



CSCE 670 - Information Storage and Retrieval

Lecture 23: Large Language Models for Ranking

Yu Zhang

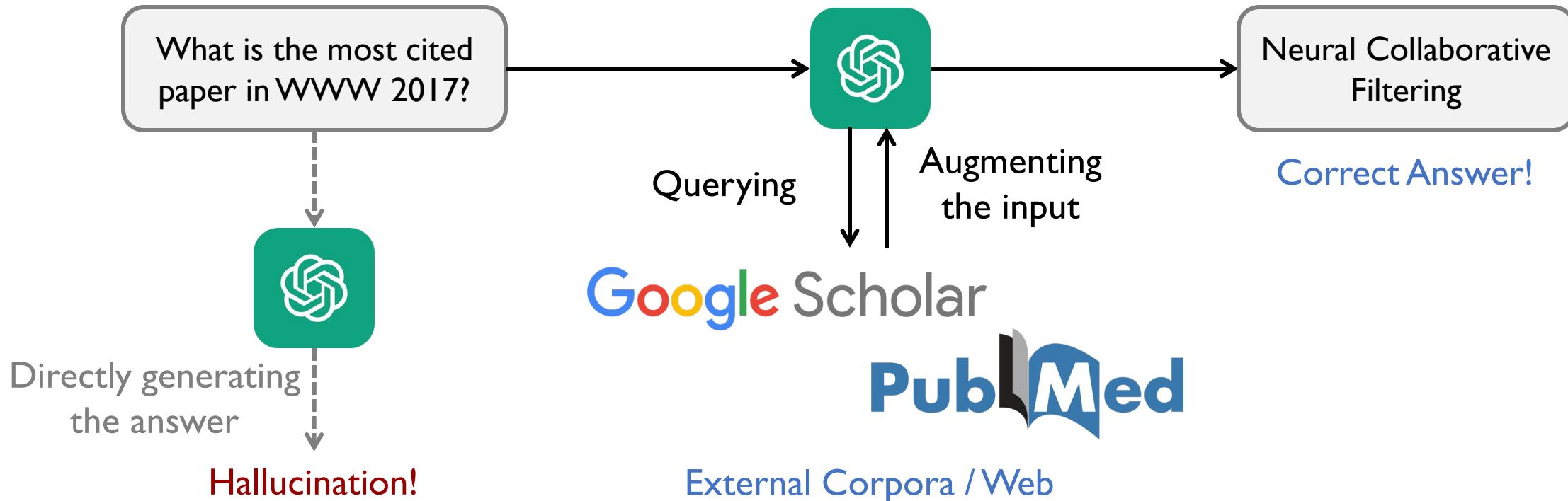
yuzhang@tamu.edu

November 13, 2025

Course Website: <https://yuzhang-teaching.github.io/CSCE670-F25.html>

Recap: How do search engines help LLMs?

- Retrieval-Augmented Generation (RAG)
- Reasoning-Search Interleaved LLMs (Search-RI)



Today: How do LLMs help search engines?

- **Initial Idea:** Feed the query and all candidate documents to the LLM, and let it output the ranking
- Is this strategy generalizable?

Based on the query below, rank the three documents by relevance.
Output only the final ranking.

Query: michael jordan statistician

Document 1: Michael Irwin Jordan is an American scientist, professor at the University of California, Berkeley, research scientist at the Inria Paris, and researcher in machine learning, statistics, and artificial intelligence.

Document 2: Michael Jeffrey Jordan is an American former professional basketball player and current businessman, who is a minority owner of the Charlotte Hornets of the National Basketball Association (NBA).

Document 3: An Overview of Large Language Models for Statisticians. Wenlong Ji, Weizhe Yuan, Emily Getzen, Kyunghyun Cho, Michael I. Jordan, Song Mei, Jason E Weston, Weijie J. Su, Jing Xu, Linjun Zhang. Large Language Models (LLMs) have emerged as transformative tools in artificial intelligence (AI), exhibiting remarkable capabilities across diverse tasks such as text generation, reasoning, and decision-making.

1, 3, 2

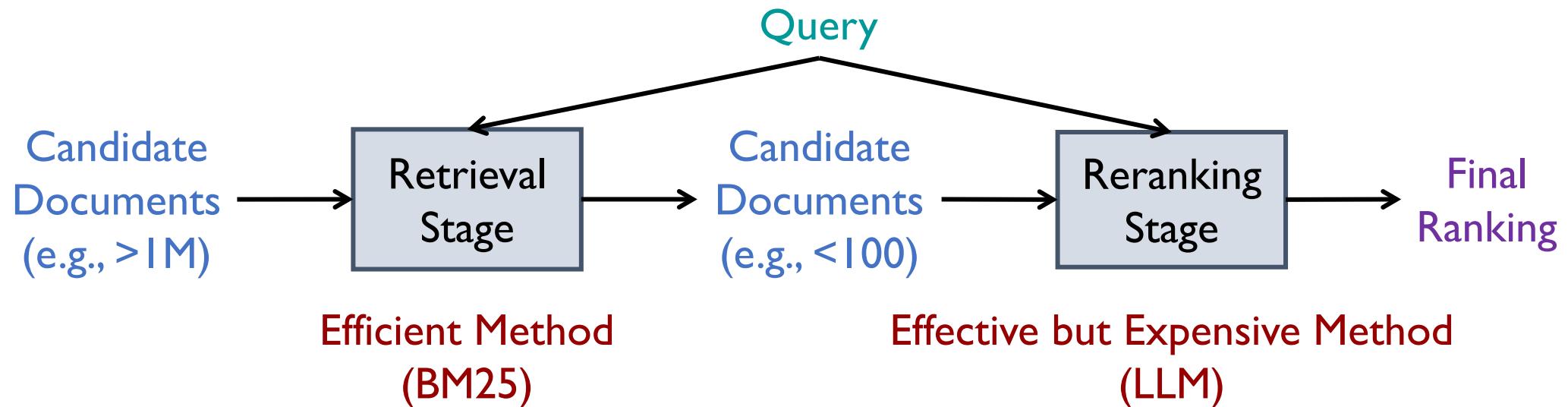


+ Ask anything



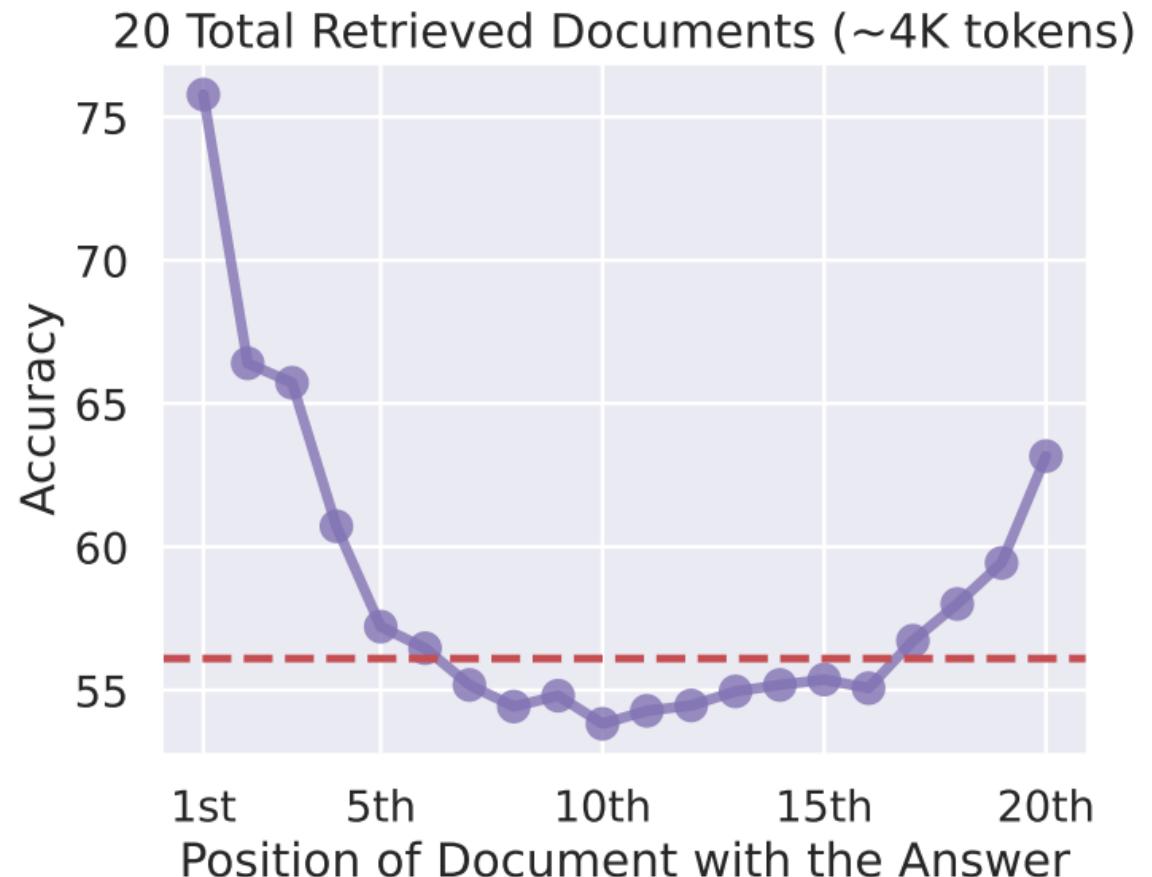
How do LLMs help search engines?

