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

Weiwei Sun[1] **Lingyong Yan**[2] **Xinyu Ma**[2] **Pengjie Ren**[1] **Dawei Yin**[2] **Zhaochun Ren**[1]
[1]Shandong University [2]Baidu Inc.
{sunnweiwei,lingyongy,xinyuma2016}@gmail.com
{renpengjie,zhaochun.ren}@sdu.edu.cn yindawei@acm.org

Abstract

Large Language Models (LLMs) have demonstrated a remarkable ability to generalize zero-shot to various language-related tasks. This paper focuses on the study of exploring generative LLMs such as ChatGPT and GPT-4 for relevance ranking in Information Retrieval (IR). Surprisingly, our experiments reveal that properly instructed ChatGPT and GPT-4 can deliver competitive, even superior results than supervised methods on popular IR benchmarks. Notably, GPT-4 outperforms the fully fine-tuned monoT5-3B on MS MARCO by an average of 2.7 nDCG on TREC datasets, an average of 2.3 nDCG on eight BEIR datasets, and an average of 2.7 nDCG on ten low-resource languages Mr.TyDi. Subsequently, we delve into the potential for distilling the ranking capabilities of ChatGPT into a specialized model. Our small specialized model that trained on 10K ChatGPT generated data outperforms monoT5 trained on 400K annotated MS MARCO data on BEIR. The code to reproduce our results is available at www.github.com/sunnweiwei/RankGPT

1 Introduction

Large Language Models (LLMs), such as ChatGPT and GPT-4 (OpenAI, 2022, 2023), are revolutionizing natural language processing with strong zero-shot and few-shot generalization. By pre-training on large-scale text corpora and alignment fine-tuning to follow human instructions, LLMs have demonstrated their superior capability in language understanding, generation, interaction, and reasoning (Ouyang et al., 2022).

The success of LLMs has also attracted broad attention in the Information Retrieval (IR) community. These studies mainly focus on exploiting the generation ability of LLMs, rather than exploring the LLMs' ability for searching. For example, New Bing utilizes GPT-4 to generate responses based on the retrieved documents (Microsoft, 2023). As a result, it is still unclear whether LLMs, e.g., ChatGPT, are good at search.

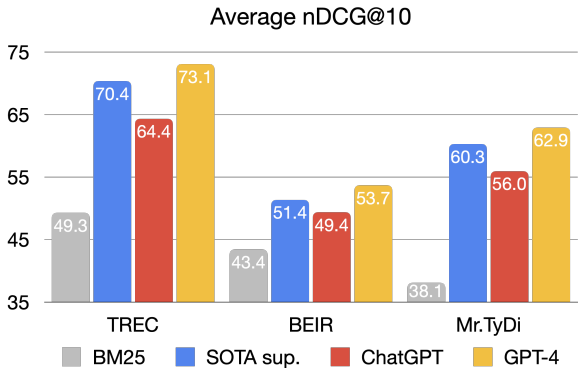

Figure 1: Average results of ChatGPT and GPT-4 (zero-shot) on passage re-ranking benchmarks (TREC, BEIR, and Mr.TyDi), compared with BM25 and previous best supervised systems (SOTA sup., e.g., monoT5 (Nogueira et al., 2020)).

To this end, this paper aims to investigate the potential of LLMs in relevance ranking for IR. Specifically, we focus on the following two questions:

- **(RQ1)** How does ChatGPT perform on passage re-ranking tasks?
- **(RQ2)** How to imitate the ranking capabilities of ChatGPT to a smaller, specialized model?

To answer the first question, we explore two strategies (Sachan et al., 2022a; Liang et al., 2022) to instruct ChatGPT performing on passage re-ranking tasks, which we named *instructional query generation* and *instructional relevance generation*. However, we observe that these methods have limited performance in re-ranking and heavily rely on the availability of log-probability of model output. We thus propose an alternative *instructional permutation generation* approach, which instructs the LLMs to directly output the permutations of a group of passages. In addition, we introduce a sliding windows strategy that allows LLMs to rank an arbitrary number of passages.

Three well-established passage ranking benchmarks are used for evaluation: (i) TREC-DL19 and DL20 (Craswell et al., 2020); (ii) eight specific passage retrieval tasks in BEIR (Thakur et al., 2021), and (iii) a multilingual passage ranking benchmark My.TyDi for ten low-resource languages (Zhang et al., 2021). Our results show that GPT-4 with zero-shot instructional permutation generation outperforms supervised systems on almost all datasets. As shown in Figure 1, GPT-4 outperforms previous state-of-the-art by an average of 2.7, 2.3, and 2.7 nDCG on TREC, BEIR, and My.TyDi, respectively.

To answer the second question, we introduce a permutation distillation technique to imitate the passage ranking capabilities of ChatGPT into a smaller, specialized ranking model. Specifically, we randomly sample 10K queries from MS MARCO, each of which is retrieved by BM25 with 20 candidate passages. We utilize ChatGPT to obtain the permutation of the candidate passages and then employ a list-wise objective (Wang et al., 2018) to optimize a student model, ensuring that it learns to generate the same permutation as ChatGPT. Our experiments reveal that the student model outperforms fully fine-tuned monoT5 by an average of 1.53 nDCG on BEIR. The proposed distillation method also shows advantages in cost-efficiency.

In summary, the contributions of this paper are tri-fold:

- We examine instruction methods for LLMs on passage re-ranking tasks and introduce a novel permutation generation approach; See Section 2 for details.
- We comprehensively evaluate the capabilities of ChatGPT and GPT-4 on various passage re-ranking benchmarks; See Section 3 for details.
- We propose a distillation approach of learning specialized models with the permutation generated by ChatGPT; See Section 4 for details.

