NextGen Scientific Publication
American International Journal of Computer Science and Technology
Volume 8 Issue 2, Pg. No. 61-83, AIJCST-V8I2P106, 2026
https://doi.org/10.63282/3117-5481/AIJCST-V8I2P106

Original Article

An Intelligent Cross-Border Remittance Processing Framework Integrating Machine Learning-Based Payment Orchestration and Java Spring Boot Microservices for Scalable International Financial Transactions

***Chandra Sekhar, Nimmakayala**
Independent Researcher, USA.

Abstract:

Cross-border remittances continue to be costly and unstable. The global average cost of transferring USD 200 was 6.36% of the value transferred in Q3 2025. On average, banks charged 14.99% for such transfers. The G20 roadmap has set a target of a global average cost of 1% with no corridor having a cost greater than 3% by the end of 2027. However, the Bank for International Settlements (BIS) does not believe this target will be met on time. Static priority lists and lowest-quote rules used today to route money transfers do not account for the status of individual providers. Therefore, when a transaction fails, it is only after the timeout period expires and the costs associated with retrying the transaction are incurred that the system realizes there is a problem. This paper defines an architecture that divides evidence, prediction, policy, and execution. A calibrated, provider-conditioned settlement probability model uses lagged, leakage-free attempt telemetry. A governed ranking policy utilizes the output of the model along with explicit fees, foreign exchange spreads, predicted latency, and stability terms, subject to hard cost and latency limitations. Twelve Java Spring Boot microservices isolate provider heterogeneity to an adapter layer. They also offer idempotency, bounded retry with exclusion, per-provider circuit breaking, saga compensation, and reconciliation. Evidence levels dictate evaluation. On a documented synthetic dataset, the reference model achieved a held-out ROC-AUC of 0.723. The proposed policy decreased expected cost per settled transaction by 1.9 percent when compared to the highest-performing static baseline. The success-rate difference had a bootstrap confidence interval that included zero. Post-hoc calibration only decreased the expected calibration error from 0.326 to 0.300. Analytical estimation provided the orchestration tier capacity. No latency, throughput, or availability measurements were made.

Keywords:

Cross-Border Remittance, Payment Orchestration, Machine Learning, Java Spring Boot, Microservices Architecture, Financial Technology, Intelligent Payment Routing, Distributed Systems.

Article History:

Received: 21.01.2026

Revised: 26.02.2026

Accepted: 04.03.2026

Published: 13.03.2026

1. Introduction

Remittance flows on an international scale represent some of the most consistent cross-border transfers of funds in terms of size in the global economy. In 2023, the officially reported remittance transfers to low-and middle-income countries (LMIC) totaled USD 656 billion, and according to the World Bank, these were expected to grow by approximately 2.3% per year to USD 671 billion in

Copyright by the Author(s). This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International (CC BY-NC-ND 4.0) License (https://creativecommons.org/licenses/by-sa/4.0/

2024 and USD 690 billion in 2025 [2]; and for over sixty economies, the remittance transfers represent at least three percent of their respective Gross Domestic Product [1]. Stating the purpose is relatively straightforward; value needs to be transferred from the sender in one jurisdiction to the recipient in another, in a different currency, across different regulatory systems and through various institutional frameworks that do not share either a common ledger or message standard, or a standard definition of when a transaction becomes final.

The price of remittances continues to be high. The World Bank Remittance Prices Worldwide (RPW) database has reported that as of the third quarter of 2025, the average cost to send $200 USD globally was 6.36%, slightly lower than what was reported as 6.49% during the first quarter [1]. There was also a significant difference in costs among different types of service providers; Banks had an average cost of 14.99%, Money Transfer Operators (MTOs) had an average cost of 4.72%, and Post Offices/Mobile Operators both had an average cost of 5.58%. However, only one percent of the sample were mobile operators [1]. In terms of regional averages, Sub-Saharan Africa had the highest average cost at 8.46%, while the Middle East/North Africa/Afghanistan/Pakistan region had the lowest average cost at 5.11% [1]. To meet the Sustainable Development Goal Target 10.c, which states that average costs should fall below three percent by 2030, and to achieve the G20 Roadmap target of reducing retail transaction costs to a global average of one percent and ensuring that no corridor exceeds three percent by the end of 2027, further reductions in costs will be necessary [3].

The progress has been relatively little. The Bank for International Settlements (BIS) has found that although most of the policy actions contained in the G20 roadmap have now been completed, the 2027 deadline set for completing all targets will likely not be met, and only a limited amount of benefit has yet been seen by end-users [4]. The Committee on Payments and Market Infrastructures (CPMI) and Financial Stability Board (FSB) 2025 monitoring survey of 82 jurisdictions has identified interoperability, extended hours of operation, ISO 20022 alignment and standardized API frameworks as the key determinants of whether positive outcomes occur [5]. Therefore, the critical constraint is no longer solely infrastructure, but rather how each transaction moves through existing provider networks.

An operator with multiple provider relationships makes a continuous decision. A given corridor, amount, currency pair and time of day define which providers are technically and legally able to process the transaction. Each provider offers different fees, different foreign exchange (FX) spreads, different expectations for processing times, different histories of operating reliability, and different settlement finalities. The operator must choose one prior to knowing how well the transaction will turn out. Static configurations have been used to resolve this issue in the past. This includes establishing a priority order among the available providers; selecting the provider with the lowest quoted fee; or applying a deterministic set of rules developed by the staff. Although these strategies provide transparency and auditing capabilities, they are unable to recognize changes within the network. They cannot tell if a provider is offering low prices because it is highly efficient or if it is offering low prices because it is failing. Additionally, they cannot adjust their strategy based on the degradation of a provider's service quality that occurs over hours instead of days.

Cost will be maximized in terms of expectations. If an attempt at processing fails, then that attempt does not come at zero cost. The money was tied up, the customer was not given credit, and another transaction was processed, which costs more money. The processing times for transactions can vary significantly from what can be represented by a set of static rules. A provider that has failed to process a payment can keep the payment open until a timeout occurs; then they realize the payment has failed. The burden of managing and updating the priority lists for each provider increases as the number of providers increases. Also, the accuracy of these priority lists decreases over time due to drift in how the providers operate.

The next need for resiliency is that any type of failure (transient, terminal, or ambiguous) should be resolved without either the sending party being charged twice or the receiving party being paid twice. Java and Spring Boot provide a workable platform to support RESTful API contracts using Spring Web, authentication through Spring Security, persistence via Spring Data, discovery with Spring Cloud, events through Kafka, and circuit breaking via Resilience4j. Decomposition of microservices allows every concern to have its own service that has an explicitly defined contract and data boundary [12]. Experience with decomposed core banking systems demonstrates that decomposition can occur; however, it is a process of defining data ownership and consistency rather than simply splitting apart the system [6].

The research gap is the lack of an integrated, explicitly stated and objectively evaluated framework that views the selection of providers as a decision with constraints, where calibrated probability is one of multiple inputs, represents cross-border aspects as first-

class decision variables and defines the mechanisms to make the execution of the decision safe. This paper presents a constrained cost and latency-aware formulation based on expected cost per settled transaction; a 12-service Spring Boot specification with defined data ownership and contracts; a leakage-safe estimation pipeline with calibration and drift monitoring; an executable model for handling failures; and an evaluation that distinguishes between simulated, analytical and planned evidence, but does not claim to measure actual performance in production.

2. Related Work and Literature Review

2.1. Cross-Border Remittance Systems

Traditional transfers use a process called correspondent banking; this is where the sending bank uses a series of intermediate banks to make payments to the receiving bank. Every step of the way has its own set of charges, a cut-off time and an FX conversion, all of which are invisible to the sender. The RPW methodology calculates the overall costs as the sum of all fees charged and the difference between the exchange rate given to customers and the mid-market rate [1]. Since the FX spread must be modeled, the provider with the lowest advertised fee may ultimately be the most expensive once the spread is factored in. Digital platforms and mobile money have altered the cost dynamics for certain payment corridors; however, they do not eliminate fragmentation [1]. The BIS assessment that the G20 targets will most likely not be achieved [4], suggests that operator-level optimization should complement infrastructure reforms, rather than replace them.

2.2. Payment Orchestration and Provider Selection

Payment orchestration uncouples the remitter or merchant from each individual bank or network to decide for every single transaction what connection to use. Orchestration is typically implemented via configurable rule sets: cheapest-eligible, highest-priority, or weighted allocation with manual overrides. Rule-based orchestration continues to be used because it can be audited, is deterministic and can be altered without having to deploy a model. However, rule-based orchestration is limited in that it represents a static representation of a dynamic system. For example, a cheapest-eligible rule will continue to route transactions to a provider whose authorization rate has fallen until such time as a human changes the configuration. The transparency objective of the roadmap requires the remitter/merchant to disclose all of the transaction costs associated with making/receiving payments, foreign exchange rates/conversion charges, delivery time-frames, and tracking capabilities for payment status [3].

2.3. Machine Learning in Financial Transaction Processing

Supervised learning has been developed for classification of transaction-level risks. Dal Pozzolo et al. report practitioners' experience with detecting fraudulent use of credit cards; they highlight that the most important factor when choosing metrics for class-imbalanced datasets is the choice of metric itself. They also discuss how the time lag between when a transaction occurs and when labels become available can affect classification performance. Finally, they document how sampling can distort the estimated probability distributions for the classes [10]. Lucas and Jurgovsky reviewed the same type of problems and emphasized that the operational measure of performance should be based on the costs associated with errors rather than simply accuracy [15]. These findings are applicable to settlement predictions as well. Tabular baseline models such as gradient boosted trees and random forests continue to provide strong performance [7], [8]. Due to the heterogeneous nature of routing features (which include both continuous and categorical variables), their low cardinality, and the fact that many of them may be unobserved at the time a model is initially trained (cold start), tree-based ensemble methods and regularized linear models represent the logical choice of candidate models. The primary risk in this area is distribution shifts:

2.4. Microservices in Financial Applications

The microservice architecture represents an application as independently deployable services that have been aligned with the business' capabilities, where each service manages its own data independently from other services and communications between these services are lightweight [12]. According to Aydemir and Başçiftçi, performance-efficient microservice core banking can be achieved; however, decomposition will require careful consideration of distributed consistency [6]. Additionally, Spring Boot provides the common substrate for developing applications using Java [13]. Distributed consistency is the challenging aspect of building microservices. Saga, a long-running transaction made up of multiple local transactions, each having a compensating action, was developed by Garcia-Molina and Salem [9] and is precisely what a cross-border payment is. Compensation is asymmetric: an authorization can be reversed; however, once a payment has been settled, it can only be refunded. Therefore, Section IV-F separates the authorization from the settlement process. Resilience patterns are well established; however, they are typically defined without operational parameters. The Resilience4j circuit breaker has three states: closed, open, and half-open [14] and, without scope per

provider, protects one service instance while other instances continue to receive traffic. As such, the breakers used in this study were keyed per provider and shared their state via a distributed store.

