

# ADL HW3 Report

## Q1 : Retriever & Reranker Tuning

### Retriever:

- 訓練資料構建

1. Anchor (Query):

- 來源 : train.txt "rewrite" field
- 格式 : "query: " + query text (E5 model requirement)
- 總共 : 31,526 queries

2. Positive Sampling:

- 方法 : 從 qrels.txt 中擷取標籤為 1 的文字
- 格式 : "passage: " + passage text
- 驗證 : 確保所有 PID 都存在於 Corpus 中
- 統計 : 每個 query 包含 1 個 positive passage

3. Negative Sampling:

- 策略 : 從語料庫中隨機抽取
- 原因 : 避免 evidences 欄位中的標註錯誤
- 數量 : 每個 query 包含 4 個顯式負樣本
- In-batch negatives: 31 (batch\_size - 1)
- 總負樣本 : ~35 per query

- 損失函數

使用 MultipleNegativesRankingLoss (MNRL / InfoNCE Loss)

1. 公式

對於一個 batch 中的 query  $q_i$ :

$$\mathcal{L} = -\log \frac{\exp(\text{sim}(q_i, p_i^+)/\tau)}{\sum_j \exp(\text{sim}(q_i, p_j)/\tau)}$$

$q_i$  : query embedding

$p_i^+$  : positive passage embedding

$p_j$  : 所有 passages (positive + negatives)

$\text{sim}(\cdot, \cdot)$ : cosine similarity

$\tau$  : temperature (預設 0.05)

## 2. 運作原理：

- Bi-encoder architecture: Query 和 passage 分別編碼
- Cosine similarity: 使用 normalized embeddings 計算相似度
- Contrastive learning: 拉近 query 與 positive 的距離, 推遠與 negatives 的距離
- In-batch negatives: 同一 batch 內其他 queries 的 positives 自動成為當前 query 的 negatives

## 3. 優勢：

- 高效利用 batch 內的樣本 (無需額外計算)
- 隨著 batch size 增加, negative 數量增加, 效果更好
- 適合 dense retrieval 任務

## ● Hyperparameters (超參數)

### 1. Model

- Base model: intfloat/multilingual-e5-small
- Max sequence length: 512 tokens
- Embedding dimension: 384

### 2. Training

- Epochs: 5
- Batch size: 32
- Learning rate: 2e-5

- Warmup steps: 1000
- Optimizer: AdamW
- Learning rate schedule: Linear warmup + linear decay

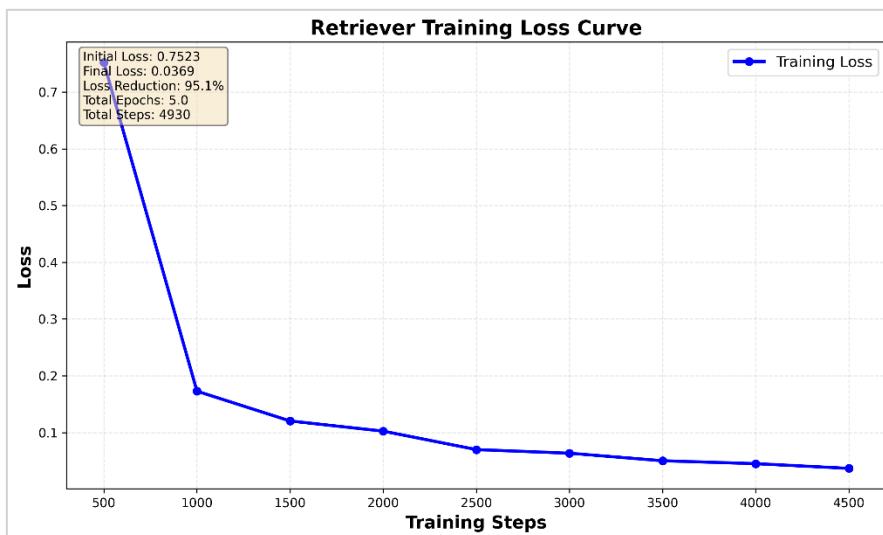
### 3. Data

- Number of negatives: 4 (explicit) + 31 (in-batch)
- Dataset size: 31526 queries