- **Initial Idea:** Feed the query and all candidate documents to the LLM, and let it output the ranking
- Can we generalize this idea to 1,000 or 1,000,000 candidate documents?
 - No! Because of the maximum input length limit, prohibitive costs, ...
 - How to address this?
 - The retrieval-reranking paradigm



How do LLMs help search engines?

- **Initial Idea:** Feed the query and all candidate documents to the LLM, and let it output the ranking
- Can we generalize this idea to **20** candidate documents?
 - Not entirely! Because LLMs “**lost in the middle**” when the context is long
 - If LLMs are more powerful with no more than 5 candidate documents, how can we make it rank 20 documents?



Background: Three Types of Ranking Strategies

Three Types of Ranking Strategies

Pointwise

 → Score: 7

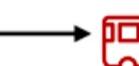
 → Score: 14

 → Score: 21

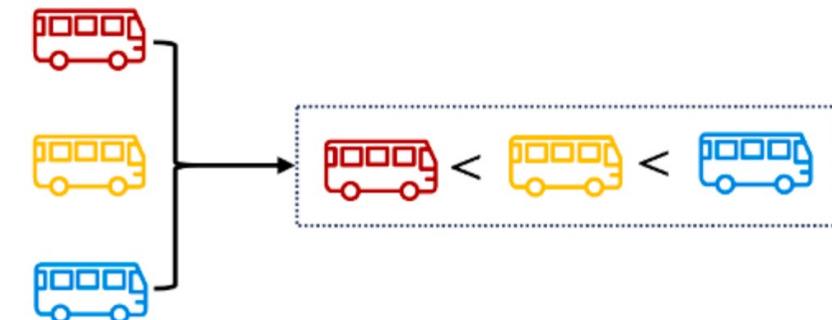
Pairwise

,  →  < 

,  →  < 

,  →  < 

Listwise



- Final Output:

 <  < 

Pointwise

- Given a **query q** and a candidate **document d** , directly calculate $\text{score}(\mathbf{q}, \mathbf{d})$
- Examples?
 - TF-IDF
 - BM25
 - Dual Embedding Space Model (DESM) [in the Neural Ranking lecture]
 - Cross-Encoder [in the BERT-Based Ranking lecture]
 - ...

Pairwise

- Given a **query q** and two candidate **documents d_1, d_2** , predict if d_1 should be ranked higher than d_2
- Examples?
 - Ranking SVM [in the Learning to Rank lecture]
 - RankNet [in the Learning to Rank lecture]
 - Bayesian Personalized Ranking (BPR)
 - Dense Passage Retrieval [in the Neural Ranking lecture]
 - CoBERT [in the BERT-Based Ranking lecture]
 - ...

Listwise

- Given a **query q** and ALL candidate **documents d_1, d_2, \dots, d_N** , output the ranking list of d_1, d_2, \dots, d_N
- Examples
 - Our initial idea using ChatGPT
 - LambdaRank can be either pairwise or listwise
 - LambdaMART can be either pairwise or listwise
 - ...

How to apply these three strategies
when an LLM is available?

Pointwise Ranking w/ LLMs: Method 1 [Liang et al., arXiv 2022]

- Given a **query** q and a candidate **document** d , directly calculate score(q, d)

Passage: {{passage}}

Query: {{query}}

Does the passage answer
the query?

Yes (or No)

The LLM's output will be **binary**, but you need a **continuous** score for ranking. What can you do?

Given a passage and a query, predict whether the passage includes an answer to the query by producing either 'Yes' or 'No'.

Passage: {{passage}}

Query: {{query}}

Does the passage answer the query?

Answer:

Pointwise Ranking w/ LLMs: Method 1 [Liang et al., arXiv 2022]

- Check the model's **probabilities** for outputting the “Yes” or “No” token!
- If the model outputs “Yes”, then $\text{score}(q, d) = 1 + p(\text{“Yes”})$
- If the model outputs “No”, then $\text{score}(q, d) = 1 - p(\text{“No”})$
- Limitation?
 - Rely on the output probabilities of an LLM, which was unavailable for many closed-source LLM APIs
 - Currently, ChatGPT’s web interface still does not show token probabilities, but you can get them via the OpenAI Responses API

```
json
"logprobs": true,
"top_logprobs": 5
```

 Copy code

Pointwise Ranking w/ LLMs: Method 2 [Sachan et al., EMNLP 2022]

- Given a **query** q and a candidate **document** d , directly calculate $\text{score}(q, d)$

Please write a question
based on this passage.
Passage: {{passage}}
Query:

{{query}}

The relevance between a **query** and a **document** is measured by the probability of the model to generate the **query** based on the **document**.

Please write a question based on this passage.
Passage: {{passage}}
Question: {{query}}

- How to calculate this probability?

Pointwise Ranking w/ LLMs: Method 2 [Sachan et al., EMNLP 2022]

- How to calculate this probability?

$$\prod_t p(\mathbf{q}_t | \mathbf{i}, \mathbf{d}, \mathbf{q}_{<t})$$

- \mathbf{i} : instruction; \mathbf{d} : document; $\mathbf{q}_{<t}$: the first $(t - 1)$ tokens in the query
- Or equivalently,

$$\text{score}(\mathbf{q}, \mathbf{d}) = \frac{1}{|\mathbf{q}|} \sum_t \log p(\mathbf{q}_t | \mathbf{i}, \mathbf{d}, \mathbf{q}_{<t})$$

Please write a question based on this passage.

Passage: {{passage}}

Question: {{query}}

- Still rely on the output **probabilities** of an LLM
- Can we rely on the output tokens only?

