Real-Time Financial Fraud Detection & Monitoring System

A Proof of Concept on Real-Time Financial Fraud Detection & Monitoring System

---

Project Type

Proof of Concept (POC) – Real-Time Financial Fraud Detection & Monitoring System

---

Prepared By

Team Name: StreamForce

Project Leadership

Project Team Leader
Bhagath B

Project Team Members & Roles

I. Data Ingestion Engineer

- Bhagath B

II. Transformation Engineer

- Jothilingam D

III. Machine Learning Engineer

- Devesh raaj

IV. Dashboard & Documentation Engineer

- Angad Kumar

Project Foundations

0.1 Business Context

Financial institutions process **millions of digital transactions every day** across multiple channels such as:

- Online payments
- Point-of-sale (POS) transactions
- Card-not-present transactions
- Cross-border payments

With the rapid growth of digital payments, fraud patterns have become:

- Highly dynamic
- Low-latency
- Distributed across geographies and devices

Traditional **batch-based fraud detection systems** are no longer sufficient, as they:

- Detect fraud after financial loss has occurred
- Fail to adapt quickly to new fraud behaviors
- Provide limited real-time visibility to operations teams

0.2 Problem Statement

The organization requires a **real-time fraud detection and monitoring system** that can:

- Ingest transaction data continuously
- Detect suspicious activity as transactions occur
- Scale with increasing transaction volumes
- Provide immediate visibility to fraud analysts
- Maintain auditability and transparency

Core Challenges Identified

1. **Latency** – Delayed detection leads to higher financial loss
2. **Volume** – Millions of transactions per day
3. **Fraud Rarity** – Fraud events are typically <5% of total traffic
4. **Evolving Patterns** – Fraud tactics change frequently

5. **Operational Visibility** – Analysts need real-time insights

---

0.3 Project Objective

The primary objective of this project is to **design and implement a production-aligned, real-time fraud detection platform** capable of identifying anomalous transactions and providing actionable insights to business and operations teams.

High-Level Objectives

- Detect fraudulent transactions in **near real time**
- Reduce detection latency from hours to seconds
- Enable proactive fraud prevention
- Provide business-ready dashboards and metrics
- Ensure scalability and fault tolerance

0.4 Scope Definition

In Scope

- Real-time transaction ingestion using streaming pipelines
- Data processing using Medallion Architecture (Bronze–Silver–Gold)
- Anomaly-based fraud detection using machine learning
- Real-time fraud scoring and alert generation
- Interactive dashboards for fraud monitoring

Out of Scope

- Direct integration with banking core systems
- Automated blocking or transaction rejection
- Manual case management workflows
- Regulatory reporting automation

> **Note:** The project is positioned as a **platform-level proof of concept**, designed to be extensible into a full production system.

0.5 Success Criteria

The project will be considered successful if:

Category	Success Metric
Latency	Fraud scoring occurs within seconds
Scalability	Pipeline handles large transaction volumes
Accuracy	Model identifies anomalous behavior effectively
Reliability	Fault-tolerant streaming execution
Usability	Dashboards enable clear fraud monitoring
Reproducibility	ML models and data pipelines are repeatable

0.6 Key Assumptions

- Historical fraud data is representative of real-world patterns
- Synthetic streaming simulation approximates live transaction flows
- Anomaly detection is sufficient for initial fraud identification
- Cloud-based analytics infrastructure is available
- Fraud analysts consume insights via dashboards rather than raw data

---

0.7 Constraints

Technical Constraints

- No access to live banking transaction streams
- Open-source and free datasets only
- Model inference must support streaming execution

Operational Constraints

- Limited labeled data for real-time fraud
- Model must adapt to unseen fraud patterns
- Streaming jobs must remain cost-efficient

0.8 Non-Functional Requirements

Requirement	Description
Scalability	Must scale horizontally with data volume
Fault Tolerance	Recover from failures without data loss
Observability	Monitor pipeline health and performance
Data Governance	Maintain schema stability and lineage
Maintainability	Modular, team-owned components

0.9 Stakeholder Perspective

Primary Stakeholders

- Fraud Operations Team
- Data Engineering Team
- Machine Learning Team
- Business & Risk Analysts

Stakeholder Expectations

- Real-time visibility into fraud activity
- Trustworthy and explainable outputs
- Minimal operational complexity
- Clear documentation and traceability

0.10 Architectural Philosophy

This project follows these core principles:

- **Streaming-First Design**
- **Separation of Concerns**
- **Medallion Architecture**
- **Batch Training, Streaming Inference**
- **Business-Ready Outputs**

These principles guide all technical and design decisions throughout the project lifecycle.

