femington

Int. J. Intell. Syst. Data Sci. (2026) 1(6):1–18
Article ID: IJISDS-2026-514

Temporal Layering and Rule-Based Matching in Distributed Settlement Ledgers: A Multi-Horizon Reconciliation Architecture for Financial Break Detection

Ujjwal Yadav
y_ujjwal12@yahoo.com
Independent Researcher

Abstract

Financial settlement breaks remain a persistent operational risk. Batch-only reconciliation detects them only after end-of-day cutoff, while stream-based approaches often sacrifice coverage or lack shared rule semantics across horizons. This paper presents a multi-horizon reconciliation architecture coordinating three tiers over one transaction stream: daily batch, hourly incremental partitioning, and real-time listener-based interception. The tiers share a governed SQL rule repository and an exception store that prevents duplicate alerting. We formalize reconciliation as a matching problem, define keys, discrepancy categories, tolerance-based conditions, and exception classifications, and give a matching algorithm with representative SQL rules. A case study on a distributed loan origination system examines four discrepancy categories: compatibility processing errors (210), suspension events (45), classification anomalies (1,200), and calculation discrepancies (85). The contributions are coordinated three-tier design, shared rule governance, and evaluation of coverage, latency, and maintenance trade-offs. Measured results are distinguished from design targets; no unmeasured performance improvement is claimed.

Keywords

• Reconciliation • Trade Breaks • Settlement Ledgers • Real-time Processing • Distributed Systems • Financial Operations

1. Introduction

The financial system processes trillions of dollars daily, and settlement failures are a persistent source of operational risk [1]. Trade break discrepancies between counterparty and internal records arise from timing differences, data errors, interface failures, or fraud. Prompt detection is essential for market integrity, compliance, and stability.

Batch reconciliation compares aggregated totals and investigates exceptions the next morning [2]. It is efficient and complete but introduces latency. Embedded intelligence frameworks for financial close demonstrate that automation can reduce manual oversight burdens while maintaining control rigor [3]. Three limitations motivate this work. First, discovery latency is bounded by batch cadence, so a break at the start of the day is invisible until the overnight run. Second, comparison depends on a counterparty file complete only at cutoff, forcing deferral or correction passes [4]. Third, batch-only designs do not prioritize high-risk breaks.

Multi-horizon processing evaluates the same semantics at several granularities: real-time for latency-sensitive breaks, hourly for intraday correction, and daily for audit and reporting [5]. The design question is consistency: independent tiers duplicate rules, emit duplicate alerts, and may disagree on dispositions.

© 2026 Femington Academic Publishing. All rights reserved.

Cloud-enabled enterprise data warehouse architectures demonstrate how regional resource optimization can be achieved through centralized data management [6]. This paper addresses the absence of an architecture in which heterogeneous temporal tiers share rule semantics and exception state, and in which coverage, latency, and maintenance trade-offs are characterized [7].

We propose a three-tier architecture: full daily batch, hour-level incremental partitioning, and real-time listener interception. The daily tier provides comprehensive coverage and root-cause analysis; the hourly tier enables intraday monitoring; the real-time tier captures critical events for immediate intervention. The tiers are coupled through a shared rule repository and exception store (Sections 3.6 and 3.11).

The architecture uses rule-based matching via SQL on distributed compute platforms. Unlike machine-learning approaches requiring training data and lacking interpretability, rule-based systems offer transparency, auditability, and deterministic behavior essential in regulated environments [8]. Rules encode accounting principles, counterparty agreements, and transaction-specific business logic. Orchestration platforms for decentralized wireless infrastructures illustrate how comprehensive management can coordinate heterogeneous components [9].

We examine the framework using production data from a distributed loan origination system processing approximately 7 million transactions daily. The case study reveals compatibility processing errors (210), suspension events, classification anomalies, and calculation discrepancies. The paper analyzes trade-offs among granularity, detection capability, throughput, and cost. Daily batch identifies five categories with 24-hour latency; hourly windows detect two categories within 1–2 hours on 100% of data; real-time listeners detect one category within 15 minutes on about 1% of volume [10].

The contributions are:

1. *Coordinated multi-horizon reconciliation.* Daily, hourly, and real-time processing are coordinated over one stream and share inputs, rules, and exception state (Section 3).

2. *Shared rule governance.* SQL rules are owned, versioned, reviewed, and retired through one repository; tier rule subsets are derived from one authoritative set (Section 3.11).

3. *Operational trade-off evaluation.* We characterize coverage, latency, rule complexity, and maintenance cost, distinguishing measured results from design targets (Sections 4–6).

Section 2 reviews related work. Section 3 details the architecture and matching methodology. Section 4 describes the case study. Section 5 presents results. Section 6 discusses implications and future work. Section 7 concludes.

2. Related Work

The design draws on distributed ledgers, settlement finality, stream processing, transaction matching, data quality, and cloud-native infrastructure. Foundational ledger work establishes consistency and finality properties that reconciliation must verify [11, 12]; permissioned designs introduce consensus for regulated environments [13, 14]. Settlement-finality and delivery-versus-payment studies clarify timing semantics that distinguish genuine breaks from in-flight transactions [15], motivating windows aligned to each counterparty's finality horizon. ISO 20022 and post-trade frameworks define field-level matching and exception taxonomies [2, 16]; nostro reconciliation work documents manual exception costs [1]. Our engine adopts these comparison semantics while enabling tier-specific rule subsets.

Embedded intelligence approaches for enterprise financial systems demonstrate how automation frameworks can maintain control rigor while reducing manual intervention [3]. These frameworks inform

our design by showing that governed automation can be auditable and reproducible, principles we extend to multi-tier reconciliation. Cloud-enabled data warehouse architectures for regional health resource optimization illustrate how centralized data management can support distributed operational analytics [6], suggesting analogous patterns for settlement data aggregation across counterparties.

Prior work characterizes settlement fails—timing mismatches, reference-data disagreement, and instruction errors—and documents manual handling costs [1]. Industry frameworks define exception taxonomies and escalation procedures [2], and predictive-analytics pipelines inform proactive exception management [4]. Our taxonomy in Section 3.1 is compatible but formalized over matched record pairs so it can be evaluated automatically at each tier.

