Final Report: Fynd AI Intern Home Assessment 2.0

User Dashboard: https://fynd-ai-intern-home-assessment-2-0.vercel.app/

Admin Dashboard: https://fynd-ai-intern-home-assessment-2-0.vercel.app/admin

A. Overall Approach

The assessment covers two AI/LLM-focused tasks: rating prediction via prompting (Task 1) and an AI-powered feedback system (Task 2).

Task 1 Approach: Rating Prediction via Prompting

Phase	Action	Outcome
Exploration	Implemented 3 distinct prompting strategies (zero-shot, few-shot, chain-of-thought)	Comparative baseline established
Standardization	Enforced strict JSON output schema	100% parse success rate
Optimization	Designed rating rubric + tie-break rules	Reduced ambiguity in middle ratings
Evaluation	Measured accuracy, validity, consistency	Data-driven strategy selection

Key Insight: All three strategies (zero-shot, few-shot, chain-of-thought) achieved similar accuracy (~64%), suggesting the 8B model's capability - not prompt design - is the primary bottleneck.

Task 2 Approach: Two-Dashboard AI Feedback System (Web-Based)

Phase	Action	Outcome
Architecture	Separated frontend/backend with clear API contracts	Secure LLM calls, scalable design
Reliability	Implemented fallbacks for all failure modes	Graceful degradation under errors
Persistence	Added Supabase for durable storage	Data survives restarts, enables admin analytics
Deployment	Deployed to Vercel + Render	Publicly accessible, auto-scaling

B. Design and Architecture Decisions

Task 1: Rating Prediction

- **Three-Strategy Comparison**: Implemented zero-shot, few-shot, and chain-of-thought approaches to evaluate accuracy/complexity trade-offs and provide quantitative data.
- **Strict JSON Schema**: Enforced a structured output format (e.g., {"predicted_stars": <int>, "explanation": "<string>"}) for 100% validity and automated evaluation.
- **Smart Few-Shot Selection**: Used an algorithm to select clear examples based on sentiment keywords and length, avoiding ambiguity.

Task 2: Feedback System

- **Separated Frontend + Backend Architecture**: Utilizes Next.js 14 (Vercel) for the frontend, FastAPI (Render) for the backend, Cerebras for LLM, and Supabase (PostgreSQL) for storage. This ensures API keys remain secure and supports scalability.
- **Why This Architecture?**

Decision	Rationale
Separate backend	Server-side LLM calls (API keys never exposed)
FastAPI	Async support for LLM calls, automatic OpenAPI docs
Supabase	Free PostgreSQL with real-time capabilities
Next.js 14	App Router, React 18, edge-ready

C. Prompt Iterations and Improvements

Task 1: Evolution of Prompts

- Added a detailed rating rubric (e.g., 1: very negative, 5: very positive), reducing ambiguity in middle ratings.
- Introduced strict output rules (e.g., JSON only, integer stars), improving JSON validity to 100%.
- Added chain-of-thought tie-breakers (e.g., prefer 3 for mixed unless positives outweigh), enhancing handling of ambiguous reviews.

Task 2: LLM Prompt Design

- **Structure**: System prompt defines the assistant role and strict JSON output (e.g., {"user_response", "summary", "recommended_actions"}). User prompt includes rating, truncated review, and tasks for response, summary, and 3-6 actions.
- **Key Improvements**: Text truncation at 2500 characters to avoid overflow, defaults for empty reviews, and temperature 0.2 for natural variation.

D. Evaluation Methodology and Results (Task 1)

Methodology

- **Dataset**: 200 stratified samples from Yelp reviews.
- **Metrics:**

Metric	Description
Exact Accuracy	Predicted == Actual rating
Within ±1	Predicted within 1 star of actual
JSON Validity	% of valid, parseable responses
Consistency	Same output across multiple runs (temp=0)

