ADL HW3 Report

## Q1: Retriever & Reranker Tuning (5%)

### Retriever Training Process (2.5%)

#### 1. Training data

The retriever model is trained using query–passage pairs derived from a labeled dataset.

**Anchor**: Each query.

**Positive sampling**: For each query, we retrieve its relevant passages as positive sample.

**Negative sampling**: Instead of explicit negatives, we use the *MultipleNegativesRankingLoss* to automatically treats other positive passages within the same batch as negative examples for the current query.

#### 2. Loss function

I use MultipleNegativesRankingLoss as the contrastive loss function.

The loss for query $i$ is:

$$L_i = -\log \frac{\exp(\mathrm{sim}(q_i, p_i)/\tau)}{\sum_j \exp(\mathrm{sim}(q_i, p_j)/\tau)}$$

where:

$sim(\cdot,\cdot)$ is cosine similarity,

$p_i$ is the positive passage,

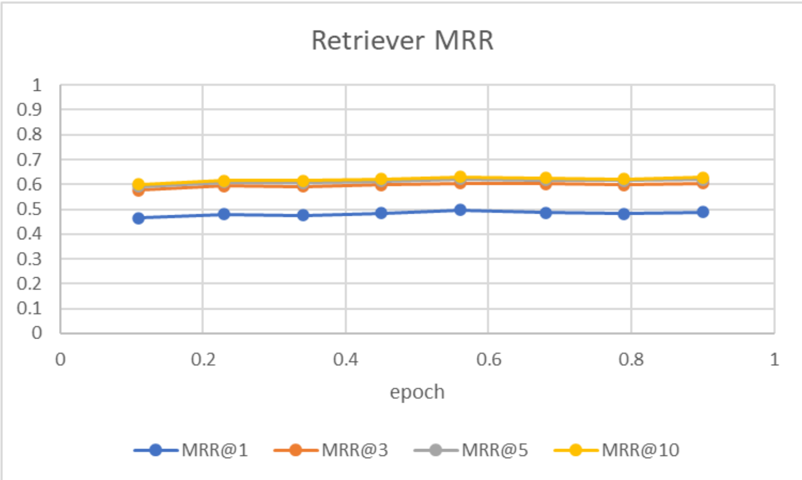
$p_j, (j \neq i)$ are in-batch negatives,

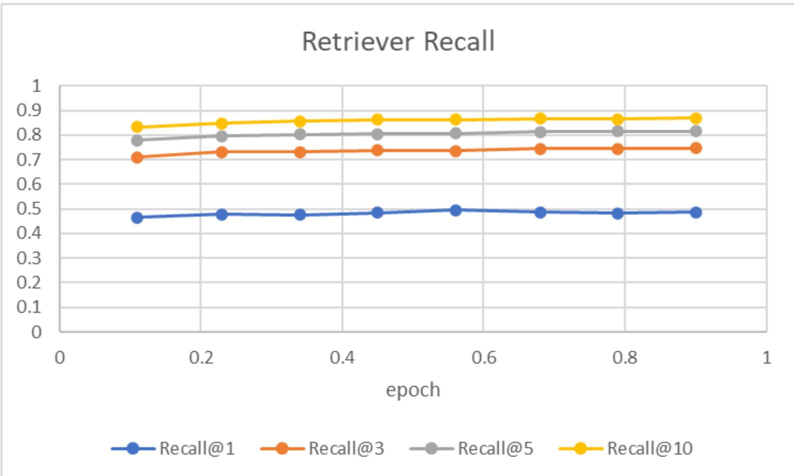
$\tau$ is a temperature hyperparameter (implicitly handled inside the implementation).

## 3. Hyperparameters

| Hyperparameter | Value |
|---|---|
| Base model | intfloat/multilingual-e5-small |
| Batch size | 32 |
| Epochs | 1 |
| Learning rate | 2e-5 (default) |
| Warmup steps | 10% of total steps |
| Evaluation steps | Every 100 steps |
| Loss function | MultipleNegativesRankingLoss |
| Optimizer | Default |
| Seed | 42 |
| Mixed precision | Enabled (use_amp=True) |

## 4. Training curve




The retriever achieves recall@10 = 0.8687 on evaluation set after 1 epoch. (0.8256 on public test set.)

