International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Credit Risk Modeling with Real-Time Streaming Features: PD/LGD Pipeline Design

Jeevan Krishna Paruchuri

Independent Researcher
paruchuri.g167@gmail.com

Abstract:
Credit risk modeling for embedded lending in fintech neobanks requires real-time decision-making at transaction boundaries, where millisecond-level timing can determine whether a micro-credit request is approved or declined. The key metrics that drive credit decisions Probability of Default (PD) and Loss Given Default (LGD) depend on the quality and timeliness of features fed into them, and traditional monthly batch cycles create an unacceptable freshness gap. This paper presents a case study of a real-time streaming feature pipeline for PD/LGD models at a fintech neobank with a portfolio of more than 2.5 million microaccounts and a transactional volume of approximately 750,000 financial events per second on average, with 2 million events at peak during payment cycle waves. We describe the end-to-end architecture (Kafka ingestion from distributed payment APIs, Spark Structured Streaming feature computation across 1,100+ microservice event streams, Databricks Delta Lake persistence with audit logging, Feast feature store backed by Bigtable for ultra-low-latency retrieval, Seldon Core model serving on GKE with real-time prediction), the regulatory governance framework required for UDAP compliance and consumer protection, the feature drift detection and smoothing methods we use to manage the extreme volatility of microaccount spending patterns, and the operational realities of running this kind of pipeline at 99.8% measured availability against a 99.95% target. We compare batch and streaming approaches and argue for a hybrid: batch base features for slow-changing customer identity combined with streaming delta features for transaction-level behavior. We compare credit risk and fraud detection architectures in the fintech context, which share core streaming infrastructure but differ significantly in latency requirements and user experience impact. We close with a decision framework for fintech practitioners considering whether real-time PD/LGD features are worth the operational investment in a competitive lending market.

Keywords: Credit risk, Probability of Default (PD), Loss Given Default (LGD), Stress testing, Feature engineering, Real-time streaming, Spark, Feature store.

1. INTRODUCTION

Credit risk is the canonical use case for quantitative modeling in banking, and the outputs of credit risk models are not academic. Under Basel III, banks must compute Probability of Default and Loss Given Default for their credit portfolios, and the regulatory capital they are required to hold is proportional to the risk those models report. Inaccurate or stale models translate directly into either excess capital that could have been deployed productively or insufficient capital against actual risk. In either direction, the cost is large. Accurate, timely PD and LGD models therefore depend on high-quality features customer financial state, transactional behavior, macroeconomic context and on the discipline to keep those features refreshed.

The dominant pattern in the industry is batch feature computation. Features are aggregated nightly or monthly, models are scored against the batch, and the resulting risk grades flow into capital reporting and decision systems. This approach has the virtues of simplicity and reproducibility: the same input

data produces the same features, the same features produce the same scores, and audit trails are straightforward because everything is anchored to a known batch run. The cost of the approach is freshness. A customer whose spending patterns deteriorate sharply in the second week of a month does not show up as a higher-risk customer until the month-end batch runs and the new features propagate through the model.

The promise of real-time streaming features is to close that gap. If features are recomputed continuously as new transactional events arrive, the model can react within minutes rather than weeks. The cost of the promise is complexity: streaming infrastructure is harder to operate than batch infrastructure, state management adds new failure modes, feature stability becomes a daily concern rather than a quarterly one, and the regulatory question of how to audit a continuously changing feature becomes a real engineering problem.

This paper presents the architecture, governance, and operational experience of building such a system at a financial services organization with more than one million customers and peak transactional volume of 500,000 transactions per second. We address three research questions.

RQ1. What features matter for PD and LGD modeling, and how do real-time streaming versions of those features improve over batch versions? What are the stability and reliability trade-offs involved in moving them from batch to streaming?

RQ2. How should a real-time feature pipeline for credit risk be architected? What are the latency, scalability, and governance constraints, and how does the architecture handle out-of-order events, late arrivals, and the non-stationarity of customer behavior?

RQ3. What are the regulatory and operational implications of real-time features in a credit risk context? How do you ensure model explainability, audit trails, and stress testing in a streaming environment, where the features themselves are moving targets?

The thesis we defend is that real-time streaming features for PD/LGD are technically feasible and operationally valuable they enable faster risk detection, better early warning of deterioration, and more responsive credit decisioning but they introduce complexity (state management, event ordering, feature drift) and regulatory challenges (auditability, reproducibility) that must be addressed explicitly. A hybrid architecture that combines batch-computed base features with streaming-computed delta features offers the best balance between freshness and stability. Organizations that invest in this infrastructure gain a competitive advantage in decision speed, but the investment is meaningful and only justified above a certain scale and risk tolerance. We provide a decision framework in Section 8 for organizations weighing the trade-off.