0.11 Phase Completion Summary

PHASE 0 establishes the foundation by clearly defining:

- The business problem
- Project goals
- Scope boundaries
- Constraints and assumptions
- Success criteria

This ensures all downstream design and implementation decisions remain **aligned with real business needs**.

PHASE 1 – Data Ingestion Architecture

1.1 Purpose of Data Ingestion Layer

The Data Ingestion layer serves as the **entry point** of the fraud detection platform. Its primary responsibility is to **reliably ingest financial transaction data as a continuous stream**, ensuring that all downstream components receive data in a **consistent, fault-tolerant, and scalable manner**.

This layer is designed to:

- Support near real-time processing
- Preserve raw data fidelity
- Enable reprocessing and auditability
- Scale with transaction volume

1.2 Data Source Selection

1.2.1 Dataset Used

IEEE-CIS Fraud Detection Dataset was selected due to:

- Large transaction volume
- Realistic fraud distribution (~3.5%)
- Industry relevance
- Availability as an open-source dataset

1.2.2 Source Tables

Dataset	Description
Transaction Table	Core transaction attributes (amount, cards, addresses)
Identity Table	Device, browser, and network metadata
Sample Submission	Reference file (not used in ingestion)

> **Design Decision:**
> The `sample_submission.csv` file is excluded from ingestion, as it does not represent transactional events.

1.3 Streaming Simulation Strategy

1.3.1 Problem Addressed

Live banking transaction streams are **not publicly available**. To emulate real-time behavior, historical data must be transformed into a **continuous event stream**.

1.3.2 Simulation Approach

The system uses a **Python-based streaming data generator** that:

- Reads historical IEEE-CIS data
- Emits small batches of transactions at regular intervals
- Writes incremental files into a monitored directory
- Simulates transaction arrival patterns

1.3.3 Design Principles

Principle	Explanation
Micro-batching	Mimics real-time ingestion
Stateless generation	Ensures reproducibility
Time-based emission	Controls ingestion rate
Configurable volume	Enables stress testing

1.4 Ingestion Technology Selection

1.4.1 Databricks Structured Streaming

Chosen for:

- Native integration with Delta Lake
- Exactly-once processing guarantees
- Fault-tolerant execution model
- Seamless scalability

1.4.2 Auto Loader (cloudFiles)

Auto Loader is used to ingest streaming files because it:

- Automatically detects new files
- Handles schema inference and evolution
- Maintains ingestion checkpoints
- Simplifies streaming file ingestion

1.5 Bronze Layer Design

1.5.1 Role of the Bronze Layer

The Bronze layer stores **raw, immutable streaming data** exactly as it arrives from the source.

Key characteristics:

- Append-only
- No business transformations
- Schema enforcement
- Ingestion metadata captured

1.5.2 Bronze Table Contents

Each record includes:

- Original transaction fields
- Event timestamp
- Ingestion timestamp
- Source file metadata
- Corrupt record capture (`_rescued_data`)

1.5.3 Schema Philosophy

> **No transformation at Bronze level.**
> Any data modification at this stage would compromise auditability and reprocessing capability.

1.6 Fault Tolerance & Reliability

1.6.1 Exactly-Once Semantics

Structured Streaming ensures:

- No duplicate records
- No data loss during failures
- Consistent processing guarantees

1.6.2 Checkpointing Strategy

- Each ingestion pipeline uses an isolated checkpoint
- Checkpoints track:
    - Processed files
    - Offsets
    - Streaming state

Operational Insight:
Checkpoint reset is used **only during development** to reprocess data. In production, checkpoints remain stable.

1.7 Data Quality Handling

Scenario	Handling
Schema mismatch	Captured via _rescued_data
Corrupt records	Preserved for analysis
Missing values	Passed downstream (not cleaned here)

Rationale:
Data quality remediation belongs to the **Silver layer**, not ingestion.

1.8 Security & Governance Considerations

- No sensitive PII is exposed
- Raw data access restricted to engineering teams
- Lineage preserved via metadata
- Schema evolution controlled

1.9 Performance Considerations

- File sizes optimized for streaming efficiency
- Controlled ingestion rate
- Auto Loader scalability ensures horizontal growth
- Bronze tables stored as Delta Lake for fast reads

1.10 Phase Completion Summary

PHASE 1 delivers:

- A scalable streaming ingestion pipeline
- Reliable Bronze layer storage
- Audit-friendly raw data capture
- A foundation for downstream processing

All subsequent transformations and analytics rely on the **stability and correctness** of this layer.