2 Passage Re-Ranking with LLMs

Passage re-ranking is a key task in IR, which aims to rerank a set of passages that are retrieved by first-stage retrieval models (e.g., BM25) (Lin et al., 2020; Fan et al., 2021). Existing ranking models usually rely on large-scale annotated datasets, such as MS MARCO (Campos et al., 2016). However, supervised methods suffer from the high cost of annotation, weak generalizability to new domains or languages, and restricted commercial use (Izacard et al., 2022a). Meanwhile, recent advances of

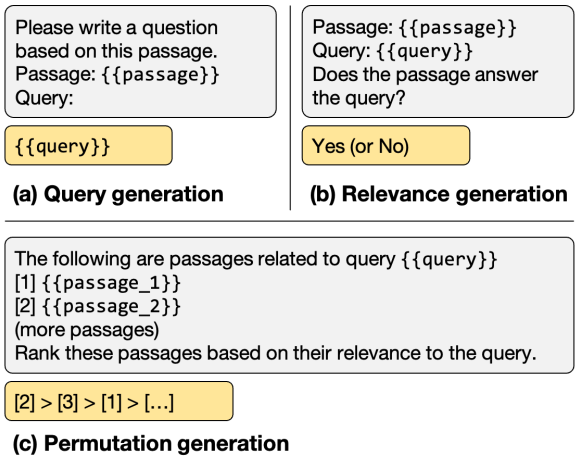

Figure 2: Three types of instruction for zero-shot passage re-ranking with LLMs. The gray and yellow blocks indicate the inputs and outputs of the model. (a) Query generation relies on the log probability of LLMs to generate the query based on the passage. (b) Relevance generation instruction LLMs to output relevance judgment. (c) Permutation generation generates a ranked list of a group of passages. See Appendix A for detailed prompts.

LLMs in NLP demonstrate their excellent ability on few/zero-shot learning and instruction-following capabilities (Brown et al., 2020). Therefore, most of the existing work investigates the usage of LLMs for improving IR systems by data generation (Bonifacio et al., 2022). In this section, we first review two existing methods performing on ChatGPT with well-designed instructions, i.e., instructional query generation (Sachan et al., 2022a) and relevance generation (Liang et al., 2022). Then, we introduce a novel instructional permutation generation method with a sliding windows strategy. Figure 2 illustrate examples of three types of instruction; all the detailed instructions are included in Appendix A.

2.1 Instructional Query Generation

Query generation has been studied in Sachan et al. (2022a); Muennighoff (2022), in which the relevance between a query and a passage is measured by the log-probability of the model to generate the query based on the passage. Figure 2 (a) shows an example of instructional query generation. Formally, given query q and a passage p_i, their relevance score s_i is calculated as:

$$s_i = \frac{1}{|q|} \sum_t \log p(q_t | q_{<t}, p_i, \mathcal{I}_{\text{query}}) \quad (1)$$

where $|q|$ denotes the number of tokens of q, q_t denotes the t-th token of q, $\mathcal{I}_{\text{query}}$ denotes the in-

structions, referring to Figure 2 (a). The passages are then ranked based on relevance score s_i.

2.2 Instructional Relevance Generation

Relevance generation is employed in HELM (Liang et al., 2022). Figure 2 (b) shows an example of instructional relevance generation, in which LLMs are instructed to output "Yes" if the query and passage are relevant or "No" if they are irrelevant. The relevance score s_i is measured by the probability of LLMs generating the word 'Yes' or 'No':

$$s_i = \begin{cases} 1 + p(\text{Yes}), & \text{if output Yes} \\ 1 - p(\text{No}), & \text{if output No} \end{cases} \quad (2)$$

where $p(\text{Yes}/\text{No})$ denotes the probability of LLMs generating Yes or No, and the relevance score is normalized into the range [0, 2].

The above two methods rely on the log probability of LLM, which is often unavailable for LLM API. For example, at the time of writing, OpenAI's `ChatCompletion` API does not provide the log-probability of generation[1]. Next, we introduce a novel instructional permutation generation approach that only uses text-based outputs to re-rank passages. We introduce a sliding windows strategy to address the LLMs maximum length limitation.

2.3 Instructional Permutation Generation

Since ChatGPT and GPT-4 have a strong capacity for multi-turn dialogue, text understanding, and reasoning, therefore, we test whether they can directly output a rank list given a set of candidate passages. As illustrated in Figure 2 (c), our approach involves inputting a group of passages into the LLMs, each identified by a unique identifier (e.g., `[1]`, `[2]`, etc.). We then ask the LLMs to generate the permutation of passages in descending order based on their relevance to the query. The passages are ranked using the identifiers, in a format such as `[2] > [3] > [1] >` etc.

The proposed method ranks passages directly without producing an intermediate relevance score. Moreover, since the recent LLMs such as ChatGPT and GPT-4 typically utilize a dialogue interface, we develop a *chat instruction* to facilitate the ranking process through a dialogue between a `user` and an `assistance`. See Appendix A.4 for details.

Sliding Windows Strategy. Due to the limited number of input tokens allowed for the LLMs, we are only able to rank a limited number of passages using the permutation generation approach. To address this limitation, we propose a sliding windows strategy that enables the permutation generation instructed LLMs to rank an arbitrary number of passages. Figure 3 shows an example of re-ranking 8 passages using sliding windows. Suppose the first-stage retrieval model retrieves M passages; we re-rank these passages in a back-to-first order using sliding windows. Specifically, we define two hyperparameters: window size (w) and step size (s). We first use the LLMs to rank the passages from $(M - w)$-th to M-th, and then slide the window in steps of s. Then we can re-rank the passages in the range from $(M - w - s)$-th to $(M - s)$-th. We repeat this process of ranking w passages within the window and sliding the window forward with step size s, until all passages have been re-ranked.

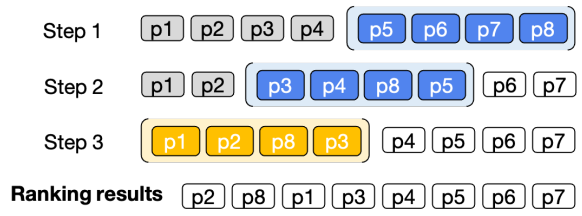

Figure 3: Illustration of re-ranking 8 passages using sliding windows a window size of 4 and a step size of 2. The blue color represents the first two windows, while the yellow color represents the last window. The sliding windows are applied in back-to-first order, meaning that the first 2 passages in the previous window will participate in the re-ranking of the next window.

Remark. Discussion of the efficiency and effectiveness of the sliding window strategy. Efficiency: The sliding window strategy involves inputting $(M//s) \times w$ passages to the model. Each passage is repeated, on average, $w//s$ times. For example, with $w = 20$ and $s = 10$, each passage is repeated twice when inputted to the model. Effectiveness: The sliding window strategy could be considered an approximation of ranking all passages, and its accuracy could be improved by repeating the process multiple times. Assuming that we have a perfect re-ranker that can always correctly rank the passages within windows, using the sliding window process K times would result in the top $K \times (w - s)$ passages in the final ranking that is equivalent to directly ranking all passages. In practice, we use $w = 20, s = 10$, and $K = 1$.