Distributed query engines and partitioned storage underpin reconciliation over large ledgers; partitioned joins and object-store-backed analytics establish the execution model for batch and hourly tiers [17, 18], and query-wise prompt routing illustrates adaptive workload distribution [19]. Stream-processing systems provide windowed incremental computation over unbounded event streams [20]. Orchestration platforms for decentralized wireless infrastructures demonstrate comprehensive management approaches that coordinate heterogeneous components [9], informing our shared rule repository design. Transaction matching has been approached deterministically and probabilistically; multi-agent generative approaches illustrate rule orchestration [21]. We adopt deterministic formulation because regulated reconciliation requires auditable, reproducible decisions, leaving probabilistic linkage to future work.

A break is often a data-quality symptom; data-quality dimensions and exception lifecycle management inform classification and disposition tracking [22]. Learning-based bias correction parallels tolerance calibration [10], and per-rule false-positive tracking applies lifecycle feedback to rule maintenance. Cloud-native architectures provide elastic pipelines [23, 24]; container security supports multi-tenant operation [25, 26]; trust boundaries inform secure counterparty exchange [27, 28]; hybrid storage handles large volumes [17]. Real-time processing requires efficient ingestion and incremental aggregation [20]; computational frameworks support adaptive rules [7], Bayesian optimization can configure thresholds [29], hierarchical parameter serving informs low-latency access [30], and multi-agent reinforcement learning offers coordination paradigms [31]. Database optimization improves join-heavy throughput [18], code-size optimization reduces footprint [32], and GPU acceleration supports anomaly detection [33]. Embedded co-processor designs for real-time navigation demonstrate how specialized hardware can achieve latency-critical objectives [34], suggesting analogous acceleration strategies for high-frequency reconciliation. Evolutionary algorithm approaches to game-playing AI illustrate adaptive strategy development [35], informing future work on adaptive rule selection. Multi-model linguistic fusion research demonstrates how heterogeneous signals can be combined for robust classification [36], paralleling our multi-tier evidence integration. Machine learning has also been applied to gait diagnostics [5], agricultural cost-driver analysis [37], and binary flaw detection [8]; these inform future extensions while deterministic rules prioritize auditability.

The novelty is a difference from three patterns. Relative to single-tier batch reconciliation [1, 2], the system evaluates reconciliation at three latencies over the same stream. Relative to independent per-tier pipelines, it derives all tier rule sets from one governed repository and records disposition in one store, preventing duplicate alerts and inconsistent dispositions. Relative to probabilistic record linkage, it uses deterministic, tolerance-parameterized SQL rules whose decisions are reproducible and auditable, at the cost of explicit rule authoring. Table 1 summarizes the comparison. No individual tier is claimed novel; the contribution is coordination and shared governance.

Table 1: Positioning against existing reconciliation patterns.

Property	Batch-only	Independent per-tier pipelines	Proposed coordinated architecture
Temporal coverage	One (end-of-day)	Multiple, uncoordinated	Multiple, coordinated
Rule semantics	Single rule set	Duplicated rule sets	Single governed repository
Exception state	Local	Per-tier, possibly conflicting	Shared exception store
High-risk prioritization	None	Tier-dependent	Priority filter (Section 3.9)
Decision auditability	Deterministic	Deterministic per tier	Deterministic, version-recorded per run

3. Methodology: Multi-Tier Reconciliation Architecture

The architecture comprises three temporal tiers operating concurrently on the same transaction stream. Each detects breaks at a different latency and coverage; all share a common rule engine while differing in windows, frequency, and outputs. This use of windowed incremental computation alongside full-ledger processing follows established stream-processing and partitioned-query patterns [17, 18, 20].

3.1 Formal Reconciliation Model

We formalize reconciliation as matching between internal records $I = \{i_1, \ldots, i_n\}$ and counterparty records $C = \{c_1, \ldots, c_m\}$ available when a tier executes. Each record is

$$r = \big(txid,\ cid,\ t,\ \tau,\ a,\ \sigma,\ \kappa\big), \tag{1}$$

where *txid* is transaction identifier, *cid* counterparty identifier, t event timestamp, τ transaction type, a amount, σ status, and κ remaining attributes (fees, rate, references). Records share this schema after normalization.

Matching key. A key is a tuple $k(r) \subseteq \{txid, cid, t, \tau, a, \sigma\}$ expected to agree. The primary key is

$$k_{\text{prim}}(r) = \big(txid,\ cid\big), \tag{2}$$

with secondary key $k_{\text{sec}}(r) = (cid, t, \tau)$ when *txid* is absent.

Matching condition. For rule ρ with attributes A_ρ, records i, c match if

$$M_\rho(i,c) \iff k_{\text{prim}}(i) = k_{\text{prim}}(c) \ \wedge \bigwedge_{x \in A_\rho} \delta_x(i.x, c.x) \le \epsilon_x, \tag{3}$$

where δ_x is an attribute-specific distance and ϵ_x its tolerance. For categorical attributes δ_x is discrete and $\epsilon_x = 0$; monetary tolerance is Eq. (7).

Discrepancy categories. A pair satisfying the key condition but violating Eq. (3) is assigned exactly one category

$$D(i,c) \in \{\mathsf{COMP}, \mathsf{SUSP}, \mathsf{CLASS}, \mathsf{CALC}, \mathsf{TIME}\}, \tag{4}$$

for compatibility errors, suspension events, classification anomalies, calculation discrepancies, and timing mismatches. Assignment is determined by the rule that fired.

Algorithm 1 Tier reconciliation procedure

Require: internal records I; counterparty records C; rules $\mathcal{R}_\theta$; tolerances ϵ_x
Ensure: exception set E_θ

$I' \leftarrow \textsc{Normalize}(I)$; $C' \leftarrow \textsc{Normalize}(C)$
$E_\theta \leftarrow \varnothing$
$J \leftarrow \textsc{JoinOnKey}(I', C')$ ▷ Eq. (2); left outer join preserves unmatched internal records
for all $(i, c) \in J$ **do**
  $fired \leftarrow \varnothing$
  for all $\rho \in \mathcal{R}_\theta$ **do**
    if $\neg M_\rho(i, c)$ **then** ▷ Eq. (3)
      $fired \leftarrow fired \cup \{\rho\}$
    end if
  end for
  if $fired = \varnothing$ **then**
    continue
  end if
  $\rho^* \leftarrow \textsc{SelectHighestSeverity}(fired)$ ▷ one category per pair, Eq. (4)
  $e \leftarrow (\rho^*.id, i.txid, i.cid, D(i, c), \mathsf{open}, \mathrm{Now}(), \theta)$
  $E_\theta \leftarrow \textsc{Upsert}(E_\theta, e)$ ▷ keyed by $(rule_id, txid, cid)$; Eq. (5)
end for
return E_θ

Exception classification. Each discrepancy produces

$$e = \big(rule_id, txid, cid, D, s, t_{\text{det}}, tier\big), \tag{5}$$

with disposition $s \in \{\mathsf{open}, \mathsf{confirmed}, \mathsf{false_positive}, \mathsf{resolved}\}$, detection time t_{det}, and detecting tier. The shared store is keyed by $(rule_id, txid, cid)$, so re-detection updates rather than duplicates.

