Building an AI System for Identifying Fraudulent Sales Calls

Sakshi Choudhary

Asansol Engineering College, B.tech'25

Gajendra Diwedi

Delhi School of Economics, MA Economics

Period of Internship: 14th Jan 2025 - 30th April 2025

Report submitted to: IDEAS – Institute of Data Engineering, Analytics and Science Foundation, ISI Kolkata

Abstract

With the increasing volume of customer interactions and the potential for fraud in sales processes, traditional monitoring methods fall short in scalability and intelligence. This project introduces an AI-powered fraud detection pipeline that analyzes raw sales call transcripts to extract business-relevant features and detect fraudulent behavior. Leveraging Large Language Models (LLMs), specifically LLaMA 3.2 interfaced through Ollama, the system transforms unstructured textual data into a structured featured dataset. Python automates the entire feature extraction and model inference process, while Power BI visualizes insights such as fraud patterns, suspicious agent behavior, and red-flag conversations. The solution is lightweight, open-source, and customizable, offering high adaptability across domains.

Introduction

In today's highly dynamic business environment, sales calls remain a critical touchpoint between organizations and customers. These interactions, while instrumental in driving revenue, also open avenues for unethical sales practices, misinformation, and even outright fraud. Companies often struggle with detecting such behavior in time, especially when it is buried within thousands of unstructured transcripts generated daily by call centers. Traditional approaches—such as manual auditing or keyword-based flagging—are not only time-consuming but also error-prone and lack contextual understanding.

The relevance of this project stems from the growing need for **automated, intelligent fraud detection mechanisms** in customer-facing industries like insurance, banking, e-commerce, and telecom AI, particularly Natural Language Processing (NLP), presents a transformative solution to this challenge by analyzing vast amounts of unstructured textual data in a scalable and consistent manner.

This project uses **Large Language Models (LLMs)** like **LLaMA 3.2** via **Ollama** to evaluate call transcripts and identify suspicious behavior. The model's ability to understand natural human conversation and its nuances makes it far superior to traditional fraud detection rules.

Background Material Survey

Several studies and tools in recent years have explored the potential of AI in compliance monitoring. Existing approaches fall into the following categories:

- **Manual Auditing Systems**: Highly accurate but infeasible for large datasets due to cost and time constraints.
- **Rule-based Keyword Detectors**: Fast but overly simplistic; they lack understanding of context, tone, and intent.
- **Supervised ML Classifiers**: Require extensive labeled datasets and often struggle with generalization to unseen patterns.
- **LLM-Powered Systems**: Models like GPT, LLaMA, and Claude demonstrate a contextual understanding of language, capable of extracting nuanced insights from long-form dialogue.

The primary motivation for this project is to address the limitations of current fraud detection techniques in handling unstructured data. Specific objectives include:

- **Reducing Human Effort**: Automating the analysis of transcripts to save hours of manual review.
- **Enhancing Accuracy**: Leveraging contextual understanding of LLMs to detect subtle fraudulent tactics.
- **Generating Actionable Insights**: Providing easy-to-interpret dashboards for managers to take timely decisions.

Objective

The main objective of this project is to develop a comprehensive and automated system for detecting fraudulent patterns in sales conversations. Key goals include:

1. **Automated Feature Extraction**: Design a system that can parse and process raw sales call transcripts to identify key indicators relevant to fraud.

2. **Contextual Understanding**: Leverage LLaMA 3.2, a state-of-the-art open-source large language model, to interpret the nuanced context of sales conversations.

3. **Structuring Unstructured Data**: Convert raw conversational text into a structured format with measurable features like sentiment, red flag indicators, and potential fraud scores.

4. **Visualization and Reporting**: Create intuitive Power BI dashboards to help sales managers and compliance teams monitor and assess potential fraudulent activity.

5. **Scalability**: Ensure the system can scale across large datasets without performance degradation.

Technologies Used

Python (.py)

- Used for scripting automation of the entire pipeline.
- Responsible for reading raw transcript files, cleaning text, generating prompts, sending prompts to the LLM, and saving structured outputs.

Excel

- Acts as both the input and output format.
- Raw call transcripts are stored in tabular form.
- Output files contain the featured dataset with fraud detection indicators.

Ollama

- Lightweight runtime interface for running LLaMA locally.
- Enables quick model loading and inference without relying on cloud APIs or GPU clusters.
- Helps integrate the LLM into local or on-premise enterprise environments.

LLaMA 3.2:latest

- A powerful open-source transformer model known for its ability to understand language context.
- Extracts red flags, misleading sales tactics, and customer sentiment from call transcripts.
- Responds to prompts by summarizing intent, highlighting unethical behavior, and classifying content.

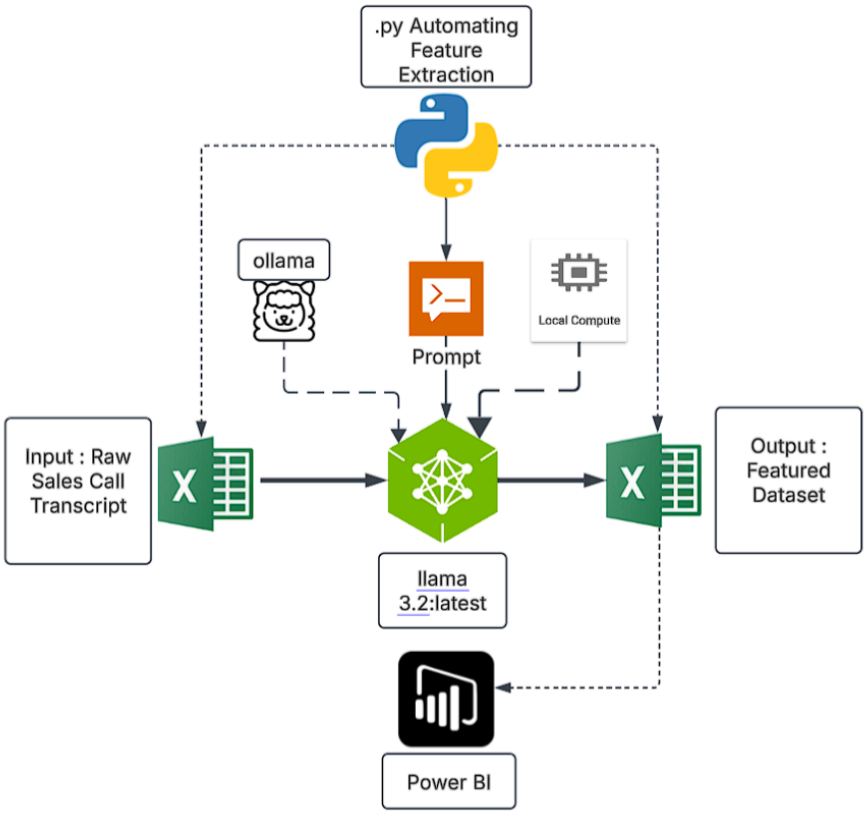
Prompt Engineering

- Prompts guide the LLM to extract relevant features (e.g., “Identify if any statements in this transcript contain exaggerated promises.”)
- Fine-tuned to elicit accurate and structured outputs from the model.

Power BI

- Transforms the structured output into visual insights.
- Allows stakeholders to explore metrics such as:
    - Fraud detection trends over time.
    - Agent-specific risk dashboards.
    - Frequently used flagged phrases.
    - Correlation between call length and fraud risk.

Methodology



1. **Input: Raw Call Transcripts**

- Source data: An Excel file where each row represents a sales call.

- Columns include agent name, call ID, timestamp, and full call transcript.

- Calls may vary in length, tone, and structure, posing a challenge for consistency.

2. Automated Feature Extraction (Python Script)

A Python automation script processes the transcript dataset and extracts key business intelligence and compliance-related features using two sets of carefully crafted prompts:

Tech: Python + pandas/numpy

Purpose: Preprocess transcripts into:

- Sentiment scores (Agent & Customer)
- Tone classification
- Engagement metrics (speech ratio, turns, word count)
- Compliance indicators (security keywords, sensitive info request)
- Keyword tagging (refund, pricing, testimonials, escalation)

Your Prompt1:

Please analyze the provided sales call transcript and extract key business insights based on predefined categories. Identify the Agent Name, Customer Name, and Company Name mentioned in the conversation. Determine whether the customer has made specific inquiries by marking Product Inquiry, Service Inquiry, Return Inquiry, Refund Inquiry, and Technical Support Inquiry as 1 if mentioned and 0 if not. Identify customer issues such as Quality Issue, Trust Issue, Pricing Issue, Sizing Issue, Delivery Issue, or Other Issues, marking them as 1 if they appear in the conversation and 0 if they do not. Extract the solutions provided by the agent, including Product Information, Discount Offer, Customer Testimonials, or Escalation to a Manager, marking them as 1 if mentioned and 0 otherwise. Assess the Tone of the Agent and Tone of the Customer, categorizing them as Professional with 1, Sales-Oriented with 2, Apologetic with 3, or Neutral with 4 for the agent, and Neutral with 1, Skeptical with 2, Frustrated with 3, or Interested with 4 for the customer.Analyze Customer Sentiment and Agent Sentiment, classifying them as Positive with 1, Neutral with 2, or Negative with 3. Additionally, check for compliance and risk factors by determining whether the agent Requested Sensitive Information, whether the conversation Adhered to Regulatory Standards, and whether there is any indication of Fraudulent Activity, marking each as 1 for Yes and 0 for No. Ensure all extracted features align with the predefined categories and numerical representations for consistency and accuracy.

