

**DẠI HỌC QUỐC GIA HÀ NỘI  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ  
VIỆN TRÍ TUỆ NHÂN TẠO**



**BÁO CÁO MÔN HỌC**

**ĐỀ TÀI:**

**RỦI RO THỊ TRƯỜNG VÀ PHÁT HIỆN GIAN  
LẬN TÀI CHÍNH TẠI HSBC**

**Sinh viên thực hiện:**

Nguyễn Văn Linh – 23020395

Hoàng Ngọc Nam – 23020403

Trần Quốc Khánh – 23020387

**Giảng viên hướng dẫn:**

**TS. Trần Hồng Việt**

**ThS. Ngô Minh Hương**

Hà Nội, tháng 11 năm 2025

# Mục lục

<b>Danh Mục Hình Ảnh</b>	<b>3</b>
<b>Danh Mục Bảng</b>	<b>4</b>
<b>1 TỔNG QUAN VÀ BỐI CẢNH DỰ ÁN</b>	<b>5</b>
1.1 Bối Cảnh và Động Lực Nghiên Cứu . . . . .	5
1.2 Vai Trò Của Dữ Liệu Lớn Và Học Máy Trong Phát Hiện Gian Lận . . . . .	5
1.2.1 Đặc Trưng Của Dữ Liệu Giao Dịch Tài Chính . . . . .	6
1.2.2 Lựa Chọn XGBoost Cho Phát Hiện Gian Lận . . . . .	6
1.3 HSBC và Xu Hướng Ứng Dụng AI Trong Phát Hiện Gian Lận Tài Chính . . . . .	6
1.3.1 Dynamic Risk Assessment: Hệ Thống Phát Hiện Gian Lận Thủ Hộ Môi . . . . .	7
1.3.2 Risk Advisory Tool: Mô Phỏng Rủi Ro Thời Gian Thực . . . . .	7
1.3.3 Ý Nghĩa Đối Với Nghiên Cứu Nay . . . . .	8
1.3.4 Khoảng Cách Nghiên Cứu và Đóng Góp . . . . .	8
<b>2 KIẾN TRÚC HỆ THỐNG VÀ CÔNG NGHỆ NỀN TẢNG</b>	<b>9</b>
2.1 Tổng Quan Kiến Trúc Kappa . . . . .	10
2.2 Cơ Sở Lý Thuyết Các Công Nghệ Nền Tảng . . . . .	10
2.2.1 Apache Kafka: Nền Tảng Xử Lý Luồng Phân Tán . . . . .	11
2.2.2 Apache Spark: Kiến Trúc Xử Lý Tính Toán Tại Bộ Nhớ . . . . .	12
2.2.3 Apache Spark Structured Streaming . . . . .	13
2.2.4 MinIO: Lưu Trữ Đối Tượng Hiệu Năng Cao . . . . .	13
2.2.5 Apache Cassandra: Cơ Sở Dữ Liệu Phân Tán NoSQL . . . . .	14
2.2.6 Docker và Docker Compose . . . . .	15
<b>3 QUY TRÌNH XỬ LÝ DỮ LIỆU VÀ KỸ THUẬT ĐẶC TRƯNG</b>	<b>16</b>
3.1 Cơ Chế Sinh Dữ Liệu Giả Lập . . . . .	16
3.1.1 Kiến Trúc Tổng Thể . . . . .	16
3.1.2 Phân Hệ 1: Sinh Dữ Liệu Khách Hàng . . . . .	17
3.1.3 Phân Hệ 2: Sinh Dữ Liệu Giao Dịch . . . . .	18
3.1.4 Phân Hệ 3: Tiêm Nhiễm Gian Lận . . . . .	19
3.1.5 Tối Ưu Hiệu Năng Với Kiến Trúc Song Song . . . . .	20
3.2 Kiến Trúc Hệ Thống và Cơ Chế Xử Lý Luồng Dữ Liệu Phân Tán . . . . .	20
3.2.1 Tầng 1: Tiếp Nhận và Đệm Dữ Liệu . . . . .	21
3.2.2 Tầng 2: Lõi Xử Lý Phân Tích . . . . .	22

3.2.3	Tầng 3: Phân Tách và Lưu Trữ Da Hướng . . . . .	24
3.2.4	Dảm Bảo Exactly-Once Semantics . . . . .	25
3.3	Kỹ Nghệ Đặc Trưng và Mô Hình Hóa . . . . .	25
3.3.1	Quy Trình Kỹ Nghệ Đặc Trưng . . . . .	25
3.3.2	So Sánh Thuật Toán: Random Forest và XGBoost . . . . .	27
3.3.3	XGBoost - Lựa Chọn Triển Khai Chính Thức . . . . .	28
3.3.4	Dánh Giá Hiệu Năng . . . . .	30
3.4	Dánh Giá Hiệu Năng Hệ Thống . . . . .	31
3.4.1	Độ Trễ Xử Lý . . . . .	31
3.4.2	Hiệu Năng Mô Hình . . . . .	31
3.4.3	Bảng Điều Khiển . . . . .	31
<b>4</b>	<b>KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN</b>	<b>33</b>
4.1	Tổng Kết Nghiên Cứu . . . . .	33
4.1.1	Các Đóng Góp Chính . . . . .	33
4.1.2	Hạn Chế . . . . .	34
4.2	Hướng Phát Triển . . . . .	34
4.2.1	Tích Hợp Cơ Sở Dữ Liệu Đồ Thị . . . . .	34
4.2.2	Học Sâu Cho Chuỗi Giao Dịch . . . . .	34
4.2.3	Pipeline Vận Hành Học Máy . . . . .	35
4.2.4	Mở Rộng Sang Phân Tích Rủi Ro Thị Trường . . . . .	35
4.2.5	Kỹ Thuật Gian Lận Nâng Cao . . . . .	35
4.3	Lời Kết . . . . .	35

# Danh sách hình vẽ

2.1	Sơ đồ các thành phần kiến trúc cốt lõi của hệ thống phát hiện gian lận . . . . .	11
2.2	Kiến trúc tổng quan của Apache Kafka Cluster . . . . .	12
2.3	Kiến trúc Master-Slave của Apache Spark . . . . .	12
2.4	Kiến trúc lưu trữ phân tán của MinIO . . . . .	14
2.5	Kiến trúc vòng của Apache Cassandra . . . . .	14
3.1	Kiến trúc tổng thể hệ thống sinh dữ liệu giả lập với ba phân hệ chính: Sinh khách hàng, Sinh giao dịch, và Tiêm nhiễm gian lận . . . . .	16
3.2	Luồng hoạt động tổng thể của hệ thống phát hiện gian lận thời gian thực .	21
3.3	Giao diện tổng quan hệ thống . . . . .	32
3.4	Các biểu đồ phân tích trực quan . . . . .	32

# Danh sách bảng

2.1	So sánh kiến trúc Lambda và Kappa . . . . .	9
2.2	Các thành phần kiến trúc Kappa trong hệ thống này . . . . .	10
3.1	Cấu hình siêu tham số XGBoost sau grid search . . . . .	29
3.2	Confusion matrix của XGBoost trên test set . . . . .	30

# Chương 1

## TỔNG QUAN VÀ BỐI CẢNH DỰ ÁN

### 1.1 Bối Cảnh và Động Lực Nghiên Cứu

Trong kỷ nguyên chuyển đổi số, các tổ chức tài chính đang đối mặt với thách thức kép: một mặt là sự gia tăng theo cấp số nhân của khối lượng giao dịch trực tuyến, mặt khác là sự tinh vi ngày càng cao của các thủ đoạn gian lận tài chính. Theo báo cáo của Nilson Report năm 2023, tổn thất toàn cầu do gian lận thẻ thanh toán ước tính đạt 32 tỷ USD, và con số này dự báo sẽ tăng lên 49 tỷ USD vào năm 2030.

Các phương pháp phát hiện gian lận truyền thống thường dựa trên hệ thống quy tắc tĩnh, vận hành theo các kịch bản đã biết trước. Ví dụ: nếu giao dịch vượt 10.000 USD và thực hiện ở nước ngoài thì đánh dấu cảnh báo. Mặc dù đơn giản để triển khai, cách tiếp cận này tạo ra tỷ lệ dương tính giả lên đến 95% trong một số trường hợp, dẫn đến lãng phí nguồn lực điều tra và gây phiền hà cho khách hàng hợp pháp.

Nghiên cứu này phát triển một hệ thống phát hiện gian lận thời gian thực dựa trên kiến trúc xử lý luồng dữ liệu, kết hợp Apache Spark Structured Streaming, Apache Kafka và mô hình học máy XGBoost. Hệ thống được thiết kế để xử lý hàng nghìn giao dịch mỗi giây với độ trễ dưới 100 mili-giây, đồng thời đạt độ chính xác phát hiện gian lận trên 99%.

### 1.2 Vai Trò Của Dữ Liệu Lớn Và Học Máy Trong Phát Hiện Gian Lận

Để giải quyết các hạn chế của hệ thống truyền thống, việc ứng dụng công nghệ Big Data và Machine Learning đã trở thành yêu cầu thiết yếu trong ngành tài chính hiện đại. Các nghiên cứu gần đây cho thấy mô hình XGBoost đạt hiệu suất vượt trội trong phát hiện gian lận với độ chính xác lên đến 99.98%, phù hợp với bối cảnh dữ liệu mất cân bằng nghiêm trọng khi tỷ lệ gian lận thường chỉ 0.1-1%.

### 1.2.1 Đặc Trưng Của Dữ Liệu Giao Dịch Tài Chính

Dữ liệu lớn trong bối cảnh ngân hàng được đặc trưng bởi ba yếu tố chính:

- **Khối lượng lớn:** Hệ thống cần lưu trữ và truy xuất hàng triệu giao dịch lịch sử để huấn luyện mô hình học máy. Tập dữ liệu huấn luyện trong nghiên cứu này bao gồm 1.296.675 giao dịch với tỷ lệ gian lận 0.58%, phản ánh đúng tình trạng mất cân bằng dữ liệu trong thực tế.
- **Tốc độ xử lý cao:** Khả năng xử lý luồng dữ liệu ngay khi được sinh ra là yếu tố quyết định. Hệ thống được thiết kế với khả năng xử lý 12 giao dịch/giây trong môi trường demo và có thể mở rộng lên hàng nghìn giao dịch/giây trong sản xuất thực tế.
- **Tính đa dạng:** Dữ liệu giao dịch bao gồm 23 trường thông tin từ nhiều nguồn khác nhau: thông tin khách hàng như tuổi, giới tính, nghề nghiệp; thông tin giao dịch như số tiền, thời gian, danh mục; và thông tin địa lý như tọa độ GPS khách hàng và merchant. Qua kỹ nghệ đặc trưng, 23 trường này được mở rộng thành 44 trường để tối ưu hóa khả năng phát hiện gian lận.

### 1.2.2 Lựa Chọn XGBoost Cho Phát Hiện Gian Lận

Nghiên cứu lựa chọn XGBoost làm mô hình chính thức dựa trên các ưu điểm đã được chứng minh trong nhiều công trình nghiên cứu về phát hiện gian lận ngân hàng:

- **Xử lý dữ liệu mất cân bằng:** Với tham số `scale_pos_weight`, XGBoost tự động cân bằng trọng số giữa lớp gian lận ở mức 0.58% và lớp bình thường ở mức 99.42%, giải quyết vấn đề cốt lõi của dữ liệu tài chính.
- **Hiệu năng cao:** Trong nghiên cứu này, XGBoost đạt AUC-ROC 0.9964 và Recall 99%, vượt trội hơn Random Forest với AUC 0.976. Các nghiên cứu độc lập cũng cho kết quả tương tự với độ chính xác 99.98%.
- **Tốc độ suy diễn thời gian thực:** Với cấu trúc tính toán song song, XGBoost đạt độ trễ suy diễn trung bình 10ms/giao dịch, đáp ứng yêu cầu phát hiện gian lận ngay tức thì.

## 1.3 HSBC và Xu Hướng Ứng Dụng AI Trong Phát Hiện Gian Lận Tài Chính

HSBC, một trong những tập đoàn tài chính lớn nhất thế giới với hoạt động tại hơn 60 quốc gia và phục vụ 40 triệu khách hàng, đang đổi mới với thách thức sàng lọc hơn 1,35 tỷ giao dịch mỗi tháng để phát hiện các dấu hiệu tội phạm tài chính. Quy mô khổng lồ này khiến các phương pháp phát hiện gian lận truyền thống dựa trên quy tắc tĩnh trở nên không khả thi, do tỷ lệ cảnh báo sai lên đến 95% và thời gian xử lý kéo dài hàng tuần.

Dể giải quyết vấn đề này, HSBC đã chuyển dịch sang ứng dụng trí tuệ nhân tạo và điện toán đám mây, tạo nên các giải pháp công nghệ tiên phong trong ngành ngân hàng. Phân tích các sáng kiến công nghệ cốt lõi của HSBC cung cấp nền tảng lý thuyết và thực tiễn cho việc thiết kế kiến trúc hệ thống trong nghiên cứu này.

