations, cost objects, or posting periods. Oracle Fusion Financials uses a similar concept with Account Rules and Journal Approval Workflows. The workflow validates a ledger combination and applies approval steps to journal entries before posting the transaction to the ledger [3]. In Microsoft Dynamics 365 Finance, Account Structures and Advanced Rules work together to define valid combinations of dimensions per main account [5].

On all platforms, validation is tightly coupled to the ERP application server, which acts as a lookup table for reference data (chart of accounts, dimension hierarchies, and period status). Because of this, validation throughput is limited by the capacity of the ERP application server and the performance of the queries to the ERP system database.

2.2 Problems with Customary Synchronous Validation

The synchronous invocation of validation logic during the workflow of the ERP transaction introduces several design limitations, especially at the enterprise scale.

First, synchronous processing establishes a dependency between the submission of a transaction and its validation, forcing both users and automated processes to wait for the validation pipeline to complete. While such

latency is acceptable for individual transactions, big batch uploads of thousands of journals (generated by payroll systems, fixed asset modules, or consolidation engines) can take minutes to validate all the journals, and this may be blocking downstream processes and user interfaces within the specific modules themselves [4].

Secondly, the validation logic is tightly bound to the application tier of the ERP. ERP application servers are stateful entities that maintain session context, database connection pools, and application locks. However, adding application server capacity to achieve more throughput in terms of validation may require more licensing, more provisioning, and more database connections, which do not scale linearly with transaction volume [6].

Third, synchronous ERP validation loads the same infrastructure that ERP uses to execute other functions. In a month-end close, for instance, payroll runs and consolidation cycles compete with validation for the same CPU and I/O resources used by financial reporting, user interface construction, and background batch processing [2].

2.3 Distributed and Event-Driven Processing Architectures

Motivated by the limitations of synchronous, centralized processing, distributed and event-driven architectures are widely used in enterprise computing. In event-driven architectures (EDAs), data producers and consumers are decoupled from each other through asynchronous messages, allowing high-throughput, event-streaming applications to be easily scaled horizontally by starting new consumer containers [7].

Apache Kafka provides persistent ordered message delivery, horizontal scalability, and consumer group semantics [8]. Cluster management in Kubernetes, which is an elastic container orchestration platform, enables dynamic scaling of processing workloads in accordance with background queue depth and processing latency feedback signals that are found in variable-throughput workloads [9].

Microservice architectures complement the event-driven pattern , by allowing the application's monolithic logic to be encapsulated in distinct, independently deployable services with clear interfaces. This allows the validation rules to be independently changed, scaled, and monitored and the validation pipeline to be easily refactored and evolved over time without disrupting the overall system [10].

2.4 Financial Data Quality and Validation Research

Data quality research, particularly in financial services, has long referred to the benefits of preventing data quality problems by validating data as it is entered into the appropriate system. Research on enterprise data warehouses in the financial services industry has demonstrated that when validation errors are not caught until a later stage such as a reconciliation or audit, remediation costs are considerably higher [11].

Automated rule-based validation frameworks have been explored extensively in the context of data warehousing and ETL (extract-transform-load) pipelines, where declarative rule engines reduce the implementation and maintenance costs of validation rules as compared to procedural code implementing the rules [12]. Recent research on continuous accounting, which seeks to spread period-close processes across the whole fiscal period, has identified the need for validation tools that can maintain high throughput continuously and not just during batch processing windows [2].

Other than the above studies, however, few have proposed a solution to the architectural challenge of decoupling financial transaction validation from the enterprise application tiers of enterprise resource planning at the enterprise scale, which is what the RTFDVP architecture in this article seeks to address [13].

3. System Architecture

3.1 Architectural Overview

The RTFDVP validation layer is an external service that runs separately from the core of the ERP application. It intercepts and validates financial transactions before posting on the ERP application. It runs a pipeline of validation rules by calling a distributed processing engine. Its architecture is based on three principles: separation of validation from ERP application logic, asynchronous event-driven processing, and modular rule encapsulation.

The platform consists of the following seven components: the User Interface Layer, the Validation API Gateway, the Validation Rule Engine, the Distributed Processing Engine, the Financial Reference Data Service, the Event Streaming Layer, and the ERP Integration Module. Each component has its own purpose, and interactions between components are defined in documented API contracts and event schemas allowing independent development and scaling of components.

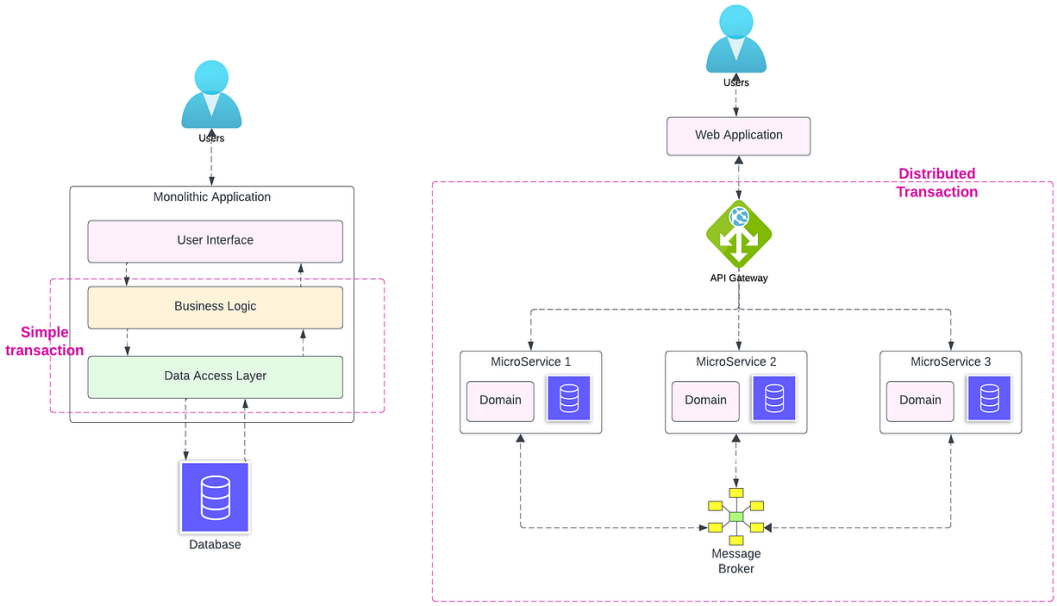

Figure 1: System Architecture Diagram

3.2 User Interface Layer

For financial users such as accountants, controllers, and finance system administrators, the User Interface Layer includes a web-based front-end to request validations, see the status of validation jobs, see the results of validations, and manage configurations of validation rules. The layer is designed to support both single transactions and batch upload scenarios.

