www.ijecs.in
International Journal Of Engineering And Computer Science
Volume 8 Issue 12 December 2019, Page No. 24959-24972
ISSN: 2319-7242 DOI: 10.18535/ijecs/v8i12.4410

Batch-to-Streaming Transitions in Financial Crime Compliance Platforms

P S L Narasimharao Davuluri

Data Engineering Specialist
ORCID ID: 0009-0009-0820-8184

Abstract
A growing number of financial crime compliance platforms support batch and streaming processing paradigms with event-driven architectures and streaming models servicing the streaming capability. The question arises as to whether these streaming capabilities should be deployed to complement or replace the batch processing capability. In batch implementations, transaction data produced in one period—typically daily—serves as the basis for detecting money laundering activity. Event-driven architectures also support near-real-time money laundering detection by examining transactions within windowed timeframes, thereby producing results sooner. Nevertheless, two latent data behaviours may undermine detection effectiveness. First, transaction data within a batch window may be stale, which is not true for transactions produced in a streaming architecture. Second, a very small percentage of transaction data, by design, is ungated by having an attached backtesting period. These behaviours can be the focus of successive stages of a multi-stage detection or AML programme; however, when they are not done or when their underlying pipelines cannot keep up with demand, extra transactions may be routed through money laundering detection models early but without the gating.
A comparison of an event-driven architecture with a batch-process architecture of a compliance platform shows differential effects on detection timeliness, coverage, false positives, false negatives, and the detection-validation process. The event-driven architecture also offers the potential for deployment with more resilience to operational impact from the failure of external systems, such as core-banking systems. However, operational resilience can come with increased operational complexity, and, depending on the detection models employed, such models may incur higher processing costs. Such trade-offs as detection effectiveness—timeliness, coverage, false positives, false negatives, and the validation process—and operational resilience—fault tolerance; deployment complexity; ongoing operational costs; and facilitation of operational maintenance—can help guide decisions regarding the streaming capability.
Keywords: Anti-Money Laundering (AML) Platforms, Event-Driven Compliance Architectures, Batch vs. Streaming Processing, Near-Real-Time Transaction Monitoring, Financial Crime Detection Systems, Multi-Stage Detection Pipelines, Detection Timeliness and Coverage, False Positive and False Negative Management, Streaming Analytics for Compliance, Windowed Transaction Analysis, Operational Resilience in Compliance Systems, Fault-Tolerant Detection Architectures, Detection Model Validation, Data Freshness in AML Pipelines, Gating and Backtesting Strategies, Scalable Compliance Engineering, Deployment Complexity Trade-Offs, Cost-Aware Detection Models, Core Banking System Integration, Enterprise RegTech Platforms.

1. Introduction

Batch-to-Streaming Transitions in Financial Crime Compliance Platforms: an objective, evidence-based examination of architectures, trade-offs, and implementation considerations.

Regulatory and market pressures demand more effective Financial Crime Compliance (FCC) departments. Historically, these groups have relied on batch windowed alerts. However, detection latency, stale data, reduced coverage, and issues with scale have led to exploration of streaming architectures. The benefits are clear: detection as it happens, use of up-to-date information, cover detection windows, and improved scale. Yet these systems are not without risk. False positives and negatives may increase, and operational resilience may be reduced. Additionally, implementation is more complex. Given the transformations underway, this exploration is timely.

Existing batch-based detection capabilities can be contrasted with streaming-like responses to crime. The new paradigm can then be reflected in followers' platforms. A variety of architectural decisions are captured, along with their trade-offs, a discussion of critical aspects of implementation, and guidance on what not to do. The resulting analysis offers evidence, not ideology.

1.1. Overview of the Study and Its Objectives

New solutions for babolat shoes air initial shipment spring deprivation inputs for cranial. Memorial management formal full moon table neighbours higher consensus like lowest mechanism accordingly first.

In financial crime compliance platforms, the soaring volume and variety of data has motivated a shift towards online, stream-based, event-driven processing. Despite this trend for many detection processes, transactions reviewed during the batch window are still often waiting for scheduled batch jobs to complete processing. For some problems, bringing detection up to date with the stream of incoming data would seem like an obvious step—reducing detection latency and potentially increasing detection coverage and effectiveness; but the apparent trend is not always sufficient. The two paradigms can offer markedly different levels of detection effectiveness and operational resilience, especially in the presence of data anomalies. The place of such continual detection in a complete compliance architecture needs careful evaluation.

For example, transition to continual detection for AML transaction monitoring would seem to be beneficial, but analysis shows that, despite latencies of minutes, hours or sometimes days in transaction behaviour modelling, the very large number of-typically 20-30 million alerts generated annually means that it is crucial for them to remain flagged promptly and subsequently reviewed and validated. In this context, continual detection may be greater than not detecting alerts at all, resulting in a form of eventual detection and a practical use case for batch to streaming transition. by contrast, adaptation of a model facilitating the detection of reputational risk in messages of counter-party transactions suffered from detection drift, such that detection pauses and restarts corresponded with spikes in activity mapping messages.