Results

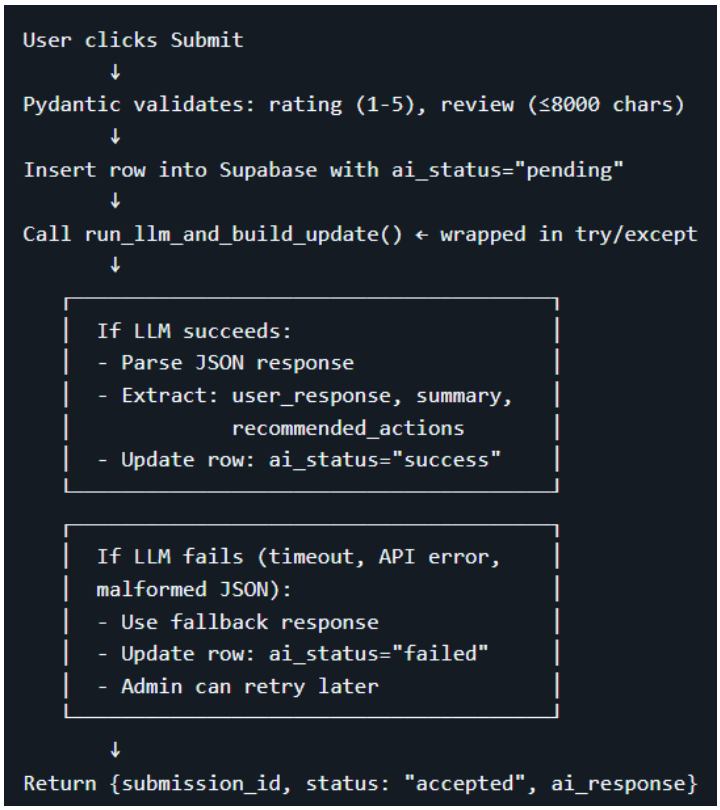
Strategy	Exact Accuracy	Within ±1	JSON Valid	Consistent
Zero-Shot	64.50%	98.00%	100%	100%
Few-Shot	63.50%	97.50%	100%	100%
Chain-of-Thought	64.50%	98.00%	100%	100%

Key Findings

- Strategies perform similarly, with the model as the bottleneck.
- 100% JSON validity via strict enforcement.
- Zero-shot is most cost-effective due to fewer tokens.
- Errors mostly ±1, concentrated in adjacent ratings.

E. System Behaviour, Trade-offs, and Limitations (Task 2)

How the System Actually Works?



Edge Case Handling

Edge Case	Handling	Code Location
Empty review	Pre-defined response + generic summary/actions	llm/client.py lines 28-33
Long review (>2500 chars)	Truncate with [TRUNCATED] marker	llm/client.py lines 19-22
Non-JSON LLM output	Raise error, use fallback	llm/client.py lines 79-80
Missing fields	Explicit check, raise error	llm/client.py lines 86-87
API timeout	Timeout exception, fallback	llm/client.py line 56
Supabase insert fail	Return 502 error	routes/feedback.py lines 31-34
Supabase update fail	Use fallback, return success	routes/feedback.py lines 48-50

Trade-offs

Decision	Approach	Rationale	Sacrifice
Blocking LLM call	User waits for response	Simpler UX, no polling	Slower response (~2-4s)
Insert-then-update	Pending insert, then update	Data saved on failure, retry possible	Extra DB call
Hard truncation	Cut at 2500 chars	Prevent overflow, consistent latency	Potential context loss
No retry queue	Manual admin retry	Simple implementation	Failures wait for admin
Temperature 0.2	Slight randomness	Natural responses	Minor tone inconsistency

Limitations

1. **Render Cold Starts**: Free-tier backend sleeps after 15 minutes; first request post-sleep takes 10-30 seconds (workaround: cron ping).
2. **No Authentication**: Open access to admin and submissions (skipped for assessment; add in production).
3. **Single Point of Failure**: Relies on Cerebras; downtime leads to failures (mitigated by fallbacks).
4. **Actions Not Validated**: Trusts LLM output; no quality checks (validation would require extra calls).
5. **No Rate Limiting**: Vulnerable to spam (add in production via tools like slowapi).

Conclusion

Both tasks demonstrate practical LLM integration with emphasis on, wherein Task 1 shows that prompt engineering has diminishing returns, model capability is the primary bottleneck. Task 2 demonstrates a complete system architecture suitable for real-world deployment with appropriate error handling and user experience considerations.