For high-volume batch uploads, the interface exposes a file ingestion endpoint, which accepts structured transaction files in standard formats (CSV, XML, or JSON) and returns an immediate acknowledgement and a job tracking ID upon receipt of the file. The actual validation work is decoupled from the user experience. In this case, the user can see the progress of the batch validation in real time and get a notification when the validation is complete, as opposed to waiting for the synchronous ERP validation.

3.3 Validation API Gateway

The Validation API Gateway is the only entry point for validation requests from the User Interface Layer, the automated integration pipelines, and direct API consumers. It is a stateless RESTful service that handles authentication and authorization for requests, request parsing, schema validation, and forwarding the requests to the Event Streaming Layer.

The gateway performs a light pre-validation step, or structural schema checks, on incoming transaction data. It then publishes requests onto the event stream. Pre-validation rejects malformed requests at the perimeter and prevents wasting downstream processing power on invalid payloads that can no longer be resolved later. The gateway also provides rate limiting and back-pressure signaling to protect the processing pipeline from being overwhelmed by upstream clients.

3.4 Validation Rule Engine

The Validation Rule Engine is the engine that encodes the financial validation rules that are applied to each transaction. The engine is implemented as a set of stand-alone executable rule modules that each encode a specific validation category: account existence, dimensional combination, accounting period status, and business rules. The rule modules take in transaction data and reference data and produce structured validation result records.

Because of the rule engine's modularity, rules can be added or modified in the future without affecting the rest of the validation pipeline. New rule types such as intercompany balance validation or regulatory reporting compliance

checks can be added as modules and can be configured without affecting other modules or the orchestration. Rules are managed in a rule repository that supports versioning, which allows rules to be rolled out in stages or rolled back if misconfigured.

3.5 Distributed Processing Engine

The Distributed Processing Engine is the runtime environment for validation worker instances. It consists of stateless processing units that read transaction events from the Event Streaming Layer, invoke the corresponding rule modules from the Validation Rule Engine, and publish the validation result back to the event stream. Because the workers are stateless, new worker instances can be dynamically added in response to queue depth signals to provide horizontal scalability without the need for coordination or state transfer.

The worker instances run as containerized microservices in a Kubernetes cluster and can be automatically scaled based on metrics such as queue depth, processing latency, and CPU utilization. The processing engine is fault tolerant. When a worker instance fails during transaction processing for an event that has not been acknowledged, the event is redelivered to another available worker instance.

3.6 Financial Reference Data Service

The Financial Reference Data Service maintains an up-to-date cache of the various reference data required by the validation rule modules, such as the chart of accounts, financial dimension hierarchies, dimension combination validity matrices, accounting period statuses, and business rules definitions. This prevents the workers from querying the ERP system's database with every validation, which would have re-introduced the ERP database as a bottleneck.

The reference data cache is repopulated via a change-data-capture (CDC) integration with the reference data tables of an ERP system. Any changes to the chart of accounts, dimension structure or period statuses are cached within a configurable propagation window, typically seconds to low tens of seconds. The service exposes reference data to validation workers via a high-performance in-memory data store, enabling sub-millisecond reference data lookups per validation check.

3.7 Event Streaming Layer

An Event Streaming Layer implements the asynchronous messaging layer to connect the three microservices: the validation API Gateway, Distributed Processing Engine and the ERP Integration Module. The Event Streaming Layer is implemented using a distributed log platform, such as Apache Kafka, which provides durable, ordered delivery, along with configurable retention and consumer group semantics.

The transactions published from the Validation API Gateway are processed in parallel by several validation worker instances, which process multiple transactions concurrently within the batch. The validation results are consumed by a set of result aggregation services that compare the generated results against the original batch, compute batch-level validation summaries, and perform downstream actions such as notifying the user on batch completion and forwarding transactions to the ERP Integration Module upon validation.

4. Financial Validation Workflow

4.1 Workflow Overview

The validation workflow controls the entire lifecycle of the financial transaction, i.e., from submitting to posting the transaction to the ERP. To keep the throughput as high as possible, validation processes are run asynchronously and in parallel wherever possible, while not violating the order of the validation rules for a transaction that is not dependent on any other transaction.