### 4. Hardware

- Device: CUDA (GPU) – A100 40GB (Google Colab)
- Mixed precision: Enabled (use\_amp=True)

- Training Loss Curves



### 1. Key Observations:

- 快速收斂：Loss 在第一個 epoch 內從 0.7523 大幅下降到 0.1729
- 穩定下降：Loss 持續穩定下降，沒有震盪現象
- 最終 loss：0.0369 (相比初始值降低 95.1%)
- 備註：最後是跑到 4930 步，但 loss 有回升至 0.14，有過擬合跡象，因此最後選擇 loss 最低的 Checkpoint 作為 best model



## 2. 完整訓練曲線顯示：

Loss 曲線平滑下降、Learning rate 採用 linear warmup (前 1000 steps) 然後 linear decay。Gradient norm 整體穩定，在 step 2000 達到最大值 5.52

- 評估結果 (Retriever Only + baseline Reranker)

Metric	Score	Baseline	Improvement
Recall@10	0.8734	0.780	+11.9%
MRR@10	0.7632	0.695	+9.8%

在微調完 Retriever 後前兩個指標就已經超越 baseline 了，證明 Retriever 微調效果良好。

## Reranker:

- 訓練資料構建
  1. Anchor (Query):
    - 來源: train.txt "rewrite" field
    - 每個 query 對應一個問題
  2. Positive Sampling:
    - 來源：透過 qrels.txt 取得每個 query 的正確 passage ID
    - 從 corpus.txt 中提取對應的 passage 文本
    - 每個 query 配對 1 個 positive passage，標籤為 1
  3. Negative Sampling:
    - 策略 1：優先使用 train.txt 中 evidences 欄位內標籤為 0 的 passages (BM25 hard negatives)
    - 策略 2：若 hard negatives 不足，從 corpus 隨機採樣補充
    - 每個 query 配對 N 個 negative passages，標籤為 0
    - 實驗中測試了每個 batch 4 個 negative
  4. 最終資料格式：
 

Dataset columns: ['query', 'passage', 'label']

    - query: 問題文本
    - passage: passage 文本
    - label: 1 (positive) 或 0 (negative)
- 損失函數
  1. 使用 Binary Cross-Entropy Loss (BCE Loss) :
    - 公式： $L = -[y \cdot \log(p) + (1-y) \cdot \log(1-p)]$
    - 其中 y 是標籤 (0 或 1)，p 是模型預測的機率
    - 適合處理 reranking 的二元分類任務
- Hyperparameters (超參數)
 

我進行了兩組實驗，以下是主要的超參數設定：

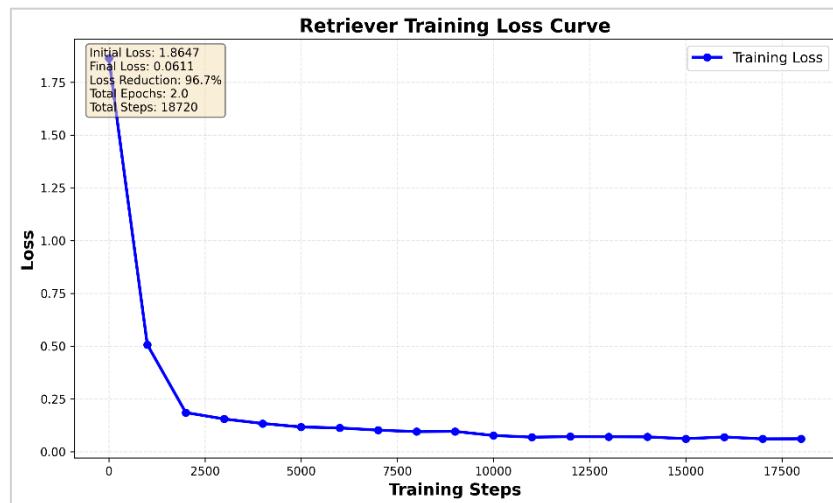
實驗	Learning Rate	Epochs	Batch Size	Neg per Query	Warmup Ratio
Exp1	2e-5	2	16	4	0.1
Exp2	5e-6	1	16	4	0.1

