WARSAW UNIVERSITY OF TECHNOLOGY

FACULTY OF MATHEMATICS AND INFORMATION SCIENCE

Real-time fraudulent transactions detection

Big Data Analytics

Salveen Singh Dutt (317298)

Karina Tiurina (335943)

Mikołaj Malec (298828)

Patryk Prusak (305794)

Supervisor

mgr inz. Jakub Abelski

Warsaw 2024

Contents

Introduction

The goal of this project is to plan and implement financial transactions processing system which identifies suspicious and fraudulent activity in real-time. Given the large volume of incoming data, the project will utilize big data technologies as well as advanced machine learning algorithms for anomaly detection.

GitHub repository: https://github.com/salveendutt/Big-Data-Analytics.

Updates from Milestone 1

The project architecture layout was updated to better follow the Lambda architecture. In particular:

- master data storage was moved to the Batch layer;

- included NoSQL database as a storage for data querying in the Serving layer;

- Hive has been changed to Cassandra as the main storage solution;

- the architecture schema was updated to better represent the data flow.

1 High level description

The main idea is to implement an automatic transactions processing so that anytime a fraudulent activity occurs, the transaction is blocked for further manual review. The aim is to reduce financial losses of the end-users and to enhance the security of online payment, ensuring a safer experience for all customers.

There are two main end-users of the project: financial institutions (we will call them 'Managers') and their customers executing the payments. Although both categories can benefit from the solution, in our implementation we will mainly focus on Managers to limit additional data in storage.

The list below contains main features that we expect to implement for Managers:

1. Fraudulent transactions are automatically highlighted so that it is easier to identify suspicious activity;

2. The history of transactions is stored and available for later review;

3. A dashboard with statistics of fraudulent activity is available and customizable for better localisation of issues (e.g. too large amount, unusual location);

4. Anomaly-detection model is continuously updated so that fraud detection utilizes new historical data and is more accurate on future transactions;

5. Data streaming processing and batch jobs are customizable so that the testing of model's performance is simplified.

2 Data sources

Due to strict security regulations on personal and financial data, it is quite challenging to find open source real transactions data both for model training and streaming. Therefore, available synthetic and anonymized datasets will be used. The table below contains description of the data sources. Each data source is described in more detail in the dedicated subsections.

Data Source	Content	Volume	Fraud, %	Link
1. Fraudulent Transactions Data	Dataset for predicting fraudulent transactions for a financial company.	6,362,620 rows and 10 columns (493.53 MB)	0.13%	Kaggle
2. Credit Card Fraud	Contains features with transactional context.	1,000,000 transactions (58.9 MB)	8.7%	OpenML
3. Credit Card Transactions Synthetic Data Generation	A collection of synthetic credit card transaction data.	1,785,308 transactions; 5,000 customers; (153.66 MB)	3%	Kaggle
4. Credit Card Fraud Detection	Transactions made by credit cards in September 2013 by European cardholders.	284,807 transactions (150.83 MB)	0.17%	Kaggle

Table 1: Data sources

Data-streaming API was implemented from scratch. The assumption was that it would use the above datasets; with a specified time-frame, it would choose a random transaction which was not used for training and push it for further processing. For the testing purposes, the probability of a fraudulent transaction will be set manually to some high enough constant value. Details description of the implented streaming API can be found in chapter 3.

The content of the datasets is extremely different which makes it impossible to combine them into a single dataset. Therefore, each dataset will be treated separately for both streaming and ML training.

2.1 Fraudulent Transactions Data (Kaggle)

Dataset contains transactions for a financial company, indicating whether it is fraudulent or not. Data for the case is available in CSV format having 6362620 rows and 10 columns. Full column description is presented in the table 2.

The following transformation should be done to pass dataset to the ML models:

Column	Content	Type
step	maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation)	int
type	type of the transaction. Available values: CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER	str
amount	amount of the transaction in local currency	float
nameOrig	customer who started the transaction	str
oldbalanceOrg	initial balance before the transaction	float
newbalanceOrig	new balance after the transaction	float
nameDest	customer who is the recipient of the transaction	str
oldbalanceDest	initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants)	float
newbalanceDest	new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants)	float
isFraud	this is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control or customers accounts and try to empty the funds by transferring to another account and then cashing out of the system	int
isFlaggedFraud	the business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction	int

Table 2: Columns description. Dataset 1: 'Fraudulent Transactions Data' from Kaggle

1. 'type' column values CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER transformed to 1, 2, 3, 4 and 5 respectively;

2. a new attribute 'isMerchant' is calculated. 1 if 'nameDest' starts with 'M', 0 - otherwise.