[1] https://platform.openai.com/docs/api-reference/chat/create

3 An Empirical Study

3.1 Datasets and Metrics

Our experiments are conducted on three benchmarks, TREC-DL (Craswell et al., 2020), BEIR (Thakur et al., 2021), and Mr.TyDi (Zhang et al., 2021).

TREC is a widely used benchmark dataset in information retrieval research. We use the test sets of the 2019 and 2020 competitions: (i) TREC-DL19 contains 43 queries, (ii) TREC-DL20 contains 54 queries.

BEIR consists of diverse retrieval tasks and domains. We choose eight tasks in BEIR to evaluate the models, relatively small test sets, while basically covering the BEIR evaluation tasks. (i) *Covid* retrieves scientific articles for COVID-19 related questions. (ii) *NFCorpus* is a bio-medical IR data containing queries harvested from NutritionFacts. (iii) *Touche* is a argument retrieval datasets. (iv) *DBPedia* retrieves entities from DBpedia corpus. (v) *SciFact* retrieves evidence from the research literature for claims verification. (vi) *Signal* retrieves relevant tweets for a given news article title. (vii) *News* retrieves relevant news articles for news headlines. (viii) *Robust04* provides a robust dataset that evaluates poorly performing topics.

Mr.TyDi is a multilingual passages retrieval dataset, which includes ten low-resource languages: Arabic, Bengali, Finnish, Indonesian, Japanese, Korean, Russian, Swahili, Telugu, and Thai. We use the first 100 samples in the test set of each language for evaluation.

Metrics. We re-rank the top-100 passages retrieved by BM25 using pyserini[2] and use nDCG@{1, 5,10} as evaluation metrics.

3.2 Implementation Details

We utilize four LLMs provided in the OpenAI API [3] in our experiments:

- `text-curie-001` - GPT-3 model with approximately 6.7 billion parameters (Brown et al., 2020).
- `text-davinci-003` - GPT-3.5 model trained with RLHF (Reinforcement Learning from Human Feedback) and approximately 175 billion parameters (Ouyang et al., 2022).
- `gpt-3.5-turbo` - The underlying model of ChatGPT (OpenAI, 2022).
- `gpt-4` - GPT-4 model (OpenAI, 2023).

Note that query and relevance generation requires access to the log probability of the LLMs output. However, at the time of writing, the OpenAI API does not support returning log-probability for `gpt-3.5-turbo` and `gpt-4` API.

We use `temperature=0` when calling the API. For permutation generation with sliding windows, we use window size=20, step=10, and run sliding windows for one pass (i.e., $K = 1$). We truncate the passages so that the input messages are less than the maximum length of the model (8192 for GPT-4 and 4096 for others). In BEIR and Mr.TyDi datasets, we only use GPT-4 to re-rank the top 30 passages re-ranked by ChatGPT to reduce cost.

3.3 Baselines

We include state-of-the-art supervised and unsupervised passage re-ranking methods for comparison. The **supervised baselines** are:

- **monoBERT** (Nogueira and Cho, 2019): A cross-encoder re-ranker based on BERT-large, trained on MS MARCO.
- **monoT5** (Nogueira et al., 2020): A sequence-to-sequence re-ranker that uses T5 to calculate the relevance score[4].
- **TART** (Asai et al., 2022): A supervised instruction-tuned passage re-ranker trained on 37 datasets, including over 5 million instances. The model is initialized from FLAN-T5-XL and serves as a baseline in the BEIR dataset.
- **mmarcoCE** (Bonifacio et al., 2021): A 12-layer mMiniLM-v2 cross-encoder model[5] trained on mmarco, a translated version of MS MARCO. mmarcoCE serves as a baseline for Mr.TyDi.

The **unsupervised baselines** are:

- **UPR** (Sachan et al., 2022a): Unsupervised passage ranking with instructional query generation. Due to its superior performance, we use the FLAN-T5-XL (Chung et al., 2022) as the LLM of UPR.
- **InPars** (Bonifacio et al., 2022): monoT5-3B trained on pseudo data generated by GPT-3.

[2] https://github.com/castorini/pyserini
[3] https://platform.openai.com/docs/api-reference
[4] https://huggingface.co/castorini/monot5-3b-msmarco-10k
[5] https://huggingface.co/cross-encoder/mmarco-mMiniLMv2-L12-H384-v1

Table 1: **Results on TREC-DL19 and TREC-DL20.** All models (except InPars) re-rank the top-100 passages retrieved by BM25. Best performing unsupervised and overall system(s) are marked bold.

Method		DL19			DL20		
		nDCG@1	nDCG@5	nDCG@10	nDCG@1	nDCG@5	nDCG@10
BM25		54.26	52.78	50.58	57.72	50.67	47.96
Supervised							
monoBERT (340M)		79.07	73.25	70.50	78.70	70.74	67.28
monoT5 (220M)		79.84	73.77	71.48	77.47	69.40	66.99
monoT5 (3B)		79.07	73.74	71.83	**80.25**	72.32	68.89
Unsupervised							
UPR (FLAN-T5-XL)		51.55	53.71	53.85	63.27	59.41	56.02
InPars (monoT5-3B)†		-	-	-	-	-	66.12
LLM API	Instruction methods						
`text-curie-001`	Relevance generation (4-shot)	39.53	40.02	41.53	41.98	34.80	34.91
`text-curie-001`	Query generation	50.78	50.77	49.76	50.00	48.36	48.73
`text-davinci-003`	Query generation	37.60	44.73	45.37	51.25	47.46	45.93
`text-davinci-003`	Permutation generation	69.77	64.73	61.50	69.75	58.76	57.05
`gpt-3.5-turbo`	Permutation generation	82.17	71.15	65.80	**79.32**	66.76	62.91
`gpt-4`	Permutation generation	**82.56**	**79.16**	**75.59**	78.40	**74.11**	**70.56**

† InPars re-rank top-1000 BM25 passages. We cite its results from original paper.