### 其他參數

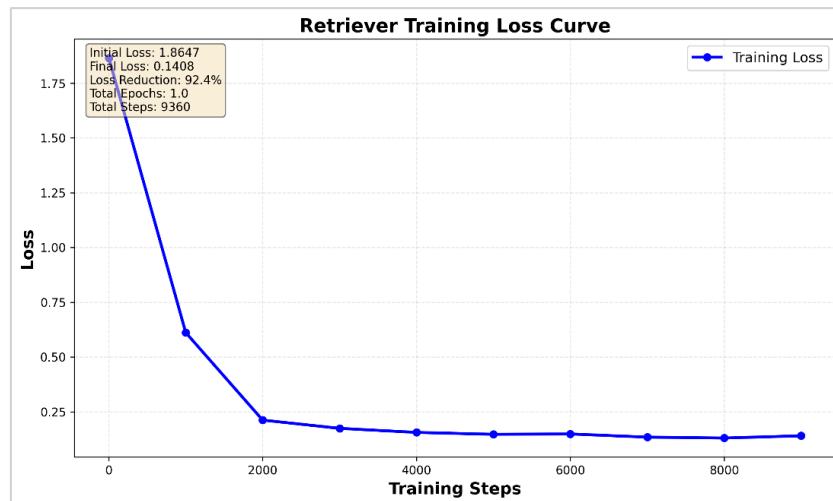
- Optimizer: AdamW
- Evaluation steps: 3000
- Mixed precision: BF16/FP16
- Base model: cross-encoder/ms-marco-MiniLM-L-12-v2

### ● Training Loss Curves

#### 1. Exp1 :



#### 2. Exp2 :



從兩次訓練的 loss 曲線可以看到模型的 loss 都有在穩定下降，但後續測試時效能卻不太如意。

會微調兩次的原因是因為使用第一次微調的 reranker 去進行推論時，發現 MRR@10 (after rerank) 比用 baseline model 還差，判斷可能是因為過擬合的關係把原先已經用 ms macro 訓練集訓練好的 model 的效能變差，於是第二次調低了學習率再微調一次以及 epoch 再訓練一次，雖然效果有比第一次微調的好，但還是遠低於原先用 cross-encoder/ms-marcos-MiniLM-L-12-v2 還要差很多，所以依據實驗結果，我決定使用 **base model** 來作為我的 **reranker**，cross-encoder/ms-marcos-MiniLM-L-12-v2 已經在大規模 ms macro 資料集上訓練過了，效能已經夠強了，且先前使用微調的 retriever 搭配 baseline reranker 進行推論就已超過 public baseline 的標準。

- 評估結果 (Retriever + Reranker)

1. 微調 retriever + baseline reranker

```
Queries evaluated: 3342
Recall@10: 0.8734
MRR@10 (after rerank): 0.7632
```

2. 微調 retriever + 第一次微調的 reranker

```
Queries evaluated: 3342
Recall@10: 0.8734
MRR@10 (after rerank): 0.2484
```

3. 微調 retriever + 第二次微調的 reranker

```
Queries evaluated: 3342
Recall@10: 0.8734
MRR@10 (after rerank): 0.3541
```

4. 結論：

經過 3 組實驗後發現，base model (cross-encoder/ms-marcos-MiniLM-L-12-v2) 在 ms macro 上的預訓練已經非常強大，在我們的小規模數據集上微調容易導致過擬合。因此最終決定直接使用 base model 作為 reranker 的提交版本。