Generally, the dataset and clean and structured, no other preprocessing, except for the described above is necessary. Potential issue is that it is highly unbalanced (less than 1% of fraud transactions), and contains quite small amount of features (6) for model training.

2.2 Credit Card Fraud (OpenML)

This dataset captures transaction patterns and behaviors that could indicate potential fraud in card transactions. The data is composed of several features designed to reflect the transactional context such as geographical location, transaction medium, and spending behavior relative to the user's history.

The whole dataset is in the numeric form, therefore, no additional preprocessing is necessary. The only transformation is to rename the target variable to 'isFraud' To match other datasets format. The target feature balance is much better than for the previous dataset: 8.7%

Column	Content	Type
distance_from_home	This is a numerical feature representing the geographical distance in kilometers between the transaction location and the cardholder's home address.	float
distance_from_last_transaction	This numerical attribute measures the distance in kilometers from the location of the last transaction to the current transaction location.	float
ratio_to_median_purchase_price	A numeric ratio that compares the transaction's price to the median purchase price of the user's transaction history.	float
repeat_retailer	A binary attribute where '1' signifies that the transaction was conducted at a retailer previously used by the cardholder, and '0' indicates a new retailer.	[0, 1]
used_chip	This binary feature indicates whether the transaction was made using a chip (1) or not (0).	[0, 1]
used_pin_number	Another binary feature, where '1' signifies the use of a PIN number for the transaction, and '0' shows no PIN number was used.	[0, 1]
online_order	This attribute identifies whether the purchase was made online ('1') or offline ('0').	[0, 1]
fraud	A binary target variable indicating whether the transaction was fraudulent ('1') or not ('0').	[0, 1]

Table 3: Columns description. Dataset 2: 'Credit_Card_Fraud_' from OpenML

of fraud. Additional complication is that feature 'amount' is not available. A ratio to the median purchase of the same customer is provided instead.

2.3 Credit Card Transactions Synthetic Data Generation (Kaggle)

This dataset is a collection of synthetic credit card transaction data. The data is designed to mimic the characteristics of real credit card transactions while ensuring privacy and compliance with data protection regulations such as the General Data Protection Regulation (GDPR). It contains 1,785,308 transactions for 5000 customers.

Like other datasets, it is quite unbalanced with 3% of fraudulent transactions. The following preprocessing should be done, before passing rows to ML task:

1. 'entry_mode' column values Contactless, Chip and Swipe transformed to 1, 2, and 3 respectively;

Column	Content	Type
transaction_id	Random string containing specific transactions id	str
post_ts	Date and time of the transaction	str
customer_id	Specific customer id	str
bin	Bank Identification Number	int
terminal_id	Specific terminal id	int
amt	Transaction amount	float
entry_mode	Mode of the transaction. Possible values are Contactless, Chip and Swipe.	str
fraud	Target variable containing 1 for fraudulent transaction and 0 otherwise	int
fraud_scenario	Additional label for the transaction. 97% of the dataset has value 0. No specific description for each scenario is provided.	int, [0, 1, 2]
mean_amount	Average transaction amout for a specific customer	float
std_amount	Standard deviation of the transaction amout for a specific customer	float
mean_nb_tx_per_day	Mean number of transactions per day for a specific customer	float
customer_bin	Bank Identification Number of a customer	int

Table 4: Columns description. Dataset 3: 'Credit Card Transactions Synthetic Data Generation' from Kaggle

2. 'amt' renamed to 'amount',

3. 'fraud' renamed to 'isFraud',

4. transaction data itself contains only customer id. Therefore, an additional step is to find the customer and add related features to the output.

2.4 Credit Card Fraud Detection (Kaggle)

The dataset contains transactions made by credit cards in September 2013 by European cardholders. It contains 284,807 transactions with only 492 (less than 1%) of fraud ones. Due to confidentiality issues, the dataset contains only numerical input variables which are the result of a PCA transformation.

This dataset contains relatively small amount of data compared to others (about 280k of transactions). Therefore, we are planning to keep it as an additional dataset in case of any issues with others.