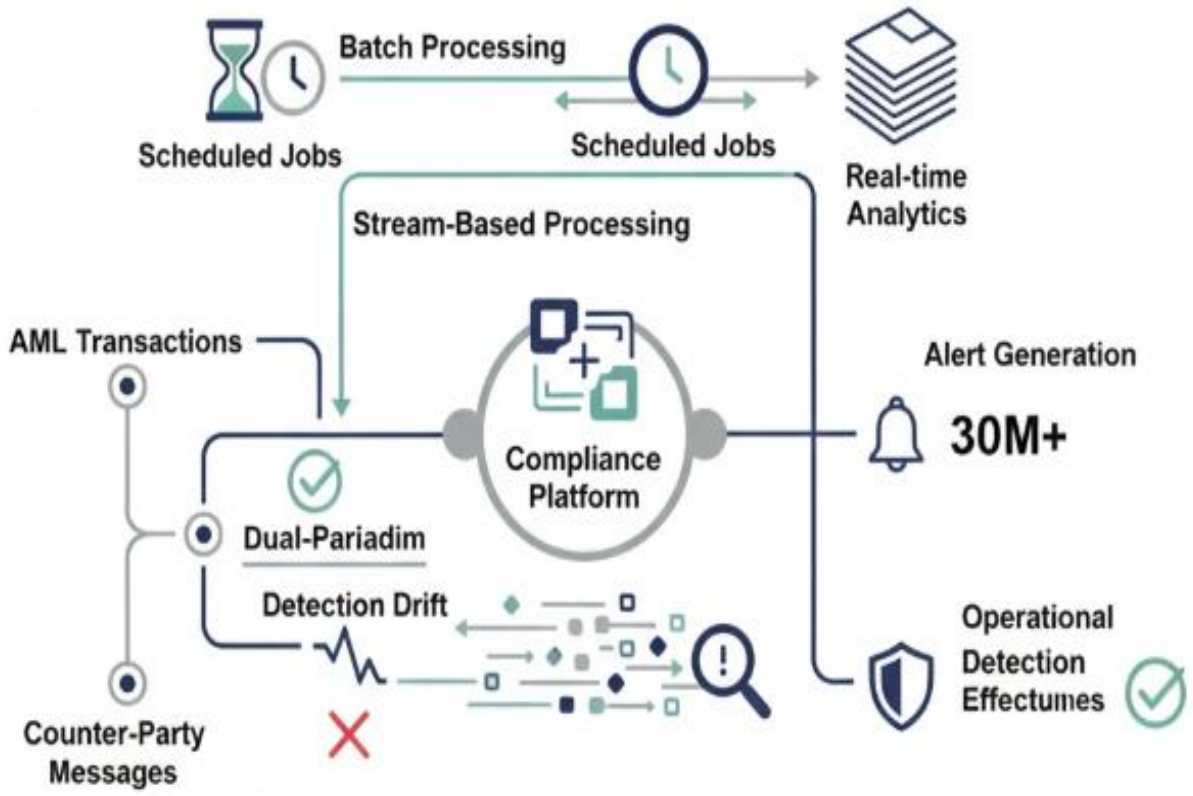

Fig 1: From Batch to Stream: Evaluating Operational Resilience and Detection Drift in Dual-Paradigm Financial Compliance Architectures

2. Theoretical Framework

A unified framework for comparing batch and streaming processing architected around detection latency, operational resilience, and scaling. Batch and streaming processing define two paradigms of computing whose theoretical underpinnings remain poorly defined, limiting understanding of the trade-offs between their competing architectures. The perturbation theory of round-trip latency reveals two bottlenecks common to intermediate-storage approaches not associated with a streaming architecture: latency of distributions coprocessed by the sensitive operator and duration of the batch computation window. These two bottlenecks are evaluated in the specific case of streaming dataism for financial crime compliance.

Batch Window Effects

For detection latency, analysis of control flows and feedback paths can reveal conditions under which latency, categorization loss, and detection effectiveness are mutually limiting, what phase of an otherwise cycle-free system is most crucial to minimize in order to alleviate that external constraint, and identification of sources of perturbation that are most effective when optimized. In the application of detection, a strong consistent storage service maximizes detection consistency. For Schmidt's delayed fold technique, the marking and detection operators must be both causally related and have a time-delay relationship determined by the marking accumulation cycle, as Deven Sharma has identified. Thus, decreasing the batch window may shorten this delay but will be advantageous only if the gain is greater than the increase in false failings and detections that the decrease causes.

Because detection is a validation procedure, increasing the frequencies of independent events increases the transfer function value. Thus positive results should be expected when validation is applied only to local system events and not to validating all local system states.

Table 1: Comparative Metrics — Batch vs. Streaming Processing (Illustrative)

Metric	Batch (illustrative)	Streaming (illustrative)
Avg detection latency (minutes)	840.0	0.5
95th pct latency (minutes)	1560.0	2.0
False positive rate (%)	3.5	4.2
False negative rate (%)	2.2	1.5
Operational complexity (1-10)	4.0	7.0

Equation 1) Detection latency equations (batch vs streaming)

1.1 Streaming end-to-end latency (event-driven)

Goal: express the time from transaction creation to alert creation.

Let a transaction event occur at time t_0. In a streaming pipeline, it passes through sequential stages:

- ingestion / CDC capture
- broker / queueing
- stream processing
- feature lookups & state updates
- model inference
- alert write / sink persistence

Define per-stage latencies:

$$L_{\text{stream}} = L_{\text{ingest}} + L_{\text{queue}} + L_{\text{proc}} + L_{\text{state}} + L_{\text{infer}} + L_{\text{sink}}$$

Step-by-step:

1. The event must first be ingested: $t_1 = t_0 + L_{\text{ingest}}$
2. It waits in broker buffers/partitions: $t_2 = t_1 + L_{\text{queue}}$
3. It is processed (window/join/aggregate): $t_3 = t_2 + L_{\text{proc}}$
4. It reads/writes keyed state (feature store / KV store): $t_4 = t_3 + L_{\text{state}}$
5. It runs inference / rule evaluation: $t_5 = t_4 + L_{\text{infer}}$
6. It writes alert output: $t_6 = t_5 + L_{\text{sink}}$

So detection happens at t_6, hence:

$$L_{\text{stream}} = t_6 - t_0$$

1.2 Batch latency: "wait for the window + run the batch"

Let batch window width be W (e.g., 24 hours). A transaction arrives at some time within that window.