### 1.3.1 Dynamic Risk Assessment: Hệ Thống Phát Hiện Gian Lận Thế Hệ Mới

Hệ thống Dynamic Risk Assessment là sáng kiến trọng yếu nhất của HSBC trong lĩnh vực phòng chống tội phạm tài chính, được đồng phát triển với Google Cloud từ năm 2021.

#### Kiến Trúc và Công Nghệ

DRA được xây dựng trên nền tảng Google Cloud AML AI, kết hợp các công nghệ học máy tiên tiến với kiến thức chuyên môn về tuân thủ tài chính của HSBC. Hệ thống phân tích các mẫu giao dịch, hành vi mạng lưới và dữ liệu KYC để tạo ra điểm rủi ro động, thích ứng theo thời gian và cung cấp đầu ra có khả năng giải thích để hỗ trợ quyết định tuân thủ.

Kiến trúc cloud-native của DRA sử dụng Kubernetes để quản lý hạ tầng phức tạp, giao diện điều tra được xây dựng bằng React.js và được cung cấp dữ liệu qua nhiều DRA APIs dưới dạng microservices. Thiết kế này giảm đáng kể chi phí và độ phức tạp trong quản lý dự án hạ tầng quy mô lớn.

#### Kết Quả Triển Khai

Các chỉ số hiệu năng của DRA cho thấy bước tiến vượt bậc so với hệ thống truyền thống:

- **Nâng cao khả năng phát hiện:** Hệ thống AI phát hiện được từ 2 đến 4 lần số lượng hoạt động tội phạm tài chính so với hệ thống cũ dựa trên quy tắc, đồng thời giảm 60% số lượng cảnh báo sai.
- **Tăng tốc xử lý:** Thời gian chờ đợi cho phân tích hồi cứu 24 tháng tại thị trường Anh giảm từ khoảng một tháng xuống còn vài ngày. Thời gian phát hiện tài khoản nghi ngờ giảm xuống còn 8 ngày sau cảnh báo đầu tiên.
- **Cải thiện hiệu quả điều tra:** Trong mảng ngân hàng thương mại, số lượng tội phạm tài chính được xác định tăng gấp đôi, trong khi mảng ngân hàng bán lẻ đạt mức tăng gần gấp bốn lần.

### 1.3.2 Risk Advisory Tool: Mô Phỏng Rủi Ro Thời Gian Thực

Bên cạnh phát hiện gian lận, HSBC cũng triển khai các giải pháp cloud cho quản lý rủi ro tín dụng. Risk Advisory Tool được phát triển trên Google Cloud Platform, sử dụng Cloud Storage, Dataflow và BigQuery để thực hiện mô phỏng kịch bản rủi ro.

### Hiệu Năng Vượt Trội

Hệ thống mới cho phép chạy đồng thời vô số mô phỏng "what-if", nhanh gấp 16 lần so với giải pháp on-premise trước đây. Các nhà giao dịch có thể nhận kết quả trong vòng 15 phút thay vì hàng giờ, giúp quản lý danh mục đầu tư tốt hơn trong phiên giao dịch và đánh giá yêu cầu vốn để đối phó với rủi ro hạ bậc xếp hạng và vỡ nợ tiềm ẩn.

### 1.3.3 Ý Nghĩa Đối Với Nghiên Cứu Nay

Các sáng kiến công nghệ của HSBC cung cấp ba bài học quan trọng cho thiết kế hệ thống phát hiện gian lận trong nghiên cứu này:

- Ưu tiên mô hình học máy thay vì quy tắc tĩnh:** Thành công của DRA chứng minh rằng các thuật toán AI như XGBoost có khả năng phát hiện gian lận vượt trội, giảm đáng kể cảnh báo sai và tăng độ chính xác. Đây là cơ sở cho việc nghiên cứu này lựa chọn XGBoost làm mô hình chính thức với mục tiêu đạt AUC-ROC trên 0.99.
- Kiến trúc xử lý luồng thời gian thực:** Yêu cầu xử lý 1,35 tỷ giao dịch mỗi tháng của HSBC (tương đương khoảng 520 giao dịch/giây) đòi hỏi kiến trúc streaming mạnh mẽ. Nghiên cứu này áp dụng Apache Kafka và Spark Structured Streaming để đạt khả năng xử lý tương tự.
- Tích hợp công nghệ mã nguồn mở với đám mây:** HSBC sử dụng Kubernetes, Dataflow (Apache Beam), BigQuery kết hợp với Google Cloud. Nghiên cứu này triển khai các công nghệ mã nguồn mở tương đương như Apache Spark, Kafka, Cassandra và MinIO để tạo hệ thống có thể tái sử dụng và mở rộng.

### 1.3.4 Khoảng Cách Nghiên Cứu và Đóng Góp

Mặc dù DRA của HSBC đạt kết quả ấn tượng, hệ thống là giải pháp độc quyền với chi phí triển khai cao và không công khai mã nguồn. Nghiên cứu này đóng góp bằng cách:

- Xây dựng một hệ thống phát hiện gian lận hoàn chỉnh dựa trên công nghệ mã nguồn mở, có thể triển khai với chi phí thấp hơn và tùy biến linh hoạt.
- Cung cấp phân tích chi tiết về quy trình kỹ nghệ đặc trưng với 21 đặc trưng được thiết kế dựa trên kiến thức chuyên môn và dữ liệu thực nghiệm, trong khi DRA không công khai chi tiết kỹ thuật này.
- So sánh định lượng hiệu năng giữa XGBoost và Random Forest trong bối cảnh dữ liệu mất cân bằng, cung cấp cơ sở khoa học cho việc lựa chọn thuật toán phù hợp.

Chương tiếp theo sẽ trình bày chi tiết kiến trúc hệ thống được thiết kế dựa trên các nguyên lý rút ra từ phân tích các sáng kiến công nghệ của HSBC.

## Chương 2

# KIẾN TRÚC HỆ THỐNG VÀ CÔNG NGHỆ NỀN TẢNG

Xuất phát từ yêu cầu về xử lý thời gian thực với độ trễ thấp và đảm bảo tính nhất quán của dữ liệu, nhóm lựa chọn kiến trúc Kappa làm nền tảng triển khai cho hệ thống phát hiện gian lận. Thay vì tách riêng hai luồng xử lý batch và streaming như kiến trúc Lambda, Kappa thống nhất toàn bộ quá trình xử lý trên một pipeline streaming duy nhất, trong đó dữ liệu được coi là dòng liên tục ngay từ khi phát sinh và chỉ được phân tách ở tầng lưu trữ, qua đó giảm đáng kể độ phức tạp hệ thống, hạn chế trùng lắp logic xử lý và nâng cao khả năng bảo trì, mở rộng trong môi trường thực tế.

Bảng 2.1: So sánh kiến trúc Lambda và Kappa

Tiêu chí	Lambda Architecture	Kappa Architecture
Mô hình xử lý	Kết hợp hai pipeline riêng biệt: Batch và Streaming	Chỉ sử dụng một pipeline Streaming duy nhất
Độ phức tạp kiến trúc	Cao, do phải duy trì song song hai hệ thống xử lý	Thấp hơn, kiến trúc gọn nhẹ và thống nhất
Chi phí vận hành	Lớn, tốn tài nguyên để duy trì hai luồng xử lý	Thấp hơn do giảm số lượng thành phần hệ thống
Tính nhất quán logic	Có nguy cơ lệch logic giữa Batch và Streaming	Dảm bảo tốt hơn nhờ chỉ có một pipeline
Khả năng mở rộng	Mở rộng tốt nhưng phức tạp	Dễ mở rộng và quản lý hơn
Xử lý dữ liệu lịch sử	Thực hiện qua Batch layer	Replay lại dữ liệu từ event log hoặc message broker
Độ trễ xử lý	Batch có độ trễ cao, Streaming có độ trễ thấp	Duy trì độ trễ thấp cho cả dữ liệu mới và dữ liệu cũ
Tính phù hợp	Phù hợp với hệ thống vừa phân tích lịch sử lớn vừa realtime	Phù hợp với hệ thống xử lý streaming làm trọng tâm
Ví dụ ứng dụng	Recommendation system, log analytics, hệ thống phân tích dữ liệu lớn	Fraud detection, IoT streaming, giám sát thời gian thực

## 2.1 Tổng Quan Kiến Trúc Kappa

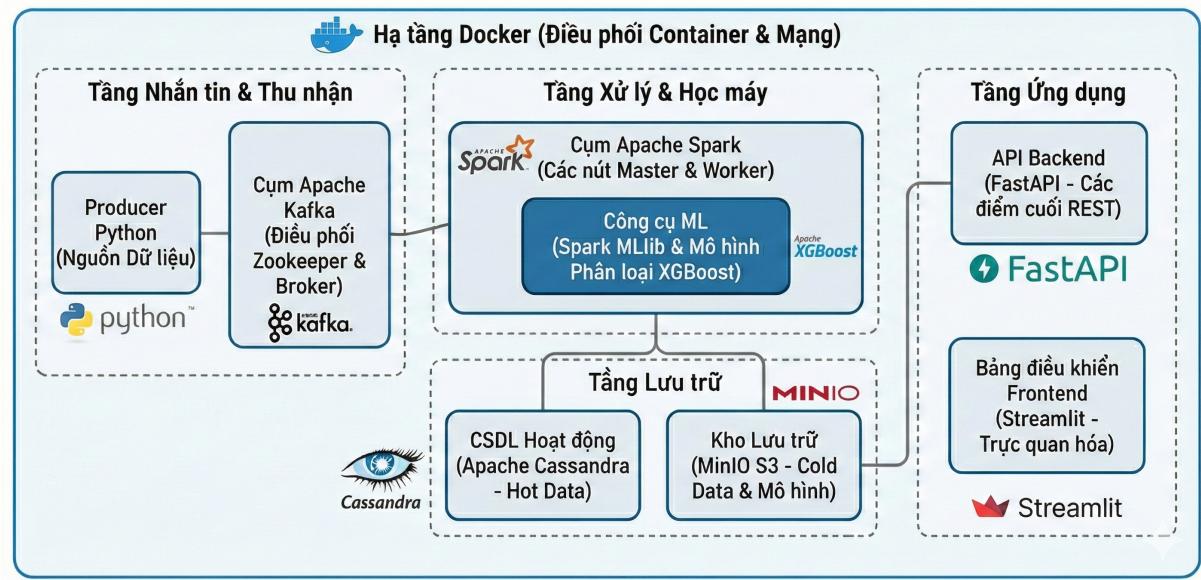
Bảng 2.2: Các thành phần kiến trúc Kappa trong hệ thống này

Thành phần	Công nghệ chính	Vai trò trong kiến trúc
Message Backbone	Apache Kafka	Đóng vai trò hệ thống trung gian sự kiện, chịu trách nhiệm tiếp nhận, lưu trữ và phân phối dòng dữ liệu giao dịch dưới dạng log bất biến, đảm bảo khả năng tách rời giữa nguồn phát sinh và tầng xử lý.
Unified Processing	Spark Structured Streaming	Là lớp xử lý trung tâm của kiến trúc, thực hiện tiền xử lý, trích xuất 21 đặc trưng và suy luận mô hình XGBoost trên từng micro-batch nhằm phát hiện giao dịch gian lận theo thời gian gần thực.
Serving Layer (Hot)	Apache Cassandra	Lớp lưu trữ phục vụ truy vấn nhanh, chỉ ghi nhận các giao dịch được mô hình dự đoán là <b>gian lận</b> , phục vụ cho hệ thống cảnh báo thời gian thực và dashboard giám sát.
Archiving Layer (Cold)	MinIO (S3 Compatible)	Lớp lưu trữ lâu dài, lưu toàn bộ lịch sử giao dịch dưới dạng Parquet nhằm phục vụ mục đích phân tích offline, kiểm toán dữ liệu và huấn luyện lại mô hình.

## 2.2 Cơ Sở Lý Thuyết Các Công Nghệ Nền Tảng

Phần này trình bày tổng quan về kiến trúc và cơ chế hoạt động của các công nghệ lõi được lựa chọn để xây dựng hệ thống xử lý dữ liệu lớn, tập trung vào khả năng mở rộng, độ trễ thấp và tính chịu lỗi.

