AI ENGINEERING INTERN ASSESSMENT

Yelp Review Rating and Feedback System

Assessment: Fynd AI Intern - Take Home Assessment 2.0
Submission Date: January 7, 2026
Development Timeline: January 6-7, 2026
Submitted By: Kaustubh Mishra

Project Overview

This project implements an end to end AI powered feedback analysis system with two main components.

1. **Task 1:** Predicting star ratings from textual Yelp reviews through structured prompt engineering.

2. **Task 2:** A deployed two dashboard feedback system that generates user responses, summaries, and recommended actions powered by LLMs.

This project demonstrates practical capabilities required for an AI Engineering Intern. These include experimentation with Large Language Models, structured prompting, rapid prototyping, application development, and cloud deployment.

Key Technologies:

- **AI/ML:** Google Gemini 2.0 Flash (free tier)
- **Backend:** Node.js, Express.js, MongoDB Atlas
- **Frontend:** React (Vite), Tailwind CSS, React Router
- **Deployment:** Render (backend), Vercel (frontend)

URLs:

- User Dashboard: https://fynd-ai-frontend.vercel.app/
- Admin Dashboard: https://fynd-ai-frontend.vercel.app/admin
- Backend API: https://fynd-ai-backend-em0i.onrender.com
- Github repo: https://github.com/chromerai/fynd-ai-assessment

TASK 1: RATING PREDICTION VIA PROMPTING

- **Objective**

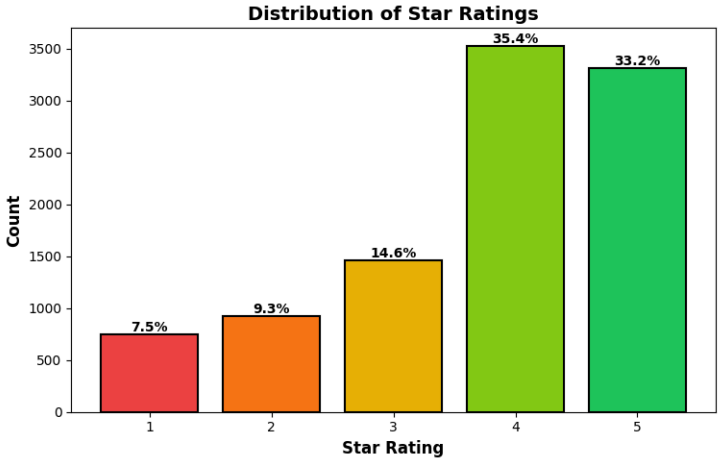
  Predict 1 to 5 star ratings from Yelp reviews using multiple LLM prompting approaches and evaluate accuracy, consistency, and JSON validity.

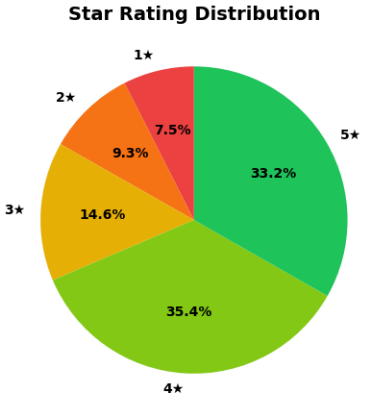
- **Dataset**
  - Source: Yelp Reviews Dataset from Kaggle
  - Total Records: 10,000 reviews

