FIRE-LLM: Efficient Filtering and Listwise Reranking with Large Language Models

No Institute Given

Abstract. Large language models (LLMs) have shown significant promise in enhancing information retrieval (IR) pipelines, particularly in reranking tasks. However, existing reranking approaches, e.g. listwise reranking, are often hampered by high computational costs and limited scalability due to sequential, non-parallelizable inference steps. Here we propose **Filter and Rerank Efficiently with LLM (FIRE-LLM)**, a novel three-stage retrieval pipeline that combines: (i) a fast sparse retrieval stage using BM25, (ii) an LLM-based filtering step to reduce candidate passages, and (iii) a single-pass listwise reranking stage. Experiments conducted on TREC Deep Learning Track 2019-2020 benchmarks demonstrated that FIRE-LLM could achieve competitive performance compared to state-of-the-art methods like RankGPT, while reducing the inference time and energy consumption by up to **14**× for large candidate sets. These efficiency gains make FIRE-LLM particularly suitable for large-scale and real-time retrieval applications. Our analysis of FIRE-LLM applications highlights the balance between performance, efficiency and environmental sustainability, indicating that this pipeline provides a practical framework for optimizing LLM-based retrieval systems.

Keywords: information retrieval · carbon analysis · large language model

1 Introduction

Recently, large language models (LLMs) have been widely used in information retrieval (IR), thereby enhancing four key modules: query rewriting, retrieval, reranking and reader [15]. **Query rewriting**, early in the IR pipeline, uses LLM generation capabilities for improved precision and expressiveness through query refinement. **Retrieval** has transitioned from conventional BM25 [11] to neural models projecting queries and passages into high-dimensional vector spaces, hence benefiting from the semantic understanding of LLMs while maintaining efficiency. **Reranking**, the task of refining the passage order, is shifting towards complex matching methods beyond simple vector inner products. LLMs enable simultaneous comparison of multiple passages, thus greatly enhancing the reranking quality by direct confrontation. **Reader**, which intuitively organizes answer texts, leverages LLMs to gain insight into user intent and dynamically generate responses, thereby revolutionizing results presentation.

Building on these advances, we specifically focus on reranking, exploring the potential of LLMs as zero-shot rerankers [2,9,16]. Reranking with LLMs is

typically approached through three paradigms. **Pointwise reranking** treats each passage-query pair independently, with relevance measured either by directly generating relevance labels or scores, or by estimating the likelihood of generating the query from a given passage. **Pairwise reranking** compares pairs of passages utilizing efficient sorting algorithms (e.g., heapsort or bubble sort) based on pairwise relevance comparisons. **Listwise reranking** evaluates a whole list of passages at once. The main benefits of this approach are the ability to capture interactions between many passages, and the reduced computational cost as several passages are processed at once.

Here we propose to use the listwise reranking paradigm due to its widely proven efficacy [6,12,17]. We experimentally demonstrate the inefficiency of reranking large set of passages (>30-40) at once. Actually, most state-of-the-art methods use the so-called 'sliding window' strategy which involves reranking a set of passages in smaller chunks in an iterative fashion so as to cover the whole set. This method is more reliable and efficient but—due to multiple non-parallelizable LLM inferences—it has the drawback of being computationally costly.

Here we present our novel approach called Filter and Rerank Efficiently with LLM (FIRE-LLM). We have incorporated a filtering stage before the reranking operation so as to be able to reduce large sets of passages into a manageable sized list for single inference reranking. This stage also relies on LLMs to filter out irrelevant passages from the initial retrieval set. This approach leads to greater results than direct reranking, i.e. nearing the performance of state-of-the-art sliding window strategies. Above all, our method is much more efficient with respect to inference time and energy consumption than any sliding window strategy, hence making it a primary choice for real-world applications.

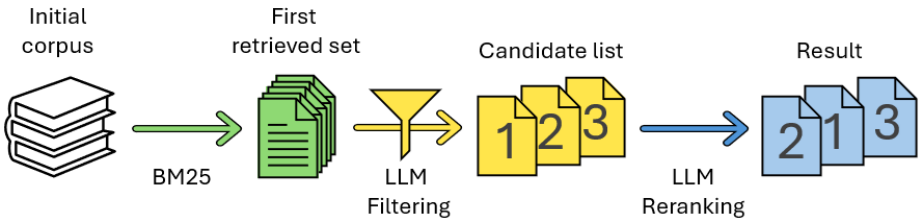

Fig. 1: Illustration of our three-stage FIRE-LLM retrieval pipeline.