- **Promptagator++** (Dai et al., 2023): A 110M cross-encoder re-ranker trained on pseudo queries generated by FALN 137B.

3.4 Results on TREC

Table 1 lists the evaluation results on TREC datasets. From the results, we see that: (i) GPT-4 with permutation generation instruction achieves the best results on both datasets. As an unsupervised method, it outperforms the previous-best system monoT5 (3B) by 3.76 and 1.67 in terms of nDCG@10 in DL19 and DL20, respectively. (ii) ChatGPT performs comparably to GPT-4 on nDCG@1, but lags behind it on nDCG@10. Davinci model (`text-davinci-003`) performs poorly compared to ChatGPT and GPT. This may be because of the generation stability of Davinci and ChatGPT trails that of GPT-4. We analyze the stability of Davinci, ChatGPT, and GPT-4 in Appendix C. (iii) Permutation generation method outperforms the query or relevance generation methods in instructing LLMs to re-rank passages. Specifically, the improvement in the capability of the LLM (e.g., Curie model to Davinci model) does not result in significant performance gains for the query generation approach. This suggests that the association between the query generation probability and the relevance score does not become more relevant with a larger model and better instruction following. In contrast, the passage re-ranking capability of the permutation generation method could steadily improve along with the LLMs' proficiency in following instructions. This indicates that the permutation generation method is more effective in leveraging LLMs' capabilities for passage re-ranking than the query generation or relevance generation methods.

3.5 Results on BEIR

Table 2 shows the results on BEIR datasets. From the results, we see that: (i) GPT-4 with permutation generation instruction outperforms the state-of-the-art system(s) on nearly all datasets and achieves the best average performance across eight tasks. Specifically, it surpasses monoT5 (3B) with an average of 2.3 nDCG. (ii) ChatGPT also shows strong results on BEIR by outperforming baselines trained on large-scale annotated data and only lag behind monoT5 (3B).

3.6 Results on Mr.TyDi

Table 3 lists the results on Mr.TyDi of ten low-resource languages. Overall, GPT-4 performs better than the supervised system in most languages, with an average improvement of 2.65 nDCG over mmarcoCE. However, GPT-4 performs less than mmarcoCE in certain low-resource languages such as Bengali, Telugu, and Thai. This may be attributed to the weaker language modeling ability of GPT-4 in these languages and the fact that text in low-resource languages tends to consume more tokens than English text, leading to the over-cropping

Table 2: **Results (nDCG@10) on BEIR.** Best performing unsupervised and overall system(s) are marked bold.

Method	Covid	NFCorpus	Touche	DBPedia	SciFact	Signal	News	Robust04	Avg
BM25	59.47	30.75	**44.22**	31.80	67.89	33.05	39.52	40.70	43.42
Supervised									
monoBERT (340M)	70.01	36.88	31.75	41.87	71.36	31.44	44.62	49.35	47.16
monoT5 (220M)	78.34	37.38	30.82	42.42	73.40	31.67	46.83	51.72	49.07
monoT5 (3B)	80.71	**38.97**	32.41	44.45	**76.57**	32.55	48.49	56.71	51.36
TART-Rerank (FLAN-T5-XL)	75.10	36.03	27.46	42.53	74.84	25.84	40.01	50.75	46.57
Unsupervised									
UPR (FLAN-T5-XL)	68.11	35.04	19.69	30.91	72.69	31.91	43.11	42.43	42.99
InPars (monoT5-3B)†	78.35	-	-	-	-	-	-	-	-
Promptagator++ (zero-shot)†	76.0	36.0	27.8	41.3	73.6	-	-	-	-
Promptagator++ (few-shot)†	76.2	37.0	38.1	43.4	73.1	-	-	-	-
LLM API (Permutation generation)									
`gpt-3.5-turbo`	76.67	35.62	36.18	44.47	70.43	32.12	48.85	50.62	49.37
+ `gpt-4` Rerank‡	**85.51**	**38.47**	**38.57**	**47.12**	**74.95**	**34.40**	**52.89**	**57.55**	**53.68**

† InPars re-ranks the top-1000 BM25 passages, while Promptagator++ re-ranks the top-200 Promptagator retrieved passages. The results for these two models are cited from original papers. All other models re-rank the same BM25 top-100 passages.
‡ 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.

Table 3: **Results (nDCG@10) on Mr.TyDi.** Note that we use the first 100 sample in test set for evaluation. mmarcoCE denotes a mMiniLM-based cross-encoder trained on mmarco (multilingual MS MARCO) datasets (Bonifacio et al., 2021). Best performing system(s) are marked bold.

Method	BM25	mmarcoCE	`gpt-3.5`	`+gpt-4`
Arabic	39.19	68.18	71.00	**72.56**
Bengali	45.56	**65.98**	53.10	64.37
Finnish	29.91	54.15	56.48	**62.29**
Indonesian	51.79	69.94	68.45	**75.47**
Japanese	27.39	49.80	50.70	**58.22**
Korean	26.29	44.00	41.48	**49.63**
Russian	34.04	53.16	48.75	**53.45**
Swahili	45.15	60.31	62.38	**67.67**
Telugu	37.05	**68.92**	51.69	62.22
Thai	44.62	**68.36**	55.57	63.41
Avg	38.10	60.28	55.96	**62.93**

- We use `gpt-4` to re-rank the top-30 passages re-ranked by `gpt-3.5-turbo`.

of passages. Similar trends are observed with ChatGPT, which is on par with the supervised system in most languages, lags behind in some low-resource languages, and consistently trails behind GPT-4 in all languages.

3.7 Ablation Study

We conducted an ablation study on TREC to gain insights into the detailed configuration of permutation generation methods. Table 4 illustrates the results. First, we investigated the inference of the number of passages to be re-ranked. As seen from the results of variants (1-3), re-ranking more passages resulted in performance gains. Next, we studied the inference of the initial passage order. While our standard implementation utilizes the ranking result of BM25 as the initial order, we examined two alternative variants: random order (4) and reversed BM25 order (5). The results reveal that the model performance is highly sensitive to the initial passage order. This could be because BM25 provides a relatively good starting passage order, enabling satisfactory results with only a single sliding window re-ranking. Furthermore, we studied the inference of the number of sliding window passes. Models (6-7) in Table 4 show that re-ranking more times may improve nDCG@10, but it somehow hurt the ranking performance on top passages (e.g., nDCG@1 decreased by 3.88). Finally, re-ranking the top 30 passages using GPT-4 showed notable accuracy improvements (see the model (8)). This provides an alternative method to combine ChatGPT and GPT-4 in passage re-ranking to reduce the high cost of using the GPT-4 model.