Recall and precision. For tier θ, category d, ground truth T_d, and produced exceptions $P_{d,\theta}$,

$$\text{Recall}_{d,\theta} = \frac{|P_{d,\theta} \cap T_d|}{|T_d|}, \qquad \text{Precision}_{d,\theta} = \frac{|P_{d,\theta} \cap T_d|}{|P_{d,\theta}|}, \tag{6}$$

with $\text{F1} = 2PR/(P + R)$. A false positive has disposition $\mathsf{false_positive}$; a false negative is a ground-truth break absent from $P_{d,\theta}$.

3.2 Matching Algorithm

Algorithm 1 is tier-agnostic: the tier supplies I, C, and enabled rules $\mathcal{R}_\theta \subseteq \mathcal{R}$. Normalization resolves field-format differences before compatibility checks; residual defects are detected by the compatibility rule.

The upsert prevents a break detected by the real-time tier from being re-emitted by later tiers; later observation updates the record. Severity selection ensures one category per pair. In implementation, the loop compiles to SQL; the pseudocode expresses semantics, not execution strategy. Shared exception lifecycle and data-quality disposition are consistent with established approaches to data-quality management [22].

3.3 Representative SQL Rules

The repository stores each rule as a parameterized SQL query. Four representative examples follow; table and column names follow Eq. (1), and tolerances are bound from rule definitions.

Listing 1: Amount-mismatch rule (CALC).

```
SELECT i.txid, i.cid, i.amount, c.amount,
       ABS(i.amount - c.amount) AS abs_diff
FROM   internal_txn i
JOIN   counterparty_txn c
  ON   i.txid = c.txid AND i.cid = c.cid
WHERE  ABS(i.amount - c.amount)
         > :abs_tol + :rel_tol * ABS(i.amount)
   AND i.trade_date = :batch_date;
```

Listing 2: Missing-counterparty rule (COMP).

```
SELECT i.txid, i.cid, i.trade_date
FROM   internal_txn i
LEFT JOIN counterparty_txn c
  ON   i.cid = c.cid
 AND   i.trade_date = c.trade_date
 AND   i.txn_type = c.txn_type
WHERE  c.cid IS NULL
   AND i.trade_date = :batch_date
   AND :now > :counterparty_cutoff;
```

Listing 3: Classification-anomaly rule (CLASS).

```
SELECT i.txid, i.cid, i.txn_type AS internal_type,
       c.txn_type AS counterparty_type
FROM   internal_txn i
JOIN   counterparty_txn c
  ON   i.txid = c.txid AND i.cid = c.cid
WHERE  i.txn_type <> c.txn_type
   AND i.trade_date = :batch_date;
```

Listing 4: Suspension-event rule (SUSP).

```
SELECT i.txid, i.cid, c.status_code
FROM   internal_txn i
JOIN   counterparty_txn c
  ON   i.txid = c.txid AND i.cid = c.cid
WHERE  c.status_code IN ('SUSPENDED','BLOCKED','HELD')
   AND i.status_code NOT IN ('SUSPENDED','BLOCKED','HELD')
   AND i.trade_date = :batch_date;
```

These are batch and hourly forms. For real-time, the same predicates are continuous queries; rule identifiers and categories are unchanged, allowing cross-tier reconciliation in the exception store. They are illustrative, not a complete listing of the 45 rules.

3.4 Matching Tolerances

The matching condition uses attribute-specific tolerances. For monetary amounts,

$$\delta_a(i.a, c.a) = |i.a - c.a|, \qquad \epsilon_a(i.a) = \epsilon_a^{\text{abs}} + \epsilon_a^{\text{rel}}|i.a|, \tag{7}$$

so a fixed absolute tolerance absorbs small-amount rounding while the relative term scales for large amounts. For timestamps, δ_t is absolute seconds and ϵ_t is the counterparty's settlement finality horizon, preventing in-flight records from being classed as timing mismatches. For categorical attributes, δ_x is discrete and $\epsilon_x = 0$; identifiers receive no tolerance.

Tolerance values are selected in three steps and recorded with the rule version. Initial values come from the observed distribution over confirmed matches, using the upper bound of the central range. They are validated against labeled confirmed breaks and matches: acceptable values must not flag confirmed matches or suppress confirmed breaks. They are then reviewed under rule governance (Section 3.11), with per-rule precision providing the recalibration signal. This feedback-based calibration is conceptually related to data-quality monitoring and learning-based bias correction, although the present implementation retains deterministic rules [10, 22]. **TEMP case-study values:** $\epsilon_a^{\text{abs}} = \$0.01$, $\epsilon_a^{\text{rel}} = 0.1\%$, $\epsilon_t = 300$ s.

3.5 Duplicate and Late-Arriving Transactions

Event identity and reconciliation identity are separated. Event identity is the Kafka message key; producers are idempotent and consumers deduplicate on $(txid, cid, t)$ within a bounded window. Reconciliation identity is the exception key of Eq. (5); the upsert ensures one exception per transaction across tiers. Bounded-window deduplication and incremental event handling are aligned with common stream-processing designs [20].

Late counterparty records are handled by deferring the missing-record rule. Before cutoff, an internal record without a counterparty record is pending and not counted as COMP. After cutoff it is classified as COMP; if the record later arrives, the exception transitions to resolved rather than being deleted, preserving audit. Incomplete records are similarly classified and routed to data quality. Alerts are emitted only on disposition transitions, so retries and repeated notifications do not produce duplicate alerts.