2.5. Research Gaps

Four gaps are identified. The institutional work of measuring outcome variables (such as costs, processing times, transparency, and accessibility) does not provide guidance for operators to make decisions regarding individual transactions within the constraints of these outcome variables [1, 3, 4, 5]. While microservice literature demonstrates that financial systems can be broken down into microservices and managed accordingly [6, 12, 13], it fails to relate this decomposition to a decision-making process based on the correct management of data. Although machine learning literature defines how to select, calibrate, and handle shifts in classification models [7, 8, 10, 11, 15], it defines prediction as the outcome variable and not as an input to a constrained and auditable policy. Finally, although transaction-processing literature defines sagas [9] and resiliency patterns [14], it does not connect them to decisions made prior to understanding the outcome of the transaction. The framework presented below addresses all four of these gaps simultaneously, separating learned quantities from governed quantities. Table I lists all of the references used in this paper and identifies the gap left open by each.

Table 1. Comparative Review of Existing Cross-Border Remittance Research and its Remaining Gap

Source	Type/year	Focus	Contribution used here	Remaining gap
[1] World Bank RPW	Dataset, 2025 (Q3 2025)	Corridor and provider cost of sending USD 200	Defines total-cost measurement incl. FX margin; provider-type averages	No decision procedure for an operator
[2] World Bank	Report, Jun. 2024	Remittance volumes, projections, risks	Volume and risk context; cost series	No routing/architecture guidance
[3] FSB / G20	Targets, 2021–22	Cost, speed, access, transparency targets	Numeric outcome constraints; total-cost disclosure requirement	Targets, not mechanisms
[4] BIS Bulletin 119	Bulletin, 2025	State of play vs. G20 targets	Evidence that targets are unlikely to be met; modest end-user gains	No transaction-level remedy
[5] CPMI/FSB Brief 13	Survey, 2025 (82 jurisdictions)	Interoperability, ISO 20022, API frameworks	Infrastructure preconditions (API standardisation)	No provider-selection model
[6] Aydemir & Başçiftçi	Journal, 2022	Microservice core banking performance	Feasibility of decomposed banking systems	No routing decision model
[7] Chen & Guestrin	Conference, 2016	Scalable tree boosting	Tabular learner candidate	Not payment-specific
[8] Breiman	Journal, 2001	Random forests	Tabular learner candidate; OOB error	Not payment-specific
[9] Garcia-Molina & Salem	Conference, 1987	Sagas, compensation	Compensation semantics for multi-step execution	No pre-decision uncertainty
[10] Dal Pozzolo et al.	Journal, 2014	Fraud detection practice	Metric choice under imbalance; delayed labels	Fraud, not settlement success
[11] Gama et al.	Journal, 2014	Concept drift adaptation	Drift taxonomy and detection, families	No routing application
[12] Lewis & Fowler	Article, 2014	Microservice definition	Service boundaries, decentralised data	No financial decision model

[13] Spring Boot docs	Documentation, 2025	Framework capabilities	Implementation substrate	No architecture guidance
[14] Resilience4j docs	Documentation, 2025	Circuit breaker semantics	Per-provider breaker state machine	No policy integration
[15] Lucas & Jurgovsky	Survey, 2020	ML fraud detection survey	Cost-weighted evaluation of imbalanced tasks	Not settlement prediction

3. Problem Formulation and Research Methodology

3.1. Problem Definition

A request is defined as a vector x of features, including the corridor, amount a, source/target currencies, channel, sender tier, time-of-day, and the upstream risk decision. Let E(x) be the subset of integrated providers P that are available to process requests based on their corridor coverage, currency compatibility, amount limits, operational status, and regulatory permissions. For each p in E(x), the operator has access to a quoted fee f_p, an FX rate r_p (the spread s_p is calculated by comparing r_p to a reference mid rate), historical processing time distributions, and recent attempt-level telemetry data. Since the operator does not know if the selected provider p* will complete settlement when making the selection, the operator makes the decision under uncertainty. There are three conflicting objectives in selecting the optimal provider: minimizing expected cost per completed settlement; minimizing expected time to funds availability; and achieving a success rate that meets the service level agreement. In addition to these conflicts, there are five other complexities involved in this decision-making process: outcomes are delayed, meaning that the labels associated with a decision at t arrive at t+Δ; provider behavior is dynamic; constraints are difficult to satisfy; execution can fail after a decision is made; therefore, the selection process must be re-entrant and allow for an exclusion set; finally, the decision must be explainable after the fact.

3.2. Research Objectives

O1: a provider-conditioned settlement probability estimator whose input variables can be causal at the time of making decisions and whose output variables are calibrated. O2: a cost model which includes provider fee, foreign exchange spread and a per attempt fixed charge expressed as a percent of the transaction value. O3: a constrained ranking policy that combines the estimated probability with controlled cost, latency, and stability factors. O4: a service decomposition and execution protocol that protects against failure, such as being idempotent, having a limited number of retries with exclusion, circuit breaker and compensation. O5: evaluation of simulated vs. analytical vs. planned measurements.

3.3. System Requirements

Functional requirements include: initiation, validation and screening, eligibility determination, scoring, ranking, execution, status tracking, notification, reconciliation and audit. Non-functional requirements include: latency budgets per hop, availability of the orchestration path, exactly-once effects on top of at-least-once delivery, auditability of decisions, provider isolation and traceability of the active model. Table II lists every requirement and design element that meets it and the proposed verification technique for it. It was intentionally paired: if there isn't a verification technique specified for a requirement, then it is not considered a specification but an aspiration.

Table 2. Functional and Non-Functional Requirements with the Satisfying Design Element and Proposed Verification

ID	Requirement	Satisfying element	Proposed verification
FR1	Accept a remittance request idempotently	API gateway + initiation service, client key	Replay test: duplicate POST returns identical resource
FR2	Validate schema, limits, currency and sanctions	Validation service + screening adapter	Rule matrix tests; list-version persistence check
FR3	Determine eligible provider set	Eligibility filter over provider registry	Coverage-matrix unit tests; expiry boundary cases
FR4	Estimate per-provider settlement	ML inference service (batched)	Held-out ROC-AUC, ECE (Table 6)

	probability		
FR5	Select provider under cost/latency ceilings	Orchestration engine, Eq. (4)	Trace replay against baseline policies (Table 8)
FR6	Execute with duplicate suppression	Adapter layer + idempotency keys	Fault injection: duplicate callback, retry storm
FR7	Publish status changes	Kafka topics, outbox relay	Consumer replay; duplicate-event test
FR8	Reconcile against provider statements	Reconciliation service	Seeded discrepancy injection test
FR9	Retain an auditable decision record	Decision table + audit store	Field-completeness audit on sample
NFR1	Decision latency budget per hop	Batched inference; cache reads	Histogram percentiles from load test (planned)
NFR2	Availability of orchestration path	Multi-replica deployment, HPA, PDB	Chaos test: replica loss, zone loss (planned)
NFR3	Exactly-once effects under at-least-once delivery	Idempotent consumers + key store	Replay of 10^5 events with duplicate injection
NFR4	Provider isolation	Per-provider bulkheads and breakers	Saturation of one provider, observe others
NFR5	Governed, versioned policy	Immutable weights + policy version in decision	Config validation tests (Listing 2)
NFR6	Traceability of model in force	Model version + feature hash persisted	Join decision record to registry entry

3.4. Research Methodology

The process of the research was organized into six stages: reviewing documentation and literature; analyzing requirements resulting in Table II and the service boundaries; designing the architecture, decision model and contract; implementing the decision layer; evaluating the decision layer at three levels of depth; and analyzing findings resulting from the simulation as opposed to those resulting from design reasoning. The evaluation included executing a simulation of the decision layer using a synthetic data set that had a known generating process, creating an analytical capacity model and defining but not executing a load, fault-injection and drift evaluation. This staging is provided since the validity of the research is based upon it: this is a well-defined design with a partially evaluated decision layer.

3.5. Assumptions and Constraints

A1: six providers are modelled across bank, MTO, mobile money and card rails with an assumed coverage matrix. A2: simulated fee (60–145 basis points) and spread (18–41 basis points) parameters are plausible but not calibrated to RPW corridor averages and sit below the bank average of [1], so the simulation supports conclusions about relative policy behaviour, not achievable absolute cost. A3: outcomes are drawn from a documented logistic process, so separability is a property of the generator. A4: labels are treated as immediately observable, overstating available information and making reported discrimination an upper bound. A5: FX is modelled as observed quotes with a fixed spread per provider; intraday volatility and quote expiry appear in the design but not the simulation. A6: the provider simulator for the planned load evaluation is assumed to support configurable latency and error injection; none was operated. A7: the capacity model assumes Poisson arrivals, exponential service and 18 ms mean service demand assumptions, not measurements. A8: sanctions screening is a synchronous gate with internal latency out of scope. A9: no customer data of any kind was used.

4. Proposed Intelligent Cross-Border Remittance Framework

4.1. Framework Overview

The framework is built around one single decision point, and it separates four concerns that are frequently confused: evidence, prediction, policy, and execution. Figure 1 illustrates all aspects of the framework. A sender accesses the system via mobile, web or partner API; the gateway terminates transport security, provides authentication, and limits rates; the identity service identifies

senders, tiers and limits; validation applies schemas, screens for sanctions, and checks currency and limits; the engine coordinates eligibility, scoring and ranking; the inference service calculates the calibrated probability of successful delivery and estimates the cost and latency associated with each eligible provider; the foreign exchange service quotes rates with clearly defined spreads, expirations and auditing trails; the risk service evaluates rules and calculates an anomaly score; the adapter layer translates provider APIs into standard formats, manages provider-specific circuit breakers and idempotency keys; and finally, the event bus publishes updates to monitoring, reconciliation, notification, and auditing systems. There are two critical characteristics of the framework: first, only the engine can determine which provider will process the transaction; therefore, there is only one decision-making process, and it can be audited; second, only the adapter layer understands how each provider processes transactions; therefore, adding new providers does not require changes to the decision-making logic of the system; instead, it simply requires creating a new adapter and configuring its settings.

