# Fraud Detection System - Technical Specification

## System Architecture

### Overview

The fraud detection system is built as a modular, scalable architecture with the following components:

1. **Core ML Engine** (BankFraudDetector)
2. **Web Dashboard** (Streamlit)
3. **API Server** (FastAPI)
4. **Multi-Agent Pipeline** (LangGraph)

### Component Interaction

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ Streamlit │ │ FastAPI │ │ Multi-Agent │  
│ Dashboard │◄──►│ Server │◄──►│ Pipeline │  
└─────────────────┘ └─────────────────┘ └─────────────────┘  
 │ │ │  
 └───────────────────────┼───────────────────────┘  
 │  
 ┌─────────────────┐  
 │ Bank Fraud │  
 │ Detector │  
 │ (Core ML) │  
 └─────────────────┘

## Data Flow

### 1. Transaction Input

transaction\_data = {  
 "transaction\_id": "TXN\_001",  
 "amount": 150.00,  
 "customer\_id": "CUST\_001",  
 "transaction\_type": "ONLINE",  
 "hour": 14,  
 "day\_of\_week": 2,  
 "location": "DOMESTIC",  
 "device\_type": "MOBILE",  
 "card\_present": False  
}

### 2. Feature Engineering Pipeline

def engineer\_bank\_features(df):  
 # Time-based features  
 df['is\_weekend'] = df['day\_of\_week'].isin([5, 6]).astype(int)  
 df['is\_night'] = ((df['hour'] >= 22) | (df['hour'] <= 6)).astype(int)  
 df['is\_business\_hours'] = ((df['hour'] >= 9) & (df['hour'] <= 17)).astype(int)  
   
 # Amount-based features  
 df['amount\_log'] = np.log1p(df['amount'])  
 df['is\_high\_value'] = (df['amount'] > df['amount'].quantile(0.95)).astype(int)  
   
 # Transaction features  
 df['is\_online'] = (df['transaction\_type'] == 'ONLINE').astype(int)  
 df['is\_international'] = (df['location'] == 'INTERNATIONAL').astype(int)  
 df['card\_not\_present'] = (~df['card\_present']).astype(int)  
   
 return df

### 3. Model Prediction

def predict\_transaction\_risk(transaction\_data):  
 # Feature preparation  
 features = prepare\_features(transaction\_data)  
   
 # Model ensemble prediction  
 predictions = {}  
 for model\_name, model in self.models.items():  
 pred = model.predict(features)  
 prob = model.predict\_proba(features)[0, 1]  
 predictions[model\_name] = {'prediction': pred, 'probability': prob}  
   
 # Risk level determination  
 risk\_level = determine\_risk\_level(predictions)  
   
 return {  
 'risk\_level': risk\_level,  
 'risk\_probability': max\_prob,  
 'model\_predictions': predictions,  
 'recommended\_action': get\_action(risk\_level)  
 }

## Machine Learning Models

### 1. Logistic Regression

* **Purpose**: Baseline model with interpretability
* **Advantages**: Fast training, interpretable coefficients
* **Use Case**: Initial fraud screening

### 2. Random Forest

* **Purpose**: Robust ensemble method
* **Advantages**: Handles non-linear relationships, feature importance
* **Use Case**: Primary fraud detection model

### 3. Isolation Forest

* **Purpose**: Anomaly detection
* **Advantages**: Detects outliers without labeled fraud data
* **Use Case**: Unsupervised fraud detection

### Model Training Process

def train\_bank\_models(df, test\_size=0.2):  
 # Feature preparation  
 X = df[feature\_columns]  
 y = df[fraud\_column]  
   
 # Data splitting  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X, y, test\_size=test\_size, random\_state=42, stratify=y  
 )  
   
 # Feature scaling  
 scaler = RobustScaler()  
 X\_train\_scaled = scaler.fit\_transform(X\_train)  
 X\_test\_scaled = scaler.transform(X\_test)  
   