PHASE 2 – Data Transformation & Silver Layer

2.1 Purpose of the Silver Layer

The Silver layer represents the **trusted, standardized, and analytics-ready view of transactional data**. Its primary objective is to transform **raw streaming data from the Bronze layer** into **clean, structured datasets** that can safely support:

- Machine learning training
- Real-time fraud scoring
- Business analytics
- Downstream aggregations

Unlike the Bronze layer, which prioritizes data fidelity, the Silver layer prioritizes **data quality, consistency, and stability**.

2.2 Design Principles

The following principles guided all transformation decisions:

Principle	Description
Streaming safety	All transformations must work under streaming execution
Determinism	Same input produces same output
Minimal state	Avoid long-lived streaming state
Schema stability	Prevent downstream breakage
Reusability	Data usable across multiple teams

2.3 Input Data Characteristics

The Silver layer consumes data from:

- Bronze streaming table (`stream_bronze_data`)
- Structured Streaming execution model

- Event-time based ingestion

Challenges Identified

- All fields initially ingested as strings
- Mixed data quality
- Late-arriving events
- High column cardinality

2.4 Data Cleaning Strategy

2.4.1 Type Casting & Standardization

All raw fields are explicitly cast to correct data types:

Column	Data Type
TransactionID	BIGINT
TransactionDT	BIGINT
TransactionAmt	DOUBLE
isFraud	INT
card1	INT
event_timestamp	TIMESTAMP

Rationale:
Explicit casting avoids silent schema drift and ensures ML compatibility.

2.4.2 Null Handling

- Transactions with missing or invalid amounts are removed
- Optional categorical fields are retained as-is
- Nulls are not aggressively imputed at this stage

Rationale:
Imputation decisions depend on ML and analytics use cases and are deferred downstream.

2.5 Event-Time Handling

2.5.1 Event Timestamp Standardization

- `event_timestamp` is treated as the **single event-time column**
- All downstream time-based logic references this column exclusively

---

2.5.2 Watermark Strategy

A watermark of **10 minutes** is applied to handle late-arriving events:

withWatermark(event_timestamp, "10 minutes")

Why Watermarks Matter

- Prevent unbounded state growth
- Enable safe downstream joins
- Maintain streaming stability

> **Critical Rule Enforced:**
> Only **one event-time column** exists per streaming DataFrame to avoid Spark execution errors.

2.6 Feature Preparation Philosophy

2.6.1 Why Only Row-Level Features?

Early experimentation revealed that:

- Streaming window aggregations produced empty or unstable outputs
- `availableNow` triggers did not emit partial windows
- Stream–stream joins introduced complexity and errors

Final Decision

> **Silver layer contains only stateless, row-level features.**

2.6.2 Row-Level Features Created

Feature	Description
`is\_high\_value\_txn`	Flag for high-amount transactions
`log\_transaction\_amount`	Log-scaled transaction amount
`is\_international\_txn`	Address mismatch indicator

These features:

- Do not require aggregation
- Are streaming-safe
- Are reusable for ML and analytics

2.7 Silver Base Table

Purpose

The Silver Base table (`silver_transactions_base`) acts as the **canonical transactional dataset** for the entire platform.

Key Characteristics

- Streaming table
- Clean schema
- Event-time aware
- Reusable across batch and streaming workloads

2.8 Batch Feature Engineering for ML

Why Batch Instead of Streaming?

Streaming aggregations:

- Depend on window completion
- Do not guarantee deterministic output
- Are difficult to debug

Batch Aggregations Enable:

- Reproducible feature generation
- Stable ML training datasets

- Time-range joins without streaming state

Architectural Separation:
Streaming → real-time scoring
Batch → ML training & feature generation

2.9 Metadata Management

- Ingestion metadata (`_source_file`, `_file_size`, etc.) preserved in Silver
- Metadata is **not propagated to Gold**
- Enables traceability and debugging

2.10 Error Handling & Lessons Learned

Common Issues Encountered

- Multiple event-time column conflicts
- Stream–stream join limitations
- Empty streaming windows

Resolutions

- Strict event-time discipline
- Stateless Silver layer design
- Batch-based feature computation

2.11 Phase Completion Summary

PHASE 2 delivers:

- Clean, standardized Silver datasets
- Streaming-safe transformations
- Stable schema for ML and analytics
- Clear separation between ingestion and feature logic

This phase establishes the **data foundation required for reliable fraud detection**.

