LLM-Based Semantic Re-Ranking Report

Author: Brandon Lewis
Date: October 21, 2025
Github repo: https://github.com/BranPLewis/LLM_reranking

1. Project Overview

This project aimed to evaluate the effectiveness of a Large Language Model (LLM) in improving the relevance ranking of search results. The core task was to build a system that uses an LLM to assign a relevance score to query-candidate pairs and then re-rank the candidates based on this score. The performance of this new ranking was then compared against a provided baseline ranking using standard information retrieval metrics.

2. Prompt Design and Methodology

2.1. Prompt Design

To elicit a structured, numeric judgment from the LLM, a simple and direct prompt was designed. The primary goal was to minimize variability and ensure the model's output could be easily and reliably parsed into a numeric score.

The prompt used for each API call was:

Query: {query_text}\n
Candidate passage: {candidate_text}\n
Give me a rating from the candidate passage to the query on a scale from 1 (not relevant) to 5 (highly relevant). Respond only with a single number.

Reasoning:

- **Clear Inputs:** The Query and Candidate passage are clearly delineated.
- **Explicit Instructions:** The final paragraph specifies the 0-5 scale and, most importantly, instructs the model to "Respond only with a single number." This constraint was crucial for automating the parsing of the llm_score.

2.2. Implementation Methodology

The process was implemented in a Python script (rerank.py) and followed these steps:

1. **Data Loading:** The rag_sample_queries_candidates.csv dataset was loaded into a pandas DataFrame.
2. **API Integration:** For each row in the DataFrame, the script made an API call to an LLM, sending the prompt with the corresponding query and candidate text.
3. **Scoring and Latency:** The numeric response from the API was parsed and stored as llm_score. The time taken for each API call (latency) was recorded.

4. **Re-Ranking:** After all scores were collected, a new llm_rank was generated for each query group by sorting the candidates based on the llm_score in descending order.
5. **Evaluation:** The performance of both the original baseline_rank and the new llm_rank was calculated using Precision@3, Recall@3, and nDCG@3, with the gold_label serving as the ground truth.

3. Results and Analysis

3.1. Performance Evaluation

The LLM-based re-ranking demonstrated a significant improvement over the baseline across all measured metrics. The average metrics calculated across all queries are summarized in the table below.

Metric	**Baseline Score**	**LLM Score**	**Improvement**
Precision@3	0.2000	0.5667	**+183.3%**
Recall@3	0.2333	0.7500	**+221.5%**
nDCG@3	0.2195	0.7305	**+232.8%**

Discussion of Results:
What I notice from these results is after querying relevance into a well used LLM produces an improved score over the original gold_label score of the raw csv file. The baseline score is a simple mathematical comparison, finding similar values in the query and the candidate text in order to measure relevance. Sending the query and the candidate text through an LLM gets much closer to measuring actual text and language. The purpose of this assignment was to use an LLM to assign it to be an actual human evaluator for similarity, or something very close to. The scores presented right above this do well to show just how much closer to a real human evaluator of text and language gemini_2.5_fast came.

3.2. Latency and Feasibility Analysis

While effective, the latency of the API calls is a critical factor for real-world application.

- **Total Processing Time:** 381.83 seconds (~6.4 minutes)
- **Average Time Per Query:** 3.71 seconds

An average latency of **3.71 seconds** to evaluate a single candidate passage is prohibitively high for any real-time, user-facing search application where response times are expected to be in milliseconds. This demonstrates the trade-off between the high accuracy of modern LLMs and their computational cost.

I did not compare different LLM's in order to measure latency and drop-off difference, but the average time for each query listed above is a longer amount of time then the average person can type out a google search prompt. It is worth noting that the average human reading speed is far faster than the overall response time for the LLM. The LLM measured in this project cannot out evaluate the average human.

4. Conclusion and Future Work

This experiment successfully demonstrated that an LLM can dramatically improve search relevance ranking over a baseline embedding model. The semantic capabilities of the model led to an over 200% improvement in key metrics like Recall and nDCG.

However, the high latency of this approach makes it impractical for production-level, real-time search.

Future Work:

- **Model Exploration:** Test smaller, faster, or fine-tuned models to see if a better balance between accuracy and latency can be achieved.
- **Prompt Engineering:** Experiment with few-shot prompting to potentially improve consistency and reduce variance in scores.
- **Batching:** Investigate if sending multiple requests in parallel batches could improve the overall throughput of the system.