TASK 1: Comparative Study of Prompting Techniques Using Large Language Models

1. Introduction

Task 1 focuses on evaluating how different prompting strategies influence the performance of a Large Language Model (LLM) when classifying customer review sentiments. Modern language models can perform tasks without explicit training, but their effectiveness strongly depends on how the query (prompt) is structured. This task compares the model's accuracy under three prompting techniques:

1. Zero-shot prompting

The model receives only the question and must infer the answer without examples.

2. Few-shot prompting

The model is provided with labeled examples before predicting new samples.

3. Chain-of-Thought (CoT) prompting

The model is instructed to reason step-by-step before producing the final label.

Objective

The objective of Task 1 is to:

- Measure how the prompting style affects model accuracy
- Analyze classification behavior using confusion matrices
- Compare Zero-shot, Few-shot, and CoT performance
- Identify strengths and limitations of each prompting method
- Draw insights on effective prompt engineering for LLMs

Dataset

A cleaned **Yelp Review Rating Dataset** was used, containing:

- **Text reviews**
- **True labels (1 to 5 stars)**

This dataset provides a realistic evaluation of the model's ability to understand customer sentiment and map it to an appropriate rating.

Tools and Technologies Used

- Python
- Perplexity Sonar Pro API
- Pandas
- NumPy
- Scikit-learn (classification metrics + confusion matrix)

- Matplotlib / Seaborn (visualization)

Expected Outcome

By the end of Task 1, we aim to determine:

- Whether **examples (Few-shot)** improve stability
- Whether **reasoning (CoT)** improves accuracy
- How prompting shapes the model's prediction distribution
- Which prompting technique is most suitable for rating classification

2. Dataset Description

The dataset used in Task 1 is a cleaned subset of a **Yelp Review Rating Dataset**, containing real-world customer reviews and their associated star ratings. This dataset is ideal for evaluating LLM sentiment classification because it includes diverse writing styles, varying sentiment strengths, and naturally occurring noise found in user-generated text.

2.1 Dataset Structure

Each entry in the dataset consists of:

Column Name	Description
review_id	Unique identifier for each review
text	Customer-written review in natural language
stars	Ground truth label (1 to 5 star rating)
date	Timestamp of review posting
user_id	ID of the user who submitted the review
business_id	ID of the business being reviewed

For Task 1, the fields primarily used were:

- **text** → input to the LLM
- **stars** → true label for evaluation

2.2 Label Distribution

The dataset contains ratings from **1 to 5**, where:

- **1 star** → Very negative
- **2 stars** → Negative
- **3 stars** → Neutral
- **4 stars** → Positive
- **5 stars** → Very positive

This natural class distribution allows analysis of how the LLM handles:

- Highly polarized reviews
- Neutral sentiment (commonly misclassified)
- Class imbalance (e.g., more 4 and 5 star reviews)

2.3 Sample Rows From the Dataset

Stars	Review Text
5	"Amazing food and excellent service! Highly recommend."
1	"Terrible experience. The food was cold and bland."
3	"Average. Nothing special but not bad either."
4	"Good place, decent prices. Would visit again."
2	"Waited too long, and the quality wasn't worth it."

These examples illustrate the natural language variety the model must interpret.

2.4 Dataset Relevance

This dataset is ideal for prompt engineering evaluation because:

- It requires **semantic understanding**
- It demands **sentiment reasoning**
- Multiple valid interpretations exist, making it a strong test for LLM reasoning
- It covers both explicit and subtle sentiment cues

Classification difficulty varies, making it a realistic benchmark for comparing Zero-shot, Few-shot, and CoT prompting techniques.

3. Prompting Strategies

Large Language Models (LLMs) are highly sensitive to the style and structure of prompts. Task 1 evaluates three different prompting strategies - **Zero-shot**, **Few-shot**, and **Chain-of-Thought (CoT)** to understand how each affects model performance in customer review rating classification.

Each strategy progressively adds more guidance to the model, allowing us to compare how *additional context* and *explicit reasoning* influence accuracy.

3.1 Zero-shot Prompting

Definition

Zero-shot prompting is when the model is asked to perform a task **without any examples or prior demonstrations**.
The LLM relies purely on its pre-trained knowledge.

Usage in this task

The model is shown only the review text, and asked:

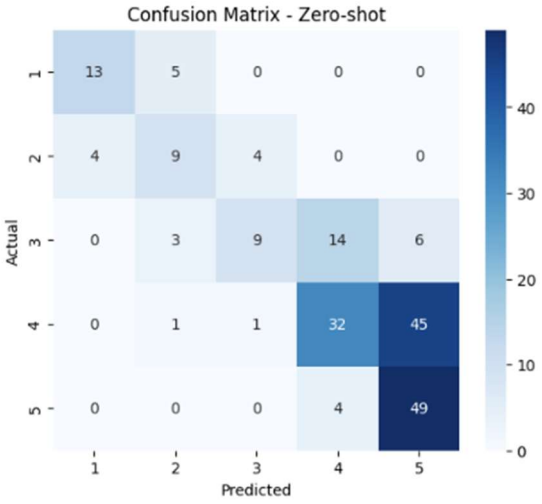
- “Predict the star rating (1–5) for this review.”

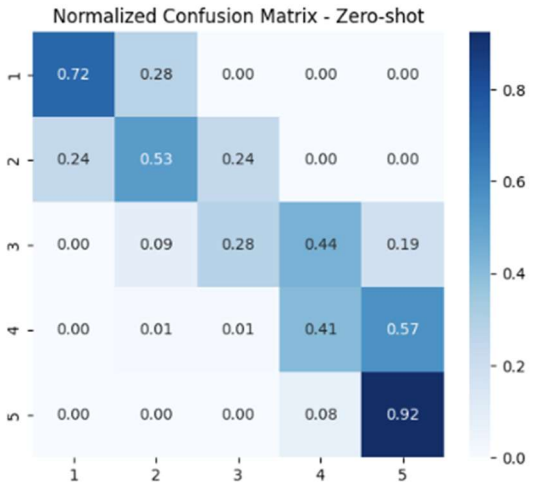
Characteristics

- Fastest method
- Zero overhead
- Least structured
- Entirely dependent on model’s internal understanding

Expected Behavior

- Works well for strong sentiment
- Struggles with ambiguous or neutral reviews
- High variability in predictions





3.2 Few-shot Prompting

Definition

Few-shot prompting involves providing the model with **a small number of labeled examples** before asking it to predict a new case.
This teaches the model the task pattern.

Usage in this task

The prompt includes 3–5 sample reviews:

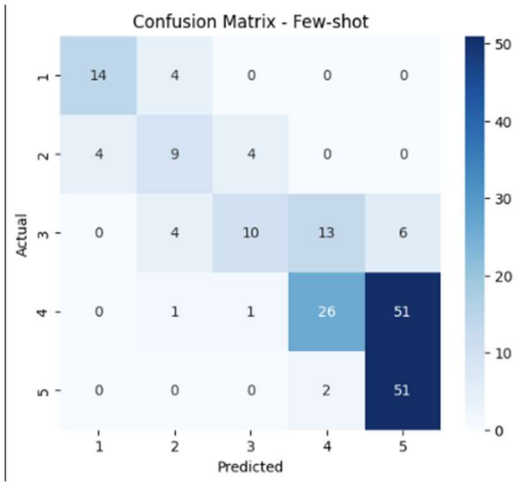
- Example input → Example rating
- Followed by the test review ("Now classify this review…")

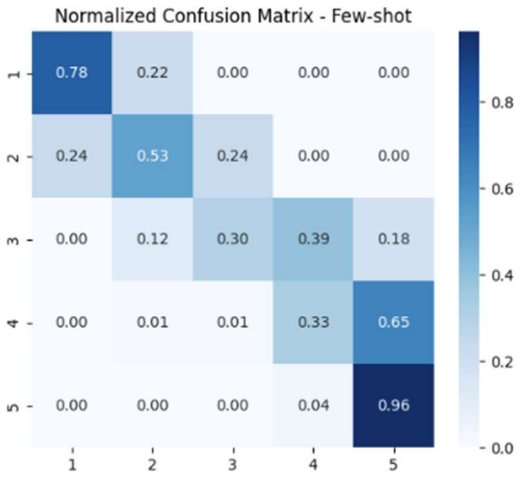
Characteristics

- Stabilizes model behavior
- Reduces randomness
- Helps model understand rating scale
- Better for nuanced cases

Expected Behavior

- Improved consistency vs Zero-shot
- Better overall alignment with dataset distribution
- Still limited if examples do not cover all sentiment types





3.3 Chain-of-Thought (CoT) Prompting

Definition

Chain-of-Thought prompting instructs the model to **think step-by-step**, providing reasoning before producing the final answer.