We analyze the model's unexpected behavior in Appendix C, and the cost of API in Appendix D.

4 Distillation to Specialized Models

Although ChatGPT and GPT-4 are highly capable, they are also expensive. As a result, there is increasing research focused on distilling the capabilities of ChatGPT into specialized models. For instance, Fu et al. (2023) and Magister et al. (2022) have

Table 4: **Ablation study on TREC-DL19.** We use `gpt-3.5-turbo` with permutation generation with different configuration.

	Method	nDCG@1	nDCG@5	nDCG@10
	BM25	54.26	52.78	50.58
	`gpt-3.5-turbo`	82.17	71.15	65.80
	Number of re-ranked passages			
(1)	Re-rank top-50	77.13	67.93	64.50
(2)	Re-rank top-30	74.81	66.87	62.01
(3)	Re-rank top-20	75.58	66.19	60.89
	Initial passage order			
(4)	Random order	26.36	25.32	25.17
(5)	Reverse order	36.43	31.79	32.77
	Number of re-ranking			
(6)	Re-rank 2 times	78.29	69.37	66.62
(7)	Re-rank 3 times	78.29	69.74	66.97
(8)	Re-rank top-30 by `gpt-4`	80.23	76.70	73.64

distilled the reasoning ability of LLMs into smaller models. Self-instruct (Wang et al., 2022) and Alpaca (Taori et al., 2023) describe an iterative approach to distill GPT-3 using its own output. Likewise, recent studies (Gilardi et al., 2023) demonstrate that ChatGPT outperforms crowd-worker annotators on text-annotation tasks.

In this paper, we present a novel permutation distillation method that aims to distill the passage re-ranking capability of ChatGPT into a specialized model. The key difference between our approach and previous methods is that we directly use the model-generated permutation as the target, without introducing any inductive bias such as consistency-checking or log-probability manipulation (Bonifacio et al., 2022; Sachan et al., 2022b). To achieve this, we randomly sample 10K queries from MS MARCO and retrieve 20 candidate passages using BM25 for each query. The objective of distillation aims to reduce the differences between the permutation outputs of the student and teacher models.

4.1 Training

Formally, suppose we have a query q and M passages $(p_1, \ldots, p_M)$ retrieved by BM25 ($M = 20$ in our implementation). ChatGPT with instructional permutation generation could produce the ranking results of the M passages, denoted as $R = (r_1, \ldots, r_M)$, where $r_i \in [1, 2, ..., M]$ is the rank of the passage p_i. For example, $r_i = 3$ means p_i ranks third among the M passages. Now we have a specialized model $s_i = f_\theta(q, p_i)$ with parameters θ to calculates relevance score s_i of paired (q, p_i) using a cross-encoder. Using the permutation R generated by ChatGPT, we consider the following losses to optimize the student model:

Listwise Cross-Entropy (CE) (Bruch et al., 2019)**.** Listwise CE is the wide-use loss for passage ranking, which considers only one positive passage and defines the list-wise softmax cross-entropy on all candidate's passages:

$$\mathcal{L}_{\text{Listwise\_CE}} = -\sum_{i=1}^{M} \mathbb{1}_{r_i=1} \log\left(\frac{\exp(s_i)}{\sum_j \exp(s_j)}\right)$$

where $\mathbb{1}$ is the indicator function.

RankNet (Burges et al., 2005)**.** RankNet is a pairwise loss that measures the correctness of relative passage orders:

$$\mathcal{L}_{\text{RankNet}} = \sum_{i=1}^{M} \sum_{j=1}^{M} \mathbb{1}_{r_i<r_j} \log(1 + \exp(s_i - s_j))$$

when using permutation generated by ChatGPT, we can construct $M(M-1)/2$ pairs.

LambdaLoss (Wang et al., 2018)**.** The LambdaLoss further accounts for the nDCG gains of the model ranks. LambdaLoss uses the student model's rank, denoted as $\pi = (\pi_1, \ldots, \pi_M)$, where π_i is the model predicted rank of p_i with a similar definition with ChatGPT rank R. The loss function is defined as:

$$\mathcal{L}_{\text{Lambda}} = \sum_{r_i<r_j} \Delta\text{NDCG} \log_2(1 + \exp(s_i - s_j))$$

in which ΔNDCG is the delta of NDCG which could be compute as $\Delta\text{NDCG} = |G_i - G_j||\frac{1}{D(\pi_i)} - \frac{1}{D(\pi_j)}|$, where $D(\pi_i)$ and $D(\pi_j)$ are the position discount functions and G_i and G_j are the gain functions used in NDCG (Wang et al., 2018).

Pointwise Binary Cross-Entropy (BCE). We also include the Pointwise BCE as the baseline loss for supervised methods, which is calculated based on each query-document pair independently:

$$\mathcal{L}_{\text{BCE}} = -\sum_{i=1}^{M} \mathbb{1}_{r_i=1} \log \sigma(s_i) + \mathbb{1}_{r_i\neq 1} \log \sigma(1 - s_i)$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the logistic function.

4.2 Implementation

We initialize the parameters of the student model with DeBERTa-v3-base[6]. To calculate the relevance score, we use a cross-encoder architecture

[6] `https://huggingface.co/microsoft/deberta-v3-base`

Table 5: **Results (nDCG@10) of specialized models.** Best performing specialized model(s) are marked bold. The label column denotes the relevance judgements used in model training, where MARCO denotes use MS MARCO annotation, ChatGPT denotes use the outputs of permutation generation instructed ChatGPT as labels. BEIR (Avg) denotes average nDCG on eight BEIR datasets, and the detailed results are at Table 8.

