Fine-Tuning LLaMA for Multi-Stage Text Retrieval

Xueguang Ma∗
University of Waterloo
Waterloo, Canada
x93ma@uwaterloo.ca

Liang Wang
Microsoft Research Asia
Beijing, China
wangliang@microsoft.com

Nan Yang
Microsoft Research Asia
Beijing, China
nanya@microsoft.com

Furu Wei
Microsoft Research Asia
Beijing, China
fuwei@microsoft.com

Jimmy Lin
University of Waterloo
Waterloo, Canada
jimmylin@uwaterloo.ca

1 INTRODUCTION

Text retrieval is crucial in various natural language comprehension tasks [25], including web search [1], open-domain question answering [2], and fact verification [34]. Retrieval also plays an important role in enhancing the effectiveness of large language models (LLMs) in a retrieval-extended generation (RAG) pipeline [15, 31]. This approach not only mitigates hallucinations but also enables LLMs to access external knowledge [12, 42].

A typical multi-stage text retrieval pipeline consists of a retriever, designed to efficiently locate the top-k relevant texts from a (potentially large) corpus, and a reranker, which further refines the order of the retrieved candidates to improve output quality [22]. Both retrievers and rerankers have significantly benefited from the advent of pre-trained language models based on Transformers [37] such as BERT [7] and T5 [30]. These models are fine-tuned to encode queries and documents into vector representations for retrieval [13, 16] or to directly score the relevance between a query and a document for reranking [23]. Several solutions have been introduced to enhance the effectiveness of retrievers and rerankers with improved data creation or training strategies [9, 29, 38, 40, 41, 44].

Recent LLMs with billions of parameters such as GPT-4 [24] and LLaMA [35, 36] have exhibited extraordinary capabilities in many NLP tasks, surpassing previous smaller models [43]. For retrieval, recent methods such as RankGPT [32], LRL [18], and PRP [28] have explored prompting LLMs to perform zero-shot listwise or pairwise ranking as text generation tasks. Work such as HyDE [10] and Query2Doc [39] have used LLMs to augment user queries. These methods rely on the strong generative capabilities of LLMs. However, we see a few potential issues. First, these methods do not address the entire multi-stage pipeline, as it is challenging to cast first-stage retrieval as text generation. Second, they do not leverage labeled data when available. Finally, prompting-based rerankers are not efficient because they do not support parallel scoring; also, their multi-pass decoding design and sliding window strategy [18, 32] present efficiency bottlenecks.

Therefore, we argue that fine-tuning state-of-the-art LLMs to function as retrievers and rerankers in multi-stage pipelines can yield better effectiveness than older, smaller models. Previous work such as GTR [21], SGPT [19], and cpt-text [20] discussed fine-tuning language models with billions of parameters to generate dense embeddings. However, LLaMA has demonstrated even better effectiveness on natural language generation tasks than previous open-source models, suggesting that it might serve as a better backbone.

ABSTRACT

While large language models (LLMs) have shown impressive NLP capabilities, existing IR applications mainly focus on prompting LLMs to generate query expansions or generating permutations for listwise reranking. In this study, we leverage LLMs directly to serve as components in the widely used multi-stage text ranking pipeline. Specifically, we fine-tune the open-source LLaMA-2 model as a dense retriever (repLLaMA) and a pointwise reranker (rankLLaMA). This is performed for both passage and document retrieval tasks using the MS MARCO training data. Our study shows that fine-tuned LLM retrieval models outperform smaller models. They are more effective and exhibit greater generalizability, requiring only a straightforward training strategy. Moreover, our pipeline allows for the fine-tuning of LLMs at each stage of a multi-stage retrieval pipeline. This demonstrates the strong potential for optimizing LLMs to enhance a variety of retrieval tasks. Furthermore, as LLMs are naturally pre-trained with longer contexts, they can directly represent longer documents. This eliminates the need for heuristic segmenting and pooling strategies to rank long documents. On the MS MARCO and BEIR datasets, our repLLaMA–rankLLaMA pipeline demonstrates a high level of effectiveness.

CCS CONCEPTS

• Information systems → Retrieval models and ranking.