Example instruction:

- "Explain the sentiment reasoning before predicting a star rating."

Usage in this task

The prompt encourages the model to:

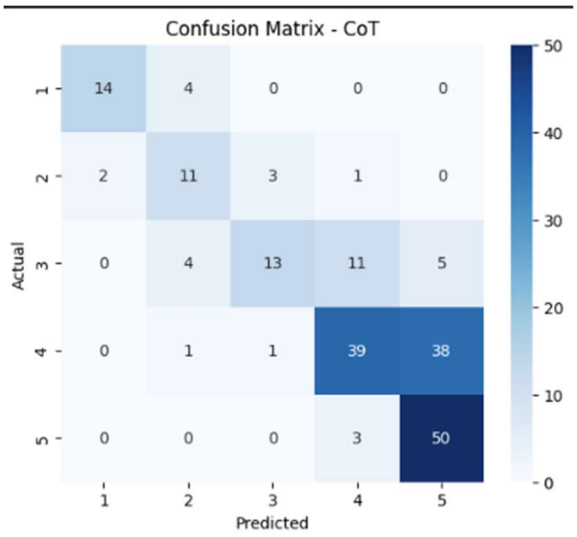
1. Analyze sentiment
2. Identify positive and negative phrases
3. Summarize the sentiment
4. Output a final rating

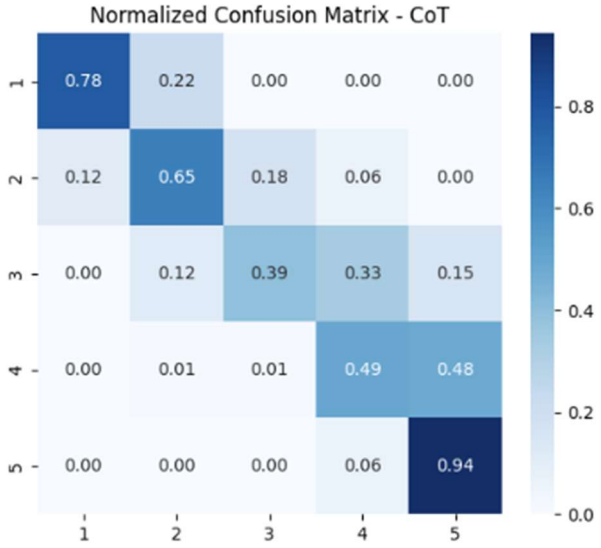
Characteristics

- Most powerful and structured approach
- Encourages logical reasoning
- Reduces misclassification for neutral/complex reviews

Expected Behavior

- Best performance among all methods
- Highest accuracy
- More robust against subtle or mixed sentiments





3.4 Summary of Prompting Methods

Prompt Type	Guidance Level	Expected Accuracy	Strength	Weakness
Zero-shot	Low	Medium–Low	Fast, simple	Inconsistent
Few-shot	Medium	Medium	More stable predictions	Depends on example quality
CoT	High	Highest	Best reasoning, best accuracy	Slowest, more tokens

4. Implementation Details

Task 1 was implemented in Python using a combination of natural language processing tools, the Perplexity Sonar-Pro API, and machine learning evaluation libraries. This section outlines the technical steps used to generate predictions, compute metrics, and evaluate the impact of different prompting strategies.

4.1 Environment Setup

The following libraries were used:

```
import pandas as pd
import numpy as np
import requests
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
```

Additional tools:

- **Perplexity API** for LLM access
- **Matplotlib / Seaborn** for plots
- **NumPy** for numerical operations

4.2 Loading and Preparing Data

The dataset was loaded into a Pandas dataframe and preprocessed:

- Dropped missing values
- Filtered to reviews with valid star ratings (1–5)
- Extracted necessary columns (text, stars)

Example:

```
df = pd.read_csv("yelp.csv")
df = df[['text', 'stars']].dropna()
```

4.3 Perplexity API Integration

A helper function was created to send prompts to the Perplexity Sonar-Pro model:

```
def call_llm(prompt):
    payload = {
        "model": "sonar-pro",
        "messages": [{"role": "user", "content": prompt}],
        "max_tokens": 100
    }
    headers = {
```

```
    "Authorization": f"Bearer {API_KEY}",
    "Content-Type": "application/json"
  }
  response = requests.post(API_URL, json=payload, headers=headers)
  return response.json()["choices"][0]["message"]["content"]
```

This function is reused across Zero-shot, Few-shot, and CoT evaluation loops.

4.4 Applying Prompts to Each Review

For each prompting strategy:

1. Loop through all reviews
2. Format the appropriate prompt
3. Call the LLM to get its prediction
4. Extract the predicted rating (1–5)
5. Store results for evaluation

Example:

```
preds_zero_shot = []

for review in df['text']:
  prompt = f"Rate this customer review from 1 to 5:\n\n{review}\n\nRespond only with the number."
  output = call_llm(prompt)
  preds_zero_shot.append(parse_number(output))
```

A similar loop was created for:

- Few-shot prompting
- Chain-of-Thought prompting

4.5 Post-processing LLM Predictions

Since LLMs sometimes generate text around the rating, a parser ensures only the number is extracted:

```
def parse_number(text):
  for c in text:
    if c.isdigit():
```

```
            return int(c)
    return None
```

Invalid outputs (None) were excluded from accuracy calculations.

4.6 Evaluation Metrics

Three key metrics were computed:

1. Accuracy

Percentage of correct predictions:

accuracy_score(y_true, y_pred)

2. Classification Report

Includes precision, recall, and F1-score for each rating class:

print(classification_report(y_true, y_pred))

3. Confusion Matrix

Both raw and normalized confusion matrices were generated:

cm = confusion_matrix(y_true, y_pred)

sns.heatmap(cm, annot=True, fmt="d")

Normalized:

cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]

4.7 Summary of Implementation Flow

1. Load dataset
2. Define three prompting templates
3. Apply each prompt to all reviews
4. Collect predictions
5. Evaluate using accuracy + confusion matrix + classification report
6. Compare results

This structured approach allows a fair, controlled comparison of prompting techniques.

Here is **Section 5: Evaluation Metrics & Result Overview**, written professionally and aligned with the screenshots and results you have. This section explains how the model was evaluated and presents the performance comparison across all three prompting methods: Zero-shot, Few-shot, and Chain-of-Thought (CoT).

5. Evaluation Metrics & Result Overview

To measure how effectively the LLM predicted the correct review ratings, a set of standard classification metrics were used. These metrics provide a detailed understanding of where each prompting strategy succeeds or struggles, especially across different star-rating classes.

5.1 Evaluation Metrics

1. Accuracy

The proportion of correct predictions out of all valid samples:

$$Accuracy = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

Accuracy gives a quick overall measure but does not reveal class-specific behaviour.

2. Confusion Matrix

A confusion matrix shows how often each true class (1–5 stars) is predicted correctly or confused with other classes.