Your Prompt2:

Please analyze the provided sales call transcript and extract the following quantitative features based on the conversation. Calculate the total number of turns, which represents the total back-and-forth exchanges between the agent and the customer. Determine the total word count in the conversation, along with the individual word count for the agent and the word count for the customer. Identify the total number of questions asked, as well as the number of questions asked by the agent and the number of questions asked by the customer. Compute the speech ratio, which is the ratio of the number of words spoken by the agent to the number of words spoken by the customer. If a discount or any offer is given by agent and mark as 1 if given else 0 if not given , . Identify and count the number of unique product features discussed, such as "lightweight," "warm," or "sizing." Analyze the conversation for positive and negative keywords, counting the occurrences of words that indicate positive sentiment (e.g., "great," "helpful," "amazing") and words that indicate negative sentiment (e.g., "problem," "concerned," "issue"). Finally, determine the resolution status, assigning 1 if the issue was resolved (customer agreed or expressed satisfaction) and 0 if the issue remained unresolved (customer remained skeptical or did not confirm resolution or purchase). Ensure all extracted values are in numerical form and align with the defined categories for accuracy and consistency.

Extracted Features and Data Types

Feature	Data Type
Call ID	Numeric
Transcript	Text
Agent Name	Nominal
Customer Name	Nominal
Company Name	Nominal
Product Inquiry	Boolean
Service Inquiry	Boolean
Return Inquiry	Boolean
Refund Inquiry	Boolean
Technical Inquiry	Boolean
Quality Issue	Boolean
Trust Issue	Boolean
Pricing Issue	Boolean
Sizing Issue	Boolean
Delivery Issue	Boolean
Other Issue	Boolean
Solution Product Information	Boolean
Solution Discount Offer	Boolean

Feature	Data Type
Solution Customer Testimonials	Boolean
Solution Manager Escalation	Boolean
Tone of the Agent	Categorical
Tone of the Customer	Categorical
Customer Sentiment	Categorical
Agent Sentiment	Categorical
Requested Sensitive Information	Boolean
Adhered to Regulatory Standards	Boolean
Total Turns	Numeric
Total Word Count	Numeric
Agent Word Count	Numeric
Customer Word Count	Numeric
Total Questions	Numeric
Agent Questions	Numeric
Customer Questions	Numeric
Speech Ratio	Numeric
Discount Offer	Boolean
Unique Product Features	Numeric
Positive Word Count	Numeric
Negative Word Count	Numeric
Resolution Status	Categorical
Fraudulent Activity	Boolean

.PY Code for Automation and Feature Extraction

1 Code Breakdown

1.1 1. Import Libraries

We begin by importing necessary libraries:

Import Libraries

```
import pandas as pd
import ollama
import datetime
import json
```

- `pandas`: For handling dataframes and working with Excel files. - `ollama`: To interact with the Ollama API for processi transcripts. - `datetime`: To handle timestamps. - `json`: For parsing the JSON response from the Ollama API.

1.2 2. Parsing JSON Output

The function `parse_json` extracts relevant features from the JSON output returned by the API.

Parse JSON Function

```
def parse_json(json_output):
    # Load the JSON response as a Python object (list of dictionaries)
    parsed_data = json.loads(json_output)

    # Assuming the response is a list with a single dictionary, extract the features
    if isinstance(parsed_data, list) and len(parsed_data) > 0:
        parsed_data = parsed_data[0]

    # Return the features as a dictionary
    return {
        "Agent Name": parsed_data.get("Agent Name", None),
        "Customer Name": parsed_data.get("Customer Name", None),
        "Company Name": parsed_data.get("Company Name", None),
        "Product Inquiry": parsed_data.get("Product Inquiry", 0),
        "Service Inquiry": parsed_data.get("Service Inquiry", 0),
        "Return Inquiry": parsed_data.get("Return Inquiry", 0),
        "Refund Inquiry": parsed_data.get("Refund Inquiry", 0),
        "Technical Support Inquiry": parsed_data.get("Technical Support Inquiry", 0),
        "Quality Issue": parsed_data.get("Quality Issue", 0),
        "Trust Issue": parsed_data.get("Trust Issue", 0),
        "Pricing Issue": parsed_data.get("Pricing Issue", 0),
        "Sizing Issue": parsed_data.get("Sizing Issue", 0),
        "Delivery Issue": parsed_data.get("Delivery Issue", 0),
        "Other Issues": parsed_data.get("Other Issues", 0),
        "Product Information": parsed_data.get("Product Information", 0),
        "Discount Offer": parsed_data.get("Discount Offer", 0),
        "Customer Testimonials": parsed_data.get("Customer Testimonials", 0),
        "Escalation to a Manager": parsed_data.get("Escalation to a Manager", 0),
        "Tone of the Agent": parsed_data.get("Tone of the Agent", None),
        "Tone of the Customer": parsed_data.get("Tone of the Customer", None),
        "Customer Sentiment": parsed_data.get("Customer Sentiment", None),
        "Agent Sentiment": parsed_data.get("Agent Sentiment", None),
        "Requested Sensitive Information": parsed_data.get("Requested Sensitive
            Information", 0),
        "Adhered to Regulatory Standards": parsed_data.get("Adhered to Regulatory
            Standards", 0),
        "Total Turns": parsed_data.get("Total Turns", 0),
        "Total Word Count": parsed_data.get("Total Word Count", 0),
        "Agent Word Count": parsed_data.get("Agent Word Count", 0),
        "Customer Word Count": parsed_data.get("Customer Word Count", 0),
        "Total Questions Asked": parsed_data.get("Total Questions Asked", 0),
        "Agent Questions Asked": parsed_data.get("Agent Questions Asked", 0),
        "Customer Questions Asked": parsed_data.get("Customer Questions Asked", 0),
        "Positive Word Count": parsed_data.get("Positive Word Count", 0),
        "Negative Word Count": parsed_data.get("Negative Word Count", 0),
        "Resolution_Status": parsed_data.get("Resolution_Status", 0),
        "speech_ratio": parsed_data.get("speech_ratio", 0.0),
        "Engagement_Score": parsed_data.get("Engagement_Score", 0.0),
        "Discount_Impact_Score": parsed_data.get("Discount_Impact_Score", 0),
        "Fraudulent_Activity": parsed_data.get("Fraudulent_Activity", 0)
    }
```

The function loads the JSON response and extracts key features, such as agent name, customer sentiment, and product/service inquiries. If the JSON contains multiple records, it processes the first entry.

1.3 3. Loading Input Data

We load the customer service transcripts from an Excel file:

Loading Input Data

```
input_df = pd.read_excel('input_transcript.xlsx')
```

1.4 4. Setting Up the Output Dataframe

We define the structure of the output dataframe, specifying the columns we expect after processing each transcript:

Setting Up Output Dataframe

```
output_df = pd.DataFrame(columns=[
    "Agent Name", "Customer Name", "Company Name", "Product Inquiry", "Service Inquiry",
    "Return Inquiry", "Refund Inquiry", "Technical Support Inquiry", "Quality Issue",
    "Trust Issue", "Pricing Issue", "Sizing Issue", "Delivery Issue", "Other Issues",
    "Product Information", "Discount Offer", "Customer Testimonials", "Escalation to a
       Manager",
    "Tone of the Agent", "Tone of the Customer", "Customer Sentiment", "Agent Sentiment",
    "Requested Sensitive Information", "Adhered to Regulatory Standards", "Total Turns",
    "Total Word Count", "Agent Word Count", "Customer Word Count", "Total Questions Asked"
       ,
    "Agent Questions Asked", "Customer Questions Asked", "Positive Word Count",
    "Negative Word Count", "Resolution_Status", "speech_ratio", "Engagement_Score",
    "Discount_Impact_Score", "Fraudulent_Activity"
])
```

The dataframe is initialized with the necessary column names, which will store the extracted features.

1.5 5. Processing the First Transcript

The first transcript is selected for processing:

Processing First Transcript

```
row = input_df.iloc[0]  # Select the first transcript
sample_transcript = row['Transcript']
```

The transcript text is extracted from the `Transcript` column of the input dataframe.