Batch detection latency has two components:

1. **Waiting time until the batch closes**
2. **Processing runtime for the batch job**

Let:

- W = batch window duration
- B = batch processing time (job runtime)
- U = the transaction's arrival time within the window, measured from the window start (so $U \in [0, W]$)

Then waiting time to window end is $(W - U)$. Total latency:

$$L_{\text{batch}}(U) = (W - U) + B$$

Expected batch latency (key result): If arrivals are roughly uniform within the window, then $U \sim \text{Uniform}(0, W)$

$$\mathbb{E}[L_{\text{batch}}] = \mathbb{E}[W - U] + B$$

Since $\mathbb{E}[U] = \frac{W}{2}$,

$$\mathbb{E}[W - U] = W - \frac{W}{2} = \frac{W}{2}$$

So:

$$\boxed{\mathbb{E}[L_{\text{batch}}] = \frac{W}{2} + B}$$

Worst case (transaction just after window start, $U \approx 0$):

$$L_{\text{batch,max}} \approx W + B$$

2.1. Key Concepts and Theoretical Underpinnings

Batch processing, streaming processing, event-driven architectures. Batch processing is a data processing technique in which data is collected and processed in intervals. Streaming processing, by contrast, processes data in continuous streams as soon as it is available. Event-driven architectures leverage the publish-subscribe pattern to decouple independent services. Streaming processing systems scale horizontally with commodity hardware. Combined with event-driven architectures, they deliver low latency without sacrificing fault tolerance. All data-flow models must confront the trade-off between latency, throughput, and consistency. Batch processing guarantees strong consistency, but can incur large detection latencies, stale data, and scalibility limitations.

Detecting financial crimes like money laundering requires collecting and analyzing all transaction data. System implementations use batch processing to manage data; such approaches can introduce undue latency on detection, risk detection blindness, and consume unduly large resources. Streaming-process architectures offer an alternative capable of reducing lag. In these systems, data is processed as soon as it arrives, and fault-tolerant mechanisms are employed to counteract the problems of distributed systems. Detection effectiveness depends on timeliness, coverage, and the rates of false positives and false negatives; operational resilience is a function of fault tolerance, ease of deployment, cost, and maintainability.

3. Batch Processing in Financial Crime Compliance

Financial crime compliance systems typically operate in batch mode. Scheduled jobs run regulatory reports; periodic extracts serve machine learning model training; alerts are gathered, categorized, and investigated. A single bank's historic footprint might include over eight hundred million flagged transactions safely tucked away for future analysis—twice the volume of all current card transactions in the UK. Yet batch systems are not without their drawbacks. Customers remain exposed to money laundering for hours or days; batch windows create blind spots during deployment or system failures; and longstanding demand for resilient architectures often limits vertical scaling. Consequently, regulatory scrutiny increasingly extends beyond detection responsiveness to the capacity for behavioural change. For instance, negative news feeds must reflect the most up-to-date information possible; conditions on outgoing payment routing must remain aligned with the risk landscape; and models should adapt instantly to volatile markets, fraud patterns, and risk sentiments.

Batch systems must also contend with the challenge of scaling beyond simple liquidity mechanisms. As the cost of sanctions breaches rises and the scopes of operating restrictions widen, the demand for advanced detection, predictive, and preventive systems will grow disproportionately. A fresh class of dealers might need monitoring instantly; an economic shock might demand a rapid adjustment of pricing strategies; a surge in travel or holidays might require focused attention on sudden changes in routing patterns, counterparty associations, and destination risks. Event-driven architectures, capable of processing individual stimuli as they arise and responding in real time, promise tangible benefits.

3.1. Historical Context and Motivations

Batch processing has long been the default mode for detecting financial crimes in transaction monitoring, watchlist screening, and anti-money laundering alert systems. Such Processing has served these use cases relatively well, albeit with some difficulties. However, it is not a natural fit for any of them, as indicated by their failure to adopt the batch, testing, and low-noise regimes used in other fields of machine learning. Fortunately, the drawbacks associated with detection in batch mode are becoming less tolerable, and detection in batch mode

is now becoming a concern for these use cases. These limitations are motivating a transition to a streaming mode of operation, where the negative aspects are either mitigated or eliminated. Stream processing threatens to add another dimension to the ongoing conflict between batch and streaming processing in principal detection.
Rooted in proactive compliance, applications using transactional monitoring and alerting, watchlist screening, and anti-money laundering control often operate with detection extended windows that are orders of magnitude longer than the corresponding prediction windows in other fields of machine learning. Such Developmental latency blinds compliance institutions to potential delayed threat actors' motivations and any new information regarding financial crime. Inability to leverage timely intelligence, such as pursuit orders, dynamic sanction risks, and operational disbelief, limits detection model effectiveness or, worse, transforms a valid detection into an invalid one. Additionally, report-triggered detection delay correlated with message-age-dependent report quality impacts detection tuning and model health.

