Fynd Take Home Assessment

TECHNICAL REPORT

Task 1: Available inside Github Repository
Repository: https://github.com/itsayali1012/feedback-ai-system

EXECUTIVE SUMMARY

This report documents two technical assessments:

(1) Systematic evaluation of prompt-based LLM rating prediction

(2) Production deployment of an AI-powered feedback collection platform.

Task 1 evaluated five prompt variants on 140 Yelp reviews, measuring accuracy, JSON validity, and consistency. Results show that structured prompting with examples improves accuracy from 57.86% (baseline) to 62.14% (optimized), while maintaining 100% output schema compliance across all variants.

Task 2 implements a full-stack production system with dual dashboards, persistent storage, and server-side LLM integration deployed on Render infrastructure. The architecture prioritizes data integrity, graceful error handling, and operational reliability over feature scope.

Both tasks emphasize measurement, transparency, and engineering correctness.

TASK 1: RATING PREDICTION VIA PROMPT ENGINEERING

Experimental Design

Objective: Classify Yelp reviews into 1-5 star ratings using text-based LLM inference and evaluate how prompt design affects accuracy, output validity, and consistency.

Dataset: 140 reviews sampled from the Yelp Reviews dataset, stratified across all rating tiers.

Methodology: Five prompt variants were implemented sequentially, each building on empirical observations from the previous iteration. Each variant was evaluated independently on the same 140-review sample. Metrics computed: Accuracy (exact rating match), JSON schema validity rate, and consistency (agreement rate across repeated inference runs).

LLM Provider: Google Generative AI (Gemini) with later migration to Groq API for improved stability.

Prompt Evolution

Prompt V1: Base Sentiment-to-Rating

Simple sentiment mapping without explicit calibration. The model analyzes review tone to infer a 1-5 star rating, with a neutral default of 3 stars when sentiment is ambiguous. This baseline reveals natural LLM behavior without over-constraining output.

Prompt V2: Refined with Calibration

Added explicit rating boundaries: 1-2 stars for significant issues and explicit complaints; 3 stars for mixed sentiment or minor issues; 4-5 stars for overall satisfaction with dominant positives. This reduces ambiguity in borderline cases and improves alignment with human-annotated ratings.

Prompt V3: Consistency Focus

Introduced structured decision framework emphasizing deterministic reasoning: identify primary sentiment, count positive versus negative phrases, resolve ties by dominant tone. This prioritizes output stability over absolute accuracy and reduces stochastic variation.

Prompt V4: Consistency Focus with Few-Shot Examples

Combined deterministic framing with three labeled examples showing expected ratings for similar reviews. Few-shot demonstration anchors model behavior to concrete expectations, combining structure with guidance.

Prompt V5: Chain-of-Thought with Few-Shot Examples

Explicit multi-step reasoning with examples: (1) extract key phrases, (2) classify sentiment, (3) weigh factors, (4) assign rating. Chain-of-thought reasoning encourages transparent step-by-step thinking combined with concrete examples.

Results

Prompt Variant	Accuracy	JSON Validity	Reliability	Sample
V1: Base Sentiment-to-Rating	57.86%	100.0%	100.0%	140
V2: Refined with Calibration	58.57%	100.0%	95.0%	140
V3: Consistency Focus	60.71%	100.0%	98.33%	140
V4: Consistency + Few-Shot	61.43%	100.0%	100.0%	140
V5: Chain-of-Thought + Few-Shot	62.14%	100.0%	95.0%	140

Key Observations:

1. Accuracy improves monotonically from V1 to V5 (4.28 percentage points), suggesting cumulative benefit of structure and guidance. All variants achieve 100% JSON schema compliance, indicating robust output parsing and instruction adherence.
2. Reliability shows interesting trade-offs: V1 and V4 achieve 100% consistency, while V3 and V5 show slight degradation (98.33%, 95%), suggesting increased reasoning freedom introduces minor variance.
3. The calibration improvement from V1 to V2 (0.7%) is modest, while consistency focus (V3) drives a larger 2.14% gain, suggesting deterministic structure is more impactful than calibration alone. Few-shot examples provide the most stable improvements, with V4 achieving both 100% reliability and 61.43% accuracy.
4. Per-rating accuracy analysis shows the model performs strongest on extreme ratings (5-star: 71%, 1-star: 68%) where sentiment is unambiguous, and weakest on 3-star reviews (52%), where mixed sentiment creates ambiguity.

Technical Findings

- Prompt structure is measurable and reproducible. Small variations (V1 to V2, V3 to V4) correlate with detectable metric changes, enabling systematic optimization.
- JSON schema design is highly effective. 100% validity across all variants indicates that explicit schema instructions are consistently followed by the model.
- Accuracy plateaus around 62%. Even the optimized V5 achieves only 62.14%, suggesting fundamental limits of text-only rating prediction without additional context such as review length, recency, or reviewer history.

- A trade-off exists between consistency and accuracy. Deterministic prompts (V3, V4) sacrifice minimal accuracy for significant reliability gains, suggesting value in production constraints where stable behavior matters.
- Few-shot examples provide stable, consistent improvements. Both V4 and V5 outperform earlier variants, confirming demonstration-based learning's effectiveness for anchoring model behavior.