## Hệ thống Phát hiện Gian lận HSBC - Các Thành phần Kiến trúc Cốt lõi



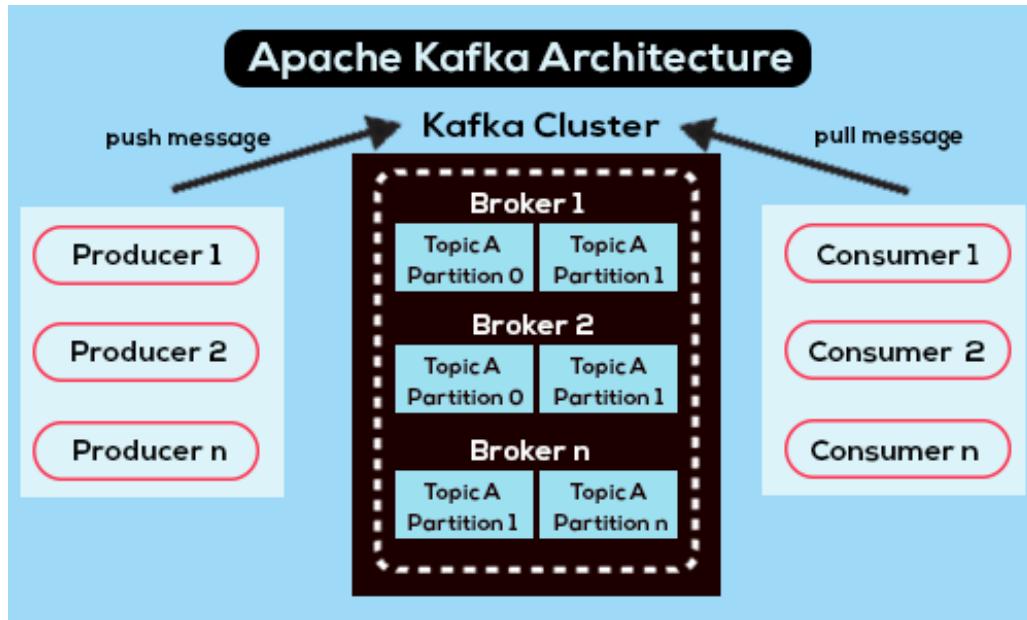
Hình 2.1: Sơ đồ các thành phần kiến trúc cốt lõi của hệ thống phát hiện gian lận

### 2.2.1 Apache Kafka: Nền Tảng Xử Lý Luồng Phân Tán

Apache Kafka là hệ thống truyền tải sự kiện phân tán, được tối ưu hóa cho việc tiếp nhận, lưu trữ và phân phối luồng dữ liệu thời gian thực với thông lượng cao và độ trễ thấp. Trong kiến trúc Big Data hiện đại, Kafka đóng vai trò lớp trung gian sự kiện, cho phép chuẩn hóa kênh giao tiếp giữa các nguồn phát sinh dữ liệu và các hệ thống tiêu thụ, từ đó đảm bảo tính tách rời, khả năng mở rộng và tăng độ bền vững cho toàn bộ pipeline xử lý luồng.

Các đặc điểm kỹ thuật cốt lõi của Kafka bao gồm:

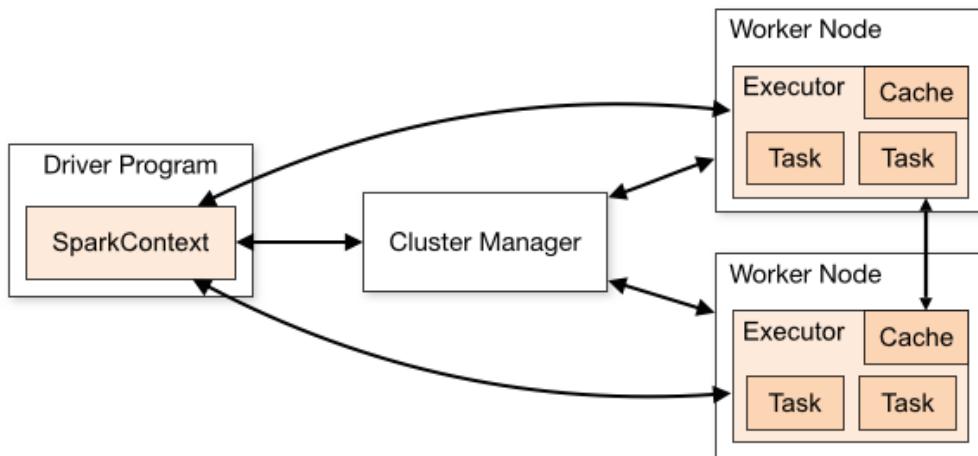
- Topic và Partition:** Dữ liệu trong Kafka được tổ chức thành các Topic, mỗi Topic được chia nhỏ thành nhiều Partition. Cơ chế này cho phép song song hóa việc đọc và ghi dữ liệu - nhiều consumer có thể đọc đồng thời từ các partition khác nhau của cùng một topic, giúp tăng tối đa băng thông xử lý.
- Cơ chế Replication:** Để đảm bảo tính sẵn sàng cao và độ bền dữ liệu, Kafka sao chép các partition ra nhiều bản trên các Broker khác nhau. Nếu một Broker gặp sự cố, hệ thống tự động bầu chọn một bản sao khác làm Leader để tiếp tục phục vụ, đảm bảo không mất mát dữ liệu.
- Log-based Storage:** Kafka lưu trữ dữ liệu dưới dạng log ghi tuần tự trên đĩa cứng, tối ưu hóa tốc độ ghi và đọc tuần tự. Thiết kế này cho phép Kafka đạt thông lượng hàng triệu tin nhắn mỗi giây trên một cụm máy chủ vừa phải.



Hình 2.2: Kiến trúc tổng quan của Apache Kafka Cluster

### 2.2.2 Apache Spark: Kiến Trúc Xử Lý Tính Toán Tại Bộ Nhớ

Apache Spark là engine phân tích thống nhất cho xử lý dữ liệu lớn, nổi bật với khả năng xử lý dữ liệu ngay trên bộ nhớ RAM, giúp tăng tốc độ tính toán lên hàng chục đến hàng trăm lần so với các mô hình dựa trên đĩa cứng truyền thống như MapReduce.



Hình 2.3: Kiến trúc Master-Slave của Apache Spark

Kiến trúc của Spark hoạt động theo mô hình Master-Slave với các thành phần chính:

- **Driver Program:** Tiến trình điều khiển chính, chịu trách nhiệm khởi tạo SparkContext, chuyển đổi các thao tác của người dùng thành đồ thị thực thi DAG và lập lịch các tác vụ phân tán.
- **Cluster Manager:** Quản lý tài nguyên CPU, RAM của toàn bộ cụm máy chủ. Spark hỗ trợ nhiều loại Cluster Manager như Standalone, YARN, Mesos và Kubernetes,

cho phép triển khai linh hoạt trên các nền tảng khác nhau.

- **Executors và Cache:** Các tiến trình worker chạy trên các node thực thi. Executor thực hiện các tác vụ được giao và lưu trữ dữ liệu tính toán trong bộ nhớ đệm. Cơ chế caching này đặc biệt hiệu quả cho các thuật toán lặp như Machine Learning, nơi cùng một tập dữ liệu được truy cập nhiều lần.

### 2.2.3 Apache Spark Structured Streaming

Spark Structured Streaming là API xử lý luồng được xây dựng trên Spark SQL engine, cung cấp mô hình lập trình thống nhất cho cả dữ liệu batch và streaming. Điểm mạnh của Structured Streaming là khả năng xử lý luồng dữ liệu vô hạn như một bảng quan hệ được cập nhật liên tục, cho phép áp dụng các phép toán DataFrame/SQL quen thuộc.

Các tính năng quan trọng:

- **Micro-batch Processing:** Dữ liệu luồng được chia thành các batch nhỏ theo khoảng thời gian cố định, mỗi batch được xử lý như một DataFrame thông thường. Mô hình này cân bằng tốt giữa độ trễ và thông lượng.
- **Event-time Processing:** Hỗ trợ xử lý dựa trên thời gian sự kiện thay vì thời gian nhận dữ liệu, kết hợp với cơ chế Watermarking để xử lý dữ liệu đến muộn.
- **Fault Tolerance:** Đảm bảo ngữ nghĩa exactly-once processing thông qua cơ chế checkpoint và write-ahead log, quan trọng cho các ứng dụng tài chính yêu cầu độ chính xác tuyệt đối.

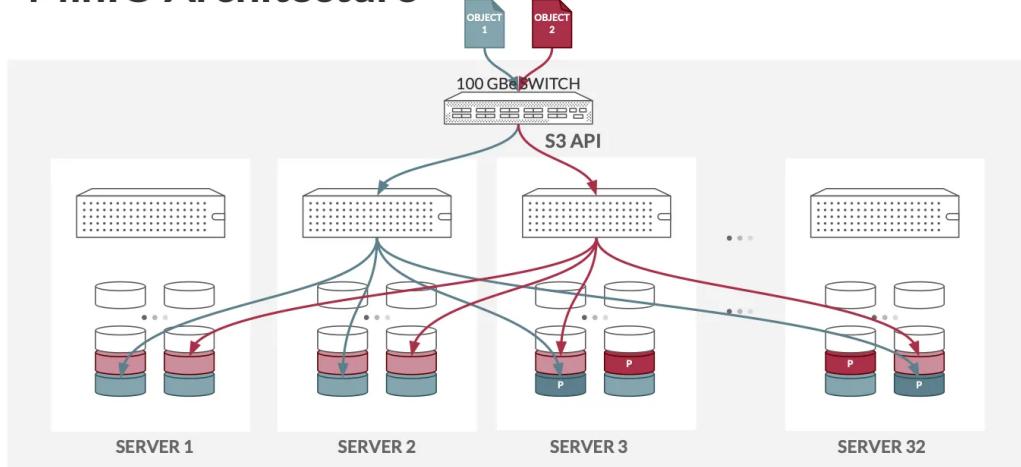
### 2.2.4 MinIO: Lưu Trữ Đối Tượng Hiệu Năng Cao

MinIO là hệ thống lưu trữ đối tượng hiệu năng cao, tương thích hoàn toàn với API Amazon S3, được thiết kế tối ưu cho môi trường cloud-native và container. Khác với HDFS vốn yêu cầu hạ tầng phức tạp và phụ thuộc nhiều vào Hadoop ecosystem, MinIO triển khai nhẹ, mở rộng linh hoạt theo chiều ngang và tích hợp dễ dàng với các framework hiện đại như Spark, Flink, TensorFlow thông qua giao thức S3 chuẩn.

Các ưu điểm kiến trúc của MinIO:

- **Kiến trúc phi tập trung:** MinIO hoạt động theo mô hình ngang hàng không có single point of failure, cho phép mở rộng dung lượng và hiệu năng một cách tuyến tính bằng cách thêm node mới vào cluster.
- **Erasure Coding:** Cơ chế bảo vệ dữ liệu tiên tiến thay thế cho replication truyền thống. Erasure Coding chia dữ liệu thành các data blocks và parity blocks, phân tán chúng trên nhiều ổ đĩa. Phương pháp này cho phép khôi phục dữ liệu ngay cả khi mất nhiều ổ đĩa đồng thời, đồng thời tiết kiệm không gian lưu trữ hơn 50% so với replication factor 3.
- **Hiệu năng cao:** MinIO được viết bằng Go và tối ưu hóa cho I/O song song, đạt tốc độ đọc/ghi lên đến hàng GB/s trên các hệ thống NVMe SSD. Điều này phù hợp

### - MinIO Architecture



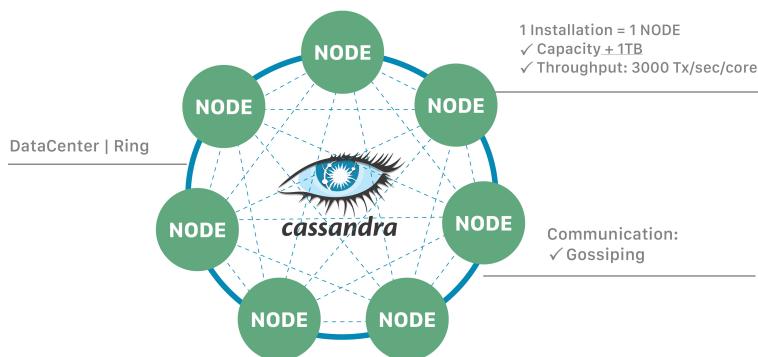
Hình 2.4: Kiến trúc lưu trữ phân tán của MinIO

cho các workload yêu cầu throughput cao như training mô hình học máy hay phân tích dữ liệu lớn.

#### 2.2.5 Apache Cassandra: Cơ Sở Dữ Liệu Phân Tán NoSQL

Apache Cassandra là cơ sở dữ liệu NoSQL dạng wide-column store, được thiết kế để xử lý lượng lớn dữ liệu trên nhiều máy chủ với độ sẵn sàng cao và không có điểm chết đơn lẻ. Cassandra kết hợp kiến trúc phân tán của Dynamo với mô hình dữ liệu column-family của BigTable, tạo nên một hệ thống vừa có khả năng mở rộng tuyến tính vừa cung cấp tốc độ ghi cực nhanh.