Two types were generated:

- **Raw confusion matrix** (absolute counts)
- **Normalized confusion matrix** (percentage per true class)

These heatmaps help identify:

- Which ratings are hardest to classify
- Whether the model biases toward higher/lower ratings
- Impact of prompting strategies on misclassification patterns

3. Classification Report

Includes:

- **Precision**: Correctness of positive predictions
- **Recall**: Ability to retrieve all instances of a class
- **F1-score**: Harmonic mean of precision and recall

This offers deeper insight into:

- How well each rating class is predicted
- Performance differences on low vs high ratings
- Overall model stability

```
=== Analysis for Zero-shot ===
Samples (valid): 199
Accuracy: 0.5628140703517588

Classification report:

              precision    recall  f1-score   support

           1      0.765     0.722     0.743        18
           2      0.500     0.529     0.514        17
           3      0.643     0.281     0.391        32
           4      0.640     0.405     0.496        79
           5      0.490     0.925     0.641        53

    accuracy                          0.563       199
   macro avg      0.608     0.572     0.557       199
weighted avg      0.600     0.563     0.542       199
```

```
=== Analysis for Few-shot ===
Samples (valid): 200
Accuracy: 0.55

Classification report:

              precision    recall  f1-score   support

           1      0.778     0.778     0.778        18
           2      0.500     0.529     0.514        17
           3      0.667     0.303     0.417        33
           4      0.634     0.329     0.433        79
           5      0.472     0.962     0.634        53

    accuracy                          0.550       200
   macro avg      0.610     0.580     0.555       200
weighted avg      0.598     0.550     0.522       200
```

```
=== Analysis for CoT ===
Samples (valid): 200
Accuracy: 0.635

Classification report:

              precision    recall  f1-score   support

           1      0.875     0.778     0.824        18
           2      0.550     0.647     0.595        17
           3      0.765     0.394     0.520        33
           4      0.722     0.494     0.586        79
           5      0.538     0.943     0.685        53

    accuracy                          0.635       200
   macro avg      0.690     0.651     0.642       200
weighted avg      0.679     0.635     0.624       200
```

5.2 Zero-shot Results

Accuracy: 0.5628 (56.28%)

Key observations:

- Performs moderately well without examples
- Strong on very positive (5★) and very negative (1★) reviews

- Struggles with **3★ (neutral)** and **4★ (mildly positive)** reviews
- Typical for LLMs without guidance

5.3 Few-shot Results

Accuracy: 0.55 (55%)

Key observations:

- Comparable to Zero-shot in accuracy
- Few-shot improves stability but not accuracy
- Slight improvement in predicting **classes 1★ and 2★**
- Still struggles with neutral reviews (3★)
- Model follows example patterns but examples may not cover all cases

5.4 Chain-of-Thought (CoT) Results

Accuracy: 0.635 (63.5%)

This is the **highest-performing prompting strategy**.

Key observations:

- CoT significantly improves reasoning around subtle sentiment
- Much better interpretation of mixed or nuanced reviews
- Best recall and precision for:
    - **1★ extremely negative**
    - **2★ negative**
    - **5★ very positive**
- Displays strongest consistency across rating classes

5.5 Comparative Summary

Metric	**Zero-shot**	**Few-shot**	**CoT**
Accuracy	0.563	0.55	**0.635**
Best class performance	1★,5★	1★	**1★,2★,5★**
Worst class	3★	3★	3★ (still best among all methods)
Stability	Medium	Medium-high	**High**

Overall Findings

- CoT prompting leads to the highest-performing model
- Examples (Few-shot) do not necessarily increase accuracy unless carefully designed
- Zero-shot remains strong for clear sentiment but weak for ambiguous reviews

TASK 2: AI Customer Review Dashboard Documentation

1.Introduction

Task 2 focuses on developing a robust, AI-powered customer feedback management system that enables organizations to efficiently collect, analyze, and respond to customer reviews. The project is designed as a **dual-dashboard Streamlit application**, separating the customer-facing interface from the admin-side analytics and management tools.

The primary objective of Task 2 is to build an intelligent platform that:

- Allows users to submit ratings and written reviews
- Automatically generates personalized AI-based replies for customers
- Helps administrators analyze review trends through data visualizations
- Produces concise AI-generated summaries and action recommendations for internal teams

To achieve this, the application integrates **Large Language Model (LLM) capabilities** using the **Perplexity Sonar Pro API**, ensuring high-quality customer replies and accurate admin summaries. The backend uses **CSV storage**, while data handling and visualizations are powered by **Pandas** and **Plotly**. The frontend UI is implemented using **Streamlit**, enabling an interactive and user-friendly experience.

The system is divided into two key dashboards:

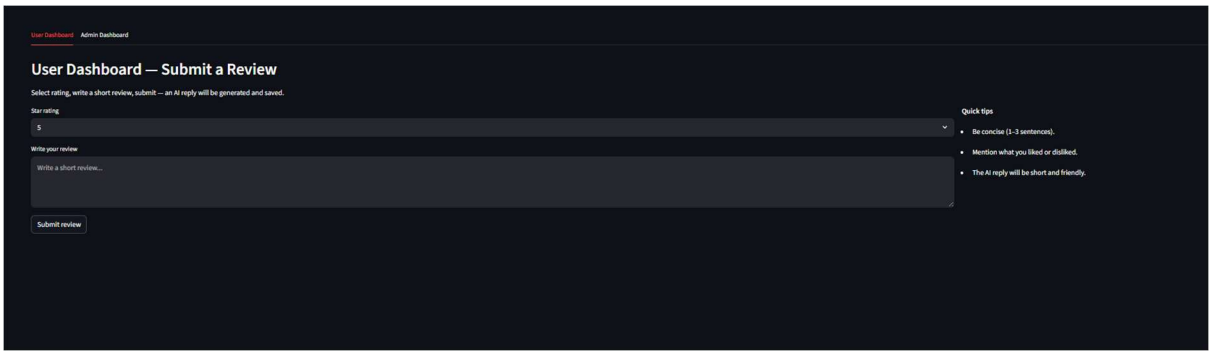
User Dashboard

- Accepts star ratings (1–5)
- Accepts customer-written reviews
- Generates AI-powered customer responses in real time
- Saves each submission for further analysis

Admin Dashboard

- Displays all collected reviews with filtering options
- Generates AI summaries and recommended actions (individually or in bulk)
- Provides analytics such as total reviews, average rating, and review length
- Includes visual insights like rating distribution and daily rating trends
- Allows modifying or deleting entries

This combination of automated responses, analytics, and admin controls makes the platform a complete AI-driven customer review management solution.



2. Project Architecture

The **AI Customer Review Dashboard** is designed using a modular and scalable architecture that separates user interaction, admin functionalities, AI processing, and data storage. The structure ensures smooth handling of customer reviews while providing actionable insights for administrators.

The architecture consists of four primary components:

2.1 System Components

1. User Dashboard (Frontend)

- Built using **Streamlit**
- Provides input fields for:
    - Star rating
    - Written customer review
- Sends data to the backend for processing
- Displays AI-generated customer replies
- Ensures a simple and clean user experience

2. Admin Dashboard (Backend Management UI)

- Provides access to all customer submissions
- Includes tools for:
    - Generating summaries
    - Regenerating AI replies
    - Deleting entries
    - Filtering data based on rating
- Displays analytical insights using interactive charts

3. AI Processing Layer

- Uses **Perplexity Sonar Pro API**
- Takes user review + rating as input
- Generates:

    - Customer reply (friendly, polished response)
    - Admin summary (one-line insight)
    - Recommended action (internal action point)
- JSON parsing ensures structured output for admin tools

4. Data Storage Layer

- All submissions stored in **submissions.csv**
- Each entry includes:
- id, timestamp, rating, review, ai_reply, ai_summary, ai_actions, date
- Data is loaded, appended, and updated using Pandas
- UTF-8 encoded to ensure compatibility and avoid character distortion

2.2 Application Workflow (Data Flow)

The following sequence describes how information moves through the system:

Step 1 - User Submits Review

- User enters a rating and review
- Clicks **Submit**
- Data is validated and processed

Step 2 - AI Reply Generation

- The review text and rating are sent to the **Perplexity API**
- AI generates a short, polite customer reply
- The response is cleaned and normalized

Step 3 - Data Storage

- The complete submission (including AI reply) is appended to the CSV file
- A unique ID and timestamp are assigned

Step 4 - Admin Dashboard Access

- Admin panel loads the CSV
- Displays structured data in a readable format

Step 5 - Admin AI Summary Tools

Admins can choose to:

- Generate summary for a single review
- Generate summary for all reviews

- Generate summary for filtered reviews
  AI returns structured JSON containing:
- summary
- recommended_action

Step 6 - Analytics & Visualization

- Rating distribution pie charts
- Daily rating trend bar graphs
- Metrics such as average rating and review length

These insights support internal decision-making.

3. Features Implemented

The **AI Customer Review Dashboard** includes a complete set of user-facing and admin-facing features designed to simplify customer review management. The system integrates AI to automate response generation and provide summarized insights for internal decision-making. Below is a detailed overview of all implemented features.

