Detailed Design Document

Collaborators: Shoval Benjer 319037404, Adir Amar 209017755, Alon Berkovich 208432625

Database Schema:

As our explicit way of achieving our goals is through providing as quick as immediate solutions to customers, a sufficient data base is required. Amongst various self-explanatory reasons to have as robust and rich data base as can be, there is one that stands above all – to minimize the amount of information asked of a customer for achieving a single valued identification to his case. The fact of the matter is that the more variables our data base contains, the easier and faster it will be to determine a response, the easier it will be to relieve the customers of repetitive information supplies and most importantly – the easier it is to SATISFY our customers.

Amongst, of course, the customer's personal information, a sufficient database will include system logs (such as prior inquiries, dates, IDs, statuses, Channel of Complaint, etc.), user's input data (such as account types and Consumer Complaint Narrative - which is the free text of complaint received by the customer) and our NLP model derived data (such as text sentiment scores, urgency Levels, Issue classification and missing information identification).

The data we chose as an example is of a banking complaints system. The data displayed here is tabular, and consists of various information describing instances of complaints. Within the various information are key values such as "Date received", "Complaint ID", "Customer ID" and combinations to ensure a single valued identification of an inquiry. Furthermore, some column's values serve as categories for other column's to better clarify on the contents of an inquiry. for example, the "Sub-issue" column provides a more "in depth" description to the "Issue" column. Most importantly, and this is where we step in, the "Consumer complaint narrative" column provides a free text, by the customer complaining, to feed NLP model to analyze and to determine a "verdict".

From there on out it is fully up to our product to better the lives of the customers and businesses alike - by substantially facilitating customers user experience, by befriending the customers and by prompting them to further engage with the system, by optimizing workload distributions and output consistency and eventually, by significantly increasing customer satisfaction rates.

1.1 Users Table

Column Name	Data Type	Constraints
user_id	SERIAL	PRIMARY KEY
username	VARCHAR	UNIQUE, NOT NULL
Email	VARCHAR	UNIQUE, NOT NULL
password	TEXT	NOT NULL
Role	VARCHAR	CHECK (role IN ('customer', 'support_agent', 'admin')), NOT NULL
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP

1.2 Complaints Table

Column Name	Data Type	Constraints
complaint_id	SERIAL	PRIMARY KEY
user_id	INTEGER	REFERENCES Users(user_id)
category_id	INTEGER	REFERENCES Categories(category_id)
status_id	INTEGER	REFERENCES ComplaintStatus(status_id)
description	TEXT	NOT NULL
submitted_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP
last_updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP

1.3 Categories Table

Column Name	Data Type	Constraints
category_id	SERIAL	PRIMARY KEY
Name	VARCHAR	UNIQUE, NOT NULL
description	TEXT	

1.4 ComplaintStatus Table

Column Name	Data Type	Constraints
status_id	SERIAL	PRIMARY KEY
status_name	VARCHAR	UNIQUE, NOT NULL
description	TEXT	

1.5 ComplaintHistory Table

Column Name	Data Type	Constraints

Column Name	Data Type	Constraints
history_id	SERIAL	PRIMARY KEY
complaint_id	INTEGER	REFERENCES Complaints(complaint_id)
status_id	INTEGER	REFERENCES ComplaintStatus(status_id)
changed_by	INTEGER	REFERENCES Users(user_id)
changed_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP
change_comments	TEXT	

1.6 Indexing Strategies

1.6.1 Primary Keys and Foreign Keys:

- **Ensure each table has a primary key for unique identification.**
- **Foreign keys establish relationships between tables, enforcing referential integrity.**

1.6.2 Indexes on Foreign Key Columns:

- **Create indexes on columns frequently used in JOIN operations to enhance query performance.**
- **For example, indexing user_id in the Complaints table improves retrieval of all complaints submitted by a specific user.**