Figure 1 illustrates our retrieval pipeline. The fast retrieval stage reduces the very high numbers of passages to manageable sizes like 100 or 200. This set is then filtered by the LLM into a small list of 20 passages, for example. Lastly, the LLM reranks the list to obtain the final order of relevant passages.

We evaluated FIRE-LLM on two widely-used IR benchmark datasets, i.e. the TREC Deep Learning Track 2019 and 2020 [4]. We compared our method against the state-of-the-art sliding window strategy which was re-implemented using the same backbone LLM as our method to ensure a fair comparison. This

allowed precise measurement of the inference time, energy consumption and CO_2 emissions and showcased the substantial gap between the methods. The results demonstrated that, while maintaining comparable reranking performance, FIRE-LLM was 2 to 5 times faster and more energy-efficient for reranking 100 passages, and this advantage increased significantly until reaching ≈14 times faster for 300 passages.

Task definition. Our task can be defined as follows: Given a query q and a corpus of passages P, the system must predict a subset of passages relevant to the query $s(q, P) = \{p_i, \ldots, p_j\}$. It must then rerank the subset according to the relevancy of each passage to the query $R(s, q) = \{p_1, \ldots, p_n\}$. This paradigm is necessary to retrieve information from a fixed trusted corpus. Modern information retrieval approaches tend to rely on LLMs, and dealing with the task as formulated above ensures that the information retrieved comes solely from the trusted corpus, and not from the LLM 'memory'. This paradigm is well suited in contexts where information must be retrieved from curated internal sources, e.g. scientific and medical applications, or from confidential or sensitive documents.

2 Related Work

The recent integration of LLMs into IR pipelines has sparked extensive research, particularly to optimize the reranking phase. We primarily explored two complementary directions among existing approaches: methods that leverage the inherent advantages of listwise reranking, enabling direct comparison of multiple passages, and strategies that implement a preliminary filtering step before reranking, thus reducing computational overhead by limiting the reranking context to the most relevant passages. Below, we review key contributions in both directions to contextualize our proposed methodology.

2.1 Listwise reranking

Listwise reranking has become increasingly popular [6,10,12] because of its effectiveness in taking full advantage of LLM capabilities, particularly by providing means for models to simultaneously analyze and directly compare multiple passages. Here we focus on two influential papers that clearly demonstrate the strengths of the listwise paradigm, which guided our methodological choices.

The authors of [12] demonstrate how to fully exploit LLM capabilities for reranking through a listwise approach. They highlight the impacts of backbone LLMs on the reranking performance, hence necessitatting listwise reranking to fully leverage the latest LLM capabilities. Their so-called RankGPT approach revolves around the use of the sliding window technique. By this strategy—which involves ranking passages back to front—the full pipeline is able to handle any number of previously retrieved passages, while maintaining control over the number of passages processed by the LLM at each step. By defining the window size and the sliding step size, the LLM can rerank the full list of passages via

back-to-first passage iteration, with only a limited number of passages processed at each step. This method allows the LLM to handle large passage sets without exceeding its context window, or without degrading the reranking performance by processing too many passages at once.

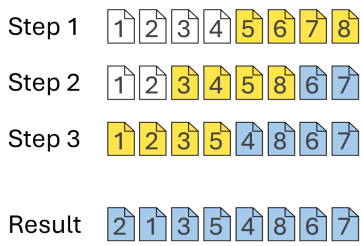

Fig. 2: Sliding window strategy example. The sliding window is represented in yellow with the reranked passages in blue. For a sliding window of size 4 with a step-size of 2, a list of 8 passages will be reranked in 3 steps.

Figure 2 illustrates a case of reranking 8 passages with a sliding window of size 4 and a step size of 2. The LLM processes the last 4 passages and then, after a list update, the window shifts frontwards by 2 passages, and the LLM processes those next 4 passages, and so on until all passages have been processed. Listwise reranking has two limitations: (1) The performance is highly sensitive to the passage order in the prompt, with a substantial gap between the best and the worst order. (2) When using sliding windows, multiple iterations are required to process all passages, which can be computationally expensive. This is generally unsuitable for real-time applications where rapid response times are crucial as the iterations cannot be parallelized due to the sequential nature of sliding windows.