PHASE 3 – Feature Engineering Strategy

3.1 Role of Feature Engineering in Fraud Detection

Feature engineering is the **primary driver of fraud detection effectiveness**. Fraud rarely manifests as a single abnormal transaction; instead, it emerges through **patterns of behavior over time**.

This phase defines how transactional data is transformed into **signals that expose abnormal behavior**, while ensuring compatibility with:

- Real-time inference
- Batch training
- Scalable execution

3.2 Feature Design Principles

All features follow these core principles:

Principle	Description
Behavioral focus	Capture user and card behavior
Temporal awareness	Reflect changes over time
Streaming compatibility	Safe for real-time scoring
Reproducibility	Deterministic feature values
Interpretability	Explainable to stakeholders

3.3 Feature Categories

The feature set is divided into **two complementary categories**.

3.4 Transaction-Level Features (Stateless)

Purpose

Capture **single-transaction risk indicators**.

Characteristics

- Computed per row
- No historical dependency
- Streaming-safe

Features Implemented

Feature	Description
TransactionAmt	Raw transaction amount
log_transaction_amount	Log-transformed amount
is_high_value_txn	High-value threshold flag
is_international_txn	Address mismatch indicator

Rationale

These features provide **immediate risk context** and serve as foundational inputs to the ML model.

3.5 Behavioral Features (Stateful)

Purpose

Capture **patterns across multiple transactions**.

Behavioral Signals Captured

Signal	Meaning
Transaction frequency	Sudden bursts of activity
Average amount	Deviations from normal spending
Amount variance	Volatility in transaction behavior
Merchant diversity	Unusual merchant patterns

3.6 Batch-Based Behavioral Feature Generation

Why Batch Processing?

During experimentation, streaming window aggregations introduced:

- Non-deterministic outputs
- Empty results under finite triggers
- Complex state management

Final Decision

> Behavioral features are computed **in batch**, not streaming.

This ensures:

- Stable ML training datasets
- Reproducibility across experiments
- Simplified debugging and validation

3.7 Time Window Design

Window Size Selection

Window	Reason
5 minutes	Captures rapid fraud bursts
Sliding logic	Models short-term behavior

Design Considerations

- Short enough to detect quick attacks
- Long enough to capture behavioral patterns
- Aligned with event-time semantics

3.8 Feature Aggregation Strategy

Features are aggregated **per card (`card1`)**, as fraud often clusters around card-level entities.

Aggregated Features

Feature	Description
txn_count_5min	Number of transactions
avg_amount_5min	Average spend
stddev_amount_5min	Spend variability
product_diversity_5min	Merchant variety

3.9 Feature Joining Strategy

Training Join Logic

- Features joined to transactions using:
    - Card identifier
    - Time-range conditions

Why Time-Range Join?

- Prevents data leakage
- Ensures temporal correctness
- Aligns behavior with transaction time

3.10 Feature Stability & Drift Awareness

Stability Measures

- Fixed window sizes
- Explicit aggregation logic
- Controlled feature definitions

Drift Indicators (Future Scope)

- Shifts in feature distributions
- Changes in fraud score distribution
- Increase in false positives

3.11 Feature Reusability

The same feature set is used for:

- ML training
- Real-time scoring
- Analytics & dashboards

This ensures **consistency across the entire platform**.

3.12 Phase Completion Summary

PHASE 3 delivers:

- Well-defined fraud-relevant features
- Clear separation between stateless and behavioral features
- Stable, reproducible feature engineering
- Alignment with real-time ML constraints

This phase provides the **signal foundation** required for effective anomaly detection.

PHASE 4 – Machine Learning Architecture

4.1 Role of Machine Learning in the System

Machine learning is responsible for **identifying anomalous transaction behavior** that may indicate fraud. Unlike rule-based systems, the ML component adapts to **previously unseen patterns**, making it suitable for evolving fraud tactics.

This system employs **anomaly detection**, rather than pure supervised classification, to handle:

- Limited real-time labels
- Delayed fraud confirmation
- Emerging fraud behaviors

4.2 Model Selection Rationale

4.2.1 Why Anomaly Detection?

Challenge	Solution
Fraud rarity	Anomaly-based approach
Label delay	Unsupervised learning
Pattern evolution	Behavior-driven detection

4.2.2 Selected Model: Isolation Forest

Isolation Forest was selected due to:

- Ability to isolate rare observations
- Scalability to large datasets
- Minimal feature assumptions
- Strong industry adoption

> **Key Insight:**
> Fraudulent transactions are more “isolated” in feature space than normal behavior.

