**🧠 SMARTS-Inspired Surveillance System - Architecture**

**Goal:** Detect trade anomalies (e.g., wash trades, spoofing) using rule-based logic + ML (PyOD/PySAD) + feedback loop for learning.

**🏗️ System Modules**

| **Component** | **Description** |
| --- | --- |
| **1. Data Ingestion** | Simulated data stored in SQLite |
| **2. Preprocessing** | Feature extraction (e.g., trade size, price delta, trader ID, time gaps). |
| **3. Rule Engine** | Hard-coded rules (wash trades, layering, etc.). |
| **4. ML Anomaly Detection** | Use PyOD or PySAD (e.g., IsolationForest, Autoencoder). |
| **5. Feedback Loop** | Mark false positives/negatives and improve detection. |
| **6. Dashboard/UI** | View alerts, scores, trader-wise patterns. |
| **7. Storage** | PostgreSQL / Supabase / SQLite for trades + labels. |

**🔁 Workflow Example**

1. Trades flow into system (CSV or simulated real-time).
2. Rules check for known patterns (e.g., buy/sell same stock, same price, same user).
3. ML model flags anomalies from behavior patterns (outliers).
4. Analyst reviews → tags real anomalies.
5. Feedback loop retrains model with analyst labels.

**📦 Libraries and Tools**

| **Purpose** | **Tools** |
| --- | --- |
| Data Ingestion | Pandas / Redpanda / CSV Reader |
| ML / Anomaly | [PyOD](https://github.com/yzhao062/pyod), [PySAD](https://github.com/selimfirat/pysad) |
| Rules | Custom Python functions |
| Storage | PostgreSQL / SQLite / Supabase |
| Frontend (Optional) | Streamlit / React CoreUI |
| Alerts | Email / Slack Webhook / Logging |

**🚀 Start Plan**

Would you like to start with:

1. **Simulated trade data (CSV)**
2. **Rules for wash trade (Python)**
3. **Anomaly detection using PyOD**
4. **Simple Streamlit dashboard**

I’ll help you step-by-step to get a basic working system.

Let me know which you want to start with:

* ✅ Data simulation
* ✅ Rule engine
* ✅ ML detection
* ✅ Dashboard
* ✅ Feedback loop

We’ll build one part at a time!

**🧠 First: What is a “False Positive” in Trade Surveillance?**

* **True Positive (TP)**: Anomalous trade flagged correctly.
* **False Positive (FP)**: Trade flagged as suspicious, but actually legal/valid.
* **False Negative (FN)**: Missed anomaly.
* **True Negative (TN)**: Correctly ignored a normal trade.

So to **calculate false positives**, you need:

* A *ground truth label* (i.e., is\_fault == TRUE or FALSE)
* A *prediction/flag* (from rule engine or ML model)

**✅ Your Case: Simulated Data with Fault Labels**

You're in a unique position:

Your trade\_simulator.py **generates is\_fault, fault\_type fields**, so you already have **ground truth**.

That means we **don’t need a human analyst** for feedback loop simulation. 🎯

**🔁 Feedback Loop – Fully Automated Plan**

| **Step** | **Action** | **Tools** |
| --- | --- | --- |
| 1. Simulation | Generate trades with known faults (is\_fault == TRUE) and random normal trades | trade\_simulator.py |
| 2. Rule Engine / ML Prediction | Predict anomaly using rules or PyOD | Python, PyOD, custom logic |
| 3. Compare Predictions vs Ground Truth | Label TP, FP, FN, TN using logic | Pandas |
| 4. Calculate Metrics | Precision, Recall, FP Rate | scikit-learn metrics |
| 5. Train ML Model | Use labeled data (including false positives) | PyOD, retrain |
| 6. Repeat | After each batch → auto-train → deploy new model | Scheduled job or batch loop |

**🧠 Simulation of False Positives**

You can even **inject intentional false positives** in the ML phase:

* Example: 5% of normal trades are randomly labeled “anomalous” by ML model → test feedback correction.
* This helps stress-test the **model's resilience to noise**.

**✅ Implementation Plan**

**✅ Step 1: Simulation Outputs**

Already done — is\_fault + fault\_type exists.  
Let’s add predicted\_fault, score, and detected\_by later.