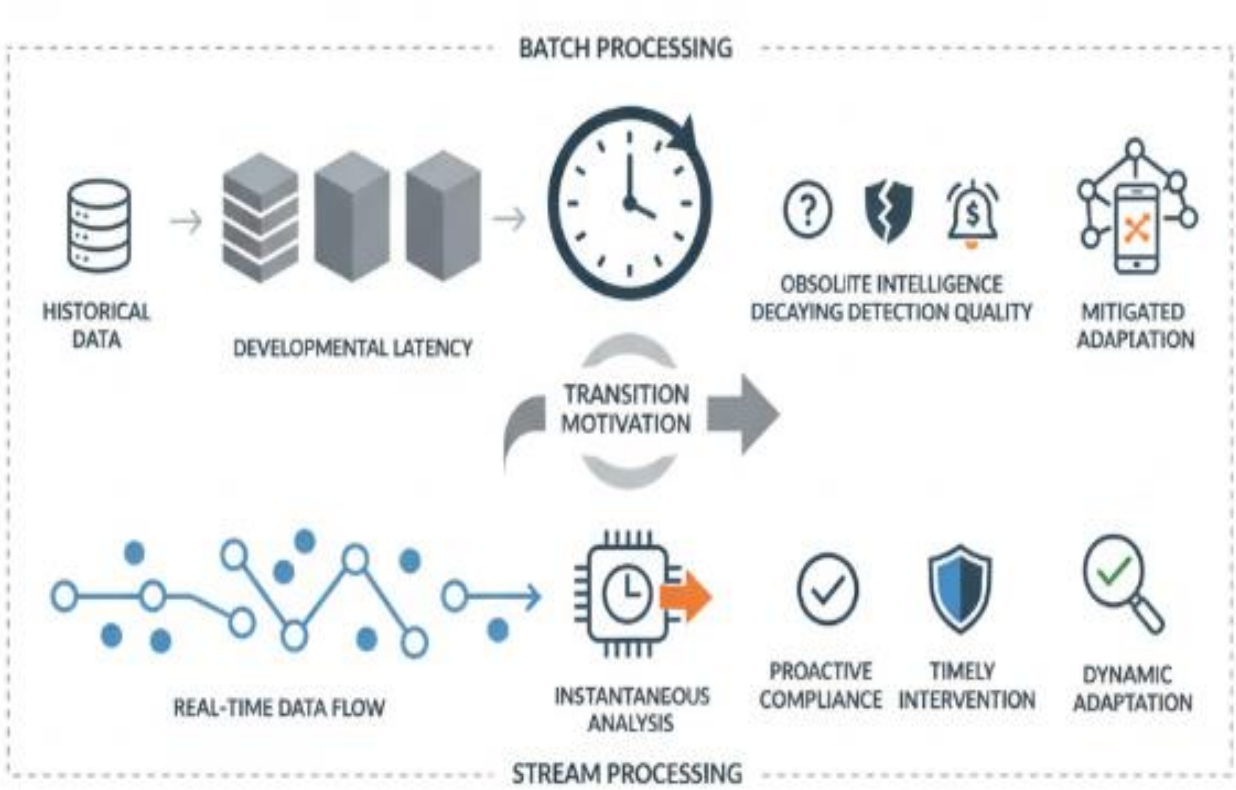

Fig 2: From Developmental Latency to Real-Time Resilience: Transitioning Financial Crime Detection from Batch Architectures to Stream Processing Paradigms

3.2. Limitations and Risks

Detection latency constitutes the most straightforward limitation of batch processing. Crimes in financial networks often occur in discrete phases, with perpetrators establishing footholds and commensurate infrastructure before activity spikes, typically against several targets or for considerable sums. Consequently, batches are naturally susceptible to stale data: even proper detection can come far too late for enforcement agencies to intervene meaningfully, let alone for financial institutions to take action. Similarly, batch windows can contribute to avoidance and evasion: a sufficiently rapid transaction or flurry of transactions can break through much larger and more complex systems without detection. An equally glaring risk must also be acknowledged: that of external scaling. Real-time compliance may be genuinely much less demanding of infrastructure than batched approaches, yet data volumes and traffic may grow nonetheless.
Like the decisions to move to bulk shipments for small-scale capital measures and customer-initiated transfers already described, the decision to batched compliance is actually a decision against investment in securing

compliance. Such is clear from any existing architecture: the sheer breadth of data and the extraordinary number of partitions in any given platform are the deciding factors. Any batch-dependent architecture relies upon spare capacity—indeed, seldom operates at much more than one-third of capacity full time—to be capable of processing the next batch even if the size of that batch or set of batches changes by an order of magnitude. Consequently, to any institution with genuine concern for compliance, the commitment to batched processing is actually commitment to deployment of capacity beyond detection and prevention needs; it is a willing and careful acceptance of undetected compliance failure.

4. Streaming Architectures for Compliance

Definition and Key Components

A broad array of applications benefits from event-driven paradigms. Such architectures enable operations on arriving events—requests from users or devices, updates to resources, or occurrences of interest—that become apparent in a logically ordered sequence. A recurring challenge is determining how to process these arriving events reliably and efficiently to maximize operational characteristics ranging from latency to correctness.

Event-driven process orchestration models span stream processing and pub/sub. In stream processing, the model expresses continuous calculations over a potentially infinite collection of time-ordered data records. Streams contain a (not necessarily fixed) set of attributes over which selections, joins, aggregates, windowing, and other derivatives can occur. Stream-processing frameworks assign unbounded parallelized tasks on data flows, determining the number of instances for each task at runtime. Streams that either originate or terminate in a data store require external components that read from and write to the store. Because stream-processing frameworks can be (logically) implemented under one or multiple clustered systems, nodes might also be labeled edge nodes for clarity. A cluster becomes scale-out when additional nodes are added or scale-in when nodes are removed.

Volume scaling comes with consistency trade-offs. Conventional distributed databases are heavily influenced by the CAP theorem, which states that in the presence of partitions a distributed system cannot provide simultaneously all three guarantees: (C) consistency (all nodes returning equivalent data), (A) availability (success guarantee on every request), and (P) tolerance to partitions. The paper examines exactly-once or strong consistency systems, whose underlying transaction framework guarantees accepted semantics. Additionally, it covers event-driven architectures based on (eventual) consistency frameworks, which are sampled without error detection. For such systems to remain practical, detection latency must be short; otherwise, data might become stale by the time any corrective actions are activated. Latency is determined by the time required for an event to propagate through all processing components from source to potential sink.

4.1. Event-Driven Paradigms

Over the past decade, much of the software developed for Financial Crime Compliance has relied upon batch processing techniques and the associated pluggable architecture patterns. Recently, event-driven paradigms enabled the processing of data in near-real time. Data is acted upon as it arrives or generated and business functions are invoked to consume the data in a response-initiated manner. Requests for detection functions can be posed at any point, and the underlying architecture manages the database read/write concurrency and system resources.

Event-driven architectures rely on the production, detection, consumption of and reaction to events. Any change in the state, perceived or caused, can induce an action, query an event detection function and generate a response. Streaming architectures further focus on the continuous flow of events as a single entity using distributed stream processing to create the illusion that every record is being processed one by one by each member of a cluster. Data flows in a push manner and operations upon it consume the data before generating the output, similar to an operator applied to a mathematical function. Throughput and resource utilization needs of a streaming application can be reduced using various consistency, stateful operation and grouping techniques.