The authors of [6] conducted experiments to demonstrate the superiority of listwise reranking with LLMs over pointwise reranking with and without LLMs in a zero-shot setting. They compared a pretrained model for pointwise reranking, and both pointwise and listwise approaches using GPT-3 as the backbone model. The comparisons were focused on the task of reranking the first 100 passages retrieved by BM25 on various datasets. For these experiments, they also used a sliding window strategy to manage all passages in their listwise reranking. They obtained better results for all three datasets with the listwise approach. They then conducted experiments using the listwise approach on a single list of 10 and 20 passages previously retrieved by the other pointwise methods. The results showed that this intermediate step, combined with a single listwise reranking pass, generated better results than the other methods.

These influential listwise approaches guided our work, but we introduce a novel strategy designed to overcome the computational constraints of the sliding window technique, thereby enabling efficient scalable reranking without negatively impacting the performance.

2.2 Filtering before reranking

The strategy of introducing a filtering stage before reranking, although promising for reducing computational costs, has only been assessed in a few recent studies. In the following subsection, we present key contributions from these studies to highlight the relevance and potential of this under-explored approach.

The authors of [7] proposed empirical experiments to compare various list truncation strategies before the reranking stage. They showed that supervised methods did not outperform unsupervised methods, and that the outcomes obtained with fixed reranking depths were close to matching the effectiveness/efficiency trade-offs achieved by other methods. They also observed that the type of initial retriever had a marked impact on the optimal truncation depth, and that effective retrievers allowed for much shallower truncation depths, thereby maintaining an excellent effectiveness/efficiency trade-off.

The authors of [8] proposed a prefiltering stage before reranking to improve the reranking performance. They showed that this stage considerably improved the reranking performance by reducing the number of false positives in the reranking process. They used an LLM to produce a relevance score for each passage, according to the query, from 0 to 1, where 0 means a completely irrelevant passage and 1 is a fully relevant one. They then used a threshold to filter out passages with a score below this threshold. They sought the optimal threshold using qrels from each dataset (as expert annotation), and through an iterative process, they adjusted the threshold to maximize the f1 score of the LLM output on the qrels.

Our approach similarly builds on these preliminary filtering methods by leveraging their demonstrated performance benefits to effectively offload the computational burden of the subsequent reranking stage, thus optimizing the entire retrieval pipeline.

3 Methodology

Like many authors [6,12,15], we aim to leverage the strengths of LLMs in information retrieval tasks while mitigating their computational costs. Our proposed methodology FIRE-LLM involves a three-stage retrieval pipeline: fast initial retrieval, LLM-based filtering and LLM-based reranking. This strategy enables us to benefit from the trade-off between the computational efficiency of conventional retrieval methods and the advanced reasoning capabilities of LLMs, thereby resulting in a high-performance retrieval pipeline. Our goal is to achieve performance metrics comparable to those of state of the art inline reranking methods, while considerably reducing the number of LLM calls. This reduction is crucial for practical applications as it directly lowers the overall computational cost and environmental footprint of the retrieval process.

First-stage fast retrieval. BM25 [11], the well known efficient sparse retrieval method, is implemented in the first stage. The initial retrieval stage must fulfill

three critical requirements to ensure optimal pipeline performance. First, it must process large passage collections with sufficient speed and efficiency to maintain practical utility in real-world applications. Second, the method must effectively capture semantic relationships between queries and passages to ensure that relevant passages are not prematurely discarded. Finally, the retrieval process must generate a candidate set that is compact enough to fit in the context windows of modern LLMs, thus enabling efficient subsequent processing. This stage is the most commonly used in IR pipelines to compare reranking methods.

LLM-based filtering stage. The second stage implements a filtering mechanism using an LLM to remove irrelevant passages from the initial retrieval set. This intermediate filtering stage serves as a crucial quality control mechanism between the initial retrieval and the final ranking. The filtering stage reduces noise in the passage set, which enhances the system precision while maintaining high recall rates. The LLM evaluates the relevance of each passage to the query in a single pass, thereby enabling rapid and efficient filtering of the candidate set.