Label	Method / Loss	DL19	DL20	BEIR (Avg)
∅	BM25	50.58	47.96	43.42
∅	ChatGPT	65.80	62.91	49.37
MARCO	Pointwise BCE	65.57	56.72	39.43
MARCO	Listwise CE	65.99	57.97	35.45
MARCO	RankNet	66.34	58.51	38.79
MARCO	LambdaLoss	64.82	56.16	40.62
ChatGPT	Listwise CE	65.39	58.80	47.74
ChatGPT	RankNet	66.56	59.43	**50.53**
ChatGPT	LambdaLoss	**67.17**	**60.56**	49.77

similar to monoBERT that concatenates the query and passage with a `[SEP]` token and uses the representation of the `[CLS]` token. To generate candidate passages, we randomly sample 10k queries and use BM25 to retrieve 20 passages for each query. We then re-rank the candidate passages using the `gpt-3.5-turbo` API with permutation generation instruction, at the cost of approximately $40. During training, we employ a batch size of 32 and utilize the AdamW optimizer with a constant learning rate of $5e-5$. The model is trained for two epochs. Additionally, we implement models using the original MS MARCO labels for comparison.

4.3 Results

Table 5 lists the results of specialized models, and Table 8 includes the detailed results. Our findings can be summarized as follows: (i) Models trained with ChatGPT demonstrate competitive or superior performance compared to supervised models on in-domain data TREC, while significantly outperforming them on BEIR datasets by over 10 nDCG on average. This may be due to the more accurate and comprehensive relevance judgments of ChatGPT compared to MA MARCO labels, which allows the model to avoid overfitting shortcut patterns (Arabzadeh et al., 2021). (ii) RankNet and LambdaLoss outperform Listwise CE, probably because they can better utilize the permutation generated by ChatGPT, whereas Listwise CE only utilizes the top-ranked passage as the single positive. (iii) The specialized model trained with RankNet loss achieves an average nDCG of 50.53 on BEIR, outperforming monoBERT and monoT5 (220M) by 3.37 and 1.53, respectively. This result exhibits the potential of distilling LLMs for IR, as the distilled models are trained using only 10K samples, costing approximately $40 for API calls, and without human annotation. This approach is much more cost-efficient than previous systems that have been trained on over 400K annotated data.

5 Conclusion

In this paper, we conduct a comprehensive study on passage re-ranking with instruction-following LLMs. Besides the institutional query generation and relevance generation methods, we introduce a novel permutation generation approach to fully explore the power of LLMs. Our experiments on three benchmarks have demonstrated the capability of ChatGPT and GPT-4 in passage re-ranking. Furthermore, we propose a permutation distillation method and show its advantages over existing supervised approaches in terms of effectiveness and efficiency.

For future work, we suggest some promising directions on LLMs for IR: (1) LLMs as relevance annotators. Data labeling is quite expensive in IR, and as our pilot experiments have demonstrated the effectiveness of the relevance judgments generated by LLMs, we believe it deserves further exploration. (2) Instruction-tuning LLMs for a universal information access system. Instruction-tuning LLMs for diverse ranking tasks, such as passage ranking, entity ranking, response ranking, evidence ranking and etc., has great potential toward a more powerful, universal information access system. (3) End-to-end IR model. Existing multi-stage IR systems always follow a "index-retrieve-rank" pipeline, and the separation of different components makes it hard for end-to-end learning. Considering the remarkable performance of LLMs, it's possible to use only one LLM covering every component in the IR system, such as retrieval and ranking. (4) Improving the efficiency of LLMs. Though effective, current LLMs generally have hundreds of billions of parameters, and deploying them to real industrial scenarios is prohibitively expensive. Thus, improving the efficiency of LLMs, such as reducing to small models, and lightweight learning, is very worthy of further exploration.

References

Negar Arabzadeh, Alexandra Vtyurina, Xinyi Yan, and Charles L. A. Clarke. 2021. Shallow pooling for sparse labels. *Information Retrieval Journal*, 25:365 – 385.

Akari Asai, Timo Schick, Patrick Lewis, Xilun Chen, Gautier Izacard, Sebastian Riedel, Hannaneh Hajishirzi, and Wen tau Yih. 2022. Task-aware retrieval with instructions. *ArXiv*, abs/2211.09260.

Luiz Henrique Bonifacio, Hugo Queiroz Abonizio, Marzieh Fadaee, and Rodrigo Nogueira. 2022. Inpars: Data augmentation for information retrieval using large language models. In *SIGIR*.

Luiz Henrique Bonifacio, Israel Campiotti, Roberto de Alencar Lotufo, and Rodrigo Nogueira. 2021. mmarco: A multilingual version of ms marco passage ranking dataset. *ArXiv*, abs/2108.13897.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*.

Sebastian Bruch, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2019. An analysis of the softmax cross entropy loss for learning-to-rank with binary relevance. In *SIGIR*.

Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. 2005. Learning to rank using gradient descent.

Daniel Fernando Campos, Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, Li Deng, and Bhaskar Mitra. 2016. Ms marco: A human generated machine reading comprehension dataset. *ArXiv*, abs/1611.09268.

Hyung Won Chung, Le Hou, S. Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Wei Yu, Vincent Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed Huai hsin Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *ArXiv*, abs/2210.11416.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Fernando Campos, and Ellen M. Voorhees. 2020. Overview of the trec 2020 deep learning track. *ArXiv*, abs/2102.07662.

Zhuyun Dai, Vincent Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith B. Hall, and Ming-Wei Chang. 2023. Promptagator: Few-shot dense retrieval from 8 examples. In *ICLR*.

Yixing Fan, Xiaohui Xie, Yinqiong Cai, Jia Chen, Xinyu Ma, Xiangsheng Li, Ruqing Zhang, and Jiafeng Guo. 2021. Pre-training methods in information retrieval. *ArXiv*, abs/2111.13853.

Yao Fu, Hao-Chun Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. 2023. Specializing smaller language models towards multi-step reasoning. *ArXiv*, abs/2301.12726.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2022. Precise zero-shot dense retrieval without relevance labels. *ArXiv*, abs/2212.10496.

Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. 2023. Chatgpt outperforms crowd-workers for text-annotation tasks. *ArXiv*, abs/2303.15056.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022a. Towards unsupervised dense information retrieval with contrastive learning. *TMLR*.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane A. Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022b. Few-shot learning with retrieval augmented language models. *ArXiv*, abs/2208.03299.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher R'e, Diana Acosta-Navas, Drew A. Hudson, E. Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel J. Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan S. Kim, Neel Guha, Niladri S. Chatterji, O. Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas F. Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2022. Holistic evaluation of language models. *ArXiv*, abs/2211.09110.