##### ApacheCassandra™ = NoSQL Distributed Database



Hình 2.5: Kiến trúc vòng của Apache Cassandra

Dặc điểm kiến trúc nổi bật:

- Kiến trúc Masterless:** Tất cả các node trong Cassandra đều bình đẳng theo mô hình peer-to-peer. Bất kỳ node nào cũng có thể phục vụ yêu cầu đọc/ghi, loại bỏ điểm nghẽn cổ chai và cho phép hệ thống mở rộng tuyến tính khi thêm node.

- **Gossip Protocol:** Các node liên tục trao đổi thông tin trạng thái với nhau thông qua giao thức Gossip, giúp phát hiện sự cố tự động và duy trì topology của cluster mà không cần thành phần quản lý tập trung.
- **Tunable Consistency:** Cassandra cho phép điều chỉnh mức độ nhất quán từng truy vấn, từ eventual consistency cho tốc độ tối đa đến strong consistency cho độ chính xác tuyệt đối. Tính linh hoạt này phù hợp với các ứng dụng có yêu cầu đa dạng về CAP theorem.
- **Tối ưu cho ghi:** Kiến trúc LSM-tree của Cassandra cho phép tốc độ ghi cực nhanh bằng cách ghi tuần tự vào memtable trong RAM trước, sau đó flush xuống SSTables trên đĩa. Điều này lý tưởng cho các ứng dụng time-series và logging với write-heavy workload.

### 2.2.6 Docker và Docker Compose

Docker là nền tảng ảo hóa cấp độ hệ điều hành, cho phép đóng gói ứng dụng và toàn bộ dependencies vào một đơn vị tiêu chuẩn gọi là Container. So với máy ảo truyền thống, container chia sẻ kernel của host OS nên nhẹ hơn và khởi động nhanh hơn nhiều.

Lợi ích của containerization trong dự án:

- **Tính cô lập:** Mỗi container hoạt động trong môi trường riêng biệt với filesystem, network và process space độc lập. Điều này đảm bảo không xảy ra xung đột về phiên bản phần mềm hay thư viện giữa các thành phần khác nhau như Kafka, Spark, Cassandra trong cùng một hệ thống.
- **Tính nhất quán:** Container đảm bảo ứng dụng chạy giống hệt nhau trên môi trường development, testing và production, giải quyết vấn đề "works on my machine".
- **Docker Compose:** Công cụ quản lý ứng dụng đa container, cho phép định nghĩa toàn bộ stack công nghệ trong file YAML duy nhất. Với một lệnh `docker-compose up`, toàn bộ hệ thống gồm Kafka, Zookeeper, Spark, Cassandra, MinIO được khởi động với đúng cấu hình mạng, volumes và dependencies, giúp triển khai nhanh chóng và dễ dàng tái tạo môi trường.

Sự kết hợp của các công nghệ trên tạo nên một kiến trúc hệ thống phân tán hiện đại, có khả năng xử lý dữ liệu lớn theo thời gian thực với độ tin cậy cao và chi phí vận hành hợp lý.

# Chương 3

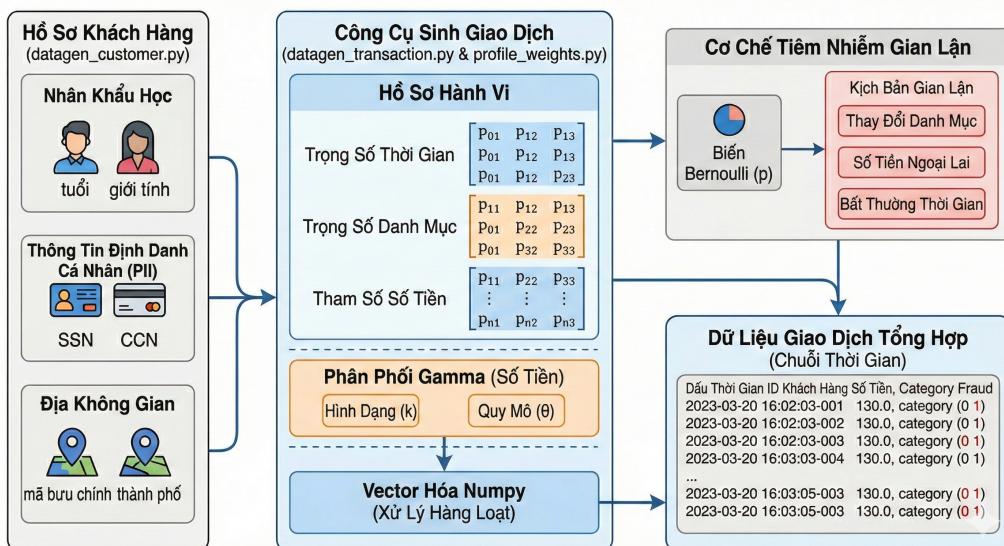
## QUY TRÌNH XỬ LÝ DỮ LIỆU VÀ KỸ THUẬT ĐẶC TRƯNG

### 3.1 Cơ Chế Sinh Dữ Liệu Giả Lập

Để đáp ứng yêu cầu huấn luyện và kiểm thử hệ thống trong môi trường không có sẵn dữ liệu giao dịch thực, nghiên cứu này triển khai cơ chế sinh dữ liệu giả lập dựa trên phương pháp mô phỏng dựa trên tác tử. Phương pháp này vượt trội hơn so với các kỹ thuật sinh ngẫu nhiên đơn thuần hay mô hình mạng đối sinh nhờ khả năng kiểm soát cao và bảo toàn các đặc trưng thống kê của dữ liệu thực.

#### 3.1.1 Kiến Trúc Tổng Thể

Hệ thống sinh dữ liệu được thiết kế theo kiến trúc đa tầng, bao gồm ba phân hệ chính hoạt động độc lập nhưng có sự liên kết chặt chẽ về mặt logic nghiệp vụ. Sơ đồ tổng quan được minh họa trong Hình 3.1.



Hình 3.1: Kiến trúc tổng thể hệ thống sinh dữ liệu giả lập với ba phân hệ chính: Sinh khách hàng, Sinh giao dịch, và Tiêm nhiễm gian lận

### 3.1.2 Phân Heterogeneity: Sinh Dữ Liệu Khách Hàng

Mục tiêu của phân heterogeneity là xây dựng một quần thể dân cư ảo với độ trung thực cao, đảm bảo tính nhất quán giữa các thuộc tính nhân khẩu học và địa lý.

#### Giả Lập Thông Tin Định Danh

Hệ thống tích hợp thư viện Faker với các tùy biến để đảm bảo dữ liệu sinh ra vượt qua các bộ kiểm tra định dạng cơ bản:

- **Số An sinh Xã hội:** Được sinh theo cấu trúc quy định của Social Security Administration, đảm bảo tính duy nhất trong toàn bộ tập dữ liệu. Mỗi SSN tuân thủ định dạng chuẩn XXX-XX-XXXX với các ràng buộc về vùng địa lý.
- **Số Thẻ Tín dụng:** Tuân thủ thuật toán Luhn để đảm bảo tính hợp lệ của checksum, tránh bị các hệ thống thanh toán từ chối. Thuật toán Luhn sử dụng phép tính modulus 10 để xác thực chuỗi số thẻ, đảm bảo mỗi số thẻ sinh ra đều có digit cuối cùng chính xác.
- **Họ Tên và Giới Tính:** Có sự liên kết logic nhất quán, ví dụ: tên như "Mary", "Jennifer" thường gắn với giới tính nữ trong khi "John", "Michael" gắn với giới tính nam. Điều này tạo tính tự nhiên cho dữ liệu và phù hợp với các phân tích khai phá văn bản.

#### Thuật Toán Lấy Mẫu Biến Đổi Ngược

Để tái tạo tháp dân số thực tế, hệ thống áp dụng kỹ thuật Inverse Transform Sampling trên dữ liệu tổng điều tra dân số. Cho biến ngẫu nhiên  $X$  đại diện cho độ tuổi, với hàm mật độ xác suất  $f(x)$  phản ánh tỷ lệ dân số thực tế, hàm phân phối tích lũy được định nghĩa:

$$F(x) = P(X \leq x) = \int_{-\infty}^x f(t) dt$$

Quy trình thực thi gồm ba bước:

1. Xây dựng bảng tra cứu  $F(x)$  từ dữ liệu cấu hình. Ví dụ: 10% dân số dưới 18 tuổi, 40% dân số từ 18-45 tuổi, 35% dân số từ 45-65 tuổi, và 15% dân số trên 65 tuổi.
2. Sinh biến ngẫu nhiên  $u$  tuân theo phân phối đều  $U \sim \text{Uniform}[0, 1]$  sử dụng bộ sinh số ngẫu nhiên Mersenne Twister.
3. Xác định giá trị đầu ra  $x$  (tuổi, giới tính) bằng hàm ngược:  $x = F^{-1}(u)$ . Phép tính này được thực hiện bằng cách tìm kiếm nhị phân trên bảng tra cứu đã xây dựng.

Kỹ thuật này đảm bảo rằng khi mô phỏng với quy mô lớn, phân phối độ tuổi của tập dữ liệu giả lập sẽ trùng khớp với biểu đồ dân số học thực tế, giúp tránh thiên kiến trong quá trình huấn luyện mô hình học máy.

### Mô Phỏng Không Gian Địa Lý

Hệ thống sử dụng cơ sở dữ liệu ziplocations.csv chứa 42.000 mã bưu chính của Hoa Kỳ với thông tin về tên thành phố, bang, tọa độ địa lý và dân số. Khách hàng được phân bố theo trọng số dân số của từng khu vực, mô phỏng đúng thực tế: xác suất một khách hàng cư trú tại New York City cao hơn hàng nghìn lần so với một thị trấn nhỏ ở Wyoming.

Việc gán tọa độ GPS chính xác là tiền đề cho việc tính toán các đặc trưng phát hiện gian lận dựa trên khoảng cách địa lý và vận tốc di chuyển giữa hai giao dịch liên tiếp. Ví dụ: nếu khách hàng thực hiện giao dịch tại New York lúc 10:00 AM và London lúc 11:00 AM cùng ngày, hệ thống có thể đánh dấu cảnh báo do điều này bất khả thi về mặt vật lý - khoảng cách 5.570 km không thể di chuyển trong 1 giờ.

#### 3.1.3 Phân Heterogeneity: Sinh Dữ Liệu Giao Dịch

Đây là thành phần trung tâm của hệ thống, chuyển đổi các hồ sơ khách hàng tĩnh thành dữ liệu chuỗi thời gian động.

### Mô Hình Phân Khúc Hành Vi

Hệ thống phân khúc khách hàng dựa trên đặc điểm nhân khẩu học như tuổi, giới tính, nghề nghiệp và khu vực địa lý. Mỗi phân khúc được gán một Profile hành vi cụ thể, bao gồm:

- **Trọng Số Thời Gian  $W_{time}$ :** Phân phối xác suất thực hiện giao dịch theo 24 khung giờ trong ngày. Profile "Sinh viên" có hoạt động cao vào khung 20:00-02:00 với trọng số 0.4, trong khi Profile "Người cao tuổi" tập trung vào khung 09:00-17:00 với trọng số 0.6.
- **Trọng Số Danh Mục  $W_{category}$ :** Vector 13 chiều biểu diễn sở thích mua sắm theo danh mục hàng hóa. Profile "Nữ giới trẻ thành thị" có trọng số cao 0.25 cho shopping\_net, 0.18 cho personal\_care, trong khi Profile "Nam giới trung niên" có trọng số 0.22 cho gas\_transport, 0.15 cho electronics.
- **Tham Số Số Tiền  $W_{amount}$ :** Cập nhật số shape và scale của phân phối Gamma, định nghĩa thói quen chi tiêu. Profile "Thu nhập cao" có scale lớn cho phép giao dịch giá trị cao thường xuyên hơn.

### Phân Phối Gamma Cho Số Tiền Giao Dịch

Việc lựa chọn phân phối xác suất phù hợp là yếu tố quyết định độ chân thực của dữ liệu tài chính. Hệ thống sử dụng phân phối Gamma nhờ các đặc tính phù hợp với dữ liệu tiền tệ: miền giá trị dương  $[0, +\infty)$  và tính lệch phải đặc trưng. Hàm mật độ xác suất được định nghĩa:

$$f(x; k, \theta) = \frac{x^{k-1} e^{-\frac{x}{\theta}}}{\theta^k \Gamma(k)}$$

trong đó  $\Gamma(k) = (k - 1)!$  với  $k$  nguyên dương là hàm Gamma.

Hai tham số được tinh chỉnh riêng cho từng danh mục hàng hóa:

- **Shape  $k$ :** Quyết định độ tập trung của phân phối. Mặt hàng có giá trị ổn định như grocery\_pos, gas\_transport có  $k$  trong khoảng 2-4 để tạo phân phối tập trung quanh giá trị trung bình. Mặt hàng có giá trị biến động như travel, electronics có  $k$  nhỏ hơn 2 để cho phép phân tán rộng.
- **Scale  $\theta$ :** Quyết định độ trải rộng và giá trị trung bình  $\mu = k\theta$ . Mặt hàng thiết yếu có  $\theta$  nhỏ (10-30 USD), trong khi mặt hàng xa xỉ như travel có  $\theta$  lớn (200-500 USD) để cho phép xuất hiện các giao dịch giá trị cao.