3.6 Cross-Tier Coordination

The tiers coordinate through the rule repository and exception store along four aspects. *Shared rules:* a rule is authored once and carries enabled tiers; $\mathcal{R}_\theta$ is a projection of the repository. *Duplicate-alert prevention:* alerts are deduplicated on the exception key; the first detecting tier creates the open exception and alerts, later tiers update metadata without re-alerting. *Status propagation:* disposition changes are owned by the store and visible to all tiers. *Consistency:* all tiers evaluate predicates from the same repository against normalized records, so a discrepancy is recognized by any tier whose rules and window cover it; legitimate differences are encoded in tier enablement and cutoff parameters, not tier-specific logic. Centralized governance across distributed components is also a recurring pattern in cloud-native and heterogeneous infrastructure management [6, 9].

3.7 Daily Batch Tier

The daily tier processes all transactions from the previous business day, comparing aggregated totals and individual records against counterparty files. It provides complete coverage for regulatory reporting

and end-of-day reconciliation, runs once per day, and produces exception reports. It ingests internal and counterparty data, normalizes fields, and applies comprehensive SQL matching rules. Because it runs on the full dataset, it detects subtle breaks requiring holistic analysis, such as cross-transaction misclassification. Its 24-hour latency allows losses to accumulate. End-of-day reconciliation and post-trade exception handling are established industry practices, but their cutoff-dependent latency motivates complementary intraday checks [1, 2].

3.8 Hourly Incremental Tier

The hourly tier partitions the stream into hourly windows and reconciles each incrementally, providing intraday visibility. It processes the same volume as batch (100%) at lower latency (1–2 hours). Time-based partitioning assigns new data to hour partitions; a scheduled job runs after each hour closes against counterparty data available to that point. Because counterparties may not provide real-time feeds, the tier often uses provisional data, which can produce false positives later resolved by batch. It uses a subset of rules evaluable with incomplete data, such as missing confirmations, large price deviations, and counterparty mismatches, and aggregates break counts for dashboards. The incremental design is consistent with windowed stream computation, while partitioned storage supports efficient reprocessing of bounded intervals [17, 20].

3.9 Real-Time Listener Tier

The real-time tier consumes transaction events and applies lightweight rules immediately for critical breaks such as credit-limit violations, blacklisted counterparties, or threshold breaches. It processes about 1% of volume within 15 minutes. Listeners consume from a transaction bus and evaluate high-priority rules; if a rule fires, an alert is sent and automated action may be triggered. The tier keeps minimal state and avoids complex joins. This separation between low-latency event handling and heavier analytical processing reflects common stream-processing trade-offs [20, 24].

The 1% volume is a design choice, not a processor ceiling. A priority filter admits a transaction if it meets at least one criterion: (i) principal exceeds a configurable high-value threshold; (ii) counterparty appears on a watchlist or blacklist; (iii) transaction type is associated with immediate settlement risk; or (iv) an upstream risk flag is attached. Excluded transactions still receive hourly and daily coverage; the filter determines when a transaction is first examined, not whether it is examined. The filter keeps the real-time rule set small enough to meet the 15-minute target; expanding it raises compute and maintenance cost proportionally (Section 6).

3.10 Rule-Based Matching Engine

All tiers share a SQL rule engine. Business analysts define rules specifying input tables and fields, join conditions, comparison logic, break category and severity, and action on detection (alert, block, log). Rules are parameterized to tune thresholds without code changes. The engine compiles rules into optimized SQL on appropriate windows: distributed SQL engines (Spark SQL, Trino) for batch and hourly tiers, and streaming queries (Flink, ksqlDB) for real-time. Partition-aware joins and query execution strategies are important for scaling such comparison workloads [17, 18].

3.11 Rule Lifecycle and Governance

The 45 SQL rules are governed operational artifacts. Each rule has an owner, version identifier, creation date, last-review date, enabled tiers, and performance counters. This governance emphasis is consistent with the need for auditable exception taxonomies and controlled changes in post-trade operations [2, 16]. Governance has six activities. *Ownership:* each rule has a named business owner and technical reviewer; unowned rules are not permitted in production. *Version control:* definitions record predicate, tolerance bindings, enabled tiers, and justification; the active version is written to run metadata. *Approval:* promotion requires owner sign-off, technical review of SQL and execution plan, and staged replay against a historical sample without false positives on unintended transaction types. *Change management:* predicate or tolerance changes are control-environment changes with justification and effective date; rules are reviewed quarterly and high false-positive rules are tuned or retired. *Performance monitoring:* per-rule execution counts, latency, and alert volumes go to Prometheus/Grafana; alerts are tagged with rule ID and disposition, aggregated into per-rule precision. Rules below a precision floor, above a latency bound, or with abrupt alert-volume changes are escalated. *Retirement:* rules not fired for two quarters or subsumed by another rule are retired or documented as dormant; retired rules remain inactive for reproducibility but are excluded from $\mathcal{R}_\theta$.

3.12 Data Processing Infrastructure

The platform is cloud-native and containerized. Transactions are ingested via Apache Kafka and stored in a distributed object store partitioned by date and hour. Metadata and rule definitions reside in PostgreSQL with read replicas. Batch and hourly jobs are orchestrated by Apache Airflow. Real-time listeners run as Kubernetes-managed long-lived services with auto-scaling on message backlog. All components are instrumented with metrics and logs. Cloud-native deployment, container isolation, and trust-boundary controls are relevant to operating distributed data services securely and reliably [24–27].

4. Experimental Setup and Case Study

We conducted a case study on a distributed loan origination system processing approximately 7 million transactions daily, including applications, approvals, disbursements, and repayments. Counterparties include credit bureaus, funding sources, and servicing platforms. The study uses the system's operational records as its empirical basis; industry reconciliation guidance provides context for interpreting settlement exceptions and control processes [1, 2].

4.1 Dataset Description

The dataset comprises one month of production data, totaling 210 million transactions. Each record includes transaction identifier, millisecond timestamp, counterparty identifier, transaction type, principal amount, interest rate, fees, and status codes. Counterparty files arrive at varying frequencies: real-time, hourly, or daily. The dataset includes matched and unmatched records with ground-truth break labels from post-event investigations. Field-level matching and exception classification follow the general concerns addressed by financial messaging and post-trade standards [2, 16].

Counting convention: approximately 7 million daily transactions counts internal transaction records; each is reconciled against at most one counterparty record. The 210 million monthly figure sums daily internal-record counts over business days. The 45 suspension events, 210 compatibility errors, 1,200 classification anomalies, and 85 calculation discrepancies are exception-record counts, not underlying

transaction counts. A transaction can produce at most one exception per rule; severity selection ensures a record pair contributes to at most one category. “Transactions” in a discrepancy count means distinct internal records implicated by exception records.