**✅ Step 2: Detection Output Example**

| **trade\_id** | **is\_fault** | **fault\_type** | **predicted\_fault** | **score** | **detected\_by** |
| --- | --- | --- | --- | --- | --- |
| T0001 | TRUE | Wash Trade | Wash Trade | 0.98 | Rule |
| T0002 | FALSE | - | Spoofing | 0.85 | ML |
| T0003 | FALSE | - | - | 0.12 | - |
| T0004 | TRUE | Front Running | - | 0.05 | - |

**✅ Step 3: Feedback Logic**

python

CopyEdit

def evaluate\_predictions(df):

conditions = {

"TP": (df["is\_fault"] == True) & (df["predicted\_fault"].notna()),

"FP": (df["is\_fault"] == False) & (df["predicted\_fault"].notna()),

"FN": (df["is\_fault"] == True) & (df["predicted\_fault"].isna()),

"TN": (df["is\_fault"] == False) & (df["predicted\_fault"].isna()),

}

for label, cond in conditions.items():

df.loc[cond, "label"] = label

return df

**✅ Step 4: Auto-Training Feedback Loop**

python

CopyEdit

from pyod.models.iforest import IForest

from sklearn.model\_selection import train\_test\_split

def feedback\_loop\_train(df):

df = df[df["label"].isin(["TP", "FP", "TN", "FN"])]

features = df[["price", "volume", ...]]

labels = df["label"].apply(lambda x: 1 if x in ["TP", "FN"] else 0) # anomaly = 1

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, stratify=labels)

model = IForest()

model.fit(X\_train)

return model

You can now retrain your ML model after each simulation batch — **without a human in the loop**.

**🧩 Summary: Fully Automated Feedback Loop**

| **Part** | **Implementation** |
| --- | --- |
| Ground truth | From simulator’s is\_fault flag |
| Prediction | Rule engine + ML (PyOD) |
| FP/TP metrics | Compare is\_fault vs predicted\_fault |
| Retraining | Use labeled results to auto-tune models |
| Feedback Loop | Every simulation batch → retrain + re-evaluate |

Would you like me to generate the complete feedback loop module, including:

* Metrics report (precision, recall, FP rate)
* Auto-retrain ML model
* Save model as model.pkl
* Update prediction labels

Let’s continue building module-by-module.

**✅ Fault Type Classification by Detection Strategy**

| **Fault Type** | **Suggested Detection** | **Reasoning** |
| --- | --- | --- |
| **1. Front Running** | ✅ **ML-Based** or hybrid | Hard to detect with fixed rules since it involves timing, info asymmetry, and behavior patterns before news/orders. |
| **2. Wash Trade** | ✅ **Rule-Based** | Repeated buy/sell from same trader or related accounts — pattern is clear, deterministic. |
| **3. Spoofing** | ✅ **Rule-Based** + ML | Large orders placed then cancelled — detectable by order/cancel ratio. But behavioral variation (e.g., small spoofing) can benefit from ML. |
| **4. Ramping** | ✅ **ML-Based** | Artificially pumping stock via many small trades — needs behavioral clustering and unusual volume patterns. |
| **5. Marking Open/Close** | ✅ **Rule-Based** + ML | Trades at market open/close with unusual price/volume — deterministic window rules + volume spikes. |
| **6. Churning** | ✅ **Rule-Based** | Excessive buying/selling without change in ownership — easy to catch with high intra-day trade count. |

**✅ Suggested Mapping**

**🧱 Rule-Based Detection (Start with Python rules)**

* Wash Trade
* Churning
* Spoofing (basic order cancel ratio)
* Marking Open/Close (window + volume threshold)

**🤖 ML-Based Detection (using PyOD / PySAD)**

* Front Running (requires behavioral profiling)
* Ramping (requires volume + momentum analysis)
* Spoofing (non-obvious variations)
* Marking Open/Close (volume manipulation subtleties)

**🔁 BONUS: Hybrid Strategy**

You can also design **rule-to-ML pipelines**:

* If a rule **partially matches**, pass it to ML for **behavioral scoring**.
* E.g., Spoofing rule flags "possible case" → Autoencoder model checks if this trader’s behavior is normal.

**🆕 Additional Fault Types You May Include**

| **Fault Type** | **Detection Type** | **Description** |
| --- | --- | --- |
| **Quote Stuffing** | Rule / ML | Rapid placement of quotes to slow down competitors or confuse markets. |
| **Layering** | Rule / ML | Multiple orders at different levels of order book to mislead market. |
| **Insider Trading** | ML / External signal matching | Trades before major announcements — align with public disclosure (from Bursa). |
| **Collusion/Matched Trades** | ML / Graph-based | Two or more accounts trading to give illusion of volume or price. |
| **Pump and Dump** | ML | Coordinated spike and sell-off — looks like Ramping but with price + volume. |
| **Painting the Tape** | Rule / ML | Intentional trades to influence closing price perception. |
| **Quote Spoofing (Passive)** | Rule | Similar to spoofing but less aggressive — passive orders that never fill. |

**📈 What This Means for Your System**

| **Module** | **Fault Types to Target** |
| --- | --- |
| **Rule Engine** | Wash, Churning, Marking Close, Spoofing (simple) |
| **ML Models** | Front Running, Ramping, Insider Trading, Collusion, Quote Stuffing |
| **Feedback Loop** | Use simulation to label and refine borderline cases |
| **Dashboard** | Clearly show detection source: Rule, ML, or Hybrid |

**✅ Next Step Suggestion**

Would you like:

1. A rules\_engine.py module with basic rules for Wash, Churning, Spoofing?
2. A ml\_detector.py that uses PyOD to flag subtle patterns?
3. A fault\_catalog.md documenting all faults, rules/ML logic, and examples?