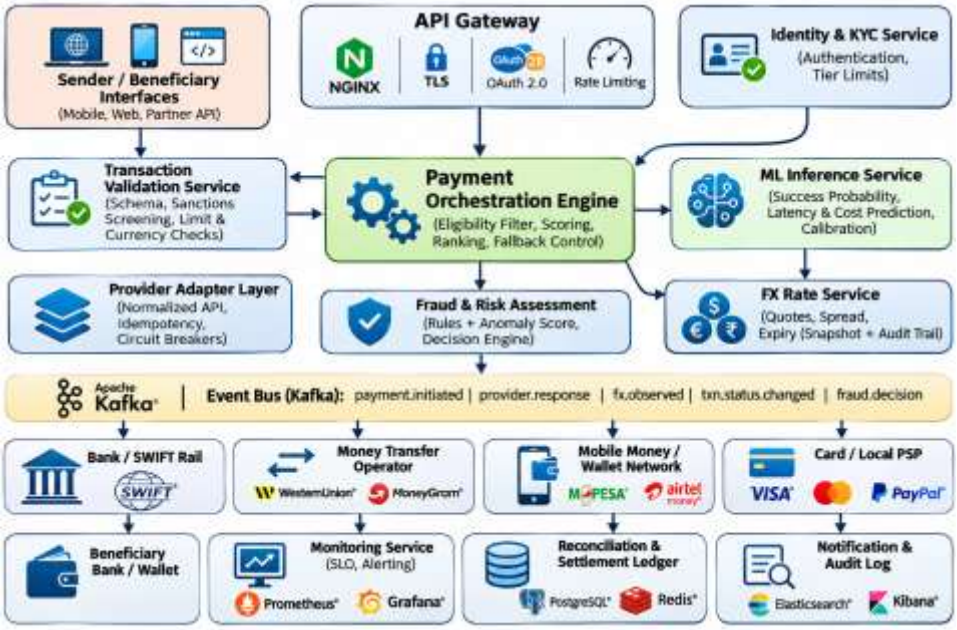


Figure 1. Overall Intelligent Cross-Border Remittance Framework. Evidence Flows Upward from Provider and FX Sources; the Orchestration Engine is the Single Decision Point; Execution is Delegated to the Adapter Layer, and All State Changes are Published to the Event Bus for Monitoring, Reconciliation and Audit

4.2. Intelligent Payment Orchestration Engine

The engine executes the same eight steps shown in Figure 2 for every transaction request it processes. Those include receiving the request with an idempotency key provided by the client, validating the request, screening the request, getting a risk decision about the transaction, halting if there is a risk block, building E(x) from the provider registry based on whether or not they cover the transaction, use currency, have an amount limit, have a working schedule, have permission to operate, and then remove providers whose breaker is open at that time, rather than after they have been scored; collect features up to a maximum staleness bound; get estimates for all of the providers in one batch call; rank E(x) according to the governed policy and hard constraints; issue an idempotency key scoped to (transaction, provider, attempt), call the adapter, evaluate the response from the adapter, and either publish a confirmation event or enter fallback. Unlike a single-pass router, which does not allow for retrying, the exclusion-set mechanism allows for a retry to be considered a new decision made over E(x) less X, where X represents the providers that have failed previously on the same transaction. Since X is persistent, the decision to retry is able to be reproduced at any later date.

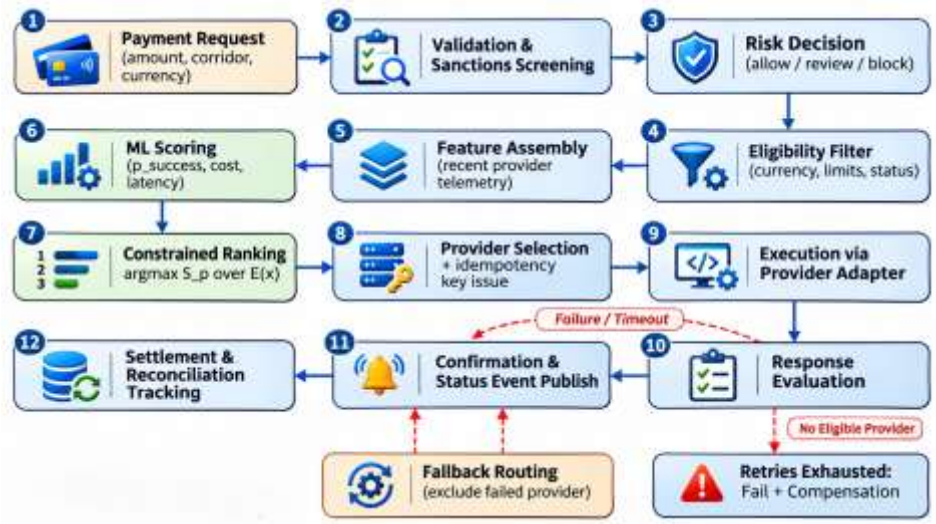


Figure 2. Intelligent Payment Orchestration Workflow. Solid Arrows Denote the Nominal Path; Dashed Arrows Denote the Failure Path, in which the Failed Provider is Added to an Exclusion Set, and Ranking is Re-Entered with the Reduced Eligible Set

4.3. Machine Learning-Based Payment Selection

The learning objective is the binary categorization of the occurrence where provider p completes transaction x during the agreed-upon timeframe. The sample unit is the (transaction, provider) pair of those pairs which are acceptable for analysis; it is not the individual transaction, as the choice is based upon comparison of multiple providers. There are four types of features as presented in Table IV: provider-static characteristics including rail type, fees, and spread; provider-dynamic short-window completion attempts success rates, median and high-percentile latency, and timeout frequencies, lagged so that decisions made at time t use only outcomes completed prior to time t; transaction-related characteristics such as amount, corridor, hour, day, and channel; and corridor-by-provider interactive characteristics, the actual essence of the issue. Model selection was conducted conservatively; i.e., regularized logistic regression with indicator variables for both providers and corridors was used as a reference model due to its simplicity in interpretation, low cost to implement, and robustness when dealing with limited data; however, gradient-boosted decision trees [7] and random forests [8] were evaluated as potential alternatives. Evaluation/calibration of the model was performed as a distinct phase since the policy applies probabilities to monetary values. Figure 4 illustrates the model as a versioned artifact: each response contains a model version and a hash code representing the feature vector used to generate the response; the engine records both pieces of information along with the decision itself.

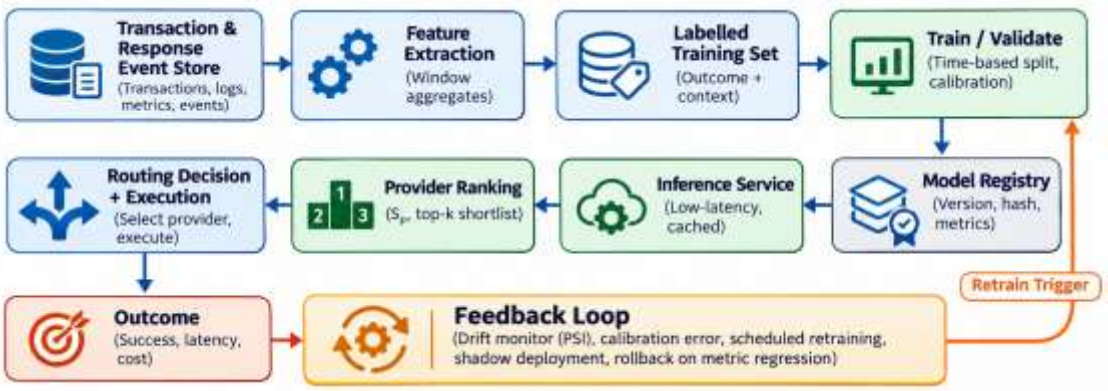


Figure 3. Machine Learning-Based Payment Routing Pipeline. The Feedback Path Closes the Loop from Observed Outcomes through Drift Monitoring to Retraining and Shadow Deployment; Model Artefacts are Versioned so that Every Persisted Decision References the Model that Produced it

4.4. Payment Provider Integration

A single internal contract connects providers using four operational functions (Quote, Execute, Query Status, and Cancel/Compensate). Each adapter converts this contract into the provider's protocol, either REST with JSON, ISO 20022 Exchange, or Batch File and standardizes the response taxonomy into four categories: ACCEPTED, REJECTED, INDETERMINATE and DUPLICATE. The distinction between REJECTED and INDETERMINATE is the key component of the contract. REJECTED indicates clearly that no transaction occurred, while INDETERMINATE requires resolution via status query prior to attempting another execution. Adapter configurations exist as data, not as code; therefore, providers may be onboarded, suspended, or re-priced without requiring new deployments of decision services.

4.5. FX Conversion and Cost Evaluation

FX is considered to be a first-class cost component as opposed to a simple conversion utility. When converting amount "a" from currency "C_s" to currency "C_t", the FX service provides the reference mid rate, the provider's rate, the implied spread in basis points, a unique quote identification number, and an expiration date. The cost model then converts the spread to monetary terms, adds the stated fee, and adds any fixed per-conversion attempt charge. Two rules apply. If a quote has expired before the estimated completion date of the transaction, the quote is not available for use. The filter will need to either update its list of available quotes from the provider or remove the provider entirely, as attempting to complete a transaction using an expired quote is a common cause of reconciliation issues.

4.6. Transaction Lifecycle Management

A finite state machine has ten possible states as illustrated in Fig. 8; these states include INITIATED, VALIDATED, PENDING, PROCESSING, AUTHORIZED, SETTLED, FAILED, CANCELLED, REVERSED, and REFUNDED. A named event triggers each transition, which is then verified against the machine and recorded with the transaction identifier, current and next states, event identifier, actor, and timestamp. Since repairing invalid transitions silently would destroy the audit trail, the machine prevents invalid transitions rather than fixing them. AUTHORIZED and SETTLED are separate due to the fact that they have different compensation types. A provider can reverse an AUTHORIZATION; however, if a payment is SETTLED, the beneficiary institution must return the funds. Collapsing these two compensation types into one would result in refunds becoming untraceable. Reconciliation compares the ledger entries for each provider and settlement date with the statement received from the provider. Any differences identified are classified as either timing, fee, foreign exchange (FX) or missing. Instead of automatically correcting these differences, reconciliation creates owned items to be resolved.