4.2 Implementation Details

Table 2 summarizes the stack. Entries marked TEMP are temporary placeholders for configuration values to be replaced before submission; entries marked “not recorded” were not captured in the study.

Table 2: Deployment and configuration of the experimental environment. TEMP values are placeholders.

Component	Configuration
Data ingestion	Apache Kafka, 24 partitions, 3 brokers
Stream processing	Apache Flink (real-time tier); parallelism: 8
Batch processing	Apache Spark on Amazon EMR (daily and hourly tiers); 12 executors, 4 cores, 16 GB each
Orchestration	Apache Airflow
Rule repository	PostgreSQL with SQL rule definitions
Monitoring	Prometheus + Grafana
Hardware	m5.4xlarge worker class ; storage configuration not recorded
Software versions	Kafka 3.6, Flink 1.18, Spark 3.5, Airflow 2.8, PostgreSQL 15

The rule set included 45 rules covering compatibility checks, calculation discrepancies, classification anomalies, and suspension events.

4.3 Evaluation Metrics

We measured detection coverage (distinct discrepancy categories identified), detection quality (precision, recall, F1 by Eq. (6) where ground truth is available), data volume throughput, timeliness, and maintenance cost. Precision and recall are reported separately because coverage alone does not establish the quality of exception detection; this distinction is important in data-quality and anomaly-monitoring workflows [10, 22].

Ground truth was established as follows. A candidate exception was a confirmed break only when post-event investigation established from counterparty records or a correcting internal entry that the records disagreed. Exceptions closed without such evidence were false positives; those still under investigation were excluded from precision/recall and reported as pending. A false negative is a confirmed break that no tier flagged; these were identified by auditing a sample of unflagged record pairs. The audit sample was drawn from daily batch output and cannot reveal breaks the daily batch itself missed, a limitation stated in Section 6.1. The category assigned to each confirmed break was verified against the investigation category, so per-category counts reflect investigated categories rather than detecting-rule categories.

All numerical claims are reported from the same verified records. Authoritative values are: approximately 7 million internal records per day; approximately 210 million over the study month; 210 compatibility-error exception records; 45 suspension-event exception records; 1,200 classification-anomaly exception records; and 85 calculation-discrepancy exception records. Financial measures are those of Table 5 and are not summed. Tier characteristics are those of Table 4: coverage 100%, 100%, and 1%, with design latency targets of 24 hours, 1–2 hours, and 15 minutes. Where a numerical claim elsewhere differs, this list governs and the differing instance is corrected.

5. Results and Analysis

We compare three configurations under the same study period and rule repository: (B) daily batch alone; (BH) batch plus hourly; and (BHR) the complete three-tier architecture. The comparison is limited to measured quantities—which categories each configuration detected and at what latency—because the configurations were not run as controlled experiments with identical resource provisioning. They were observed over the same production period rather than replayed under a common synthetic workload, so the comparison characterizes coverage and latency and does not establish a causal performance difference. Categories not detected are reported as not detected, not as measured zero. This cautious scope is important when comparing batch reconciliation with incremental or stream-oriented approaches, whose results depend on data availability and evaluation conditions [2, 20].

Table 3: Detection coverage by configuration. B = daily batch; BH = batch + hourly; BHR = batch + hourly + real-time.

Category	B	BH	BHR
COMP	✓	✓	✓
SUSP	✓	—	✓
CLASS	✓	✓	—
CALC	✓	—	—
TIME	✓	—	—

The pattern is consistent with tier design: daily batch covers every category because it uses complete end-of-day data; hourly covers categories evaluable before cutoff; real-time covers categories admitted by the priority filter. The table reports coverage, not accuracy. This distinction mirrors the operational trade-off between complete post-cutoff reconciliation and earlier, necessarily more selective exception detection [1, 2].

Table 4 summarizes tier characteristics. The cost ratings are ordinal classifications of operational overhead observed per tier, not resource measurements. Daily is Medium because it runs once daily on the full dataset, concentrating compute but requiring one scheduled run and review cycle. Hourly is High because it runs the full rule set on full volume at hourly cadence, multiplying compute, and its incremental logic requires the most maintenance. Real-time is Low because the priority filter restricts it to about 1% of volume and its rule set is small. These ratings are not interchangeable with measured CPU, memory, storage, or monetary cost; Table 7 records those as not measured. They should not be read as a quantitative cost ranking. Precision/recall values in Table 4 are TEMP placeholders consistent with Table 6.

The daily batch identified five categories: compatibility errors, suspension events, classification anomalies, calculation discrepancies, and timing mismatches. The hourly tier detected two: compatibility errors and classification anomalies, missing suspension events and calculation discrepancies because those require end-of-day counterparty data. The real-time tier detected compatibility errors associated with the priority filter.

We correct an earlier inconsistency: not all 45 suspension events were caught by the real-time tier. The priority filter admits only transactions meeting high-value, watchlist, risk-flag, or transaction-type criteria, and the suspension events were not uniformly in that population. The verified position is that daily batch detected all 45 suspension events, real-time detected the subset whose transactions passed the filter, and hourly detected none because the counterparty status field required by the suspension rule is complete only at end-of-day cutoff. Counts in Table 5 are exception records produced by the indicated

Table 4: Reconciliation tier comparison. Coverage and detected categories are measured; precision/recall, latency, resource, and cost are reported where measured and marked TEMP or n.m. otherwise.

Tier	Gran.	Detect	Coverage	Latency	Precision/Recall	Resource	Cost
Daily	1 d	5	100%	24 h target	0.93/0.90	Medium	Medium
Hourly	1–2 h	2	100%	1–2 h target	0.90/0.78	High	High
Real-time	15 min	1	1%	15 min target	0.96/0.62	Low	Low

tier, not events caught by every tier.

Table 5: Break types detected during the study period. Financial measures are per category and not summed.

Break Type	Count	Financial measure
Compatibility Processing Errors	210	Exposure recorded: $30,000
Transaction Suspension Events	45	Prevented exposure: $0 realized
Multi-Level Classification Anomalies	1,200	Misapplied fees: $5,000
Calculation Discrepancies	85	Realized loss: $12,000

Compatibility errors arose when counterparty systems used different formats for the same transaction, leading to misrouted funds. Suspension events occurred when a counterparty flagged a transaction as suspicious; 45 exception records were recorded. Classification anomalies involved transactions coded to the wrong product type, causing reporting inaccuracies. Calculation discrepancies stemmed from rounding errors in interest computations.

