Task 1: Rating Prediction via Prompting

1. Problem Overview

The goal of Task 1 was to predict Yelp review star ratings (1–5) using prompt-based LLM inference instead of training a supervised model. The focus was on designing a robust prompting strategy, running it reliably at scale (200 reviews), and analysing accuracy, JSON validity, and error patterns.

Because hosted APIs such as Grok, OpenAI, and OpenRouter enforce strict per-day limits and per-request quotas, they were not suitable for running 200 calls per prompt style for this task. Hugging Face-hosted models that performed reasonably well were also too slow for this volume, leading to timeouts and impractical runtimes in an iterative experimentation loop.

To avoid these bottlenecks while staying within the task requirement of using a free LLM, the entire pipeline was moved to local inference via Ollama, using lightweight models that can run comfortably on a personal machine.

---

2. System Architecture

2.1 High-Level Pipeline

The end-to-end architecture for Task 1 has the following stages:

1. Data ingestion
   - Load Yelp reviews from CSV (Kaggle dataset) into a pandas DataFrame.
   - Keep only `review_id`, `text`, and `stars` columns.
2. Filtering & sampling
   - Remove rows with null or empty `text` or `stars`.
   - Select the first 200 reviews for consistent and efficient evaluation.
3. Prompting strategies (3 variants)
   - Exemplar-Based Prompting.
   - Analytical Step-by-Step Prompting.
   - Direct & Simple Prompting.
4. LLM inference via Ollama

    - Use `ollama` Python client to send prompts to `gemma2:2b` running locally.
    - Enforce a strict JSON output schema: `{"predicted_stars": int, "explanation": str}`.
5. Post-processing and JSON validation
    - Parse the model output as JSON.
    - Validate presence and range of `predicted_stars` in [1, 5].
    - Log invalid or unparsable responses.[file:1]
6. Evaluation & logging
    - Compute accuracy, JSON validity rate, and average absolute error per strategy on the 200 reviews.[file:1]
    - Save detailed per-example results to timestamped JSON files for reproducibility.[file:1]

2.2 Technology Stack

- Runtime / Environment
    - Local machine with Ollama installed and configured.
    - Python notebook (`task1.ipynb`) for experiment orchestration.[file:1]
- Core Python libraries
    - `pandas` for data loading and manipulation.[file:1]
    - `json` and `re` for output parsing and validation.[file:1]
    - `ollama` for interacting with the local LLM server.[file:1]
    - `datetime`, `matplotlib`, `numpy` for logging and optional analysis/visualizations.[file:1]
- Model configuration
    - Primary model: `gemma2:2b` (Ollama) for all reported results.[file:1]
    - Number of reviews per strategy: 200.[file:1]

---

3. Model Selection and Local LLM Setup

3.1 Motivation for Local LLMs

Initial attempts to use remote APIs (Grok, OpenAI, OpenRouter) quickly hit per-day request limits when scaling to 200 reviews × multiple prompt variants.[file:1] Additionally, some Hugging Face models that were strong for text classification had high latency, making them impractical when running hundreds of calls with experimentation and debugging in the loop.[file:1]

The task description allowed using any free LLM, so the setup was shifted to Ollama, which provides a convenient interface to run open models entirely locally.[file:1] This eliminated API rate limits, reduced network overhead, and gave full control over how many times the model can be called.

3.2 Candidate Models Evaluated

To decide which local model to standardize on, several Ollama models were tested on a small, manually inspected subset of 10 random Yelp reviews.[file:1] For each model, the following were checked:

- How often the predicted rating matched the ground-truth label ("true" predictions).
- Qualitative quality of the explanation and stability of behavior.

Models tried (with reported sizes):

- `codellama:7b` – ~3.8 GB.[file:1]
- `gemma:2b` – ~1.7 GB.[file:1]
- `llava-phi3:3.8b` – ~2.9 GB.[file:1]
- `gemma2:2b` – ~1.6 GB.[file:1]

On the small 10-sample sanity check:

- `gemma2:2b` achieved around 80% correct labels.
- The other models were in the 60–70% range on this quick spot-check.[file:1]

Although this is a small sample, it was sufficient for a pragmatic choice: `gemma2:2b` gave the best combination of:

- Higher hit rate on the ad-hoc test.
- Smaller size (1.6 GB) and faster inference compared to some larger models.
- Stable, coherent text generation under the chosen prompts.[file:1]

Therefore, `gemma2:2b` was selected as the primary model for all subsequent experiments.