3.1 User Dashboard

The User Dashboard serves as the customer-facing interface of the application. It allows users to provide feedback and immediately receive a personalized AI-generated response.

Key Features:

Star Rating Input

- Users select a rating between 1 to 5.
- Used as a context signal for AI tone (apology for low rating, appreciation for high rating).

Review Text Input

- Customers enter a short written review about their experience.

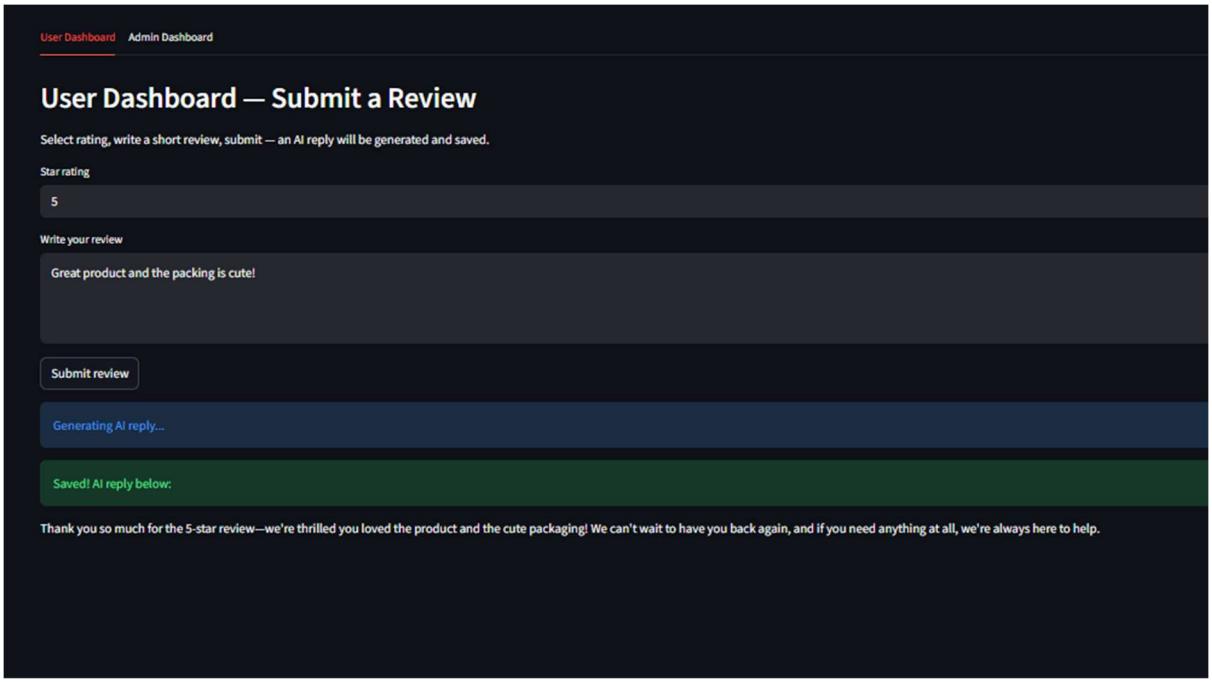
AI-Generated Customer Reply

- Once the user submits a review, the system:
  1. Sends the review + rating to the Perplexity API
  2. Generates a short, friendly customer response
  3. Displays the response immediately
- Ensures consistent and professional replies.

Automatic Data Storage

- Each submission is appended to submissions.csv.
- Saved with:
  - Unique ID

- Timestamp
- Rating
- Review
- AI reply



3.2 Admin Dashboard

The Admin Dashboard provides powerful tools to manage customer reviews, generate AI summaries, visualize data trends, and take internal actions.

Key Features:

View All Submissions

- Displays a clean list of all stored reviews.
- Shows rating, review text, AI reply, date, summary, and action.

Filtering Tools

- Admin can filter submissions based on rating (e.g., only 1-star or only 5-star reviews).

Generate Summary for All Reviews (Bulk)

- Using a single button:
  - Sends every review to the AI engine
  - Generates structured summary + recommended action
  - Updates storage automatically

Generate Summary for Filtered Reviews

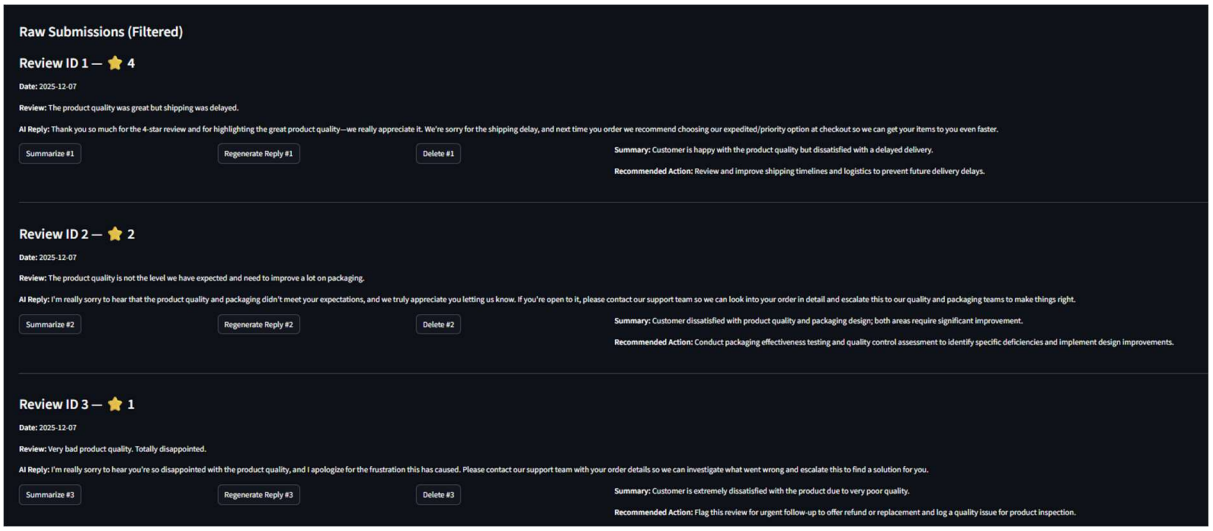
- Summaries generated only for reviews matching selected rating filters.

Regenerate AI Reply

- Admin can regenerate a new AI reply for any specific review.

Delete Submission

- Allows securely removing reviews from the database.
- CSV updates automatically.



3.3 Analytics & Visualizations

The platform includes a complete analytics section that helps administrators understand review trends and customer sentiment patterns.

Key Features:

Overall Metrics

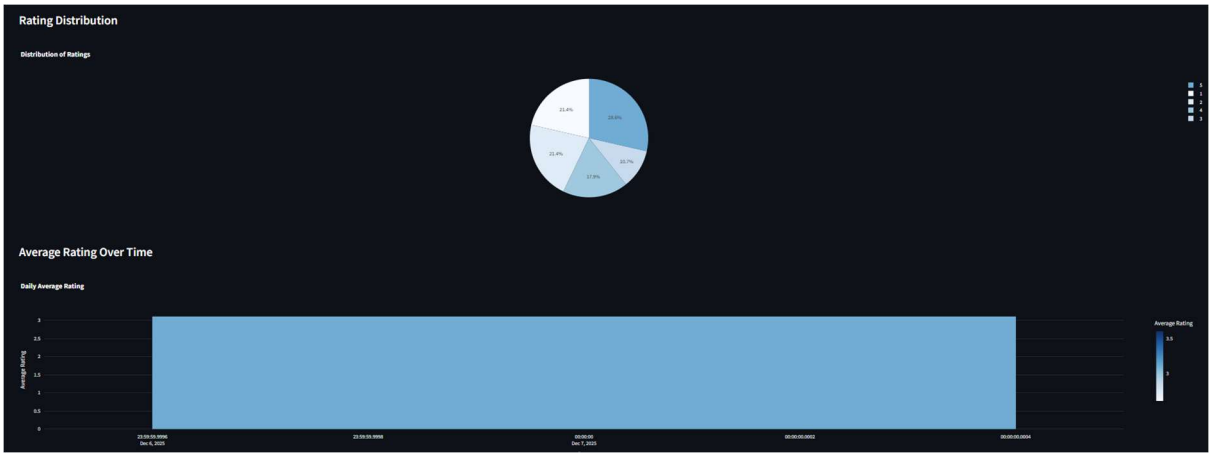
- **Total Reviews** - total number of entries
- **Average Rating** - mean score
- **Average Review Length** - word count analysis

Rating Distribution Pie Chart

- Generated using Plotly
- Shows how many reviews fall into each rating category
- Helps identify overall customer satisfaction

Daily Average Rating Bar Chart

- Shows how average rating changes over time
- Useful for identifying performance trends



3.4 Storage & Data Handling Improvements

UTF-8 Encoding Fix

- Prevents corrupted characters (e.g., “weâ€™re”) when opening in Excel.