4.3 Training Data Strategy

4.3.1 Training Data Source

Training data is sourced from:

- `silver_transactions_base`
- Batch-generated behavioral features

Only **historical, complete datasets** are used for training.

4.3.2 Batch Training Philosophy

> **All ML training is performed in batch mode.**

Reasons:

- Deterministic training runs
- Reproducible experiments
- Easier evaluation and debugging
- Stable feature availability

Streaming data is **never used directly** for model training.

4.4 Feature Vector Composition

Input Features

Category	Features
Transaction	Amount, log amount
Risk flags	High-value, international
Behavior	Frequency, avg, stddev, diversity

All features are:

- Numeric
- Scaled
- Free of nulls

4.5 Model Training Pipeline

Training Workflow :

Silver Layer (Batch Read)

↓

Batch Feature Aggregation

↓

Feature Scaling

↓

Isolation Forest Training

↓

Model Evaluation

4.5.1 Feature Scaling

- StandardScaler is applied
- Prevents dominance of large numeric ranges
- Ensures stable tree isolation behavior

4.6 Model Evaluation Strategy

Although the model is **unsupervised**, historical fraud labels are used **only for evaluation**.

Metrics Used

Metric	Purpose
ROC-AUC	Ranking performance
PR-AUC	Precision on rare fraud

> **Important:**
> Labels are **not** used during training, only during validation.

4.7 MLflow Integration

4.7.1 Why MLflow?

MLflow is used to:

- Track experiments
- Log parameters and metrics
- Register trained models
- Enable reproducible deployments

4.7.2 Unity Catalog Considerations

Due to Unity Catalog constraints:

- Model stages are not used
- Model **aliases** are used instead
- Model signatures are mandatory

This ensures:

- Compliance
- Version control
- Deployment safety

4.8 Model Registration & Versioning

Each training run:

- Logs hyperparameters
- Logs evaluation metrics
- Registers a new model version
- Preserves input/output schema

This enables:

- Rollbacks
- Model comparison
- Auditability

4.9 Separation of Training and Inference

Aspect	Training	Inference
Mode	Batch	Streaming
Data	Historical	Live
Execution	Deterministic	Low-latency
Frequency	Periodic	Continuous

This separation is **critical** for production stability.

4.10 Phase Completion Summary

PHASE 4 delivers:

- A production-aligned ML architecture
- Robust anomaly detection logic
- Reproducible training pipelines
- Governed ML lifecycle via MLflow

This phase enables the system to **identify suspicious behavior accurately and consistently**.

PHASE 5 – Real-Time Fraud Scoring

5.1 Objective of Real-Time Scoring

The purpose of real-time fraud scoring is to **apply trained machine learning models to live transaction streams** and generate **actionable risk signals within seconds of transaction occurrence**.

This phase operationalizes the ML model by embedding it directly into the **streaming data pipeline**, enabling:

- Near real-time anomaly detection
- Continuous fraud monitoring
- Low-latency business response

5.2 Design Constraints

Real-time ML scoring introduces unique challenges:

Constraint	Implication
Streaming execution	No long-running ML state
Event-time handling	Late-arriving data
Throughput	Millions of records
Model loading	Efficient reuse
Fault tolerance	No duplicate scoring

These constraints guided all architectural decisions in this phase.

5.3 Inference Strategy Selection

5.3.1 Batch vs Streaming Inference

Option	Evaluation
Stream–stream ML	Complex, unstable
Row-by-row scoring	High overhead
Micro-batch scoring	✅ Selected

5.3.2 Selected Approach: `foreachBatch`

`foreachBatch` enables:

- Processing streaming data in micro-batches
- Applying Python-based ML inference
- Writing results using batch semantics

> **Key Advantage:**
> This approach avoids streaming state while preserving real-time behavior.

5.4 Scoring Pipeline Flow

Silver Transactions (Streaming)
↓
Micro-Batch Trigger
↓
Feature Preparation
↓
ML Model Inference
↓
Fraud Score & Flag Generation
↓
Gold Fraud Predictions Table

5.5 Model Loading Strategy

- ML model is loaded once per streaming job
- Model retrieved via MLflow registry
- Input schema enforced at inference time

This ensures:

- Low latency

- Schema consistency
- Deployment safety

5.6 Feature Handling During Scoring

5.6.1 Feature Consistency

- Same features used as training
- Explicit type casting
- Missing behavioral features filled with defaults

Rationale:
Real-time scoring must remain robust even when certain aggregated features are unavailable.