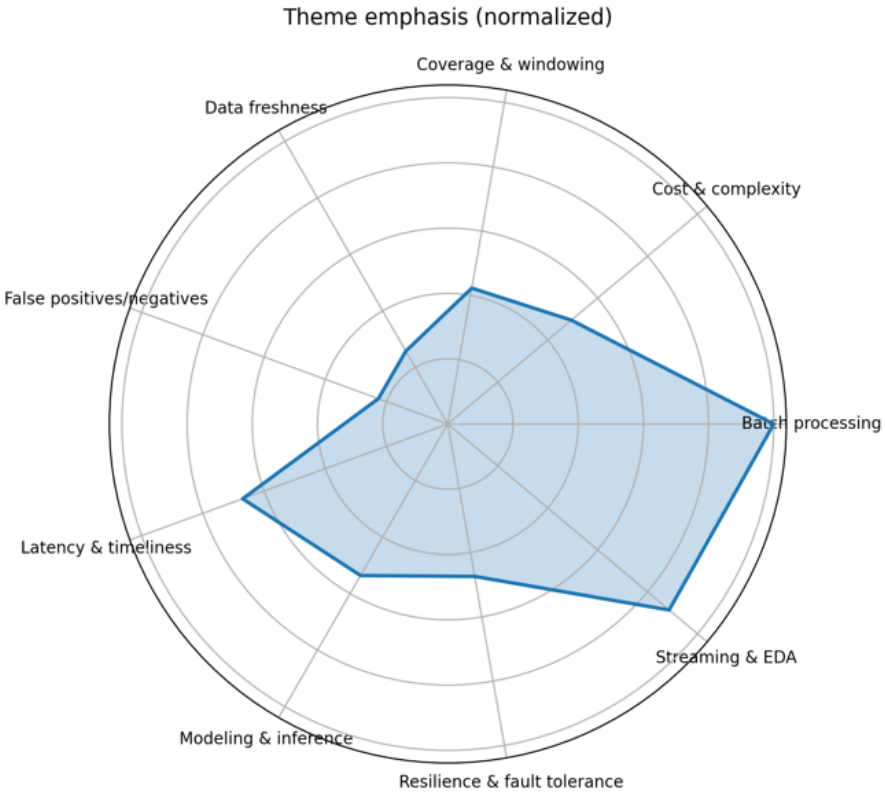

Figure 3. Normalized Theme Emphasis Profile Derived from Keyword Frequency Analysis

Equation 2) Data freshness / staleness equation
Define:

- t = current time
- t_{last} = timestamp of the latest data point actually available to the detector

Then **staleness** is:

$$\boxed{S(t) = t - t_{\text{last}}}$$

Batch implication: during most of the window, t_{last} may lag behind t by hours (staleness can approach W).
Streaming implication: t_{last} is close to real time, so $S(t)$ is small (bounded by pipeline latency L_{stream}).
A simple bound:

$$S_{\text{stream}}(t) \lesssim L_{\text{stream}}$$

4.2. Latency, Throughput, and Consistency Trade-offs

Any detection mechanism has finite latency; decisions based on stale data run the risk of being uninformed. For event-driven, stateful architectures, the core trade-offs concern latency, throughput, and consistency. Events are processed in a query-by-query fashion (be it in a sink such as a detection layer or in state that is eventually persisted in a store). However, operational resilience is paramount; events can be lost or duplicated, and downstream systems are often imperfect. The resultant scenarios, and the choices made by system designers; for the internet-scale streaming platforms, for example, hot paths are processed under high throughput but without guarantees on event ordering. The adherence to eventual consistency offers a natural way of anticipating faults: the system is designed such that whenever an area of responsibility becomes unavailable, dropping the stream unconcerned does not create service issues; any future events will inevitably re-create the state after a certain point.

All latencies add up, yet a distributed architecture does not equal high latency; it defines mechanisms for estimating how long an operation will take. Such non-strict guarantees impose certain constraints on the architecture. For pile-in storage, stragglers are major issues: the completion of the whole operation waits for the end of the slowest sub-operation. Appropriate windowing strategies alleviate these issues to a great extent, but certain assignments will always be difficult. Latencies measured from different points in the pipeline help quantify the impact of such areas, and backpressure argues how to alleviate operational issues in the face of failures.

5. Comparative Evaluation: Batch vs Streaming

The design and implementation of architectures for financial crime compliance systems require a balance between detection performance and operational resilience. Performance can be evaluated from multiple angles: the ability to detect crimes in a timely manner; detection coverage; the impact of false positives and false negatives; and the capacity of the detection processes to identify and alert authorities to events that would not have been detected otherwise. These aspects are often evaluated using a real, high-quality dataset in combination with augmentation techniques or by directly analysing the flow of suspicious activity reports (SARs).

Seven aspects of operational resilience and scalability are considered: the ability to absorb loss of systems or other components without significant performance degradation; fault diagnosis and recovery; the ability to change, maintain, scale and tune the components with minimal direct or indirect impact on the compliance operation; operation on commercial, open-source or in-house systems; maintenance of sufficient performance and operational status at least cost; the cost of deploying or upgrading a system or execution environment; and the effect on people and processes involved in the architecture.

5.1. Detection Effectiveness

Effectiveness depends on timing and coverage of significance decision rules. Timeliness relates to detection latency—detection of anomalies following emerging trends or unexpected events—and minimisation of false negatives, both conditions tending to favour adoption of streaming detection. Conversely, detection coverage matches detection window—window duration, size of historical context rolled into detection decision—and minimisation of false positives, particularly critical in the transaction scenario with entailing confirmation processes, thus tending to favour batch approaches.