 # Model training  
 models = {  
 'Logistic Regression': LogisticRegression(random\_state=42),  
 'Random Forest': RandomForestClassifier(n\_estimators=100, random\_state=42),  
 'Isolation Forest': IsolationForest(random\_state=42, contamination=y\_train.mean())  
 }  
   
 # Training and evaluation  
 for name, model in models.items():  
 model.fit(X\_train\_scaled, y\_train)  
 y\_pred = model.predict(X\_test\_scaled)  
 auc\_score = roc\_auc\_score(y\_test, y\_pred)  
   
 return results, X\_test\_scaled, y\_test

## Feature Engineering Specification

### Time-Based Features

| Feature | Description | Calculation |
| --- | --- | --- |
| is\_weekend | Weekend transaction flag | day\_of\_week in [5, 6] |
| is\_night | Night-time transaction flag | hour >= 22 or hour <= 6 |
| is\_business\_hours | Business hours flag | 9 <= hour <= 17 |

### Amount-Based Features

| Feature | Description | Calculation |
| --- | --- | --- |
| amount\_log | Log-transformed amount | log(1 + amount) |
| is\_high\_value | High-value transaction flag | amount > 95th percentile |
| amount\_percentile | Amount percentile rank | rank(amount) / total\_count |

### Transaction Features

| Feature | Description | Calculation |
| --- | --- | --- |
| is\_online | Online transaction flag | transaction\_type == 'ONLINE' |
| is\_atm | ATM transaction flag | transaction\_type == 'ATM' |
| is\_international | International transaction flag | location == 'INTERNATIONAL' |
| card\_not\_present | Card-not-present flag | not card\_present |

### Customer Behavior Features

| Feature | Description | Calculation |
| --- | --- | --- |
| avg\_amount | Customer average transaction amount | groupby(customer\_id)['amount'].mean() |
| std\_amount | Customer transaction amount std | groupby(customer\_id)['amount'].std() |
| transaction\_count | Customer transaction count | groupby(customer\_id)['amount'].count() |
| fraud\_rate | Customer historical fraud rate | groupby(customer\_id)['is\_fraud'].mean() |

## Risk Assessment Framework

### Risk Levels

1. **SAFE** (0-30th percentile)
   * Action: Allow transaction
   * Monitoring: Standard
2. **LOW\_RISK** (30-70th percentile)
   * Action: Monitor closely
   * Monitoring: Enhanced
3. **MEDIUM\_RISK** (70-85th percentile)
   * Action: Require additional verification
   * Monitoring: High
4. **HIGH\_RISK** (85-100th percentile)
   * Action: Block transaction
   * Monitoring: Immediate

### Risk Score Calculation

def calculate\_risk\_score(transaction):  
 risk\_score = (  
 transaction['is\_high\_value'] \* 2 +  
 transaction['is\_international'] \* 3 +  
 transaction['card\_not\_present'] \* 2 +  
 transaction['is\_night'] \* 1 +  
 transaction['previous\_fraud\_flag'] \* 5  
 )  
 return risk\_score

## API Specification

### Endpoints

#### 1. Health Check

GET /health

**Response:**

{  
 "status": "healthy",  
 "models\_loaded": 3,  
 "last\_training": "2024-12-19T10:30:00Z",  
 "version": "1.0.0"  
}

#### 2. Predict Transaction

POST /predict  
Content-Type: application/json

**Request Body:**

{  
 "transaction\_id": "TXN\_001",  
 "amount": 150.00,  
 "customer\_id": "CUST\_001",  
 "transaction\_type": "ONLINE",  
 "hour": 14,  
 "day\_of\_week": 2,  
 "location": "DOMESTIC",  
 "device\_type": "MOBILE",  
 "card\_present": false  
}

**Response:**

{  
 "transaction\_id": "TXN\_001",  
 "risk\_level": "LOW\_RISK",  
 "risk\_probability": 0.25,  
 "recommended\_action": "MONITOR\_CLOSELY",  
 "model\_predictions": {  
 "Logistic Regression": {"prediction": 0, "probability": 0.20},  
 "Random Forest": {"prediction": 0, "probability": 0.25},  
 "Isolation Forest": {"prediction": 0, "probability": 0.30}  
 }  
}