KEYWORDS

Large Language Model, Dense Retrieval, Reranker

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGIR ’24, July 14–18, 2024, Washington, DC, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0431-4/24/07
https://doi.org/10.1145/3626772.3657951

∗Work done during Xueguang’s internship at MSRA.
and warranting further exploration. Additionally, none of the models referenced above are fully optimized for a multi-stage retrieval pipeline. Thus, we investigate the following research question: How do state-of-the-art LLMs perform when specifically fine-tuned for multi-stage text retrieval?

Our study answers this question by conducting a comprehensive investigation into fine-tuning LLaMA-2 [35] as both a dense retriever and a pointwise reranker, which we refer to as repLLaMA and rankLLaMA, respectively. We find that LLMs surpass previous smaller models in terms of effectiveness for both retrieval and reranking using only a straightforward training regime and exhibiting strong zero-shot effectiveness. Compared to methods that directly prompt large language models to generate permutations [32], fine-tuning large language models as rerankers can be more effective due to the use of labeled data and be more efficient due to decoupled parallel inference. Furthermore, we observe that LLMs, which are inherently pre-trained on longer contexts, are capable of representing entire documents, thereby eliminating the need for segmenting and pooling strategies for document retrieval.

2 METHOD

2.1 Retriever

Our retriever model, called repLLaMA, follows the bi-encoder architecture proposed in DPR [13], but with the backbone model initialized with LLaMA. In contrast to previous dense retriever models based on BERT, which take the representation of the prepended [CLS] token as the representation of the text input, repLLaMA computes the vector embedding of a query or a document as:

$$V_{Q} = \text{Decoder}(Q)$$

where Decoder(·) represents the LLaMA model, which returns the last layer token representation for each input token. As the attention mechanism of LLaMA is uni-directional, we take the representation of the last token as the representation of the input sequence $t_1 \ldots t_k$, either a query $Q$ or a document $D$.

Relevance of $D$ to $Q$ is computed in terms of the dot product of their corresponding dense representations $V_Q$ and $V_D$ as $\text{Sim}(Q, D) = \langle V_Q, V_D \rangle$. The model is then optimized end-to-end according to InfoNCE loss with a set of negative documents that includes both hard negatives, which are sampled from the top-ranking results of an existing retrieval system, and in-batch negatives, which are derived from the positive documents and hard negative documents associated with other queries in the same training batch. In practice, dense retrieval training tends to benefit from a larger set of hard negatives and in-batch negatives.

2.2 Reranker

Our rankLLaMA reranker model is trained as a pointwise reranker. This approach involves passing a query and a candidate document together as model input, with the model generating a score that indicates the relevance of the document to the query [23]. In more detail, rankLLaMA reranks a query–document pair as follows:

$$\text{input} = \{'\text{query: }[Q], \text{document: }[D]\}'$$

$$\text{Sim}(Q, D) = \text{Linear}((\text{Decoder}(\text{input}))[-1])$$

where Linear(·) is a linear projection layer that projects the last layer representation of the last token to a scalar. Similar to the retriever, the model is optimized by contrastive loss, but, in this case, the negative documents do not involve in-batch negatives.

To train a reranker that is optimized to rerank candidates from a specific retriever in a multi-stage pipeline, hard negatives should be sampled from the top-ranking results from that retriever. Specifically, in our case, the hard negative training data for rankLLaMA are selected from the top-ranking results of repLLaMA.

3 EXPERIMENTS

3.1 Passage Retrieval

3.1.1 Dataset. We train our retriever and reranker models with LLaMA-2 on the training split of MS MARCO passage [1]. As discussed in Section 2.1, the use of hard negatives is crucial for effective training. We use a blend of BM25 and CoCondenser [9] hard negatives to ensure that the hard negatives are derived from both sparse and dense retrieval results, thereby enhancing the diversity of the samples. For the reranker, we select the hard negatives from the top-200 candidates generated by the retriever.

We evaluate the effectiveness of our models using the development split of the MS MARCO passage ranking task and the TREC DL19/DL20 passage ranking tasks [3, 4]. Following standard practice, we adopt MRR@10 and nDCG@10 as the evaluation metrics. In addition, we assess the zero-shot effectiveness of repLLaMA and rankLLaMA on the 13 publicly available datasets of BEIR [33].