5.6.2 Schema Enforcement

Strict schema matching is enforced to prevent:

- Silent feature mismatches
- Model input errors
- Invalid predictions

5.7 Fraud Score Interpretation

5.7.1 Fraud Score

- Derived from Isolation Forest anomaly score
- Higher score = higher suspicion

5.7.2 Anomaly Flag

A binary flag (`is_anomaly`) is generated using a configurable threshold.

Value	Meaning
1	Suspicious transaction
0	Normal transaction

5.8 Gold Output Design

Output Table: `fraud_predictions`

Column	Description
TransactionID	Unique transaction identifier
event_timestamp	Event time
card1	Entity identifier
TransactionAmt	Amount
fraud_score	ML anomaly score
is_anomaly	Fraud flag

The table is:

- Pre-created
- Append-only
- Schema-stable

5.9 Checkpointing Strategy

Purpose

Ensure **exactly-once scoring** and recovery from failures.

Strategy

- Dedicated checkpoint directory per scoring job
- Checkpoints track processed offsets
- Reset checkpoints only during development

> **Operational Insight:**
> Changing checkpoints reprocesses data—used intentionally for backfills and testing.

5.10 Failure Handling

Scenario	Handling
Empty micro-batch	Safely skipped
Model load failure	Job fails fast
Schema mismatch	Explicit error
Partial write	Atomic Delta commit

5.11 Performance Considerations

- Pandas conversion optimized per batch
- Controlled batch sizes
- No long-lived state
- Scales horizontally with cluster size

5.12 Phase Completion Summary

PHASE 5 delivers:

- Reliable real-time ML inference
- Low-latency fraud scoring
- Exactly-once prediction guarantees
- Stable Gold-level transaction outputs

This phase enables **operational fraud detection in real time**.

PHASE 6 – Gold Layer & Business Metrics

6.1 Purpose of the Gold Layer

The Gold layer represents the **final, business-consumable view** of the fraud detection system. Its purpose is to transform **machine-learning-scored transactions** into **aggregated, interpretable, and decision-ready datasets**.

This layer is optimized for:

- Fraud monitoring
- Operational reporting
- Trend analysis
- Executive visibility

Unlike Silver, the Gold layer contains **no raw or intermediate data**—only **high-value analytical outputs**.

6.2 Design Principles

The Gold layer is designed using the following principles:

Principle	Description
Business alignment	Metrics map directly to fraud KPIs
Read optimization	Fast query performance
Schema stability	Predictable analytics
Minimal complexity	Easy dashboard integration
Audit readiness	Transparent calculations

6.3 Gold Data Inputs

The Gold layer consumes data from:

- `fraud_predictions` (ML-scored transactions)
- Stable, append-only Delta tables
- Event-time aligned transaction records

No direct dependency exists on Bronze or Silver tables.

6.4 Core Gold Tables

6.4.1 Transaction-Level Gold Table

Table: `fraud_predictions`

Purpose:
Store ML-scored transaction events.

Column	Description
TransactionID	Unique transaction identifier
event_timestamp	Event occurrence time
card1	Card/entity identifier
TransactionAmt	Transaction amount
fraud_score	ML anomaly score
is_anomaly	Fraud indicator

This table serves as the **foundation for all downstream analytics**.

6.4.2 Time-Based Fraud Metrics

Table: `gold_fraud_metrics_time`

Purpose:
Monitor fraud trends over time.

Metrics Included:

- Total transactions
- Fraud transactions
- Fraud rate (%)

Business Questions Answered:

- Is fraud increasing?
- Are there sudden spikes?
- How does fraud evolve over time?

6.4.3 High-Risk Entity Metrics

Table: `gold_high_risk_cards`

Purpose:
Identify repeat offenders and risky entities.

Metrics Included:

- Total transactions per card
- Fraud count per card
- Fraud rate per card

Business Questions Answered:

- Which cards require investigation?
- Is fraud concentrated or distributed?

6.4.4 Fraud Alert View

Table: `gold_latest_fraud_alerts`

Purpose:
Provide an operational view of current fraud alerts.

