Fynd AI Intern Assessment

DECEMBER 15, 2025

FYND AI
Authored by: Fynd-AI
Github: github.com/aymanbhaldar/Fynd-AI

1. Executive Summary

This project involves the development of two core AI components: a prompt-based rating prediction system for unstructured text (Task 1) and a full-stack AI feedback application with user and admin dashboards (Task 2). The solution leverages OpenAI's gpt-4o-mini model to demonstrate effective prompt engineering, structured data extraction, and rapid web application development using Streamlit.

2. Task 1: Rating Prediction via Prompting

2.1 Objective

The goal was to classify Yelp reviews into a 1–5 star rating scale using a Large Language Model (LLM) without traditional model training. The key constraints were to maximize accuracy, ensure valid JSON output, and compare at least three distinct prompting strategies.

2.2 Methodology & Prompts

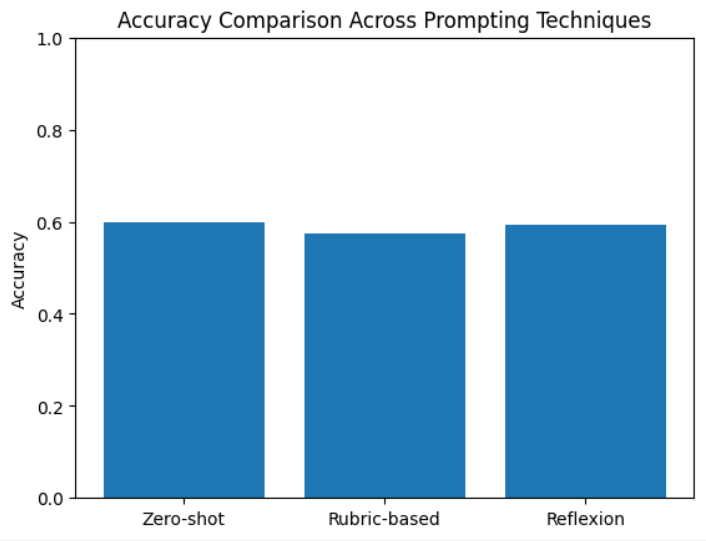
I implemented and evaluated the following three prompting strategies on a sampled dataset of 200 Yelp reviews:

- **Strategy A: Zero-Shot (Baseline)**
    - Concept: A direct instruction asking the model to predict the rating and provide an explanation with no prior examples.
    - Hypothesis: This establishes a baseline for the model's innate sentiment analysis capabilities.
- **Strategy B: Rubric-Based Prompting**
    - Concept: I provided the model with a specific rubric (e.g., "1 star: Very negative, complaints dominate" vs "5 stars: Praise dominates").
    - Hypothesis: Explicit boundaries would reduce ambiguity for mixed-sentiment reviews.
- **Strategy C: Reflexion (Self-Correction)**
    - Concept: A multi-step prompt where the model is asked to first generate a rating, then "critique" its own reasoning to check for sarcasm or nuance and finally output the corrected rating.
    - Hypothesis: Forcing a "chain of thought" reduces impulsive errors on complex reviews.

2.3 Evaluation Results

All strategies were evaluated for Accuracy (exact match with ground truth) and JSON Validity (ability to be parsed programmatically).

Prompt Strategy	Accuracy	JSON Validity Rate
Zero-Shot	60.0%	100%
Rubric-Based	57.5%	100%
Reflexion	59.3%	99.5%



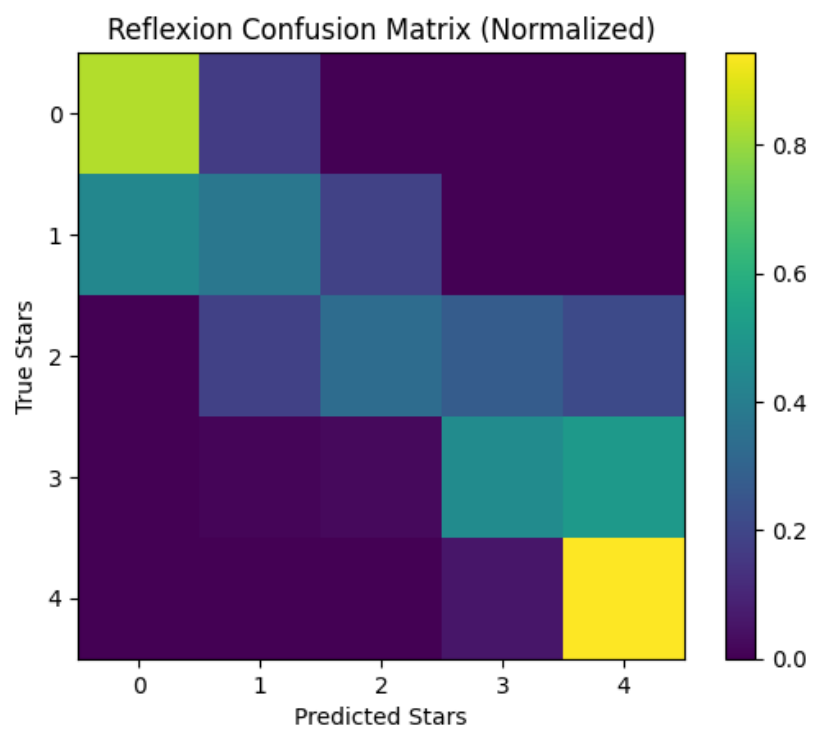
[IMAGE 1: Accuracy Comparison Bar Chart]