3.1.2 Implementation Details. We initialize our models with the LLaMA-2-7B checkpoint\footnote{https://huggingface.co/meta-llama/Llama-2-13b-hf} and train on 16 × 32G V100 GPUs. For repLLaMA, we append an end-of-sequence token $/s/$ to the input sequence and take its final layer representation as the dense representation (4096 dimensions). Additionally, we normalize these dense representations into unit vectors during both the training and inference stages, ensuring that their L2-norms are equal to 1. After encoding the entire corpus, we end up with a 135G flat index for brute-force search.

For rankLLaMA, we find that appending $/s/$ to the input sequence causes loss overflow error when fine-tuning LLaMA-2 with 16-bit floating point precision. Thus, we use the final layer representation of the last token in the passage to calculate the similarity score. A challenge in fine-tuning LLMs for retrieval is the high GPU memory costs associated with contrastive learning, as it requires large batch sizes for in-batch negatives. To address this, we employ recently proposed solutions, including LoRA [11], flash attention [6], and gradient checkpointing to reduce GPU memory usage. Both the retriever and reranker are trained with a batch size of 128, with 15 hard negative passages sampled for each query. At inference time, repLLaMA retrieves the top-1000 passages from the corpus and rankLLaMA reranks the top-200 passages retrieved by repLLaMA. To explore whether increases in model size can further improve effectiveness, we also train a version of rankLLaMA using LLaMA-2-13B as the model initialization.\footnote{https://huggingface.co/meta-llama/Llama-2-13b-hf}

3.1.3 In-Domain Evaluation. As shown in Table 1, repLLaMA outperforms all competing methods for retrieval, achieving the highest
Table 1: The effectiveness of repLLaMA and rankLLaMA on the MS MARCO passage corpus compared to baselines.

| Model | Source prev. top-k | Dev DL19 | DL20 MRR@10 nDCG@10 nDCG@10 |
|-------|-------------------|----------|-------------------|-------------------|
| BM25  [17] | -                  | 18.4     | 50.6              | 48.0              |
| ANCE  [41] | 125M               | 30.0     | 64.5              | 64.6              |
| CoCondenser [9] | 110M               | 38.2     | 71.7              | 68.4              |
| GTR-base [21] | 110M               | 36.6     | -                 | -                 |
| GTR-XXL [21] | 4.8B               | 38.8     | -                 | -                 |
| OpenAI Ada2 [20] | 2                 | 34.4     | 70.4              | 67.6              |
| bi-SimLM [38] | 110M               | 39.1     | 69.8              | 69.2              |
| repLLaMA | 7B                 | 41.2     | 74.3              | 72.1              |
| monoBERT [23] | 110M               | (a) 1000 | 37.2              | 72.3              |
| cross-SimLM [38] | 110M               | (g) 200  | 43.7              | 74.6              |
| RankT5 [44] | 220M               | (d) 1000 | 43.4              | -                 |
| rankLLaMA | 7B                 | (h) 200  | 44.9              | 75.6              |
| rankLLaMA-13B | 13B                | (i) 200  | 45.2              | 76.0              |
| nRankVicuna [27] | 7B                 | (a) 100  | -                 | 66.8              |
| PRP [28] | 20B                | (b) 100  | -                 | 72.7              |
| RankGPT3.5 [32] | ?                  | (c) 100  | -                 | 65.8              |
| RankGPT3 [32] | ?                  | (p) 30   | -                 | 75.6              |

Table 2: Zero-shot effectiveness of repLLaMA and rankLLaMA on BEIR datasets.

<table>
<thead>
<tr>
<th>Method</th>
<th>Model Size</th>
<th>BEIR-13 Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM25 [17]</td>
<td>-</td>
<td>43.7</td>
</tr>
<tr>
<td>GTR-XXL [21]</td>
<td>4.8B</td>
<td>49.3</td>
</tr>
<tr>
<td>Ada2 [20]</td>
<td>?</td>
<td>52.1</td>
</tr>
<tr>
<td>SGPT [19]</td>
<td>5.8B</td>
<td>52.1</td>
</tr>
<tr>
<td>repLLaMA</td>
<td>7B</td>
<td>55.1</td>
</tr>
<tr>
<td>RankT5 [44]</td>
<td>220M</td>
<td>53.7</td>
</tr>
<tr>
<td>rankLLaMA</td>
<td>7B</td>
<td>56.6</td>
</tr>
<tr>
<td>rankLLaMA-13B</td>
<td>13B</td>
<td>56.5</td>
</tr>
</tbody>
</table>