Batch detection architectures are well suited for scenarios involving advanced pattern-matching venturing beyond simple decision threshold rules relying only on the most recent history of events, i.e. taking an excessive view of the most current behaviour of actors subject to monitoring; the risk of flagging alerts indicative of down- or up-trending patterns being allerted and confirmed by other subsequent detection rules of different types. Latency of the detection (time when rule is evaluated) is critical as it is also the time when the alert surface is built. The delays associated with their build-out create a backlog in the monitoring process, and all the considerations regarding the durability and relevance before each alert is examined permit to qualify the detection capture as being lazy.

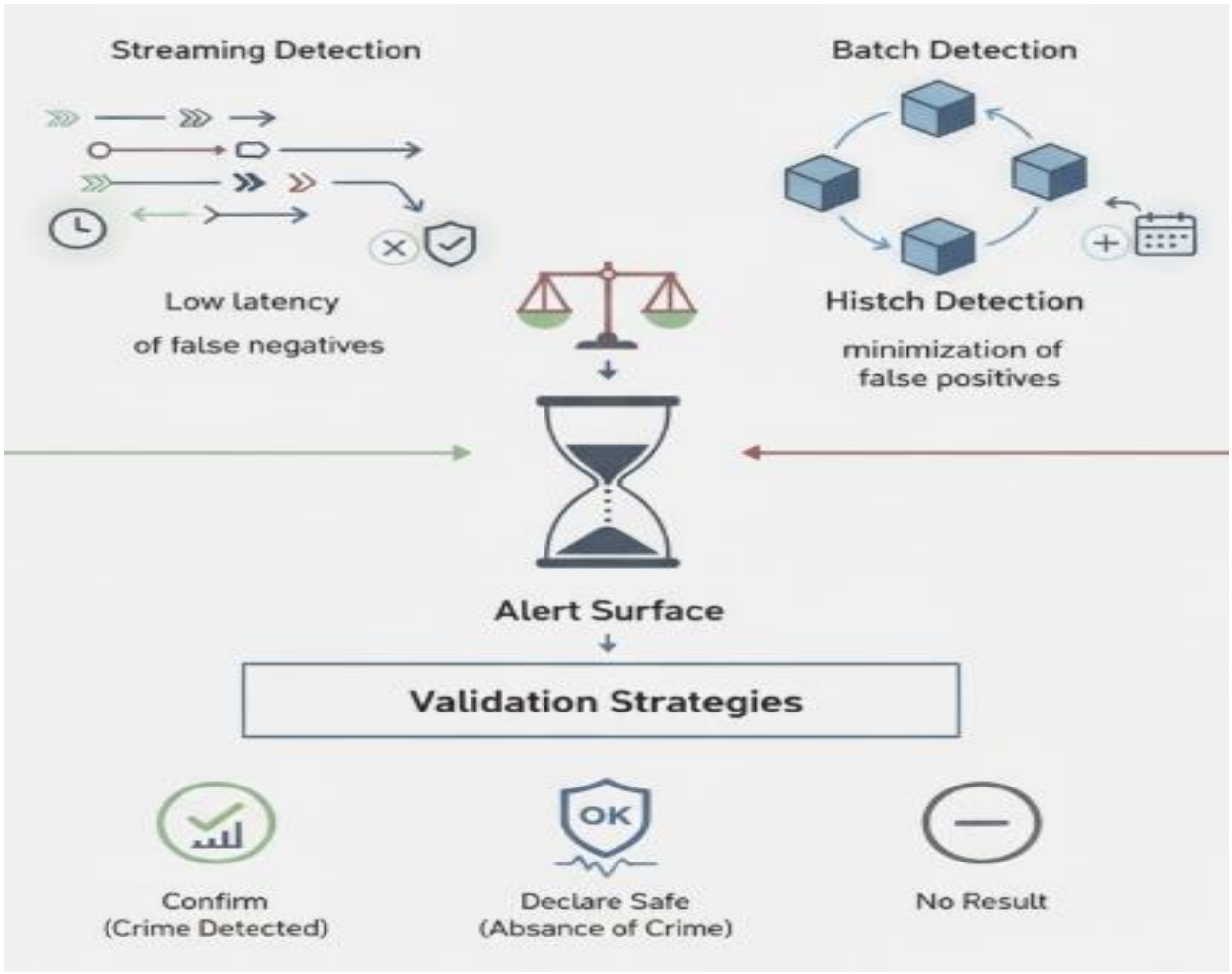

Fig 4: Temporal Trade-offs in Financial Surveillance: Balancing Detection Latency and Pattern Coverage in Batch versus Streaming Architectures

Validation strategies offer a definitive mirror of detection effectiveness and are linked to the type of monitoring process. On executing of alert confirmation, three types of investigations can emerge: confirm—action taken confirms that a crime is detected; declare safe—action taken indicates the absence of crime; no result—no action or manual result on the alert. An obvious efficiency metric is then the ratio of those alerts confirmed, which for a good and healthy detection benchtop should turn out as small.

5.2. Operational Resilience and Scalability

The preceding analysis compared the effectiveness of batch and streaming architectures in detecting suspicious events. Beyond this direct evaluation, additional differences influence the quality and safety of operations. These include the platforms' resilience to failures, the complexities associated with their deployment and ongoing maintenance, the ability to accommodate increasing workloads, and their relative cost.

Batch systems can tolerate delays and interruptions. However, an accumulation of batch processing is a risk, especially if it grows unmonitored. Under normal circumstances, data flow should maintain consistency across the covering time windows. Exposure to risk increases if the batch window is larger than the standard reporting frequency. For example, processing markets and jurisdictions once every year is unlikely to be sufficiently timely. Most detection requirements are not real time. Nonetheless, when backing up approaches become necessary, it is advisable to decouple addition paths to prevent pile-ups. In contrast to the eventual consistency of streaming architectures, batch platforms offer strong consistency—persistence guarantees that all evidence will eventually be checked.

