Fynd AI Intern Assessment: Short Report

Akshata More

December 15, 2025

Project Resources

GitHub Repository: `https://github.com/akshacodes/Fynd_Internship_Assessment`
Live Web Application: `https://fyndassessmentakshata.streamlit.app/`

Summary

This report outlines my work on the two tasks for the AI Engineering Intern assessment. For **Task 1**, I experimented with different ways to prompt a local LLM to classify Yelp reviews. For **Task 2**, I built and deployed a feedback system using Streamlit and Google Sheets to handle real-time user data.

---

1 Task 1: Rating Prediction via Prompting

1.1 Objective

The challenge was to see if a smaller open-source model (`gemma2:9b`) could accurately predict a 1-5 star rating from a review and, crucially, return the result in a strict JSON format that a system could actually parse.

1.2 My Approach

I utilized a local Ollama setup to run the model and wrote a Python script to test it against a sample of **200 reviews** from the Kaggle dataset. I tracked two metrics: whether the model got the rating right (Accuracy) and whether it outputted valid JSON (Validity).

1.3 Prompt Strategies Tested

I iterated through three specific designs:

1. **Approach 1 (S1): Zero-Shot (The Baseline)**
   - **What I did:** I simply asked the model to "analyze sentiment" and gave it the JSON schema, with no examples.
   - **Why:** I needed a baseline to see if the model understood the task "out of the box."
2. **Approach 2 (S2): Few-Shot (The Guide)**
   - **What I did:** I provided 3 concrete examples (a 1-star rant, a 3-star mixed review, and a 5-star praise) inside the prompt before asking it to classify the target review.
   - **Why:** Models often struggle with specific formatting. I hypothesized that "showing" it what I wanted would work better than just "telling" it.
3. **Approach 3 (S3): Chain-of-Thought (The Thinker)**
   - **What I did:** I asked the model to "think step-by-step" about the pros and cons before giving the final rating.

- **Why:** This is usually a gold standard for reasoning tasks, so I wanted to see if it would improve accuracy on complex reviews.

1.4 Results (200 Samples)

Strategy	**Accuracy**	**JSON Validity**	**My Observation**
S1: Zero-Shot	40.00%	58.00%	Struggled heavily with formatting.
S2: Few-Shot	**69.50%**	**100.00%**	**The clear winner.** Stable and accurate.
S3: Chain-of-Thought	33.00%	51.50%	Over-explained and broke the JSON parser.

Table 1: Performance comparison on 200 reviews.

1.5 What I Learned

The results were actually quite surprising. While I expected Chain-of-Thought (S3) to be ”smarter,” it actually hurt performance here. The model became too verbose, trying to explain its logic so much that it often forgot to close the JSON brackets or added extra text that broke my parser.

Few-Shot (S2) was the absolute best approach. By simply giving it three examples, the model achieved **100% valid output** and nearly **70% accuracy**. It proves that for structured tasks, giving clear examples is far more effective than asking for complex reasoning.

---

2 Task 2: AI Feedback System (Web App)

2.1 Architecture Choices

I built a full-stack web app using ”Streamlit” because it allows for rapid UI development in Python.

- **Why Google Sheets?** I chose Google Sheets (via `gspread`) as my database instead of a CSV file. Since Streamlit Cloud spins up fresh containers on every reboot, a local CSV file would be wiped daily. Google Sheets gives me a persistent, free cloud database that is easy to inspect manually.
- **AI Engine:** I integrated the **Google Gemini API** (`gemini-flash-latest`) because it provides fast, free-tier access for summarizing reviews.

2.2 Design Challenges & Solutions

1. **Handling API Quotas:** During testing, I hit the ”429 Quota Exceeded” error because I was initially using a preview model. I updated the code to use the stable `gemini-flash-latest` version and added error handling so the app doesn’t crash if the API is busy.
2. **Data ”Cleaning” Filtering:** Real-world data is messy. Sometimes the AI service returns an error string instead of a valid summary. I implemented a filter in the Admin Dashboard to automatically detect and hide these ”corrupted” rows so they don’t skew the data.
3. **Bonus Analytics:** To make the Admin Dashboard more useful, I added an **Analytics Overview** section. This includes a bar chart for rating distribution and a line chart for daily review volume, helping store owners spot trends instantly.

2.3 Deployment Status

The app is fully deployed on Streamlit Community Cloud.

- **User Dashboard:** Publicly accessible for submitting reviews.
- **Admin Dashboard:** Live view of incoming data with AI summaries and analytics charts.
- **Security:** I used **Streamlit Secrets** to manage the Google Service Account credentials, ensuring no private keys were ever committed to GitHub.

Final Thoughts

Task 1 showed me that ”fancier” prompting (Chain-of-Thought) isn’t always better, sometimes you just need to show the model what you want (Few-Shot). Task 2 required handling real-world deployment constraints like persistent storage and API limits, resulting in a robust application that I’m proud to submit.