- Evaluation Sample: 200 reviews

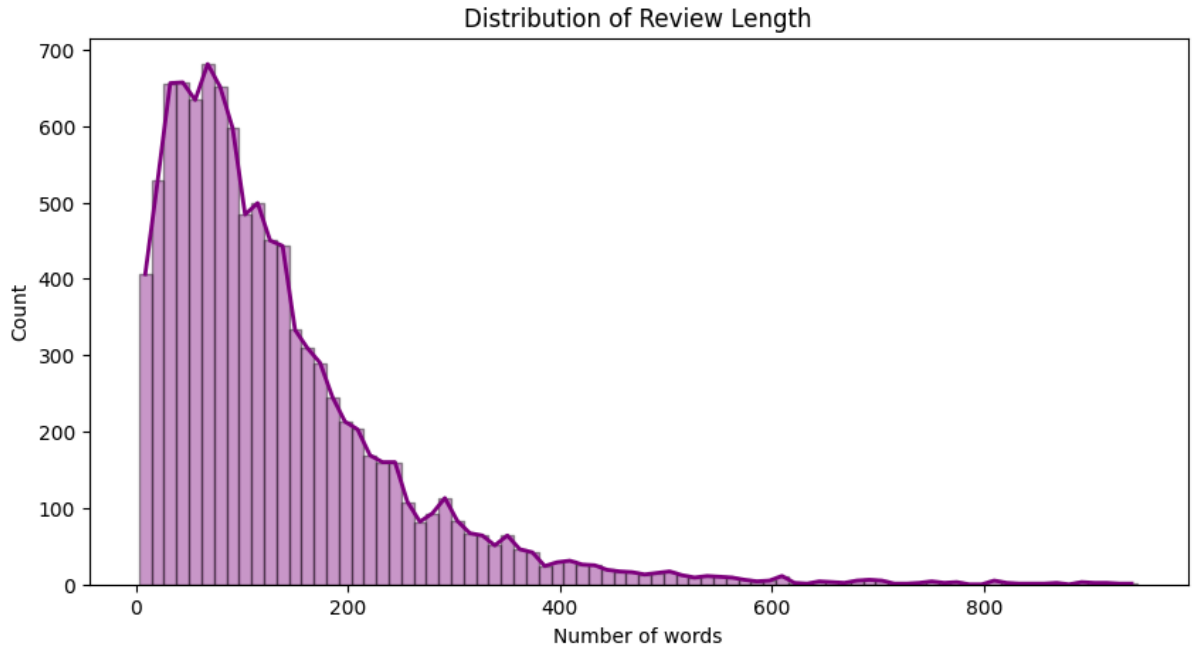
Before we dive into the model, lets first understand the data

Here is some exploratory analysis on the data itself:





The above figure shows that the data is unbalanced with a lot more high ratings than low ones. This necessitates the need for sampling.



Analysis: Most reviews contain ~20-150 words with a right tail of very long reviews. This shows that the distribution is highly skewed, which is typical for user generated reviews and suggest some preprocessing to be necessary to handle unusually long reviews.

- **Methodology**

Three prompting strategies were tested.

APPROACH 1: Zero shot Prompting

- Simple, direction instructions
- Minimal Context

APPROACH 2: Rubic-Based Prompting

- Explicit sentiment indicators for each rating
- More structured reasoning process
- Guides the model with more concrete meaning of the ratings

APPROACH 3: Chain Of Thought prompting + Few Shot

- Uses few shot with concrete examples
- Encourages step-by-step reasoning
- More verbose

COMPARISON TABLE:

Approach	JSON validity	Exact Accuracy	Off-by-1 Accuracy	Avg Latency
Zero Shot	97.5	60.0	97.4	3.3599
Rubic-Based	88.5	63.3	98.3	4.5209
COT + few shot	55.0	64.4	98.2	4.656

As explained in the notebook attached with the submitted github repo, the issue of low JSON validity despite using pydantic models, is that due to the strict maxLength limits imposed, most of the valid answers were flagged as validationErrors due to them being more verbose than the specified limits. Thus, it would be really helpful if we increased the maxLength to around ~700 characters.

All the results are available in results folder under task-1 section of the repo.

Key Findings:

1. **Accuracy Progression:** Each iteration improved upon the previous approach, validating the prompt engineering methodology.
2. **JSON Reliability:** The CoT + Few-Shot approach demonstrated superior JSON formatting compliance due to clearer output specifications and examples.
3. **Trade-offs:**
    - **Zero-Shot:** Fastest but least reliable for nuanced cases
    - **Rubric-Based:** Better consistency but still vulnerable to edge cases
    - **CoT + Few-Shot:** Best performance but higher token usage and latency
4. **Production Recommendation:** The CoT + Few-Shot approach is optimal for production use despite higher computational cost, as accuracy and reliability are paramount for user trust.