Listwise [Sun et al., EMNLP 2023]

Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents

Weiwei Sun^{1*} Lingyong Yan² Xinyu Ma² Shuaiqiang Wang²
Pengjie Ren¹ Zhumin Chen¹ Dawei Yin^{2†} Zhaochun Ren^{3†}

¹Shandong University, Qingdao, China ²Baidu Inc., Beijing, China

³Leiden University, Leiden, The Netherlands

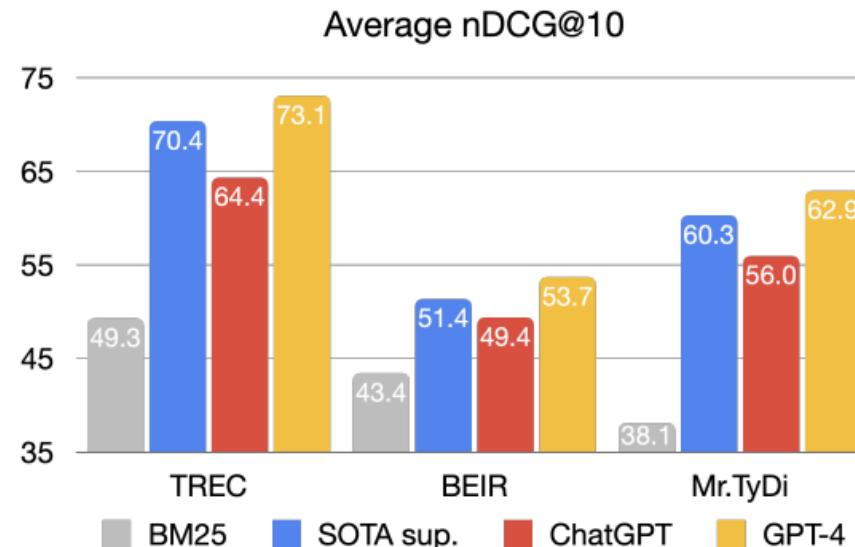
{sunnweiwei,lingyongy,xinyuma2016,shqiang.wang}@gmail.com

{renpengjie,chenzhumin}@sdu.edu.cn yindawei@acm.org

z.ren@liacs.leidenuniv.nl

Abstract

Large Language Models (LLMs) have demonstrated remarkable zero-shot generalization across various language-related tasks, including search engines. However, existing work utilizes the generative ability of LLMs for Information Retrieval (IR) rather than direct passage ranking. The discrepancy between the pre-training objectives of LLMs and the ranking objective poses another challenge. In this paper, we first investigate generative LLMs such



Listwise Ranking w/ LLMs

- Given a **query q** and ALL candidate **documents d_1, d_2, \dots, d_N** , output the ranking list of d_1, d_2, \dots, d_N

This is RankGPT, an intelligent assistant that can rank passages based on their relevancy to the query.

The following are {{num}} passages, each indicated by number identifier []. I can rank them based on their relevance to query: {{query}}

[1] {{passage_1}}

[2] {{passage_2}}

(more passages) ...

The search query is: {{query}}

I will rank the {{num}} passages above based on their relevance to the search query. The passages will be listed in descending order using identifiers, and the most relevant passages should be listed first, and the output format should be [] > [] > etc, e.g., [1] > [2] > etc.

The ranking results of the {{num}} passages (only identifiers) is:

Instruction used
for **text-davinci-003**

Instruction used for gpt-3.5-turbo and gpt-4

system:

You are RankGPT, an intelligent assistant that can rank passages based on their relevancy to the query.

user:

I will provide you with {{num}} passages, each indicated by number identifier []. Rank them based on their relevance to query: {{query}}.

assistant:

Okay, please provide the passages.

user:

[1] {{passage_1}}

assistant:

Received passage [1]

user:

[2] {{passage_2}}

assistant:

Received passage [2]

(more passages) ...

user

Search Query: {{query}}.

Rank the {{num}} passages above based on their relevance to the search query. The passages should be listed in descending order using identifiers, and the most relevant passages should be listed first, and the output format should be [] > [], e.g., [1] > [2]. Only response the ranking results, do not say any word or explain.

Listwise Ranking w/ LLMs

- Given a **query q** and ALL candidate **documents d_1, d_2, \dots, d_N** , output the ranking list of d_1, d_2, \dots, d_N

The following are passages related to query {{query}}

[1] {{passage_1}}

[2] {{passage_2}}

(more passages)

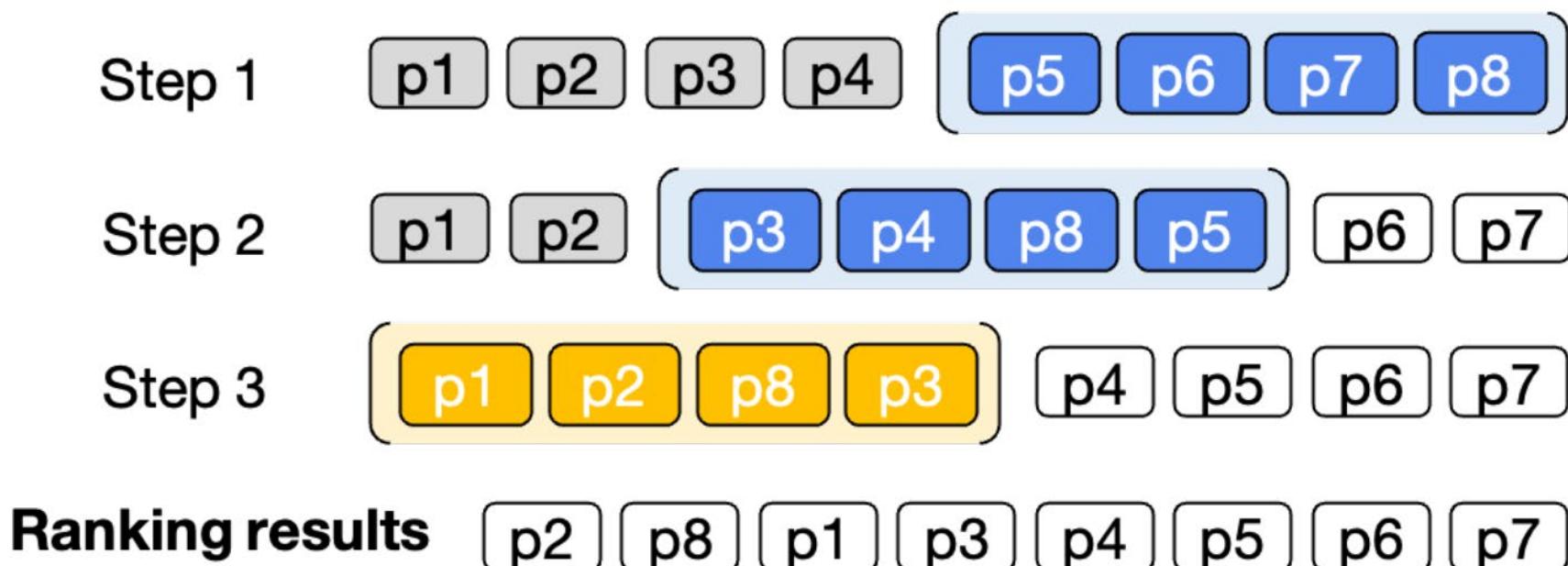
Rank these passages based on their relevance to the query.

[2] > [3] > [1] > [...]

- Previous unresolved issue:** If LLMs are more powerful with no more than 5 candidate documents, how can we make it rank 20 documents?