1.6.3 Partial Indexes:

- **Use partial indexes for queries that filter on specific conditions.**
- **For instance, if most queries involve open complaints, create an index on the Complaints table where status_id corresponds to 'open'.**

```
CREATE INDEX idx_open_complaints
ON Complaints (status_id)
WHERE status_id = (SELECT status_id FROM ComplaintStatus WHERE status_name =
'open');
```

1.6.4 GIN Indexes for Full-Text Search:

- **If the system supports searching within complaint descriptions, implement a GIN (Generalized Inverted Index) on the description column to accelerate full-text search operations.**

```
CREATE INDEX idx_complaint_description
ON Complaints
USING gin(to_tsvector('english', description));
```

1.6.5 Regular Maintenance:

- Perform routine maintenance tasks such as VACUUM ANALYZE to update statistics and optimize query planning.

```
VACUUM ANALYZE;
```

Data Integrity and Validation

- **Data Types:**
  - Choose appropriate data types to prevent invalid data entries.
  - For example, use VARCHAR with length constraints for usernames and emails to enforce reasonable limits.
- **CHECK Constraints:**
  - Implement CHECK constraints to enforce valid data ranges or formats.
  - For instance, ensure the role column in the Users table only contains predefined values.

```
CREATE TABLE Users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password TEXT NOT NULL,
    role VARCHAR(20) NOT NULL CHECK (role IN ('customer', 'support_agent', 'admin')),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **NOT NULL Constraints:**
  - Apply NOT NULL constraints to columns that require mandatory data, ensuring essential information is always provided.

Performance Considerations

- **Query Optimization:**
  - Analyze query performance using the EXPLAIN command to understand execution plans and identify bottlenecks.

```
EXPLAIN ANALYZE
SELECT * FROM Complaints
WHERE user_id = 1;
```

- **Connection Pooling:**
  - Implement connection pooling to manage database connections efficiently, reducing overhead and improving performance.
- **Load Balancing:**
  - Distribute database load across multiple servers to enhance performance and ensure high availability.

Scalability and Maintenance

- **Partitioning:**
    - **Consider table partitioning for large tables to improve query performance and maintenance.**
    - **For example, partition the Complaints table by date ranges if the dataset is extensive.**
- **Regular Backups:**
    - **Schedule regular backups to prevent data loss and ensure quick recovery in case of failures.**

1. Validation Rules:

1.1 Data Integrity Constraints:

- CHECK Constraints: Enforce specific conditions on data entries to maintain validity. For example, to ensure that complaint descriptions are not empty:

```
ALTER TABLE Complaints
ADD CONSTRAINT description_not_empty CHECK (description <> '');
```

This constraint prevents the insertion of complaints with empty descriptions.

Sling Academy

- Regular Expressions: Utilize regular expressions within CHECK constraints to validate data formats. For instance, to validate that an email column contains properly formatted email addresses:

```
ALTER TABLE Users
ADD CONSTRAINT valid_email CHECK (email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');
```

This ensures that only valid email formats are accepted.

Sling Academy

2. Referential Integrity:

- Foreign Key Constraints: Ensure that relationships between tables remain consistent. For example, linking a complaint to a valid user:

```
ALTER TABLE Complaints
ADD CONSTRAINT fk_user
FOREIGN KEY (user_id) REFERENCES Users(user_id);
```

This guarantees that each complaint is associated with an existing user.

3. Enumerated Types for Statuses:

- ENUM Types: Define enumerated types for columns that should only contain a limited set of values. For instance, to restrict complaint statuses:

```
CREATE TYPE complaint_status AS ENUM ('open', 'in_progress', 'resolved', 'closed');
ALTER TABLE Complaints
```