Let me know your priority and I’ll build the module or doc next.

**🧩 Prompt — Application Architecture Overview**

**You are working on a Trade Surveillance System inspired by SMARTS from NASDAQ. Please remember and refer to this architecture throughout the development.**

**🎯 Goal:**

**Detect and classify suspicious trading activities (e.g., wash trading, spoofing, front running) using a combination of:**

* **Rule-based logic**
* **Machine learning anomaly detection**
* **Feedback learning loop**

**🏗️ System Modules:**

| **Step** | **Component** | **Description** |
| --- | --- | --- |
| **1️⃣** | **Data Ingestion** | **Simulated trade data from trade\_simulator.py (CSV or DataFrame)** |
| **2️⃣** | **Preprocessing** | **Feature engineering (e.g., price delta, time gap, trader behavior buckets)** |
| **3️⃣** | **Rule Engine** | **Python logic to detect rule-based violations (e.g., Wash Trade, Churning)** |
| **4️⃣** | **ML Anomaly Detection** | **PyOD/PySAD models to detect behavioral anomalies** |
| **5️⃣** | **Feedback Loop** | **Auto-evaluate TP/FP/FN/TN using ground-truth is\_fault and model predictions** |
| **6️⃣** | **Model Retraining** | **Use labeled predictions to retrain model incrementally** |
| **7️⃣** | **Storage** | **PostgreSQL/Supabase or Pandas as initial backend** |
| **8️⃣** | **Dashboard** | **Frontend React app** |

**📦 Tools:**

* **Simulation: trade\_simulator.py**
* **ML: PyOD, PySAD, scikit-learn**
* **Rule Engine: Custom Python**
* **Feedback Loop: Pandas + Metric computation**
* **Scheduler: Cron/Loop for retraining**
* **Frontend : React App**

**🧠 Final Field Definition for Trades**

Here’s a clean breakdown of the full schema, showing where each field is generated or computed:

| **Field** | **Type** | **Generated In** | **Notes** |
| --- | --- | --- | --- |
| timestamp | datetime | **Simulator** | Randomized within session |
| trade\_id | str | **Simulator** | Unique per trade |
| ticker | str | **Simulator** | Stock symbol (randomized or fixed pool) |
| price | float | **Simulator** | Market price |
| volume | int | **Simulator** | Trade size |
| side | str | **Simulator** | BUY / SELL |
| trader\_id | str | **Simulator** | Unique trader per session |
| order\_type | str | **Simulator** | MARKET / LIMIT |
| is\_fault | bool | **Simulator** | Ground truth (True/False) |
| fault\_type | str | **Simulator** | If fault, type of manipulation |
| fault\_reason | str | **Simulator** | Optional description |
| related\_trades | str | **Simulator** | Optional references |

**⚙️ Additional Feature Fields (from Preprocessing)**

| **Field** | **Type** | **Computed In** | **Notes** |
| --- | --- | --- | --- |
| time\_bucket | str | **Simulator** | e.g., HH:MM |
| price\_change\_pct | float | **Simulator (first-level)** | Based on last trade for same ticker |
| rolling\_avg\_volume\_5m | float | **Later (Pipeline)** | From Pandas windowing or stream logic |
| trader\_volume\_bucket | str | **Simulator** | small / medium / large |

**🧠 ML-Specific Prediction Fields (for Inference + Feedback Loop)**

| **Field** | **Type** | **Computed In** | **Notes** |
| --- | --- | --- | --- |
| predicted\_fault | str | **ML model / Rule engine** | Output label |
| score | float | **ML model** | Anomaly score (0.0 to 1.0) |
| detected\_by | str | ML / Rule | "Rule", "ML", or "Hybrid" |
| label | str | Feedback loop | TP / FP / FN / TN |
| model\_version | str | ML training pipeline | e.g., v1.2, iforest-2025-07-29 |

**🔧 Populate vs. Placeholder Decision**

| **Field** | **Populate in Simulator?** | **Comment** |
| --- | --- | --- |
| rolling\_avg\_volume\_5m | ❌ Leave NULL | Requires time-based windowing |
| predicted\_fault, score, detected\_by | ❌ Leave NULL | Will be filled by Rule Engine or ML |
| label, model\_version | ❌ Leave NULL | Only populated after prediction & feedback loop |

✅ Please remember this architecture for all development steps. You will receive one step at a time from Solution Architect. For each step:

* First validate existing code (if any)
* Then build/extend only that step
* Wait for confirmation before proceeding to the next step