We directly input all the previously retrieved passages into the context window of the LLM in dictionary format. All texts are thereby associated with a single id and are provided to the model, encapsulated with instructions, and we then append the query text with a third instruction. Those instructions prompt the model to carefully read the passages, compare them with the query and output a fixed number of ids. With proper formatting, our pipeline can parse the model output as a list of ids, and retrieve the associated passages prior to channelling them into the last pipeline stage. We also ask the model to simultaneously allocate a relevancy score to each chosen id. This gives us a preliminary order for the output list that will guide the last reranking stage.

Our experiments demonstrated that incorporating this filtering stage between initial retrieval and final reranking markedly enhances the overall system performance. This step helps prevent the inline-reranking LLM from being overwhelmed by an excessive number of false positives in its context, as the reranking task is highly sensitive to the context length (number of passages processed). In contrast, the filtering stage can handle a larger number of passages without being burdened by this high sensitivity, as it only needs to determine whether a passage is relevant or not, without ranking them.

LLM-based reranking. The final stage involves inline reranking, with the LLM processing all remaining passages simultaneously in a single pass. This approach represents a major advance over traditional passage-by-passage reranking methods. By enabling direct comparison between passages within the same context window, the system can leverage the LLM's sophisticated understanding of the relative passage importance and thematic relationships. This simultaneous processing approach reduces the total number of required LLM calls, while also enabling more nuanced ranking decisions thanks to the collective analysis of the entire passage set. The inline reranking methodology efficiently harnesses the LLM's knowledge and reasoning capabilities while maintaining computational efficiency.

Here we provide a newly filtered list of passages—also in a dictionary format—associated with single ids. Here again they are encapsulated with instructions and the query is then appended. The instructions are to rerank the passage list in relevancy order according to the query. The model is asked to output the reranked list of passages, while only providing their id. This gives us the final reranked list, whose size depends on the number of passages output by the filtering stage.
Prompt engineering. For both LLM stages, we have developed carefully crafted prompts that prioritize reliability and consistency over maximum performance through extensive prompt engineering. Our approach to prompt design fosters clear, unambiguous instructions that generate structured, easily parsable outputs. This design philosophy ensures robust system behaviour and seamless integration with the rest of the pipeline. By maintaining consistent response patterns, we create a stable system foundation while preserving flexibility to accommodate future prompt optimizations. This approach to prompt engineering reflects our commitment to building practical, deployable systems that balance performance with operational reliability.

4 Experiments and Analysis

We conducted a series of experiments to asses the effectiveness and efficiency of our proposed FIRE-LLM method. We first conducted experiments to demonstrate the ineffectiveness of reranking the whole initial retrieval set in a single pass, even if it fits within the LLM context window. This was done to highlight the need for a sliding window approach or, as we propose, a filtering stage before the reranking process. We then conducted experiments to systematically evaluate the two critical parameters of our retrieval pipeline: (1) the cardinality of the initial passage set retrieved during the first stage, and (2) the filter threshold applied in the subsequent filtering phase. This dual-parameter analysis serves us to highlight the optimal initial retrieval size/filter threshold combinations, and their respective impacts on the performance of the subsequent reranking phase and the computational efficiency of the overall pipeline. We finally compared our FIRE-LLM method to the state-of-the-art RankGPT inline reranking method to showcase the effectiveness of our approach in achieving high quality results while substantially reducing the computational time and costs associated with LLM processing.

The following subsections present our experimental setup and a detailed analysis of the results, thus providing empirical evidence on the efficacy of FIRE-LLM.

4.1 Experimental Setup

Datasets. We assessed our approach against two standard passage retrieval benchmarks: TREC Deep Learning Track 2019 (TREC-DL-2019) and 2020 (TREC-DL-2020) [4]. All experiments were conducted in a zero-shot setting using

the official test queries from both datasets.

Computing infrastructure. All experiments were performed on a computing node consisting of two NVIDIA A100 GPUs (80GB) and eight AMD Milan EPYC 7543 CPUs.

Models. For the initial retrieval stage, we retrieved the top 100 (or 200 or 300) passages using BM25 through `pyserini` [5]. The Qwen2.5 instruction-tuned model with 72 billion parameters (Qwen2.5-72B-Instruct) [13] was used for the subsequent LLM stages. This choice was based on the findings we obtained in small-scale preliminary experiments, i.e. it achieved one of the best performance/size ratios among the open-source models we tested at the time. All LLMs benefited from 8-bit quantization to optimize memory usage and inference speed while maintaining model quality. The models operated with a 32,768 token context window and supported output sequences with up to 8,192 token lengths.