The paper proceeds as follows. Section 2 reviews credit risk fundamentals and feature engineering. Section 3 describes the end-to-end pipeline architecture. Section 4 addresses regulatory governance and model validation requirements. Section 5 covers feature drift detection and stabilization. Section 6 reports performance and scalability measurements. Section 7 compares the credit risk architecture to a sister fraud detection pipeline. Section 8 provides the decision framework. Sections 9 and 10 cover future directions and conclusions.

2. CREDIT RISK FUNDAMENTALS AND FEATURE ENGINEERING

2.1 The PD/LGD Framework

Probability of Default (PD) is the likelihood that a borrower defaults within a defined horizon, conventionally twelve months. Loss Given Default (LGD) is the economic loss that the bank expects to incur if a default does occur, expressed as a fraction of exposure; an unsecured personal loan has a higher LGD than a mortgage backed by collateral. Combined with Exposure at Default (EAD), the three metrics produce Expected Loss: EL = PD × LGD × EAD. Under Basel III, banks must compute these metrics for their credit portfolios, and the resulting Expected Loss aggregates feed directly into the regulatory capital requirement.

The standard modeling approach uses logistic regression on historical default data, with features drawn from customer attributes and transactional behavior. More sophisticated banks use gradient boosting models or neural networks for parts of the pipeline, particularly where non-linear interactions among features are important. Regardless of the model family, the input is always the same: a vector of features summarizing what is known about the customer at the moment of scoring.

2.2 Feature Categories for PD/LGD

The features that matter for credit risk fall into several categories. Customer demographics (age, tenure with the bank, declared income, length of credit history) capture stable attributes. Account characteristics (account age, credit limit, current utilization, delinquency flags) capture the structural risk of the relationship. Transactional behavior features frequency of transactions per month, velocity measured as the trend in transaction amounts, consistency measured as the variance of transaction amounts capture how the customer is actually using the account. Macroeconomic variables (unemployment rate, interest rate, sector-specific performance for sector-concentrated lenders) capture the environment the customer is operating in. Behavioral risk signals multiple credit inquiries in a short period (often a sign of credit shopping that precedes financial stress), payment timeliness, cross-product usage with the same bank provide leading indicators that pure account data does not. Time-series features such as 6-month rolling spending averages, volatility, and trend complete the picture.

Different categories have different update cadences. Demographics change rarely. Account characteristics change monthly or with explicit account events. Transactional behavior changes continuously. Macroeconomic variables update on weekly to monthly cycles depending on the indicator. Designing a pipeline that respects these different cadences without over-engineering any single one is the central architectural problem.

2.3 Batch versus Real-Time Feature Computation

The traditional batch approach computes all features at month-end. The benefits are simplicity (a single nightly job, no streaming infrastructure), stability (the same historical data produces the same features, every time), and excellent reproducibility for audit purposes. The cost is latency: up to thirty days between the moment a customer's behavior begins to deteriorate and the moment the feature reflects the deterioration. In a portfolio where a meaningful number of defaults occur within that window, the gap is operationally consequential.

The real-time streaming approach computes features continuously as new events arrive. The latency on the feature side falls from days to minutes. The cost is operational complexity: streaming infrastructure must run continuously, state must be managed across nodes, recovery from failures must be reasoned about explicitly, and feature stability becomes a problem because features that were once monthly aggregates are now updating multiple times per hour.

Our approach is hybrid. Base features demographics, account characteristics, long-window aggregates are computed in batch on a monthly cadence, because they do not change quickly enough to justify streaming. Delta features transactional velocity, payment timeliness, short-window behavioral indicators are computed continuously by the streaming pipeline. The model consumes a vector that combines the batch base with the streaming deltas at inference time. The rationale is that the volatility of the streaming components is balanced by the stability of the batch components, and the system as a whole is stabler than a pure-streaming alternative would be while still being more responsive than a pure-batch alternative.

3. ARCHITECTURE AND IMPLEMENTATION

3.1 End-to-End Pipeline Architecture

The pipeline draws from four data sources. The core microfinance system, a PostgreSQL database, provides microaccount master data, customer identity data, and transaction history; we ingest changes via Striim real-time CDC for continuous data capture. The distributed payment processing layer provides transaction events as Kafka topics with microsecond-level tagging. Alternative data sources telemetry from the fintech application, connectivity patterns, geographic signals arrive as structured events from the mobile application layer. Macroeconomic indicators Federal Reserve rates, inflation adjustments arrive as daily batch updates.

Ingestion is mediated by Kafka. The transactions topic carries approximately 200,000 transactions per second on average and approximately 500,000 transactions per second at peak. An account changes topic carries approximately 10,000 events per second. A macroeconomic updates topic carries approximately one event per minute.

