Oct 21 2025
Austin Espinoza

LLM ReRanking

My query

“””

You are evaluating search results.

For the query below, rate EACH candidate strictly on topical relevance on a 0-1 scale
(0 = not relevant, 1 = highly relevant). Judge each candidate independently.

Return ONLY a minified JSON object mapping candidate_id -> score (float), e.g.:
{"a":0.5,"b":1.0}
No ```json```

<query>
{QUERY_TEXT}
</query>

<candidates>
{CANDIDATE_BLOCK}
</candidates>

“””

Variants tested (and why I reverted)

- **0–100 integer scale**: I tried widening the scale to reduce ties, but in practice it **did not improve** my metrics on this dataset. It also increased token usage per call and caused me to **hit rate limits**.
  Mitigation: I added a **5-second sleep after each batch** (one query) to avoid rate-limit errors.

- **0–5 scale** and **explanations + score**: these either produced more ties or added parsing headaches without clear gains.

- **No code fences**: I explicitly told the model to avoid ```json fences to keep parsing simple.

Evaluation (averages)

- **Baseline** (given):
  `precision@3 = 0.633, recall@3 = 0.842, nDCG@3 = 0.873`

- **My LLM (0–1 scale)**:
  `precision@3 = 0.317, recall@3 = 0.392, nDCG@3 = 0.321`

Interpretation. My LLM re-ranking underperformed the embedding baseline. Likely reasons: (1) the model over-scored generic ML passages that were tangential to the query, (2) batch effects still produced clustered scores despite the 0–1 guidance, and (3) the baseline embeddings were already strong for these queries. The 0–100 trial didn't help and increased latency/rate-limit friction, so I reverted to 0–1 with a cleaner prompt.

Costs & latency

Batching by query kept calls low, but the **0–100** trial pushed me into rate limits. I added a **5s delay per batch** and did not hit further errors. With the final 0–1 setup, latency and costs were manageable.

Failure → fix

Early outputs included ```json code fences, which broke `json.loads`. I changed the prompt to say "Return ONLY minified JSON… No ```json```,".