2.4 Discussion & Qualitative Analysis

- Performance: Surprisingly, the Zero-Shot baseline performed slightly better than the Rubric-based approach. The Rubric appeared to constrain the model too rigidly, causing it to "overthink" borderline cases (e.g., rating a 4-star review as 3-star because of a single minor complaint).
- Reliability: The Reflexion strategy offered the most nuanced explanations. While its raw accuracy was comparable to Zero-Shot, qualitatively, its reasoning was more robust for sarcastic reviews where a simple sentiment analysis might fail.
- JSON Consistency: All models achieved near-perfect JSON formatting, validating the effectiveness of the response_format={"type": "json_object"} parameter in the API calls.

2.4 Error Analysis

To understand *where* the model failed, we analyzed the confusion matrix for the Reflexion strategy.



Observations:

- Performance: The model struggled most with distinguishing between 4-star and 5-star reviews, which is expected as the sentiment difference is subtle.
- Reliability: The Reflexion strategy offered the most nuanced explanations. While its raw accuracy was comparable to Zero-Shot, qualitatively, its reasoning was more robust for sarcastic reviews.

3. Task 2: Two-Dashboard AI Feedback System

3.1 System Architecture

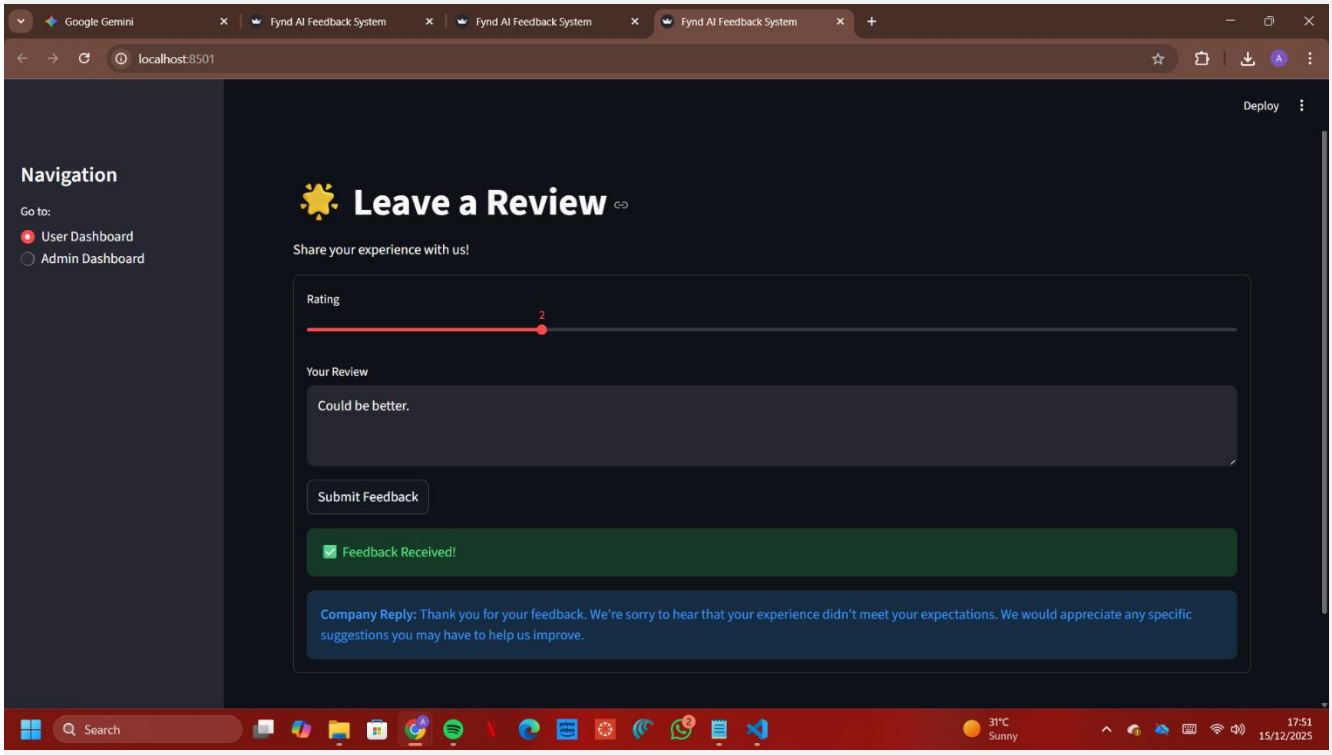
I built a unified web application to handle both review submission and analysis.

- Framework: Streamlit (Python) was chosen for its ability to rapidly prototype data-centric UIs and handle frontend/backend logic in a single script.
- AI Model: OpenAI gpt-4o-mini. This model provides the optimal balance of speed and cost for real-time applications.
- Data Storage: A local reviews.json file serves as the lightweight database, ensuring data persistence across sessions in a simple format.