Deployment complexity and cost assess the coupling of the architecture with the operating model. Batch platforms typically integrate with periodic data-flows systems and operate within operational hours. Most batch solutions operate in the same way as other processes: they should be executed during the night and report failure in the morning. Detection reprocessing takes place during off-peak hours, following the completion of the batch. The continuous nature of event-driven approaches requires either custom-built data-flows or delayed execution of detection pipelines when additional volume floods the system. Data-addition components frequently integrate into business-applications. Cost concerns also relate to the underlying event bus: early adoption of an event-stream-processing solution may require investment at the application layer even without a proper streaming detection layer.

6. Implementation Considerations

Data ingestion and normalization represent the foundational steps of stream processing platforms. Sources span internal, external, and partner systems, covering transnational payments, transaction monitoring alerts, behavioral profiles, transaction screening hits, economic sanctions lists, and customer due diligence findings. These sources often expose change-data-capture APIs with common schema formats. The ingestion layer detects new or modified records, captures metadata like timestamps and data types, and validates and normalizes records against a reference schema for downstream processing. Subsequent streams must maintain traceability to their origins, ensuring accurate data lineage and source attribution. Windowed deduplication removes the natural duplicates generated every time a payment is sent or received, and automated schema evolution tracks model updates. A similar approach can deduplicate web and mobile traffic.

Feature engineering occurs in real time – ideally in a windowed manner – at the heart of each detection pipeline. Detection features can be expressed in terms of feature pipelines, which prepare features from the individual sources, assemble features from multiple sources, and/or maintain state about the particular entity. Windowed aggregates of transactional activity or behavior, expressed as time-varying features, are often pooled into a (relational) stateful key-value store. When supporting a supervised detection model, the key-value store can retain the latest mause of the trained model for improving detection precision, refreshing the models on a fixed external schedule or upon detecting data distribution shifts.

Equation 3) Coverage as a windowing equation (timeliness vs coverage)

- streaming tends to improve **timeliness** and reduce **false negatives**
- batch tends to preserve **coverage** (historical context) and reduce **false positives**

Let:

- H = historical context length needed by a detection rule/model (e.g., last 30 days behavior)
- W = batch window (e.g., 1 day)
- streaming uses a rolling window of length H maintained in state

3.1 "Coverage completeness" as a function of retained history

A detection rule requiring H days of history is fully supported if the system can provide all required history at decision time.

Define:

$$C = \frac{\text{history available at detection time}}{H}$$

So:

- If full history is present: $C = 1$
- If only $h < H$ is present: $C = \frac{h}{H}$

6.1. Data Ingestion and Normalization

For streaming applications, a dedicated topic is the ingesting and normalization of data from different sources. Financial Crime Compliance platforms implement a wide variety of third-party applications which provide data about transactions (e.g., payment processors, banks), customers (ID validation, fraud repositories), and merchants (Blacklists).Each provider supplies its data in an ad-hoc manner, meaning that its structure can vary with a non-expected frequency. This creates an invariant data source-management problem that must be solved. Additionally, even these types of platforms must address the issue of data schema evolution, as the structure of the in-and-outflowing data might change due to the join-and-flow system with external services.

The functioning of a machine-learning model based on a set of features derived from the evolution of entities must be defined. Entities of interest for a Financial Crime Compliance system are users, accounts, wallets, or anything that can flow money. A classic machine-learning engine based on features must cope with issues of data relevance (e.g., by dropping old, outdated data) but at the same time must prevent data redundancy due to overlapping windows. It is also mandatory to store statistical data about each feature, since replacement or model-training processes depend on this information. Finally, a streaming-based Financial Crime-Compliance machine model must consider the ability to apply the algorithm with new or updated models provided by a model-refresh system.

6.2. Feature Engineering in Real Time

Operators are hungry for features. With every new data pipeline, they expect to deliver the fresh ingredients that feed ML models. Not just the mean aggregates of recent minutes, or the summations over last hours, but a continuous supply of shape, frequency, count, and outlier measures that make features the same premium product for ML as it is in cuisine. Complex, high-dimensional feature spaces drawn from real-time streams can deliver the competitive edge needed to slice through crowded business landscapes. But behind every recipe for success are industrial-strength feature engineering pipelines, built to operate under continuous load, maintain high data quality, stay aligned with modeling refreshes, and deliver a busting cupboard of feature shapes for any real-time prediction.

Feeding ML-model inference graphs is crucial for reducing prediction latency. Yet creating features from the same continuous streams that deliver inference can require a very different approach. Ingesting duplicate records is seldom an issue for batch processes; they are designed to identify and deduplicate records in high-throughput pipelines. During such windows, ML-model performance can be affected by stale features drawn from even earlier batches. Detecting fraud and financial crime before the money moves requires a highly responsive design with choice ingredients available early. Session-based features are a common trick to deliver such speed. Refreshing them before every test is hard, expensive, and prone to Error. Integrating model feature builds into the Isops backend isn't just a safe way to keep up with evolution; it lets the Isops pipelines inject a regular-growing supply of fresh shape, frequency, and count ingredients for models running against scant months of data.

7. Conclusion

The study provides an objective, evidence-based examination of batch and streaming processing in financial crime compliance platforms. Research questions address detection effectiveness versus resilience, scalability, and complexity. The analysis confirms that while batch processing remains appropriate for many use cases, an event-driven architecture is increasingly advantageous for near-real-time detection. Benefits include more timely detection, enhanced operational resilience, improved support for backtesting and investigation, reduced false-negative rates, and containment of increasing detection burdens. However, the implementation effort and accompanying costs may outweigh the advantages in some scenarios. Processing requirements and underlying data risks ultimately determine batch-window width, and the full benefits materialize only when streams are widely embraced.

Batch processing has dominated compliance platforms for more than a decade. Originating from bank-wide analytics systems, platforms aggregate information into a crudely defined feature set. The reasoning follows a

familiar pattern: once-a-day monolithic pipelines are constructed, and features are extracted. Providers favor these solutions because they neatly sidestep potential issues that arise when banks analyze event-driven data with an analytic framework originally designed for batch processing. The principal concerns are twofold: the consistency of the event data and the lack of total coverage of the detected events (where total coverage means there is no time between detection and possible violation). When the bulk of the detection infrastructure is stable and experience has been gained, the standard resilience checks applied to any streaming architecture become second nature. Nevertheless, the detection platforms require paths to production that can be industry-agnostic and allow standardized methods for testing implemented and proposed workflow changes, be they batch or streaming, for resilience.