```
ADD COLUMN status complaint_status NOT NULL DEFAULT 'open';
```

This ensures that only predefined statuses can be assigned to complaints.

Performance Benchmarks

Establishing performance benchmarks is crucial for assessing and maintaining the efficiency of your system. Consider the following metrics:

1. Response Time Metrics:

- Database Query Performance: Aim for query response times under 100 milliseconds for standard operations. Regularly monitor and optimize slow queries using PostgreSQL's EXPLAIN ANALYZE to identify bottlenecks.
- API Response Time: Ensure that API endpoints respond within 200 milliseconds under typical load conditions to provide a responsive user experience.

2. Throughput Metrics:

- Transactions Per Second (TPS): Measure the number of transactions the system can handle per second. For a high-volume complaint management system, a benchmark of 1,000 TPS is desirable. Recent benchmarks indicate that PostgreSQL 17 offers a 2% performance improvement over version 16, highlighting the importance of staying updated with the latest versions for optimal performance.

Data System Reviews

3. Scalability Considerations:

- Connection Pooling: Implement connection pooling to manage database connections efficiently, reducing overhead and improving performance.
- Load Testing: Conduct regular load testing to ensure the system can handle peak loads, such as 10,000 complaints per hour, without degradation in performance.

By implementing these validation rules and establishing clear performance benchmarks, you can enhance the reliability and efficiency of your complaint management system. Regular monitoring and optimization are key to maintaining optimal performance as your system scales.

2. Frameworks:

Frontend:

- **Next.js (React Framework):** For server-side rendering and static site generation, ensuring performance and SEO.
- **TypeScript:** Adds type safety, improving code quality and maintainability.
- **Tailwind CSS:** Simplifies UI design with utility-first CSS classes.

Backend:

- **FastAPI or Flask:** Lightweight, modular frameworks for building RESTful APIs. FastAPI is recommended for asynchronous support and performance.
- **PostgreSQL:** A robust, SQL-based relational database for structured data handling and advanced querying.
- **MongoDB (Optional):** Suitable for applications requiring flexible schema design and unstructured data handling.

NLP Integration:

- **Hugging Face Transformers:** For state-of-the-art NLP tasks (text classification, sentiment analysis) with extensive model support.
- **spaCy:** Optimized for efficient NLP with pre-trained models for practical applications.
- **Cohere API:** For cutting-edge NLP models, particularly for text generation and classification.

Development Tools:

- **VS Code + GitHub Copilot:** Enhances developer productivity with AI-assisted coding and real-time collaboration.
- **GitHub Actions:** For CI/CD workflows to automate testing and deployment.

Deployment:

- **Vercel:** Optimized for Next.js applications with a free tier for hosting and deployment. Seamless integration with GitHub Actions for CI/CD.

3. NLP Workflow

3.1 Data Preprocessing: Refining Inputs

- **Advanced Cleaning**: Expand text cleaning to include:
    - Spell-checking with context-aware algorithms.
    - Handling emojis and special domain-specific terminology.
- **Domain-Specific Stop Words**: Create a custom stop word list relevant to your complaint context.
- **Semantic Lemmatization**: Use WordNet or spaCy's language models to map words into complaint-specific semantics.
- **Augmentation**: Use paraphrasing models to enrich training datasets and address underrepresented complaint types.

3.2 Feature Extraction: Capturing Meaning

- **Hybrid Features**:
  - Use **TF-IDF** for sparse, high-frequency word analysis.
  - Integrate **BERT embeddings** for semantic depth.
- **Custom Embedding Layers**:
  - Train embeddings using Word2Vec or FastText on your domain-specific data to capture contextual nuances.
- **Attention Mechanisms**:
  - Implement attention layers to weigh critical complaint components during classification.

3.3 Classification: Tailoring the Model

- **Model Selection**:
  - **Lightweight Models**: For simpler pipelines, try Logistic Regression or SVM with tuned hyperparameters.
  - **Deep Learning**: Use pre-trained BERT models like DistilBERT (for speed) or fine-tune RoBERTa for classification tasks.
  - Consider **multi-label classification** if complaints can belong to more than one category.
- **Optimization**:
  - Use grid search or Optuna for hyperparameter tuning.
  - Regularize deep learning models with dropout or weight decay to prevent overfitting.

3.4 Sentiment Analysis: Contextual Understanding

- **Fine-Tuned Models**:
  - Train BERT or RoBERTa for sentiment tasks with domain-specific labels (e.g., frustration, urgency).
- **Aspect-Based Sentiment Analysis**:
  - Use models like Aspect-based Sentiment Analysis (ABSA) to capture sentiment related to specific complaint elements.

3.5 Post-Processing and Evaluation

- **Confidence Calibration**:
  - Use Platt Scaling or Isotonic Regression to ensure confidence scores align with probabilities.
- **Misclassification Handling**:

    - Introduce manual review workflows for low-confidence predictions.
- **Explainability**:
    - Employ SHAP or LIME to generate interpretability metrics for the classification pipeline.

3.6 System Integration and Real-Time Capability

1. API Design

- Use FastAPI or Flask to wrap your model, providing endpoints for:
    - Complaint classification.
    - Sentiment analysis.
    - Confidence evaluation.

2. Database Integration

- Implement PostgreSQL or MongoDB to store categorized complaints and model feedback for retraining.
- Use ER diagrams to design schemas optimized for query performance.

3. Real-Time Processing

- **Batch vs. Stream Processing**:
    - Implement Kafka or RabbitMQ for streaming large complaint volumes.
    - Use asynchronous model inference for low-latency responses.

4. Continuous Improvement

- Deploy automated retraining pipelines with Airflow or Prefect to incorporate new complaint data.
- Regularly monitor model drift using evaluation dashboards.

For a practical implementation of a BERT-based NLP pipeline for classifying customer complaints: NLP-Based Customer Complaint Classification Using BERT

3.7 Integrated NLP Workflow

1. **Input**:
    - Customer complaints (textual data) received from various channels (email, app, website, etc.).
2. **Data Ingestion**:
    - Collect incoming complaints via APIs or batch uploads.

    - Store raw data in a centralized database or processing pipeline.
3. **Preprocessing**:
    - **Step 1**: Clean and normalize text (remove special characters, lowercase conversion, stopword removal).
    - **Step 2**: Tokenize and lemmatize text to reduce to base forms.
    - **Step 3**: Identify and handle missing or irrelevant data (e.g., invalid complaints).
4. **Feature Extraction**:
    - Convert cleaned text into numerical representations (e.g., TF-IDF, embeddings).
    - Include metadata (e.g., complaint timestamp, customer ID) as additional features.
5. **Classification**:
    - Apply a trained machine learning model to categorize the complaint into predefined categories (e.g., product issue, delivery delay).
    - Assign a confidence score to each prediction.
6. **Sentiment Analysis**:
    - Analyze the tone of the complaint (positive, neutral, negative).
    - Attach a sentiment confidence score for reliability.
7. **Post-Processing**:
    - Combine category and sentiment results into a structured report for each complaint.
    - Flag low-confidence cases for manual review.
8. **Storage and Integration**:
    - Store results (categories, sentiments, metadata) in the database.
    - Provide structured reports to a customer success team or integrate them into a ticketing system.
9. **Feedback and Continuous Learning**:
    - Monitor system performance in real-time.
    - Retrain models periodically based on flagged or manually corrected cases.
10.

3.8 Workflow Pseudocode in steps:

START

1. Receive complaint data from API or batch upload.

2. Preprocess the complaint text:

- Clean the text (remove special characters, lowercase, remove stopwords).

- Tokenize and lemmatize.

3. Extract features:

- Convert text to numerical format (e.g., embeddings or TF-IDF).

- Include metadata for context.