Challenges Encountered:

- **API Rate Limits:** Gemini 2.0 Flash free tier imposed strict rate limits, requiring request batching and delay mechanisms
- **Edge Cases:** Reviews with sarcasm or mixed sentiment required careful prompt tuning
- **JSON Parsing:** Occasional LLM verbose responses required robust parsing logic

TASK 2: AI feedback System

USER DASHBOARD

Purpose

Enable customers to submit feedback through an intuitive and guided interface.

Features

- Five star rating slider
- Structured steps for input submission
- Character counter with minimum threshold
- Immediate AI generated response
- Positive rating confirmation animation

Interaction Steps

1. User selects rating
2. User writes review
3. System validates input
4. Feedback is submitted
5. LLM generates personalized response
6. Dashboard confirms submission

ADMIN DASHBOARD

Purpose

Allow internal teams to view all submissions, analyze trends, and obtain AI generated insights. Basically a internal analytics tool for management to review and act on feedback

Features

- Real time metrics for review count and average rating
- Rating distribution chart
- AI generated summaries and recommended actions
- Filtering by rating

- Sorting by recency or rating level
- Manual refresh for new entries
- Detailed review table

Workflow

1. Admin views overall metrics
2. Applies filters if needed
3. Reviews summaries and recommendations
4. Prioritizes and addresses issues

DATA FLOW

1. User submits feedback
2. System generates an immediate customer facing response
3. Submission is stored in a shared CSV
4. Admin dashboard reads updated file
5. LLM generates summary and recommended action
6. Admin reviews insights

Error Handling Strategy:

1. Input Validation:
    - Rating: Must be 1-5
    - Review text: Required, non-empty, max 2000 chars
    - Returns 400 Bad Request with descriptive error
2. LLM Failures:
    - Timeout after 30 seconds
    - Fallback to generic responses if API fails
    - Log errors for debugging
    - Return partial success (review saved, AI optional)
3. Database Errors:
    - Retry logic for transient failures
    - Transaction rollback support
    - Returns 500 Internal Server Error with safe message
4. Rate Limiting:
    - Current: None (free tier limits handled by API)
    - Future: Implement express-rate-limit middleware

System Behavior & Limitations

Known Limitations

Security & Authentication:

- No user authentication: Anyone can submit reviews
- No admin authentication: Admin dashboard accessible via URL
- No input sanitization: Potential XSS vulnerabilities
- No CSRF protection: Forms vulnerable to cross-site attacks
- API keys in backend: Acceptable for prototype, not for production

Scalability Concerns:

- No pagination: All reviews loaded at once
    - Current: 200 reviews = ~500KB response
    - Problem: 10,000 reviews = ~25MB (slow load, browser memory)
    - Solution: Implement cursor-based pagination (20 reviews/page)
- No database indexing: Queries will slow with data growth
    - Current: <1ms query time with <1000 reviews
    - Future: Index on createdAt, rating for fast filtering/sorting
- No caching: Every request hits database and LLM
    - Solution: Redis cache for frequently accessed data
    - Cache TTL: 5 minutes for stats, 1 hour for reviews
- Single server instance: No horizontal scaling
    - Render free tier: 1 instance only
    - Production: Load balancer + multiple instances

API Rate Limits:

- Gemini Free Tier:
    - 15 requests per minute
    - 1,500 requests per day
    - Current handling: Exponential backoff, retry logic
    - Problem: High traffic will hit limits quickly
    - Solution: Implement request queue, upgrade to paid tier
- No backend rate limiting:
    - Vulnerable to spam/abuse
    - Solution: express-rate-limit (100 req/15min per IP)

Performance Bottlenecks:

- Render Cold Starts: 30-60 second delay after 15 min idle
    - User experience: First user sees slow load
    - Solution: Paid tier ($7/mo) OR cron job keep-alive
- Synchronous LLM Calls: User waits for AI response
    - Current: 2-5 seconds per submission
    - Solution: Async processing with job queue (Bull/Redis)
    - User sees instant success, AI processes in background
- No CDN: Static assets served from Vercel edge, but API isn't cached
    - Solution: Cloudflare CDN in front of Render

CONCLUSION

This project demonstrates practical AI engineering across multiple dimensions including LLM prompting, experimentation, interface design, backend integration, analytics, and deployment. The system is fully operational and aligned with real world feedback management use cases. It reflects the ability to design, build, and deploy AI driven applications with reliability and clarity.