Key Attributes:

- Sorted by fraud score
- Near real-time visibility
- Analyst-friendly format

6.5 Key Fraud KPIs Defined

KPI	Definition	Usage
Fraud Count	Number of anomalous transactions	Volume monitoring
Fraud Rate	Fraud / total transactions	Risk normalization

Fraud Score	Model confidence	Alert prioritization
High-Risk Entity	Repeat fraud occurrence	Investigation focus
Fraud Amount	Total suspicious amount	Impact assessment

6.6 Aggregation Strategy

6.6.1 Time Granularity

- Aggregations performed at **minute-level**
- Supports both real-time monitoring and trend analysis
- Balances granularity with performance

6.6.2 Event-Time Alignment

- All aggregations use `event_timestamp`
- Ensures temporal accuracy
- Prevents ingestion-time bias

6.7 Data Quality & Consistency

- All metrics derived from stable Gold tables
- No null-sensitive calculations
- Controlled joins and groupings
- Deterministic aggregations

6.8 Governance & Compliance Considerations

- No raw PII exposed
- Metrics are explainable
- Calculations documented
- Data lineage preserved

6.9 Operational Usage Patterns

Fraud Operations Team

- Monitor fraud spikes
- Investigate alerts
- Prioritize cases

Business & Risk Teams

- Analyze fraud trends
- Measure impact
- Inform strategy

Engineering Teams

- Monitor pipeline health
- Validate outputs

6.10 Phase Completion Summary

PHASE 6 delivers:

- Business-ready fraud metrics
- Stable analytics tables
- Clear KPIs for monitoring
- Foundation for dashboards and reporting

This phase ensures that **ML insights are translated into actionable business intelligence**.

PHASE 7 – Dashboards & Monitoring

7.1 Purpose of Dashboards & Monitoring

The dashboard layer is the **primary interface between the fraud detection system and business users**. Its purpose is to convert **machine-learning outputs and aggregated metrics** into **clear, actionable insights** that support:

- Real-time fraud monitoring
- Operational decision-making
- Investigation prioritization
- Executive reporting

Dashboards ensure that fraud risks are **visible, interpretable, and actionable**.

7.2 Target Users & Usage Scenarios

7.2.1 Fraud Operations Team

- Monitor fraud activity in real time
- Identify suspicious transactions
- Respond quickly to fraud spikes

7.2.2 Risk & Business Analysts

- Analyze fraud trends
- Understand fraud distribution
- Measure fraud impact

7.2.3 Engineering & Platform Teams

- Validate pipeline outputs
- Monitor data freshness
- Ensure system reliability

7.3 Dashboard Design Principles

The dashboards follow industry best practices:

Principle	Description
Real-time visibility	Near-live updates
Clarity	Minimal visual clutter
Actionability	Focus on decision-driving metrics
Consistency	Standard time granularity
Drill-down support	Investigation-ready views

7.4 Dashboard Categories

Dashboards are divided into **four logical categories**, each serving a distinct purpose.

7.5 Fraud Overview Dashboard

Objective

Provide a **high-level snapshot** of fraud activity across the system.

Key Visualizations

- Total transactions vs fraud transactions
- Fraud count over time
- Fraud rate (%) trend

Data Sources

- `gold_fraud_metrics_time`

Business Questions Answered

- Is fraud activity increasing?
- Are there sudden spikes?
- How does fraud compare to normal traffic?

7.6 Fraud Trend & Rate Dashboard

Objective

Analyze **temporal patterns** in fraud behavior.

Key Visualizations

- Fraud count per minute
- Fraud rate moving averages
- Time-based comparisons

Value

- Detect coordinated fraud attacks
- Identify unusual transaction bursts

7.7 High-Risk Entity Dashboard

Objective

Identify **repeat offenders and risky entities**.

Key Visualizations

- Top high-risk cards
- Fraud concentration distribution
- Fraud rate per entity

Data Sources

- `gold_high_risk_cards`

Business Questions Answered

- Which cards are responsible for most fraud?
- Is fraud concentrated or spread out?

7.8 Fraud Alert Monitoring Dashboard

Objective

Enable **operational fraud response**.

Key Visualizations

- Live fraud alerts table
- Sorted by fraud score
- Filters by card, amount, timestamp

Data Sources

- `gold_latest_fraud_alerts`

Value

- Immediate visibility into suspicious transactions
- Faster investigation and escalation

7.9 Fraud Impact Dashboard

Objective

Quantify the **business impact of fraud detection**.

Key Visualizations

- Total fraud amount detected
- High-value fraud transactions
- Severity distribution

Business Questions Answered

- What is the financial exposure?
- Are high-value frauds increasing?

