REPORT

TASK 1: RATING PREDICTION VIA PROMPTING

Link: https://github.com/Loki-33/fynd-Assessment
Notebook Link:
https://colab.research.google.com/drive/1UDQRranwESGfxOZCcYP8rLKPAF6wR6GY?usp=sharing

Model Used: xiaomi/mimo-v2-flash:free

OVERVIEW

1. OpenRouter is used for API calls
2. Three prompting approaches are used and compared

APPROACH-1: DIRECT METHOD

- **Design:** Simple, straightforward instruction
- **Rationale:** Minimal complexity, direct task specification
- **Expected Strengths:** Fast, consistent JSON format
- **Expected Weaknesses:** May lack nuanced understanding

SYSTEM PROMPT:

```
You are a rating prediction system.
Analyze reviews and predict star ratings from 1 to 5.
Always respond with valid JSON only, no other text.
Format: {"predicted_stars": <number>, "explanation":
"<reason>"}
```

APPROACH-2: SENTIMENT ANALYSIS METHOD

- **Design:** Structured reasoning with explicit sentiment mapping
- **Rationale:** Guides model through analytical process
- **Expected Strengths:** Better reasoning, more accurate predictions
- **Expected Weaknesses:** More verbose, potential JSON format issues

SYSTEM PROMPT:

```
You are an expert sentiment analyzer for business reviews.
Follow this process:
 1. Identify positive and negative aspects
 2. Assess overall sentiment intensity
 3. Map to star rating:
   - 5 stars: Extremely positive, enthusiastic
   - 4 stars: Positive with minor issues
   - 3 stars: Mixed or neutral feelings
   - 2 stars: Mostly negative with few positives
   - 1 star: Extremely negative, angry

Respond only with valid JSON: {"predicted_stars":
<number>,"explanation": "<reason>"}
```

APPROACH-3: FEW SHOT METHOD

- **Design:** Provides 5 examples covering all rating levels
- **Rationale:** Learning by example, pattern recognition
- **Expected Strengths:** Better calibration, consistent scale usage
- **Expected Weaknesses:** Longer prompts, higher token usage

SYSTEM PROMPT:

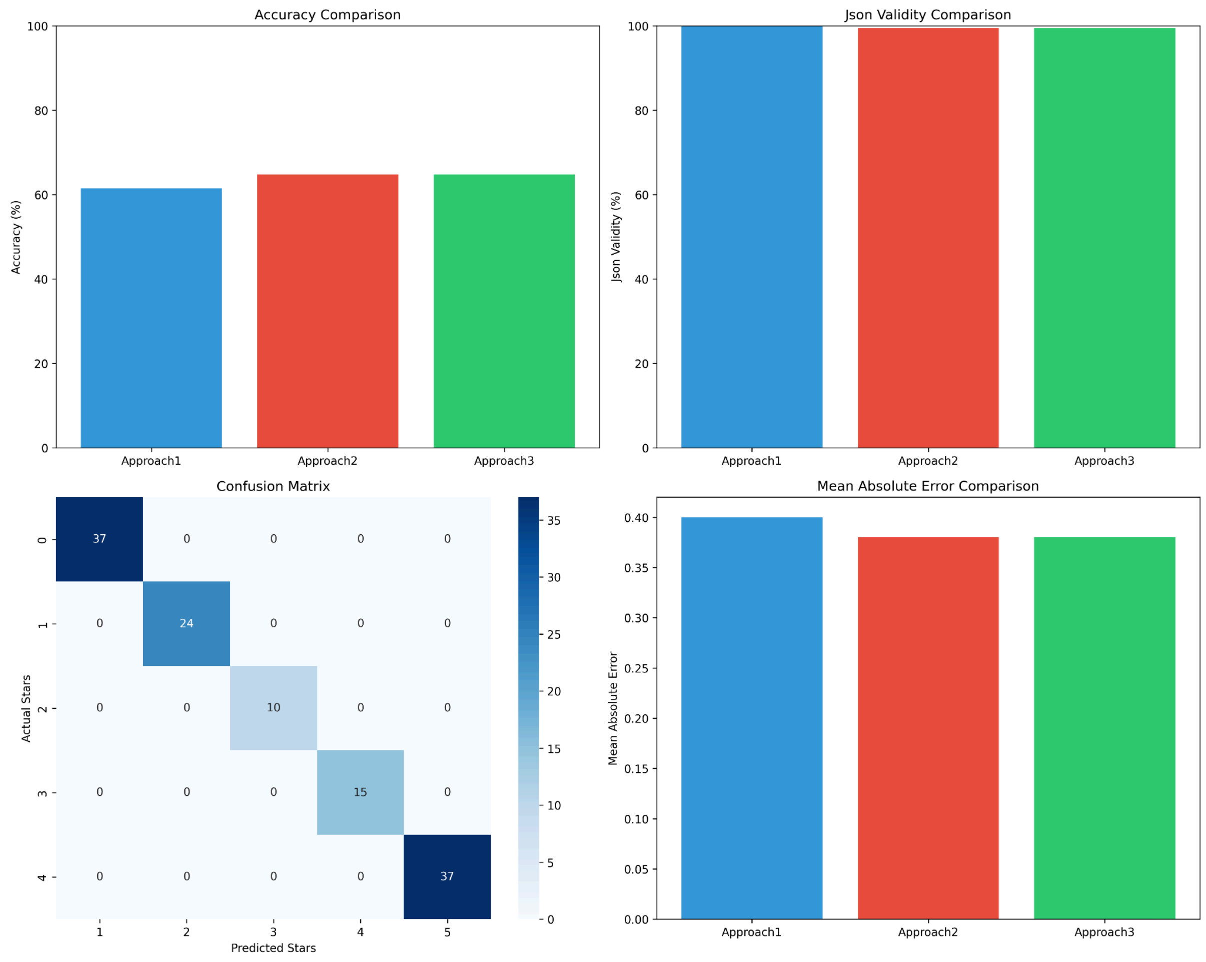
```
You are a rating prediction expert.
Learn from the examples and predict rating for new reviews.
Always return valid JSON only."""

user_prompt = f"""Predict the rating for this review:
"{review_text}"
Using the following examples:
 1. Review: "Absolutely amazing! Best service ever, food was
delicious!" -> 5 stars
 2. Review: "Pretty good overall, service was nice but food
was a bit cold." → 4 stars
 3. Review: "It was okay, nothing special but not bad either."
→ 3 stars
 4. Review: "Disappointing experience, slow service and cold
food." → 2 stars
 5. Review: "Terrible! Worst experience ever, will never
return!" → 1 star
Return only valid JSON: {{"predicted_stars": <number>,
"explanation": "<reason>"}}.
```