#### 3. Model Information

GET /model-info

**Response:**

{  
 "models": ["Logistic Regression", "Random Forest", "Isolation Forest"],  
 "performance\_metrics": {  
 "auc\_scores": {  
 "Logistic Regression": 0.85,  
 "Random Forest": 0.92,  
 "Isolation Forest": 0.78  
 }  
 },  
 "feature\_count": 17,  
 "training\_date": "2024-12-19T10:30:00Z"  
}

## Dashboard Specification

### Layout Structure

┌─────────────────────────────────────────────────────────────┐  
│ Header (Title + Status) │  
├─────────────────────────────────────────────────────────────┤  
│ Metrics Row: AUC, Precision, Recall, F1-Score │  
├─────────────────────────────────────────────────────────────┤  
│ Main Content Area │  
│ ┌─────────────────┐ ┌─────────────────┐ │  
│ │ Transaction │ │ Performance │ │  
│ │ Feed │ │ Charts │ │  
│ └─────────────────┘ └─────────────────┘ │  
├─────────────────────────────────────────────────────────────┤  
│ Alerts & Logs Section │  
│ ┌─────────────────┐ ┌─────────────────┐ │  
│ │ Risk Alerts │ │ Transaction │ │  
│ │ (Cards) │ │ Logs │ │  
│ └─────────────────┘ └─────────────────┘ │  
└─────────────────────────────────────────────────────────────┘

### Components

#### 1. Metrics Dashboard

* **AUC Score**: Model discrimination ability
* **Precision**: Accuracy of positive predictions
* **Recall**: Sensitivity to fraud detection
* **F1-Score**: Harmonic mean of precision and recall

#### 2. Transaction Feed

* Real-time transaction display
* Risk level indicators
* Transaction details on hover

#### 3. Performance Charts

* ROC curves for all models
* Confusion matrices
* Feature importance plots

#### 4. Alert System

* Color-coded risk alerts
* High risk: Red cards
* Medium risk: Orange cards
* Low risk: Yellow cards

#### 5. Transaction Logs

* Detailed transaction history
* Timestamp and risk information
* Search and filter capabilities

## Performance Requirements

### Response Times

* **Dashboard Load**: < 3 seconds
* **Transaction Prediction**: < 500ms
* **API Response**: < 200ms
* **Model Training**: < 5 minutes (for 5K transactions)

### Scalability

* **Concurrent Users**: 10+ dashboard users
* **Transaction Throughput**: 100+ transactions/minute
* **Data Volume**: 10K+ transactions in memory

### Accuracy Targets

* **AUC Score**: > 0.85
* **Precision**: > 0.80 (at 0.90 recall)
* **Recall**: > 0.90 (for fraud detection)
* **False Positive Rate**: < 0.10

## Security Requirements

### Data Protection

* No PII storage in models
* Transaction data anonymization
* Secure model serialization

### Access Control

* Local deployment for sensitive data
* Network-level security for shared access
* API authentication for production

### Input Validation

* Transaction data validation
* Feature range checking
* Error handling and logging

## Deployment Specifications

### Local Development

# Requirements  
Python 3.8+  
8GB RAM  
2GB disk space  
  
# Dependencies  
pip install -r requirements.txt  
  
# Startup  
python3 -m streamlit run src/dashboard.py --server.port 8501

### Production Deployment

# Docker  
docker build -t fraud-detection-dashboard .  
docker run -p 8501:8501 fraud-detection-dashboard  
  
# Streamlit Cloud  
# Connect GitHub repository  
# Deploy with main file: src/dashboard.py

### Monitoring

* Application health checks
* Model performance monitoring
* Error logging and alerting
* Resource usage tracking

**Document Version**: 1.0.0 **Last Updated**: December 2024 **Author**: AI Assistant