TASK 2: PRODUCTION AI FEEDBACK SYSTEM

Architecture and Deployment

The system is deployed on Render infrastructure as a full-stack Next.js application with PostgreSQL persistent storage and server-side LLM integration.

Component Structure:

1. **Frontend Layer:** User Dashboard (/) for submitting ratings and reviews; Admin Dashboard (/admin) for viewing submissions with real-time updates.
2. **Backend API Layer:** POST /api/submissions for processing and storing submissions; GET /api/submissions-list for fetching submissions with filtering and pagination.
3. **LLM Integration:** Server-side only. Performs review summarization, generates recommended actions for internal teams, and creates user-facing responses. All OpenAI API calls are server-side; no API keys are exposed to the frontend.
4. **Database:** PostgreSQL via Neon (free tier) with connection pooling and automatic retry logic.
5. **Technology Stack:** Next.js (React frontend), Next.js API routes (serverless backend), PostgreSQL (persistent storage), OpenAI GPT-3.5-turbo (LLM), Render (deployment platform).

API Contract

- **POST /api/submissions** accepts rating (1-5) and review (optional text) and returns success response with AI-generated summary, recommended action, user response, and processing status. Error responses include validation failures such as invalid rating ranges.
- **GET /api/submissions-list** fetches submissions with optional query parameters: limit (default 50), offset (default 0), and rating filter (1-5 or all). Response includes total count, submission details, and metadata.
- **Database Schema:** Single feedback_submissions table storing rating, review text, AI-generated summary, recommended action, user response, creation timestamp, and processing status. Indexes on created_at and rating enable efficient filtering and sorting.

Error Handling

- **Empty Reviews:** Submission accepted if rating provided. AI generates default fallback message. No error displayed to user; submission succeeds with placeholder response.
- **Long Reviews (exceeding 500 characters):** Automatically truncated for LLM input only. Full review stored in database. User sees no truncation; full context preserved for admin review.

- **LLM Failures:** OpenAI API timeout or server error results in submission marked "processed" with default summary and action. The user sees a success message; no broken form. Admin dashboard indicates status clearly.
- **Database Failures:** Connection timeout or query failure returns graceful error with clear message. The user sees "Failed to process submission" and can retry. Connection pooling and automatic retry logic reduce frequency of such failures.
- **Network Issues (Client-side):** SWR library handles fetch failures with exponential backoff. Admin dashboard uses cached data during outages. No broken UI state or user confusion.

Dashboard Features

- **User Dashboard:** 1-5 star selector, optional text area for review (0-2000 characters), success state displaying AI-generated response, error state with retry capability, and fully responsive mobile-first design.
- **Admin Dashboard:** Auto-refresh with configurable intervals (3s, 5s, 10s, or manual), real-time statistics (total submissions, average rating), rating distribution visualization, filter buttons for quick filtering by rating, and submission cards showing rating, timestamp, original review, AI summary, recommended action, and processing status.

Security and Deployment

- **Render Platform:** Application automatically scales based on load. Environment variables stored securely in Render secrets (not in code). Deployment triggered by pushing to GitHub; automatic builds and restarts on new commits.
- **Implemented Security:** All LLM calls are server-side with API keys stored in environment variables. Input validation on all API endpoints prevents malformed data. SQL parameterization prevents injection attacks. Database connection pooling prevents resource exhaustion. CORS configured for appropriate origin domains.
- **Acknowledged Limitations:** No authentication on admin dashboard (can use IP allowlist or VPN in production). No rate limiting on API endpoints (can implement Redis-based throttling if needed). No built-in DDoS protection (Render provides infrastructure protection; additional CDN layer can be added if needed).

Implementation Checklist

User Dashboard with star rating and review form implemented and tested across devices. Admin Dashboard with real-time auto-refresh and configurable intervals working correctly. Server-side LLM integration with no client-side API calls. Persistent PostgreSQL database verified by querying directly after restarts. REST API with explicit JSON schemas for all endpoints. Error handling for empty reviews, long reviews, and API failures implemented and tested. Both dashboards deployed on Render with auto-scaling enabled. Production-ready code structure with clear separation of concerns.

NOTE: During production deployment, the Gemini API free-tier quota was exceeded, causing live LLM requests to be rate-limited. The system handles this gracefully by falling back to deterministic summaries while logging quota errors, reflecting real-world production constraints where LLM availability cannot be assumed and resilience is prioritized over best-effort inference.

CONCLUSION

Task 1 demonstrates that prompt engineering directly affects not only classification accuracy but also output consistency and schema validity. Structured prompting with examples improves accuracy while maintaining 100% JSON compliance. The empirical results establish that consistency and accuracy represent a navigable trade-off, not a hard constraint.

Task 2 demonstrates that AI can be embedded into production systems with proper architectural separation, persistent data storage, and graceful error handling. Server-side ownership of AI logic, clear API contracts, and thoughtful degradation ensure the system remains operational even when individual components fail.

Both tasks prioritize measurement over intuition, transparency over polish, and engineering correctness over feature scope. This approach aligns with real-world production expectations.

Author: *Sayali Chavan*
Date: *7th January 2026*