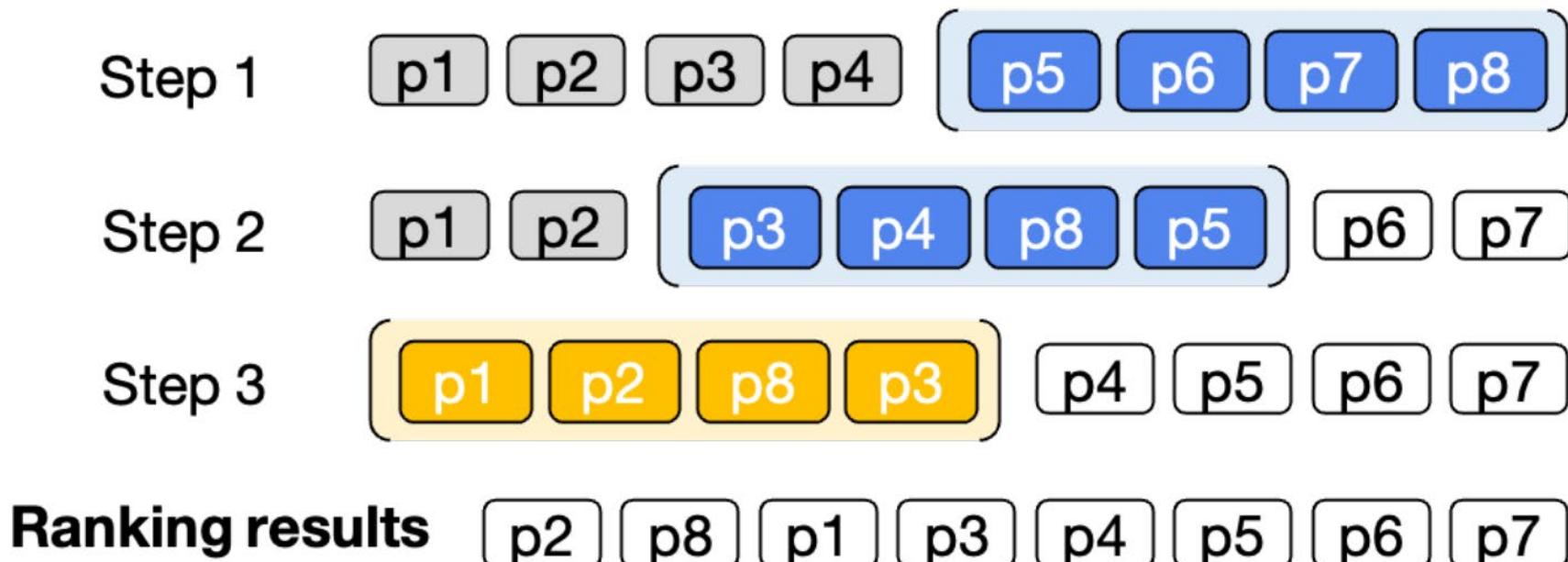
Sliding Window

- The LLM ranks only w documents at a time (e.g., $w = 4$)
- The LLM ranks all candidate in a **back-to-first** order using a sliding window
- The top- s documents in the previous window will participate in the next window (e.g., $s = 2$)



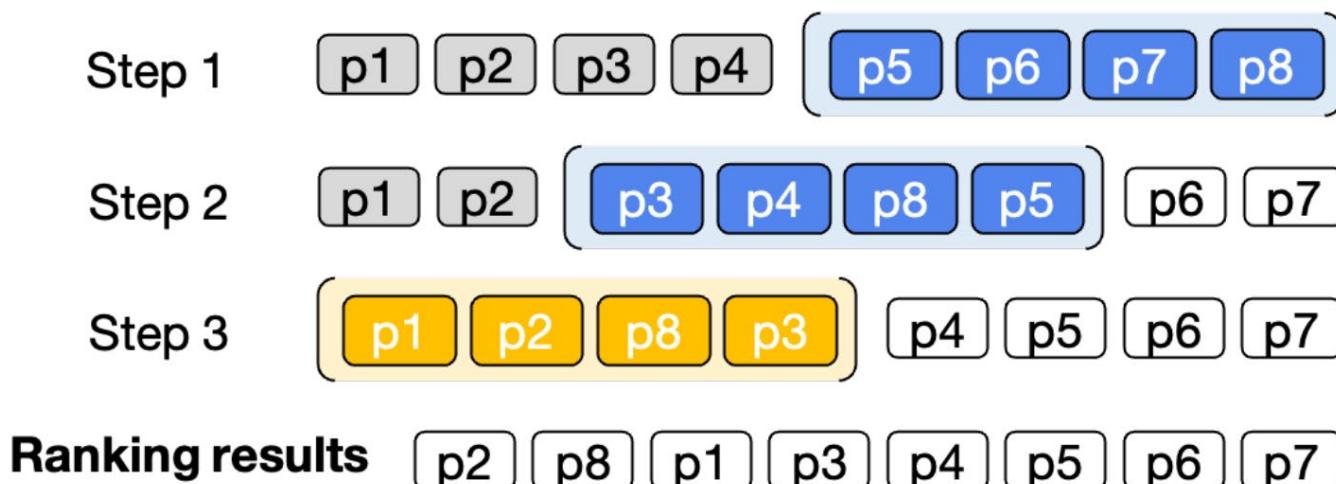
Sliding Window

- Assume there are N documents to be reranked. How many times do you need to call the LLM?
 - $\left\lceil \frac{N-w}{w-s} \right\rceil + 1$ (e.g., $\left\lceil \frac{8-4}{4-2} \right\rceil + 1 = 3$)



Sliding Window

- Why back-to-first?
 - Ensure that the **top portion** of the ranking list is more accurate, while the ordering toward the end is relatively less important
- How many of the top-ranked documents can this method guarantee to be absolutely accurate? (Assume the LLM is perfect at ranking)
 - Top- S



Results on TREC and BEIR

Method	DL19	DL20	Covid	NFCorpus	Touche	DBPedia	SciFact	Signal	News	Robust04	BEIR (Avg)
BM25	50.58	47.96	59.47	30.75	44.22	31.80	67.89	33.05	39.52	40.70	43.42
Supervised											
monoBERT (340M)	70.50	67.28	70.01	36.88	31.75	41.87	71.36	31.44	44.62	49.35	47.16
monoT5 (220M)	71.48	66.99	78.34	37.38	30.82	42.42	73.40	31.67	46.83	51.72	49.07
monoT5 (3B)	71.83	68.89	80.71	38.97	32.41	44.45	76.57	32.55	48.49	56.71	51.36
Cohere Rerank-v2	73.22	67.08	81.81	36.36	32.51	42.51	74.44	29.60	47.59	50.78	49.45
Unsupervised											
UPR (FLAN-T5-XL)	53.85	56.02	68.11	35.04	19.69	30.91	72.69	31.91	43.11	42.43	42.99
InPars (monoT5-3B)	-	66.12	78.35	-	-	-	-	-	-	-	-
Promptagator++ (few-shot)	-	-	76.2	37.0	38.1	43.4	73.1	-	-	-	-
LLM API (Permutation generation)											
gpt-3.5-turbo	65.80	62.91	76.67	35.62	36.18	44.47	70.43	32.12	48.85	50.62	49.37
gpt-4 [†]	75.59	70.56	85.51	38.47	38.57	47.12	74.95	34.40	52.89	57.55	53.68

Table 1: **Results (nDCG@10) on TREC and BEIR.** Best performing unsupervised and overall system(s) are marked bold. All models except InPars and Promptagator++ re-rank the same BM25 top-100 passages. [†]On BEIR, we use gpt-4 to re-rank the top-30 passages re-ranked by gpt-3.5-turbo to reduce the cost of calling gpt-4 API.