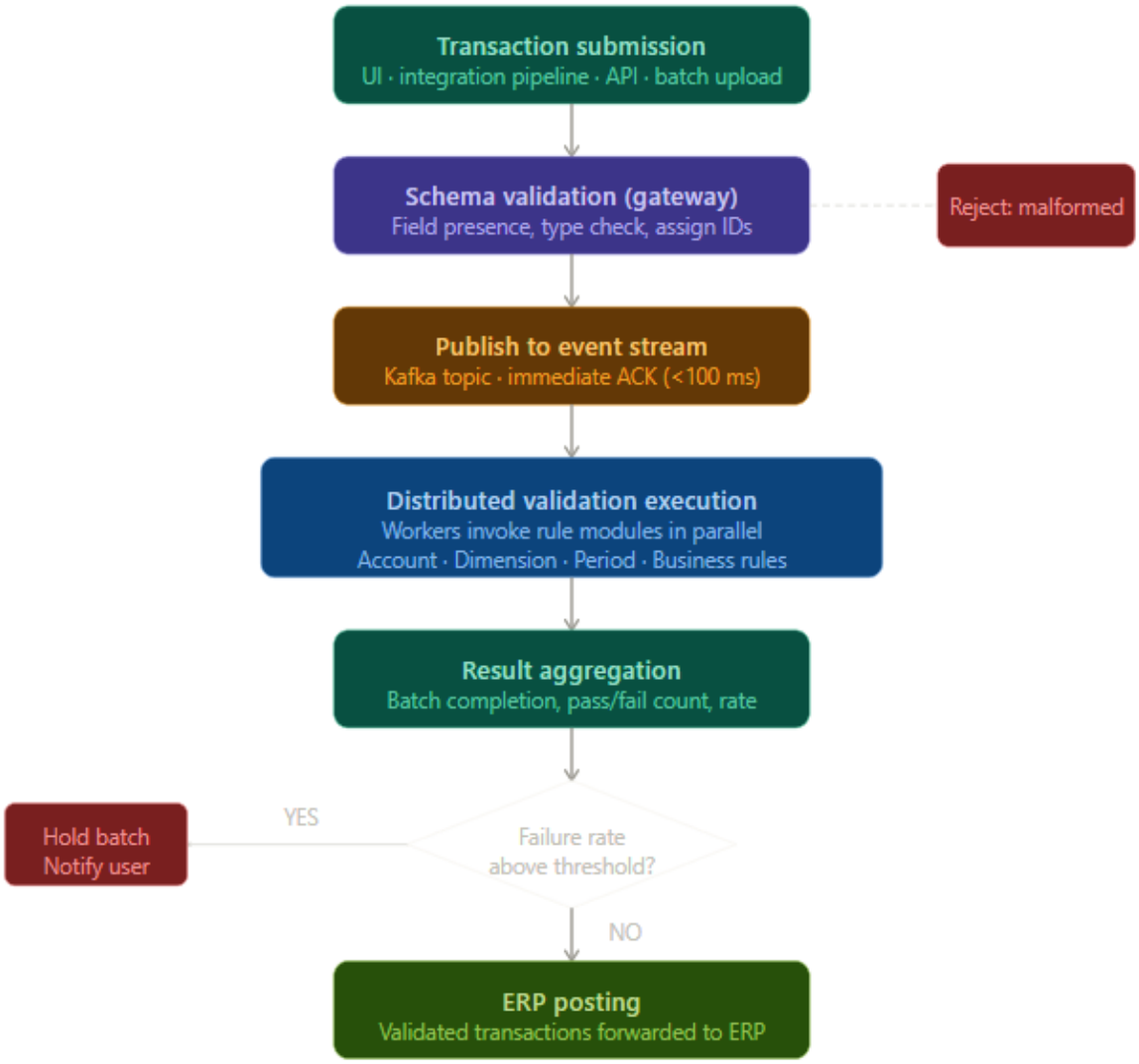

Figure 2: Validation Workflow lifecycle

4.2 Transaction Submission

Validation begins by submitting a financial transaction (or a batch of transactions) to the Validation API Gateway. This can be initiated by a user via the User Interface Layer, set up by automated integration pipelines that pass down transactions from upstream systems in the flow of data, or through API calls by third-party applications that are hooked up to the RTFDVP.

The gateway starts by performing schema validation, ensuring that required fields are present and of the correct type. A transaction ID and batch ID are assigned, and the transaction event is published to the Event Streaming Layer. An acknowledgment is returned to the submitting client with the batch ID and the expected processing duration.

Furthermore, the acknowledgement response is immediate (within 100 milliseconds), regardless of transaction volume, as opposed to the blocking wait to complete ERP validations in synchronous workflows.

4.3 Distributed Validation Execution

Validation worker instances read transaction events from the pending validation topic proposed by the Event Streaming Layer. Each worker invokes the validation rule modules in the preconfigured order for each transaction

event. Each validation rule module produces a result showing whether the transaction event passes or fails validation and a machine-readable error code and human-readable error message if the event was invalid.

All the reference data calls made by rule modules are done via calls to the Financial Reference Data Service, whose in-memory sub-millisecond response times make rule logic the largest component of per-transaction processing time for practical purposes. This allows very high throughput rates to be achieved per worker.

Once all rule modules have evaluated the transaction, the worker publishes a validation result event to the Event Streaming Layer, including the transaction ID, the pass/fail outcome, and the evaluation result of each rule. The worker then acknowledges the consumed transaction event, indicating that the transaction has been processed, and removes it from the pending events queue.

4.4 Result Aggregation and Notification

These events are consumed by result aggregation services, which calculate and persist the total number of validations completed for the batch. Once the transactions in the batch have all been processed, a completion event is published to the user's or system's notification channel. This event contains the complete number of transactions in the batch, the number of passed and failed transactions, and the failure rate.

When the failure rate exceeds a configurable threshold, the aggregation service can stop any further ERP forwarding of that entire batch and wait for user approval so that partial posting of batches that fail due to systemic validation exceptions does not occur. When the failure rate is below the threshold set, the successful transactions are forwarded to the ERP Integration Module for posting.

4.5 ERP Posting

Transactions that pass all the validation processing are passed to the ERP Integration Module. The module converts the validated transaction data to the format required to invoke the integration interface of the ERP system and passes the transactions to the ERP system for posting. Since the transactions being passed to the ERP system for posting have already been fully validated by the RTFDVP, the validation layer of the ERP system can post the transactions immediately.

5. Validation Rule Engine

5.1 Rule Engine design principles

The Validation Rule Engine is based on the principle of declarative rule encapsulation, i.e., each validation category is implemented as a self-contained rule module with a common invocation interface to enable the orchestration layer to invoke a rule module without requiring knowledge of its underlying implementation. As a result, the logic of rules can be changed independently without requiring changes to the orchestration layer or other rule modules, e.g., changing account validation logic for a newly activated account or deactivated account ranges.