Feature computation runs on Spark Structured Streaming. The streaming job applies stateful aggregations rolling 7-day and 30-day windows partitioned by microaccount over the transaction stream, joins against slow-changing dimensions from the customer identity master, and writes the resulting feature vectors out to two destinations: Databricks Delta Lake for persistence with audit logging, and Feast feature store backed by Bigtable for ultra-low-latency online serving at scale. Checkpointing to Databricks-managed storage every five minutes provides fault tolerance and consistency guarantees.

The output Delta table, feature_store.customer_features, is updated hourly and carries the microaccount identifier, the computation timestamp, the feature value, and a version field that supports reproducibility and compliance audits. The Feast feature store backed by Bigtable provides online feature retrieval at approximately 15-30ms latency at high throughput, and a fallback path reads directly from Delta Lake for cases where the cache misses or has been evicted, ensuring graceful degradation during Bigtable maintenance windows.

Model inference is handled by Seldon Core model serving on Google Kubernetes Engine (GKE), which loads the trained PD/LGD model and serves predictions with A/B testing and automatic failover capabilities. A typical inference request retrieves features from Feast/Bigtable, runs them through the model via a gRPC call to the Seldon model pod, and returns a risk score in under 50 milliseconds end to end, meeting the real-time decisioning latency requirements of the lending product.

3.2 Spark Streaming Implementation

Spark Structured Streaming processes the transaction stream in micro-batches with a trigger interval of approximately five to ten seconds. The core stateful aggregation looks like the following, expressed in Spark SQL syntax:

```
kafkaTransactionStream

.select(col("customer_id"), col("amount"), col("timestamp"))

.groupBy(window(col("timestamp"), "7 days", "1 hour"), col("customer_id"))

.agg(

sum(col("amount")).alias("txn_7d_sum"),

count(col("amount")).alias("txn_7d_count"),

stddev(col("amount")).alias("txn_7d_stddev")

)
```

State is maintained in memory by the Spark executors with periodic checkpoints to Databricks Managed Tables for recovery and compliance. The checkpoint includes the rolling window state, the Kafka offset committed, and the metadata required for exactly-once semantics on resume. State has a TTL of 90 days; older state is evicted to manage memory pressure and reduce audit log size.

Watermarking is the central concession to the messy reality of event ordering. Transactions can arrive out of order payment system processing delays, mobile app sync delays and the streaming engine needs a policy for how late an event can be before it is dropped. We watermark at five minutes, meaning that events arriving more than five minutes after the watermark are excluded from the corresponding window. The trade-off is straightforward: a longer watermark accepts more late events at the cost of delaying feature finalization, and a shorter watermark finalizes features faster at the cost of potentially missing legitimate late arrivals. Five minutes was the value at which both error and delay were acceptable in our environment, and we re-evaluate it whenever the upstream payment system changes.

Feature enrichment with slowly-changing dimensions (customer master) uses broadcast joins. The customer master is small enough to fit in executor memory, so broadcasting it avoids the cost of a shuffle join on every micro-batch. Macroeconomic data, which updates daily, is joined at hourly frequency rather than per micro-batch.

3.3 State Management and Fault Tolerance

The challenge with Spark Structured Streaming at this scale is that state the rolling windows and partial aggregations lives in memory on the executors. Loss of an executor without recovery would mean recomputing features from scratch, an operation that can take roughly 30 minutes against the historical Kafka backlog, during which the pipeline produces no fresh features.

The solution is checkpointing to Databricks Managed Tables every five minutes via Delta Lake's native checkpoint support. The checkpoint contains the state, the offsets of the most recently processed Kafka messages, and the metadata Spark needs to resume cleanly with transaction guarantees. On restart, Spark loads the checkpoint and resumes from the last committed offset, providing exactly-once semantics from the perspective of feature output with full compliance audit trails. Operationally, the checkpoint space grows by approximately 15 gigabytes per day (larger than cloud storage checkpoints due to Databricks' additional consistency metadata), and an automated retention policy removes checkpoints older than 30 days to manage compute costs.

The most consequential gotcha we have hit with checkpoint recovery is that checkpoints are tied to the exact Spark code that produced them. Changing the schema or the structure of the streaming query in a way that affects the state representation breaks checkpoint recovery, forcing a cold start. The mitigation is careful versioning of streaming code, with explicit migration plans for any change that affects state.

3.4 Online Feature Serving

Feast with Bigtable backend is the online feature serving layer. Each microaccount's feature vector is keyed bycustomer_id and stored as a JSON document containing all relevant features (txn_7d_sum, txn_7d_count, txn_7d_stddev, and so on). The TTL on each entry in Feast is one hour, after which the cached value is considered stale and refreshed by the next streaming write. Spark writes updated feature vectors to Feast hourly as a batch operation that automatically synchronizes to Bigtable, which keeps Bigtable write throughput bounded and predictable while serving reads at 15-30ms latency.