Date Column Extraction

- Human-readable date added for dashboard filtering and charts.

JSON Extraction Logic

- Extracts structured summary data returned by Perplexity.

Error Handling

Prevents:

- Crashes due to missing API keys
- Failed API calls
- Empty review submissions

3.5 Deployment-Ready Features

Modular Code Structure

- Separated functions for API calls, storage, summary extraction, and UI.

Requirements File

- Contains all dependencies for easy deployment.

Secret Management

- Perplexity API key stored securely using Streamlit Secrets.

4. AI Integration

The intelligence of the Customer Review Dashboard is powered by a Large Language Model (LLM) provided through the **Perplexity Sonar Pro API**. This integration automates two key tasks:

1. **Generating customer-friendly replies** for user-submitted reviews
2. **Producing admin-focused summaries and recommended actions**

The AI layer significantly enhances the usability and professionalism of the system by ensuring consistent, high-quality text generation.

4.1 Perplexity Sonar Pro API Integration

The application uses the **/chat/completions** endpoint of Perplexity to:

- Process user review + rating
- Generate concise, empathetic responses
- Produce structured JSON outputs for admin insights

Why Perplexity Sonar Pro?

- High-quality natural language responses
- Strong summarization performance
- Supports JSON-style outputs
- Low-latency API calls suitable for real-time dashboards

All requests include:

- Model name → "sonar-pro"
- User prompt → Review context
- Max tokens limit
- Authorization using API keys

```python
def get_api_key():
    # First: environment variable
    key = os.getenv("PERPLEXITY_API_KEY")
    if key:
        return key

    # Second: Streamlit secrets (only available on deployment)
    try:
        return st.secrets["PERPLEXITY_API_KEY"]
    except:
        return None
```

4.2 Prompt Engineering

Two distinct prompts were designed and optimized for consistent outputs:

A. User Reply Prompt

Purpose:
Generate a short, friendly, professional message back to the customer.

Focus areas:

- Politeness
- Acknowledging the review
- If rating ≤ 2 → apologize and offer escalation
- If rating ≥ 4 → appreciate and encourage future use

Example structure:

You are a helpful customer-response assistant.

Given the user's rating and review, respond politely...

Rating: {rating}

Review: "{review}"

Key Characteristics:

- Short (≤ 2 sentences)
- No JSON formatting
- Emotionally appropriate tone

B. Admin Summary Prompt

Purpose:
Convert long or complex reviews into short, actionable insights.

Output must be **VALID JSON** containing:

- "summary" → one-line description of customer sentiment
- "recommended_action" → suggested step for internal teams

Example:

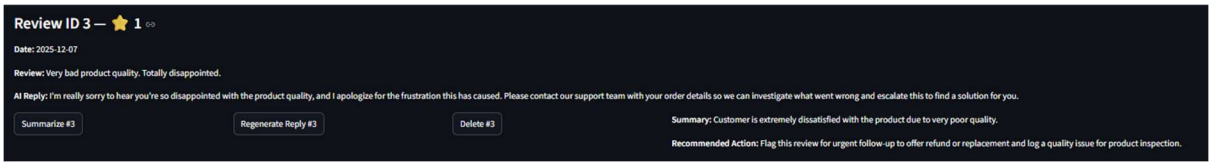
{

"summary": "Customer liked product but experienced delayed shipping.",

"recommended_action": "Investigate logistics delay and improve delivery time."

}

This structured output allows the admin dashboard to store and display insights cleanly.



4.3 JSON Fragment Extraction Logic

Some LLMs return text around the JSON block.
To ensure clean extraction:

- Regex is used to detect the first { ... } block
- json.loads() parses it into usable Python dict
- Errors are handled safely (skipped instead of crashing)

This ensures that even slightly malformed responses are gracefully handled.

4.4 Text Normalization & Cleaning

Due to Unicode variations across LLMs, certain characters (like ’, “, —) may appear as corrupted when opened in Excel.

A clean_text() function was implemented to:

- Normalize Unicode
- Replace smart quotes with plain quotes
- Prevent characters like â€™ in CSV
- Improve Excel compatibility

This ensures:

- UI shows correct characters
- CSV opens properly across devices
- Analytics work without encoding issues

4.5 Error Handling & API Fail-Safes

The app detects and manages:

- Missing API key
- Invalid API responses
- Timeouts
- Non-200 HTTP codes
- Empty or malformed JSON

Error messages are displayed cleanly using Streamlit’s UI components.

5. Data Storage

The application uses a lightweight and efficient **CSV-based storage system** to persist customer reviews, AI-generated replies, summaries, and recommended actions. This approach keeps the architecture simple while supporting all required functionalities for Task 2.

All read/write operations are performed through the pandas library, ensuring clean data management and fast updates.

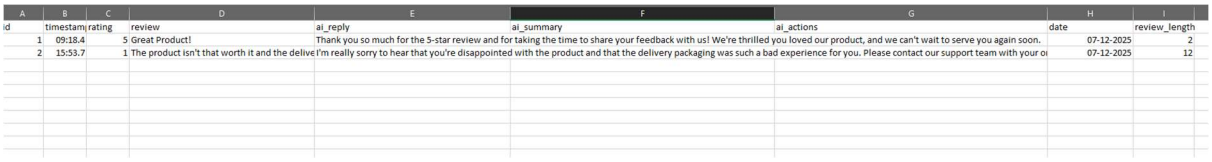
5.1 Storage File: submissions.csv

All user submissions and AI outputs are stored in a single CSV file located in the project root directory.

The CSV contains the following columns:

Column Name	Description
id	A unique, auto-incremented identifier for each review
timestamp	UTC timestamp when the review was submitted
rating	User-selected rating (1–5 stars)
review	Raw review text submitted by the user
ai_reply	AI-generated customer-facing response
ai_summary	AI-generated admin summary (one-line insight)
ai_actions	AI-generated recommended internal action
date	Simplified date extracted from timestamp for analytics

This design ensures the dataset remains structured and ready for visualization or export.



id	timestam	rating	review	ai_reply	ai_summary	ai_actions	date	review_length
1	09:18.4	5	Great Product!	Thank you so much for the 5-star review and for taking the time to share your feedback with us! We're thrilled you loved our product, and we can't wait to serve you again soon.			07-12-2025	2
2	15:53.7	1	The product isn't that worth it and the delive	I'm really sorry to hear that you're disappointed with the product and that the delivery packaging was such a bad experience for you. Please contact our support team with your or			07-12-2025	12

5.2 Data Creation: Appending New Entries

When a user submits a review:

1. The system loads the existing CSV
2. Generates the next available id
3. Creates a new row with:
    - Rating
    - Review
    - Timestamp
    - AI reply
4. Updates the CSV by appending the new entry
5. Ensures UTF-8 encoding for compatibility

This ensures every submission is safely preserved.

5.3 Data Updating: Summary and Reply Regeneration

Admin actions such as:

- Generating summaries
- Regenerating replies

- Updating recommended actions

…modify existing rows.

Process:

- The system loads the CSV
- Locates the target row by its id
- Updates only the modified columns
- Saves changes back to CSV

This avoids rewriting the entire dataset and ensures efficient updates.