7.10 Monitoring & Operational Metrics

Beyond fraud insights, dashboards support **system monitoring**:

Metric	Purpose
Data freshness	Detect ingestion delays
Event volume	Identify pipeline issues
Alert velocity	Prevent analyst overload

7.11 Alerting Strategy (Future Scope)

While this project focuses on dashboards, the design supports future alerting:

- Threshold-based alerts on fraud rate
- Spike detection
- Integration with email / messaging systems

7.12 Governance & Access Control

- Dashboards read only Gold-layer data
- Access restricted by user roles
- Sensitive data masked or excluded
- Metrics documented and auditable

7.13 Documentation Integration

Each dashboard includes:

- Metric definitions
- Data source references
- Interpretation guidance
- Known limitations

This ensures dashboards are **self-explanatory and trustworthy**.

7.14 Phase Completion Summary

PHASE 7 delivers:

- Real-time fraud visibility
- Business-ready dashboards
- Clear operational insights
- A monitoring layer aligned with real-world fraud workflows

This phase ensures that **machine learning outputs translate into real business action**.

PHASE 8 – Operational & Production Considerations

8.1 Purpose of This Phase

PHASE 8 ensures that the fraud detection platform is **not just functional**, but also:

- Scalable
- Reliable
- Secure
- Maintainable
- Ready for real-world deployment

This phase addresses the **operational lifecycle** of the system after development and initial rollout.

8.2 Scalability Strategy

8.2.1 Horizontal Scalability

The platform is designed to scale horizontally at every layer:

Component	Scaling Mechanism
Streaming ingestion	Auto Loader parallelism
Silver transformations	Structured Streaming
ML scoring	Micro-batch parallelism
Dashboards	Read-optimized Gold tables

Key Insight

> The system scales with data volume rather than user load.

8.3 Performance Optimization

Ingestion Layer

- Optimized file sizes for streaming
- Controlled micro-batch intervals
- Efficient schema inference

Processing Layer

- Stateless transformations
- Minimal streaming state
- Batch feature computation

Analytics Layer

- Pre-aggregated Gold tables
- Time-based partitioning
- Cached dashboard queries

8.4 Reliability & Fault Tolerance

8.4.1 Checkpoint Management

- Dedicated checkpoints per streaming job
- Exactly-once processing guarantees
- Controlled checkpoint resets for backfills

> **Operational Rule:**
> Checkpoint resets are never performed in production without change approval.

8.4.2 Failure Recovery

Failure Type	Recovery Mechanism
Node failure	Automatic retry
Job restart	Resume from checkpoint
Partial writes	Atomic Delta commits
Data corruption	Raw Bronze reprocessing

8.5 Observability & Monitoring

System Health Indicators

- Streaming lag
- Batch processing time
- Row throughput
- Error rates

Operational Dashboards

- Pipeline freshness
- Processing latency
- Alert volumes

8.6 Security & Compliance

Data Security

- No raw PII exposed in Gold
- Access restricted via role-based controls
- Sensitive fields masked or excluded

Compliance Readiness

- Full data lineage
- Deterministic transformations
- Explainable ML outputs

This design supports regulatory review and audit requirements.

8.7 Model Governance

Model Lifecycle Controls

- MLflow experiment tracking
- Versioned model registry
- Controlled model promotion

Retraining Strategy

- Periodic batch retraining
- Performance-based model updates
- Historical comparison using evaluation metrics

8.8 Change Management

Types of Changes

- Schema changes
- Feature updates
- Model upgrades
- Dashboard enhancements

Governance Approach

- Impact analysis before deployment
- Backward compatibility checks
- Documentation updates per change

8.9 Reprocessing & Backfill Strategy

Supported Scenarios

- Historical reprocessing
- Feature recalculation
- Model rescoring

Mechanism

- Checkpoint reset (controlled)
- Bronze-to-Gold replays
- Versioned outputs

8.10 Limitations & Known Constraints

- No direct transaction blocking
- No automated case management
- Synthetic streaming input
- Limited real-time labels

These constraints are acknowledged and documented.

8.11 Future Enhancements

Technical Enhancements

- Supervised fraud models
- Feature store integration
- Real-time alerting system
- Model drift detection

Business Enhancements

- Case management workflows
- Regulatory reporting
- Automated decisioning

8.12 Phase Completion Summary

PHASE 8 delivers:

- A production-aligned operational model
- Scalability and reliability guarantees
- Governance and compliance readiness
- A roadmap for future evolution

This phase ensures the system can **operate sustainably in a real enterprise environment**.