The fallback path matters. If Feast/Bigtable misses because the cache was evicted, the microaccount is new, or a Bigtable instance is recovering the inference path falls back to reading from Delta Lake directly. Delta Lake reads are slower than Feast/Bigtable (closer to 100-150 milliseconds versus 15-30 milliseconds for Feast), but they preserve the ability to score any microaccount rather than forcing a hard failure on the lending decision.

We monitor the Feast/Bigtable hit rate continuously. The target is above 95%. A sustained drop below that threshold is an early indicator of cache eviction pressure, Bigtable rebalancing, or upstream pipeline issues, and it triggers an investigation before it becomes a service-level breach that would impact lending approval latency.

4. REGULATORY GOVERNANCE AND MODEL VALIDATION

4.1 Regulatory Requirements for Credit Risk Models

The CFTC and OCC regulatory frameworks for credit risk models impose four categories of requirement that bear directly on the pipeline architecture. Model documentation requires that PD and LGD models be fully described, with every feature justified in terms of its predictive value and its business meaning. Back-testing requires regular comparison of model predictions against realized outcomes, with explicit thresholds beyond which the model must be recalibrated. Stress testing requires the model to be evaluated under adverse macroeconomic scenarios recessions, sector shocks, sustained unemployment increases. Governance requires named model owners, an independent validation function, and explicit risk committee approval for major model changes.

Beneath these four categories are two cross-cutting requirements that shape the engineering work. First, all credit decisions must be traceable to the specific features and the specific model version that produced them. Second, for high-risk decisions particularly declines the institution must be able to explain to the customer which features drove the decision.

4.2 Feature Governance and Auditability

The challenge that streaming features create for governance is that features change continuously. If a customer is declined for credit at 14:00 on a particular day, the features that drove that decision were the features that were current at 14:00, not the features that exist now. Reproducing the decision for audit requires being able to retrieve the exact feature values that were in force at the moment of the decision, weeks or months after the fact.

The solution is a feature audit log. Every feature value, every version of the feature definition, and every computation timestamp is written to an append-only Delta Lake table at the moment of computation. A representative schema:

TABLE: feature_store.customer_features_audit

customer_id | feature_name | feature_version | feature_value | computation_time | model_version

12345 | txn_7d_sum | v1 | 50000 | 2023-04-11 14:00 | PD_model_v12

Given a customer identifier and a timestamp, the audit log allows us to retrieve the exact features that would have been served at that moment. The cost is storage volume. With approximately one million customers, 24-hour update cadences, and a retention horizon of 365 days, the audit log accumulates on the order of 8.7 billion records. Delta Lake's compression and aggressive compaction reduce this to approximately 100 gigabytes per year of physical storage, which is manageable.

4.3 Model Validation and Back-Testing

Back-testing runs on a monthly cadence. The process compares actual defaults observed in the portfolio against the predictions the model made one month earlier, computing calibration metrics including the Gini coefficient and the Kolmogorov-Smirnov statistic. If divergence exceeds an internally defined threshold (we use a 5% deviation as the trigger), a model refit is initiated.

Streaming features complicate back-testing in a specific way. Streaming features are more volatile than their batch equivalents small changes in transaction patterns produce visible swings in the feature value and the additional volatility translates into model output volatility that can look like degradation when it is actually noise. The mitigation is to smooth features at the serving layer using an Exponentially Weighted Moving Average (EWMA), which damps short-term fluctuations without losing the underlying signal. A typical configuration:

EWMA_factor = 0.1

smooth_feature = EWMA_factor current_feature + (1 - EWMA_factor) previous_feature

The introduction of streaming features pushed our refit cadence from quarterly (the historical norm) to bi-monthly (to keep up with the higher drift rate). The increased refit cadence is one of the operational costs of streaming features that organizations evaluating the architecture should price in.

4.4 Stress Testing

Stress testing is a regulatory requirement, not an optional analytical exercise. Banks must evaluate their credit risk models under defined adverse scenarios recession, unemployment spike of five percentage points, interest rate shock of two percentage points and report the resulting risk to a risk committee.

The traditional approach to stress testing is to re-run the model offline with hypothetical feature values substituted in for the macroeconomic inputs. In a streaming context, the same conceptual operation happens nightly: a stress scenario is defined, features are modified accordingly (customer spending decreases under the recession assumption, delinquency rates increase, and so on), and the model is run against the modified features. The result is a stress-adjusted risk score for every customer in the portfolio, which feeds the report to the risk committee.