Rules are written using a rule definition language. Each rule consists of a condition and an action. The condition and action parts of a rule specify the conditions under which an error is reported (from a validation failure) and the error code, severity level, and message that will be returned when the condition is satisfied. This separation enables non-technical rule administrators to create and modify rules through the configuration interface rather than through the application code.

5.2 Account Validation

Account validation checks whether the ledger account that is specified in each line item in the transaction exists in the chart of accounts and whether it is valid on the date when the transaction is posted. The rule module checks the Financial Reference Data Service's (FRDS) chart of accounts cache for the following.

The first existence check checks if the account identifier exists in the chart of accounts. If the account identifier does not exist, the transaction returns a critical failure status since it cannot be posted to any ledger:

IF account_id NOT IN chart_of_accounts

THEN FAIL: ERROR_CODE = "ACC_001," MESSAGE = "Account does not exist in the chart of accounts."

Active account validation also verifies that the account is not blocked or deactivated on the date the transaction is posted. Accounts can be deactivated on an effective date, and any transaction posted to a deactivated account must be rejected:

IF account_status is not = ACTIVE AS OF transaction_date

THEN FAIL: ERROR_CODE = "ACC_002", MESSAGE = "Account is inactive for the specified posting date"

The account type check tests whether the transaction type is a valid match for the account type (balance sheet, profit and loss, or statistical) defined for the corresponding journal line. For instance, a journal entry that credits a revenue account with a debit posting type may indicate data entry errors:

IF account_type NOT IN permitted_types FOR transaction_type

THEN WARN: ERROR_CODE = "ACC_003"; MESSAGE = "Account type inconsistent with transaction type."

5.3 Financial Dimension Validation

Validating financial dimensions means checking that the dimensional combination specified for each line item is one of the allowed dimensional combinations in the organization's dimension hierarchy. Cost centers, departments, projects, business units, etc. provide the analytical granularity to financial reporting. Invalid combinations of these dimensions yield data quality failures, which are propagated through reporting pipelines.

The dimension combination check validates the dimension values in the transaction against those in the Account Structures and Advanced Rules synchronized from the ERP system:

IF (main_account, cost_center, department) NOT IN valid_dimension_combinations

THEN FAIL: ERROR_CODE = "DIM_001," MESSAGE = "Dimension combination is not permitted for this account."

The dimension existence check phase verifies that the dimension values specified in the request exist and are active within their dimension hierarchy.

IF cost_center NOT IN active_cost_centers

THEN FAIL: ERROR_CODE = "DIM_002," MESSAGE = "Cost center does not exist or is inactive."

If an organization requires dimensions for some accounts, the dimension completeness check verifies if all these dimensions are filled:

IF required_dimension IS NULL

THEN FAIL: ERROR_CODE = "DIM_003," MESSAGE = "Mandatory dimension value is missing."

5.4 Accounting Period Validation

Accounting period validation ensures that the transaction posting date is within the limits of an open accounting period in the ledger that contains the transaction. Period statuses are maintained in the Financial Reference Data Service's period calendar cache and updated by the CDC integration when financial administrators open or close accounting periods.

The period status check is the primary period validation rule:

IF accounting_period_status(posting_date, ledger_id) IS NOT = OPEN

THEN FAIL: ERROR_CODE = "PER_001", MESSAGE = "Accounting period is closed or does not exist for the specified posting date."

For organizations that provision for a future period, the future period check determines if future-dated transactions can exist:

IF posting_date > current_period_end AND future_period_posting is not = PERMITTED

THEN FAIL: ERROR_CODE = "PER_002," MESSAGE = "Future period posting is not permitted."

5.5 Business Rule Validation

Business rules validation defines the organization-specific rules not covered by the list of account, dimension, and period rules. Each organization or industry has a different set of rules, making it one of the main customizations of enterprise ERP during the implementation stage.

The approval threshold rule enforces segregation of duties and financial authorization limits by triggering an approval workflow for transactions above certain values.

IF ABS(transaction_amount) > approval_threshold(account_id, cost_center)

THEN REQUIRE_APPROVAL: ERROR_CODE = "BUS_001", MESSAGE = "Transaction exceeds authorization threshold; approval workflow initiated."

With the intercompany balance rule, intercompany journal entries are posted in such a way that they net to zero.

IF SUM(intercompany_line_amounts) = 0

THEN FAIL: ERROR_CODE = "BUS_002," MESSAGE = "Intercompany journal entry is out of balance."

The duplicate detection rule identifies transactions that have the same key attributes: date, account, amount, and reference.

IF transaction MATCHES existing_transaction ON (date, account, amount, reference)

THEN WARN: ERROR_CODE = "BUS_003," MESSAGE = "Potential duplicate transaction detected."

6. Distributed Processing Engine

6.1 Scalability Design

The requirements from the throughput for the distributed processing engine must include elastic scaling capabilities to handle the intrinsically volatile nature of financial transactions. During period close cycles, peak throughput requirements can be an order of magnitude higher than the average daily transaction volume. To deal with peak loads and to reduce resource costs, the Distributed Processing Engine must be able to scale processing power out and in.

Elasticity is managed using the Kubernetes Horizontal Pod Autoscaling (HPA). The HPA configuration scales the worker deployments based on event stream queue depth metrics that are exposed from the Apache Kafka cluster by an HPA metrics adapter. HPA will scale up the number of replicas of worker pod instances when the depth of the input queue grows beyond a high-water threshold and scale down the number of replicas when the depth of the input queue falls below a low-water threshold. Demand-based scaling guarantees a bounded latency for processing any input transaction rate within the scale range specified.

6.2 Fault Tolerance and Message Delivery

The distributed processing engine is built with at-least-once message delivery semantics. To avoid message loss of transaction events in case of worker failure, workers consume transaction events and confirm successful processing by writing back a validation result to the Kafka topic. If the worker fails after processing the event but before it publishes the processed result, the Kafka consumer group coordinator recognizes the failure and reassigns the event to another worker instance for processing.