Prediction Failures of Listwise Ranking w/ LLMs

- **Missing:** When LLMs only outputs a partial list of the input documents
 - E.g., [2] > [4] > [1]
- **Rejection:** LLMs refuse to perform the ranking task and produce irrelevant outputs
- **Repetition:** LLMs output the same document more than once
 - E.g., [2] > [4] > [3] > [1] > [4]
- **Inconsistency:** The same list of documents have different output rankings when they are fed in with different order or context

Pairwise [Qin et al., NAACL 2024]

Large Language Models are Effective Text Rankers with Pairwise Ranking Prompting

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen,
Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, Michael Bendersky

Google Research

{zhenqin,jagerman,kaihuibj,hlz,junru,lyyanle,jmshen,tianqiliu,jialu,metzler,xuanhui,bemike}@google.com

Abstract

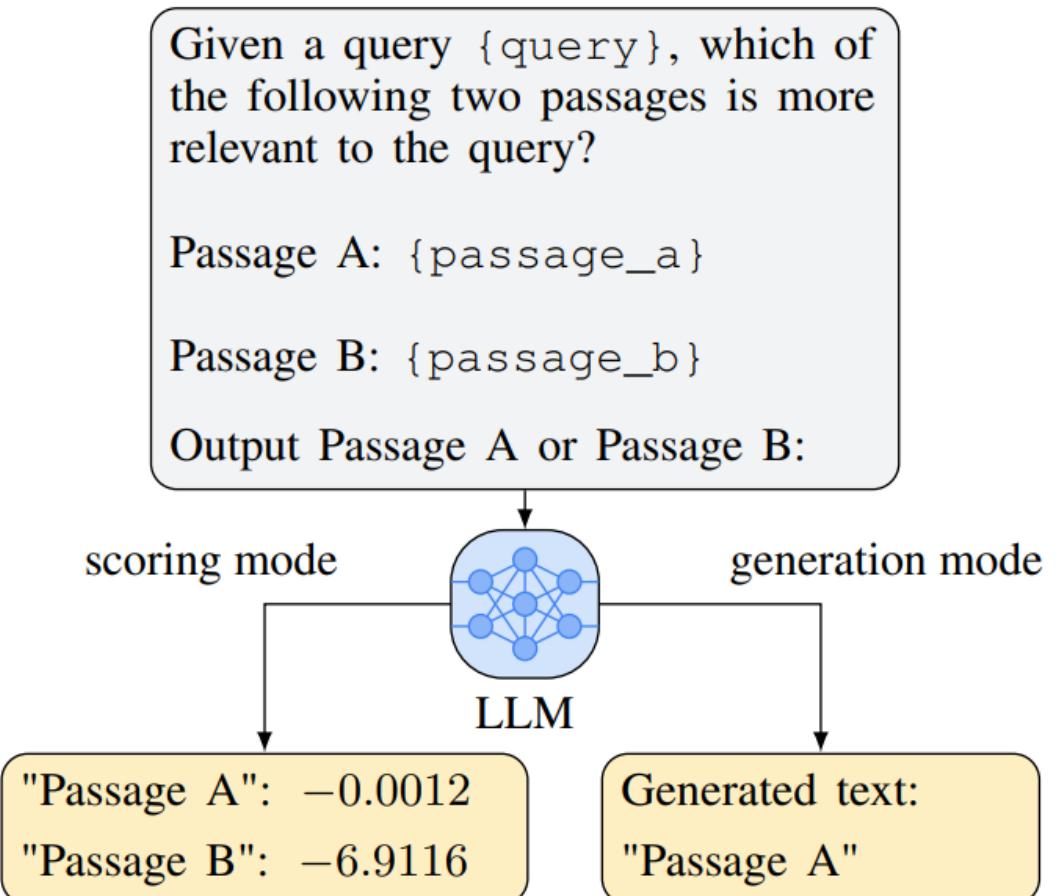
Ranking documents using Large Language Models (LLMs) by directly feeding the query and candidate documents into the prompt is an interesting and practical problem. However, researchers have found it difficult to outperform fine-tuned baseline rankers on benchmark datasets. We analyze pointwise and listwise ranking prompts used by existing methods and argue that off-the-shelf LLMs do not fully understand these challenging ranking formulations. In this paper, we propose to sig-

potentially trained with millions of labeled examples, even in the zero-shot setting (Kojima et al., 2022; Agrawal et al., 2022; Huang et al., 2022; Hou et al., 2023).

However, there is limited success for the important text ranking problem using off-the-shelf LLMs (Ma et al., 2023). Existing results usually significantly underperform well-trained baseline rankers (e.g., Nogueira et al. (2020); Zhuang et al. (2023)). The only exception is a recent approach proposed by Sun et al. (2023b), which depends on

Pairwise Ranking w/ LLMs

- Given a query q and two candidate documents d_1, d_2 , predict if d_1 should be ranked higher than d_2
- Scoring mode:** Compare the output probabilities $p(\text{"Passage A"})$ and $p(\text{"Passage B"})$
- Generation mode:** Directly check the output tokens



Pairwise Ranker → Ranking for the Entire List

- All pair comparisons (**PRP-Allpair**): $O(N^2)$ API calls

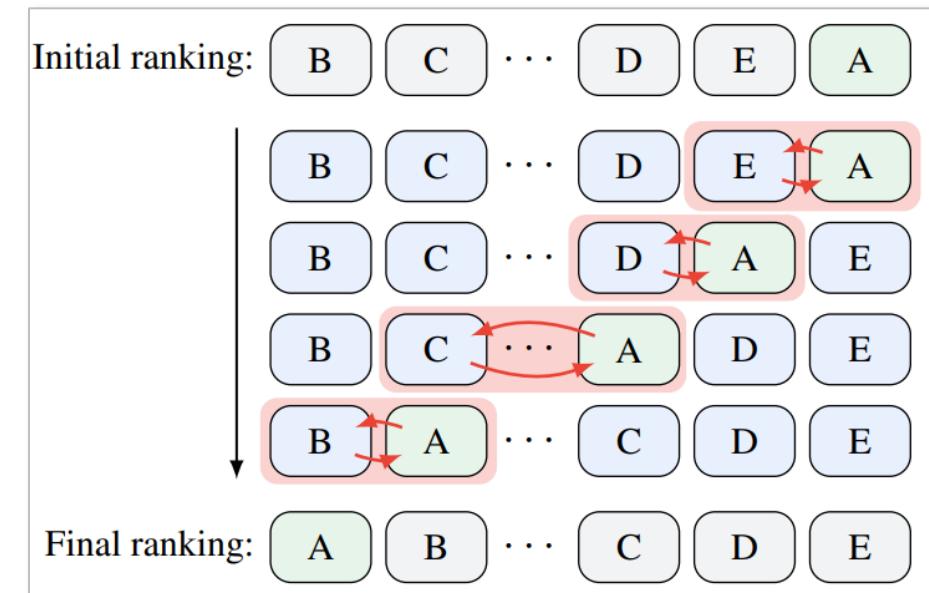
$$\text{score}(q, d) = 1 \times \sum_{d' \neq d} \mathbb{I}(d > d') + 0.5 \times \sum_{d' \neq d} \mathbb{I}(d = d')$$

- Sorting-based (**PRP-Sorting**): $O(N \log N)$ API calls

- Quicksort or Heapsort

- Sliding window (**PRP-Sliding**): $O(N)$ API calls

- Back-to-first
 - One-pass Bubblesort
 - k -pass Bubblesort (k is small)



Pairwise Ranker → Ranking for the Entire List

- Sliding window (**PRP-Sliding**): $O(N)$ API calls
 - Back-to-first
 - k -pass Bubblesort (k is small)
 - Guarantee the top- k documents to be absolutely accurate (Assume the LLM is perfect at ranking)
- Comparison between Pointwise, Pairwise, and Listwise ranking w/ LLMs:

Method	# of LLM API Calls	Generation API	Scoring API
Pointwise	$O(N)$	No	Yes
Listwise	$O(N)$	Yes	No
Pairwise	$O(N^2), O(N \log N), O(N)$	Yes	Yes

