

使用 DeepEval 指標優化 RAG 系統說明

阮荏浩 s1411232035 資工三一

一、實驗目的

本實驗的目標是使用 DeepEval 提供的多項評估指標，系統性地分析與優化 Retrieval-Augmented Generation (RAG) 系統的回答品質，確保模型回答：

- 不產生幻覺 (Hallucination)
- 僅根據檢索到的內容作答
- 回答與問題高度相關

二、RAG 系統架構說明

本作業所使用之 RAG 流程如下：

1. Query Rewriting

- 將使用者問題轉為較適合檢索的語句

2. Hybrid Retrieval (Qdrant)

- Dense Embedding (語意相似)
- Sparse (BM25 關鍵字)
- 使用 RRF (Reciprocal Rank Fusion) 整合結果

3. Re-ranking

- 使用 Qwen3-Reranker-0.6B
- 對候選 contexts 重新排序，取 Top-K

4. RAG Answering

- 僅使用 re-rank 後的 contexts 作答
- 若資料不足，明確回覆「參考資料沒有提到」

三、DeepEval 使用的五個評估指標

本實驗使用以下五項 DeepEval 指標進行評估：

1. Faithfulness (忠實度)

- 檢查回答是否完全來自檢索內容
- 防止模型自行編造不存在的資訊

2. Answer Relevancy (答案相關性)

- 評估回答是否直接回應使用者問題
- 避免答非所問

3. Contextual Recall (上下文召回率)

- 評估檢索到的 contexts 是否包含回答所需資訊
- 若召回不足，可能導致回答不完整

4. Contextual Precision (上下文精確度)

- 評估 contexts 是否精簡且相關
- 避免雜訊 context 影響回答品質

5. Contextual Relevancy (上下文整體相關性)

- 綜合判斷問題與 contexts 的語意一致程度

四、實驗設定

- 評估題數：5 題（作業示範用）
- 每題皆進行：
 - RAG 生成答案
 - 回傳實際使用的 contexts
 - 使用 DeepEval 計算五項指標
- 評估結果輸出至：
 - day6_HW_questions.csv / xlsx
 - deepeval_scores.csv

五、評估結果觀察

從實驗結果可觀察到：

- **Faithfulness 幾乎皆為 1.0**
 - 表示回答皆能被 contexts 支持
- **Contextual Recall 普遍偏高**
 - Hybrid Retrieval 成功取回相關資訊
- **Contextual Precision 偶爾較低**
 - 顯示部分 contexts 仍存在冗餘
- **Answer Relevancy 有題目略低**
 - 提示可再優化 re-ranking 或 top-k 設定

六、如何根據 DeepEval 優化 RAG

根據指標結果，本實驗採取以下優化策略：

- 限制 Answer 僅能使用 contexts (提升 Faithfulness)
- 調整 re-rank top-k (改善 Precision)
- 當 context 不足時，強制回覆「參考資料沒有提到」
- 減少不必要長文本 context

1 Faithfulness (忠實度)

你這份結果幾乎都是：

Faithfulness = 1.0

2 Answer Relevancy (答案相關性)

你大概落在：

0.66 ~ 1.0

3 Contextual Recall (上下文召回率)

你這欄幾乎都是：

Contextual Recall = 1.0

4 Contextual Precision (上下文精準度)

你這欄比較分散，例如：

0.5 / 0.75 / 0.88

5 Contextual Relevancy (上下文整體相關性)

你大概在：

0.2 ~ 0.5 (部分題目)

七、遇到的問題：

1) Query Rewrite: NoneType has no attribute strip

現象

- `rewrite_query(...).strip()` 直接炸掉，顯示 `NoneType`

原因

- `llm_chat()` 在某些情況回傳 `None` (例如 API 回傳格式不符、`content` 是 `None`、或你抓錯欄位)

解法

- `llm_chat()` 一律做「安全取值」：
 - 先判斷 `response` 是不是 JSON
 - 再判斷 `choices[0].message.content` 是否為字串
 - 否則走 `fallback` (例如回傳原始 `query`)

2) LLM 回傳格式不一致: LLM response format not recognized

現象

- 印出 LLM json (unexpected): { ... 'message': {'content': None, 'reasoning': ...}}

- 甚至 content 是 None，真正文字跑到 reasoning 或其他欄位

原因

- 用的服務雖然是 /v1/chat/completions，但回傳結構不完全等同 OpenAI
- 有的會把文字放在 message.content，有的會放 message.reasoning，或乾脆 content=None

解法

- 寫一個「相容多家 chat completion」的 parser：
 - 優先拿 message.content
 - 沒有就拿 message.reasoning
 - 再不行就回傳 fallback（避免整個 pipeline 掛掉）
-

3) Hybrid Retrieval : rewrite_query is not defined

現象

- cell 8 呼叫 rewrite_query(...) 直接 NameError

原因

- cell 執行順序被打亂（Jupyter 常見）
- 沒先跑定義 rewrite 的 cell，就先跑 retrieval demo

解法

- 在 notebook 結構上「強制依賴順序」：
 - 把 rewrite 定義放在 retrieval 之前
 - demo cell 只做 demo，不定義 function
-

4) Rerank : Cannot handle batch sizes > 1 if no padding token is defined

現象

- rerank 用 tokenizer 做 batch tokenize 時炸掉

原因

- Qwen3-Reranker tokenizer 沒設定 pad_token
- 一次丟多筆 pair (batch>1) 時就需要 padding，但 padding token 不存在

解法

- 明確指定 padding token (最常見做法):
 - `tokenizer.pad_token = tokenizer.eos_token`
 - 或 `tokenizer.pad_token_id = tokenizer.eos_token_id`
 - 並在 `tokenize` 時使用 `padding=True`
-

5) Rerank 分數全部變 0.0

現象

- rerank 結果印出來全是 0.0

原因 (你踩到兩種)

1. logits 是 (N, 2) (二分類), 錯把整個 list 當成分數去 round, 或取錯維度
2. 把 list/tuple 的順序當成 (score, doc), 其實是 (doc, score)

解法

- 二分類 logits 要轉成單一分數:
 - 常用: 取正類 `logits[:, 1]`
 - 或做 softmax 後取正類機率
 - 並統一輸出結構, 例如固定回 (doc, score)
-

6) RAG Answering: HTTP 404 (你換模型後特別明顯)

現象

- 404 File Not Found, 而且 URL base: `https://ws-06.huannago.com/v1`

原因

- request 的路徑可能變成:
 - `base_url` 設錯 (少了 `/chat/completions`)
 - `LLM_API_URL` 其實該是 `.../v1/chat/completions`, 但你只給了 `.../v1`

解法

- 統一規則:

- LLM_API_URL 必須是完整 endpoint (含 /chat/completions)
 - 或在程式內自動補齊 (偵測如果結尾是 /v1 就補上 /chat/completions)
-

7) DeepEval : 大量 JSONDecodeError / invalid JSON

現象

- DeepEval 抱怨 evaluation model 輸出不是 JSON
- 出現 truths / statements / verdicts key 缺失

原因

- DeepEval 的 judge model 要求「嚴格 JSON」
- LLM 常輸出：
 - 多餘說明、markdown、重複內容
 - 或 content=None → 根本沒 JSON
- 用的模型 (例如 gemma-3-27b-it) 可能更愛碎念，更難「只吐 JSON」

解法

- 最後採用的止損方案是正確的：
 - 選「更穩的 judge」或更嚴格 prompt
 - 對 judge output 做 parse/fallback (缺 key 就補最小 JSON)
 - 並且只跑 5 題驗證流程 (避免燒時間)
-