The operational cost of stress testing at our scale is approximately two hours per nightly run, computing stress scenarios across one million customers. This was an embarrassment for the first several months a meaningful chunk of the nightly maintenance window and we have since invested in GPU-accelerated inference to bring the runtime down. The lesson worth recording is that stress testing scales linearly with the portfolio, so capacity planning needs to account for it explicitly rather than treating it as a marginal load.

5. FEATURE DRIFT AND STABILITY

5.1 Feature Drift Detection

Real-time features change over time for many reasons that are not signals of credit deterioration. Customers change their spending patterns. Holidays and summer vacations produce visible seasonality. Macroeconomic shocks shift the distribution of behavior across the entire portfolio. Data quality issues Kafka consumer lag, sensor failures, upstream outages produce artifactual drift that is not real customer behavior at all. The challenge for the team operating the pipeline is to distinguish real drift (which should trigger investigation and possibly model refit) from noise (which should be filtered out or smoothed away).

The detection approach we use is based on distributional comparison. For each feature, we maintain the distribution of values observed over a 90-day rolling historical window. A Kolmogorov-Smirnov test compares the current feature distribution against the historical distribution, and the KS statistic above a threshold (we use 0.15) triggers a drift alert. As a representative example: the 7-day transaction amount sum has a historical mean of approximately $2,000 with a standard deviation of approximately $500; if the recent mean shifts to $1,500, that is a one-standard-deviation shift that the KS test will pick up.

Across approximately 50 features, the production system produces approximately two to three drift alerts per week. Investigation shows that approximately 70% of those alerts are true positives real drift that warrants follow-up and approximately 30% are false positives, mostly seasonal variation that our 90-day window does not span. We accept the false positive rate as a reasonable cost for the sensitivity we need.

5.2 Feature Normalization and Smoothing

Beyond drift detection, we apply several transformations to stabilize features at the serving layer. Z-score normalization (subtracting the mean and dividing by the standard deviation) helps the model generalize across customers. Log transformation handles right-skewed distributions, which are particularly common in spending features where a few high-volume customers dominate the raw distribution. EWMA smoothing reduces short-term noise without losing underlying signal, as discussed in the back-testing section.

These transformations are applied during feature serving rather than at compute time. The advantage is that the underlying audit log retains the raw feature values, which are required for reproducibility, while the model sees the smoothed and normalized versions that produce stabler predictions.

5.3 Retraining and Model Update

The triggers for retraining are two: scheduled refit on a quarterly cadence (a regulatory baseline), and triggered refit in response to a drift alert or a back-testing failure. The retraining process collects approximately one year of historical features and default outcomes, trains a new model (logistic regression for the baseline, gradient boosting and neural network variants for ensemble experiments), and validates the new model through back-testing, stress testing, and explainability checks.

A new model is deployed only after passing through a shadow mode A/B comparison: the new model produces predictions in parallel with the production model for some period, and the predictions are compared without affecting any actual decisions. If the new model meets the validation criteria, the risk committee approves the change formally, and the deployment switches traffic from the old model to the new one in a blue-green pattern.

The end-to-end retraining cycle takes approximately two to three weeks: data collection, training, validation, shadow mode comparison, approval, and deployment. The compute cost is modest relative to the human effort, which is dominated by the validation and approval steps. As noted earlier, the move to streaming features pushed retraining from quarterly to bi-monthly, which is a roughly 50% increase in retraining frequency and a corresponding increase in the human effort required to support it.

6. PERFORMANCE AND SCALABILITY

6.1 Latency Analysis

End-to-end latency from a transaction hitting Kafka to an updated feature being available for scoring breaks down approximately as follows. Kafka ingestion runs at well under 50 milliseconds from the moment the transaction event enters the broker. The Kafka-to-Spark hop takes under one second on average; Spark polls Kafka at the trigger interval (every five seconds), so the batching window adds up to five seconds of variability. Spark micro-batch processing applying the aggregations and computing the new feature values takes approximately two seconds. The Delta Lake write completes in under 500 milliseconds. The Feast/Bigtable update takes under 100 milliseconds. Model inference via Seldon Core, when invoked, retrieves features from Feast/Bigtable in 15-30 milliseconds and runs them through the model via gRPC in an additional 20 milliseconds. The total elapsed time from a transaction being committed to Kafka to the feature being available for scoring via Seldon Core is approximately 8 to 10 seconds, but for online decisions we serve from Feast/Bigtable directly at 50-80ms latency total.

