AutoML FraudShield – V4: SRS + TDD + Deployment Manifest

0. Project Overview

- **Project Name:** AutoML FraudShield V4 (The "Omniscient Network" Upgrade)
- **Role:** Principal AI Architect
- **V3 Recap:** Delivered "Real-Time Velocity" using Dataflow for streaming aggregates (e.g., txn_count_10m) and a Hybrid Ensemble (XGBoost + Isolation Forest) for anomaly detection.[1111]
- **V4 Goal:** Detect organized fraud rings using **Graph Neural Networks (GNNs)** and close the loop with an automated **Active Learning System**. We also harden the multi-tenancy model using Row-Level Security (RLS).
- **Philosophy:** "Network Intelligence." We now leverage the relationships *between* entities to find fraud that looks normal in isolation but suspicious in a cluster.

Primary Architecture Changes (V3 $\to$ V4)

1. **Intelligence:** Hybrid Ensemble $\to$ **Ensemble + Graph Embeddings**.
2. **Training:** Tabular Only $\to$ **Tabular + Graph Topology**.
3. **Feedback:** Manual Retraining $\to$ **Automated Active Learning Loop**.
4. **Security:** Schema Tenancy $\to$ **Row-Level Security (RLS)**.

---

1. Scope & Requirements (SRS)

1.1 In-Scope (V4)

1. **Graph Neural Network (GNN):** Implement a nightly pipeline to construct a User-Device-IP graph and train a **GraphSAGE** model to generate user embeddings.[2222]
2. **Embedding Feature Store:** Materialize 64-dimensional user embeddings to the Vertex AI Feature Store (Online) for low-latency lookup during scoring.
3. **Active Feedback Loop:** A dedicated API endpoint (/v1/feedback) and BigQuery storage for human labels, with automated conflict validation.[3]
4. **Row-Level Security (RLS):** Enforce strict IAM policies in BigQuery so that analysts/models from Tenant A strictly cannot query rows belonging to Tenant B.

1.2 Out-of-Scope (Future V5)

- **Real-Time Graph Traversal:** We will not "walk" the graph during the API call (too slow). We rely on pre-computed embeddings.
- **Federated Learning:** Models are still centralized, though data is logically isolated.

1.3 Functional Requirements

1.3.1 Network Intelligence (FR-G)

- **FR-G1 – Graph Construction:** The system MUST construct a bipartite graph nightly:
  - **Nodes:** Customer, Terminal, IP_Address.
  - **Edges:** PERFORMED_TRANSACTION (weighted by recency).
- **FR-G2 – Inductive Training:** The pipeline MUST train a GraphSAGE model (inductive) to generate embeddings for unseen nodes without retraining the whole graph.[4]
- **FR-G3 – Vector Serving:** The scoring API MUST accept user_embedding (vector<float64>) as an input feature to the XGBoost model.

1.3.2 Active Learning (FR-L)

- **FR-L1 – Feedback Ingestion:** The system MUST accept label corrections via POST /v1/feedback.
  - Schema: { "transaction_id": str, "label": int, "analyst_id": str, "tenant_id": str }
- **FR-L2 – Label Validation:** A scheduled job MUST reject "Flip-Flop" labels (same ID labeled differently within 24h) before merging into the training set.[5]

1.3.3 Advanced Security (FR-S)

- **FR-S1 – Row-Level Security:** BigQuery access policies MUST filter all queries by tenant_id based on the authenticated service account's attributes.

1.4 Non-Functional Requirements (NFR)

- **NFR-1 – Latency (Scoring):** Graph embedding lookup MUST add $\le$ 20ms to the P95 latency (total $\le$ 320ms).
- **NFR-2 – Embedding Freshness:** Graph embeddings MUST be refreshed every 24 hours (Nightly Batch).