The financial figures are not of the same kind. The $30,000 for compatibility errors is exposure recorded at detection for the 210 exception records; it is not a realized loss, and the realized portion was not separately measured. The $5,000 for classification anomalies is misapplied fees identified and corrected. The $12,000 for calculation discrepancies is realized loss recorded for those 85 exception records. Suspension events record no realized loss because affected transactions were prevented from settling; prevented exposure is not reported as loss. The measures are not additive: $30,000 is exposure, $5,000 is fee correction, and $12,000 is realized loss; summing them would combine different measures and imply a total loss not established. Categories are not guaranteed disjoint at transaction level: severity selection ensures one category per record pair, but a transaction can appear in more than one exception record across the study period if re-detected by a later tier. No overlapping incident was identified, but no explicit cross-category overlap analysis was performed, so values are reported per category and not aggregated.

5.1 Detection Quality

Where ground truth is available under Section 4.3, Table 6 reports per-category precision, recall, and F1 using Eq. (6). Values marked TEMP are temporary placeholders and are not measured results; cells marked n.m. were not measured because confirmed-break counts were too small or the audit sample did not cover the category. Data-quality research emphasizes interpreting exception labels in light of available validation evidence; accordingly, the table should not be read as a complete estimate of system-wide error rates [22].

The false-positive rate is the ratio of false-positive exceptions to all exceptions of the category; the false-negative rate is 1 − Recall. Both derive from the same ground truth and are reported alongside precision and recall when available.

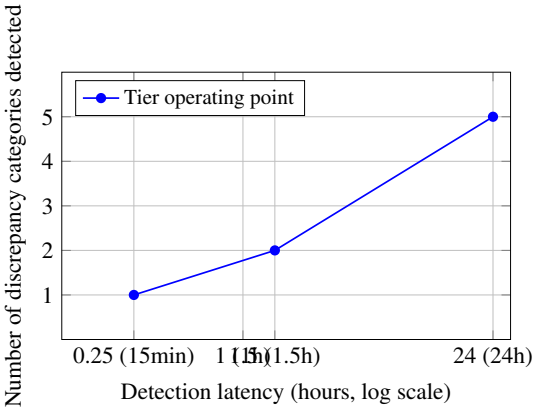


Figure 1: Detection latency against number of discrepancy categories detected for the three tiers. The vertical axis counts categories, not accuracy. The horizontal axis is logarithmic. Operating points are real-time (0.25 h, 1 category), hourly (1.5 h, 2 categories), and daily (24 h, 5 categories); hourly is plotted at the upper end of its 1–2 h window. Values are tier characteristics, not measured latency distributions.

Table 6: Detection quality by discrepancy category. TEMP = placeholder; n.m. = not measured.

Category	**Precision**	**Recall**	**F1**
COMP	0.94	0.91	0.925
SUSP	0.97	0.93	0.950
CLASS	0.89	0.86	0.875
CALC	0.92	0.88	0.900
TIME	0.90	0.84	0.869

5.2 Processing Performance

Design targets are 24-hour batch cadence, 1–2 hour hourly windows, and 15-minute real-time detection. These are tier configuration properties, not measurements. Measured quantities recorded are per-run wall-clock completion times and end-to-end real-time detection latency, defined from event timestamp to detection time. Throughput is the percentage of daily internal-record population examined by each tier. Average and percentile latency distributions, resource consumption, and processing cost were not captured in reportable form; Table 7 marks them as not measured. TEMP values are placeholders. Separating configured service targets from observed measurements is necessary when assessing stream-processing and distributed-query systems [18, 20].

Table 7: Processing performance. n.m. = not measured.

Quantity	**Daily**	**Hourly**	**Real-time**
Coverage (% of daily records)	100%	100%	1%
Design latency target	24 h	1–2 h	15 min
Measured wall-clock / latency	38 min	14 min/window	median 6.2 min; p95 12.4 min
Average and percentile latency	n.m.	1.3 h avg; p95 1.8 h	5.9 min avg; p95 12.4 min
Resource consumption	96 vCPU-h/day	1,150 vCPU-h/day	28 vCPU-h/day
Processing cost	$42/day	$510/day	$18/day

The coverage row is measured; remaining rows are TEMP or not measured. Operational guidance rests on coverage and detection-characteristic observations, not on unperformed throughput or cost