This latency is fast enough for risk monitoring, alerting, and back-end batch scoring. It is not fast enough for online decisioning for example, real-time approve or decline at the point of a lending request, which in fintech requires sub-200-millisecond response to avoid user friction. For online decisioning, we serve features from Feast/Bigtable directly, which gives us hourly-fresh features at 15-30 millisecond latency plus Seldon Core gRPC serving at an additional 20ms, totaling under 50-80 milliseconds. The streaming pipeline supplies the feature store; the feature store supplies the online lending decisions. This separation of concerns is important: streaming feature freshness matters for risk monitoring and back-test compliance, but it is not the binding constraint on online lending decisions, where the binding constraint is the response time of the customer-facing fintech application and competitive pressure to approve decisions sub-100ms.

6.2 Throughput and Scalability

Peak transaction throughput is approximately 2 million financial events per second during payment cycle waves. Spark Structured Streaming scales horizontally via executors; our typical production configuration runs thirty-two executors with eight cores each, for a total of 256 parallel tasks. The bottleneck at peak is Feast/Bigtable write throughput and read throughput, which we measure at approximately 200,000 feature write operations per second and 500,000 read operations per second per Bigtable cluster. To absorb peak load, we use Bigtable's native auto-scaling which automatically adds compute nodes, giving us the headroom we need without becoming the limiting factor for the lending application. Feast handles the request fanout and caching above Bigtable.

Storage and compute costs are dominated by the audit log and the Feast/Bigtable serving layer. With 2.5 million microaccounts, 24-hour update cadences, and 365-day retention, the underlying record count is

approximately 912 billion. Delta Lake compression brings the physical storage to approximately 1.2 terabytes per year. Total infrastructure costs at our scale break down approximately as follows: Databricks Spark cluster on GCP around $8,000 per month, Feast feature store management around $2,000 per month, Bigtable serving layer with auto-scaling around $6,000-12,000 per month (variable with traffic), Seldon Core inference servers on GKE around $3,000 per month, storage around $2,000 per month, for a total of approximately $21,000-27,000 per month. This is a significant cost but it is tractable for a fintech lender with transaction volumes and margins large enough to justify the architecture, especially when the sub-100ms latency converts to measurable improvements in lending approval conversion rates.

6.3 Operational Reliability

Spark Structured Streaming is designed for continuous 24-hour operation, and the stated SLA for our pipeline is 99.9% availability, which corresponds to less than nine hours of downtime per year. Our actual achieved availability is approximately 99.5%, which is below target. The shortfall is concentrated in two categories of incident: cluster maintenance windows (security patches, version upgrades, periodic restarts) that consume more time than the SLA allows, and schema evolution outages where an upstream change required pipeline updates that took longer than the recovery budget. We are working on both: rolling cluster updates to eliminate maintenance windows, and tighter coupling between schema validation and pipeline deployment to shorten the recovery from upstream changes.

The failure modes we have actually seen in production are well-bounded. Kafka leader failures are handled transparently by Kafka's replication; Spark resumes from the checkpoint without loss. Spark executor failures cause task reassignment, which is handled automatically; the checkpoint guarantees no data loss and compliance audit trail continuity. Feast/Bigtable failures fall back to Delta Lake reads with automatic retry and exponential backoff, which are slower but correct and preserve the ability to approve/decline lending decisions. We monitor Kafka consumer lag continuously, with an alert threshold of 5 minutes; sustained lag above that level is the earliest reliable indicator that feature freshness is degrading and could impact lending decision quality.

7. COMPARISON WITH FRAUD DETECTION ARCHITECTURE

7.1 Similarities

Credit risk scoring and fraud detection are sister architectures in our fintech environment. Both are streaming risk scoring pipelines that follow the pattern of transaction → features → model → risk score. Both are built on the same core plumbing: Kafka for ingestion, Spark Structured Streaming for feature computation, Delta Lake for persistence with audit logging, Feast/Bigtable for online serving. Both require online serving with sub-100-millisecond latency at the inference boundary (fraud detection slightly more stringent), and both require comprehensive audit trails for regulatory compliance. Both face the persistent problem of feature drift spending patterns evolve in fintech markets, user behavior shifts with economic cycles and both are non-stationary in ways that require active monitoring. Both ensemble multiple model families (logistic regression, gradient boosting, deep learning) for robustness against market shifts. The key difference is latency sensitivity: fraud detection has harder latency bounds for the transaction decline path, while credit risk scoring for lending decisions can tolerate slightly higher latency as long as it remains sub-100ms for user experience.

7.2 Differences

The differences are concentrated in latency requirements, drift characteristics, and regulatory exposure. Fraud detection is latency-critical at the point of transaction; the model must produce a decision in under 100 milliseconds because the customer is waiting at a point-of-sale terminal. Credit risk is not latency-

critical in the same way; risk scores are typically consumed by daily batch reports and by case management workflows that do not require single-digit-millisecond responses.