4. Classify the complaint:

- Use a classification model to assign a category.

- Calculate confidence scores.

5. Analyze sentiment:

- Predict sentiment (positive, neutral, negative).

- Assign confidence scores.

6. Generate a report:

- Include category, sentiment, confidence scores, and metadata.

- Flag low-confidence cases for manual review.

7. Store results in a database or ticketing system.

8. Monitor performance and collect user feedback for retraining.

9. End.

STOP

4. Performance Benchmarks

- **Objective: Establish query and API response time targets for efficient system performance.**
- **Solution:**
    - **Database Queries:**

        - **Average response time under 100ms for simple queries.**
        - **Maximum latency of 300ms for complex joins or aggregations.**
    - **API Endpoints:**
        - **Complaint submission: <200ms.**
        - **Real-time updates (e.g., complaint status changes): <150ms.**
        - **Notifications: <100ms for push and email events.**
    - **High Load Handling:**
        - **System should handle 10,000 complaints/hour with no more than 5% degradation in response time.**
    - **Implementation Tools:**
        - **PostgreSQL EXPLAIN ANALYZE for query optimization.**
        - **Profiling tools like New Relic for API monitoring.**

5. Scalability Plan

- **Objective: Ensure the database and system scale with increasing complaint volumes.**
- **Solution:**
    - **PostgreSQL Enhancements:**
        - **Horizontal Scaling: Use sharding to distribute large datasets across multiple nodes.**
        - **Read Replicas: Set up replicas to distribute read-heavy operations like analytics queries.**
        - **Connection Pooling: Use tools like PgBouncer to manage connections efficiently.**
    - **Caching:**
        - **Implement Redis for frequently accessed data (e.g., user sessions, resolved complaint summaries).**
    - **Partitioning:**
        - **Partition complaints table by date to improve performance on time-based queries.**
    - **Cloud Services:**

        - **Use AWS RDS for managed PostgreSQL with autoscaling capabilities.**

6. Backup and Recovery Plan

- **Objective: Prevent data loss and ensure quick recovery during failures.**
- **Solution:**
    - **Automated Backups:**
        - **Daily full backups and hourly incremental backups using PostgreSQL's pg_dump or AWS RDS snapshots.**
    - **Disaster Recovery:**
        - **Configure point-in-time recovery (PITR) for accidental data deletion.**
        - **Maintain offsite backups for added redundancy.**
    - **Monitoring:**
        - **Use tools like pgAdmin for backup scheduling and monitoring.**
    - **Testing:**
        - **Regularly validate backup integrity and recovery time objectives (RTO < 30 minutes).**
    - **Implementation Tools:**
        - **AWS RDS snapshots, pg_basebackup, and third-party tools like pgBackRest.**

7. Stress Testing

- **Objective: Validate system performance under heavy loads.**
- **Solution:**
    - **Tools:**
        - **Locust: For simulating concurrent complaint submissions and user actions.**
        - **JMeter: For API stress testing.**
        - **k6: For scalable performance testing.**
    - **Testing Scenarios:**
        - **Simulate 10,000 complaints/hour with varying data complexity.**
        - **Stress test API endpoints under high concurrency (e.g., 1,000 simultaneous users).**

- **Metrics to Monitor:**
    - **Latency, throughput, and error rates.**
    - **Database transaction success rates under peak load.**
- **Automation:**
    - **Use CI/CD pipelines (GitHub Actions) to automate regular performance tests.**

8. API Documentation

- **Objective: Provide clear and comprehensive documentation for all system APIs.**
- **Solution:**
    - **Tools:**
        - **Use OpenAPI (Swagger) for API specification and documentation generation.**
    - **Documentation Structure:**
        - **Endpoints: Example:**
            - **POST /complaints: Create a new complaint.**
            - **GET /complaints/{id}: Retrieve complaint details.**
        - **Request/Response Formats:**
            - **Include JSON payload examples.**
        - **Authentication:**
            - **Document OAuth or token-based authentication methods.**