To ensure idempotence of the result processing, since reprocessed events might result in result events being published multiple times, the result aggregation service de-duplicates the result events by transaction identifier before incrementing batch completion counts. This ensures that each transaction completion is only counted once per batch summary regardless of how many times the validation result event is published.

6.3 Technology Implementation

The Distributed Processing Engine may be implemented using a variety of technology stacks based on the infrastructure preferred by the organization. The reference implementation uses Apache Kafka, a stream preprocessor that provides durable log storage, configurable retention, and consumer group parallelism. Validation workers are Java or Python services packaged in containers and deployed on Kubernetes. Worker scaling is controlled via Kubernetes.

Horizontal Pod Autoscaler (HPA), based on Kafka queue depth metrics exposed by the Kubernetes Event-Driven Autoscaling operator (KEDA).

The Financial Reference Data Service uses Redis as an in-memory data store to provide sub-millisecond reads of account, dimension, and period reference data to performance-critical applications. Redis Sentinel or Redis Cluster provides high availability of the reference data cache, freeing worker instances from waiting for the reference data service to be available.

7. ERP Integration

7.1 Integration Architecture

This Integration Module connects the RTFDVP to the ERP system. It synchronizes reference data between the ERP and the Financial Reference Data Service and sends validated transactions from the RTFDVP to the ERP system for posting.

The net effect of this integration architecture is to minimize the dependencies of the RTFDVP implementation on a particular ERP platform, allowing the platform to run against multiple ERP deployments with minimal changes. The integration adapters for individual ERP platforms encapsulate their particular API calls, data transformations, authentication, etc. to provide an abstraction.

7.2 Reference Data Synchronization

Reference data synchronization is a process that keeps the Financial Reference Data Service caches synchronized at all times with the authoritative reference data within the ERP system. This is accomplished by using change-data-capture (CDC) technology to pick up changes to the ERP reference data tables of the chart of accounts, dimension values, dimension combination rules, period statuses, and business rule configurations and pushing these changes to the reference data cache.

For ERP systems that expose reference data via OData/REST APIs, the synchronization service polls the respective APIs at a configured polling frequency, fetching the updated records and caching them. For ERP systems that expose change notifications upon updates to reference data (e.g., SAP Change Document interface or the Microsoft Dynamics Business Events framework), the synchronization service subscribes to these change notifications to keep the cache up-to-date in near real-time.

7.3 Validated Transaction Forwarding

Once validated, the transactions are sent to the ERP system using the standard ERP transaction ingestion API. For Microsoft Dynamics 365 Finance, this means generating journal lines in the General Journals data entity schema and sending them to the Data Management Framework REST API [5]. For SAP S/4HANA, approved transactions are posted as IDoc or BAPI objects over the SAP integration layer [1]. For Oracle Fusion Financials, Journal Import is used as the interface to send transactions via Oracle Integration Cloud [3].

Since all transactions sent to the ERP system from the RTFDVP have already been validated, in the nominal case it is expected that such transactions will pass through an additional layer of validation without error within the ERP system. The integration module must also handle the edge case where a transaction is rejected by the ERP system after being approved by the RTFDVP (most commonly, because the two reference data systems are not in sync), sending the rejected transactions back to the RTFDVP for revalidation and user notification.

8. Experimental Evaluation

8.1 Experimental Setup

To test and evaluate the performance characteristics of the RTFDVP in comparison to synchronous enterprise resource planning (ERP) validation, a simulated enterprise financial validation environment was built using a mid- to large-enterprise financial environment with a moderately complex chart of accounts and dimension structure.

In this experiment, 50,000 financial journal entries were generated according to the same distribution found in enterprise transactions. Of these, 70% are standard financial journal entries with simple combinations of account and dimensions and no business rule validation; 20% are combinations of account and dimensions that require business rule validation against complex account hierarchies, and 10% are subject to business rule validation (threshold rule

matching and inter-company balance matching). The count of line items in each journal entry varies between three and twelve, resulting in an approximate count of 350,000 line-item validation checks.

In contrast, the synchronous validation baseline used a verification pipeline representative of an ERP, which was run sequentially for each transaction in the dataset. The RTFDVP system implemented the distributed asynchronous validation architecture in this paper with a shared pool of up to 8 concurrently executing validation workers.

The metrics we used are throughput (transactions per second), ERP-equivalent system CPU utilization, and end-user-visible transaction latency (the time between submission of a batch and availability of results to clients). Each system's metrics were recorded for five independent runs and the means computed.

8.2 Workload Characteristics

To compare the performance impact of synchronous versus asynchronous validation architectures, the transactional workload generated for the experiment required all 50,000 entries in the dataset to be inserted into the system in a single batch. This reflects a common peak-period journal upload scenario, such as a payroll posting or a journal import to consolidate financial information.

The test data setup included a chart of accounts of 8,500 active accounts, a four-financial-dimension (cost center, department, project, and business unit) structure with 120,000 valid combinations, an accounting calendar at the month level for three fiscal years, and 45 business rules, including 12 threshold-based approval rules and eight intercompany balance rules.

9. Results

9.1 Validation Throughput

Results from system evaluations have shown that the distributed RTFDVP platform outperforms the classical ERP validation architecture in terms of model checking time and memory.

Metric	Synchronous ERP baseline	RTFDVP (8 workers)
Throughput (transactions/sec)	120	1,150
Total batch processing time (50k entries)	417 s	43 s
Throughput improvement factor	—	9.58×

Table 1: Validation Throughput Comparison

The RTFDVP achieves about 1,150 tps throughput compared with the synchronous ERP validation baseline of 120 tps (9.58× higher throughput) by parallelized execution in eight worker instances, removal of blocking database lookups via an in-memory reference data cache, and removal of ERP application server overhead from the validation execution path.