## Q2 : Prompt Optimization

## ● Prompt 設計說明

### 1. 設計理念

我的 prompt 優化主要針對三個關鍵問題：

- **LLM 輸出格式控制**：Qwen 模型會輸出完整的對話模板，需要明確的答案提取策略
- **答案精確性**：要求 LLM 只基於 context 回答，避免幻覺
- **格式簡潔性**：使用簡單的 Q/A 格式，減少 LLM 重複 prompt 的可能性

### 2. System Prompt 設計

```
def get_inference_system_prompt() -> str:
    """get system prompt for generation"""
    prompt = "You are a precise QA assistant. Answer based only on the given context. If the answer is not in the context, say 'CANNOTANSWER'."
    return prompt
```

- 關鍵詞 "precise"：強調精確性
- "based only on the given context"：防止 LLM 使用外部知識
- 明確的失敗處理：定義 CANNOTANSWER 作為無法回答的標準輸出

### 3. User Prompt 設計

```
def get_inference_user_prompt(query : str, context_list : List[str]) -> str:
    """Create the user prompt for generation given a query and a list of context passages."""
    formatted_contexts = "\n\n".join([
        f"[{i+1}] {context}"
        for i, context in enumerate(context_list)
    ])

    prompt = f"""{formatted_contexts}

Q: {query}
A:"""

    return prompt
```

- 簡潔格式：使用最簡單的 Q/A 格式，降低 LLM 重複整個 prompt 的機率
- Context 編號：[1], [2], [3] 方便模型定位資訊來源
- 明確結束標記：A： 提示模型開始生成答案

### 4. Parse Function 設計

由於 Qwen 模型會輸出包含 <think> 標籤和完整對話格式的回應，我設計了多層次的答案提取策略：

- 策略 1：提取 `</think>` 之後的內容（針對 Qwen 的 thinking 機制）
- 策略 2：找最後一個 `assistant` 標記之後的內容
- 策略 3：找最後一個 `A:` 之後的內容
- 策略 4：取最後一行作為 fallback

這種多策略設計確保無論 LLM 輸出何種格式，都能較正確地提取答案。

評估結果：

```
Queries evaluated: 3342
Recall@10: 0.8734
MRR@10 (after rerank): 0.7632
Bi-Encoder CosSim: 0.3855
```

三項指標均已超過 public baseline。

- 實驗結果對比（上方敘述的是我最終使用的 Prompt 設計）

我進行了三組實驗，逐步優化 prompt 設計：

1. 我第一次是直接測試原本的 prompt，即完全沒改

```
def get_inference_system_prompt() -> str:
    """Get system prompt for generation"""
    prompt = ""
    return prompt

def get_inference_user_prompt(query : str, context_list : List[str]) -> str:
    """Create the user prompt for generation given a query and a list of context passages."""
    prompt = f"""
        {query}
        {context_list}
    """
    return prompt

def parse_generated_answer(pred_ans: str) -> str:
    """Extract the actual answer from the model's generated text."""
    parsed_ans = pred_ans
    return parsed_ans
```

在經過推論測試完之後，分數如下：

```
Queries evaluated: 3342
Recall@10: 0.8734
MRR@10 (after rerank): 0.7632
Bi-Encoder CosSim: 0.0345
```

Bi-Encoder CosSim 只有 0.0345，LLM 的輸出包含完整的 prompt 和對話格式，無法正確提取答案，答案品質極差，大部分輸出是 "assistant" 或完整的 prompt 內容。

## 2. 第二次優化了三個函式，但 `parse_generated_answer()` 只做了簡單的處理

```

def get_inference_system_prompt() -> str:
    """Get system prompt for generation"""
    prompt = "You are a precise QA assistant. Answer based only on the given context. If the answer is not in the context, say 'CANNOTANSWER'."
    return prompt

def get_inference_user_prompt(query : str, context_list : List[str]) -> str:
    """Create the user prompt for generation given a query and a list of context passages."""
    formatted_contexts = "\n\n".join([
        f"[{i+1}] {context}"
        for i, context in enumerate(context_list)
    ])

    prompt = f"""{formatted_contexts}

Q: {query}
A:"""

    return prompt

def parse_generated_answer(pred_ans: str) -> str:
    """
    Extract answer from LLM output.
    主要處理 LLM 輸出包含完整 prompt 的情況
    """
    pred_ans = pred_ans.strip()

    if not pred_ans:
        return "CANNOTANSWER"

    return pred_ans