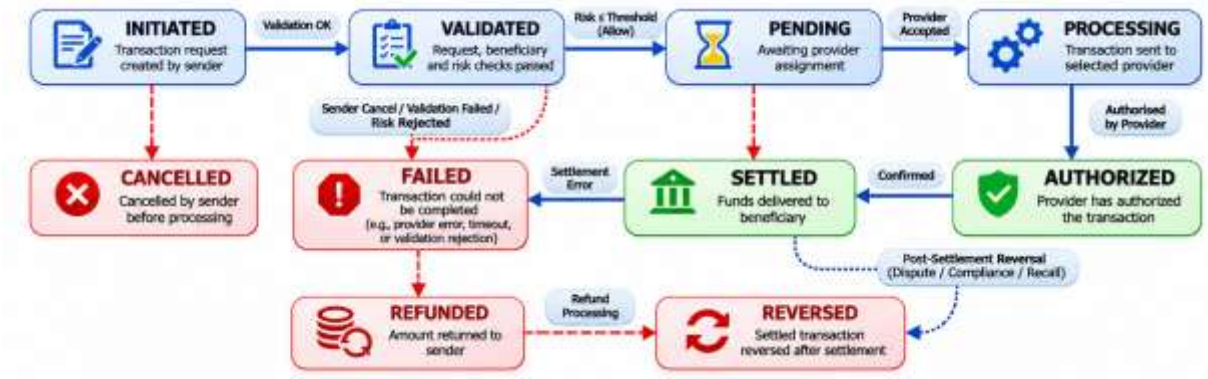


Figure 4. Transaction State Transition Diagram. Solid Arrows Denote Normal Progression; Dashed Arrows Denote Exceptional Transitions. Settled, Failed, Cancelled ad Refunded are Terminal; Reversed is Transient and Must Resolve to Refunded

4.7. Failure Handling and Recovery

Failure handling is specified by the system instead of using the default failure handling provided by the framework. Each provider call has a propagated deadline; if the deadline expires, then INDETERMINATE will be returned instead of a generic exception. There are limits on how many times a provider can be called (retry policy), the time between each attempt increases exponentially (exponential backoff) with some randomness added to help avoid multiple providers from calling at the same time (jitter).

Additionally, if a provider fails, that provider cannot be used again during the same transaction. Prior to making a new attempt, the system verifies that the first attempt did not succeed (idempotency check). Therefore, an INDETERMINATE result will not result in a duplicate payment. Each provider has its own circuit breaker, which is activated based on a certain percentage of failures or slow responses. When a breaker is activated, the provider is unavailable for selection during the transaction until a "half-open probe" successfully completes. If there are no available providers, the transaction fails, and compensation is initiated. Figure 6 illustrates the process flow, and Section V-F discusses the complexity associated with this process.

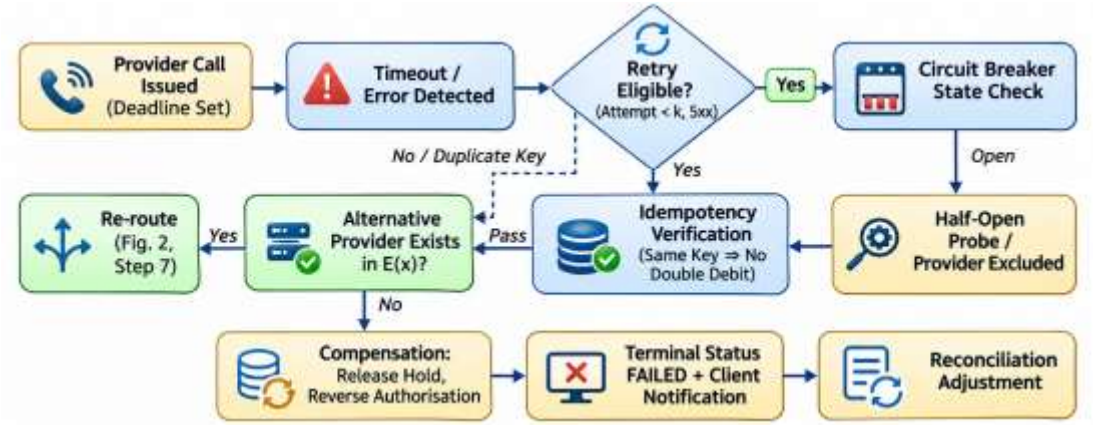


Figure 5. Transaction Failure Recovery and Resilience Workflow. The Idempotency Check Precedes Every Retry so that an Indeterminate Provider Outcome Cannot Produce a Duplicate Payment; Compensation is Entered Only when the Eligible Set is Exhausted

4.8. Security and Compliance

Security is multi-layered: Transport Layer Security (TLS) with mutual authentication among services; OAuth 2.0 with short-lived tokens, authorization enforced per endpoint and per resource owner; envelope encryption for data in motion and at rest for all account identifier information, and tokenization of that information in logs; secrets from a managed service injected at runtime; and append-only audit records of the input to a decision, the model version, the policy version, and the rule(s) used to make the decision to block. Compliance is handled as integration points. Sanctions and Politically Exposed Person (PEP) screening occur during validation and are persisted with the list version, since any decision made must be verifiable against the list in effect at the time of the decision. Transaction monitoring processes the event stream to identify structure and velocity patterns; this is a separate process from the transaction-level scoring. The Know-Your-Customer (KYC) tier determines limits and corridors and is owned by the identity service. Retention policies and field-level redaction on export provide implementation of data protection requirements.

Table 3. Proposed Framework Components, Principal Responsibility and Owned Data

Component	Principal responsibility	Owned data
API gateway	TLS termination, token validation, rate limiting, routing	No persistent business data
Identity & KYC	Authentication, tier, limits, permitted corridors	Customer, tier, limit records
Remittance initiation	Request capture, idempotent creation, tracking id	Transaction header, request payload
Validation	Schema, sanctions/PEP screening, currency and limit checks	Screening result + list version
Payment orchestration	Eligibility, feature assembly, ranking, fallback control	Decision record, attempt record, exclusion set
ML inference	Probability, cost and latency prediction, calibration	Model artefact reference, feature hash
Provider adapter	Protocol translation, normalisation, breakers, idempotency	Provider reference, raw response
FX rate	Quote, spread, expiry, reference-rate snapshot	Quote record, observation series
Fraud & risk	Rules + anomaly score, allow/review/block decision	Risk decision, contributing rules

Transaction monitoring	Cross-transaction patterns, SLO alerting	Alert records, metric series
Notification	Sender and beneficiary messaging	Message log
Reconciliation	Statement matching, discrepancy classification	Reconciliation items, ledger snapshot

5. Mathematical Model and Algorithm Design

5.1. Payment Provider Selection Model

Let x be the request context, E(x) the eligible set, and y_p(x) the indicator that provider p settles x within its committed window. The operator observes only an estimate of y_p(x). Define normalised cost c_p(x) and normalised latency t_p(x) by min-max scaling over E(x), and the instability term ρ_p as the absolute difference between the provider's short-window and long-window observed success rates, which proxies a provider deviating from its own baseline. The score is the governed linear combination of (1):

$$S_p(x) = \alpha \hat{y_p}(x) - \beta \bar{C_p}(x) - \gamma \bar{t_p}(x) - \delta \rho_p$$

Where α, β, γ, δ are non-negative weights that sum to 1. The weights are policy parameters, not learned parameters: they are chosen by the user, reviewed by risk and compliance, versioned and recorded for each decision made. This separation is the primary method used within the framework. The model provides a probability of what is occurring in the real world (a factual claim); while the weights represent a choice of preferences regarding cost, speed, reliability and stability (governance decisions). A system that learns the weights without user intervention cannot be audited. This is due to the fact that changes in routing may be due to either the real world or to user preferences; therefore, changes in routing will need to be addressed differently. The instability term prevents a quick reaction to a short period of time: a provider that has failed repeatedly but then succeeds in their last 20 attempts appears to be performing well according to a short-period statistic; however, the distance between periods will penalize them. It is bounded in [0,1], and δ is small, thus modulating rather than controlling.

5.2. Cost Optimization Model

Cost per attempt is defined in money and then expressed in basis points of the amount a, as in (2):

$$C_p(x) = f_p + \kappa_p a + s_p a + m_p, \quad bps_p = 10^4 \frac{C_p(x)}{a}$$

Where f_p is the explicit fee, κ_p the FX spread in basis points, s_p any settlement/liquidity charges, and m_p a fixed per-attempt network charge. Remittance costs are reported in basis points and are therefore used to compare with other published corridor statistics; the use of basis points also allows the cost term to be scale-free. The objective that is being minimized is expected cost per settled transaction (ECPST), Eq. (5) as opposed to cost per attempt. A cheapest quote policy will minimize the numerator while ignoring the denominator. For example, if provider A quotes 80 basis points and settles 90% of their attempts, and provider B quotes 100 basis points and settles 99% of theirs, then ECPST for provider A would be 88.9 versus ECPST for provider B at 101.0. Therefore, provider A remains cheaper; however, provider A has a 1-in-10 retry probability with latency and support implications that the cost metric does not account for.

5.3. Transaction Success Prediction

Settlement probability is estimated by a provider-conditioned model; the reference form is logistic with provider-specific coefficients over a shared feature map, Eq. (3):

$$\hat{y_p}(x) = \sigma(w_p{}^T \varphi(x) + b_p), \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

where ϕ(x) is the feature map of Table IV and w_p, b_p are estimated per provider with partial pooling towards a global vector for providers with few observations. Pooling is necessary because a newly onboarded provider has no history; shrinkage yields a defensible prior at cold start. Because the downstream use multiplies the probability by money, calibration is not optional: two models with identical area under the receiver operating characteristic curve (ROC-AUC) produce different decisions if one is overconfident. The framework therefore reports expected calibration error over M equal-frequency bins, Eq. (7), with the post-hoc map fitted on a validation period preceding the test period.