Since the data has already been processed, the only necessary transormation is to rename columns 'Amount' and 'Class' to 'amount', 'isFraud' respectively.

Column	Content	Type
Time	The seconds elapsed between the transaction and the first transaction in the dataset	int
V1 ... V28	The principal components obtained with PCA. The original features and more background information about the data are not provided.	float
Amount	Transaction amount	float
Class	Target variable; 1 for fraudulent transaction and 0 otherwise	int, [0, 1]

Table 5: Columns description. Dataset 4: 'Credit Card Fraud Detection' from Kaggle

3 Data acquisition strategy

Due to the lack of publicly available open streaming APIs, we designed and implemented a custom streaming API for real-time fraud detection. Stream API is connected to the NiFi for futher data collection and preprocessing. The following technological stack is used for the data acquisition:

1. Python,

2. Flask - for the stream API,

3. Apache NiFi - for the data collection,

4. Docker - for deployment,

5. Apache Kafka - for the data streaming,

6. Kafdrop - for Kafka monitoring,

7. Apache Cassandra - as a main storage solution

8. Zookeeper.

When server is started, stream api is avalable on localhost:5000/data/:dataset_id. Data frequency is configurable on the NiFi side and at the moment is set to 1 row per 2 seconds for each dataset. Format of each incoming transaction is JSON, containing attributes as described for each dataset in chapter 2. To simulate real-world conditions, the streaming API randomly (10%) returns a 500 error. Example screenshots of the data stream and NiFi are provided in the SKaMP_Tests.pdf file.

Below we present the NiFi flows for the data acquisition. The first flow is the overview of the whole process, the three processor groups are responsible for fetching and processing the data. Inside each of the processor groups there are:

• InvokeHTTP - to fetch the data from the stream API,

• EvaluateJsonPath - to extract the data from the JSON format,

• ReplaceText - to construct an CQL query for the Cassandra,

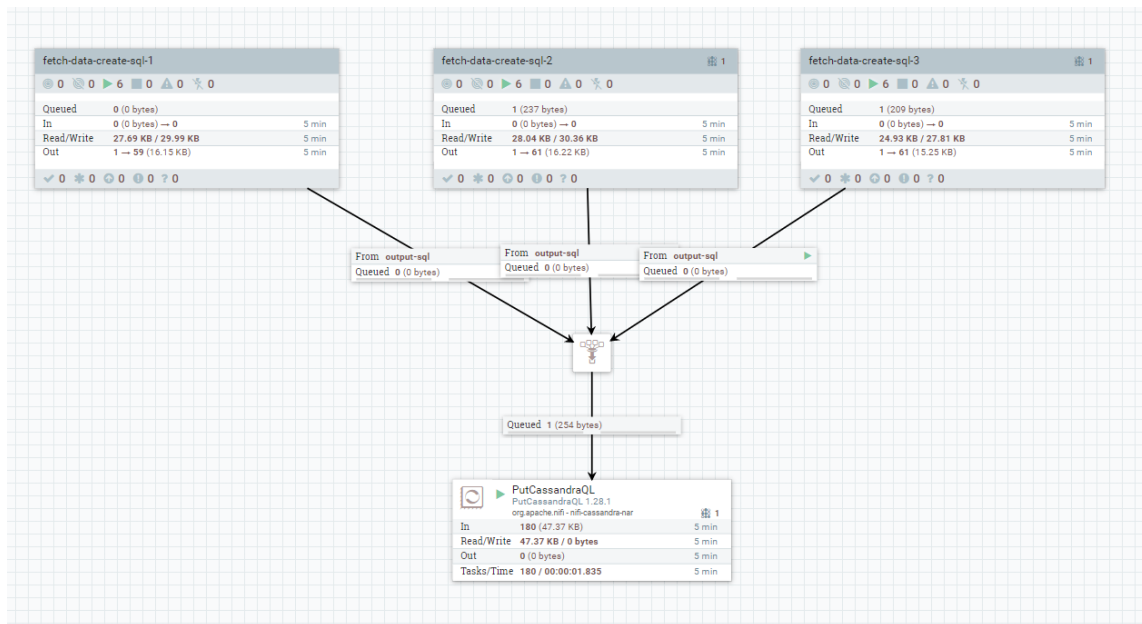• PublishKafkaRecord - to send the data to the Kafka.