Results on TREC

Method	LLM	Size	TREC-DL2019			TREC-DL2020		
			NDCG@1	NDCG@5	NDCG@10	NDCG@1	NDCG@5	NDCG@10
BM25	NA	NA	54.26	52.78	50.58	57.72	50.67	47.96
Supervised Methods								
monoBERT	BERT	340M	79.07	73.25	70.50	78.70	70.74	67.28
monoT5	T5	220M	79.84	73.77	71.48	77.47	69.40	66.99
monoT5	T5	3B	79.07	73.74	71.83	80.25	72.32	68.89
RankT5	T5	3B	79.07	75.66	72.95	80.86	73.05	69.63
Unsupervised LLM Methods								
LRL	text-davinci-003	175B	-	-	65.80	-	-	62.24
RankGPT	gpt-3	175B	50.78	50.77	49.76	50.00	48.36	48.73
RankGPT	text-davinci-003	175B	69.77	64.73	61.50	69.75	58.76	57.05
RankGPT	gpt-3.5-turbo	154B*	82.17	71.15	65.80	79.32	66.76	62.91
RankGPT	gpt-4	1T*	82.56	79.16	75.59	78.40	74.11	70.56
UPR	FLAN-T5-XXL	11B	62.79	62.07	62.00	64.20	62.05	60.34
RG	FLAN-T5-XXL	11B	67.05	65.41	64.48	65.74	66.40	62.58
UPR	FLAN-UL2	20B	53.10	57.68	58.95	64.81	61.50	60.02
RG	FLAN-UL2	20B	70.93	66.81	64.61	75.62	66.85	65.39
PRP-Allpair	FLAN-T5-XL	3B	74.03	71.73	69.75	79.01	72.22	68.12
PRP-Sorting	FLAN-T5-XL	3B	77.52	71.88	69.28	74.38	69.44	65.87
PRP-Sliding-10	FLAN-T5-XL	3B	75.58	71.23	68.66	75.62	69.00	66.59
PRP-Allpair	FLAN-T5-XXL	11B	72.09	71.28	69.87	82.41	74.16	69.85
PRP-Sorting	FLAN-T5-XXL	11B	74.42	69.62	67.81	72.53	71.28	67.77
PRP-Sliding-10	FLAN-T5-XXL	11B	64.73	69.49	67.00	75.00	70.76	67.35
PRP-Allpair	FLAN-UL2	20B	73.64	74.77	72.42	85.19	74.73	70.68
PRP-Sorting	FLAN-UL2	20B	74.42	73.60	71.88	84.57	72.52	69.43
PRP-Sliding-10	FLAN-UL2	20B	78.29	75.49	72.65	85.80	75.35	70.46

Results on BEIR

Method	LLM	Size	Covid	Touche	DBPedia	SciFact	Signal	News	Robust04	Avg
BM25	NA	NA	59.47	44.22	31.80	67.89	33.05	39.52	40.70	45.23
Supervised Methods										
monoBERT	BERT	340M	70.01	31.75	41.87	71.36	31.44	44.62	49.35	48.63
monoT5	T5	220M	78.34	30.82	42.42	73.40	31.67	46.83	51.72	50.74
monoT5	T5	3B	80.71	32.41	44.45	76.57	32.55	48.49	56.71	53.13
RankT5	T5	3B	82.00	37.62	44.19	76.86	31.80	48.15	52.76	53.34
TART-Rerank	T5	3B	75.10	27.46	42.53	74.84	25.84	40.01	50.75	48.08
Unsupervised LLM Methods										
UPR	FLAN-T5-XXL	11B	72.64	21.56	35.14	73.54	30.81	42.99	47.85	46.36
RG	FLAN-T5-XXL	11B	70.31	22.10	31.32	63.43	26.89	37.34	51.56	43.28
UPR	FLAN-UL2	20B	70.69	23.68	34.64	71.09	30.33	41.78	47.52	45.68
RG	FLAN-UL2	20B	70.22	24.67	30.56	64.74	29.68	43.78	53.00	45.24
RankGPT	gpt-3.5-turbo	154B	76.67	36.18	44.47	70.43	32.12	48.85	50.62	51.33
PRP-Allpair	FLAN-T5-XL	3B	81.86	26.93	44.63	73.25	32.08	46.52	54.02	51.33
PRP-Sorting	FLAN-T5-XL	3B	80.41	28.23	42.84	67.94	30.95	42.95	50.07	49.06
PRP-Sliding-10	FLAN-T5-XL	3B	77.58	40.48	44.77	73.43	35.62	46.45	50.74	52.72
PRP-Allpair	FLAN-T5-XXL	11B	79.62	29.81	41.41	74.23	32.22	47.68	56.76	51.67
PRP-Sorting	FLAN-T5-XXL	11B	78.75	29.61	39.23	70.10	31.28	44.68	53.01	49.52
PRP-Sliding-10	FLAN-T5-XXL	11B	74.39	41.60	42.19	72.46	35.12	47.26	52.38	52.20
PRP-Allpair	FLAN-UL2	20B	82.30	29.71	45.94	75.70	32.26	48.04	55.49	52.78
PRP-Sorting	FLAN-UL2	20B	82.29	25.80	44.53	67.07	32.04	45.37	51.45	49.79
PRP-Sliding-10	FLAN-UL2	20B	79.45	37.89	46.47	73.33	35.20	49.11	53.43	53.55

Takeaway Messages

- BM25 retrieval + LLM reranking currently the best-performing paradigm for zero-shot retrieval in unseen domains
 - Ultimately, we outperform BM25 alone in the zero-shot setting!
- If you want to leverage the LLM's output probabilities, use either a pointwise or pairwise approach
- If you want to leverage the LLM's output tokens only, use either a pairwise or listwise approach
- Sliding window is an efficient and effective strategy. It is often not only faster than all-pair comparisons and full sorting but also surprisingly performs better
 - Possibly because LLMs are NOT perfect or consistent

**Extended Content
(will not appear in quizzes or the exam)**

Fine-Tuning Open-Source LLMs [Ma et al., SIGIR 2024]

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

ABSTRACT

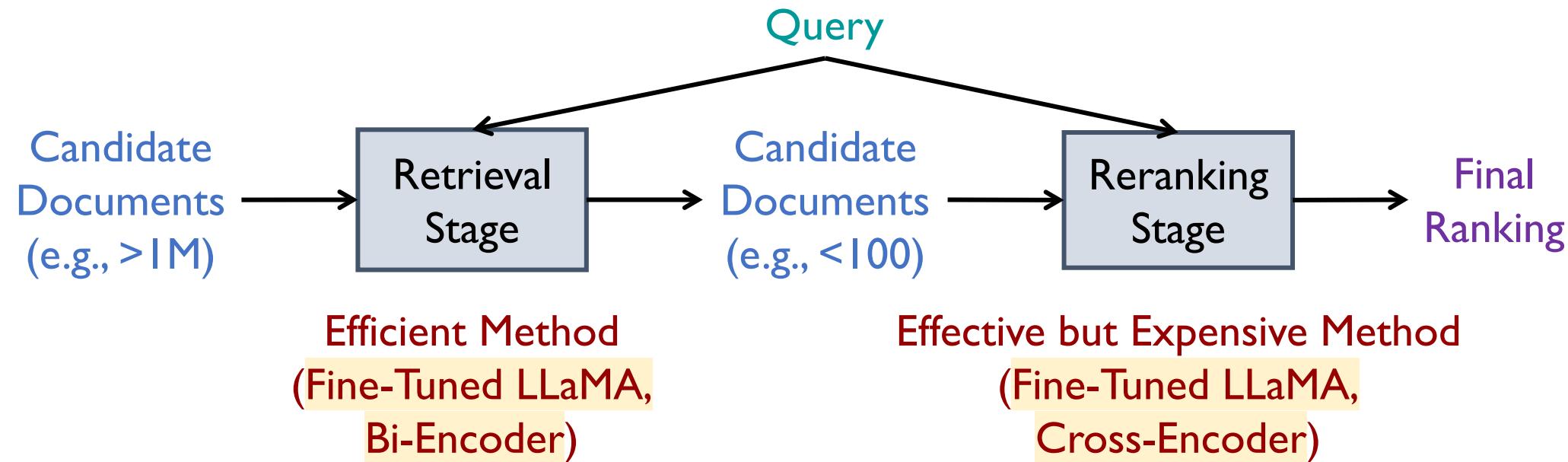
While large language models (LLMs) have shown impressive NLP capabilities, existing IR applications mainly focus on prompting LLMs to generate query expansions or generating permutations for listwise reranking. In this study, we leverage LLMs directly to serve as components in the widely used multi-stage text ranking pipeline. Specifically, we fine-tune the open-source LLaMA-2 model as a dense retriever (repLLaMA) and a pointwise reranker (rank-LLaMA). 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