4.2 Results

Direct reranking. First we assessed the performance of directly reranking all sets retrieved from BM25, and highlighted why this method is impractical. We reranked the top 100, 200 and 300 passages retrieved by BM25 using the Qwen2.5-72B-Instruct model by directly inputting the whole set of passages as a list into the LLM context window. We asked it to output a ranked list containing ids of the passages in relevancy order according to the query.

Table 1: Performances of directly reranking all sets retrieved from BM25 based on the NDCG@10 metrics for the TREC-DL-2019 and TREC-DL-2020 datasets.

Methods		TREC-DL-2019	TREC-DL-2020
Model	BM25 k	NDCG@10	NDCG@10
BM25	100	50.58	47.96
Qwen2.5-72B-Instruct	100	66.71	63.97
Qwen2.5-72B-Instruct	200	63.25	61.82
Qwen2.5-72B-Instruct	300	64.11	56.09

Table 1 shows the performance of this direct reranking method based on both the TREC-DL-2019 and TREC-DL-2020 datasets. The results revealed a marked improvement over BM25, but they were not on par with the performance of state-of-the-art inline reranking methods (shown in Table 3). We also noted a drop in performance when increasing the number of passages retrieved from BM25, thereby strongly indicating that the LLM was struggling to process too many passages at once. This is further confirmed by the reliability metrics shown in Table 2.

We measured the reliability of the direct reranking method by analyzing the generated lists and their characteristics. We extracted the following metrics that we considered relevant in our context to evaluate the LLM capacity to generate an expected output when processing a large number of passages:

- **Av. generated list length**: The average length of the generated lists, which should be as close as possible to the number of passages retrieved by BM25.
- **Never-ending generations**: The percentage of queries for which the LLM did not stop generating the output, which indicated that the LLM was overwhelmed by the number of passages and could not generate a valid output.
- **Av. n° invalid elements per list**: The average number of invalid elements in the generated lists, which corresponded to the number of ids that were not within the range of the initial set, or even non-numerical. This indicated that the LLM was hallucinating ids, which was another sign that the LLM was overwhelmed by the number of passages and could not generate a valid output.
- **Av. n° repeated elements per list**: The average number of repeated elements in the generated lists, which also indicated that the LLM was struggling to generate a valid output, i.e. it should not repeat ids in the output.
- **Av. inference time per query**: The average time taken by the LLM to generate the output for each query, which was a good indicator of the computational cost of the method.

Table 2: Reliability metrics from reranking all retrieved sets from BM25. The metrics for runs on the TREC-DL-2020 dataset are reported.

Metrics	**Runs**		
BM25 k	**100**	**200**	**300**
Av. generated list length	49.81	97.91	160.67
Never-ending generations	0%	3.70%	9.26%
Av. n° invalid elements per list	0.00	2.50	11.78
Av. n° repeated elements per list	0.24	1.43	3.22
Av. inference time per query	73s	220s	439s

These metrics are summarized in Table 2. The results show that the LLM struggled to generate valid outputs when processing >100 passages, with a considerable increase in the number of invalid and repeated elements, as well as a considerable increase in the proportion of queries leading to never-ending generations. The list length also revealed a lack of LLM capacity to respect the expected output, i.e. it was nearly half of the initial set length for all three runs. Finally, only the inference time could be an argument in favour of the use of this

method, but the increased time with the number of passages is also excessive.

Performance. For the performance evaluation, we report the NDCG@10 metrics for the TREC-DL-2019 and TREC-DL-2020 datasets. We compare our method against BM25 as baseline, as well as RankGPT, which is the state-of-the-art inline reranking method using LLMs. We re-implemented the RankGPT window strategy using the Qwen2.5-72B-Instruct model so as to compare both methods using the same model architecture.

Table 3: Performances of the pipeline for different numbers of passages retrieved in the first stage and filtered in the second stage. The NDCG@10 metrics for the TREC-DL-2019 and TREC-DL-2020 datasets are reported.