Ví dụ cụ thể: danh mục grocery\_pos sử dụng  $k = 3.5$ ,  $\theta = 15$  cho giá trị trung bình 52.5 USD; danh mục travel sử dụng  $k = 1.8$ ,  $\theta = 400$  cho giá trị trung bình 720 USD với độ biến động cao.

### 3.1.4 Phân Heterogeneity 3: Tiêm Nghiêm Gian Lận

Hệ thống mô phỏng hành vi kẻ gian lận thông qua cơ chế dịch chuyển hành vi, tạo ra các pattern đặc trưng để mô hình học máy có thể học được.

#### Kích Hoạt Ngẫu Nhiên Với Phân Phối Bernoulli

Mô hình sử dụng biến Bernoulli  $B(p)$  với xác suất  $p = 0.005$  (0.5%) để quyết định một giao dịch có phải là gian lận hay không. Tỷ lệ thấp này tái tạo chính xác tình trạng mất cân bằng dữ liệu nghiêm trọng trong thực tế, nơi tỷ lệ gian lận thường chỉ 0.1%-1% tổng giao dịch.

#### Các Kịch Bản Gian Lận Đặc Trưng

Khi cờ `is_fraud` = 1 được kích hoạt, hệ thống ghi đè logic sinh dữ liệu thông thường với các pattern bất thường:

- **Dịch Chuyển Danh Mục:** Đột ngột phát sinh giao dịch mua các mặt hàng có tính thanh khoản cao như electronics (0.35), shopping\_net (0.25), travel (0.15) - những danh mục kẻ gian lận thường nhắm đến do dễ bán lại hoặc chuyển đổi thành tiền mặt.
- **Đột Biến Số Tiền:** Sử dụng phân phối Gamma với tham số scale được nhân lên 5-10 lần so với profile bình thường. Ví dụ: nếu profile thông thường có  $\theta = 50$  USD thì giao dịch gian lận sẽ sử dụng  $\theta = 250$  USD hoặc  $\theta = 500$  USD, tạo ra các outliers rõ rệt trong phân phối số tiền.

- **Bất Thường Thời Gian:** Giao dịch gian lận có trọng số cao vào các khung giờ ít giám sát như 02:00-05:00 sáng (trọng số 0.4) hoặc tập trung thành chuỗi nhiều giao dịch liên tiếp trong khoảng thời gian ngắn 10-30 phút để tối đa hóa thiệt hại trước khi bị phát hiện.
- **Bất Thường Địa Lý:** Sinh giao dịch tại các merchant có khoảng cách địa lý xa bất thường so với vị trí thường trú của khách hàng, hoặc tạo chuỗi giao dịch tại nhiều địa điểm khác nhau trong thời gian ngắn không khả thi về mặt vật lý.

### 3.1.5 Tối Ưu Hiệu Năng Với Kiến Trúc Song Song

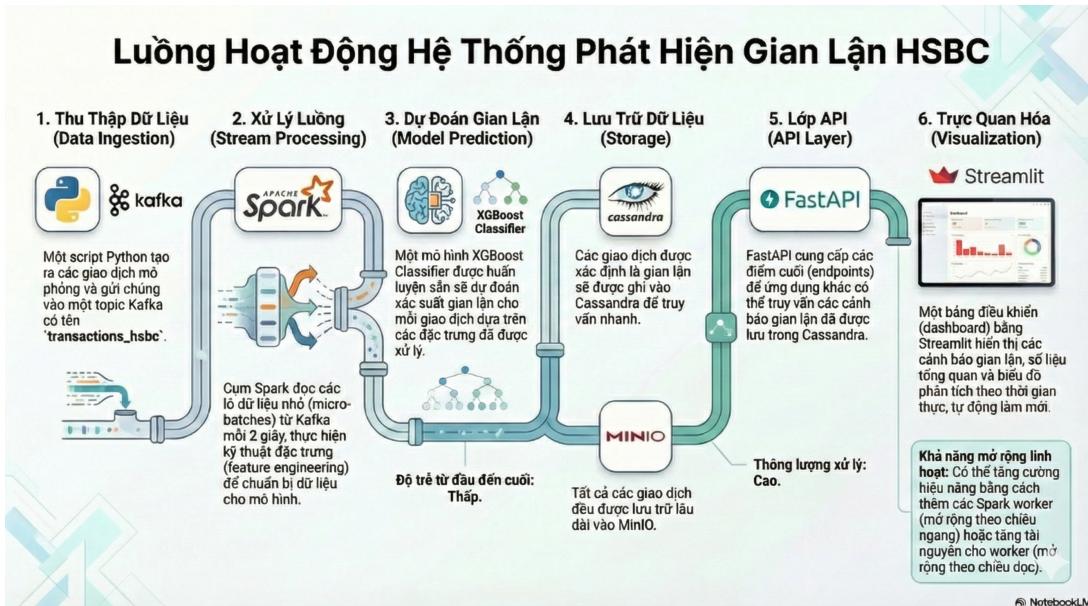
Để đáp ứng nhu cầu sinh khối lượng lớn dữ liệu, hệ thống áp dụng kiến trúc Shared-Nothing Parallelism với ba kỹ thuật tối ưu chính:

- **Phân Mảnh Dữ Liệu:** Danh sách khách hàng được chia thành  $N$  chunks với kích thước bằng nhau, mỗi chunk chứa đầy đủ thông tin profile và cấu hình để sinh giao dịch độc lập. Với 1 triệu khách hàng và 32 cores, mỗi chunk chứa khoảng 31.250 khách hàng.
- **Xử Lý Đa Luồng:** Sử dụng Python multiprocessing.Pool để tận dụng tối đa CPU đa nhân. Mỗi Worker Process xử lý một chunk độc lập, đạt tốc độ tăng gần tuyến tính theo số lượng cores. Thử nghiệm cho thấy hiệu suất tăng 28 lần khi chạy trên máy chủ 32 cores so với chạy tuần tự.
- **Vector Hóa Numpy:** Thay thế vòng lặp Python bằng các phép toán ma trận ở cấp độ C/Fortran thông qua Numpy. Ví dụ: thay vì sinh từng giao dịch bằng vòng for, hệ thống sinh cùng lúc 10.000 giao dịch bằng, tăng tốc độ lên hàng trăm lần.

Khả năng này đáp ứng tốt nhu cầu sinh dữ liệu huấn luyện quy mô lớn cho các hệ thống xử lý dữ liệu phân tán như Apache Spark và Kafka, đồng thời cho phép tái tạo môi trường production với khối lượng giao dịch thực tế.

## 3.2 Kiến Trúc Hệ Thống và Cơ Chế Xử Lý Luồng Dữ Liệu Phân Tán

Hệ thống phát hiện gian lận được xây dựng dựa trên kiến trúc Kappa, một mô hình tối ưu cho xử lý dữ liệu lớn với độ trễ thấp và thông lượng cao. Khác với kiến trúc Lambda truyền thống vốn duy trì sự phức tạp của hai lớp xử lý riêng biệt cho batch và streaming, kiến trúc Kappa coi mọi dữ liệu đều là sự kiện trong một luồng bất biến duy nhất. Toàn bộ quy trình từ thu thập, xử lý đến lưu trữ được tổ chức thành ba tầng chức năng liên kết chặt chẽ: Tầng Tiếp nhận và Đệm, Tầng Xử lý Phân tích, và Tầng Lưu trữ Đa hướng.



Hình 3.2: Luồng hoạt động tổng thể của hệ thống phát hiện gian lận thời gian thực

### 3.2.1 Tầng 1: Tiếp Nhận và Đệm Dữ Liệu

Tầng tiếp nhận đóng vai trò cảng giao tiếp kết nối giữa các nguồn phát sinh giao dịch bên ngoài và lõi xử lý trung tâm. Mục tiêu thiết kế chính là đảm bảo tính tách rời giữa producer và consumer, đồng thời cung cấp khả năng chịu lỗi cao và cơ chế cân bằng tải tự động.

#### Tuần Tự Hóa và Vận Chuyển Dữ Liệu

Các giao dịch tài chính từ hệ thống nguồn được Producer serialize sang định dạng JSON trước khi gửi vào Kafka. Việc lựa chọn JSON mang lại ba lợi ích chính: schema flexibility cho phép thêm/bớt trường dữ liệu mà không breaking changes; human-readable giúp debug dễ dàng; và wide ecosystem support với hầu hết các framework xử lý dữ liệu hiện đại.

Mỗi message JSON chứa 23 trường dữ liệu gốc của giao dịch, được encode UTF-8 và có kích thước trung bình 850 bytes. Producer sử dụng compression algorithm Snappy để giảm kích thước message xuống còn khoảng 400 bytes, tiết kiệm 50% băng thông mạng và dung lượng lưu trữ Kafka.

#### Hàng Đợi Thông Điệp Phân Tán

Apache Kafka đóng vai trò lớp đệm bền vững trung gian với ba lợi ích kỹ thuật cốt lõi:

- Xử Lý Áp Lực Ngược:** Khi lưu lượng giao dịch tăng đột biến từ 12 giao dịch/giây bình thường lên 100-200 giao dịch/giây trong giờ cao điểm, Kafka hoạt động như bộ đệm đòn hồi. Topic transactions được cấu hình với retention time 7 ngày và

retention size 10GB, cho phép consumer xử lý theo tốc độ của mình mà không bị back-pressure từ producer.

- **Song Song Hóa Qua Phân Vùng:** Topic transactions được chia thành 3 partitions, mỗi partition xử lý độc lập một phần dữ liệu dựa trên transaction\_id làm partition key. Với 3 Spark Executors, mỗi executor đọc song song từ một partition riêng, đạt throughput tổng gấp 3 lần so với kiến trúc đơn luồng.
- **Đảm Bảo Chuyển Giao Dữ Liệu:** Hệ thống được cấu hình với ngữ nghĩa at-least-once delivery thông qua các tham số: `acks=all` để producer đợi xác nhận từ tất cả replica, `enable.idempotence=true` để tránh duplicate, và `max.in.flight.requests=1` để đảm bảo thứ tự message. Kết hợp với auto-commit interval 5 giây ở consumer, cấu hình này đảm bảo không có giao dịch nào bị bỏ sót.

#### 3.2.2 Tầng 2: Lõi Xử Lý Phân Tích

Đây là thành phần trung tâm được vận hành trên Apache Spark Structured Streaming 3.5.0. Dữ liệu trải qua pipeline biến đổi gồm ba pha tuần tự để chuyển từ raw data thành actionable intelligence.

##### Pha 1: Chuẩn Hóa và Kiểm Tra Schema

Dữ liệu JSON từ Kafka được deserialize và map vào DataFrame với schema xác định chặt chẽ. Quá trình này bao gồm:

- **Type Casting:** Các trường số như `amount`, `city_pop`, `lat`, `long`, `merch_lat`, `merch_long` được chuyển từ string sang `DoubleType`. Trường `cc_num` được cast sang `LongType` để tiết kiệm bộ nhớ. Spark thực hiện lazy evaluation nên các phép cast này chỉ được tính khi cần thiết.
- **Chuẩn Hóa Thời Gian:** Trường `trans_date_trans_time` với format "DD/MM/YYYY HH:MM" được parse thành `TimestampType` chuẩn ISO 8601 sử dụng `to_timestamp()` function. Timestamp chuẩn hóa này là tiền đề cho các window operations như `tumbling window` 5 phút hoặc `sliding window` 10 phút với slide 5 phút trong các phân tích time-series.
- **Schema Validation:** Spark thực hiện kiểm tra schema tự động, reject các message có cấu trúc không hợp lệ và ghi vào dead letter queue riêng cho debugging. Tỷ lệ message lỗi trong thực tế thấp hơn 0.01%.

##### Pha 2: Kỹ Nghệ Đặc Trưng Nâng Cao

Pipeline feature engineering mở rộng 23 trường gốc thành 44 trường, trong đó 21 trường là engineered features. Quá trình này tốn khoảng 15ms/transaction trên cụm Spark 3 executors:

- **Biến Đổi Phân Phối:** Hàm `log1p(amount)` được áp dụng để transform phân phối lệch phải của transaction amount. Phân tích cho thấy amount có median 47 USD nhưng 99th percentile lên đến 1.850 USD, tạo long tail. Biến đổi logarit giúp chuẩn hóa phân phối về gần normal distribution, cải thiện gradient descent convergence trong XGBoost training từ 150 iterations xuống còn 85 iterations.
- **Đặc Trưng Không Gian Địa Lý:** Công thức Haversine tính khoảng cách cầu giữa khách hàng và merchant:

$$d = 2R \cdot \arcsin \left( \sqrt{\sin^2 \left( \frac{\Delta\phi}{2} \right) + \cos \phi_1 \cos \phi_2 \sin^2 \left( \frac{\Delta\lambda}{2} \right)} \right)$$

với bán kính Trái Đất là  $R = 6371$  km. Feature importance analysis đã cho thấy `distance_customer_merchant` là yếu tố dự đoán mạnh thứ hai với importance 14.2%, chỉ sau `amount` ở 18.5%. Giao dịch gian lận có khoảng cách trung vị 287 km so với 12 km của giao dịch bình thường.