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-augmented 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

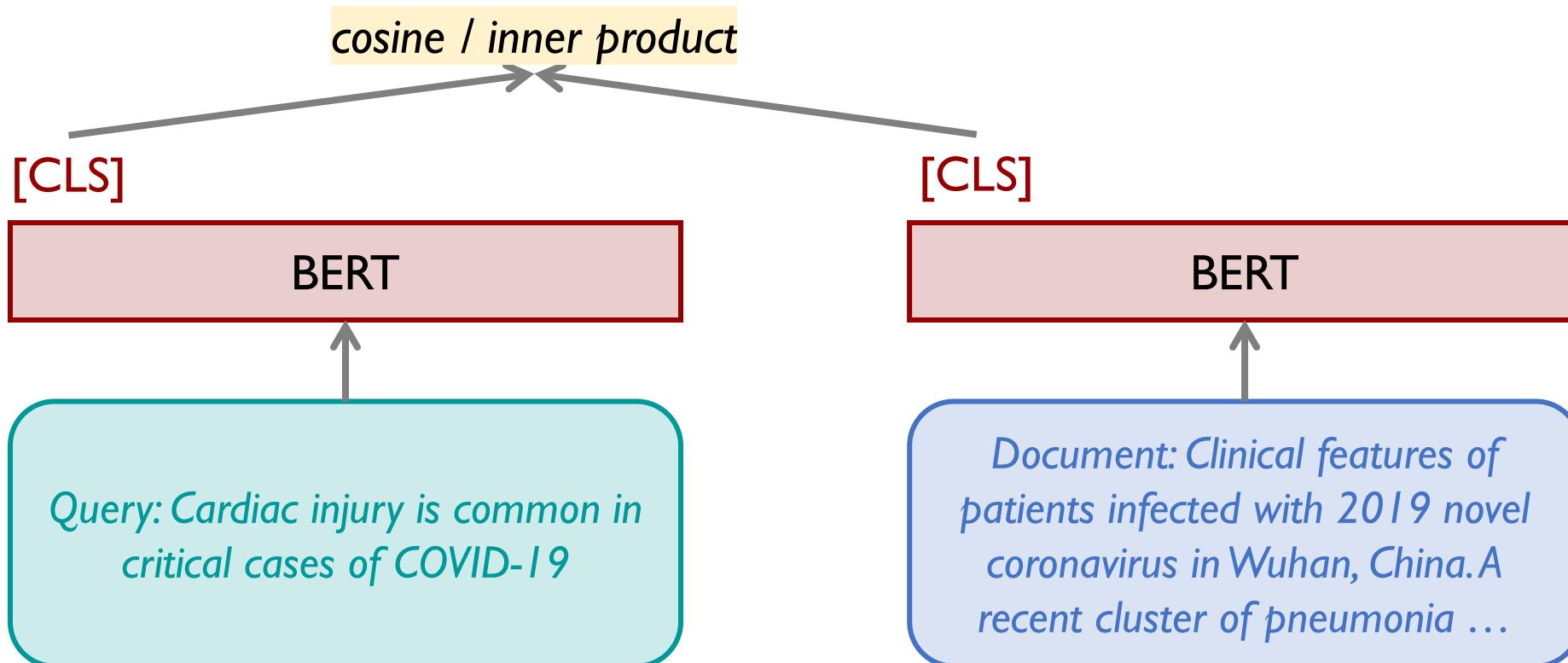
Retrieval-Reranking Paradigm



- **Key challenge:** How to get the query / document embedding given an LLM with the Transformer **decoder** architecture?

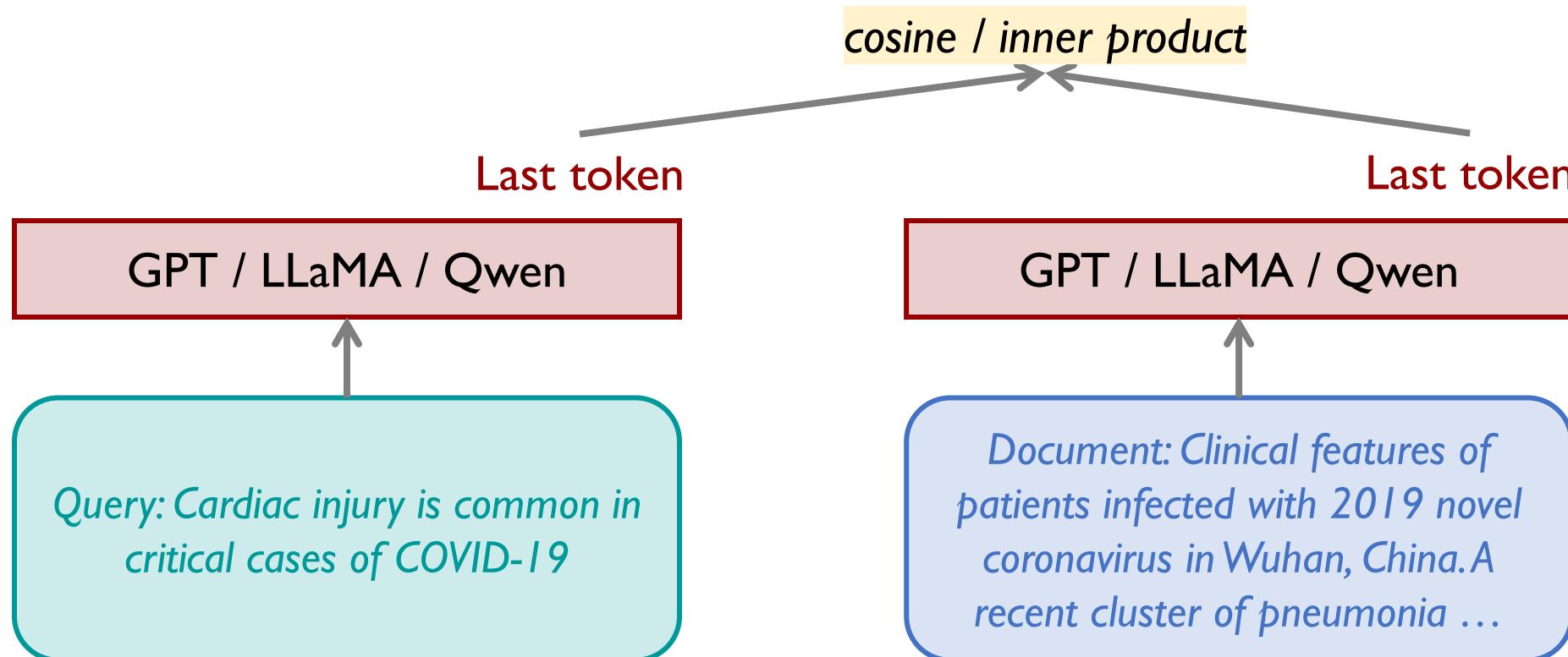
Bi-Encoder (BERT Version)

- The output vector of the [CLS] token serves as **query** / document embedding
- Can we directly apply this idea to GPT, LLaMA, or Qwen (all of which use the Transformer **decoder** architecture)? Why not?