Figure 1: NiFi flow for the data acquisition

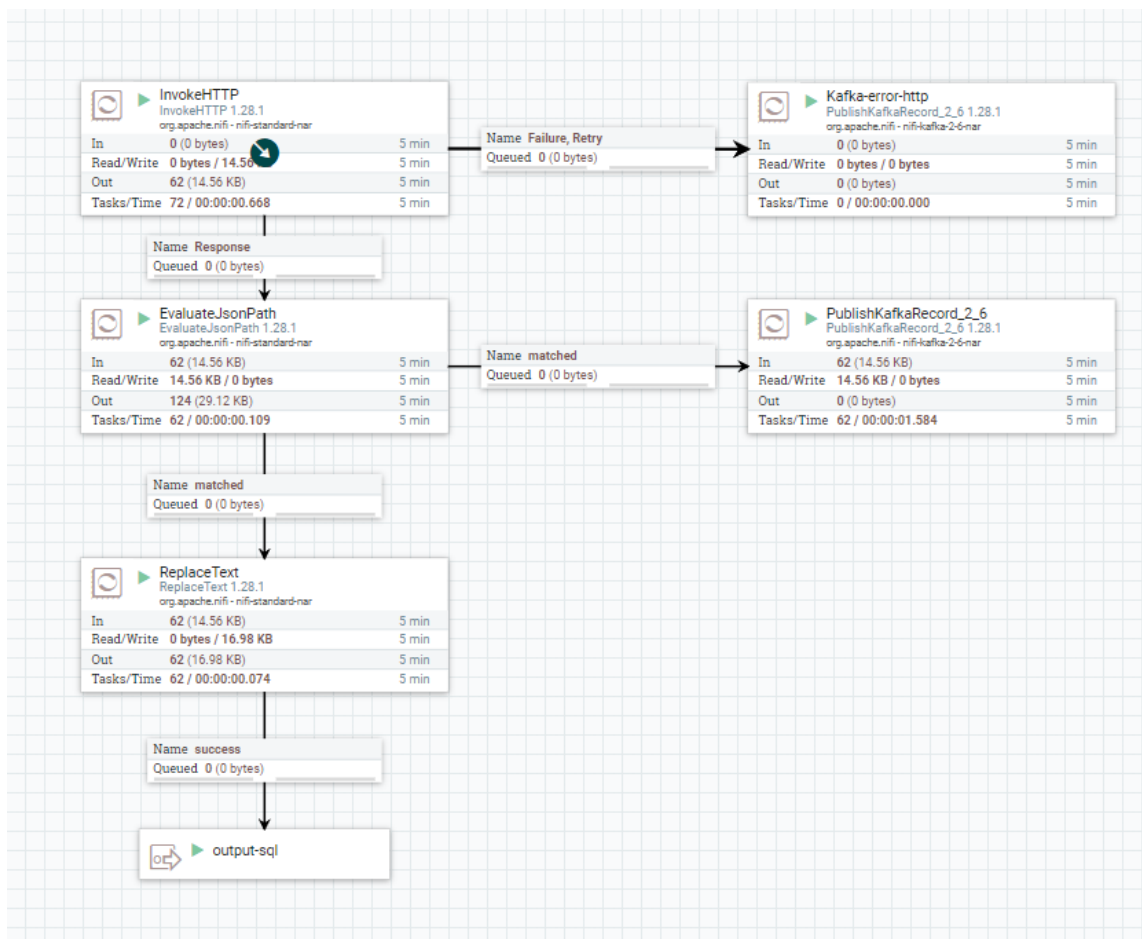Figure 2: Data processing flow in NiFi

4   Data storage strategy

Our input data consists purely of a structured data and there is no need to store special data types, such as images, text files, audio and etc. Therefore, we decided to utilize SQL-like (CQL) data warehouse system that enables analytics at a massive scale - Apache Cassandra.

As described in chapter 2, our datasets contain completely different sets of features which makes it impossible to combine them in a single table. All incoming transactions will be divided into 4 groups for each dataset. Code block below represents an example of table creation for dataset 1 of the project.