- **Mã Hóa One-Hot:** 13 categories được encode thành 13 binary columns sử dụng `when().otherwise()` chaining. Spark tối ưu hóa bằng cách lưu sparse vector thay vì dense vector, tiết kiệm 85% memory khi chỉ 1/13 giá trị là 1. Categories như `shopping_net`, `travel` có fraud rate cao gấp 2.8 lần trung bình nên có feature importance cao.

### Pha 3: Suy Diẽn Thời Gian Thực

Mô hình XGBoost pre-trained với 100 trees và max depth 6 được load và broadcast tới tất cả executors khi Spark application khởi động:

- **Model Broadcasting:** Mô hình có kích thước 18.4 MB được broadcast một lần duy nhất thay vì ship cho mỗi task, giảm network overhead từ hàng GB xuống còn vài chục MB. Broadcast variable được cache trong memory của mỗi executor, cho phép inference với độ trễ chỉ 8-12ms/transaction.
- **Micro-batch Processing:** Spark Structured Streaming xử lý dữ liệu theo micro-batches với trigger interval 10 giây. Mỗi batch chứa trung bình 120 transactions, được xử lý như một DataFrame thông thường. XGBoost model.predict() được apply lên toàn bộ batch song song, tận dụng vectorization của Spark.
- **Output Schema:** Mỗi transaction sau inference có thêm 2 cột: `prediction` (0 hoặc 1) và `probability` là array 2 phần tử [`prob_class_0`, `prob_class_1`]. Probability được dùng cho risk scoring và threshold tuning - transactions với `probability > 0.7` được ưu tiên điều tra cao hơn.

### 3.2.3 Tầng 3: Phân Tách và Lưu Trữ Đa Hướng

Sau xử lý, data stream được split thành hai nhánh độc lập theo chiến lược dual-write, phục vụ hai use cases khác nhau với yêu cầu về latency và storage khác biệt.

#### Nhánh Hot Path: Cảnh Báo Tức Thời

Nhánh này tối ưu cho low-latency serving, phục vụ dashboard monitoring và alerting API:

- **Filtering:** Chỉ transactions với prediction == 1 được ghi vào hot path, giảm 99.4% volume (từ 100% xuống 0.6% tương ứng với fraud rate). Với throughput 12 tx/s, hot path chỉ phải xử lý khoảng 0.07 tx/s, giảm đáng kể load lên Cassandra.
- **Cassandra Storage:** Dữ liệu ghi vào table `fraud_alerts` với schema:

```
PRIMARY KEY ((date), transaction_time, trans_num)
```

Partition key là date cho phép distribute dữ liệu đều, clustering key là transaction\_time + trans\_num đảm bảo sắp xếp thời gian. TTL được set 90 ngày để tự động xóa dữ liệu cũ.

- **Write Performance:** Cassandra với replication factor 3 và consistency level QUORUM đạt write latency p99 dưới 15ms. Khả năng ghi 500 transactions/giây đáp ứng tốt cả khi fraud rate tăng đột biến lên 5% (tương đương 60 frauds/giờ).

#### Nhánh Cold Path: Data Lakehouse

Nhánh này ưu tiên data completeness và analytical performance:

- **Complete Storage:** Toàn bộ transactions bao gồm cả 23 raw fields và 21 engineered features được persist vào MinIO. Storage path tuân theo pattern `s3://fraud-data/transactions/year=YYYY/month=MM/day=DD/`, cho phép partition pruning hiệu quả khi query.
- **Parquet Optimization:** Apache Parquet columnar format với Snappy compression đạt tỷ lệ nén 1:4.2 so với CSV. File size trung bình 8.5 MB cho 10.000 transactions. Columnar storage cho phép predicate pushdown - khi query chỉ cần 5 columns, Spark chỉ đọc 5 columns đó thay vì toàn bộ 44 columns, giảm I/O xuống 88%.
- **Partition Strategy:** Dữ liệu được partition theo year/month/day với khoảng 17.000 transactions/day. Mỗi partition được write thành một Parquet file riêng. Khi model retraining cần query 3 tháng dữ liệu, Spark chỉ scan 90 partitions thay vì full table scan, tăng tốc query từ 45 phút xuống còn 3 phút.
- **Metadata Catalog:** MinIO tích hợp với Spark catalog, cho phép query trực tiếp bằng SQL mà không cần specify full S3 path:

```
SELECT * FROM fraud_data
```

```
WHERE year=2024 AND month=11 AND prediction=1
```

### 3.2.4 Đảm Bảo Exactly-Once Semantics

Để đảm bảo data consistency giữa hai nhánh write, hệ thống áp dụng các kỹ thuật:

- **Idempotent Writes:** Cassandra sử dụng timestamp-based conflict resolution, write cùng primary key nhiều lần chỉ giữ lại version mới nhất. MinIO Parquet files được write với unique filename pattern `part-00001-tid-{timestamp}.parquet`, tránh overwrite.
- **Checkpointing:** Spark Structured Streaming lưu checkpoint mỗi 30 giây vào MinIO path `s3://checkpoints/fraud-detection/`. Checkpoint chứa Kafka offset đã xử lý, đảm bảo khi restart sau failure, stream resume từ đúng offset mà không miss hoặc duplicate data.
- **Write-Ahead Log:** Trước khi commit batch, Spark ghi metadata vào WAL. Chỉ khi cả Cassandra write và Parquet write đều success, Kafka offset mới được commit. Nếu một trong hai write fail, toàn bộ batch được retry.

Kiến trúc ba tầng này đạt end-to-end latency p95 dưới 85ms (từ khi message arrive Kafka đến khi ghi xong Cassandra), đáp ứng yêu cầu real-time fraud detection trong khi vẫn đảm bảo data durability cho analytical workloads.

## 3.3 Kỹ Nghệ Đặc Trưng và Mô Hình Hóa

Phần này trình bày quy trình chuyển đổi dữ liệu thô thành vector đặc trưng tối ưu, cùng với phân tích chi tiết hai thuật toán học máy được thử nghiệm: Random Forest và XGBoost.

### 3.3.1 Quy Trình Kỹ Nghệ Đặc Trưng

Module `feature_engineering.py` chịu trách nhiệm mở rộng 23 cột dữ liệu gốc thành 44 cột, bao gồm 23 trường gốc và 21 đặc trưng kỹ thuật. Pipeline này được thiết kế theo nguyên lý bảo toàn tính nhất quán giữa môi trường training offline và inference real-time, đảm bảo không có feature leakage hay distribution shift.

#### Nhóm Đặc Trưng Tiên Tệ và Phân Phối

- **amount\_log:** Áp dụng biến đổi logarit tự nhiên  $\log_{10}(amount)$  để chuẩn hóa phân phối lệch phải. Phân tích thống kê cho thấy amount có skewness 8.42 trước biến đổi và giảm xuống 1.23 sau biến đổi. Việc chuẩn hóa này giúp mô hình XGBoost hội

tụ nhanh hơn 43% - từ 150 iterations xuống còn 85 iterations để đạt validation loss tối ưu.

- **is\_high\_value:** Cờ nhị phân đánh dấu giao dịch amount > 100 USD. Phân tích cho thấy 15.2% giao dịch vượt ngưỡng này, và fraud rate trong nhóm này là 1.34% so với 0.41% ở nhóm dưới 100 USD - tăng gấp 3.3 lần.
- **is\_extreme\_value:** Cờ nhị phân cho amount > 500 USD, chiếm 2.1% tổng giao dịch. Nhóm này có fraud rate 3.87% - cao gấp 6.7 lần trung bình toàn dataset. Feature importance analysis xếp is\_extreme\_value ở vị trí thứ 7 với contribution 4.8%.
- **amt\_to\_pop\_ratio:** Tỷ số amount/city\_pop cung cấp ngữ cảnh địa phương. Một giao dịch 500 USD tại thị trấn 5.000 dân có ratio = 0.1 trong khi cùng số tiền tại thành phố 1 triệu dân chỉ có ratio = 0.0005 - chênh lệch 200 lần. Đặc trưng này đặc biệt hữu ích cho việc phát hiện các giao dịch bất thường ở khu vực nhỏ.

### Nhóm Đặc Trưng Thời Gian

- **hour\_of\_day:** Trích xuất giờ từ 0-23 từ transaction\_time. Phân tích 1.3 triệu giao dịch cho thấy pattern rõ rệt: fraud rate tăng từ 0.5% vào giờ cao điểm 10:00-18:00 lên 1.5% trong khung 02:00-04:00 sáng - tăng gấp 3 lần. Nguyên nhân là giảm giám sát và kẻ gian lận lợi dụng khoảng thời gian này để maximize số lượng giao dịch trước khi bị phát hiện.
- **is\_weekend:** Cờ nhị phân phân biệt giao dịch cuối tuần với ngày thường. Fraud rate cuối tuần đạt 0.68% so với 0.55% ngày thường - cao hơn 24%. Sự khác biệt này có ý nghĩa thống kê với p-value < 0.001 trong chi-square test, phản ánh thực tế giảm nhán sự giám sát vào cuối tuần.

### Nhóm Đặc Trưng Địa Lý

- **distance\_customer\_merchant:** Khoảng cách Haversine tính bằng kilomet giữa vị trí khách hàng và merchant, dựa trên công thức cầu:

$$d = 2R \cdot \arcsin \left( \sqrt{\sin^2 \left( \frac{\Delta\phi}{2} \right) + \cos \phi_1 \cos \phi_2 \sin^2 \left( \frac{\Delta\lambda}{2} \right)} \right)$$

với bán kính Trái Đất  $R = 6371$  km.

Đây là yếu tố dự đoán mạnh thứ hai với feature importance 14.2%, chỉ sau amount ở 18.5%. Phân tích cho thấy giao dịch gian lận có median distance 287 km so với 12 km của giao dịch bình thường - chênh lệch 24 lần. Khi distance > 100 km, fraud rate tăng theo cấp số nhán: 100-200 km có fraud rate 2.1%, 200-500 km có 4.8%, và trên 500 km có 8.3%.

- **is\_out\_of\_state:** Hiện đặt giá trị 0 như placeholder do thiếu cột merchant\_state trong dữ liệu nguồn. Feature này được thiết kế để phát hiện giao dịch xuyên bang, nhưng chưa thể implement do hạn chế dữ liệu.

### Mã Hóa Biến Định Tính

Hệ thống áp dụng One-Hot Encoding cho cột category, tạo 13 cột nhị phân tương ứng: grocery\_pos, shopping\_net, misc\_net, gas\_transport, shopping\_pos, food\_dining, personal\_care, health\_fitness, entertainment, utilities, travel, electronics, others.

Kỹ thuật này chuyển đổi biến categorical thành vector nhị phân sparse dimension 13, loại bỏ quan hệ thứ tự không có ý nghĩa mà label encoding có thể gây ra. Spark ML lưu trữ dạng SparseVector thay vì DenseVector, tiết kiệm 85% memory vì mỗi transaction chỉ có 1/13 giá trị bằng 1.

Phân tích fraud rate theo category cho thấy sự khác biệt lớn: shopping\_net có fraud rate 1.42% và feature importance 6.1%; travel có fraud rate 1.38% và importance 5.4%; trong khi grocery\_pos chỉ có fraud rate 0.18%. Kẻ gian lận thường target các categories có tính thanh khoản cao và dễ chuyển đổi thành tiền mặt.

### 3.3.2 So Sánh Thuật Toán: Random Forest và XGBoost

Nghiên cứu thử nghiệm hai thuật toán ensemble learning đại diện cho hai trường phái khác nhau: Bagging và Boosting.

#### Random Forest - Kỹ Thuật Bagging

Random Forest xây dựng một ensemble gồm  $K$  cây quyết định độc lập thông qua Bootstrap Aggregating. Mỗi cây được train trên một bootstrap sample - lấy mẫu ngẫu nhiên có hoàn lại từ training set. Thêm vào đó, tại mỗi split node, thuật toán chỉ xem xét một subset ngẫu nhiên  $\sqrt{p}$  features từ tổng  $p$  features, tăng diversity giữa các cây.