**Reranker Training Process (2.5%)**

**1. Training Data**

The reranker is trained using query-passage pairs in a supervised binary classification setting.

Each training sample is created from three components:

**Anchor**: a query q (from the training queries).

**Positive sample**: one relevant passage $p^+$, determined by non-zero relevance labels in the qrels file.

**Negative samples**: one (or more) irrelevant passages $p^-$, randomly sampled from the corpus while excluding all known relevant passages.

**2. Loss Function**

The reranker uses the Cross-Entropy Loss implemented internally in the Sentence Transformers CrossEncoder class.

Given the query-passage pair, the model outputs a single relevance score in $[0, 1]$, and the loss encourages correct binary classification:

$$\mathcal{L} = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

where:

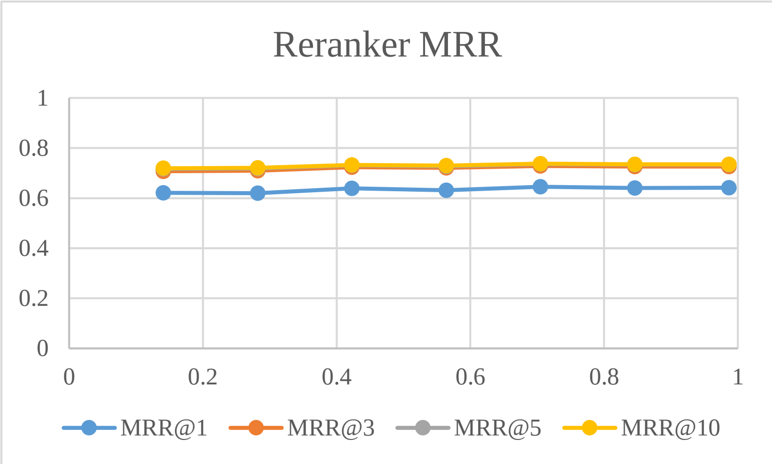
$y \in \{0, 1\}$ is the true label (relevant / non-relevant),

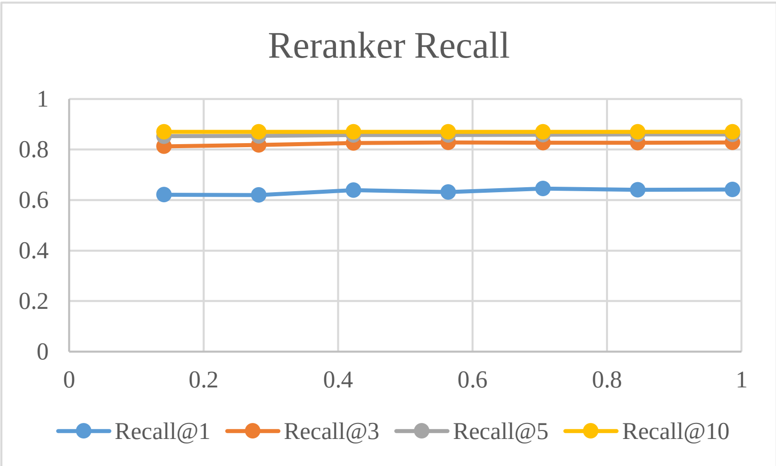
$\hat{y}$ is the model's predicted probability.

**3. Hyperparameters**

| Parameter | Value |
|---|---|
| Base model | cross-encoder/ms-marco-MiniLM-L-12-v2 |
| Lose function | Cross-Entropy Loss |
| train_epochs | 1 |
| train_batch_size | 8 |
| learning_rate | 2e-5 |
| warmup_steps | 100 |
| num_neg_per_pos | 1 |
| max_len | 512 |
| eval_steps | 1000 |
| top_k | 10 |
| eval_batch_q | 16 |
| seed | 42 |

**4. Training curve**




MRR@10 rises from 0.6275 (without reranker) to 0.7375 (with reranker) on evaluation set. (0.7302 on public test set.)