---

2. High-Level Architecture (System View)

2.1 Component Diagram

2.2 Data Flow Updates

1. **Nightly Graph Pipeline:**
   - Extract Txns $\to$ Build DGL/PyG Graph $\to$ Train GraphSAGE $\to$ Export Embeddings $\to$ Feature Store.
2. **Scoring Path (Updated):**
   - API fetches txn_count_10m (V3 Streaming) AND user_embedding (V4 Batch).
   - Ensemble Model scores using both velocity and network features.
3. **Feedback Path:**
   - Dashboard User clicks "Confirm Fraud" $\to$ API $\to$ feedback_labels Table.
   - Validation Job $\to$ training_data View (overrides raw label).

---

3. Detailed Design (TDD)

3.1 Graph Pipeline (graph/)

- **Graph Schema:**
  - **Heterogeneous Graph:**
    - Customer nodes (Features: Avg Amount, Age).
    - Device nodes (Features: Risk Score).
    - Edges: (Customer, USED, Device).
- **Model Architecture:**
  - **Algorithm:** GraphSAGE (chosen for scalability and inductive capability).
  - **Layers:** 2 Hop Neighbor Aggregation.
  - **Loss Function:** Link Prediction (predicting if Customer C will transact with Suspicious Device D).
- **Persistence:**
  - Embeddings are flattened (e.g., emb_0, emb_1, ... emb_63) and ingested into Vertex Feature Store under entity customer.

3.2 Active Learning Loop (feedback/)

Schema (fraudshield.feedback_labels):
SQL
transaction_id STRING,
correct_label INT64, -- 0 or 1
source STRING, -- e.g., "manual_review"
tenant_id STRING,
timestamp TIMESTAMP

-
- **Validation Logic:**
  - Query: Group by transaction_id. If COUNT(DISTINCT label) > 1, discard all updates for that ID and alert Operations.

Training View:
SQL
SELECT t.* EXCEPT(is_fraud),
COALESCE(f.correct_label, t.is_fraud) as label
FROM transactions t
LEFT JOIN (SELECT ... FROM feedback_labels WHERE validated=True) f
ON t.transaction_id = f.transaction_id

-

3.3 Security Implementation (RLS)

- **Mechanism:** BigQuery Row-Level Security Policies.

Policy Definition:
SQL
```
CREATE ROW ACCESS POLICY tenant_isolation
ON fraudshield.transactions
GRANT TO ("serviceAccount:tenant-a-sa@project.iam.gserviceaccount.com")
FILTER USING (tenant_id = 'tenant-A');
```
-

---

4. Deployment Manifest (DM)

4.1 Infrastructure Resources (Terraform)

- **Storage:**
  - gs://fraudshield-graph-data-{env} (Graph adjacency lists).
- **Compute:**
  - **Vertex AI Training Job:** Machine type n1-standard-4 with NVIDIA_TESLA_T4 (for GNN training).
- **BigQuery:**
  - feedback_labels table.
  - Row Access Policies (applied via Terraform or post-deployment script).

4.2 Repository Structure (V4 Extensions)

Plaintext
```
fraudshield-v4/
├── graph/                  # [NEW]
│   ├── build_graph.py      # BQ to DGL conversion
│   ├── model.py            # GraphSAGE PyTorch definition
│   ├── train.py            # Training loop
│   └── export.py           # Embedding -> Feature Store
├── feedback/               # [NEW]
│   ├── api.py              # Feedback Endpoint
│   └── validate_labels.sql # Conflict resolution query
├── infra/
│   └── terraform/
│       └── modules/
│           └── security/   # RLS Policy Definitions
└── models/
    └── ensemble_cpr/       # Updated to accept 64-dim embedding input
```

4.3 Deployment Strategy

1. **Data Backfill:** Run graph/build_graph.py on historical data to generate initial embeddings.
2. **Feature Store:** Register new feature entity user_embedding.

3. **Retrain:** Train the Hybrid Ensemble (XGB+Iso) with the new embedding features.
4. **Deploy:** Update the Endpoint with the V4 model.
5. **Security:** Apply RLS policies to BigQuery.

---

5. Summary

- **V3** gave us the **Speed** (Streaming).
- **V4** gives us the **Brain** (Graph) and the **Memory** (Feedback).
- This creates a complete "Omniscient" system: it sees what happens now (Stream), understands relationships (Graph), and learns from its mistakes (Feedback).