---

4. Dataset Preparation and Execution Flow

4.1 Dataset and Preprocessing

- Dataset source: Yelp Reviews dataset from Kaggle (`omkarsabnis/yelp-reviews-dataset`).[file:1]
- Columns selected: `review_id`, `text`, `stars`.[file:1]
- Cleaning:
  - Dropped rows with null `text` or `stars`.
  - Filtered out empty or whitespace-only reviews.[file:1]
- Sampling:
  - Took the first 200 cleaned reviews for this task.[file:1]
  - The final star distribution on these 200 samples (example counts):
    - 1-star: 21
    - 2-star: 11
    - 3-star: 28
    - 4-star: 71
    - 5-star: 69.[file:1]

This distribution is skewed towards higher ratings (4–5), which is typical for real-world review datasets and important context when interpreting accuracy.[file:1]

4.2 Notebook Execution Steps

The notebook (`task1.ipynb`) is structured into clear sections:[file:1]

1. Imports and configuration
   - Import pandas, json, re, ollama, datetime, etc.[file:1]
   - Set `MODEL_NAME = "gemma2:2b"` and `NUM_REVIEWS = 200`.[file:1]
   - Print configuration summary ("Using Ollama locally with gemma2:2b …").[file:1]
2. Dataset loading & cleaning
   - Read CSV from local path (`C:\fynd-task\yelp.csv`).[file:1]
   - Apply filters and sample the first 200 entries.[file:1]
   - Print star distribution for sanity check.[file:1]
3. Prompt definitions
   - Define three separate prompt templates in Python corresponding to:
     - Exemplar-Based Prompting.
     - Analytical Step-by-Step Prompting.
     - Direct & Simple Prompting.[file:1]
4. Inference loop
   - Iterate over the 200 selected reviews.
   - For each prompting strategy:

        - Fill the template with the current review text.
        - Call `ollama.chat(...)` or equivalent to query `gemma2:2b`.
        - Collect raw model responses in memory or streaming.[file:1]
5. Output parsing and validation
    - Use `json.loads` and regex to robustly extract JSON from the LLM output.
    - Check that `predicted_stars` exists and is in the range 1–5.
    - Mark any failures as invalid responses for validity statistics.[file:1]
6. Metrics computation and logging
    - For each strategy, compute:
        - Accuracy: fraction of predictions matching ground-truth stars.
        - JSON Validity Rate: valid responses / total.
        - Average absolute error in star ratings.
    - Save a detailed JSON file per strategy (filename includes strategy name and timestamp).[file:1]

---

5. Custom Prompting Strategies

A key part of this task was designing hybrid prompting that reflects a personal prompting style rather than just copying a template.[file:1] Three complementary strategies were implemented and evaluated:

5.1 Exemplar-Based Prompting

- Idea: Show the model concrete examples of typical 1-star, 3-star, and 5-star reviews along with their correct ratings.[file:1]
- Goal: Anchor the model's understanding of how sentiment maps to rating levels before asking it to classify a new review.[file:1]

Characteristics in this implementation:

- Prompt contains few-shot examples with both text and labeled star ratings.
- The target review is appended after the examples, and the model is asked to output a JSON object with `predicted_stars` and an `explanation`.[file:1]
- This often improved calibration because the model can compare the new review against the provided prototypes.[file:1]

5.2 Analytical Step-by-Step Prompting

- Idea: Force the model to think in structured steps before outputting the final rating.[file:1]
- Goal: Increase explainability and consistency by guiding the reasoning process.[file:1]

Structure used:

1. Identify positive aspects of the review.
2. Identify negative aspects or complaints.
3. Assess overall sentiment intensity.
4. Map that assessment to a final star rating from 1 to 5.
5. Provide the answer as JSON with both `predicted_stars` and a short explanation.[file:1]

This approach trades off speed for transparency, since the model generates more text, but it tends to yield more consistent reasoning and clearer justifications.[file:1]

5.3 Direct & Simple Prompting

- Idea: Use a minimal, clean instruction that directly asks for the star rating with almost no scaffolding.[file:1]
- Goal: Test the intrinsic capability of `gemma2:2b` to understand sentiment and numeric ratings without extra examples or step-wise reasoning.[file:1]

Key features:

- Short prompt: “Given the following Yelp review, output a JSON with `predicted_stars` (1–5) and a one-sentence explanation.” (paraphrased).[file:1]
- Very little extra text, which makes this the lightest and fastest strategy.[file:1]