1.6 6. Generating the Prompt for the API

We define the prompt that instructs the Ollama model on how to process the transcript:

Defining the API Prompt

```
prompt = """You are a highly detailed and structured data extraction tool specializing in
    analyzing customer service call transcripts. Your task is to meticulously parse a
    provided transcript and generate a JSON formatted output consistent with the following
     template. Each JSON object should represent a single turn in the conversation,
    capturing relevant information. Follow a strict, standardized format - this template
    is crucial, for consistent data analysis. Here is the template for each JSON object:
    Agent Name: String - The name of the agent who spoke. Customer Name: String - The name
     of the customer. If null, set to null. Company Name: String - The name of the company
    . If null, set to null. Product Inquiry: Integer - 1 if the product-related inquiry is
     made else 0. Service Inquiry: Integer - 1 if service-related inquiry is made else 0.
    Return Inquiry: Integer - The number representing the return-related inquiry is made
    else 0. Refund Inquiry: Integer - The number representing the refund-related inquiry
    is made else 0. Technical Support Inquiry: Integer - The number representing the
    technical support inquiry. Quality Issue: Integer - The number representing the
    quality issue is made else 0. Trust Issue: Integer - The number representing the trust
     issue is made else 0. Pricing Issue: Integer - The number representing the pricing
    issue is made else 0. Sizing Issue: Integer - The number representing the sizing issue
     is made else 0. Delivery Issue: Integer - The number representing the delivery issue
    is made else 0. Other Issues: Integer - The number representing the other issue is
    made else 0. Product Information: Integer - 1 if product or service info is given else
     0. Discount Offer: Integer - 1 if discount offer is given else 0. Customer
    Testimonials: Integer - 1 if customer testimonials is given by agent else 0.
    Escalation to a Manager: Integer - 1 if agent escalated the issue to a manager else 0.
     Tone of the Agent: String - Choose Among (Professional, Sales-Oriented, Apologetic,
    or Neutral). Tone of the Customer: String - Choose Among (Neutral, Skeptical,
    Frustrated, or Interested). Customer Sentiment: String - Choose Among (Positive,
    Neutral, or Negative). Agent Sentiment: String - Choose Among (Positive, Neutral, or
    Negative). Requested Sensitive Information: Integer - 1 if request for sensitive info
    made by agent else 0. Adhered to Regulatory Standards: Integer - 1 if agent adhere to
    regulatory standard else 0. Total Turns: Integer - The number of turns in the
    conversation. Total Word Count: Integer - The number of words in the conversation.
    Agent Word Count: Integer - The number of words in the agent's statement. Customer
    Word Count: Integer - The number of words in the customer's statement. Total Questions
     Asked: Integer - The number of questions asked. Agent Questions Asked: Integer - The
    number of questions asked. Customer Questions Asked: Integer - The number of questions
     asked. Positive Word Count: Integer - The number of positive words in the
    conversation. Negative Word Count: Integer - The number of negative words in the
    conversation. Resolution_Status: Integer - 1 if Customer issue resolved else 0.
    speech_ratio: Float - Agent Questions Asked / Customer Questions Asked.
    Engagement_Score: Float - 0.4 Total Turn + 0.6 Total Questions Asked.
    Discount_Impact_Score: Integer - Discount Offered * Resolution Status and last one is
    Fraudulent_Activity: Integer - 1 if call sounds fraudulent else 0. Now, please
    generate the JSON output exactly as described. Do not change anything - strictly
    adhere to the template."""
```

The prompt is designed to instruct the model to extract all required information in a structured format.

1.7 7. Calling the API and Generating JSON Output

The complete prompt and the transcript are sent to the Ollama API to generate the structured output:

API Request and Response

```
full_prompt = prompt + sample_transcript + "\n\nNow␣generate␣only␣the␣JSON␣and␣nothing␣
    else␣:"

response = ollama.chat(
    model="llama3.2:latest",
    messages=[{"role": "user", "content": full_prompt}]
)

json_output = response["message"]["content"]
```

The API processes the transcript and returns a structured JSON response.

1.8 8. Parsing the JSON Output

The JSON output is passed to the `parse_json` function to extract the relevant features:

Parsing JSON Output

```
features = parse_json(json_output)
```

1.9 9. Storing the Extracted Features

The extracted features are added to the output dataframe:

Storing Features

```
output_df = pd.concat([output_df, pd.DataFrame([features])], ignore_index=True)
```

The new features are appended to the dataframe.

1.10 10. Saving the Results to an Excel File

Finally, the processed data is saved to an Excel file:

Saving the Results

```
output_df.to_excel('output_transcript_first.xlsx', index=False)
```

The results are saved in `output_transcript_first.xlsx`, which can be opened for further analysis.

1.11 11. Completion Message

A message is printed when the processing is complete:

Completion Message

```
print(datetime.datetime.now().strftime("%Y-%m-%d␣%H:%M:%S") + "␣-␣Processing␣complete␣for␣
    first␣transcript.␣Output␣saved␣to␣'output_transcript_first.xlsx'.")
```

1.12 10. Sample json output from llama model using optimized prompt

The extracted features from sales call transcript using llama model:

3. Model Execution via Ollama + LLaMA 3.2

LLaMA 3.2 is used as the Natural Language Processing (NLP) engine to analyze raw sales call transcripts and extract structured business-related and fraud-detection features.

The model understands unstructured, conversational language between customers and agents and identifies hidden insights like tone, sentiment, compliance violations, fraud risk factors, and engagement metrics — tasks that are too complex for simple keyword-based methods.

The integration of LLaMA is accomplished through **prompt engineering** and **local API deployment via Ollama**, allowing for secure and efficient local processing without reliance on external cloud services.

The **prompt** is sent to the **LLaMA 3.2 model** using the **Ollama** API locally.

LLaMA Model Processing

- **LLaMA reads** the provided conversation and **returns a structured JSON output**.

- The output contains all required business features and fraud-related flags automatically.

JSON Parsing and Feature Extraction

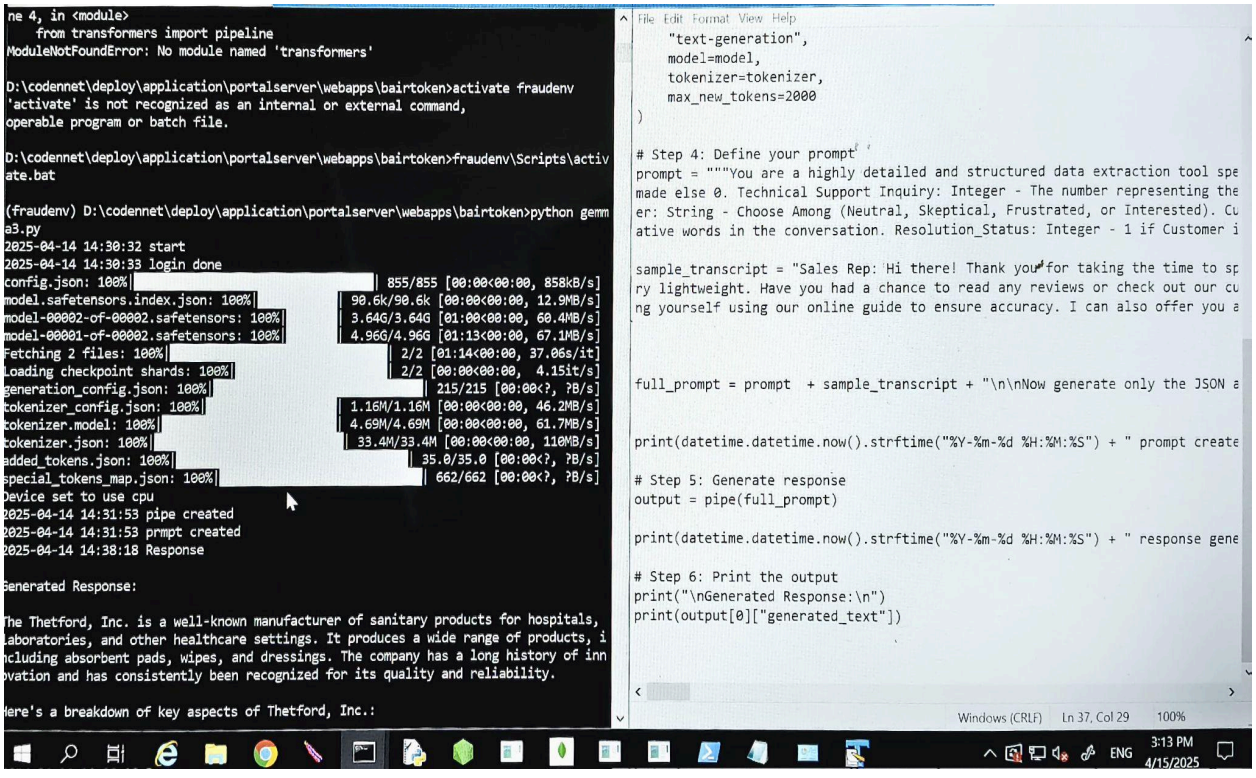
- The system **parses the JSON response** using another Python function.