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)| \; |acc(B_m) - conf(B_m)|$$

5.4. Payment Routing Optimization

Selection is a constrained argmax over the eligible set, Eq. (4):

$$p^* = \arg\max_{p \in E(x)} Sp(x) \quad s.t.\, C_p \leq C^{max}, \; \hat{t}_p \leq T^{max}$$

where C_max and T_max are hard constraints supplied by the request's service level and corridor policy. The formulation is multiple-attribute utility maximisation with hard feasibility constraints, solved by exhaustive evaluation because |E(x)| is small: a realistic operator integrates tens of providers, and the eligible set per corridor is typically below ten, so exhaustive evaluation is both optimal and cheap. This is stated because routing literature sometimes proposes heuristics justified computationally, and that justification does not hold at this scale. Multi-attempt execution makes the objective sequential: after a failure, the eligible set shrinks and the probability of eventual failure after K attempts is the product in (5), which assumes conditional independence of provider outcomes and therefore understates correlated failures across providers sharing a rail, carried as limitation L4.

$$ECPST = \frac{\left[\sum_{i=1}^{K} c_{pi}\right]}{P_settled}, \quad P_fail(K) = \prod_{i=1}^{K} (1 - y_pi)$$

5.5. Machine Learning Algorithm

Algorithm 1 specifies training and Algorithm 2 decision-time inference combined with the ranking policy. Both are stated so as to be directly implementable and to make data-boundary conditions explicit, because the most common defect in published routing pipelines is temporal leakage: a feature computed over a window containing the outcome being predicted. The lag specification in line 4 of Algorithm 1 is the mechanism that prevents it.

Algorithm 1. Training and calibration of the settlement-probability model

```
Input: attempt-level event log L over [to,t1]; feature spec F; label horizon Delta;
    validation fraction v; test fraction u
Output: calibrated model M*, calibration map g, evaluation metrics
1:  E <- project L to (transaction, provider) pairs with p in E(x)
2:  for each attempt event e in E do
3:      y(e) <- 1 if status(e)=SETTLED within Delta else 0
4:      phi(e) <- F(e) using only events finalised strictly before start(e)
5:  end for
6:  (T_tr,T_va,T_te) <- chronological split of E with fractions (1-v-u, v, u)
7:  for each candidate family m in {logistic, forest, boosting} do
8:      fit m on (phi,y) over T_tr with provider partial pooling
9:      p_va <- predict_proba(m,T_va); record ROC-AUC, PR-AUC, ECE
10: end for
11: M <- candidate with highest ROC-AUC on T_va
12: g <- fit calibration map on (predict_proba(M,T_va), y_va)
13: p_te <- g(predict_proba(M,T_te)); report metrics on T_te once
14: persist (M,g,feature hash,training window,metrics) to model registry
15: return M*=g o M and registry entry
```

Algorithm 2. Constrained ranking with fallback and exclusion

```
Input: request x; eligible set E(x); exclusion set X; weights w; ceilings; K
Output: selected provider p* or terminal failure
```

```
1:  E' <- E(x) minus X; drop p if breaker(p)=OPEN or quote(p) expired
2:  if E' empty then return TERMINAL_FAILURE (NO_ELIGIBLE_PROVIDER)
3:  phi <- assemble_features(E',x); reject features older than staleness bound
4:  (y_hat,c,t) <- batch_inference(phi)            # one call, O(|E'|)
5:  normalise c and t over E'; compute rho_p for each p
6:  S_p <- alpha*y_hat_p - beta*c_p - gamma*t_p - delta*rho_p   for p in E'
7:  F <- {p in E' : c_p <= C_max and t_p <= T_max}
8:  if F empty then F <- argmin_{p in E'} c_p subject to t_p <= T_max
9:  p* <- argmax_{p in F} S_p; ties broken by lower c_p then lower t_p
10: key <- idempotency_key(transactionId,p*,attempt)
11: outcome <- adapter.execute(p*,key,deadline)
12: if outcome=REJECTED then X <- X + {p*}; goto 1 if attempts < K
13: if outcome=INDETERMINATE then resolve by status query before any retry
14: persist decision record (phi,S,p*,model version,policy version)
15: return p*
```

5.6. Complexity Analysis

Decision-time cost is dominated by inference. For N requests with eligible sets of size at most $|E|$ and feature dimension d, scoring a logistic model costs $O(N|E|d)$ operations, Eq. (8); a tree ensemble of T trees of depth h costs $O(N|E|Th)$, negligible against a provider network call, and ranking adds $O(N|E| \log|E|)$. Because $|E|$ is small, the dominant term is I/O, so feature assembly is one versioned cache read and inference one batched call per request.

$$O(N\,|E|\,d) \text{ scoring;} \quad O(n \log n \cdot d) \text{ training}$$

Training cost is dominated by the boosted candidate, approximately $O(n \log n \cdot d \cdot T)$, which for the dataset of Table 4 completed in seconds on one core; the constraint on retraining frequency is thus the availability of leakage-free history, not computation. The latency decomposition of Eq. (6) structures the measurement plan, with each hop timed separately so a regression can be attributed to a component.

$$T_{e2e} = T_{gw} + T_{orch} + T_{ml} + T_{prov} + T_{post}$$

6. Java Spring Boot Microservices Implementation

6.1. Technology Stack

The article describes a specific implementation and an established decision-making layer. The services described were not tested in a benchmarked cluster; therefore, there is no data reporting latency, throughput, or availability as measured. The technology stack used in this environment includes Java 21, Spring Boot 3.2, Spring Web for REST APIs, Spring Security for OAuth 2.0 Resource Server Enforcement, Spring Data JPA using PostgreSQL 16, Spring Cloud for Configuration & Discovery, Kafka for Asynchronous Event Processing, Redis for Quote & Idempotency Caching, Resilience4j for Circuit Breaking & Bulkheading, Flyway for Database Migration, Micrometer with OpenTelemetry for Observability, and Docker with Kubernetes for Deployment. There is a specific purpose for each technology in the stack. Kafka is required because callback and settlement confirmation events are asynchronous and need to be replayable. Redis is needed because staleness limits and idempotency keys exist on the performance-critical path. Resilience4j is necessary because circuit breaker status must be provider-scoped and visible.

6.2. Microservices Architecture

The twelve services defined in Table V provide APIs that represent their responsibility as well as the data they control. Instead of being decomposed based on technical layers (such as a single data access layer), the services were decomposed based on business capabilities. No common data access layer existed among the services, nor did any common entity layer exist, because using a common persistence layer would have re-introduced the dependency between services that the decomposition process eliminated. Figure 3 illustrates the layout: synchronous communications, which require immediate responses to allow the calling service to continue processing; and asynchronous events that allow for eventual consistency. There was one rule imposed upon the architecture: each service was restricted from creating a database transaction that spanned multiple services. Distributed transactions across multiple

vendor-provided databases cannot be made atomic, and attempting to create such transactions results in an unusable and incorrect system.

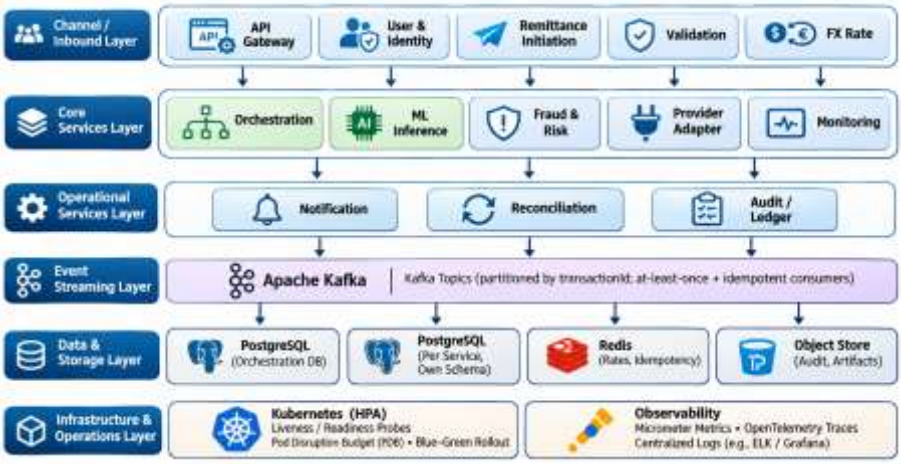


Figure 6. Java Spring Boot Microservices Architecture. Each Service Owns its Schema; Synchronous REST Calls Carry Decisions on the Request Path, and Kafka Topics Carry State Changes to Monitoring, Notification, Reconciliation and Audit Consumers

6.3. REST API Design

The external API is simple. A POST request to /v1/remittances with an Idempotency-Key header will return 202 and a tracking ID. A GET request to /v1/remittances will return the current state of the transaction and a cursor for event history. Quotes are retrieved via a GET request to /v1/fx/quotes. Orchestration provides visibility into what can be offered to customers, what decisions have been made, and why certain customer options were excluded. Inference processes a batch of feature vectors and produces a per-provider probability, cost and latency with model version and feature hash. Adapters allow for quote generation, execution, status checking and compensation. In addition, there are two conventions that are enforced: All endpoints that modify data require an idempotency key and store the hash of the body of the request. Therefore, if a customer attempts to send a duplicate request (same key, different body), the request will fail due to a conflict. Error types are classified into four categories: RETRYABLE, TERMINAL, INDETERMINATE, and POLICY_REJECTED. This allows clients to determine if an error is a result of a transient problem with a provider or if it is a result of a policy violation.

6.4. Database Design

There are five tables that comprise the orchestration schema: 1) transaction 2) attempt, one row per (transaction, provider, attempt) with an idempotency key and normalized outcome 3) decision, containing feature-vector hash, per-provider scores, selected provider, model version and policy version 4) fx_quote, containing reference rate, provider rate, spread, identifier and expiration 5) status_event, an append-only transition log. It is the separation of attempts from transactions that allows ECPST to be computable, as without an attempt record a retry would go unnoticed and the cost of failure would be impossible to attribute. Each decision refers to versioned provider configuration data; therefore, a repricing can be linked to routing behavior. Indexes are used for transaction identifier lookup on the hot path and range scan by provider and settlement date during reconciliation.