3.2.2 Implementation Details. By default we allow the retriever and reranker to take the first 2048 tokens as input without any segmentation, which is a reasonable trade-off between input sequence length and the cost of training. This approach covers about 77% of the documents in the corpus entirely. We create the training data for the document retriever and reranker models based on the 300k training examples in the training set. Similar to the approach for passage ranking, we sample the hard negative documents to train repLLaMA from the top-100 hard negatives from BM25 and our implementation of CoCondenser on the MS MARCO document retrieval training data. Here, BM25 directly indexes the complete documents, while CoCondenser retrieves documents using the aforementioned MaxP strategy. The hard negatives for rankLLaMA are selected from the top-100 results of repLLaMA.
We see that full fine-tuning achieves a much higher MRR@100 score when reranking different top-1000 retriever outputs. This approach surpasses the effectiveness of smaller models, handles longer texts, and highlights the potential for enhancing text retrieval with LLMs.

### 4 ABLATIONS AND ANALYSES

**Full Fine-Tuning vs. LoRA.** When fine-tuning LLMs, a key decision is whether to perform “full” fine-tuning or to use a parameter-efficient method such as LoRA. Table 4 compares the effectiveness of these two approaches on repLLaMA for the passage retrieval task. We see that full fine-tuning achieves a much higher MRR@10 score than LoRA on the training set, however, this improvement does not translate over to the development set. Interestingly, on the TREC DL19/DL20 datasets, which are derived from independent human judgments, LoRA demonstrates better effectiveness. This suggests that full fine-tuning may be prone to overfitting on the training set distribution, while LoRA, with significantly fewer parameters, can generalize better. For this reason, all the models presented in our main experiments (Section 3) use LoRA.

**Representation Dimensionality.** By default, repLLaMA generates 4096 dimensional vectors. We trained a variant of repLLaMA following the Matryoshka Representation Learning (MRL) training strategy [14], which enables repLLaMA to generate representations with flexible dimensionality. As shown in Table 5, the effectiveness of repLLaMA decreases with smaller vectors, but the effectiveness is largely preserved, especially for in-domain evaluation. We observe gradual degradation of effectiveness on BEIR, indicating that repLLaMA has the flexibility to adapt to constrained vector sizes.

**Reranking Candidates.** In Table 6, we show the effectiveness of using rankLLaMA to rerank candidate from retrievers other than repLLaMA. To match the original setting of monoT5 and RankT5, we rerank the top-1000 retriever outputs. Our rankLLaMA outperforms both monoT5 and RankT5 “out of the box”, showing the effectiveness of rankLLaMA without relying on repLLaMA.

**Input Sequence Length.** We investigate the effects of varying the maximum training input length and inference input length on model effectiveness for the document reranking task. Results presented in Figure 1 show a clear trend: the effectiveness of rankLLaMA improves as the maximum training length increases from 512 to 2048, with the MRR@100 score improving from 48.5 to 50.3. When the reranking input length is further increased to 4096, the MRR@100 score rises to 50.6. This demonstrates the model’s ability to exploit longer sequences for improved effectiveness.

However, we note that the gains plateau beyond a certain length, suggesting a point of diminishing returns. The MRR@100 for the model trained with a length of 4096 is only 0.3 points higher than the model trained with a length of 2048, when evaluated on input lengths that match their training lengths. Moreover, the model trained with a length of 4096 takes about 8 days to train using 16 × V100 GPUs, while the model with a length of 2048 takes about 4 days. The same relative latency costs apply to inference as well. Therefore, while rankLLaMA can handle much longer input documents, it is crucial to balance this capability with the practical considerations of computational efficiency.

### 5 CONCLUSION

Our study shows that large language models (LLMs) can be effectively fine-tuned to function as dense retrievers and pointwise rerankers, establishing a point of reference for future multi-stage retrieval systems. This approach surpasses the effectiveness of smaller models, handles longer texts, and highlights the potential for enhancing text retrieval with LLMs.