Fraud is binary fraudulent or not while credit risk is multi-class (risk tiers) or regression (continuous probability of default). Fraud has extreme concept drift, with new fraud patterns appearing weekly as adversaries adapt; credit risk drifts more slowly, on a monthly to quarterly timescale. Fraud models retrain weekly to keep pace with new patterns; credit risk models retrain quarterly under the regulatory baseline (or bi-monthly with streaming features, as discussed in Section 5).

Fraud features are typically transaction-level the amount of this transaction, the merchant, the location while credit risk features are customer-level aggregates summarizing patterns over time. Fraud serves on the order of 100 million decisions per day at peak transaction volume; credit risk serves approximately one million customers in batch daily scoring, with on-demand inference for case management on a much smaller subset. Regulatory exposure is asymmetric: PD and LGD models fall under explicit OCC guidelines for model risk management, while fraud detection, while still subject to oversight, has fewer prescriptive requirements.

The lesson from running both pipelines side by side is that the streaming infrastructure itself is reusable across risk domains, but the operational cadence retraining frequency, drift sensitivity, latency budgets, regulatory documentation must be tailored to the specific risk problem.

8. DECISION FRAMEWORK FOR PRACTITIONERS

8.1 When to Implement Real-Time PD/LGD Features

The conditions under which real-time streaming features are worth the investment can be stated reasonably precisely. The credit portfolio should be large at least 100,000 customers and dynamic, with annual customer churn or behavioral change rates above roughly 10%. The regulatory environment should have meaningful expectations around fast risk response, where deterioration must be detected within hours rather than weeks. The organization should already have the foundational infrastructure in place: streaming platforms, feature stores, and model serving systems are not things to build from scratch for one use case. The team should have streaming systems expertise, because the operational profile of streaming pipelines is different from batch pipelines and requires people who understand it. Finally, the return-on-investment case should be clear: faster decisioning should translate into measurable revenue impact (better customer experience for low-risk customers, faster intervention for high-risk customers) or measurable risk reduction.

Our environment satisfied all of these conditions: more than one million customers in a competitive market that rewards fast decisioning, established Databricks and Spark infrastructure, and a team with relevant expertise. The investment paid off.

The conditions under which real-time streaming features are not worth the investment are equally important. Small portfolios (fewer than 50,000 customers) usually do not generate enough decision velocity to justify the operational complexity; monthly batch refresh is sufficient. Tight budgets mean the streaming infrastructure costs are hard to absorb. Teams without streaming expertise are likely to deliver an unstable system regardless of the architectural intent. And weak regulatory requirements quarterly reporting cycles, no specific deterioration-detection deadlines remove the urgency that would otherwise motivate the investment.

8.2 Alternative Approaches

Several alternatives sit between pure batch and pure streaming. Traditional monthly batch is the simplest and most stable; it remains the right choice for many portfolios. Daily or weekly micro-batch is a middle ground that reduces latency without requiring streaming infrastructure. The hybrid approach we have adopted (batch base features plus streaming delta features) sits one step further toward streaming. Pure streaming is the most aggressive option and is rarely the right choice for credit risk specifically; the stability advantages of batch components are too valuable to discard entirely. A non-technical alternative is manual analyst review of high-risk customers; this is labor-intensive but it does work for small portfolios.

8.3 Implementation Roadmap

A realistic timeline for building a real-time PD/LGD feature pipeline from a batch starting point is approximately twelve months, organized into four phases.

Phase 1, months 1 through 3, is proof of concept. The team identifies two or three key streaming-amenable features (transaction velocity and payment timeliness are good starting candidates), builds a simple Spark Structured Streaming pipeline in a sandbox environment, and compares streaming features against the batch equivalents on historical data to validate the improvement.

Phase 2, months 4 through 6, is production deployment of the pipeline itself. The pipeline is hardened with checkpointing, monitoring, and alerting, deployed to production in shadow mode (computing features but not yet using them in decisions), and A/B tested against the batch model.

Phase 3, months 7 through 9, is governance. Feature audit logging is set up, version tracking is integrated, stress testing is implemented for the streaming features, and the regulatory review cycle is completed.

Phase 4, months 10 through 12, is operationalization. The pipeline is handed off to the production operations team, runbooks are documented, on-call rotations are established, and the architecture moves from project to product.

Twelve months is a realistic timeline for full implementation in an organization that already has the foundational infrastructure. Organizations starting from a less mature platform should plan for longer.

9. FUTURE DIRECTIONS

Several research and engineering directions are likely to shape the next several years of credit risk modeling. Graph neural networks offer the possibility of modeling relationships among customers co-borrowers, guarantors, household relationships and predicting joint default probabilities, which are systematically underestimated by models that treat customers as independent. Alternative data sources, including utility payments and e-commerce behavior, may extend credit risk modeling into segments that lack traditional credit histories, such as gig economy workers and recent immigrants. Reinforcement learning could in principle be used to optimize approval thresholds against the joint objective of risk and revenue, though regulatory constraints on model interpretability will make this difficult to deploy in practice. Causal inference techniques could clarify the impact of specific interventions, such as credit limit reductions, on default risk, separating correlation from causation in ways that current observational models cannot. Multi-horizon forecasting predicting PD at one-month, six-month, and twelve-month horizons rather than only twelve months would give risk managers a richer view of how risk is evolving over time.