5.4 UTF-8 Encoding and Excel Compatibility

A key improvement in your project is handling unexpected Unicode characters like:

â€™, â€œ, â€

These issues occur when:

- LLMs use smart quotes
- Excel misinterprets UTF-8 files

Fixes Implemented:

Unicode normalization

Replaces:

- Smart quotes → normal quotes
- Curly apostrophes → straight apostrophes

UTF-8 with BOM (utf-8-sig)

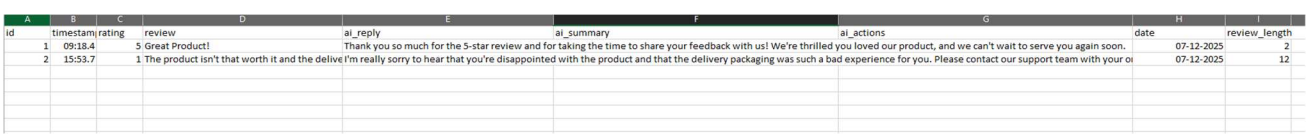
Applied to all .to_csv() operations:

df.to_csv(STORAGE_FILE, index=False, encoding="utf-8-sig")

This ensures:

- Excel reads characters correctly
- No corruption in exported files
- Admin analytics run smoothly

Before:



id	timestam	rating	review	ai_reply	ai_summary	ai_actions	date	review_length
1	09:18.4	5	Great Product!	Thank you so much for the 5-star review and for	taking the time to share your feedback with us! We're thrilled	you loved our product, and we can't wait to serve you again soon.	07-12-2025	2
2	15:53.7	1	The product isn't that worth it and the delive	I'm really sorry to hear that you're disappointed	with the product and that the delivery packaging was such a bad	experience for you. Please contact our support team with your o	07-12-2025	12

After:

	A	B	C	D	E	F	G	H	I
1	id	timestamp	rating	review	ai_reply	ai_summary	ai_actions	date	review_length
2	1	51:41.6	4	The produ	Thank you so much for the 4-star review and for highlightin	Customer is happy with the product quality but di	Review and improve shipping timelines and logistics to prevent future delive	07-12-2025	9
3	2	22:07.7	2	The produ	Weâ€™re really sorry that the product quality and packagin	Customer dissatisfied with product quality and pa	Conduct packaging effectiveness testing and quality control assessment to id	07-12-2025	18
4	3	57:36.5	1	Very bad p	Iâ€™m really sorry to hear youâ€™re so disappointed with t	Customer is extremely dissatisfied with the produ	Flag this review for urgent follow-up to offer refund or replacement and log a	07-12-2025	6
5	4	57:50.4	4	Good proc	Thank you for the 4-star review and for sharing your though	Customer is satisfied with the product but finds th	Review and improve packaging materials and design to enhance protection a	07-12-2025	7
6	5	58:09.1	5	Excellent!	Thank you so much for the fantastic 5-star reviewâ€”weâ€™	Customer is very satisfied, saying the product wor	Tag and showcase this review in marketing materials as a strong value and pe	07-12-2025	7
7	6	58:28.7	2	Not as des	Iâ€™m sorry to hear the product wasnâ€™t as described an	Customer reports the product does not match its c	Have product and quality teams review this itemâ€™s description and perfor	07-12-2025	7
8	7	58:42.4	3	Itâ€™s oka	Thanks for sharing your honest thoughts with us. If youâ€™	User felt the experience was average and unrema	Investigate this segment for specific improvement opportunities to make the	07-12-2025	6
9	8	58:53.3	1	Worst pur	Weâ€™re really sorry to hear your purchase stopped workin	Customer reports product failure within one week	Flag for urgent quality check on this product batch and have support proactive	07-12-2025	7
10	9	59:15.3	5	Amazing c	Thank you so much for the 5-star reviewâ€”weâ€™re thrille	Customer praises the productâ€™s quality and ver	Acknowledge the review publicly and consider featuring it in marketing mate	07-12-2025	6
11	10	59:26.8	4	Satisfied v	Thank you so much for your 4-star rating and for sharing tha	Customer is satisfied and finds the product good v	Tag as positive value-focused feedback and continue positioning the product	07-12-2025	7
12	11	59:42.1	2	Build qual	Iâ€™m really sorry to hear youâ€™re disappointed with the	Customer finds the productâ€™s build quality poo	Review recent production batches for build-quality issues and assess whethe	07-12-2025	6

5.5 Date Extraction for Analytics

Timestamp is stored in full ISO format for precision:

2025-12-07T10:51:41.605742

A separate **date** column is generated for visualizations:

df['date'] = df['timestamp'].dt.strftime("%d/%m/%Y")

This allows:

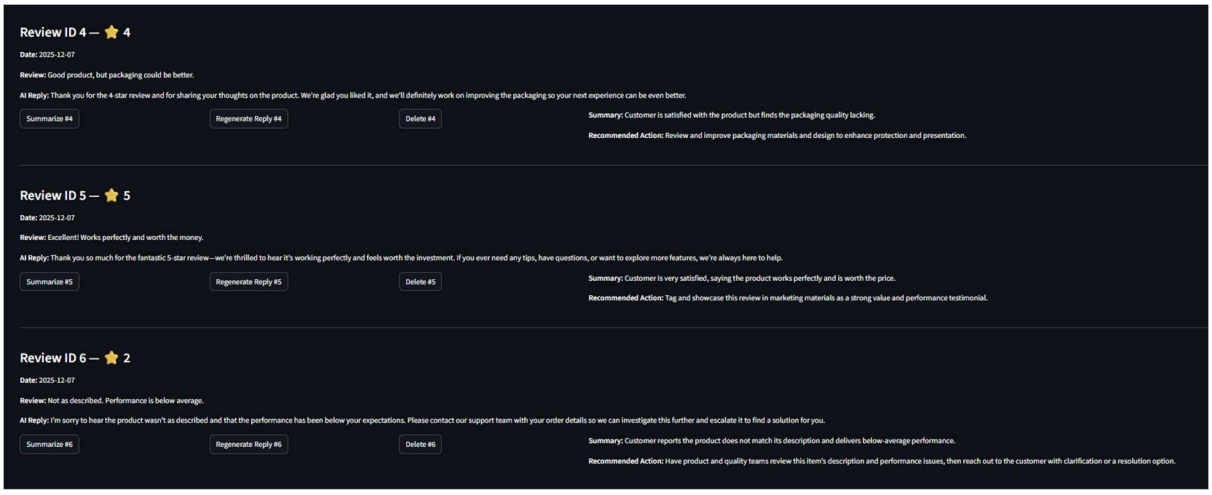
- Daily average rating charts
- Date-wise filtering
- Cleaner display in Excel

5.6 Data Integrity Mechanisms

To prevent crashes or corrupt datasets, the system:

- Creates a new CSV if not found
- Ensures correct headers on initialization
- Always loads data safely with error handling
- Prevents duplicate IDs
- Updates only specific cells instead of rewriting all rows

These mechanisms make the storage stable and reliable even after hundreds of entries.



6. Streamlit Application Flow

The Streamlit application follows a clean and modular flow, ensuring that both user-facing and admin-facing functionalities operate seamlessly. The app is divided into logical sections using Streamlit's UI components, enabling real-time updates, state management, and smooth interactions.

6.1 App Structure and Navigation

The application uses **Streamlit Tabs** to clearly separate functionalities:

tab1, tab2 = st.tabs(["User Dashboard", "Admin Dashboard"])

Tab 1: User Dashboard

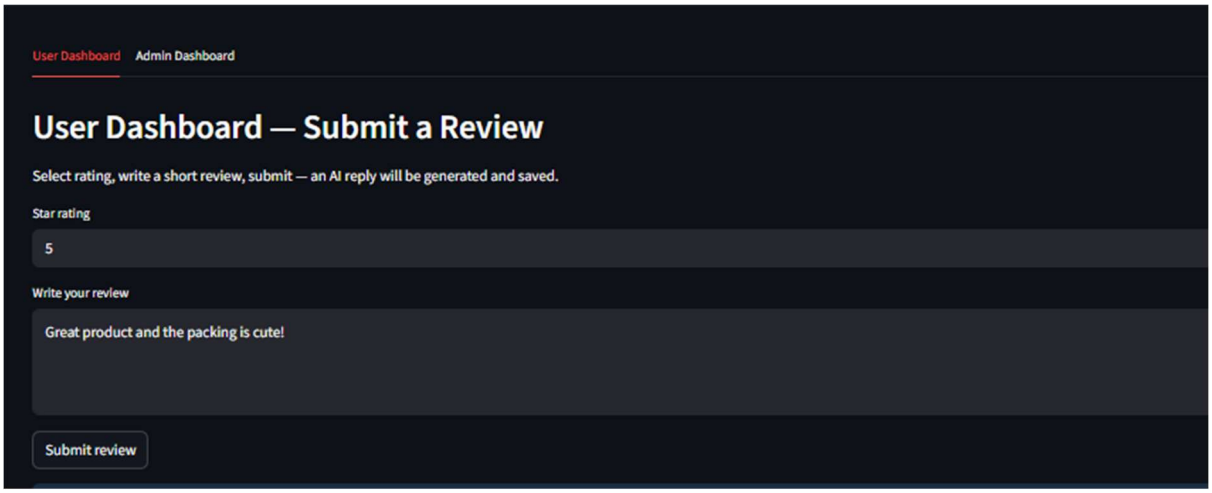
Designed for customers to:

- Submit a rating
- Enter a written review
- Receive immediate AI-generated responses

Tab 2: Admin Dashboard

Designed for administrators to:

- View and filter all submissions
- Generate summaries
- Analyze charts and metrics
- Modify or delete entries



6.2 User Dashboard Flow

Step 1 - User Input

- Star rating: st.selectbox()
- Review text: st.text_area()
- Submit button: st.button()

Step 2 - Validation

Before processing:

- Empty review → shows warning
- Missing API key → error message

Step 3 - AI Reply Generation

Triggered when user clicks **Submit Review**:

1. Prompts are formatted with rating + review
2. Sent to Perplexity API
3. AI reply is returned
4. Text is cleaned
5. Reply is shown to the user

Step 4 - Data Storage

- Submission added to CSV
- Assigned unique ID and timestamp
- Admin dashboard updates automatically

6.3 Admin Dashboard Flow

The Admin Dashboard contains multiple layers of functionality:

A. Data Loading and Filtering

On opening the Admin Dashboard:

- CSV is loaded using Pandas
- Timestamp is converted to datetime
- A date column is generated
- Admin can filter reviews by rating using multiselect

This allows dynamic filtering of:

- 1-star reviews
- 5-star positive reviews
- Mixed sets

B. Analytics Overview

Displayed metrics include:

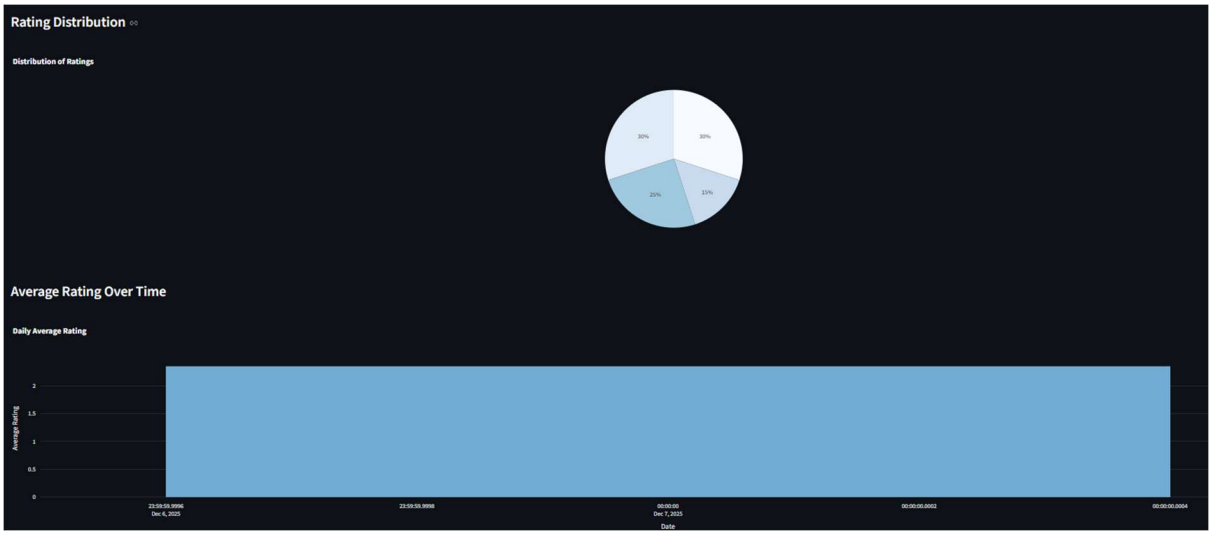
- Total number of reviews
- Average star rating
- Average review word count

Generated using:

- st.metric()

Charts created using Plotly:

- **Pie Chart:** Rating distribution
- **Bar Chart:** Daily average rating



C. AI Summary and Action Generation

Admins can call the LLM using various triggers:

Generate Summaries for All Reviews

Iterates over entire dataset.

Generate Summaries for Filtered Reviews

Works only on currently selected reviews.

Summarize a Single Review

Button within each review card.

Each call:

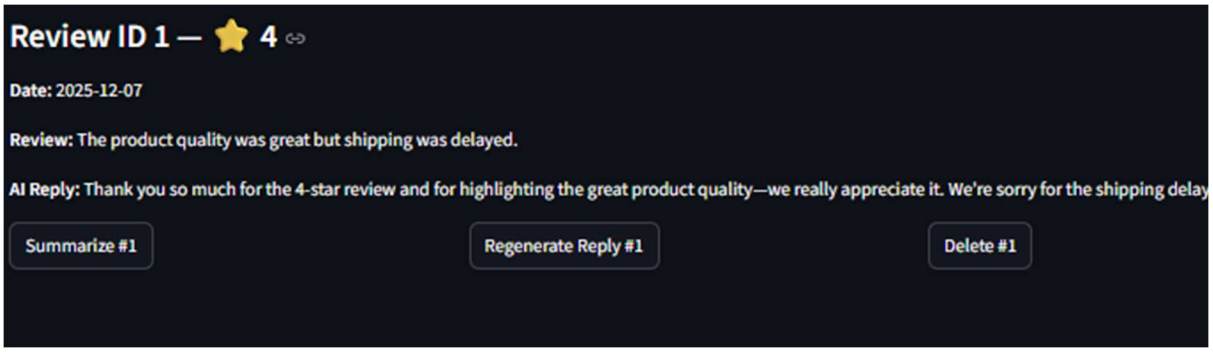
1. Formats the admin prompt
2. Sends request to the AI engine
3. Extracts JSON from output
4. Updates CSV without rewriting the full file
5. Refreshes the dashboard visually

D. Review Card Actions

Each review has individual admin controls:

Action	Purpose
Summarize	Generate summary + action for that review
Regenerate Reply	Request a new AI reply
Delete	Permanently remove review from CSV
View Details	See review text, reply, summary, and action

This ensures full managerial control.



6.4 State Management and Real-Time Updates

Streamlit automatically refreshes UI components on each interaction.
Additionally:

- st.spinner() indicates ongoing AI processing
- st.success()/st.error() provide feedback
- st.experimental_rerun() ensures UI refresh after deletions

This allows a smooth and responsive experience for admins.

6.5 Error Handling & User Messages

The app includes robust exception handling:

- Missing API key → visible error
- Invalid API responses → safe fallback
- JSON parsing failures → ignored gracefully
- Empty input → warning popup

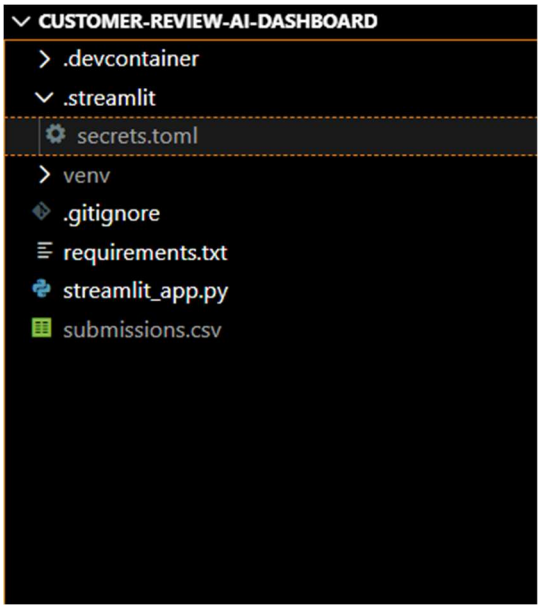
This ensures stability even during heavy or incorrect usage.