6.5. Event-Driven Communication

Events are past-tense facts with registered schemas: payment. initiated, payment.validated, risk.decided, provider.attempted, provider. responded, fx. observed, transaction.status.changed, settlement. confirmed and reconciliation.item.raised. Topics are partitioned by transaction identifier so events for one transaction stay ordered. Delivery is at-least-once and every consumer is idempotent, storing the last processed event identifier per aggregate; that is the contract that makes replay safe, and replay is required to rebuild a read model after a defect. Schemas evolve additively, a breaking change requiring a new event type rather than a version of an existing one. The outbox pattern is used where an event must not be lost: the state change and the outbox row are written in one local transaction, and a relay publishes afterward.

6.6. Transaction Consistency

Consistency is maintained through three approaches to distributed transactions. Idempotency Keys allow for safe execution of replay transactions; since each key includes the transaction ID, selected provider, and attempt number, any retry or redelivery of a message will result in the same operation being performed by the provider. Sagas provide a mechanism to compensate for multi-step processes, where each step has a compensatory action that can be performed in reverse order upon failure [9]. Optimistic Concurrency provides safety for transitions with concurrent handlers; a transition is only allowed to occur if the current row version matches the expected row version, preventing late callbacks from overwriting terminal states. The compensation vocabulary is intentionally asymmetric; once a transaction has been validated, it can be canceled; if it has been authorized, it can be reversed; if it has been settled, it can be refunded; and if an attempt remains unresolved, it can be queried for status. To treat these four forms of compensation as cancellations is what causes reconciliation failures.

6.7. Security Implementation

The authentication process uses OAuth 2.0 Client Credentials for Partner Integration, and Authorization Code with Proof Key for Code Exchange for User Channels. The validation of tokens takes place on the Gateway, and then the tokens are propagated as signed assertions as opposed to bearer tokens that can be replayed by any downstream service. Authorization is done at the resource level by the owning service and not just on the gateway because if there were only authorization on the gateway, then when a service was accessed via the mesh, it would fail open. Administrative endpoints use method-level security provider credentials are stored per provider in a secret manager and rotated. They are never logged. Audit records tokenize sensitive fields so that investigation is possible without revealing account identifiers.

6.8. Deployment Architecture

Each service is created in a container image that has a specified base image and does not run as root. The service is then deployed to Kubernetes using resource requests and limits, liveness and readiness probes, pod disruption budgets, and auto-scaling based on how many requests the service is handling and how saturated it is (rather than just how much CPU it uses). All configuration settings for each service are kept outside of the service itself and are versioned. The model file used for inference is loaded into the service with an immutable tag. If the hash value of the loaded file does not match what is listed in the manifest file, the inference service will refuse to run. Deployment is done through a blue-green strategy where there is a canary checking the success rates and distributions of decisions made by the new version of the service compared to those made by the old version of the service. Monitoring is set up around four user-visible metrics: settlement success rates, times to availability, decision latencies, and reconciliation break rates instead of host metrics. Listings 1 and 2 show examples of these patterns, but do not provide any evidence of their actual behaviors.

Listing 1. Illustrative adapter execution with idempotency and a provider-scoped breaker

```
@Service
public class ProviderExecutionService {
  private final Map<ProviderId, ProviderAdapter> adapters;
  private final CircuitBreakerRegistry breakers;
  private final AttemptRepository attempts;

  public AttemptResult execute(Decision d, Duration deadline) {
    ProviderAdapter a = adapters.get(d.providerId());
    CircuitBreaker cb = breakers.circuitBreaker(d.providerId().value());
    String key = IdempotencyKeys.of(d.transactionId(), d.providerId(), d.attemptNo());
    return cb.executeSupplier(() -> {
      try {
        ProviderResponse r = a.execute(d.payload(), key, deadline);
        return AttemptResult.of(normalise(r));      // ACCEPTED | REJECTED
      } catch (TimeoutException | TransientException e) {
        return AttemptResult.indeterminate(key);    // resolve by status query
      }
    });
```

```
  }

  @Transactional
  public void onProviderCallback(ProviderCallback c) {
    attempts.upsertByProviderRef(c.providerRef(), c.normalisedOutcome(), c.occurredAt());
    // outbox row written in the same local transaction; relay publishes after commit
  }
}
```

Listing 2. Illustrative governed ranking policy with immutable, versioned weights

```
public record PolicyWeights(BigDecimal alpha, BigDecimal beta,
                            BigDecimal gamma, BigDecimal delta, String version) {
  public PolicyWeights {
    BigDecimal s = alpha.add(beta).add(gamma).add(delta);
    if (s.compareTo(BigDecimal.ONE) != 0) throw new IllegalStateException("must sum to 1");
    if (Stream.of(alpha,beta,gamma,delta).anyMatch(w -> w.signum() < 0))
      throw new IllegalStateException("weights must be non-negative");
  }
}

public Optional<RankedProvider> select(Request x, Set<ProviderId> eligible,
                                       Set<ProviderId> excluded, PolicyWeights w) {
  List<Candidate> c = candidates.stream()
      .filter(p -> eligible.contains(p.id()) && !excluded.contains(p.id()))
      .filter(p -> !breakers.isOpen(p.id()) && p.quote().isValidAt(clock.instant()))
      .toList();
  if (c.isEmpty()) return Optional.empty();
  return c.stream()
      .filter(p -> p.costBps().compareTo(x.costCeiling()) <= 0)
      .filter(p -> p.latencyMin().compareTo(x.latencyCeiling()) <= 0)
      .max(Comparator.comparing(w::score))      // Eq. (1); ties by cost then latency
      .map(RankedProvider::of);
}
```

7. Experimental Setup and Evaluation

7.1. Experimental Objectives

The evaluation is based on four questions being answered at three different levels of evidence. First, E1 will determine if a learned model can discriminate settlement outcomes more effectively than a prevalence-only reference model. Second, E2 will determine if a policy derived from a learned model would result in selecting providers with expected costs per settled transaction that are less than those of static and/or rule-based policies applied to the same data trace. Third, E3 will assess how many instances of an orchestration tier need to be created in order to maintain a specified latency objective as load on the system grows. Finally, E4 will address how well a framework operates when there are provider outages and/or shifts in the underlying distributions of data. E1 and E2 were conducted as simulations, E3 was analyzed using analytical methods, and E4 was developed as a plan, and therefore is not reported as a result.

7.2. Experimental Environment

The execution part is a single-node simulation study on a two-core Linux container with 1 GB of memory, Python 3.12, NumPy 2.5, pandas 3.1, and scikit-learn 1.9, using regularized logistic regression, a random forest with 300 trees, and a histogram-based gradient-boosted classifier. There were no JVM, application server, database, broker, or orchestrator. Table 5 therefore separates the configuration that was executed from the configuration that was specified for the system-level evaluation.

7.3. Dataset Description

No real transaction data was used, obtained or accessed. The dataset is synthetic, generated with fixed seed 2025. The generator defines six providers with distinct fee, spread, latency and reliability parameters; eight corridors with an assumed coverage matrix; 60,000 transactions over 120 provider-day periods; and a settlement outcome drawn from a logistic model whose predictor combines provider skill, a provider-day autoregressive quality term common to all corridors, corridor effects, peak-hour effects, amount effects and a diurnal term. Provider-day telemetry was generated from the same latent term as 40 Bernoulli attempts per provider-day, and the learner received only lagged three-day and cumulative success rates computed from it, so it observes a noisy summary rather than the term itself. Because outcomes follow a specified process, separability is a property of that process, not of any real payment network: the reported discrimination shows the pipeline recovers a known signal, not achievable production accuracy.

7.4. Machine Learning Evaluation

The unit being predicted is the combination of transaction and provider that meets all requirements for being included in the study. All data was split into three parts based on time; the first part was used for training, the second for calibrating the model, and the final part was used for evaluating the model. The metrics that we will report include accuracy, precision, recall, F1, ROC-AUC, and the confusion matrix. Each metric will be evaluated at a fixed value of 0.5. We will also include ECE measurements before and after calibration. Although we used 0.5 as our evaluation threshold, this was not the operating point. Our policy used the probabilities generated by the model directly. Therefore, while we are providing precision and recall metrics as a means of reporting the performance of the model, they were not used to select the optimal model. Instead, the validation ROC-AUC metric was used to identify the best performing model.

7.5. System Performance Evaluation

The system-level design of the experiment is shown in Figure 7. It includes an open-loop workload generator, ingress and services, a provider simulator with user-defined latency and error injection, and a collection layer. In order to avoid selective reporting, the experiment's protocol is predetermined: the five-minute warm-up period is removed from analysis; the thirty-minute steady state at each load point is repeated three times; percentiles are calculated based on individual request histogram data rather than averages; the error rate is calculated by dividing failed requests by total requests; and resource utilization is measured every ten seconds and reported as the median and maximum over the steady state window. Because the experiment was never performed, only analytical capacity results are presented in Table 7, and the measured capacity column is marked as not reported.

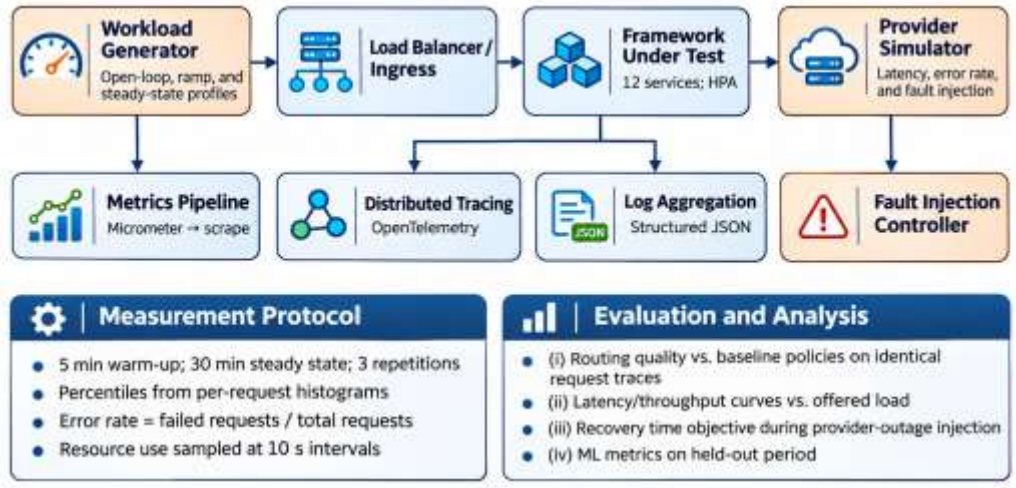


Figure 7. Experimental Evaluation Architecture. The Executed Part of this Study Uses Only the Decision-Layer Components; the Workload, Fault-Injection and Collection Paths are Specified for the System-Level Evaluation and were Not Operated

7.6. Comparative Evaluation

The study compares four policies on a single request trace with identical random numbers; therefore, any differences observed relate to selection, not to outcome sampling. The four policies being evaluated are: 1) Static lowest quoted cost; 2) Static highest long-run trailing success rate; 3) A rule-based policy which limits cost to the per-request median of all available providers and then selects

the one that has had the highest long-run success rate; and 4) The constrained scoring policy described in Equation (1) which uses the fitted model. A paired bootstrap analysis performed using 2,000 resamples provides a confidence interval for the difference in success rate between the proposed policy and the best performing static baseline.