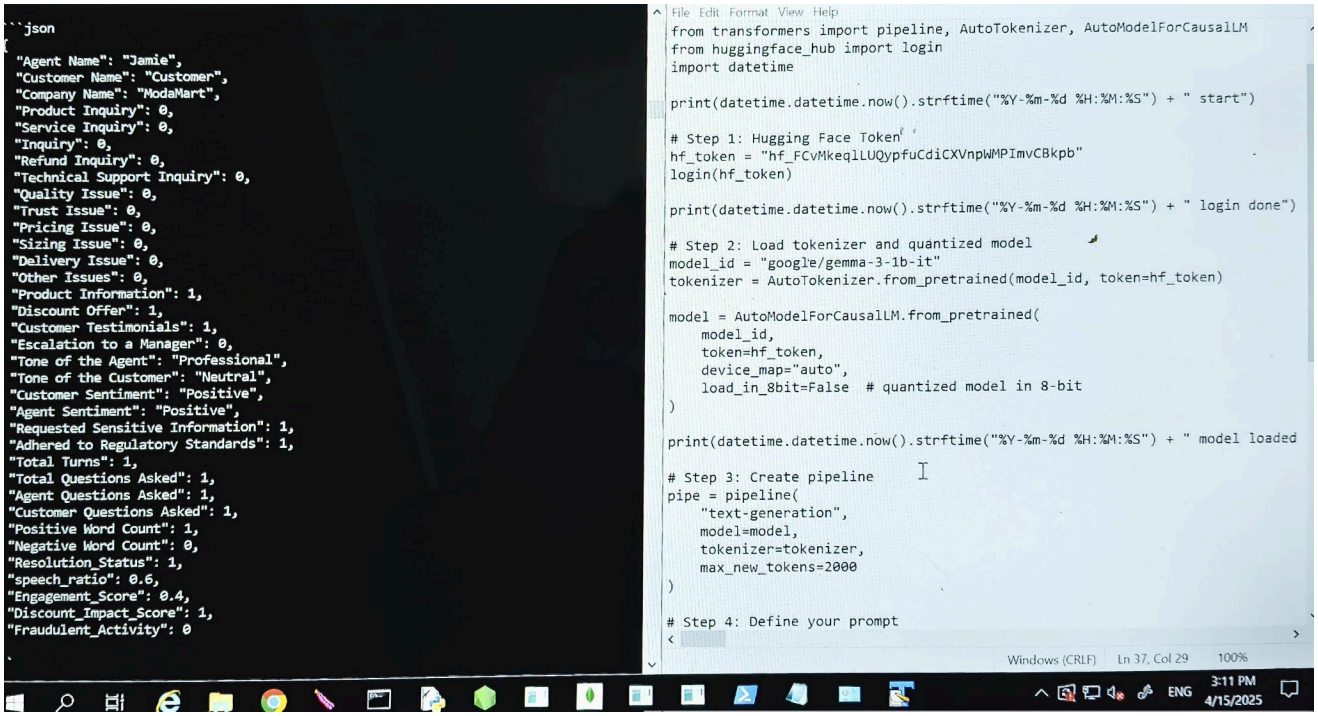
- Extracted features are appended into a **structured DataFrame** (rows and columns) and saved back into **Excel**.

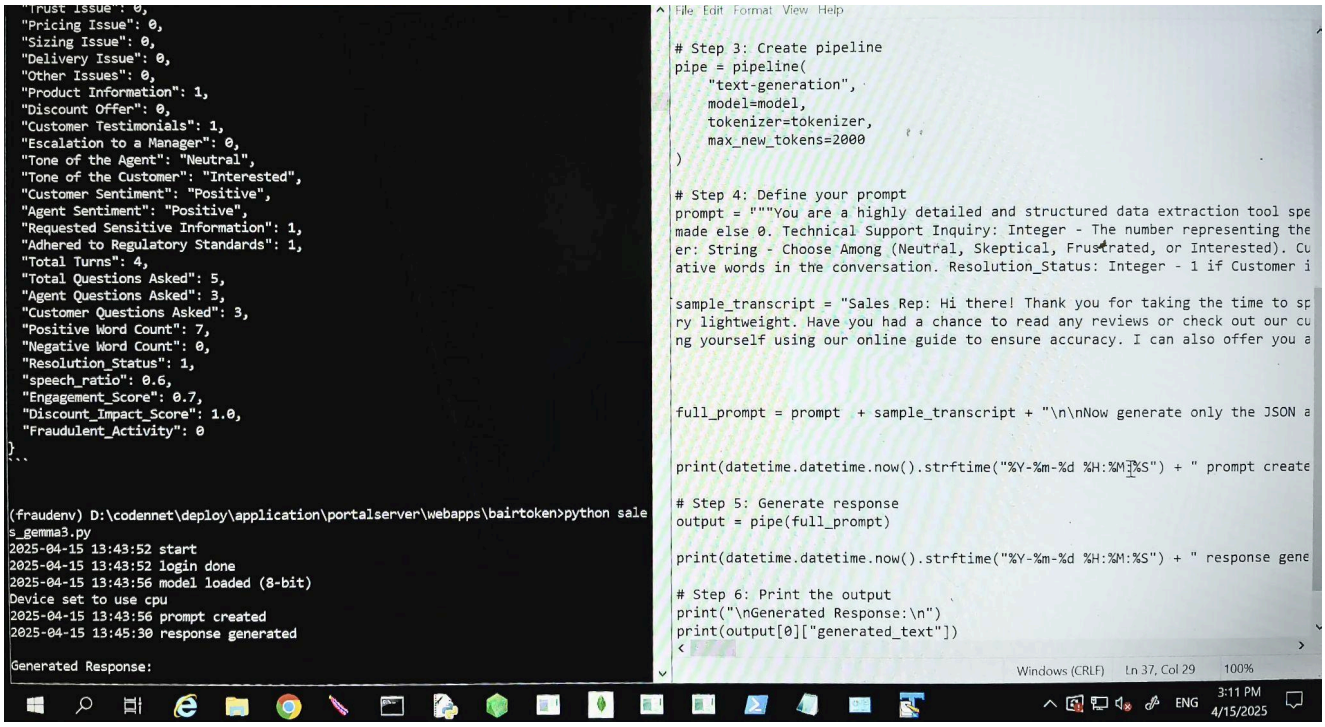
4. Output: Featured Dataset

The model's output is parsed and structured into a **feature-rich Excel dataset**, with over **35+ features** classified into:

- **Boolean** (e.g., Fraudulent Activity, Trust Issue)
- **Categorical** (e.g., Sentiment labels)
- **Numeric** (e.g., Word Counts, Speech Ratio)




```json
{
"Agent Name": "Jamie",
"Customer Name": "Customer",
"Company Name": "ModaMart",
"Product Inquiry": 0,
"Service Inquiry": 0,
"Inquiry": 0,
"Refund Inquiry": 0,
"Technical Support Inquiry": 0,
"Quality Issue": 0,
"Trust Issue": 0,
"Pricing Issue": 0,
"Sizing Issue": 0,
"Delivery Issue": 0,
"Other Issues": 0,
"Product Information": 1,
"Discount Offer": 1,
"Customer Testimonials": 1,
"Escalation to a Manager": 0,
"Tone of the Agent": "Professional",
"Tone of the Customer": "Neutral",
"Customer Sentiment": "Positive",
"Agent Sentiment": "Positive",
"Requested Sensitive Information": 1,
"Adhered to Regulatory Standards": 1,
"Total Turns": 1,
"Total Questions Asked": 1,
"Agent Questions Asked": 1,
"Customer Questions Asked": 1,
"Positive Word Count": 1,
"Negative Word Count": 0,
"Resolution_Status": 1,
"speech_ratio": 0.6,
"Engagement_Score": 0.4,
"Discount_Impact_Score": 1,
"Fraudulent_Activity": 0
}
File Edit Format View Help
from transformers import pipeline, AutoTokenizer, AutoModelForCausalLM
from huggingface_hub import login
import datetime
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S") + " start")
# Step 1: Hugging Face Token
hf_token = "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"
login(hf_token)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S") + " login done")
# Step 2: Load tokenizer and quantized model
model_id = "google/gemma-3-1b-it"
tokenizer = AutoTokenizer.from_pretrained(model_id, token=hf_token)
model = AutoModelForCausalLM.from_pretrained(
model_id,
token=hf_token,
device_map="auto",
load_in_8bit=False # quantized model in 8-bit
)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S") + " model loaded
# Step 3: Create pipeline
pipe = pipeline(
"text-generation",
model=model,
tokenizer=tokenizer,
max_new_tokens=2000
)
# Step 4: Define your prompt
Windows (CRLF) Ln 37, Col 29 100%
ENG 3:11 PM 4/15/2025


"Trust Issue": 0,
"Pricing Issue": 0,
"Sizing Issue": 0,
"Delivery Issue": 0,
"Other Issues": 0,
"Product Information": 1,
"Discount Offer": 0,
"Customer Testimonials": 1,
"Escalation to a Manager": 0,
"Tone of the Agent": "Neutral",
"Tone of the Customer": "Interested",
"Customer Sentiment": "Positive",
"Agent Sentiment": "Positive",
"Requested Sensitive Information": 1,
"Adhered to Regulatory Standards": 1,
"Total Turns": 4,
"Total Questions Asked": 5,
"Agent Questions Asked": 3,
"Customer Questions Asked": 3,
"Positive Word Count": 7,
"Negative Word Count": 0,
"Resolution_Status": 1,
"speech_ratio": 0.6,
"Engagement_Score": 0.7,
"Discount_Impact_Score": 1.0,
"Fraudulent_Activity": 0
}
```
(fraudenv) D:\codennet\deploy\application\portalserver\webapps\bairtoken>python sales_gemma3.py
2025-04-15 13:43:52 start
2025-04-15 13:43:52 login done
2025-04-15 13:43:56 model loaded (8-bit)
Device set to use cpu
2025-04-15 13:43:56 prompt created
2025-04-15 13:45:30 response generated
Generated Response:
File Edit Format View Help
# Step 3: Create pipeline
pipe = pipeline(
"text-generation",
model=model,
tokenizer=tokenizer,
max_new_tokens=2000
)
# Step 4: Define your prompt
prompt = """You are a highly detailed and structured data extraction tool spe
made else 0. Technical Support Inquiry: Integer - The number representing the
er: String - Choose Among (Neutral, Skeptical, Frustrated, or Interested). Cu
ative words in the conversation. Resolution_Status: Integer - 1 if Customer i
sample_transcript = "Sales Rep: Hi there! Thank you for taking the time to sp
ry lightweight. Have you had a chance to read any reviews or check out our cu
ng yourself using our online guide to ensure accuracy. I can also offer you a
full_prompt = prompt + sample_transcript + "\n\nNow generate only the JSON a
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S") + " prompt create
# Step 5: Generate response
output = pipe(full_prompt)
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S") + " response gene
# Step 6: Print the output
print("\nGenerated Response:\n")
print(output[0]["generated_text"])
Windows (CRLF) Ln 37, Col 29 100%
ENG 3:11 PM 4/15/2025