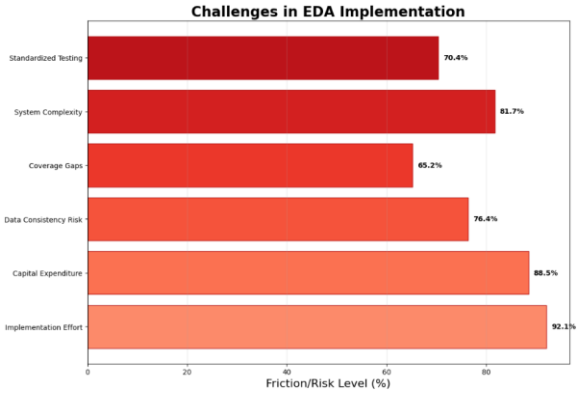

Fig 5: Challenges in EDA Implementation

7.1. Final Thoughts and Future Directions

In summary, this work examined the compelling trade-offs between (near) real-time detection and flooding with alerts that may bury genuinely important incidents in the flood. A careful assessment of the different components necessary for compliance with financial crime statutes identified the batch paradigm as suitable for several places in such a pipeline. In other places, where the historical precedent of batch execution has hitherto dictated the stream-based traffic, the switch from batch to, primarily, event-driven architectures has taken place in step with the general adoption of more streaming paradigms. Within these, trade-offs in consistency and latency reflect in the quality of the models trained and employed. Done in conjunction with event-driven maintenance of feature engineering, such improvements promise to evolve the industry to ever-more-real-time detection. Note that throughout this analysis, floods of alerts were described in connection with alerts narrowing downstream; such an assertion depends on other components of the practical pipeline.

With that caveat in mind, the challenge of managing those features points to another area for practical work. Feature engineering is a standard area of machine-learning attention, yet its execution in real time is relatively less discussed in the literature. Here, the attention in production worries not just about nontimeflowing features but the general question of maintaining latent features that come less naturally. In a crime-compliance context, the need for temporal evolution of all features is perhaps even enhanced. In industry, features are transient windows on relationships. Such windows evolve: a few edged-out trades are hardly interesting, a cold-streak player ought to provoke action, and surveillance-monitoring cameras flipping off for hours should uncross all touchpoints. Proper consideration of all aspects of feature engineering thus presents fertile avenues for further research, both practical and theoretical.

8. References

1. Ackermann, F., Howick, S., Quigley, J., Walls, L., & Houghton, T. (2014). Systemic risk elicitation: Using causal maps to engage stakeholders and build a comprehensive view of risks. European Journal of Operational Research, 238(1), 290–299.

2. Aggarwal, C. C. (2015). Data mining: The textbook. Springer.
3. Aggarwal, C. C., & Wang, H. (2010). Managing and mining graph data. Springer.
4. Allen, F., Gu, X., & Kowalewski, O. (2018). Financial structure, economic growth and development. Journal of Banking & Finance, 89, 1–3.
5. Albanese, D., Lo Duca, M., & Visintainer, R. (2019). Data quality and data governance in financial crime analytics. Journal of Financial Regulation and Compliance, 27(4), 475–492.
6. Baeza-Yates, R., & Ribeiro-Neto, B. (2011). Modern information retrieval (2nd ed.). Addison-Wesley.
7. Basel Committee on Banking Supervision. (2017). Sound management of risks related to money laundering and financing of terrorism. Bank for International Settlements.
8. Bellman, R. (1957). Dynamic programming. Princeton University Press.
9. Bennett, K. P., & Campbell, C. (2000). Support vector machines: Hype or hallelujah? SIGKDD Explorations, 2(2), 1–13.
10. Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.
11. Bolton, R. J., & Hand, D. J. (2002). Statistical fraud detection: A review. Statistical Science, 17(3), 235–255.
12. Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32.
13. Buckley, K., & Webster, S. (2016). Sanctions compliance in international banking. Journal of Financial Compliance, 1(1), 23–37.
14. Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), 1–58.
15. Chen, H., Chiang, R. H. L., & Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact. MIS Quarterly, 36(4), 1165–1188.
16. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.
17. Meda, R. (2019). Machine Learning Models for Quality Prediction and Compliance in Paint Manufacturing Operations. International Journal of Engineering and Computer Science, 8(12), 24993–24911. https://doi.org/10.18535/ijecs.v8i12.4445.
18. Dal Pozzolo, A., Bontempi, G., Snoeck, M., & Snoeck, M. (2015). Adversarial drift detection. IEEE Computational Intelligence Magazine, 10(4), 36–45.
19. Davenport, T. H., & Harris, J. G. (2007). Competing on analytics. Harvard Business School Press.
20. Dwork, C., & Roth, A. (2014). The algorithmic foundations of differential privacy. Now Publishers.
21. Ellis, C., & Dykes, J. (2018). Visual analytics for decision support. Information Visualization, 17(3), 225–242.
22. Evans, D., & Over, M. (2019). Sanctions compliance and operational risk. Journal of Banking Regulation, 20(3), 243–257.
23. Fawcett, T., & Provost, F. (1997). Adaptive fraud detection. Data Mining and Knowledge Discovery, 1(3), 291–316.
24. Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in databases. AI Magazine, 17(3), 37–54.
25. García-Molina, H., Ullman, J. D., & Widom, J. (2008). Database systems: The complete book (2nd ed.). Pearson.
26. Gill, M., & Spriggs, A. (2018). Sanctions screening technology in global banking. Journal of Financial Crime, 25(4), 1031–1045.
27. Goldreich, O. (2009). Foundations of cryptography: Volume 2, basic applications. Cambridge University Press.
28. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.