Listing 1: Apache Cassandra table creation

```
CREATE TABLE dataset1 (
    step INT,
    type TEXT,
    amount DECIMAL,
    isFlaggedFraud INT,
    isFraud INT,
    nameDest TEXT,
    nameOrig TEXT,
    newbalanceDest DECIMAL,
    newbalanceOrig DECIMAL,
    oldbalanceDest DECIMAL,
    oldbalanceOrg DECIMAL,
    PRIMARY KEY (step, nameOrig)
);
```

Preliminary list of all tables in the storage is the following:

1. dataset1 - raw data for the dataset 1;

2. dataset1_processed - processed data for the dataset 1;

3. dataset2 - raw data for the dataset 2;

4. dataset2_processed - processed data for the dataset 2;

5. dataset3 - raw data for the dataset 3;

6. dataset3_processed - processed data for the dataset 3;

7. dataset4 - raw data for the dataset 4;

8. dataset4_processed - processed data for the dataset 4.

Additionally, we will be using MongoDB for storing prepared views for fast querying during the data presentation. Typically, it will replicate the processed data from both batch and streaming layers. This includes predicted labels for new transactions as well as indication whether data was used for training.

5 Project architecture

We are planning to implement the project based on Lambda Architecture. The main data processing will be divided into three layers:

1. Speed Layer (streaming)

   - Data preprocessing including transformation to a specific format;
   - Real-time fraud detection on all of the incoming transactions.

2. Batch Layer

   - Data processing and filtering for the model training
   - ML model training with a fixed schedule (e.g. every 10 minutes)

3. Serving Layer

   - Stores processed real-time and batch data in NoSQL for fast querying
   - Client interface highlighting fraud transactions, accepting/blocking transactions;
   - Data visualization with customizable filters

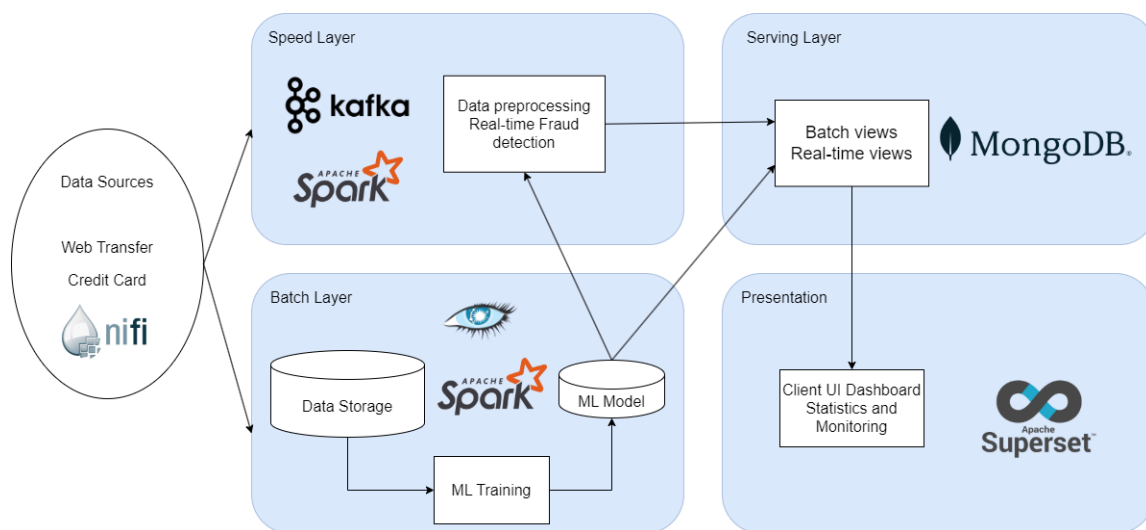Figure 1 shows an outline of the project architecture.

Figure 3: Project architecture

The following Big Data platforms will be used:

- Apache NiFi: to collect and distribute the data from different sources;

- Apache Cassandra as a data storage;

- Apache Kafka: to work with the streaming data;

- Apache Spark: to make the batch processing and model training;

- MongoDB: to store prepared batch view and real-time views for fast quering;

- Apache Superset (to be agreed with the supervisor): for data analysis on the user-interface. If the service will not be approved, UI will be implemented from scratch using JS framework, e.g. React.

6 Planned ML Tasks

This section explores potential machine learning tasks that could be applied to the available datasets for fraud and anomaly detection. The primary focus will be on identifying fraudulent transactions, developing anomaly detection models, and leveraging data-driven insights for effective fraud management.