Another experiment studied scaling of throughput with the number of RTFDVP workers, with 1 to 16 RTFDVP instances in the worker pool. The results show that throughput scales linearly up to eight workers and that adding more workers offers diminishing returns due to the event stream partitioning. This is a firm confirmation that this architecture can be used to achieve horizontal scalability to improve throughput.

9.2 System Load Reduction

The CPU usage of the ERP system on the application server during the validation process decreased by 45% in the RTFDVP configuration compared with the baseline because the validation processing is offloaded from the ERP application tier to a pool of RTFDVP validation workers.

Measurement	Synchronous ERP baseline	RTFDVP configuration
Peak ERP app-server CPU (%)	78%	43%
Average ERP app-server CPU (%)	61%	34%

CPU utilization reduction	—	−45%

Table 2: ERP system CPU utilization

In addition to throughput improvements, this approach offers operational advantages as well. Since the transaction validation processing takes place on the RTFDVP server, it can free the ERP application server to respond to interactive user requests (for example, financial reporting, inquiry, and approval requests) with a higher level of responsiveness.

9.3 User-Facing Validation Latency

User-facing validation latency, the time between entry and the test set.

Latency measure	Synchronous ERP baseline	RTFDVP (8 workers)
Average batch validation latency	8.0 s	1.5 s
Gateway acknowledgment latency	8.0 s (blocking)	<100 ms (async)
Latency reduction	—	−81.25%

Table 3: User-Facing Validation Latency

The result of implementing the RTFDVP was to reduce the average time taken to validate a period-close operation from approximately 8 seconds to approximately 1.5 seconds, a relative reduction of 81.25%. Consequently, the period-close operations are no longer a major operational annoyance to the financial users but are simply a minor inconvenience.

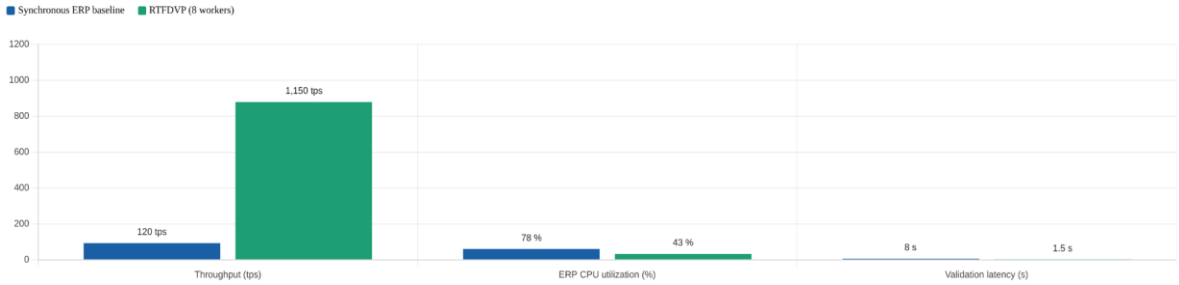

Figure 3a: Performance comparison across throughput, CPU utilization, and user-facing latency.

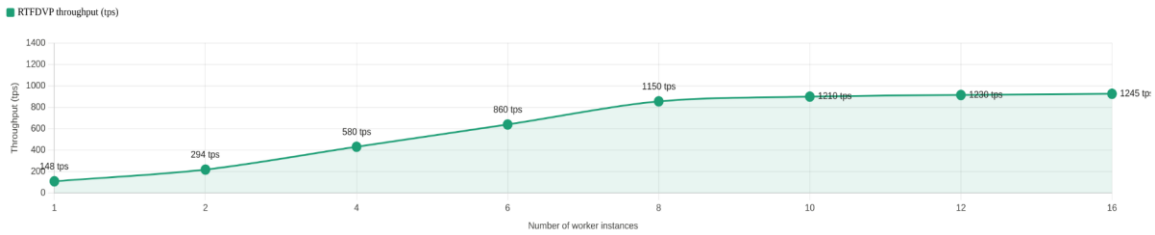

Figure 3b: Throughput scaling with worker count. Diminishing returns beyond 8 workers reflect the Kafka partition ceiling.

10. Discussion

10.1 Architectural Advantages

Our experimental setup shows meaningful gains in terms of throughput and latency for the high-volume batch workload in the distributed asynchronous architecture of the RTFDVP, thanks to the three design decisions in the architecture [17].

This provides a distinct advantage because it decouples the function of validation from the ERP application tier and avoids the validation bottleneck from the ERP application server tier. The ERP application server is designed to

support use of the capabilities of the full ERP application beyond validation, which incurs a performance penalty for each validation request. Because the RTFDVP's validation workers are cheaper to run with lower overhead, their per-process throughput is much higher than that provided by the validators [18].

Using an in-memory reference data cache eliminates all database I/O from the critical path of validation at transaction time. In customary enterprise resource planning (ERP), for every validation check that involves a reference data lookup (e.g., existence of an account, validity of a dimension, status of a period), a database query is issued. The RTFDVP circumvents this issue by checking rules in memory with sub-millisecond latency, leading to rule checks being limited by CPU throughput [19].

Horizontal worker scaling allows the RTFDVP to adapt its validation capacity to the workload. However, the validation capacity of an ERP is limited to the available capacity of the ERP application server tier, which is generally a more expensive scaling option. The RTFDVP is implemented using container-based workers, which allow the service to scale at the granularity of individual worker instances [20].

10.2 Operational and Compliance Implications

The RTFDVP architecture also offers operational benefits for enterprise financial management practice through centralized financial model governance. In addition to performance benefits, centralizing validation logic in a dedicated, independently maintainable platform presents benefits for governing financial business rules. Rule changes in customary ERP architectures typically require a change management process, testing, and promotion to production. Changes made to the RTFDVP rule repository can follow the normal change management process, but with lower risk and impact to the underlying ERP system because the core configuration of the ERP system does not change [16].

The RTFDVP has a validation audit trail, which keeps a permanent record of what rules were checked for each transaction, what tests were run, and what the results were. This information is useful for internal audit and compliance reporting. An auditor wanting to understand the effectiveness of validation controls could query the RTFDVP's result store to produce transaction validation histories without needing access to the ERP system [14], [17].