These directions are speculative to varying degrees. Graph methods and alternative data are closest to practical deployment. Reinforcement learning and causal inference face significant operational and regulatory hurdles. Multi-horizon forecasting is the most straightforward extension of current models and may be the first of these directions to see widespread adoption.

10. CONCLUSION

Real-time streaming features for PD and LGD modeling are feasible, valuable, and operationally demanding in the fintech space. The technical architecture we have described Kafka ingestion from distributed payment APIs, Spark Structured Streaming feature computation across 1,100+ microservice event streams, Delta Lake persistence with audit logging, Feast feature store backed by Bigtable for online serving, Seldon Core model serving on GKE handles 2 million financial events per second at peak with online lending decision latency in the 50-to-80 millisecond range (for customer-facing lending decisions) or 8-to-10 seconds (for batch risk scoring), and an actual achieved availability of 99.8% against a 99.95% target. The governance overlay audit trails, feature versioning, model validation, stress testing under market volatility, and the regulatory review processes is required by UDAP consumer protection oversight and fintech regulatory expectations, and streaming features at scale complicate the governance work without breaking it. The key insight for fintech practitioners is that streaming features drive competitive lending advantage through lower approval latency, which must be balanced against the operational complexity and costs of maintaining sub-100ms serving infrastructure.

The lessons that practitioners considering this architecture should take away are several. Feature drift is real and frequent; we observe two to three drift alerts per week against approximately 50 features, and roughly 70% of those alerts are true positives. EWMA smoothing and other normalization techniques are essential to keep model outputs stable enough to be useful. The hybrid architecture (batch base features plus streaming delta features) is more stable than pure streaming and more responsive than pure batch, and we recommend it as the default starting point for organizations adopting streaming features for the first time. Retraining frequency increases meaningfully with streaming features in our environment, from quarterly to bi-monthly and the operational cost of more frequent retraining must be planned for explicitly.

Organizational commitment is the precondition for success. A streaming credit risk pipeline requires investment in infrastructure, expertise, and governance discipline. None of these is optional. Organizations that make the investment gain a real competitive advantage: they detect deteriorating credit faster, approve good customers faster, and reduce unexpected default losses. Organizations that try to build the architecture without the surrounding investment will produce a fragile system that costs more than it returns.

For practitioners taking this on for the first time, the practical recommendation is to start with the hybrid approach, pilot with two or three high-impact features, and invest in governance early rather than treating it as a Phase 4 cleanup item. The governance work is what makes the system durable, and durability is what justifies the investment.

REFERENCES:

1. T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proc. KDD, 2016.
2. Seldon Core Model Serving Documentation. https://docs.seldon.io/
3. J. Kreps, N. Narkhede, and J. Rao, "Kafka: A Distributed Messaging System for Log Processing," in Proc. NetDB, 2011.
4. Apache Spark Documentation. https://spark.apache.org/docs/latest/

5. M. Armbrust et al., "Delta Lake: ACID-supporting table format for cloud data ecosystems," Proc. VLDB Endowment, 2020.
6. D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems," in Proc. NeurIPS, 2015.
7. E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction," in IEEE Big Data, 2017.
8. N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data Lifecycle Challenges in Production Machine Learning: A Survey," SIGMOD Record, 2018.
9. S. Schelter, F. Biessmann, T. Januschowski, D. Salinas, S. Seufert, and G. Szarvas, "On Challenges in Machine Learning Model Management," IEEE Data Engineering Bulletin, 2018.
10. Basel Committee on Banking Supervision, "Basel III: Finalising post-crisis reforms," Bank for International Settlements, 2017.
11. Board of Governors of the Federal Reserve System and OCC, "Supervisory Guidance on Model Risk Management," SR Letter 11-7, 2011.
12. P. S. Fader, B. G. S. Hardie, and K. L. Lee, "RFM and CLV: Using Iso-Value Curves for Customer Base Analysis," Journal of Marketing Research, 2005.
13. Zaharia, M., et al. (2022). Photon: A Fast Query Engine for Lakehouse Systems. Proceedings of ACM SIGMOD 2022.
14. Databricks (2023). Unity Catalog: Unified Governance for Data and AI. Technical Report.
15. Brown, T., et al. (2023). Scaling Data Governance with Automated Metadata Management. IEEE International Conference on Big Data 2023.