8. Results and Discussion

All values in Sections 6-A and 6-B were produced by the simulation of Section 5-C and are labelled simulated in Tables 6 and 8 Section 6-D reports analytical capacity estimates. No column in this article reports a measured value from an operated system.

8.1. Machine Learning Results

Table 6 presents the results obtained using the held-out dataset. The regularized logistic model achieved an ROC-AUC of 0.723, compared to 0.500 for a reference based solely on prevalence, with accuracy of 0.672, precision of 0.700, recall of 0.751, and F1 of 0.724 for 38,968 held-out pairs. The gradient-boosted ensemble candidate achieved an ROC-AUC of 0.706 and an F1 of 0.696. It is interesting to note the relative performance of these two approaches. Given a small number of noisy features strongly structured around providers and corridors, the low-variance linear model performed better than the larger-capacity ensemble approaches. This supports the idea that sparse data favors simpler models, as discussed in Section II-C. Both models exhibited poor calibration. ECE values were 0.326 and 0.336 prior to post-hoc calibration and 0.300 and 0.327 subsequent to post-hoc calibration. The modest reduction in ECE suggests that miscalibration does not represent a simple monotonic distortion that can be corrected by applying a single-parameter transformation. Instead, it reflects the disparity between the base rates of training and evaluation samples and/or the mis-specification of the model with respect to the underlying provider-day factor. An ECE of 0.30 represents a significant deviation from what is acceptable for a system that uses probabilities to multiply monetary values. Therefore, this result provides direct evidence that a decision-making policy should never utilize raw classifier outputs. Instead, either the classifier outputs need to be recalibrated based on contemporary data or the policy must operate in terms of rankings rather than absolute probability levels. In short, this study produces a negative finding regarding the direct use of classifier outputs; therefore, the validity of E1 depends upon the pipeline producing a sufficient signal to rank alternatives within a candidate set but not yet sufficient to rank alternatives in terms of monetary value.

8.2. Payment Routing Results

Table 8 provides a comparison of routing over 8,700 held-out transactions. Static cheapest-quote had the lowest cost per attempt of 86.85 basis points; however, it had the lowest success rate of 0.558, resulting in an ECPST of 155.8 basis points. Static long-run-success had a success rate of 0.774 and cost per attempt of 103.35 basis points, resulting in an ECPST of 133.5. On this trace, the rule-based policy selected the same options as the static long-run-success policy since the median cost cap never bound: there was a sufficient degree of alignment between the prices and reliability rankings of the six providers so that the cost constraints did not exclude any provider that the reliability rules would have selected. This result is specific to the simulation, not to rule-based routing in general. It also illustrates how rule-based policies may appear appropriate under stable conditions, yet fail when prices and reliability diverge.

The proposed policy was chosen because it would have a mean cost of 100.33 bps and a success rate of 0.766, which results in an ECPST of 130.9 bps. This is a 1.9% reduction in cost compared to the best-performing static baseline, but it has a lower success rate by 0.79 percentage points. The paired bootstrap confidence interval for the difference in success rate is [-2.05%, +0.49%]. This interval contains zero, indicating there is no statistically significant difference in success rates between the two policies. In summary, the results indicate that the policy developed by machine learning provides a slight decrease in success rate in exchange for a slightly greater reduction in average cost per attempt. The overall effect on cost savings is small enough to be within the variability of a single simulation. The study demonstrated that the ML-assisted routing policy is viable and behaves in accordance with the cost model used to develop it; however, the study does not demonstrate that ML-assisted routing leads to better settlement outcomes. It should be noted that the ML-assisted routing policy was the only policy to incorporate short-window features to account for provider-day quality terms. Because all four policies received the same set of requests and experienced the same outcome draws, the comparison between them isolates the decisions made by each policy from any variations due to network issues.

8.3. Transaction Processing Performance

There are no latency/throughput measurements reported. However, we can describe how the latency budget is structured according to Eq. (6). The operator has control over the gateway, orchestration and inference hops, which consist of several in-memory

operations and a single batched inference call. The provider hop is outside of the operator's control, varies significantly from time to time, and is often the largest component of the latency budget. Therefore, efforts should focus on optimizing the provider hop; specifically, by choosing providers with low expected latencies, establishing short-term deadlines for calls made to those providers, and maintaining warm connection pools, rather than on fine-tuning internal operations.

8.4. Scalability Analysis

The analysis model considers the orchestration tier to be an M/M/c queue with c replicas; it has Poisson arrival rates (λ) and exponentially distributed service times (average = 18 ms); the size of the queue is designed to have utilization less than or equal to 70%. For request rates of 100, 250, 500, and 1,000 requests per second, the number of replicas needed would be 3, 7, 13, and 26, respectively. The utilization would range from 60% to 69%, the average waiting time in the queue would decrease from 5.32 ms to 0.12 ms, and the average response time would decrease to approximately 18 ms. These results represent analytical predictions and do not reflect empirical observations. Service times do not follow an exponential distribution; arrival rates during periods of high demand may not be Poisson; and the analytical model does not account for downstream resource constraints, connection pool limitations, and garbage collection-induced delays. Therefore, the purpose of these results is to illustrate the nature of the requirements for scaling, rather than providing achievable throughput levels. Based upon the results of the analytical analysis, we can conclude that the orchestration tier can scale horizontally because there is no state stored within the orchestration tier prior to the decision point and persistence point of each request. On the other hand, the adapter tier will be constrained by the rate limits established by the providers, which can be viewed as bulkheads per provider.

8.5. Reliability and Fault Tolerance

Reliability was not tested. The specified reliability behavior is that a provider's failure results in their removal from further eligibility determinations until a half-open probe is successful; if there is an indeterminate result, then the status will be checked prior to attempting again; a retry will result in selecting a different provider as the previous provider joins the exclusion list; when the list of eligible providers is empty, compensation will occur, instead of an infinite loop. The specified objective for recovery time is the amount of time the breaker remains open plus the length of a probe interval, and reconciliation can detect the remaining possibility of a payment being processed through a provider whose response was lost. It is being stated that the design prevents, by its very nature, the two types of failures that have the greatest economic impact: duplicate payments and unauthorized transactions that cannot be completed.

8.6. Comparison with Existing Approaches

The ability to compare the proposed system to existing published systems is restricted due to the fact that there are no common benchmarks available. Literature on institutional systems presents outcomes as opposed to decision-making processes [1], [3], [4], [5]; microservices literature presents the feasibility of architectures as opposed to the routing models used [6], [12], [13]; machine learning literature presents performance of classification algorithms for fraudulent activity and risk as affected by varying distribution and delay of labeling [10], [15]; and design literature presents saga semantics and resilience patterns as opposed to decision-making policies [9], [14]. To date, as far as the authors know, there has been no published study that simultaneously evaluates a provider selection policy and the supporting infrastructure necessary to implement the policy on a single trace. This paper will define the requirements for such an evaluation.

8.7. Limitations

L1: The evaluation uses a synthetic generator, so separability and the routing gain are properties of the generator. L2: outcomes are treated as immediately observable, so reported discrimination is an upper bound. L3: six providers, eight corridors and 60,000 transactions are far below production cardinality. L4: Eq. (5) assumes conditional independence of provider outcomes, understating correlated failures across providers sharing a rail. L5: Real provider APIs impose rate limits and change without notice. L6: spreads are fixed per provider, omitting intraday volatility and quote expiry. L7: corridor permissions are static eligibility, whereas in practice they depend on sender and beneficiary attributes and supervisory interpretation. L8: the drift monitor and shadow deployment are specified but not exercised. L9: twelve services with independent schemas impose an operational burden the framework does not quantify. L10: no threat model or penetration test was performed.

8.8. Research Implications

The conclusion for financial technology architects is that the decision layer and the execution layer must be kept separate from each other in terms of where they exist within a system at the service boundary and then connected through a persisted record of all decisions made; this provides reproducibility, which both regulatory oversight and dispute resolution require, at the expense of creating only one additional table and one new event type. Remittance operators should use their expected costs per successful transaction as their primary internal metric, and quoting cost per attempted transaction is misleading since it minimizes costs through using unreliable providers. Payment service providers must publish their attempt-level reliability along with price, as this is what a routing engine will need to make an informed decision about whom to route to. Therefore, providers who make this information available will be selected based on merit. When researchers conduct routing studies, they should include the trace used to collect data, as well as the weightings applied to the cost metrics used, since without these two elements, a study reporting improvements may simply be reporting a favorable trace or a permissive weighting.

9. Conclusion and Future Work

The scope of this article is narrow: how an operator should select among technically feasible providers when they do not know what will happen to the provider before selection, and the execution may fail. The framework answers in four steps. A constrained ranking policy uses a calibrated settlement probability estimate along with weightings for governor-set costs, latencies and stability, allowing for the distinction between learned and governed quantities. A basis point cost model enables comparability of the objective to published corridor statistics and provides for single computation of disclosures and decisions. A twelve-service Spring Boot decomposition confines provider heterogeneity to an adapter layer, creates the engine as the sole decision point, and delivers the mechanism that allows for safe execution of decisions. A persistence and event model records every decision with feature hash, model version and policy version.