        - **Error Codes:**
            - **Define standard error responses (e.g., 400 Bad Request, 404 Not Found).**

9. Appendices

1. Literature Review and Competitor Analysis.
2. Project Repository -
3. Project Roadmap

4. UI/UX Sketch
5. Flowchart diagram sketch
6. Issues by Categories for Data source “Consumer Complaint Database”:

Issue	**General Category**
Account opening, closing, or management	Account Management Issues
Account terms and changes	Account Management Issues
Adding money	Transaction and Purchase Issues
Advertising	Product or Service Issues
Advertising and marketing	Product or Service Issues
Advertising and marketing, including promotional offers	Product or Service Issues
Advertising, marketing or disclosures	Product or Service Issues
Application processing delay	Application and Approval Issues
Application, originator, mortgage broker	Loan and Mortgage Issues
Applied for loan/did not receive money	Application and Approval Issues
Applying for a mortgage	Loan and Mortgage Issues
Applying for a mortgage or refinancing an existing mortgage	Loan and Mortgage Issues
APR or interest rate	Loan and Mortgage Issues
Arbitration	Product or Service Issues
Attempts to collect debt not owed	Debt Collection and Communication Issues
Balance transfer	Card Issues

Issue	General Category
Balance transfer fee	Billing, Fees, and Payment Issues
Bankruptcy	Loan and Mortgage Issues
Billing disputes	Billing, Fees, and Payment Issues
Billing statement	Billing, Fees, and Payment Issues
Can't contact lender	Customer Service Issues
Can't contact lender or servicer	Customer Service Issues
Can't repay my loan	Loan and Mortgage Issues
Can't stop charges to bank account	Billing, Fees, and Payment Issues
Can't stop withdrawals from your bank account	Billing, Fees, and Payment Issues
Cash advance	Card Issues
Cash advance fee	Billing, Fees, and Payment Issues
Charged bank acct wrong day or amt	Billing, Fees, and Payment Issues
Charged fees or interest I didn't expect	Billing, Fees, and Payment Issues
Charged fees or interest you didn't expect	Billing, Fees, and Payment Issues
Closing an account	Account Management Issues
Closing on a mortgage	Loan and Mortgage Issues
Closing your account	Account Management Issues
Closing/Cancelling account	Account Management Issues
Communication tactics	Debt Collection and Communication Issues
Confusing or misleading advertising or marketing	Product or Service Issues

Issue	General Category
Confusing or missing disclosures	Product or Service Issues
Cont'd attempts collect debt not owed	Debt Collection and Communication Issues
Convenience checks	Transaction and Purchase Issues
Credit card protection / Debt protection	Card Issues
Credit decision / Underwriting	Card Issues
Credit determination	Card Issues
Credit limit changed	Account Management Issues
Credit line increase/decrease	Account Management Issues
Credit monitoring or identity protection	Credit Reporting Issues
Credit monitoring or identity theft protection services	Credit Reporting Issues
Credit reporting company's investigation	Credit Reporting Issues
Customer service / Customer relations	Customer Service Issues
Customer service/Customer relations	Customer Service Issues
Dealing with my lender or servicer	Loan and Mortgage Issues
Dealing with your lender or servicer	Loan and Mortgage Issues
Delinquent account	Loan and Mortgage Issues
Deposits and withdrawals	Transaction and Purchase Issues
Disclosure verification of debt	Debt Collection and Communication Issues
Disclosures	Product or Service Issues
Excessive fees	Billing, Fees, and Payment Issues

Issue	General Category
False statements or representation	Debt Collection and Communication Issues
Fees	Billing, Fees, and Payment Issues
Fees or interest	Billing, Fees, and Payment Issues
Forbearance / Workout plans	Loan and Mortgage Issues
Fraud or scam	Fraud and Security Issues
Getting a credit card	Card Issues
Getting a line of credit	Application and Approval Issues
Getting a loan	Loan and Mortgage Issues
Getting a loan or lease	Loan and Mortgage Issues
Getting the loan	Loan and Mortgage Issues
Identity theft / Fraud / Embezzlement	Fraud and Security Issues
Identity theft protection or other monitoring services	Fraud and Security Issues