**Q2: Prompt Optimization (3%)**

**Provide a detailed explanation of how you designed your prompt. (1.5%)**

I tried three methods: basic, CoT, confidence-base.

**Method 1**: *"Answer the user concisely..."* enforces short, direct responses.

**Method 2**: *"Explain your reasoning briefly..."* encourages chain-of-thought.

**Method 3**: *"Provide an answer only if the context clearly supports it; otherwise, respond with 'CANNOTANSWER'."* tells model how to handle situations when there's no answer.

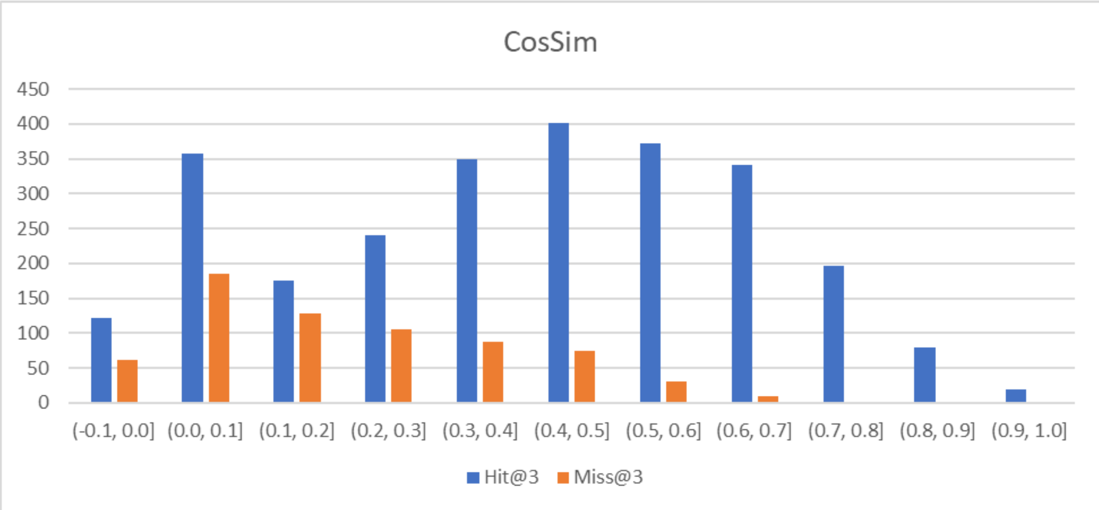
**Present at least three different prompts you experimented with. (1.5%)**

<table>
<tr><th>Method</th><th>Prompt</th><th>CosSim</th></tr>
<tr><td>1<br>Basic</td><td><pre><code>def get_inference_system_prompt():
    return "Answer the user concisely based on the context passages."
def get_inference_user_prompt(query, context_list):
    return f"Question: {query}\n\nContext:\n" + "\n\n".join(context_list)</code></pre></td><td>0.3660</td></tr>
<tr><td>2<br>CoT</td><td><pre><code>def get_inference_system_prompt2():
    return "You are an assistant that answers questions using evidence from the given passages. Explain your reasoning briefly before giving the final answer."
def get_inference_user_prompt2(query, context_list):
    return f"Question: {query}\n\nRelevant Passages:\n" + "\n---\n".join(context_list) + "\n\nPlease reason step by step and conclude with a short final answer."</code></pre></td><td>0.3397</td></tr>
<tr><td>3<br>confidence base</td><td><pre><code>def get_inference_system_prompt3():
    return "Provide an answer only if the context clearly supports it; otherwise, respond with 'CANNOTANSWER'."
def get_inference_user_prompt3(query, context_list):
    return f"Question: {query}\n\nSupporting Contexts:\n" + "\n\n".join(context_list)</code></pre></td><td>0.3572</td></tr>
</table>

Basic method is the best; CoT takes long time when inferencing; Teach model “CANNOTANSWER” does not help much.