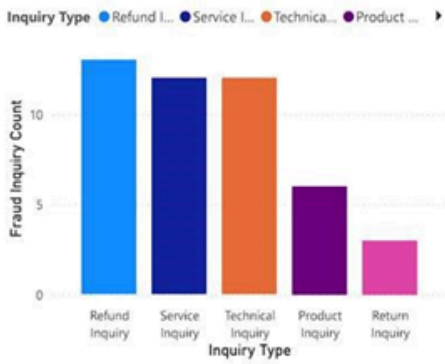
5.Results and Analysis

Feature	Insights
Sentiment Analysis	Fraud calls have higher negative sentiment for both agents and customers
Speech Ratio	Agents speak less; customers speak more in fraud calls
Tone	Agents maintain a professional tone; customers often appear skeptical
Engagement	Fraud calls have higher back-and-forth exchanges, turns, and total word count
Compliance	86% of fraud calls violate security protocols
Offers/Testimonial	Rare in fraud calls; fraud agents tend to ask more and offer less

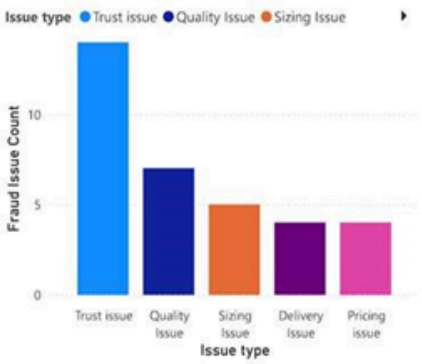
Call Overview

- Refund and Service and Tech Inquiries are primary fraud targets.
- Trust Issues account for 41.18% of fraud cases.
- Fraud Agents often escalate the issue to manager and do not give testimonials and discount
- Legitimate calls are more solution driven through product , testimonial and offers rather just escalation to manager
- Pricing Issue is also become important in legitimate calls , Trust issue being the major one

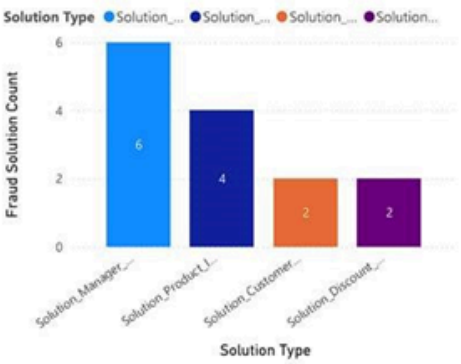
Fraud Inquiry Count by Inquiry Type and Inquiry Type



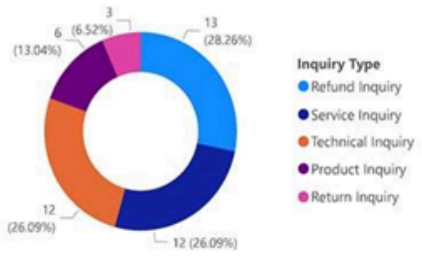
Fraud Issue Count by Issue type and Issue type



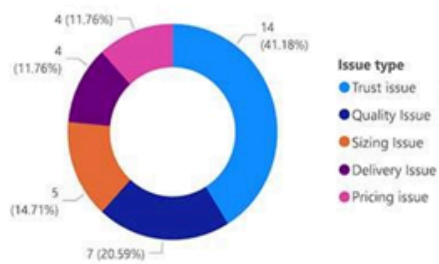
Fraud Solution Count by Solution Type and Solution Type



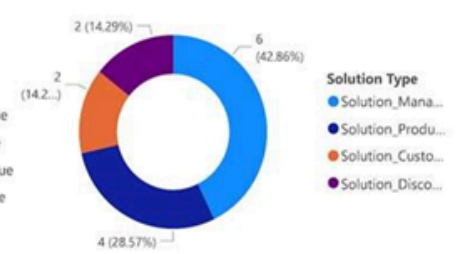
Fraud Inquiry Count by Inquiry Type



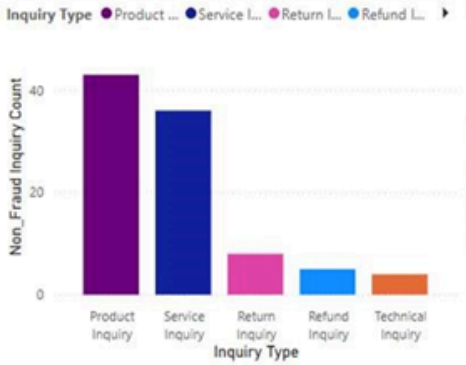
Fraud Issue Count by Issue type



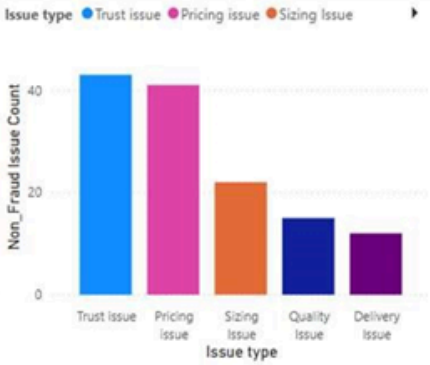
Fraud Solution Count by Solution Type



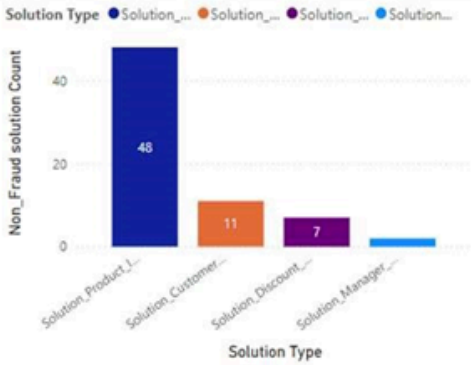
Non_Fraud Inquiry Count by Inquiry Type and Inquiry Type



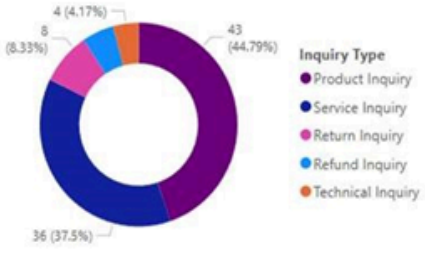
Non_Fraud Issue Count by Issue type and Issue type



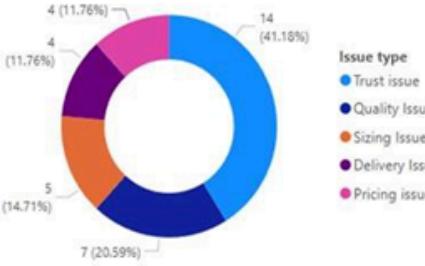
Non_Fraud solution Count by Solution Type and Solution Type