```

**推論結果：**

- Recall@10: 0.89
- MRR@10: 0.76
- **Bi-Encoder CosSim: 0.3536**

在第二次調整中，System prompt 有明確指示任務，且在 User Prompt 中有給予範例供 LLM 參考，避免 LLM 隨便回答問題，在這一次的調整中 Bi-Encoder CosSim 已經超過 public baseline 的 0.340 了。

## 3. 結果比較圖

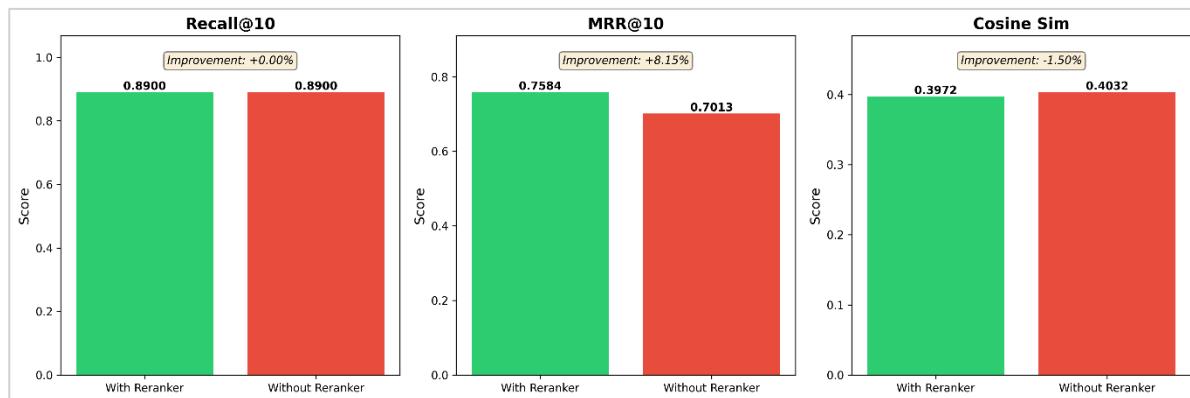
Prompt Version	System Prompt	User Prompt	Parse Function	Bi-Encoder CosSim
V1 (Baseline)	空	空	空	0.0345
V2 (Prompt 優化)	精確指示	Q/A 格式	無處理	0.3536
V3 (完整優化)	精確指示	Q/A 格式	多策略提取	0.3855
Target (Baseline)	-	-	-	0.340

#### 4. 結論

- System Prompt 的重要性：明確的任務定義 ("precise", "based only on context") 能有效引導 LLM 行為
- 簡潔格式的優勢：使用 Q: ... A: 的簡單格式比複雜的指令更有效，減少 LLM 重複 prompt 的傾向
- Parse Function 的必要性：針對 Qwen 模型的特殊輸出格式 (<think> 標籤、assistant 標記)，必須設計專門的答案提取策略
- 最終提升：透過完整的 prompt 優化，Bi-Encoder Cosine Similarity 從 0.0345 提升到 0.3855，超越 baseline (0.34) 達 13.4%。

### Q3: Additional Analysis

我針對了有 Reranker v.s. 無 Reranker 的效能對比



只使用了 100 筆樣本去做對比實驗，在 Recall 的部分沒有任何改變（因為檢索的候選集相同），MRR 提升了 **8.15%**(reranker 改善排序)，Cosine Sim 反而降低了 1.50%，有可能是因為只用了 100 筆資料測試的關係，所以效果沒有到很準確，但正常情況下，Cosine Sim 在有 reranker 時應該會比無 reranker 還好，因為 reranker 會提供更好的 context 排序。