6.1 Fraud Detection

Fraud detection is a key use case for the provided datasets, where the goal is to classify transactions as fraudulent or legitimate. This can be achieved using various supervised learning techniques. Given that fraudulent transactions are typically rare, special care must be taken to handle the class imbalance. Planned steps include:

- **Model Selection:** Applying binary classification models such as Logistic Regression, Random Forest, Gradient Boosted Trees (e.g., XGBoost, LightGBM), and Neural Networks. These models will be trained to predict the 'isFraud' label in the datasets.

- **Evaluation Metrics:** Since fraud detection deals with imbalanced data, standard metrics such as accuracy are insufficient. Precision, recall, F1-score, and the area under the Precision-Recall curve (AUC-PR) will be used to evaluate model performance.

- **Imbalanced Data Handling:** Techniques such as SMOTE (Synthetic Minority Oversampling Technique), undersampling, or cost-sensitive learning will be explored to address class imbalance.

6.2 Anomaly Detection

Anomaly detection is a complementary task to supervised fraud detection, focusing on identifying unusual transactions that deviate significantly from normal behavior. This approach is particularly useful for uncovering previously unseen fraud patterns. Proposed methods include:

- **Unsupervised Learning:** Techniques like Isolation Forests, k-means clustering, and DBSCAN will be applied to identify anomalous transactions based on transaction features.

- **Autoencoders:** Neural network-based autoencoders will be used to reconstruct transaction data, with high reconstruction errors indicating potential anomalies.

- **Density Estimation:** Probabilistic models such as Gaussian Mixture Models (GMM) will be employed to estimate the likelihood of transactions and flag those with low likelihoods as anomalies.

- **Validation:** Anomalies detected by these methods will be cross-validated against the 'isFraud' label to evaluate their effectiveness.

7 Planned batch and stream processing

This chapter describes the planned implementation of batch and stream processing flows in detail, as part of the Lambda Architecture outlined in the previous section.

7.1 Batch Processing Workflow

Batch processing will focus on historical data analysis, feature engineering, and machine learning model training. The workflow for batch processing is as follows:

1. Data Ingestion:

   - Periodic ingestion of historical transactions from data storage sources (Apache Cassandra).
   - Data preprocessing to validate and filter incomplete or invalid entries.
   - Aggregation of transactional data for customer-level or terminal-level insights.

2. Data Transformation:

   - Transformation of raw transaction data into structured formats suitable for ML model training.
   - Feature engineering, such as creating rolling statistics (e.g., transaction frequency, average amount per customer, or terminal-specific patterns).

3. Model Training:

   - Training ML models using the preprocessed data with a fixed schedule (e.g., every 30 minutes or daily, depending on model requirements).
   - Storing the trained models in a model repository for use in the serving layer and the streaming layer.

7.2 Stream Processing Workflow

The stream processing layer will handle real-time fraud detection and serve as the system's speed layer. The workflow for stream processing is as follows:

1. Data Ingestion:

- Continuous ingestion of live transaction data via Apache Kafka.

- Real-time preprocessing, including feature transformation (e.g., converting categorical variables, handling missing values).

2. Real-Time Fraud Detection:

- Use of trained ML models to perform predictions on incoming transactions.

- Updating fraud detection logic dynamically as new models are trained in the batch layer.

3. Event Streaming:

- Streaming results to MongoDB or another serving layer for downstream consumption.

- Flagging suspicious transactions for further analysis.

7.3 Serving Layer Workflow

The serving layer will provide a unified view of real-time and batch-processed data and offer tools for decision-making. The workflow is as follows:

1. Unified Data Storage:

- MongoDB will serve as the primary storage for both batch and real-time data, enabling fast querying and retrieval.

2. Client Interfaces:

- Interfaces for visualizing flagged transactions in real-time.

- Fraud management tools to allow users to accept or block flagged transactions.

3. Data Visualization:

- Customizable dashboards for exploring transaction patterns, fraud rates, and model performance.

- Apache Superset will be evaluated as the primary visualization tool; if not approved, a custom solution will be implemented using React.

7.4 Integration of Batch and Stream Processing

The Lambda Architecture ensures seamless integration between the batch and stream processing layers:

- ML models trained in the batch layer will be deployed to the stream processing layer for real-time fraud detection.

- Features engineered in the batch layer can inform the design of streaming features to ensure consistency.

- The serving layer will present a unified view of historical and real-time data, enabling holistic decision-making.