The evaluation is purposefully partial. Using a simulated trace generated from a known process, a regularized logistic model achieved a held-out ROC-AUC of 0.723. In addition, a policy based on the model's output resulted in a reduction of mean attempt cost by 2.9% and ECPST by 1.9% relative to the highest-performing static baseline; however, the success rate difference (-0.79 percentage points) was within the bootstrap confidence interval containing zero. Post-hoc calibration only reduced ECE from 0.326 to 0.300, suggesting that raw classifier output cannot be used directly to make monetary decisions, and that calibration should be considered a production concern rather than simply a reportable detail. Analytical sizing suggests that the number of replicas grows linearly with offered load when a 0.70 utilization target is maintained. There are no latency, throughput, availability, or fault recovery measurements presented in this paper.

The limitations of this research are significant. This includes the use of synthetic data, immediate labeling of data, small scale of the study, conditional independence of the performance of each provider, static exchange rates, and lack of exercise of the drift and fault-injection mechanism. What survives these limitations is a design conclusion that separating the learned probability from the governed preference, and the decision from its execution, allows for an audit of the process of routing decisions. Future research will build upon the limitations of the current study. These areas include: conducting the system-level evaluation using both the load protocol and fault injection; evaluating the system at the corridor level by calibrating cost parameters based on aggregated statistical reports from other studies [1]; developing sequential formulations of the problem (e.g., contextual bandits, reinforcement learning) that optimize Equation (5) directly using off-policy evaluation methods against a logged trace; modeling label delays explicitly; retraining the model when drift occurs, based on a documented detector; developing an FX-aware orchestration method where the quote expiration is treated as a decision variable; providing explanations for all decisions made by the system; and federating operations across different operators while providing them with reliability statistics rather than raw transaction data.

References

[1] World Bank, Remittance Prices Worldwide, Issue 54, September 2025 (data reference period Q3 2025). Washington, DC, USA: World Bank Group, 2025.

[2] World Bank, Migration and Development Brief 40: Remittances Slowed in 2023, Expected to Grow Faster in 2024. Washington, DC, USA: World Bank Group, Jun. 2024. [Online]. Available: https://documents1.worldbank.org/curated/en/099714008132436612/pdf/IDU1a9cf73b51fcad1425a1a0dd1cc8f2f3331ce.pdf

[3] Financial Stability Board, "G20 targets for enhancing cross-border payments," FSB, Basel, Switzerland, 2021 (targets endorsed by G20 Leaders; implementation approach adopted 2022). [Online]. Available: https://www.fsb.org/work-of-the-fsb/financial-innovation-and-structural-change/cross-border-payments/g20-targets-for-enhancing-cross-border-payments-2/

[4] T. Lammer, D. Rees, and T. Rice, "Enhancing cross-border payments: state of play and way forward," BIS Bulletin No. 119, Bank for International Settlements, Basel, Switzerland, 2025. [Online]. Available: https://www.bis.org/publ/bisbull119.htm

[5] Committee on Payments and Market Infrastructures and Financial Stability Board, "Enhancing cross-border payments step by step: insights from the 2025 monitoring survey," CPMI Brief No. 13, Bank for International Settlements, Basel, Switzerland, 2025. [Online]. Available: https://www.bis.org/publications/cpmi-brief-13-enhancing-cross-border-payments-step-step-insights-2025-monitoring-survey

[6] F. Aydemir and F. Başçiftçi, "Building a performance efficient core banking system based on the microservices architecture," Journal of Grid Computing, vol. 20, no. 4, art. no. 37, Dec. 2022, doi: 10.1007/s10723-022-09624-z.

[7] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD '16), San Francisco, CA, USA, 2016, pp. 785–794, doi: 10.1145/2939672.2939785.

[8] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[9] H. Garcia-Molina and K. Salem, "Sagas," in Proc. 1987 ACM SIGMOD Int. Conf. Management of Data (SIGMOD '87), San Francisco, CA, USA, 1987, pp. 249–259, doi: 10.1145/38713.38742.

[10] A. Dal Pozzolo, O. Caelen, Y.-A. Le Borgne, S. Waterschoot, and G. Bontempi, "Learned lessons in credit card fraud detection from a practitioner perspective," Expert Systems with Applications, vol. 41, no. 10, pp. 4915–4928, Aug. 2014, doi: 10.1016/j.eswa.2014.02.026.

[11] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Computing Surveys, vol. 46, no. 4, pp. 1–37, Apr. 2014, doi: 10.1145/2523813.

[12] J. Lewis and M. Fowler, "Microservices: a definition of this new architectural term," martinfowler.com, Mar. 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html

[13] Spring, "Spring Boot reference documentation," VMware Tanzu / Broadcom, 2025. [Online]. Available: https://docs.spring.io/spring-boot/index.html

[14] Resilience4j, "CircuitBreaker — Resilience4j documentation," 2025. [Online]. Available: https://resilience4j.readme.io/docs/circuitbreaker

[15] Y. Lucas and J. Jurgovsky, "Credit card fraud detection using machine learning: a survey," arXiv:2010.06479, 2020. [Online]. Available: https://arxiv.org/abs/2010.06479

Appendix

TABLE 4. Experimental environment and configuration, separated into what was executed and what is specified for the system-level evaluation.

Item	Executed (decision layer)	Specified, not executed (system layer)
Hardware	2 vCPU, 1 GB RAM container	Kubernetes cluster, ≥3 nodes, ≥8 vCPU per node
Operating system	Linux (container image)	Linux nodes, pinned image digests
Language / runtime	Python 3.12	Java 21 (Temurin)
Framework	scikit-learn 1.9.1, pandas 3.1, NumPy 2.5	Spring Boot 3.2, Spring Cloud, Resilience4j
Database	—	PostgreSQL 16, one schema per service
Messaging	—	Apache Kafka 3.7, schema registry
Cache	—	Redis 7 (quotes, idempotency keys)
Container/orchestration	None (process executed directly)	Docker, Kubernetes with HPA and PDB
ML libraries	scikit-learn (LogReg, RF, HistGradientBoosting)	Same models, ONNX or PMML export
Testing tools	Paired bootstrap (2,000 resamples)	k6 or Gatling; Chaos Mesh for fault injection
Observation tools	Script-level metrics	Micrometer, OpenTelemetry, Prometheus, Grafana
Workload	Offline replay of the held-out trace	Open-loop generator, 5 min warm-up, 30 min steady state, 3 repetitions

TABLE 5. Dataset characteristics and preprocessing. Every figure in this table describes the synthetic generator of Section VII-C; no customer data was used.

Property	Value / description	Provenance
Nature of data	Synthetic, generated with fixed seed 2025	Simulated
Settlement of labels	Bernoulli draw from a logistic generating process	Simulated (ground truth known)
Providers modelled	6 (2 bank, 2 MTO, 1 mobile money, 1 card)	Assumed configuration
Corridors modelled	8 (US-MX, US-PH, US-IN, US-NG, DE-TR, AE-PK, GB-NG, SG-ID)	Assumed configuration
Coverage matrix	Provider × corridor eligibility, assumed	Assumption A1

Transactions	60,000 over 120 provider-day periods (500 per day)	Simulated
(transaction, provider) rows used	259,783 after discarding the 4-day feature warm-up	Simulated
Held-out transactions	8,700	Simulated
Positive rate (held-out)	0.578	Simulated
Amounts	Log-normal, median ≈ USD 270, clipped to [20, 15,000]	Simulated
Provider-day telemetry	40 Bernoulli attempts per provider-day, converted to 3-day and cumulative rates	Simulated
Latent quality term	AR(1), ρ = 0.85, per provider-day, shared across corridors	Simulated
Missing values	Cold-start rows omitted by the 4-day warm-up; no imputation applied	Design choice
Split	Chronological 70 / 15 / 15 (train / calibrate / test), single test pass	Design choice
Privacy	No personal data; synthetic identifiers only	—

TABLE 6. Settlement-probability model results on the held-out period (38,968 pairs). SIMULATED: values describe the recovery of a known synthetic signal and are not evidence of production accuracy.

Metric	Logistic regression	Gradient boosting	Prevalence-only reference
ROC-AUC	0.723	0.706	0.500
Accuracy (threshold 0.5)	0.672	0.654	0.578
Precision	0.700	0.700	—
Recall	0.751	0.692	—
F1	0.724	0.696	—
ECE (raw)	0.326	0.336	—
ECE (post-hoc calibrated)	0.300	0.327	—
Confusion matrix (TN, FP, FN, TP)	9429, 7192, 5574, 16773	10013, 6608, 6892, 15455	—
Selected for the routing comparison	yes	no	no

Rows are computed on a single chronological test split; no confidence intervals are reported for discrimination metrics. ECE uses 10 equal-width bins.

TABLE 8. Routing policy comparison on 8,700 held-out transactions with common random numbers. SIMULATED: the trace and the outcome draws are synthetic; all four policies faced identical requests and identical draws, so differences isolate the selection.

Policy	Settlement success	Mean attempt cost (bps)	Mean SSlatency (min)	ECPST (bps)
Static lowest quoted cost	0.558	86.8	10.1	155.8
Static long-run success rate	0.774	103.4	12.0	133.5
Rule-based (cost cap + long-run rate)	0.774	103.4	12.0	133.5
ML-assisted constrained scoring (proposed)	0.766	100.3	11.7	130.9

ECPST = mean attempt cost ÷ settlement success (Eq. 5). Paired bootstrap difference in success between the proposed policy and the strongest static baseline: −0.79 pp, 95%% interval [−2.05, +0.49] pp over 2,000 resamples. Policy weights: α = 0.60, β = 0.25, γ = 0.10, δ = 0.05 (ex-ante assumption, not tuned on the evaluation data).

TABLE 7. Capacity behaviour of the orchestration tier. Columns 1–5 are ANALYTICAL estimates from an M/M/c model with 18 ms mean service demand and a 0.70 utilisation target; they are not measurements. The remaining rows are the planned system-level measurements, which were not executed.

Offered load λ (req/s)	Replicas c	Utilisation ρ	Mean queue delay (ms)	Mean response (ms)
100	3	0.60	5.32	23.32

250	7	0.64	1.56	19.56
500	13	0.69	0.71	18.71
1000	26	0.69	0.12	18.12
—	—	—	—	not reported
—	—	—	—	not reported
—	—	—	—	not reported

Planned measurements (not obtained): median, p95 and p99 response time; throughput at saturation; error rate; CPU and memory utilisation; recovery time after provider outage. No value is reported for these rows.