

TASK 1:

Overall Approach:

1. Data Preparation: Sampled 200 reviews from the Yelp dataset with balanced representation across all rating categories (40 reviews per star rating)
2. Prompt Design: Developed three distinct prompting approaches with varying complexity
3. Implementation: Used Google's Gemini 3 Flash API for all experiments
4. Evaluation: Measured accuracy, JSON validity rate, and consistency across approaches
5. Analysis: Compared results and identified trade-offs

In task 1 I have used zero-shot prompting, few-shot prompting, and chain-of-thought analysis approach. Now ideally, few-shot prompting should give more accuracy and it gave also but if we wanted to finetune a llm then chain-of-thought would be a more robust approach, as its intermediate steps may help the model to reach the predict closer to the actual value.

I was actually working on a project where I was building a fusion model to predict the price of stock market and a part of the model needed fine-tuning a llm for news analysis and there also I had used chain-of-thought approach and got very good results, but it might only work if number of samples is high.

Few-shot prompting seemed to be the best approach in this use case as the model does not have any context as to what is the trend or relationship between the review stars and description so giving a few examples in the prompt would help a lot in predicting review stars closer to actual value.

In the analysis if we observe all three approaches gave very similar results as the gemini is very well trained the accuracy might seem low due to model not predicting the exact value but if it is predicting a closer value like 3 instead of 4 or 4 instead of 5 then also it is a good sign we can keep refining the prompt and play with temperature and we might achieve higher accuracy.

Evaluation Methodology:

Metrics Used:

1. Accuracy: Percentage of correctly predicted ratings
 - Formula: $(\text{Correct Predictions} / \text{Total Predictions}) \times 100$
 - Primary metric for model performance
2. JSON Validity Rate: Percentage of responses that parsed successfully
 - Critical for production deployment
 - All approaches achieved 100% validity
3. Confusion Matrix Analysis:

- Examined which ratings were commonly confused
- 3-star reviews most frequently misclassified
- Adjacent ratings (e.g., 3 vs 4) harder to distinguish

Evaluation Dataset:

- 200 reviews (40 per rating category)
- Balanced sampling to avoid rating bias
- Preserved original text without preprocessing

Task 2:

Normal Operation Flow

1. User submits review on User Dashboard

- Frontend validates input
- Shows loading state
- Sends POST request to backend

2. Backend processes submission

- Validates again (defense in depth)
- Calls Gemini API 3 times (user response, summary, actions)
- Stores all data in MongoDB
- Returns user response

3. User sees AI-generated response

- Display with animation
- Option to submit another review

4. Admin Dashboard updates

- Auto-refreshes within 30 seconds (or manual refresh)
- New review appears at top of list
- Stats update automatically

Edge Cases Handled

1. Empty Review Text:

- Validation prevents submission
- Error message: "Please write a review"

2. Very Long Reviews (>5000 chars):

- Client-side validation shows character count
- Backend enforces limit

3. LLM Response Malformed:

- JSON parsing with fallbacks
- Regex extraction if JSON parsing fails
- Ultimate fallback: generic response

4. Concurrent Submissions:

- MongoDB handles with last-write-wins
- No transaction complexity needed for this use case

5. Backend Sleeping (Render free tier):

- First request may take 30-60 seconds
- Frontend shows "Connecting to server..." message
- Not ideal but acceptable for free tier

Trade-offs and Limitations

1. LLM Response Time

Trade-off:

- Generating 3 AI outputs takes 5-10 seconds
- User must wait for response

Alternatives Considered:

- Background processing (webhook/callback)
 - **Rejected:** Added complexity, worse UX (user doesn't see immediate response)
- Only generate user response immediately, others asynchronously
 - **Could be implemented:** But deployment timeline prioritized full implementation

Mitigation:

- Clear loading indicators
- User expectations set ("Processing your review...")

2. No Authentication

Current State:

- User dashboard is completely public
- Admin dashboard has no login (anyone with URL can access)

Rationale:

- Not required by assessment
- Simplifies deployment
- Focus on core functionality

Production Requirement:

- Admin dashboard MUST have authentication
- Options: JWT tokens, OAuth, API keys
- Could implement in 1-2 hours if needed

3. No Data Validation Beyond Schema

Current State:

- Validates data types and lengths
- No spam detection
- No profanity filtering

Trade-off:

- Simple implementation vs robust content moderation
- Acceptable for demo/prototype

Production Needs:

- Content moderation API integration
- Spam detection (ML-based or rule-based)
- Admin tools to flag/delete inappropriate content

4. Render Free Tier Limitations

Known Issues:

- Backend sleeps after 15 minutes of inactivity
- First request after sleep takes 30-60 seconds (cold start)
- 512 MB RAM limit

Impact:

- Poor user experience for first visitor
- Not suitable for production traffic