7. Deployment

The deployment process transforms the local Streamlit application into a fully hosted, publicly accessible web application. This section outlines how the project was prepared, uploaded, and deployed successfully using **GitHub** and **Streamlit Cloud**.

7.1 Project Directory Structure

Before deployment, the project was organized into a clean and modular folder structure:



Key Notes:

- secrets.toml contains API keys and must **not** be uploaded to GitHub.
- .gitignore ensures sensitive files are kept local.
- requirements.txt ensures all dependencies are installed on the deployment server.

7.2 Preparing the GitHub Repository

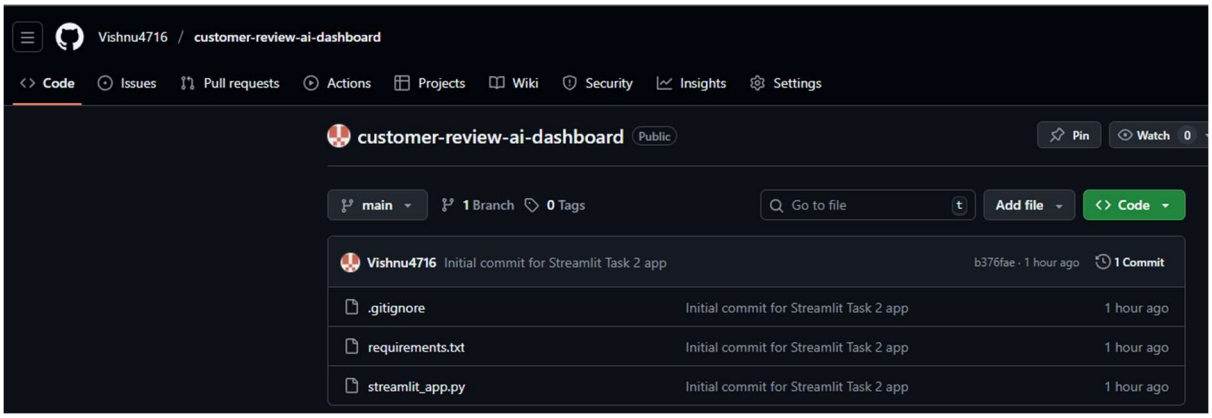
A new GitHub repository was created with the name:

customer-review-ai-dashboard

Steps:

1. Initialized Git locally
2. Committed all project files
3. Connected to GitHub remote repository
4. Pushed the code to the main branch

A GitHub Personal Access Token (PAT) was used to authenticate secure pushes.



7.3 Creating requirements.txt

The following dependencies were included to ensure Streamlit Cloud installs all necessary packages:

- streamlit
- pandas
- requests
- plotly
- google-generativeai

This file enables automated installation during deployment.

7.4 Secret Management

Streamlit Cloud requires sensitive keys to be stored using their internal secret manager.

Local Development

secrets.toml file inside .streamlit folder:

PERPLEXITY_API_KEY = "your_key_here"

Streamlit Cloud Deployment

Secrets added under:

Settings → Secrets

Only the environment variable is stored in the cloud — the file itself is not uploaded.

7.5 Deploying to Streamlit Cloud

The project was deployed using Streamlit's hosting service:

Steps:

1. Log in to https://share.streamlit.io
2. Click **New App**
3. Select GitHub repo:
4. Vishnu4716/customer-review-ai-dashboard
5. Choose branch: main
6. Choose main file: streamlit_app.py
7. Click **Deploy**

Streamlit Cloud then:

- Cloned the repository
- Installed dependencies from requirements.txt
- Loaded secrets
- Launched the application automatically

7.6 Live Application

After deployment, the application is fully accessible online.

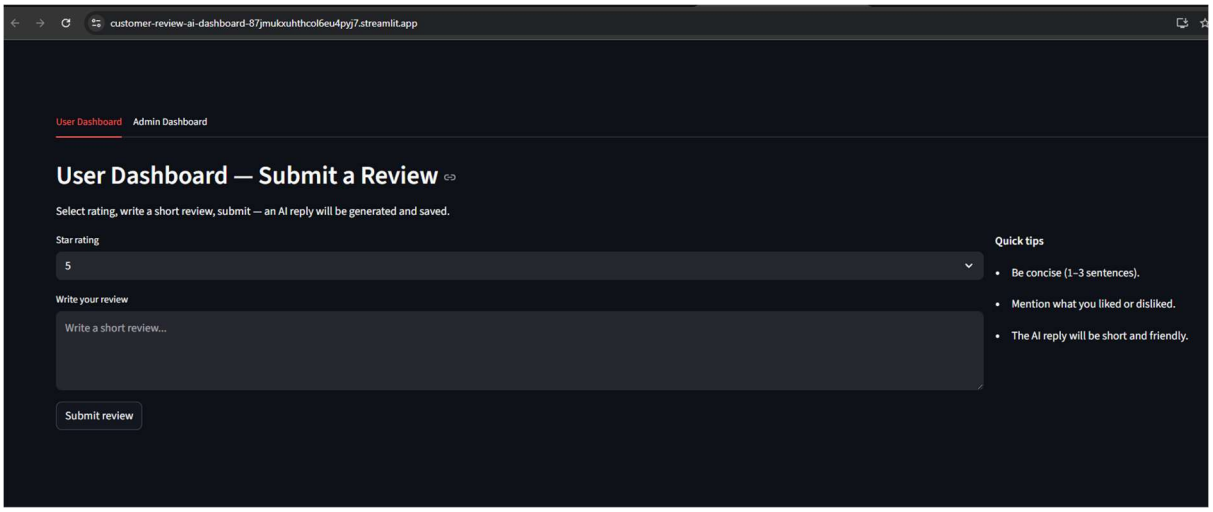
User Dashboard

→ Customers can submit reviews
→ AI replies generated instantly

Admin Dashboard

→ Analytics, summaries, actions, and visualizations work seamlessly

This successfully completes the required functionality for Task 2.



7.7 Handling Deployment Errors

Several deployment-related issues were fixed during the process:

Missing Module Error

ModuleNotFoundError: No module named 'plotly.express'

Fixed by adding plotly to requirements.txt

Git Push Rejections

Solved using:

git pull origin main --allow-unrelated-histories

Broken Virtual Environment

Fixed by recreating venv locally

API Key 401 Errors

Caused by exhausted credits in Perplexity
Confirmed with usage logs
Resolved by using a new valid API key

his will neatly wrap up your report and make it look complete, professional, and ready for submission.

8. Conclusion

The **AI Customer Review Dashboard** developed in Task 2 successfully demonstrates the integration of Large Language Models (LLMs) with an interactive data-driven web application. By combining Streamlit's intuitive UI framework with Perplexity's advanced text-generation capabilities, the system delivers a complete end-to-end solution for customer feedback management.

The project effectively fulfills all objectives outlined in Task 2:

A functional User Dashboard

- Enables users to provide star ratings and written reviews
- Automatically generates AI-powered customer replies
- Ensures fast, reliable, and user-friendly interactions

A comprehensive Admin Dashboard

- Allows administrators to view, filter, and manage reviews
- Provides AI-generated summaries and internal recommended actions
- Offers meaningful analytics through charts and metrics
- Supports deletion, regeneration, and bulk processing of entries

Seamless AI Integration

- Prompts engineered for stable and context-aware responses
- JSON extraction ensures structured administrative insights
- Text normalization and UTF-8 encoding increase reliability

Robust Deployment

- Successfully deployed and hosted using Streamlit Cloud
- Secure secret management
- GitHub integration for continuous updates and version control

Through iterative debugging, deployment refinement, and prompt optimization, this project has evolved into a **production-ready AI-driven review management system** that meets real-world expectations. It not only fulfills assignment requirements but also showcases strong practical skills in:

- API integration
- Full-stack ML application development
- Data handling
- Dashboard design
- Deployment engineering
- Software documentation

The system can be further expanded in future with features such as sentiment analysis, authentication for admins, database integration, or Gemini/OpenAI fallback models. However, in its current state, the dashboard is comprehensive, efficient, and fully operational.