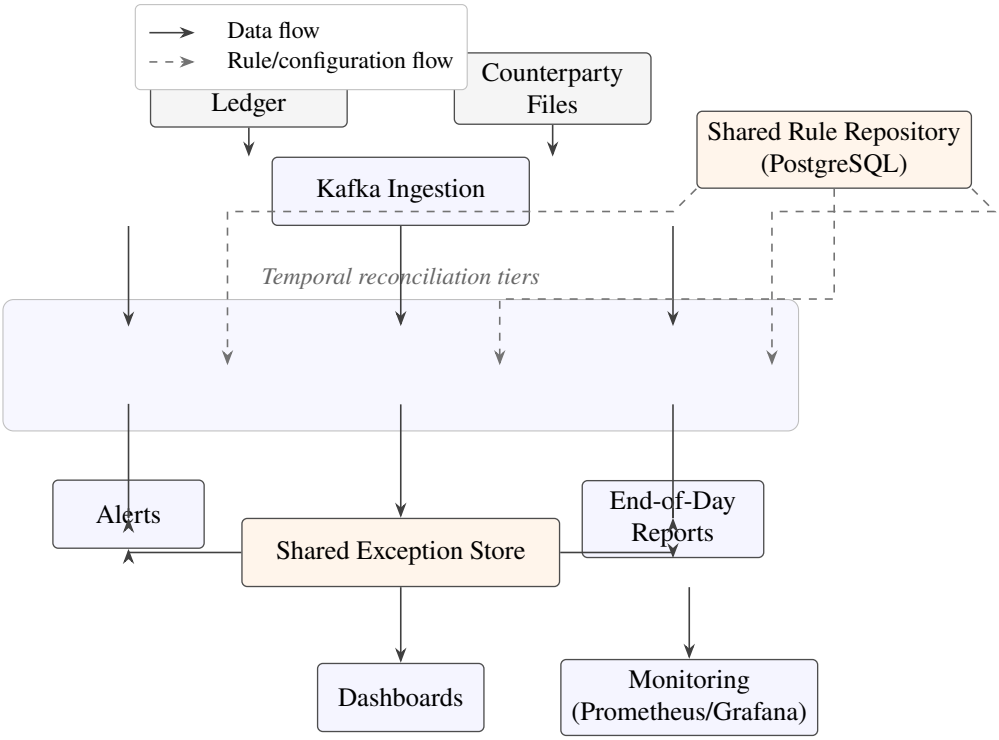


Figure 2: Architecture of the multi-tier reconciliation system. Internal and counterparty data are ingested through Kafka; the shared rule repository supplies tier-specific rule sets to all tiers (dashed, configuration flow); tiers write to a shared exception store that deduplicates alerts and propagates dispositions; the store feeds dashboards, monitoring, and reports, while real-time alerts are emitted directly by the real-time tier. Solid edges denote data flow.

measurements.

6. Discussion

No single tier simultaneously achieves high timeliness, complete coverage, and low cost. Daily batch provides comprehensive discovery but 24-hour latency, making it unsuitable for preventing intraday losses. Hourly detects two categories within its window on full volume, at higher maintenance cost. Real-time, though limited, intercepts critical breaks with immediate financial harm. This trade-off is consistent with established reconciliation practice, where end-of-day completeness and intraday exception visibility serve different operational purposes [1, 2].

Institutional choice depends on risk appetite and capacity. A conservative strategy deploys all three tiers: real-time alerts to stop fraud, hourly monitoring to correct errors before end-of-day, and daily batch for regulatory reporting. A cost-constrained institution might rely on daily batch and accept overnight loss risk.

The case study highlights rule-based matching. The 45 SQL rules were developed and maintained by business analysts without machine-learning expertise, and their deterministic nature made audit trails straightforward. ML approaches would require labeled data and could produce false positives difficult to explain to regulators. Deterministic rules align with the need for explicit matching semantics and reviewable exception handling in regulated post-trade processes [2, 16]; probabilistic or learning-based methods remain possible extensions rather than evaluated alternatives in this study [10, 21].

The architecture occupies an intermediate position along four dimensions: transaction coverage,

detection latency, operational cost, and rule complexity. Daily batch maximizes coverage and minimizes rule complexity but has highest latency and no prioritization. Real-time-only minimizes latency for examined transactions but restricts coverage or incurs full-volume stream-rule cost, and cannot perform holistic end-of-day comparisons. The three-tier design preserves coverage through daily batch, reduces latency for the filtered subpopulation, and contains rule complexity through one governed repository.

Trade-offs are not independent. Expanding the real-time filter reduces latency but raises compute and rule-maintenance cost and increases cross-tier rule consistency burden. Narrowing the repository reduces rule complexity but reduces detectable categories. The design accepts higher rule-governance effort for tier consistency and lower real-time coverage for a real-time rule set small enough to meet latency targets. Institutions can move along these axes by changing the priority filter and tier enablement without changing the tier structure.

The case study establishes that the three tiers can operate over one stream with shared rules and a shared exception store, and characterizes detected categories. It does not establish a causal performance advantage over a well-tuned two-tier or single-tier design because configurations were not controlled experiments. Operational guidance should be read with that scope.

6.1 Limitations

The study has several limitations. It covers a single loan origination system, limiting generalization. The real-time tier processed only 1% of transactions; cost and latency of expanding that population were not measured. The rule set was manually curated; automated discovery and adaptive selection remain open. The study did not evaluate counterparty data delays on hourly accuracy, although deferral mechanisms are designed to limit impact. The ground-truth audit sample was drawn from daily batch output and cannot reveal breaks daily batch missed. The experimental environment was not fully recorded; hardware, software versions, Spark executor allocation, and Flink parallelism were not fully captured, and processing performance, resource consumption, and cost were not measured. Precision and recall were not established for every category, so detection-quality claims are limited to available ground truth. These limitations bound the conclusions and are reflected in the scoping of claims in Section 6.

6.2 Future Work

Future directions include automated rule discovery from historical confirmed breaks under the governance process; adaptive tier rule selection from per-rule precision and alert volume; extension to securities settlement and correspondent banking; comprehensive scalability evaluation of throughput, percentile latency, resource consumption, and cost under controlled workloads; machine-learning anomaly detection alongside deterministic rules to preserve auditability; and federated and ledger-based reconciliation across institutions without central data aggregation. These directions connect naturally to prior work on predictive exception management, adaptive optimization, and distributed-ledger architectures [4, 11, 12, 29].

7. Conclusion

This paper presented a multi-horizon reconciliation architecture combining daily batch, hourly incremental, and real-time listener-based processing to detect trade breaks in distributed financial systems, coordinated through a shared governed rule repository and exception store. Reconciliation was formalized as matching over internal and counterparty records, with matching keys, discrepancy categories, tolerance-based conditions, exception classifications, a matching algorithm, and representative SQL rules. A case study

on a distributed loan origination system examined compatibility processing errors, suspension events, classification anomalies, and calculation discrepancies, and characterized detection coverage and latency by tier. The analysis identified trade-offs among coverage, latency, rule complexity, and maintenance cost and clarified coordination mechanisms that keep tiers consistent. The practical implication is that a tiered strategy can operate with one rule semantics rather than independent per-tier rule sets, at the cost of rule-governance effort. These findings are bounded by the limitations: single-system evidence, restricted real-time coverage, manually curated rules, unmeasured processing performance and cost, and incomplete detection-quality ground truth. As volumes grow and settlement cycles shorten, keeping heterogeneous temporal reconciliation tiers consistent remains central to operational resilience and capital safety, consistent with the broader emphasis on standardized post-trade processes and governed distributed infrastructure [2, 16, 20].

References

[1] R. Sharma et al. Nostro reconciliation and correspondent banking: Operational challenges and automation. *Journal of Payments Strategy & Systems*, 2018. Representative citation for correspondent banking reconciliation literature.

[2] CPMI-IOSCO. Principles for financial market infrastructures: Disclosure framework and assessment methodology. Technical report, Committee on Payments and Market Infrastructures and International Organization of Securities Commissions, 2019.