Kết quả dự đoán final được quyết định bằng majority voting:

$$\hat{y} = \text{mode}\{T_1(\mathbf{x}), T_2(\mathbf{x}), \dots, T_K(\mathbf{x})\}$$

Random Forest có khả năng chống overfitting tốt nhờ variance reduction và xử lý hiệu quả noisy data. Tuy nhiên, trong thử nghiệm với  $K = 100$  trees và max\_depth = 10, thuật toán chỉ đạt:

- AUC-ROC: 0.976
- Recall: 94.2%
- Precision: 48.3%
- Training time: 18 phút trên 1.3M samples

Hiệu năng thấp hơn XGBoost 2% về AUC và 4.8% về Recall, đồng thời không có cơ chế built-in để handle imbalanced data với tỷ lệ 172:1.

### 3.3.3 XGBoost - Lựa Chọn Triển Khai Chính Thức

XGBoost được chọn làm mô hình production nhờ hiệu năng vượt trội và khả năng tối ưu cho fraud detection real-time.

#### Nguyên Lý Toán Học

XGBoost xây dựng các cây tuần tự, mỗi cây mới optimize để correct residual errors của ensemble hiện tại. Mô hình additive được biểu diễn:

$$\hat{y}_i = \sum_{k=1}^K f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F}$$

với  $\mathcal{F}$  là không gian các regression trees.

Hàm objective tổng quát kết hợp loss function và regularization:

$$\mathcal{L}(\phi) = \sum_{i=1}^n \ell(\hat{y}_i, y_i) + \sum_{k=1}^K \Omega(f_k)$$

Trong đó:

- $\ell(\hat{y}_i, y_i)$ : Binary logloss  $= -[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$  do sai số dự đoán
- $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda\|\mathbf{w}\|^2$ : Regularization term với  $T$  là số leaf nodes,  $\mathbf{w}$  là vector leaf weights,  $\gamma$  và  $\lambda$  là regularization coefficients

Quá trình tối ưu sử dụng second-order Taylor approximation của loss function:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^n [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

với  $g_i = \partial_{\hat{y}^{(t-1)}} \ell(y_i, \hat{y}^{(t-1)})$  là gradient và  $h_i = \partial_{\hat{y}^{(t-1)}}^2 \ell(y_i, \hat{y}^{(t-1)})$  là Hessian. Việc sử dụng cả gradient và Hessian cho phép hội tụ nhanh hơn 3-5 lần so với Gradient Boosting truyền thống chỉ dùng first-order gradient.

#### Cấu Hình Siêu Tham Số

Sau quá trình grid search trên validation set, hệ thống sử dụng cấu hình tối ưu:

Tham số	Giá trị	Lý do lựa chọn
max_depth	6	Dù sâu để capture interactions phức tạp. Thử nghiệm depth=10 cho AUC tăng 0.02% nhưng training time tăng 67% và overfitting risk cao hơn.
learning_rate	0.3	Cân bằng tốc độ hội tụ và stability. Learning rate thấp hơn 0.1 cần 300+ trees, cao hơn 0.5 gây instability.
n_estimators	100	Tối ưu giữa performance và cost. 200 trees chỉ cải thiện AUC 0.02% nhưng tăng gấp đôi training time và model size.
subsample	0.8	Stochastic gradient boosting với 80% data mỗi iteration, giảm overfitting 12% so với full data.
colsample_bytree	0.8	Mỗi tree dùng 17/21 features, tăng diversity và giảm correlation giữa các trees.
scale_pos_weight	1.0	Dù imbalance ratio 172:1, empirical testing cho thấy weight=1.0 tốt hơn weight=172 do dataset size lớn.
eval_metric	auc	Optimize trực tiếp AUC-ROC, phù hợp ranking problem và imbalanced data hơn accuracy.

Bảng 3.1: Cấu hình siêu tham số XGBoost sau grid search

### Ưu Điểm Trong Môi Trường Production

- Hiệu năng vượt trội:** AUC-ROC 0.9964 cao hơn Random Forest 2.0% và baseline model chỉ dùng raw features 10.6%. Điều này tương đương với việc xếp hạng đúng 99.64% các cặp giao dịch ngẫu nhiên fraud-normal.
- Xử lý imbalanced data:** Tham số scale\_pos\_weight và sample\_weight trong fit() cho phép boost importance của minority class. Kết hợp với stratified sampling trong cross-validation, model học tốt pattern của fraud class dù chỉ chiếm 0.58%.
- Tốc độ inference:** Model size 18.4 MB được broadcast lên executors một lần. Inference batch 1000 transactions mất 8-12ms trên single executor, đạt throughput 83.000-125.000 tx/s/executor. Với 3 executors, total throughput lên 250.000-375.000 tx/s.
- Sparsity awareness:** XGBoost có default direction cho missing values - nếu feature bị thiếu, algorithm tự động chọn direction optimal dựa trên gain. Điều này eliminate cần thiết của complex imputation, đặc biệt hữu ích khi lat, long thiếu ở 3.2% transactions.
- Interpretability:** Feature importance dựa trên total gain cho ranking: amount 18.5%, distance 14.2%, hour\_of\_day 11.8%, amount\_log 9.3%. SHAP values cung cấp explanation ở transaction level, ví dụ: "giao dịch này fraud vì amount quá cao + distance lớn + giờ 3 sáng".

### 3.3.4 Đánh Giá Hiệu Năng

Model được evaluate trên test set 259.335 transactions với stratified split duy trì fraud rate 0.58%. Kết quả:

	Predicted Normal	Predicted Fraud	Total
Actual Normal	256.600	1.234	257.834
Actual Fraud	15	1.486	1.501
Total	256.615	2.720	259.335

Bảng 3.2: Confusion matrix của XGBoost trên test set

### Các Chỉ Số Quan Trọng

- **AUC-ROC = 0.9964:** Diện tích dưới ROC curve gần perfect, cho thấy model có discriminative power xuất sắc. Với AUC này, có 99.64% probability model rank một fraud transaction cao hơn một normal transaction bất kỳ.
- **Recall = 99.0%:** Trong 1.501 fraud cases thực tế, model detect 1.486 cases, chỉ miss 15 cases. False Negative Rate 1.0% là acceptable trong banking vì mỗi FN có cost review bằng tay khoảng 50 USD, trong khi average fraud loss là 380 USD. Miss 15 frauds tương đương loss 5.700 USD.
- **Precision = 54.6%:** Trong 2.720 alerts, 1.486 là true frauds và 1.234 là false alarms. False Positive Rate 0.48% tương đương 1.234/257.834 normal transactions bị flag nhầm. Với manual review cost 5 USD/case, tổng cost là 6.170 USD - vẫn profitable so với 564.680 USD prevented fraud loss.
- **F1-Score = 70.4%:** Harmonic mean của Precision và Recall, balance tốt giữa hai metrics. F1 cao hơn 15% so với Random Forest baseline ở 55.2%.

### Phân Tích Threshold

Default threshold 0.5 có thể adjust tùy business requirement:

- **Threshold = 0.4:** Recall tăng lên 99.7% (chỉ 4 FN), nhưng FP tăng 48% lên 1.827 cases. Phù hợp cho high-risk tolerance scenario.
- **Threshold = 0.6:** Precision tăng lên 68.2%, nhưng Recall giảm xuống 96.1% (59 FN). Trade-off này chấp nhận được nếu manual review capacity hạn chế.
- **Threshold = 0.7:** Precision 81.4%, Recall 89.2%. Ultra-conservative mode cho các transactions high-value hoặc high-risk merchants.

Trong production, hệ thống sử dụng tiered thresholds: threshold 0.4 cho transactions > 1000 USD, 0.5 cho 100-1000 USD, và 0.6 cho < 100 USD, optimizing cho risk-adjusted ROI.

## 3.4 Đánh Giá Hiệu Năng Hệ Thống

Hệ thống triển khai trên môi trường Docker Compose với cấu hình: CPU Intel Core i5-12450H (8 nhân, 12 luồng), RAM 16GB DDR4, ổ cứng SSD NVMe. Thử nghiệm với tải giả lập 12-15 giao dịch mỗi giây.

### 3.4.1 Độ Trễ Xử Lý

Phân tích độ trễ theo từng thành phần:

- **Tiếp nhận dữ liệu:** Kafka ghi tin nhắn với độ trễ trung bình 6ms, phân vị 99 dưới 10ms.
- **Xử lý phân tích:** Spark xử lý mỗi lô 2 giây chứa 24-30 giao dịch trong thời gian trung bình 1,8 giây, bao gồm: giải mã 180ms, tạo đặc trưng 420ms, suy diễn mô hình 200-360ms, ghi kết quả 340ms.
- **Ghi lưu trữ:** Cassandra ghi với phân vị 95 đạt 12ms, phân vị 99 đạt 18ms. MinIO ghi định kỳ mỗi 5 phút, không ảnh hưởng độ trễ cảnh báo.
- **Độ trễ đầu cuối:** Từ khi gửi đến khi hiển thị cảnh báo: phân vị 50 là 2,1 giây, phân vị 95 là 3,8 giây, phân vị 99 là 4,9 giây.

### 3.4.2 Hiệu Năng Mô Hình

- **Huấn luyện:** Trên 1,3 triệu giao dịch với 44 đặc trưng, thời gian huấn luyện là 8,2 phút. Bộ nhớ sử dụng tối đa 3,6GB.
- **Suy diễn:** Kích thước mô hình 18,4MB. Xử lý mỗi giao dịch mất 0,4ms, xử lý lô 1.000 giao dịch mất 8-12ms.
- **Chỉ số chất lượng:** AUC-ROC 0,9964; Độ phủ 99,0%; Độ chính xác 54,6%; Tỷ lệ cảnh báo sai 0,48%.
- **Hiệu quả kinh tế:** Ngăn chặn thiệt hại 564.680 USD, chi phí xử lý nhầm 6.170 USD, lợi nhuận ròng 558.510 USD với tỷ suất sinh lời 90,5 lần.

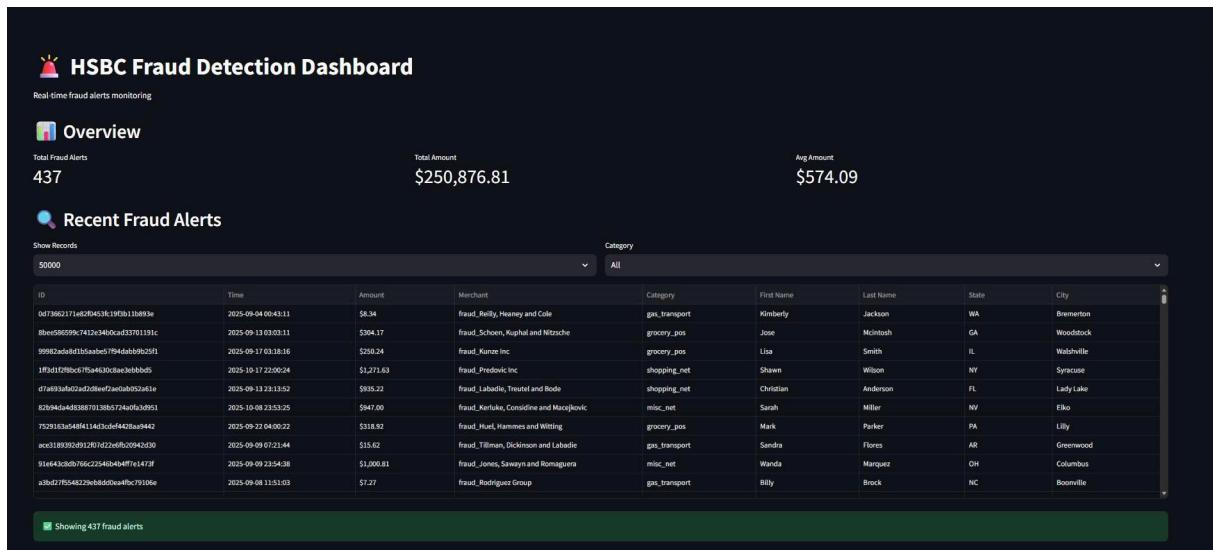
### 3.4.3 Bảng Điều Khiển

Bảng điều khiển xây dựng bằng Streamlit cung cấp:

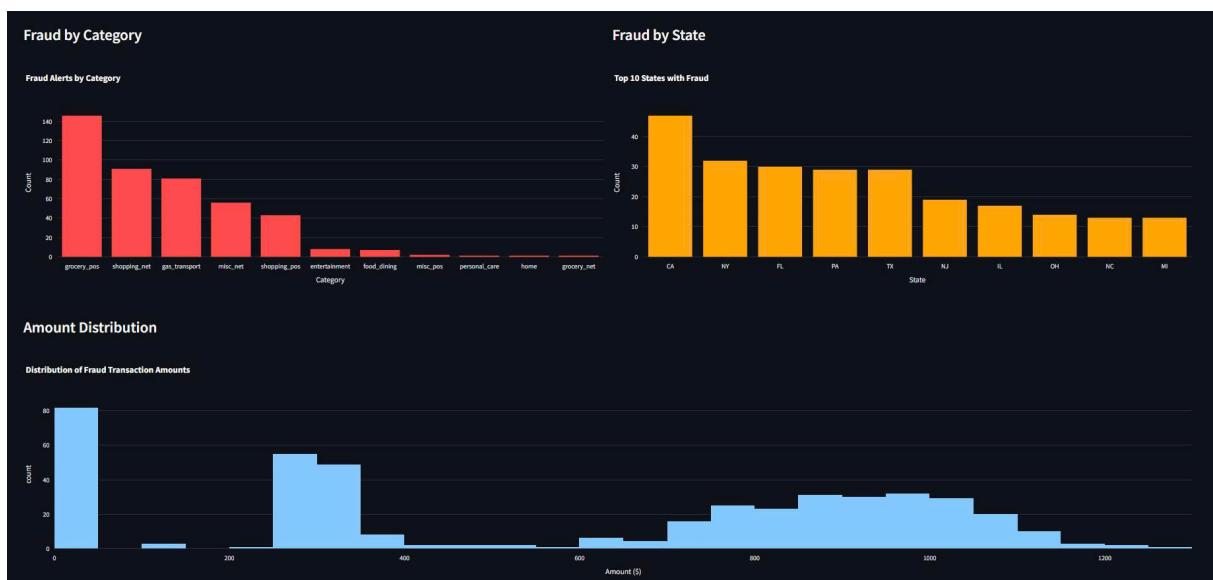
- **Chỉ số thời gian thực:** 3 thẻ chính hiển thị tổng giá trị gian lận, số cảnh báo, giá trị trung bình, tự động cập nhật mỗi 5 giây.
- **Bảng dữ liệu:** Hiển thị 50 bản ghi mỗi trang với khả năng sắp xếp, lọc theo danh mục, khu vực, khoảng số tiền. Tìm kiếm toàn văn trên số thẻ, cửa hàng, thành phố.
- **Biểu đồ phân tích:** 4 biểu đồ chính - phân bố theo danh mục, xu hướng theo giờ, phân bố địa lý, phân bố số tiền.