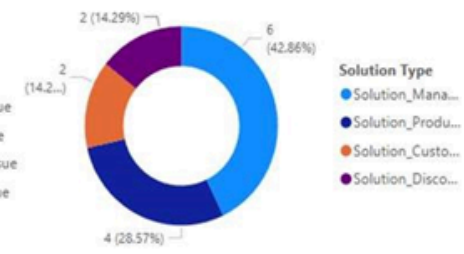
Non_Fraud Inquiry Count by Inquiry Type



Fraud Issue Count by Issue type



Fraud Solution Count by Solution Type



★Agents are always Professional while Customer are often Neutral and Skeptic

- Agents always have a professional and Sales Oriented Tone while Customers being Neutral and Skeptical during conversation a fraud call.
- Even in Legitimate Call the Tone has similar Trend
- Conclusion of fraud call drawn from tone of agent and customer may lead to biased result

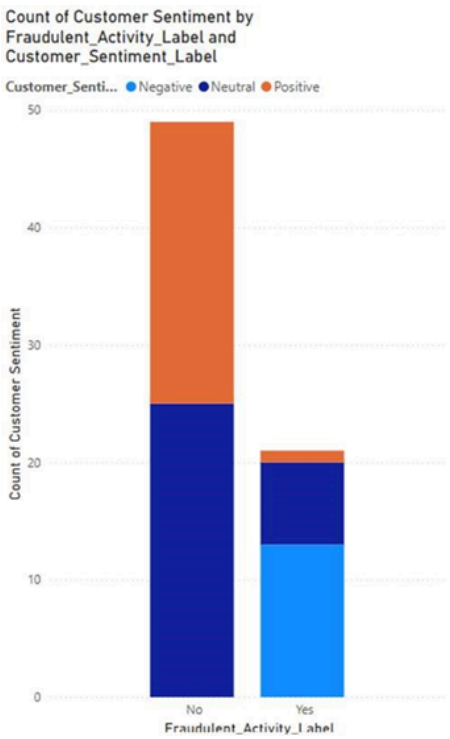
★ Agents & Customers both have Negative Sentiment in Fraud Call

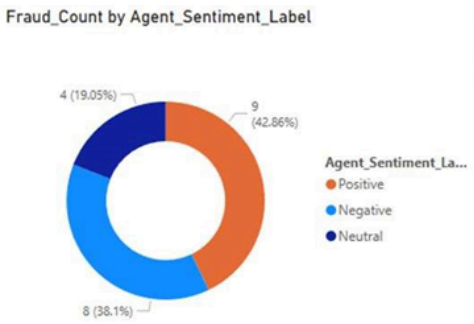
Fraud Calls:

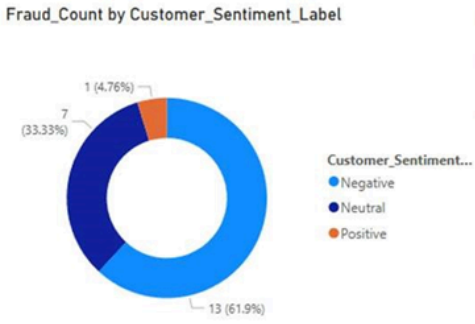
- In fraud calls agents and customers both show negative sentiment.
- Agents are more positive than customers in fraud calls.

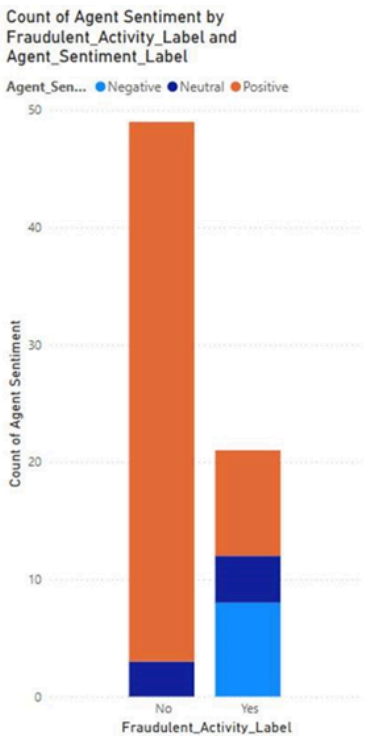
Non Fraud calls:

- In legitimate class agents are extremely positive and show no negative sentiment.
- Customers are also more neutral and positive too during the call.



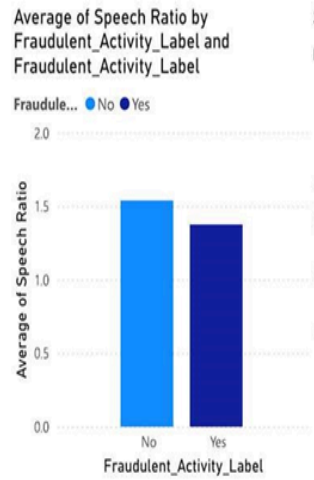


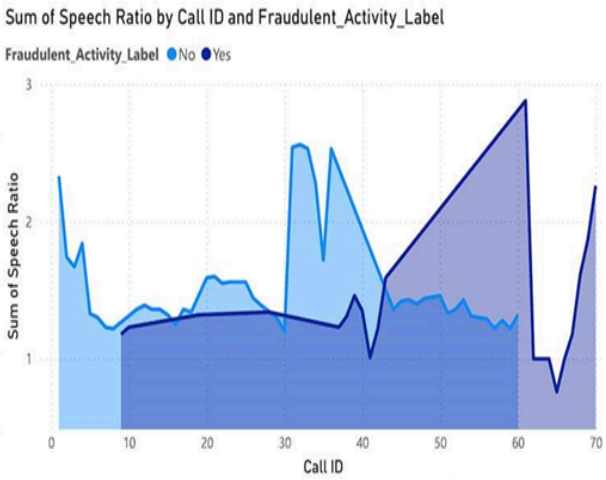


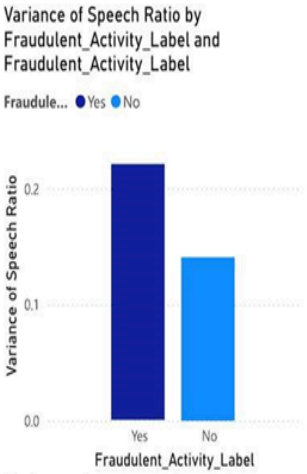


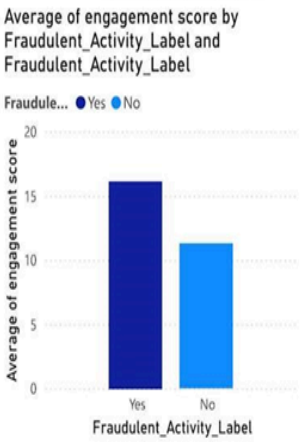
★ Fraud Call are more engaging and variable

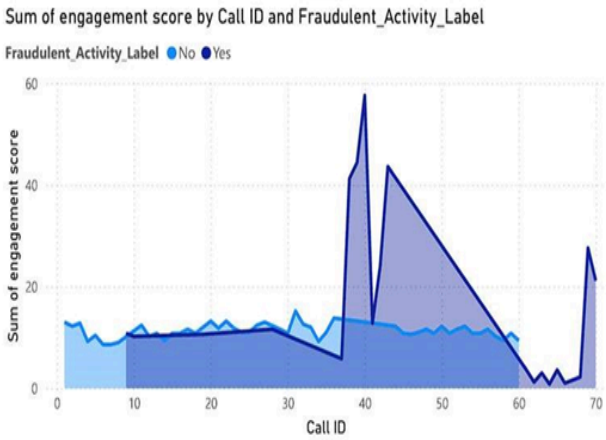
- Agent tends to speak less relative to customer during a fraud call contrary to our belief the margin is thin possible reasons could unbalanced outcome variable.
- The engagement score in fraud calls is significantly higher explaining that it is hard to convince neutral and skeptical customers.
- The Var in speech ratio and engagement score is higher explaining the uncertainty unstructured and unpredictable in the fraud conversation,whereas the legit calls are more structured and follows a consistent pattern.