Jimmy J. Lin, Rodrigo Nogueira, and Andrew Yates. 2020. Pretrained transformers for text ranking: Bert and beyond. *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. 2022. Teaching small language models to reason. *ArXiv*, abs/2212.08410.

Microsoft. 2023. Confirmed: the new bing runs on openai's gpt-4. https://blogs.bing.com/search/march_2023/Confirmed-the-new-Bing-runs-on-OpenAI%E2%80%99s-GPT-4.

Niklas Muennighoff. 2022. Sgpt: Gpt sentence embeddings for semantic search. *ArXiv*, abs/2202.08904.

Reiichiro Nakano, Jacob Hilton, S. Arun Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. Webgpt: Browser-assisted question-answering with human feedback. *ArXiv*, abs/2112.09332.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *ArXiv*, abs/1901.04085.

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document ranking with a pre-trained sequence-to-sequence model. In *Findings of EMNLP*.

OpenAI. 2022. Introducing chatgpt. https://openai.com/blog/chatgpt.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*, abs/2303.08774.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

Devendra Singh Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen tau Yih, Joëlle Pineau, and Luke Zettlemoyer. 2022a. Improving passage retrieval with zero-shot question generation. In *EMNLP*.

Devendra Singh Sachan, Mike Lewis, Dani Yogatama, Luke Zettlemoyer, Joëlle Pineau, and Manzil Zaheer. 2022b. Questions are all you need to train a dense passage retriever. *ArXiv*, abs/2206.10658.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen tau Yih. 2023. Replug: Retrieval-augmented black-box language models. *ArXiv*, abs/2301.12652.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. 2022. Transformer memory as a differentiable search index. In *NeurIPS*.

Nandan Thakur, Nils Reimers, Andreas Ruckl'e, Abhishek Srivastava, and Iryna Gurevych. 2021. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. In *NeurIPS*, volume abs/2104.08663.

Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2018. The lambdaloss framework for ranking metric optimization. In *CIKM*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language model with self generated instructions. *ArXiv*, abs/2212.10560.

W. Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. 2023. Generate rather than retrieve: Large language models are strong context generators. In *ICLR*.

Xinyu Zhang, Xueguang Ma, Peng Shi, and Jimmy J. Lin. 2021. Mr. tydi: A multi-lingual benchmark for dense retrieval. In *MRL*.

A Instructions

A.1 Query Generation Instruction

The query generation instruction (Sachan et al., 2022a) uses the log-probability of the query.

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

A.2 Relevance Generation Instruction

Following HELM (Liang et al., 2022), the relevance generation instruction use 4 in-context examples.

> Given a passage and a query, predict whether the passage includes an answer to the query by producing either 'Yes' or 'No'.
>
> Passage: Its 25 drops per ml, you guys are all wrong. If it is water, the standard was changed 15 - 20 years ago to make 20 drops = 1mL. The viscosity of most things is temperature dependent, so this would be at room temperature. Hope this helps.
> Query: how many eye drops per ml
> Does the passage answer the query?
> Answer: Yes
>
> Passage: RE: How many eyedrops are there in a 10 ml bottle of Cosopt? My Kaiser pharmacy insists that 2 bottles should last me 100 days but I run out way before that time when I am using 4 drops per day.In the past other pharmacies have given me 3 10-ml bottles for 100 days.E: How many eyedrops are there in a 10 ml bottle of Cosopt? My Kaiser pharmacy insists that 2 bottles should last me 100 days but I run out way before that time when I am using 4 drops per day.
> Query: how many eye drops per ml
> Does the passage answer the query?
> Answer: No
>
> Passage: : You can transfer money to your checking account from other Wells Fargo. accounts through Wells Fargo Mobile Banking with the mobile app, online, at any. Wells Fargo ATM, or at a Wells Fargo branch. 1 Money in — deposits.
> Query: can you open a wells fargo account online
> Does the passage answer the query?
> Answer: No
>
> Passage: You can open a Wells Fargo banking account from your home or even online. It is really easy to do, provided you have all of the appropriate documentation. Wells Fargo has so many bank account options that you will be sure to find one that works for you. They offer free checking accounts with free online banking.
> Query: can you open a wells fargo account online
> Does the passage answer the query?
> Answer: Yes
>
> Passage: {{passage}}
> Query:{{query}}
> Does the passage answer the query?
> Answer:

A.3 Permutation Generation Instruction (Text)

Permutation generation (text) is used for `text-davinci-003`.

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:

A.4 Permutation Generation Instruction (Chat)

Permutation generation instruction (chat) is 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.

B Related Work

B.1 Information Retrieval with LLMs

Recently, large language models (LLMs) have found increasing applications in information retrieval. Several approaches have been proposed to utilize LLMs for passage retrieval. For example, SGPT (Muennighoff, 2022) generates text embeddings using GPT, DSI (Tay et al., 2022) proposes a differentiable search index, and HyDE (Gao et al., 2022) generates pseudo-documents using GPT-3. In addition, LLMs have also been used for passage re-ranking tasks. UPR (Sachan et al., 2022a) and SGPT-CE (Muennighoff, 2022) introduce instructional query generation methods, while HELM (Liang et al., 2022) utilizes instruction relevance generation. LLMs are also employed for training data generation. InPars (Bonifacio et al., 2022) generates pseudo-queries using GPT-3, and Promptagator (Dai et al., 2023) proposes a few-shot dense retrieval to leverage a few demonstrations from the target domain for pseudo queries generation. Furthermore, LLMs have been used for content generation (Yu et al., 2023) and web browsing (Nakano et al., 2021; Izacard et al., 2022b; Microsoft, 2023).In this paper, we explore using ChatGPT and GPT-4 in passage re-ranking tasks, propose an instructional permutation generation method, and conduct a comprehensive evaluation of benchmarks from various domains, tasks, and languages.

B.2 LLMs Distillation

Despite their impressive capabilities, LLMs such as GPT-4 often come with high costs and lack open-source availability. As a result, considerable research has explored ways to distill the capabilities of LLMs into specialized, custom models. For instance, Fu et al. (2023) and Magister et al. (2022) have successfully distilled the reasoning ability of LLMs into smaller models. Self-instruct (Wang et al., 2022) and Alpaca (Taori et al., 2023) propose iterative approaches to distill GPT-3 using their outputs. Additionally, Sachan et al. (2022b) and Shi et al. (2023) utilize the generation probability of LLMs to improve retrieval systems. This paper presents a permutation distillation method that leverages ChatGPT as a teacher to obtain specialized re-ranking models. Our experiments demonstrate that even with a small amount of ChatGPT-generated data, the specialized model can outperform strong supervised systems.