8   Planned way of presenting the results

In this architecture, we propose to use Apache Superset as the primary tool for presenting analytical results and monitoring system performance. Superset is an open-source data visualization and business intelligence platform that is well-suited for modern data pipelines due to its extensive support for a wide range of data sources and its ease of integration into existing architectures.

8.1   Integration of Apache Superset with the Current Architecture

As illustrated in the architecture diagram:

- The **Batch Layer**, powered by Apache Cassandra, processes large-scale data and aggregates it into queryable formats.

- The **Speed Layer**, which uses Kafka for real-time data ingestion and Spark for processing, produces real-time views that are stored in MongoDB. Superset can connect to MongoDB to display these real-time analytics, provided an appropriate SQL-like connector (e.g., MongoDB BI Connector) is configured.

- The **Serving Layer** facilitates the querying of both batch and real-time views, making this data accessible to Superset for visualization in unified dashboards.

By integrating Superset with both the batch and speed layers, the system can provide users with comprehensive dashboards that include both historical trends and real-time updates. This combination is particularly beneficial for monitoring critical use cases such as fraud detection, as it allows stakeholders to view both immediate alerts and long-term patterns.

8.2   Advantages of Using Apache Superset

- **Ease of Use**: Superset provides an intuitive drag-and-drop interface for creating visualizations and dashboards, making it accessible to non-technical users.

- **Real-Time and Batch Data Integration**: The ability to combine data from the speed and batch layers into unified dashboards enables comprehensive analytics and monitoring.

- **Custom Visualization**: Superset supports a variety of visualizations, allowing users to explore data in formats that best suit their needs, from basic bar charts to advanced geographic maps.

- **Scalability**: Being lightweight and web-based, Superset can handle a growing volume of data as the architecture scales.

- **Open Source and Extensibility**: As an open-source platform, Superset can be customized and extended to fit specific requirements.

8.3 Disadvantages and Challenges

While Superset provides significant benefits, there are a few challenges and limitations to consider:

- **Indirect Support for Kafka**: Superset does not natively connect to Kafka. Real-time data from Kafka must first be stored in a queryable data store (e.g., MongoDB or a similar OLAP engine) before it can be visualized.

- **MongoDB Support**: Superset does not have native support for MongoDB. A SQL-like connector (such as the MongoDB BI Connector) or transformation of data into another supported format is required, adding some complexity.

- **Performance Considerations**: For real-time monitoring, performance can be a bottleneck if the underlying data stores (e.g., MongoDB or Cassandra) are not optimized for frequent querying by Superset.

- **Learning Curve for Advanced Features**: While the basic interface is user-friendly, advanced configurations (e.g., complex filters, custom SQL queries) may require technical expertise.

9 Tasks assignment

The table below contains the list of team members and preliminary allocation of tasks to team members.

Team member	Tasks	Supporter
Salveen Singh Dutt	Batch processing of the historical data for up-to-date model training (Batch Layer).	Karina Tiurina
Karina Tiurina	Fraud detection model training and fine-tuning; Data stream processing (Speed Layer).	Salveen Singh Dutt
Patryk Prusak	Data ingestion, collection and pre-processing.	Karina Tiurina, Salveen Singh Dutt
Patryk Prusak	Data visualisation and configuration on the UI (Serving Layer).	Karina Tiurina, Salveen Singh Dutt

Table 6: Tasks assignment

References

[1] Chitwan Manchanda, *Fraudulent Transactions Data*, Kaggle, 2022. Available at: `https://www.kaggle.com/datasets/chitwanmanchanda/fraudulent-transactions-data`.

[2] *Credit_Card_Fraud_*, OpenML, 2024. Available at: `https://www.openml.org/search?type=data&status=active&id=45955&sort=runs`.

[3] Carlos José, *Credit Card Transactions Synthetic Data Generation*, Kaggle, 2024. Available at: `https://www.kaggle.com/datasets/cgrodrigues/credit-card-transactions-synthetic-data-generation?select=transactions_df.csv`.

[4] Machine Learning Group - ULB and Andrea, *Credit Card Fraud Detection*, Kaggle, 2018. Available at: `https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud`.

[5] *Lambda Architecture*, Cazton. Available at: `https://cazton.com/consulting/enterprise/lambda-architecture`.

[6] OpenAI, *ChatGPT: Language Model*, Available at: `https://chatgpt.com`.