Improper contact or sharing of info	Debt Collection and Communication Issues
Improper use of my credit report	Credit Reporting Issues
Improper use of your report	Credit Reporting Issues
Incorrect exchange rate	Transaction and Purchase Issues
Incorrect information on credit report	Credit Reporting Issues
Incorrect information on your report	Credit Reporting Issues
Incorrect/missing disclosures or info	Product or Service Issues
Late fee	Billing, Fees, and Payment Issues

Issue	General Category
Lender damaged or destroyed property	Vehicle and Property Issues
Lender damaged or destroyed vehicle	Vehicle and Property Issues
Lender repossessed or sold the vehicle	Vehicle and Property Issues
Lender sold the property	Vehicle and Property Issues
Loan modification,collection,foreclosure	Loan and Mortgage Issues
Loan payment wasn't credited to your account	Billing, Fees, and Payment Issues
Loan servicing, payments, escrow account	Loan and Mortgage Issues
Lost or stolen check	Transaction and Purchase Issues
Lost or stolen money order	Transaction and Purchase Issues

1. **Account Management Issues**
2. **Application and Approval Issues**
3. **Billing, Fees, and Payment Issues**
4. **Card Issues**
5. **Credit Reporting Issues**
6. **Customer Service Issues**
7. **Debt Collection and Communication Issues**
8. **Fraud and Security Issues**
9. **Loan and Mortgage Issues**
10. **Product or Service Issues**
11. **Transaction and Purchase Issues**
12. **Vehicle and Property Issues**

7. Database columns description:

Column Name	Source Type	Description

Column Name	Source Type	Description
Date Received	System Logs	The date the CFPB received the complaint. For example, "05/25/2013".
Complaint Source Location	System Logs	Geographic location based on the state field (expanded to include city or ZIP code).
Channel of Complaint	System Logs	How the complaint was submitted (e.g., "Web," "Phone").
Complaint Status	System Logs	How the company responded to the complaint (e.g., "Closed with explanation").
Complaint Resolution Time	System Logs	Duration taken to resolve the complaint (calculated from "Date received" and resolution date).
Compensation Provided	System Logs	Details about refunds, replacements, or other compensations linked to complaints.
Recurring Complaint Indicator	System Logs	A flag to identify complaints linked to recurring issues.
Feedback Rating	System Logs	Post-resolution feedback score provided by the consumer.
User Action Logged	System Logs	Actions taken by the user (e.g., Complaint Submitted, Complaint Followed Up).
Customer Support Engagement Count	System Logs	Number of interactions between the user and customer support regarding the complaint.
Escalation Action Logged	System Logs	A log of escalation actions taken by the company on behalf of the user.
Sentiment Score	Created by NLP	Extracted from the complaint narrative to prioritize complaints based on emotional tone.
Complaint Category Confidence Score	Created by NLP	The model's confidence level in classifying the complaint into a specific category.
Urgency Level	Created by	Automatically determined based on keywords,

Column Name	Source Type	Description
	NLP	sentiment, or complaint severity.
Product	User Input	The type of product the consumer identified in the complaint (e.g., "Checking or savings account").
Sub-product	User Input	The type of sub-product identified in the complaint (e.g., "Checking account").
Issue	User Input	The issue identified in the complaint (e.g., "Managing an account").
Sub-issue	User Input	The sub-issue identified in the complaint (e.g., "Deposits and withdrawals").
Consumer Complaint Narrative	User Input	Consumer-submitted description of the complaint. Published only with consumer consent.
Complaint ID	System Logs	A unique identification number for each complaint.
Purchase History	Provided by Company	History of relevant transactions tied to the complaint (e.g., last purchase, frequency).
Refund or Adjustment History	Provided by Company	Details of previous refunds or adjustments made for the user.
Account Type	Provided by Company	The type of account associated with the user (e.g., Checking, Business, Savings).

8. **Example of a Fine-Tuned BERT Pipeline:**