Table 6: **Analysis of model stability on TREC.** *Repetition* refers to the number of times the model generates duplicate passage identifiers. *Missing* refers to the number of missing passage identifiers in model output. *Rejection* refers to the number of times the model rejects to perform the ranking. *RBO*, i.e., rank biased overlap, refers to the consistency of the model in ranking the same group of passages twice.

Method	Repetition↓	Missing↓	Rejection	RBO↑
`text-davinci-003`	**0**	280	0	72.30
`gpt-3.5-turbo`	14	153	7	81.49
`gpt-4`	**0**	**1**	11	**82.08**

C Model Behavior Analysis

In the permutation generation method, the ranking of passages is determined by the list of model-output passage identifiers. However, we have observed that the models do not always produce the desired output,

as evidenced by occasional duplicates or missing identifiers in the generated text. In Table 6, we present quantitative results of unexpected model behavior observed during experiments with the GPT models.

Repetition. The repetition metric measures the occurrence of duplicate passage identifiers generated by the model. The results indicate that ChatGPT produced 14 duplicate passage identifiers during re-ranking 97 queries on two TREC datasets, whereas `text-davinci-003` and GPT-4 did not exhibit any duplicates.

Missing. We conducted a count of the number of times the model failed to include all passages in the re-ranked permutation output[7]. Our findings revealed that `text-davinci-003` has the highest number of missing passages, totaling 280 instances. ChatGPT also misses a considerable number of passages, occurring 153 times. On the other hand, GPT-4 demonstrates greater stability, with only one missing passage in total. These results suggest that GPT-4 has higher reliability in generating permutations, which is critical for effective ranking.

Rejection. We have observed instances where the model refuses to re-rank passages, as evidenced by responses such as "*None of the provided passages are directly relevant to the query ...*". To quantify this behavior, we count the number of times this occurred and find that GPT-4 reject ranking the most frequently, followed by ChatGPT, while the Davinci model never refused to rank. This finding suggests that chat LLMs tend to be more adaptable compared to completion LLMs, and may exhibit more subjective responses. Note that we do not explicitly prohibit the models from rejecting ranking in the instructions, as we find that it does not significantly impact the overall ranking performance.

RBO. The sliding windows strategy involves re-ranking the top-ranked passages from the previous window in the next window. The models are expected to produce consistent rankings in two windows for the same group of passages. To measure the consistency of the model's rankings, we use RBO (rank biased overlap[8]), which calculates the similarity between the two ranking results. The findings turn out that ChatGPT and GPT-4 are more consistent in ranking passages compared to the Davinci model. GPT-4 also slightly outperforms ChatGPT in terms of the RBO metric.

Table 7: Average token cost, number API request, and $USD per query on TREC.

API	Instruction	Tokens	Requests	$USD
`text-curie-001`	Relevance generation	52,970	100	0.106
`text-curie-001`	Query generation	10,954	100	0.022
`text-davinci-003`	Query generation	11,269	100	0.225
`text-davinci-003`	Permutation generation	17,370	10	0.347
`gpt-3.5-turbo`	Permutation generation	19,960	10	0.040
`gpt-4`	Permutation generation	19,890	10	0.596
- rerank top-30	Permutation generation	3,271	1	0.098

D API Cost

In Table 7, we provide details on the average token cost, API request times, and USD cost per query. In terms of average token cost, the relevance generation method is the most expensive, as it requires 4 in-context demonstrations. On the other hand, the permutation generation method incurs higher token costs compared to the query generation method, as it involves repeated processing of passages in sliding windows. Regarding the number of requests, the permutation generation method requires 10 requests for sliding windows, while other methods require 100 requests for re-ranking 100 passages. In terms of average USD cost, GPT-4 is the most expensive, with a cost of $0.596 per query. However, using GPT-4 for re-ranking the top-30 passages can result in significant cost savings, with a cost of $0.098 per query for GPT-4 usage, while still achieving good results. As a result, we only utilize GPT-4 for re-ranking

[7] In our implementation, we append the missing passages in their original order at the end of the re-ranked passages.
[8] https://github.com/changyaochen/rbo

the top 30 passages of ChatGPT on BEIR and Mr.TyDi. The total cost of our experiments with GPT-4 amounts to $556.

Table 8: Results (nDCG@10) on TREC and BEIR. Best performing specialized and overall system(s) are marked bold. The specialized DeBERTa models are fine-tined on 10K queries using relevance judgements from MARCO or ChatGPT.

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 *train on 400K MRACO labels*											
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
Unsupervised *instructional 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` rerank	**75.59**	**70.56**	**85.51**	38.47	38.57	**47.12**	74.95	**34.40**	**52.89**	**57.55**	**53.68**
Specialized DeBERTa Models *train on 10K MARCO labels or ChatGPT predicted permutations*											
MARCO Pointwise BCE	65.57	56.72	70.82	33.10	17.08	32.28	55.37	19.30	41.52	46.00	39.43
MARCO Listwise CE	65.99	57.97	66.31	32.61	20.15	30.79	37.57	18.09	38.11	39.93	35.45
MARCO RankNet	66.34	58.51	70.29	34.23	20.27	29.62	49.01	23.22	39.82	43.87	38.79
MARCO LambdaLoss	64.82	56.16	72.86	34.20	19.51	32.55	51.88	26.22	42.47	45.28	40.62
ChatGPT Listwise CE	65.39	58.80	76.29	35.73	38.19	40.24	64.49	31.37	47.61	48.00	47.74
ChatGPT RankNet	65.75	59.34	**81.26**	36.57	**39.03**	42.10	**68.77**	31.55	**52.54**	**52.44**	**50.53**
ChatGPT LambdaLoss	**67.17**	**60.56**	80.63	**36.74**	36.73	**43.75**	68.21	**32.58**	49.00	50.51	49.77

E Results of Specialized Models

Table 8 lists the detailed results of specialized models on TREC and BEIR.