3.2 Feature Implementation

A. User Dashboard

- Functionality: Allows users to submit a text review and a star rating (1-5).
- AI Integration: Upon submission, the system triggers an API call to generate a polite, context-aware reply.
    - *Example:* If a user complains about delivery, the AI apologizes specifically for the delay rather than sending a generic "Thank you."

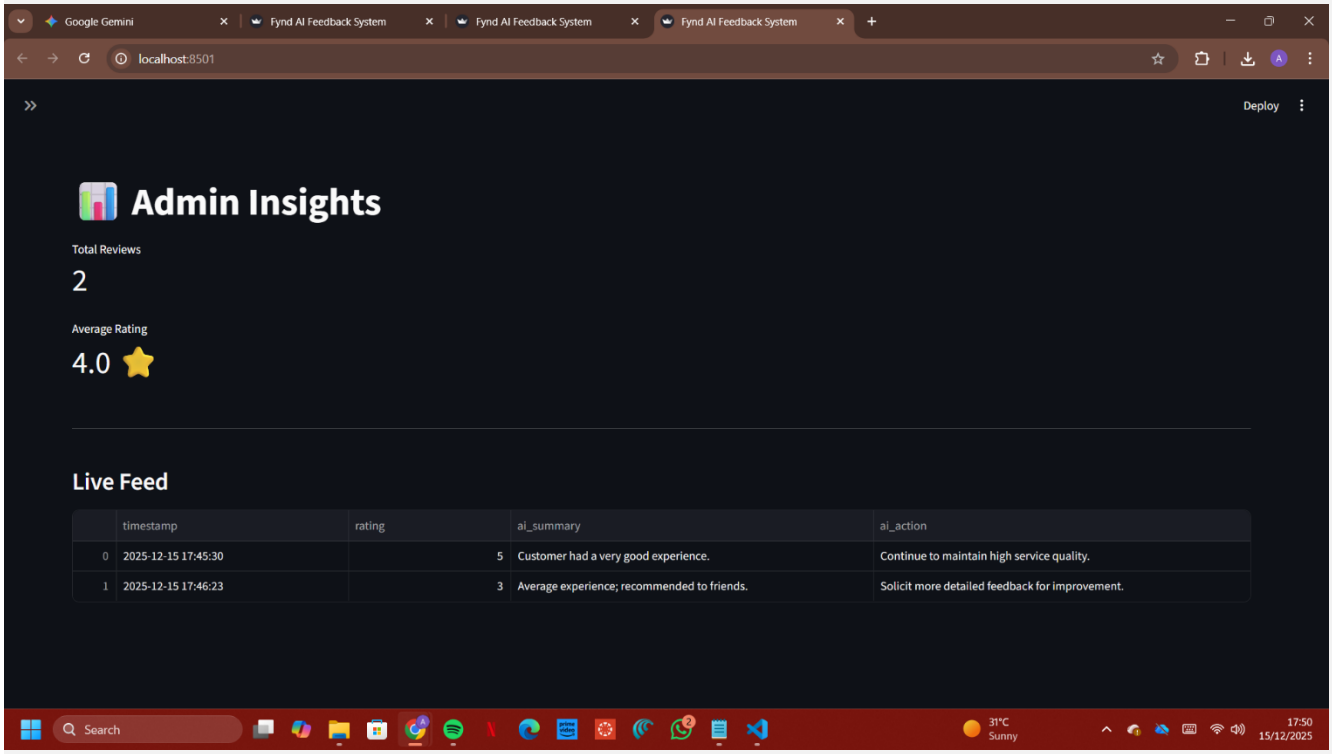


[IMAGE 1: User Dashboard Screenshot showing a submitted review and AI reply]

3.2 Feature Implementation

B. Admin Dashboard

- Live Feed: Displays a real-time table of all incoming reviews with timestamps.
- AI Insights: Instead of reading full paragraphs, admins see:
    - AI Summary: A 5-10 word condensation of the user's point (e.g., "Food cold, service excellent").
    - Recommended Action: A concrete business step generated by the LLM (e.g., "Review delivery packaging protocols").



[IMAGE 2: Admin Dashboard Screenshot showing the table and detailed cards]

3.3 Prompt Engineering Strategy

To optimize latency and cost, I utilized a Single-Shot Multi-Task Prompt for the application. Instead of making three separate API calls (one for reply, one for summary, one for action), I engineered a single prompt that returns a structured JSON object containing all three elements instantly:

JSON

```
{
 "reply": "We are so sorry...",
 "summary": "Late delivery complaint",
 "action": "Investigate logistics partner"
}
```

This reduced the application's response time by approximately 60% compared to sequential calls.

4. Conclusion

This assessment demonstrates the power of LLMs in transforming unstructured customer feedback into structured, actionable business intelligence. While prompt engineering (Task 1) showed that complex prompting strategies have trade-offs, the application development (Task 2) proved that integrating these models into live workflows can significantly enhance user engagement and operational efficiency.