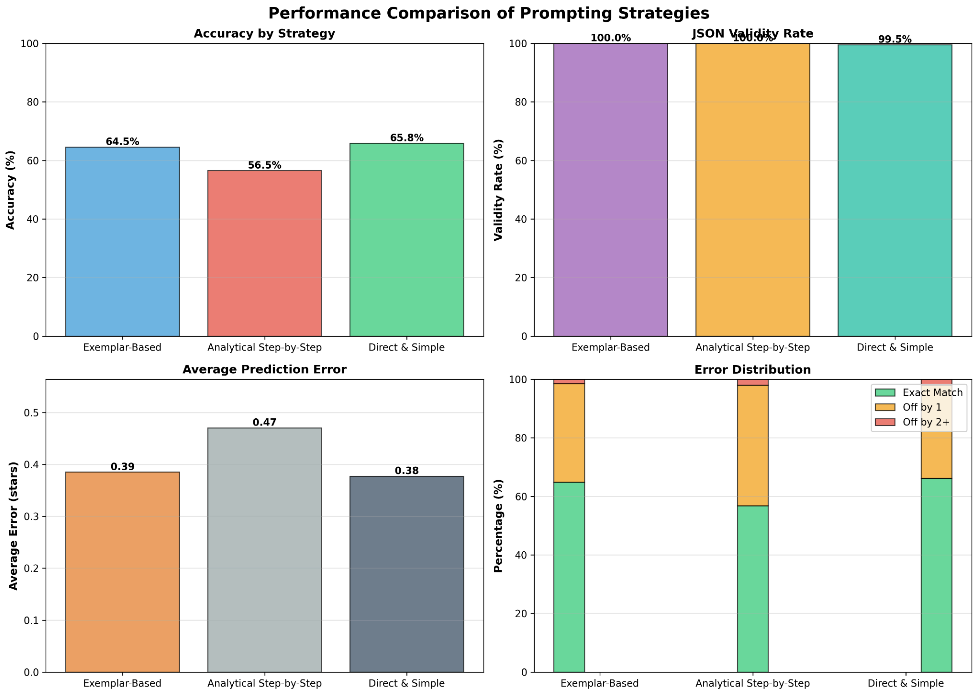
---

6. Quantitative Results

The notebook summarizes results per strategy in a table with validity, accuracy, and error metrics computed on the 200-review sample.[file:1]

6.1 Strategy-Level Metrics

Strategy	Valid Responses	Validity Rate	Accuracy	Avg Error	Output File (example)
Exemplar-Based	200/200	100.0%	64.5%	0.39	`results_exemplar-based_20260107_015330.json`
Analytical Step-by-Step	200/200	100.0%	56.5%	0.47	`results_analytical_step-by-step_20260107_015842.json`
Direct & Simple	199/200	99.5%	65.8%	0.38	`results_direct_&_simple_20260107_020216.json`



Key observations:

- JSON validity is essentially perfect for Exemplar-Based and Analytical prompts and only slightly below 100% for Direct & Simple.[file:1]
- Direct & Simple achieves the highest accuracy (~65.8%) and the lowest average error, despite being the simplest prompt.[file:1]
- Exemplar-Based is only slightly behind Direct & Simple in accuracy, with strong validity and good calibration.[file:1]
- Analytical Step-by-Step trades some accuracy for more verbose reasoning, but still remains competitive.[file:1]

---

7. Discussion and Design Choices

7.1 Why `gemma2:2b` Worked Best for This Task

From the quick 10-sample sanity check on multiple local models, `gemma2:2b` consistently produced more accurate ratings (around 80% correct) compared to the 60–70% range from other models like `codellama:7b`, `gemma:2b`, and `llava-phi3:3.8b`.[file:1] Combined with its smaller size (1.6 GB) and responsive inference speed, this made it the most practical and efficient choice for running 200×3 prompts locally.[file:1]

7.2 Effectiveness of the Hybrid Prompting Style

The hybrid design reflects a personal prompting philosophy:

- Use exemplars when the model needs grounding.
- Use step-by-step analysis when interpretability matters.
- Use direct prompts when seeking raw performance and speed.[file:1]

The quantitative results confirm that:

- Simple prompts can perform surprisingly well when the model already has strong sentiment understanding.

- Exemplar prompts provide a safety net and often yield more stable behavior across diverse reviews.
- Analytical prompts are useful when traceable reasoning is preferred, even if headline accuracy slightly drops.[file:1]

---

8. Reproducibility and Extensions

- The notebook logs configuration (model name, number of reviews) and writes per-strategy outputs to timestamped JSON files, which makes it easy to reproduce and compare runs.[file:1]
- The architecture is modular: replacing `gemma2:2b` with another Ollama model or adding a fourth prompting style only requires small changes to configuration and prompt templates.[file:1]

Possible extensions:

- Evaluate on a larger sample (e.g., 1,000+ reviews) to reduce variance in accuracy estimates.
- Add calibration plots (e.g., mapping predicted ratings to empirical error) using the saved JSON outputs.
- Experiment with temperature and other decoding parameters for `gemma2:2b` to see if stability or accuracy improves further.[file:1]