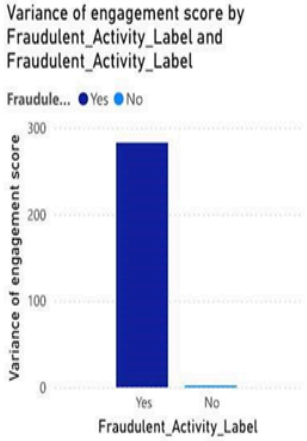




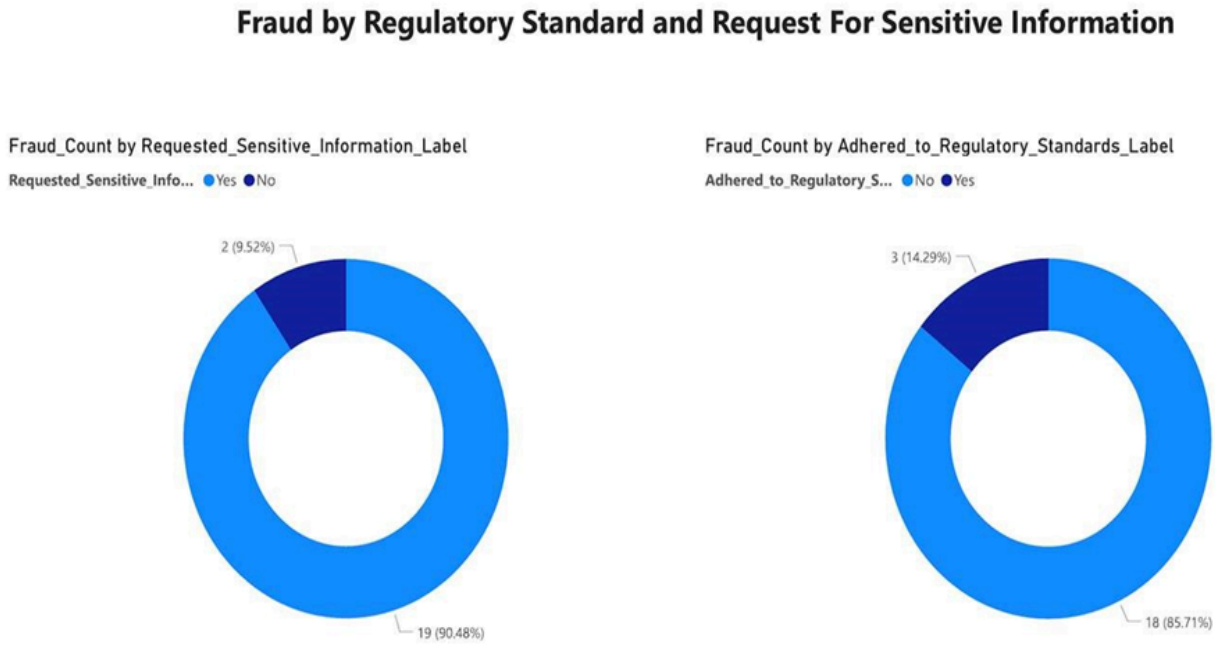






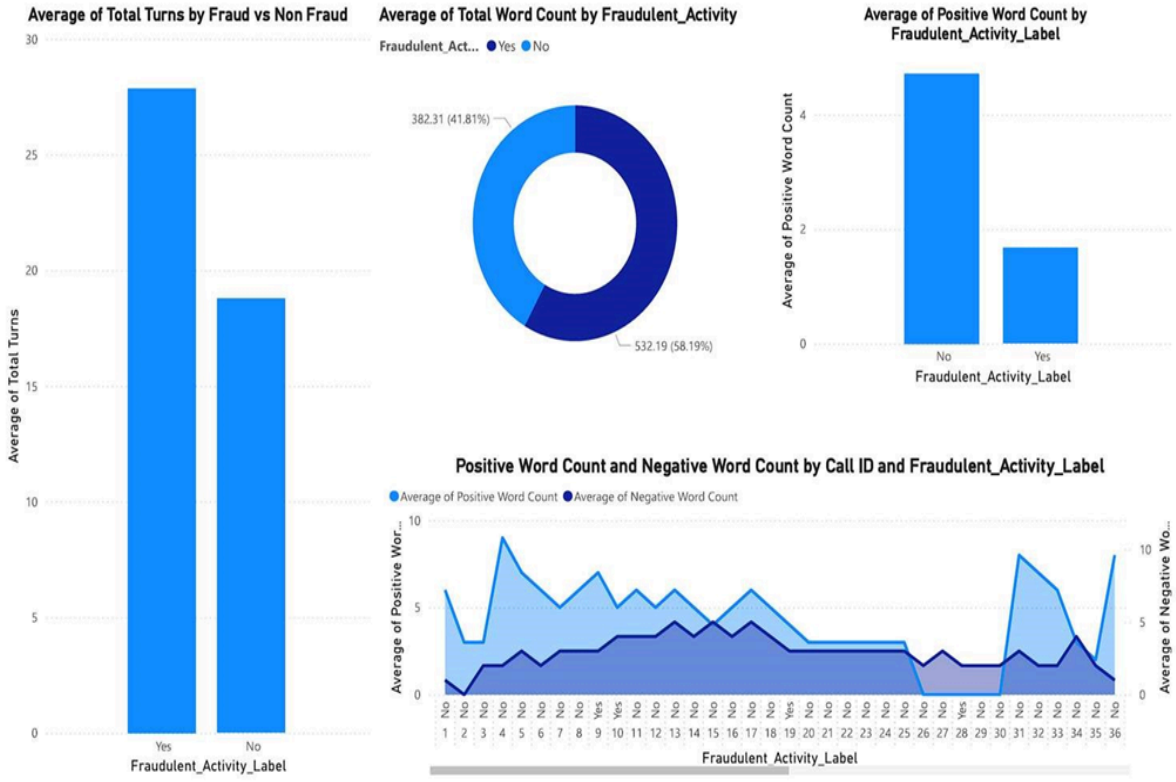


★ Fraud Call violate security Protocols



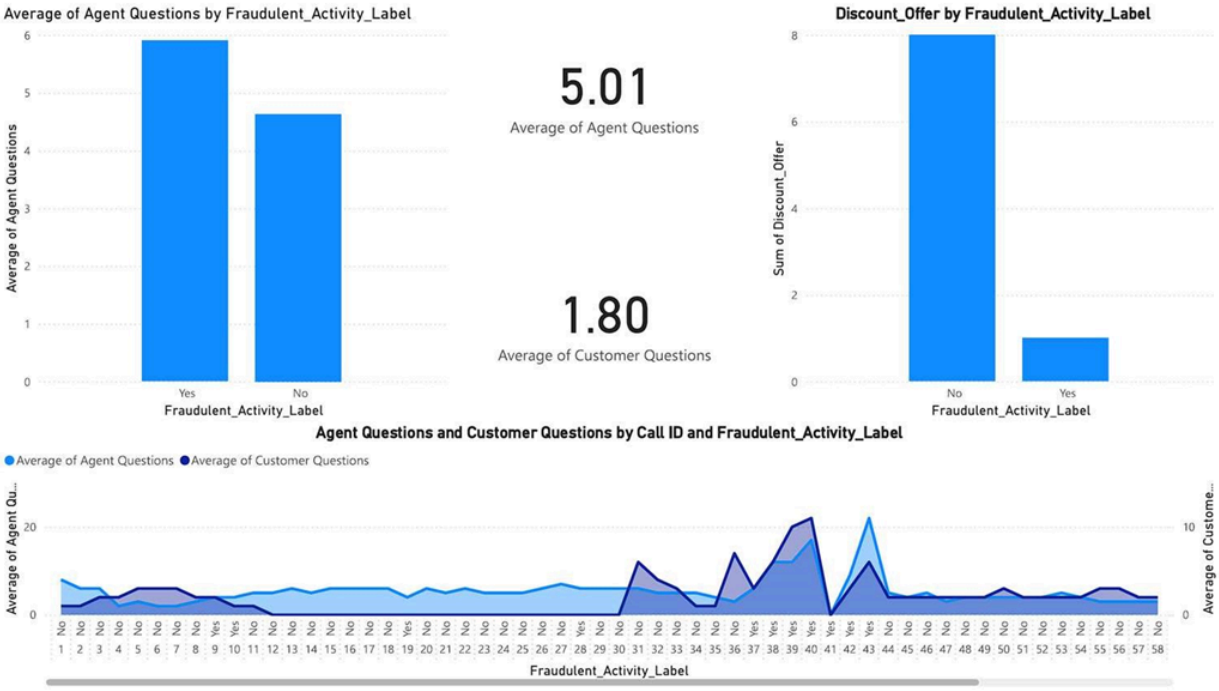
- Fraud Calls Show NonCompliance with Security Measure
- Over 90% of fraud calls Request For Sensitive Information and 86% of Fraud calls does not Adhere to regulatory standard.

★ Fraud Agents Knows less Positive words



- Fraud calls higher turns indicating more back and forth exchanges validating the higher engagement score.
- Higher engagement also leads to higher word count in fraud calls.
- Negative words dominate the fraud conversation over positive words.
- Low positive word counts, higher turn out and higher word count are possible fraud risk factors.

★ Fraud Agents Ask More and Offer Less



- Fraudsters try to extract info, try to lead the conversation and control the narrative, making them ask more questions than customers.
- Fraudsters rarely give offers or discounts because they are not selling, they are tricking the customers rather than providing incentives.

6.Comparison with other models

Alternative methods :

☐ TF-IDF for Machine Learning

TF-IDF stands for *term frequency-inverse document frequency* and it is a measure, used in the fields of information retrieval (IR) and machine learning, that can quantify the importance or relevance of string representations (words, phrases, lemmas, etc) in a document amongst a collection of documents (also known as a corpus).
Term frequency works by looking at the frequency of a *particular term* you are concerned with relative to the document. There are multiple measures, or ways, of defining frequency:

- Number of times the word appears in a document (raw count).
- Term frequency adjusted for the length of the document (raw count of occurences divided by number of words in the document).
- Logarithmically scaled frequency (e.g. log(1 + raw count)).
- Boolean frequency (e.g. 1 if the term occurs, or 0 if the term does not occur, in the document).