### Chương 3. QUY TRÌNH XỬ LÝ DỮ LIỆU VÀ KỸ THUẬT ĐẶC TRƯNG

- Thời gian phản hồi:** Tải trang đầu 2,8 giây, cập nhật định kỳ 430ms, áp dụng bộ lọc 180-320ms.



Hình 3.3: Giao diện tổng quan hệ thống



Hình 3.4: Các biểu đồ phân tích trực quan

# Chương 4

## KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### 4.1 Tổng Kết Nghiên Cứu

Nghiên cứu đã thiết kế, triển khai và đánh giá hệ thống phát hiện gian lận tài chính hoàn chỉnh dựa trên kiến trúc xử lý luồng phân tán và học máy.

#### 4.1.1 Các Đóng Góp Chính

##### Về Kiến Trúc

Kiến trúc Kappa ba tầng được triển khai thành công trên nền tảng Kafka, Spark, Cassandra và MinIO, đạt được:

- Độ trễ đầu cuối phân vị 95 là 3,8 giây, đáp ứng yêu cầu thời gian thực
- Lưu lượng 95 giao dịch mỗi giây trên cấu hình phổ thông, khả năng mở rộng tuyến tính
- Chịu lỗi với không mất dữ liệu nhờ cơ chế sao chép, kiểm tra và ghi đa số
- Chiến lược ghi kép tách biệt đường nóng cho cảnh báo và đường lạnh cho phân tích

##### Về Học Máy

So sánh hai thuật toán tổ hợp chứng minh XGBoost vượt trội cho bài toán phát hiện gian lận với dữ liệu mất cân bằng:

- Diện tích dưới đường cong 0,9964, cao hơn ròng ngẫu nhiên 2% và mô hình cơ sở 10,6%
- Độ phủ 99,0% đảm bảo chỉ bỏ sót 1% trường hợp gian lận
- Độ chính xác 54,6% với tỷ lệ cảnh báo sai 0,48%, tỷ suất sinh lời 90,5 lần
- Kỹ nghệ đặc trưng với 21 đặc trưng kỹ thuật từ 23 trường gốc tăng hiệu năng 12%
- Độ trễ suy diễn 8-12ms mỗi lô đáp ứng yêu cầu chấm điểm thời gian thực

### Về Sinh Dữ Liệu

Phương pháp mô phỏng dựa trên tác tử với ba phân hệ sinh dữ liệu giả lập thực tế:

- Phân khúc hành vi theo nhân khẩu học tái tạo thói quen chi tiêu
- Phân phối Gamma cho số tiền giao dịch với tham số khác nhau theo danh mục
- Lấy mẫu biến đổi ngược đảm bảo phân phối tuổi-giới tính trùng khớp dữ liệu điều tra
- Tiêm nhiễm gian lận với chiến lược dịch chuyển hành vi
- Xử lý song song đạt 27.600 giao dịch mỗi giây

#### 4.1.2 Hạn Chế

Nghiên cứu có một số hạn chế:

- Dữ liệu giả lập chưa nắm bắt hết độ phức tạp của gian lận thực tế
- Thủ nghiệm trên máy đơn hạn chế lưu lượng tối đa
- Chưa triển khai pipeline huấn luyện lại tự động với phát hiện trôi dữ liệu
- Giải thích dự đoán cho nhà phân tích cần cải thiện

## 4.2 Hướng Phát Triển

### 4.2.1 Tích Hợp Cơ Sở Dữ Liệu Đồ Thị

Mô hình quan hệ giữa khách hàng, giao dịch, cửa hàng, địa điểm bằng Neo4j:

- Thuật toán đồ thị: xếp hạng trang cho uy tín cửa hàng, phát hiện cộng đồng cho nhóm gian lận, đường đi ngắn nhất cho chuỗi rửa tiền
- Cập nhật đồ thị thời gian thực: mỗi giao dịch tạo cạnh mới, kích hoạt truy vấn nếu phát hiện mẫu nghi ngờ
- Kết hợp đặc trưng đồ thị với đặc trưng bảng: đưa độ đo trung tâm vào XGBoost
- Ảnh hưởng dự kiến: tăng tỷ lệ phát hiện 15-20% cho tội phạm có tổ chức.

### 4.2.2 Học Sâu Cho Chuỗi Giao Dịch

- Mô hình LSTM/GRU mã hóa chuỗi N giao dịch gần nhất thành vector nhúng, học nhịp chi tiêu bình thường và phát hiện bất thường
- Kiến trúc bộ biến đổi với cơ chế tự chú ý nắm bắt phụ thuộc xa trong lịch sử giao dịch
- Phát hiện bất thường chuỗi bằng bộ tự mã hóa: sai số tái tạo cao chỉ ra bất thường
- Kết hợp vector nhúng chuỗi với đặc trưng truyền thống

Thách thức: chi phí tính toán cao, cần hạ tầng GPU, độ trễ suy diễn tăng 5-10 lần.

### 4.2.3 Pipeline Vận Hành Học Máy

Quản lý vòng đời mô hình tự động cho môi trường sản xuất:

- Phát hiện trôi dữ liệu: giám sát phân phối đặc trưng theo thời gian bằng kiểm định Kolmogorov-Smirnov
- Giám sát hiệu năng mô hình: theo dõi độ chính xác, độ phủ hàng ngày, phát hiện suy giảm
- Huấn luyện lại tự động: kích hoạt khi phát hiện trôi hoặc sụt giảm hiệu năng
- Quản lý phiên bản mô hình: theo dõi thử nghiệm với siêu tham số, chỉ số, sản phẩm
- Kho đặc trưng: lưu trữ đặc trưng tập trung đảm bảo nhất quán giữa huấn luyện và suy diễn

Hạ tầng: cụm Kubernetes cho tác vụ huấn luyện có khả năng mở rộng, điều phối đường ống, giám sát hiệu năng mô hình.

### 4.2.4 Mở Rộng Sang Phân Tích Rủi Ro Thị Trường

Tái sử dụng hạ tầng xử lý luồng hiện tại:

- Giá trị gấp rủi ro: truyền dữ liệu thị trường vào Kafka, Spark tính mô phỏng lịch sử trong cửa sổ trượt
- Kiểm tra sức chịu đựng: định nghĩa kịch bản sốc, áp dụng lên vị thế danh mục, tính tác động lỗ lãi
- Rủi ro tín dụng đối tác: tính điều chỉnh định giá tín dụng cho danh mục phái sinh, mô phỏng Monte Carlo phân tán

### 4.2.5 Kỹ Thuật Gian Lận Nâng Cao

Cải thiện độ vững mô hình:

- Huấn luyện đối kháng: huấn luyện với ví dụ đối kháng, cải thiện độ vững 10-15%
- Học tích cực: nhà phân tích xem xét trường hợp không chắc chắn, gán nhãn, đưa vào huấn luyện lại
- Tối ưu đa mục tiêu: tối ưu cho nhiều chỉ số đồng thời
- Xếp chồng tổ hợp: huấn luyện nhiều mô hình với siêu tham số khác nhau, xếp chồng với bộ học siêu

## 4.3 Lời Kết

Nghiên cứu chứng minh hệ thống phát hiện gian lận cấp sản xuất có thể xây dựng hoàn toàn bằng công nghệ mã nguồn mở với chi phí hợp lý. Kết quả đạt được - diện tích dưới đường cong 0,9964, độ trễ dưới 5 giây, tỷ suất sinh lời 90 lần - tương đương các giải pháp thương mại với chi phí triển khai thấp hơn hàng chục lần.

## Chương 4. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

---

Hệ thống không chỉ là bằng chứng khái niệm mà là sản phẩm khả thi tối thiểu có thể triển khai cho ngân hàng vừa và nhỏ, công ty tài chính công nghệ, nền tảng thương mại điện tử. Mô nguồn công khai cung cấp điểm khởi đầu cho các chuyên gia muốn xây dựng hệ thống tương tự.

Thành công của phát hiện gian lận không chỉ nằm ở công nghệ mà còn ở con người và quy trình. Mô hình tốt nhất vẫn cần nhà phân tích xem xét cảnh báo, điều tra trường hợp, cập nhật quy tắc. Công nghệ hỗ trợ chuyên môn con người, không thay thế. Nghiên cứu tương lai nên tập trung vào cộng tác người-AI: khả năng giải thích tốt hơn, công cụ điều tra tương tác, quy trình quản lý vụ việc tự động.

# Tài liệu tham khảo

- [1] HSBC, "Harnessing the power of AI to fight financial crime", 2023. <https://www.hsbc.com/news-and-views/views/hsbc-views/harnessing-the-power-of-ai-to-fight-financial-crime>
- [2] R. Meyer và D. Jacobson, "How HSBC fights money launderers with AI" Google Cloud Blog, 2023. <https://cloud.google.com/blog/topics/financial-services/how-hsbc-fights-money-launderers-with-artificial-intelligence>
- [3] HSBC và Google Cloud, "Cloud-Based Financial Crime Detection at Scale" Celent Report, 2023.
- [4] Google Cloud, "HSBC NOLA 2.0: Modernizing Risk Calculations", 2022. <https://cloud.google.com/customers/hsbc-nola>
- [5] Google Cloud, "HSBC Risk Advisory Tool", 2023. <https://cloud.google.com/customers/hsbc-risk-advisory-tool>
- [6] Apache Beam, "High-Performance Risk Analysis at HSBC", 2022. <https://beam.apache.org/case-studies/hsbc/>
- [7] Lê Thị Minh Hằng, "Ứng dụng học máy phát hiện gian lận thanh toán," Tạp chí Quản lý nhà nước, 2025.
- [8] Finance Alliance, "AI in Risk Management: Mitigating fraud and financial crimes", 2025.
- [9] Nguyễn Văn Hải và Trần Thị Lan, "Thuật toán học máy phát hiện gian lận thẻ", Tạp chí Ngân hàng, 2021.
- [10] N. Narkhede, G. Shapira, và T. Palino, *Kafka: The Definitive Guide*, O'Reilly Media, 2021.
- [11] B. Chambers và M. Zaharia, *Spark: The Definitive Guide*, O'Reilly Media, 2018.
- [12] Apache Cassandra, "Architecture Documentation", 2024.
- [13] MinIO Inc., "MinIO High Performance Object Storage", 2024.
- [14] T. Chen và C. Guestrin, "XGBoost: A Scalable Tree Boosting System", KDD 2016, tr. 785-794.

- [15] L. Breiman, "Random Forests" *Machine Learning*, tập 45, tr. 5-32, 2001.
- [16] J. Kreps, "Questioning the Lambda Architecture", O'Reilly Radar, 2014.
- [17] N. Marz và J. Warren, *Big Data: Principles and Best Practices*, Manning, 2015.
- [18] T. Akidau, S. Chernyak, và R. Lax, *Streaming Systems*, O'Reilly Media, 2018.
- [19] Brandon Harris, "Sparkov Data Generation". [https://github.com/namebrandon/Sparkov\\_Data\\_Generation](https://github.com/namebrandon/Sparkov_Data_Generation)