**Q3: Additional Analysis (2%)**

I analyzed the distribution of CosSim on all public test data. Furthermore, I separate them into hit@3 and miss@3, in the other words, correct paragraph was sent to LLM or not. Unsurprisingly, CosSim of hit@3 is higher then miss@3. Note that no one is below -0.1.



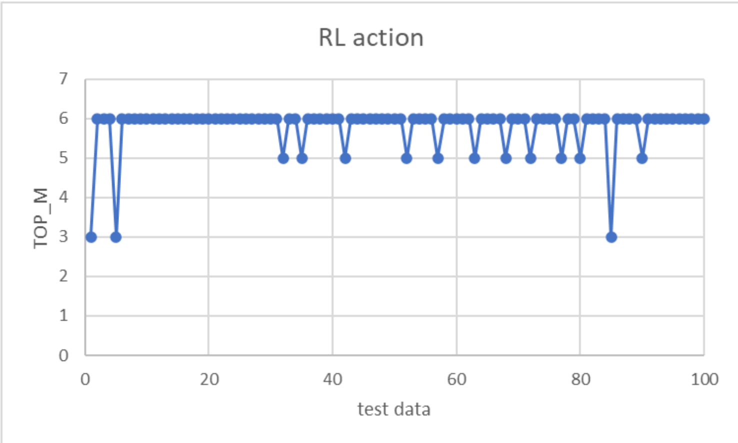

| | **hit@3** | **miss@3** | **total** |
|---|---|---|---|
| count | 2658 | 684 | 3342 |
| mean | 0.4000 | 0.2046 | 0.3600 |
| std | 0.2455 | 0.1747 | 0.2458 |

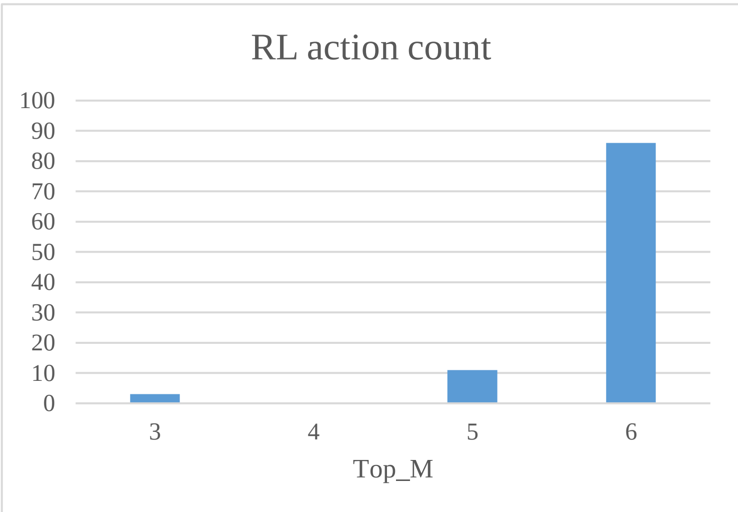
**Bonus: RL in the loop (2%)**

I adopted the A2C (Advantage Actor-Critic) algorithm from stable-baselines3, and trained the agent in a custom environment where the reward was based on the Cosine Similarity between the generated and gold answers.

The learned action distribution showed that the model preferred larger context sizes, top_m = 6 appears most times.

If top_m is too low, the correct paragraphs may not be sent to LLM. If top_m is too high, too many irrelevant paragraphs will be included. Dynamically adjusting the number of passages helps the LLM achieved higher performance. (CosSim 0.3660 → 0.3910)



## Note

All trainings are on workstation with 1 × RTX A6000 GPU.

## References


1. ChatGPT-5
2. Gemini 2.5 Pro
3. Wang, L., Yang, N., Wang, X., Joty, S., & Lin, J. (2022). Text Embeddings by Reranking. arXiv preprint arXiv:2212.03534.
4. Reimers, Nils, and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks." arXiv preprint arXiv:1908.10084 (2019).
5. Karpukhin, Vladimir, et al. "Dense Passage Retrieval for Open-Domain Question Answering." EMNLP (1). 2020.
6. Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International conference on machine learning. PmLR, 2016.