What is IDF (inverse document frequency)?

Inverse document frequency looks at how common (or uncommon) a word is amongst the corpus. IDF is calculated as follows where ***t*** is the term (word) we are looking to measure the commonness of and ***N*** is the number of documents (d) in the corpus (D).. The denominator is simply the number of documents in which the term, ***t***, appears in.

$$idf(t, D) = \log\left(\frac{N}{count(d \in D : t \in d)}\right)$$

There are three main applications for TF-IDF. These are in *machine learning, information retrieval,* and *text summarization/keyword extraction.*

Scikit-Learn

- $\text{IDF}(t) = \log\frac{1+n}{1+\text{df}(t)} + 1$

Standard notation

- $\text{IDF}(t) = \log\frac{n}{\text{df}(t)}$

- **Using TF-IDF in machine learning & natural language processing**

Machine learning algorithms often use numerical data, so when dealing with textual data or any natural language processing (NLP) task, a sub-field of ML/AI dealing with text, that data first needs to be converted to a vector of numerical data by a process known as vectorization.

- **Using TF-IDF in information retrieval**

TF-IDF also has use cases in the field of information retrieval, with one common example being search engines. Since TF-IDF can tell you about the relevant importance of a term based upon a document, a search engine can use TF-IDF to help rank search results based on relevance, with results which are more relevant to the user having higher TF-IDF scores.

- **Using TF-IDF in text summarization & keyword extraction**

Since TF-IDF weights words based on relevance, one can use this technique to determine that the words with the highest relevance are the most important. This can be used to help summarize articles more efficiently or to simply determine keywords (or even tags) for a document.

☐ Bag of Words

Bag of Words (BoW) simply counts the frequency of words in a document. Thus the vector for a document has the frequency of each word in the corpus for that document. The key difference between bag of words and TF-IDF is that the former does not incorporate any sort of inverse document frequency (IDF) and is only a frequency count (TF).

☐ Word2Vec

Word2Vec is an algorithm that uses shallow 2-layer, not deep, neural networks to ingest a corpus and produce sets of vectors. Some key differences between TF-IDF and word2vec is that TF-IDF is a statistical measure that we can apply to terms in a document and then use that to form a vector whereas word2vec will produce a vector for a term and then more work may need to be done to convert that set of vectors into a singular vector or other format. Additionally TF-IDF does not take into consideration the context of the words in the corpus whereas word2vec does.

☐ BERT - Bidirectional Encoder Representations from Transformers

BERT is an ML/NLP technique developed by Google that uses a transformer based ML model to convert phrases, words, etc into vectors. Key differences between TF-IDF and BERT are as follows: TF-IDF does not take into account the semantic meaning or context of the words whereas BERT does. Also BERT uses deep neural networks as part of its architecture, meaning that it can be much more computationally expensive than TF-IDF which has no such requirements.

Feature	TF-IDF	Bag of Words (BoW)	Word2Vec	BERT
Technique Type	Statistical	Statistical	Neural embedding	Deep contextual embedding
Context Awareness	No	No	Partial (semantic similarity)	Full (bidirectional & contextual)
Handles Word Order	No	No	No	Yes
Semantic Understanding	None	None	Captures similarity	Strong, contextual meaning
Computational Cost	Low	Very Low	Medium (requires training/lookup)	High (needs GPU, large models)
Pre-trained Models Available	Not required (corpus-specific)	Not required	Available (Google, Gensim)	Yes (HuggingFace, Google)
Sparse vs Dense Vectors	Sparse	Sparse	Dense	Dense
Memory Usage	Medium to High (large vocab size)	High (very large sparse matrix)	Low (compact vector size)	High (larger model size)
High (larger model size)	Fast for small datasets	Very fast	Needs setup	Slower due to complexity

Traditional fraud detection relied heavily on manual auditing, keyword matching, and basic rule-based systems, which were time-consuming, costly, and prone to missing subtle fraud indicators.In contrast, the proposed system using LLaMA 3.2 hosted via Ollama brings deep contextual understanding, automatic feature extraction, and rich fraud analytics.
It is scalable, secure, adaptable to new fraud tactics through prompt updates, and cost-effective, thus offering a far superior solution for modern sales environments.

Component	Implementation in Project	Advantage
Large Language Model (LLaMA 3.2)	Understands full conversation context, not just keywords. Extracts semantic, sentiment, compliance, engagement, and fraud-specific features via dynamic prompts.	Captures subtle fraud signals missed by rules or keywords. Adapts better to real conversational variability.
Prompt Engineering	Custom prompts guide LLaMA to generate structured JSON outputs tailored to fraud indicators.	Flexible: By changing prompts, new features can be extracted without retraining.
Full Automation (Python Script)	Automates ingestion, prompt generation, API interaction, JSON parsing, and Excel output.	Highly scalable and removes manual dependencies.
Local Inference (Ollama)	Runs LLaMA models locally without needing external cloud services.	Secure, fast, cost-effective, and data privacy compliant.
Multi-Feature Extraction	Extracts 40+ features including speech ratio, engagement scores, trust issue flags, tone analysis, compliance breaches, etc.	Richer feature space leads to much stronger fraud detection signals compared to simple sentiment or keyword-based models.
Visualization (Power BI)	Final outputs visualized in interactive dashboards showing fraud trends, agent behavior, and customer reactions.	Enables managers to monitor patterns in real time, improving operational decision-making.

7.Conclusion

This project successfully demonstrates the development of an AI-powered fraud detection system designed to analyze sales call transcripts and identify suspicious patterns using Natural Language Processing (NLP) and large language models (LLMs). By integrating the open-source **LLaMA 3.2 model** via **Ollama**, and automating data extraction with Python, the system converts raw, unstructured conversations into structured datasets with over 40 fraud-related features.

Key findings from the analysis show that:

- **Fraud calls exhibit higher engagement**, more negative sentiment, and a greater number of dialogue turns.
- **Compliance violations** such as requesting sensitive information or ignoring regulatory protocols were present in over **85–90%** of fraudulent calls.
- Fraud agents were often more controlling—**asking more questions** while offering **fewer incentives** like discounts or testimonials.
- **Speech patterns**, **tone classifications**, and **keyword trends** differed notably between legitimate and fraudulent interactions.

The final outputs, visualized in **Power BI dashboards**, enabled intuitive exploration of fraud risks, agent behaviors, and common red flags across calls. The pipeline proved to be scalable, efficient, and highly interpretable, reducing manual review efforts and accelerating fraud detection workflows.

This system validates the use of LLMs in enterprise fraud detection and lays the groundwork for future real-time, multilingual, and voice-integrated monitoring solutions.

8. Recommendations for Future Work

While the project has delivered an effective fraud analytics prototype, there are several opportunities for improvement and expansion:

🔹 **1. Fine-Tuning LLaMA**

- Train the model on a domain-specific corpus (e.g., insurance or banking sales calls) to enhance accuracy for niche fraud patterns.

🔹 **2. Real-Time Deployment**

- Integrate the model into live call monitoring systems to provide **instant fraud risk scoring** and alert supervisors in real-time.

🔹 **3. Voice-to-Text Conversion**

- Add an **automatic speech recognition (ASR)** layer using tools like **OpenAI Whisper** to analyze audio recordings directly, eliminating the need for manual transcription.

🔹 **4. Multilingual Capability**

- Extend fraud detection to cover **regional Indian languages** or **international markets** by including language support for Hindi, Bengali, Spanish, etc.

🔹 **5. Expanded Feature Set**

- Extract deeper behavioral features such as **emotion tagging**, **response latency**, and **conversation flow disruption** to identify more subtle fraud attempts.

9. References

1. Meta AI. (2024). *LLaMA: Open and Efficient Foundation Language Models*.
   Retrieved from https://ai.meta.com/llama

2. Ollama. (2024). *Run Large Language Models Locally*.
   Retrieved from https://ollama.com

3. OpenAI. (2023). *Whisper: Robust Speech Recognition via Large-Scale Weak Supervision*.
   Retrieved from https://openai.com/research/whisper

4. Microsoft. (2024). *Power BI Documentation*.
   Retrieved from https://learn.microsoft.com/en-us/power-bi/

5. Python Software Foundation. (2024). *Python Documentation (v3.10+)*.
   Retrieved from https://docs.python.org/3/

6. Pandas Development Team. (2024). *pandas: Python Data Analysis Library*.
   Retrieved from https://pandas.pydata.org/

7. Hugging Face. (2023). *Prompt Engineering Guide*.
   Retrieved from https://huggingface.co/learn/prompt-engineering

8. Rao, A., & Potdar, K. (2022). *AI-Based Fraud Detection Using NLP on Call Transcripts*.
   International Journal of Computer Applications, 184(21), 10-17.

9. Stack Overflow. (2023). *How to Parse JSON in Python*.
   Retrieved from https://stackoverflow.com/questions/45876903/parse-json-response-in-python