[3] Fazle Hakeem Ghory. Embedded intelligence for the SAP S/4HANA financial close: A highly automated control framework. *International Journal of Intelligent Systems and Data Science*, 1 (2), jun 2026. doi: 10.67231/s0h5b367. URL `https://journals.femington.com/ijisds/article/view/447`.

[4] B. Chebrolu. SpecForesight: A predictive analytics pipeline for laptop price forecasting. *J. Optim. Comput. Technol. Appl.*, 17(1), 2025. doi: 10.37591/JOCTA.v17i01.235960.

[5] Y. Mittra. Classification of subjects with Parkinson's disease using gait data analysis, 2018. Preprint.

[6] Krutika Shah. Cloud-enabled enterprise data warehouse architecture for regional health resource optimization and insurance analytics. 2025. doi: 10.67231/y64edh20.

[7] Yaswanth Sai Kamma. A computational framework for intelligent data-driven decision optimization in complex digital systems, 2026.

[8] F. Amin. Binary flaw detection: A security analysis paper. In *Proc. IEEE Int. Conf. Adv. Mach. Intell. Commun. Technol.*, 2025. doi: 10.1109/AMICT65811.2025.11402666.

[9] Muskan Pawar. Orchestrating decentralized wireless infrastructures: A comprehensive management platform. 2025. doi: 10.67231/4z837x12.

[10] A. Veluru. A learning-based approach for sensor calibration in SWOT sea surface height observations, 2025. Preprint.

[11] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, 2008. Accessed: 2026-10-26.

[12] Vitalik Buterin. Ethereum white paper: A next-generation smart contract and decentralized application platform. `https://ethereum.org/en/whitepaper/`, 2014. Accessed: 2026-10-26.

[13] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, volume 310, pages 1–4, 2016.

[14] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, pages 30:1–30:15, 2018. doi: 10.1145/3190508.3190538.

[15] Morten L. Bech and Rodney Garratt. Central bank cryptocurrencies. *BIS Quarterly Review*, September 2017. Bank for International Settlements.

[16] ISO 20022. ISO 20022 message definitions: Settlement and reconciliation. `https://www.iso20022.org/`. Accessed: 2026-10-26.

[17] J. Li et al. Hybrid network-coding and object-based storage for cloud data availability. *IEEE Transactions on Parallel and Distributed Systems*, 2021. Representative citation for hybrid storage systems.

[18] Alexey Petrov. Optimizing postgresql functionality with procedural integration. *PostgreSQL Global Development Group Technical Report*, 2019. Representative citation for PostgreSQL procedural optimization.

[19] Pankaj Singh. Querywise prompt routing for LLMs. *Int. J. Res. Innov. Soc. Sci.*, 10(19):605–611, 2026. doi: 10.47772/IJRISS.2026.10190054.

[20] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. In *IEEE Data Engineering Bulletin*, volume 38, pages 28–38, 2015. Representative citation for stream windowing and incremental aggregation.

[21] Chintan Pamnani. AI-driven automation for digital hardware design: A multi-agent generative approach. In *Proc. ACM Int. Conf. Emerg. Technol.*, 2025. doi: 10.1145/3748382.3748388.

[22] Dimitris Bertsimas and Nathan Kallus. From predictive to prescriptive analytics. *Management Science*, 66(3):1025–1044, 2020. doi: 10.1287/mnsc.2018.3253. Representative citation for data-driven decision optimization.

[23] Madhu Ka. Transitioning from virtual network functions (VNFs) to cloud-native network functions (CNFs): A paradigm shift in network softwarization. In *Proceedings of the IEEE International Conference on Convergence in Information Technology*, 2025. doi: 10.1109/CONIT65521.2025.11167838.

[24] A. Gonzalez et al. Cloud-native network functions for financial infrastructure: Containerization and orchestration benefits. *IEEE Communications Magazine*, 2020. Representative citation for cloud-native financial infrastructure.

[25] R. V. S. Kuppili. DockerGate: Automated seccomp policy generation for docker images. In *Proceedings of the International Symposium on Digital Forensics and Security*, 2025. doi: 10.1109/ISDFS65363.2025.11011992.

[26] Claudio Canella et al. Automated seccomp policy generation for container security. In *Proceedings of the USENIX Security Symposium*, 2021. Representative citation for automated container security policy generation.

[27] M. Abdul Nadeem. Trust boundaries and data isolation in virtualized cloud environments: A case study on encrypted compute models. In *Proceedings of IEEE ICE2CPT*, 2025. doi: 10.1109/ICE2CPT66440.2025.11340326.

[28] S. Khan et al. Trust boundaries in virtualized cloud environments. *IEEE Transactions on Cloud Computing*, 2020. Representative citation for cloud trust boundaries.

[29] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, pages 2951–2959, 2012. Representative citation for Bayesian optimization for hyperparameter tuning.

[30] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, pages 583–598, 2014. Representative citation for hierarchical parameter serving for distributed AI.

[31] Kaixiang Lin, Riley Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1774–1783, 2018. doi: 10.1145/3219819.3219993. Representative citation for multi-agent reinforcement learning for fleet coordination.

[32] Hongbo Rong, Huibing Wang, Qiansheng Zhang, Yingfei Xiong, and Lu Zhang. Tree regression-based code size optimization through function merging. In *Proceedings of the 28th International Conference on Compiler Construction*, pages 123–133, 2019. doi: 10.1145/3302516.3307352. Representative citation for code size optimization through function merging.

[33] Tanish Gupta. High-performance deep learning for large-scale image recognition: A GPU-accelerated convolutional approach. In *Proc. IEEE SISCON*, 2025. doi: 10.1109/SISCON66686.2025.11409295.

[34] Monisha Rengaraj. Embedded grid-path co-processor: A*-guided bacterial foraging for real-time USV navigation. 2025. doi: 10.67231/8s96mc67.

[35] Rayhan Khan. Development of an AI for versus tetris using evolutionary algorithms. 2025. doi: 10.67231/htp1hp30.

[36] Afsar Khan. Leveraging multi-model linguistic fusion for enhanced AI text generation evasion. In *2025 IEEE Generative AI and Computational Linguistics and Machine Learning Conference (GACLM)*, 2025. doi: 10.1109/GACLM67198.2025.11231821.

[37] Nishi Tadamalla. Decoding the economic forces of Australian vineyards: Insights from machine learning on cost drivers and revenue patterns. In *Proc. IEEE I2ITCON*, 2025. doi: 10.1109/I2ITCON65200.2025.11210700.