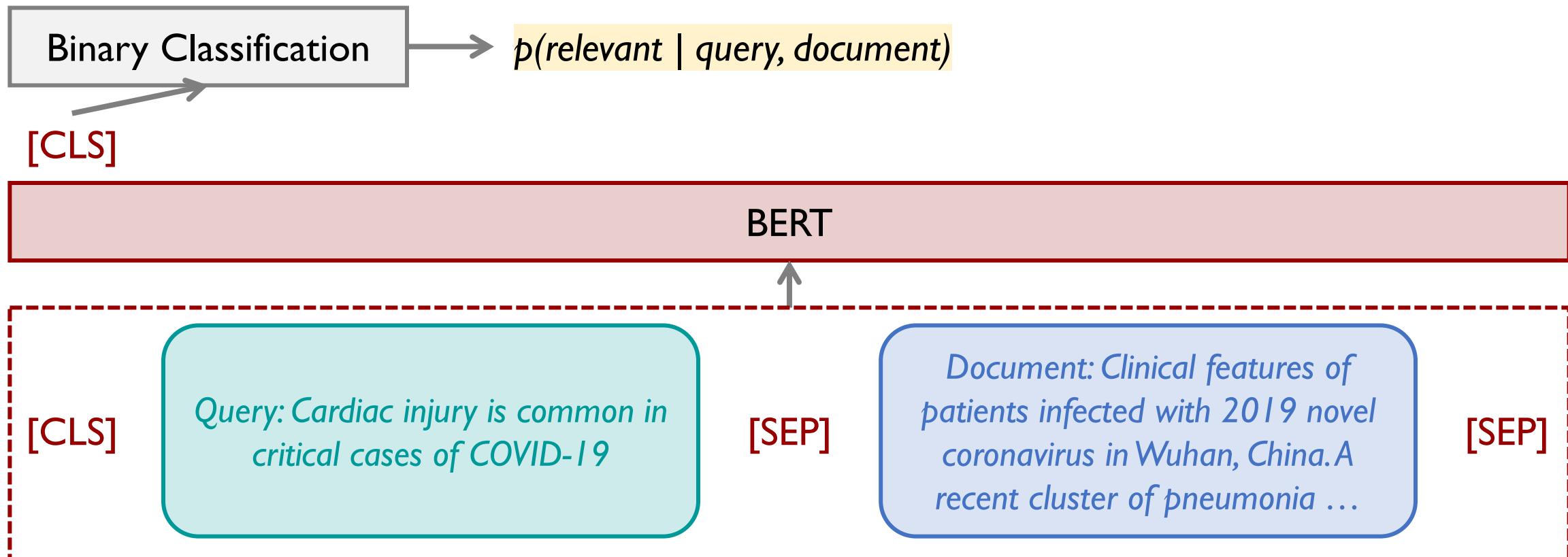
Bi-Encoder (GPT / LLaMA / Qwen Version)

- Due to the Transformer decoder's unidirectional attention mechanism, the output embedding of the first token only sees the first token itself and does not attend to any other input tokens!
- We should use the last token!



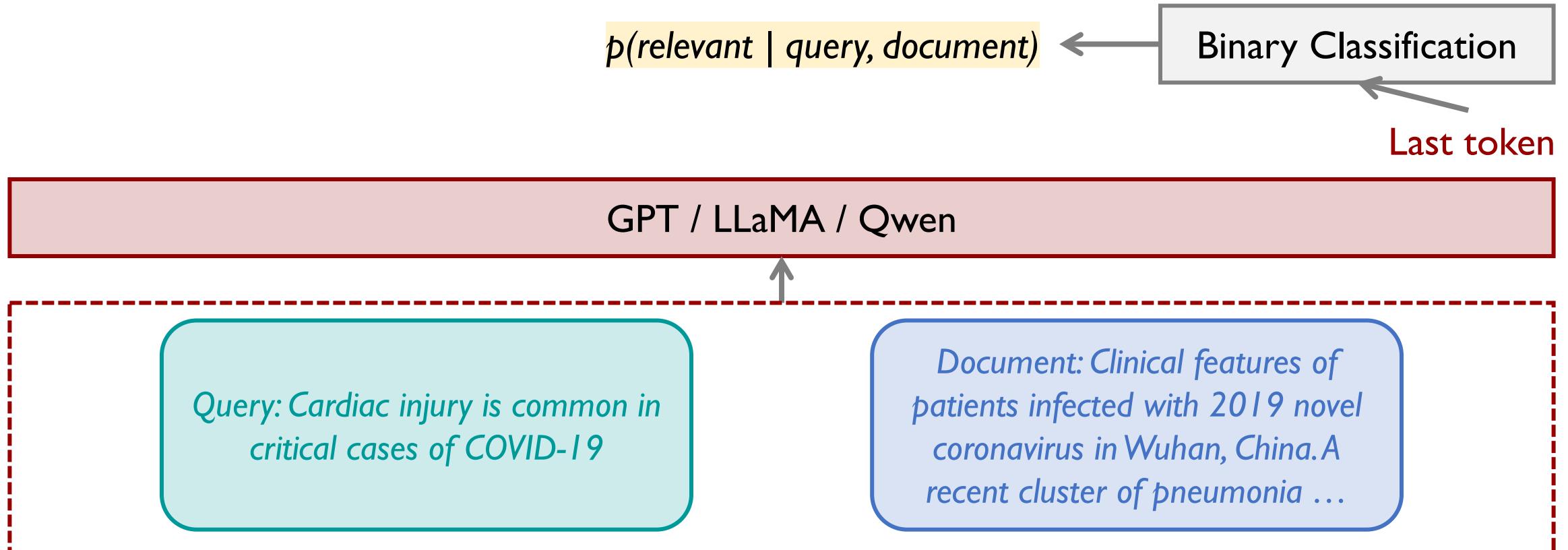
Cross-Encoder (BERT Version)

- Concatenate the **query** and **document** into a single input sequence
- Get the representation of the entire sequence and perform binary classification



Cross-Encoder (GPT / LLaMA / Qwen Version)

- Concatenate the **query** and **document** into a single input sequence
- Get the representation of the entire sequence and perform binary classification



Performance and Limitations

	Model size	Source prev. top-k	Dev MRR@10	DL19 nDCG@10	DL20 nDCG@10
<i>Retrieval</i>					
(a) BM25 [17]	-	- C	18.4	50.6	48.0
(b) ANCE [41]	125M	- C	33.0	64.5	64.6
(c) CoCondenser [9]	110M	- C	38.2	71.7	68.4
(d) GTR-base [21]	110M	- C	36.6	-	-
(e) GTR-XXL [21]	4.8B	- C	38.8	-	-
(f) OpenAI Ada2 [20]	?	- C	34.4	70.4	67.6
(g) bi-SimLM [38]	110M	- C	39.1	69.8	69.2
(h) repLLaMA	7B	- C	41.2	74.3	72.1
<i>Reranking</i>					
(i) monoBERT [23]	110M	(a) 1000	37.2	72.3	72.2
(j) cross-SimLM [38]	110M	(g) 200	43.7	74.6	72.7
(k) RankT5 [44]	220M	(d) 1000	43.4	-	-
(l) rankLLaMA	7B	(h) 200	44.9	75.6	77.4
(m) rankLLaMA-13B	13B	(h) 200	45.2	76.0	77.9
(n) RankVicuna [27]	7B	(a) 100	-	66.8	65.5
(o) PRP [28]	20B	(a) 100	-	72.7	70.5
(p) RankGPT _{3.5} [32]	?	(a) 100	-	65.8	72.9
(q) RankGPT ₄ [32]	?	(p) 30	-	75.6	70.6

Table 1: The effectiveness of repLLaMA and rankLLaMA on the MS MARCO passage corpus compared to baselines.

- **Limitation 1:** Fine-tuning is necessary. The last token only guarantees attention to all input tokens but does not ensure a good representation of the entire input. This requires training
- **Limitation 2:** Must use an open-source model; otherwise, fine-tuning is not possible

Fine-tuned LLaMA

Zero-shot Pairwise

Zero-shot Listwise



Thank You!

Course Website: <https://yuzhang-teaching.github.io/CSCE670-F25.html>