For organizations that must comply with the requirements of the Sarbanes-Oxley Act in the United States or the IFRS compliance requirements of stock exchanges around the world, the explainable, rule-based validation method of the RTFDVP provides an auditable control framework to support management assertions regarding the efficiency of the internal controls over financial reporting [13].

10.3 Limitations and Boundary Conditions

The latency advantage seen in our experiments shines through in high-throughput batch validation workloads. For example, in the case of validating a single transaction, such as a single journal entry from an interactive user interface, the latency advantage is slightly lower than that of synchronous ERP validation. This is because the overhead of publishing the event and subsequently sending the result back to the user asynchronously adds a small fixed latency penalty, which is magnified in the single transaction scenario [16].

The reference data synchronization mechanism allows a non-zero, but temporary, period of time when the reference data in the RTFDVP cache is potentially inconsistent with the authoritative reference data in the ERP system. If the reference data record is changed in the ERP system, there may be a period of time when the RTFDVP has validated a transaction based on stale reference data. The lag of change data processing depends on CDC propagation latency, which is a few seconds in the reference implementation, and the frequency of modifications to the reference data. Organizations that modify the chart of accounts or dimension structure frequently with continuous processing should take the CDC propagation latency into account and monitor for synchronization lag [17].

It should be noted that the experimental evaluation uses simulated transaction data instead of production workloads of the ERP. Although the simulation is configured for representative usage patterns, actual workloads can exhibit different characteristics, such as transaction complexity distributions, reference data access patterns, and so on, which are not covered by the simulation. Field deployments should test on a representative sample of the scale of production data [18].

10.4 Future Directions

The RTFDVP's modular nature also enables it to adapt to an evolving list of relevant developments in the enterprise financial technology landscape [19].

Machine learning anomaly detection is another natural extension of rule-based systems. Statistical models can be trained to detect transactions that deviate from a standard pattern, for example, in the transaction amount, combinations of account dimensions, or submitting the transaction at an unusual time. These could be flagged for review as another control in addition to the rule-based validation [14], [20], [21]. In contrast to the latter, anomaly detection models are capable of identifying outliers indicative of violations or mistakes, thus allowing for detection of anomalous patterns even if these are not prohibited by existing rules [22].

A further extension of the above is natural language processing (NLP) methods on journal entry narratives. Journal entry narratives are free text descriptions of the purpose and basis of a journal entry. Although they are typically part of financial documents, they are not usually validated automatically [23]. NLP models could assess the completeness and consistency of the narration with most other narrations at the entry and dimensional level and whether it matches known patterns of fraudulent or erroneous narrations [24].

Predictive validation uses machine learning models to proactively identify transactions that are likely to fail validation [25]. By fixing likely failures, either in the upstream systems (for example payroll, procurement, or treasury) or in the transaction generation workflows, predictive validation has the potential to reduce the number of transactions that fail and the associated effort to fix and resubmit them [15], [26].

5. Conclusion

This article has presented the Real-Time Financial Data Validation Platform (RTFDVP), a distributed architecture for financial transaction validation that addresses the throughput and scalability limitations of synchronous validation in traditional ERP systems. By decoupling validation logic from the ERP application tier, routing validation tasks through an event-driven distributed processing engine, and serving reference data from an in-memory cache, the RTFDVP enables validation throughput to scale independently of ERP application server capacity.

Experimental evaluation demonstrated that the RTFDVP achieves approximately 9.5× improvement in validation throughput, reduces ERP system CPU utilization by 45%, and decreases user-facing validation latency by more than 80% relative to traditional synchronous ERP validation for a representative enterprise batch workload. These performance advantages directly address the operational pain points associated with peak-period financial processing—month-end close, payroll posting, and consolidation journal import—in large enterprise environments.

The platform's modular rule engine, declarative rule configuration, and comprehensive validation audit trail provide operational and compliance benefits complementary to the performance improvements. The architecture's extensibility supports the incorporation of machine learning-based anomaly detection, NLP-based narration analysis, and predictive validation as future enhancements, positioning the RTFDVP as a foundation for intelligent financial data quality management.

The RTFDVP offers a practical and well-grounded framework for organizations seeking to modernize their financial validation infrastructure without replacing or fundamentally re-architecting their core ERP systems. By operating as an integration layer that augments rather than displaces existing ERP functionality, the platform provides a low-disruption adoption path for enterprises committed to improving financial data quality and system performance.

References

1. S. V. Grabski, S. A. Leech, and P. J. Schmidt, "A review of ERP research: A future agenda for accounting information systems," Journal of Information Systems, vol. 25, no. 1, pp. 37–78, 2011. [Online]. Available: https://doi.org/10.2308/jis.2011.25.1.37
2. A. Elragal and A. Al-Serafi, "The effect of ERP system implementation on business performance: An exploratory case study," Procedia Technology, vol. 5, pp. 418–427, 2012. [Online]. Available: https://www.researchgate.net/publication/228427747
3. R. Seethamraju, "Adoption of software as a service (SaaS) enterprise resource planning (ERP) systems in small and medium-sized enterprises (SMEs)," Information Systems Frontiers, vol. 17, no. 3, pp. 475–492, 2015. [Online]. Available: https://doi.org/10.1007/s10796-014-9506-5
4. J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in Proc. NetDB Workshop, 2011. [Online]. Available: https://notes.stephenholiday.com/Kafka.pdf
5. B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," ACM Queue, vol. 14, no. 1, pp. 70–93, 2016. [Online]. Available: https://dl.acm.org/doi/10.1145/2898442.2898444
6. Bahadir Tasdemird, "Event-Driven Microservice Architecture," in Medium, 2019. [Online]. Available: https://medium.com/trendyol-tech/event-driven-microservice-architecture-91f80ceaa21e