METRICS USED

1. Accuracy
2. Mean Absolute Error
3. JSON Validity

COMPARISON PLOT



CONCLUSION

Approach-3 i.e Few-shot Method did the best which was not surprising as the LLM then understands better what was expected of him, we could further improve the prompt for Approach-3 by providing more diverse examples, so it could further understand the task.

TASK 2: TWO-DASHBOARD AI FEEDBACK SYSTEM (WEB-BASED)

Link: https://github.com/Loki-33/fynd-Assessment

BACKEND: https://fynd-assessment-api.onrender.com/docs
USER-DASHBOARD: https://fynd-assessment-front.onrender.com/user
ADMIN-DASHBOARD: https://fynd-assessment-front.onrender.com/admin

OVERVIEW

A full-stack application for collecting and analyzing customer review using an LLM-NVIDIA: Nemotron 3 Nano 30B A3B

FEATURES

1. **Submission:** Users can submit ratings (1-5) and a detailed feedback
2. **AI-Powered Analysis:**
    a. Automated response generation for customer engagement
    b. Intelligent Review summarization
    c. Actionable insight extraction
3. **Admin Dashboard:** Can view all the reviews so far along with feedback and summaries from AI
4. **Error Handling: Handles** empty reviews, long text and API failures

BACKEND:

1. **FastAPI**: High processing Python web framework
2. **SQLAlchemy**: ORM for database operation
3. **PostgresSQL**: Primary Database
4. **OpenRouter**: LLM API for AI processing
5. **Pydantic**: Data Validation and Serialization

FRONTEND

1. **Next.js:** React framework with server-side rendering
2. **React:** Component-based UI

DEPLOYMENT

1. Deployed both the frontend and the backend in **Render**, as we can use its own database i.e postgresSQL which makes the task much easier
2. And **Git** for version control CI/CD

FOR LOCAL USE

1. Run the backend server by using the following steps:
    a. Go to “app” folder
    b. Install the requirements
    c. Set environment variables: `DATABASE_URL=postgresql://localhost/yourdb" > .env` and `"OPENROUTER_API_KEY=your_key_here" >> .env`
    d. Run server: `uvicorn main:app --reload`
2. Run the frontend now:
    a. Go to the frontend folder
    b. Set Environment Variables: `“NEXT_PUBLIC_API_URL=http://localhost:8000" > .env.local`
    c. npm install
    d. npm run dev

LIMITATIONS

1. **Cold Starts:** Render free tier spins down after 15 min inactivity (~30s startup)
2. **Database Persistence:** Free PostgreSQL expires after 90 days.
3. **Rate Limits:** OpenRouter API rate limits apply

FUTURE ENHANCEMENTS

1. Add authentication for admin dashboard
2. Implement pagination for review list
3. Add sentiment analysis visualization
4. Export Reviews to csv