7. O. Zimmermann, "Microservices tenets: Agile approach to service development and deployment," Computer Science — Research and Development, vol. 32, no. 3–4, pp. 301–310, 2017. [Online]. Available: https://doi.org/10.1007/s00450-016-0337-0
8. C. Batini, C. Cappiello, C. Francalanci, and A. Maurino, "Methodologies for data quality assessment and improvement," ACM Computing Surveys, vol. 41, no. 3, pp. 1–52, 2009. [Online]. Available: https://dl.acm.org/doi/10.1145/1541880.1541883
9. L. Cai and Y. Zhu, "The challenges of data quality and data quality assessment in the big data era," Data Science Journal, vol. 14, pp. 1–10, 2015. [Online]. Available: https://doi.org/10.5334/dsj-2015-002
10. I. Taleb, M. A. Serhani, and R. Dssouli, "Big data quality: A survey," in Proc. IEEE International Congress on Big Data, pp. 166–173, 2018. [Online]. Available: https://www.publications.scrs.in/uploads/final_menuscript/8c6ce35a82531605aa22831622550741.pdf
11. A. Bhimani and L. Willcocks, "Digitisation, 'big data,' and the transformation of accounting information," Accounting and Business Research, vol. 44, no. 4, pp. 469–490, 2014. [Online]. Available: https://doi.org/10.1080/00014788.2014.910051
12. J. Moll and O. Yigitbasioglu, "The role of internet-related technologies in shaping the work of accountants: New directions for accounting research," The British Accounting Review, vol. 51, no. 6, 2019. [Online]. Available: https://doi.org/10.1016/j.bar.2019.04.002
13. M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," Journal of Network and Computer Applications, vol. 60, pp. 19–31, 2016. [Online]. Available: https://doi.org/10.1016/j.jnca.2015.11.016
14. W. Hilal, S. A. Gadsden, and J. Yawney, "Financial fraud detection through the application of machine learning techniques: A comprehensive survey," IEEE Access, vol. 10, pp. 36029–36052, 2022. [Online]. Available: https://www.nature.com/articles/s41599-024-03606-0
15. I. Psaroudakis, F. Wolf, N. May, T. Neumann, A. Ailamaki, A. Kemper, and T. Mitschke, "Scaling up mixed workloads: A battle of data freshness, flexibility, and scheduling," Proceedings of the VLDB Endowment, vol. 8, no. 12, pp. 1–8, 2015. [Online]. Available: https://www.researchgate.net/publication/283614783
16. N. D. Benneh, "Structured Finance Mechanisms for Enhancing Long-Term Capital Formation," IPHO-Journal of Advance Research in Business Management and Accounting, vol. 2, no. 7, pp. 01–10, 2024. [Online]. Available: https://doi.org/10.5281/zenodo.19753841
17. Y. Suthari and K. B. Manam, "Behavioral Profiling and AI-Powered Security for IoT Networks in Smart City Environments," in Proc. 2025 International Conference on Artificial Intelligence's Future Implementations (ICAIFI), pp. 41–46, 2025. [Online]. Available: https://doi.org/10.1109/ICAIFI66942.2025.11325861
18. N. D. Benneh, "Central Bank Engagement and Regulatory Reforms in Strengthening Financial Systems," IPHO-Journal of Advance Research in Business Management and Accounting, vol. 3, no. 4, pp. 01–09, 2025. [Online]. Available: https://doi.org/10.5281/zenodo.19754058
19. N. D. Benneh, "Institutional Governance and Risk Management in Modern Financial Systems," IPHO-Journal of Advance Research in Business Management and Accounting, vol. 3, no. 12, pp. 56–65, 2025. [Online]. Available: https://doi.org/10.5281/zenodo.19753607
20. R. Gollapudi, "Risk-Controlled Near-Zero-Downtime Oracle Database Migration Using GoldenGate," International Journal of Computational and Experimental Science and Engineering, vol. 8, no. 3, 2022. [Online]. Available: https://doi.org/10.22399/ijcesen.5382
21. Y. Suthari, Z. Thakker, S. Govindarajan, N. Krishnan, V. Raju, and S. K. Rao Katta, "Cost Optimization Techniques for Efficient Resource Allocation in Cloud Computing Environments," in Proc. 2025 3rd International Conference on Intelligent Cyber Physical Systems and Internet of Things (ICoICI), pp. 793–797, 2025. [Online]. Available: https://doi.org/10.1109/ICoICI65217.2025.11253514
22. F. A. Clottey, "Strategic Leadership Approaches to Enhancing Procurement and Supply Chain Optimisation for Operational Excellence," Journal of International Crisis and Risk Communication Research, pp. 3425–3434, 2024. [Online]. Available: https://doi.org/10.63278/jicrcr.vi.3758
23. P. Sunil Kumar, "Index-State Uncertainty for Trustworthy Enterprise Retrieval-Augmented Generation (RAG)," Journal of Information Systems Engineering and Management, vol. 10, no. 36s, pp. 1258–1271, 2025. [Online]. Available: https://doi.org/10.52783/jisem.v10i36s.15043
24. F. A. Clottey, "The Role of Strategic Leadership in Strengthening Team Performance and Supply Chain Resilience for Business Growth," IPHO-Journal of Advance Research in Business Management and Accounting, vol. 3, no. 7, pp. 16–23, 2025. [Online]. Available: https://doi.org/10.5281/zenodo.19605201
25. D. Bajaj, V. Shah, and S. Kanaparthi, "A Practical Framework for Migrating Large Manual Test Suites to Automation Pipelines: An Industrial Case Study," Journal of Information Systems Engineering and Management, vol. 9, no. 3, pp. 1–8, 2024.
26. F. A. Quintero, "Fluid Dynamics-Based VFX Design: Trade-Offs between Visual Realism, Computational Cost, and Production Timelines," Journal of Computational Analysis and Applications (JoCAAA), vol. 31, no. 4, pp. 2722–2736, 2023. [Online]. Available: https://www.eudoxuspress.com/index.php/pub/article/view/5148