Methods			TREC-DL-2019	TREC-DL-2020
Model	BM25 k	Filter k	NDCG@10	NDCG@10
BM25	100	-	50.58	47.96
RankGPT* (gpt-3.5-turbo)	100	-	65.80	62.91
RankGPT* (gpt-4)	100	-	**75.59**	**70.56**
RankGPT** (Qwen2.5)	100	-	70.11	66.90
FIRE-LLM (Qwen2.5)	100	20	68.26	66.20
RankGPT** (Qwen2.5)	200	-	72.70	68.82
FIRE-LLM (Qwen2.5)	200	20	68.45	67.72
FIRE-LLM (Qwen2.5)	200	30	70.52	66.71
RankGPT** (Qwen2.5)	300	-	72.56	70.08
FIRE-LLM (Qwen2.5)	300	30	70.54	65.50

Table 3 shows that while RankGPT with gpt-4 had the highest performance, our method with Qwen2.5-72B-Instruct performed better than RankGPT with gpt-3.5-turbo. This could mainly be explained by the impact of the backbone LLM on the reranking performance. The main comparison was between our method and our re-implementation of RankGPT with Qwen2.5-72B-Instruct. The performance of our method was on par with that of RankGPT, with only a small difference for any dataset or number of passages retrieved in the first stage. This was a major result, as it showed that our method could perform as well as RankGPT while only requiring two LLM inferences.

Time and energy consumption. We assessed the computational efficiency of our pipeline by measuring the processing time, CO_2 emissions and energy consumption associated with each LLM stage. We took a single measurement for the whole queryset and then averaged the results to obtain a per-query

* Results directly extracted from the paper [12].

** Our re-implementation of RankGPT with Qwen2.5-72B-Instruct using the parameters of [12].

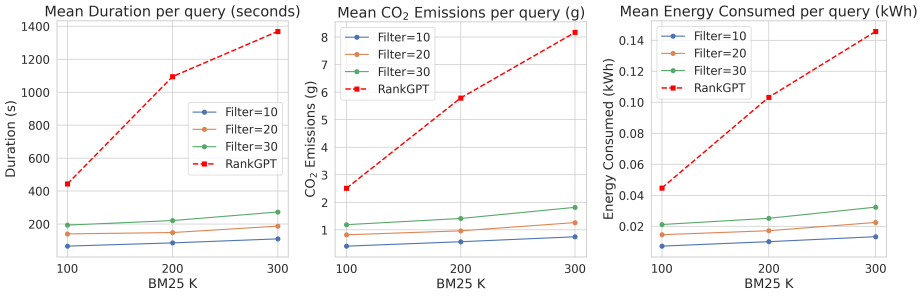

Fig. 3: Mean duration, CO_2 emissions and energy consumption per query. We compared our FIRE-LLM method with our re-implementation of RankGPT using the Qwen2.5-72B-Instruct with multiple filter thresholds.

measurement. The time was measured in seconds, while the CO_2 emissions and energy consumption were measured in grams (g) and kilowatt-hours (kWh), respectively. As we were unable to find any data on the CO_2 emissions and energy consumption of RankGPT, we compared our method against our re-implementation of RankGPT with Qwen2.5-72B-Instruct, which has the same model architecture as our method and the same window strategy as RankGPT, to ensure a fair comparison. All measurements were done using the `CodeCarbon` python package [3], which provided a reliable estimation of CO_2 emissions and energy consumption based on the power consumption of the whole system during the LLM inference process. CO_2 emissions were calculated by taking into account the energy consumption and carbon intensity of the electricity used, which varies by region and time of day. All measurements were obtained via the same computing infrastructure based in France.

The results shown in Figure 3 demonstrate that our method was much more efficient than RankGPT, i.e. notably achieving lower runtime and CO_2 emissions. This efficiency could be explained by the fact that our method performs only two fixed passes over the passage sets, regardless of their length, whereas RankGPT requires multiple sequential passes. The exact number of passes required by RankGPT can be determined using the following formula:

$$\text{Number of passes} = \left\lceil \frac{\mathrm{P} - (\mathrm{W} - \mathrm{S})}{\mathrm{S}} \right\rceil \tag{1}$$

where P is the number of passages, W is the window size, and S is the sliding step. For example, using the main configuration of RankGPT described in [12] with P=100, W=20, and S=10, the number of passes amounted to 9. Even when there were only 100 passages, the efficiency gap in terms of both processing time and CO_2 emissions was already substantial (2× to 5×); then when the number of passages increased to 200, 300, or more, the gap widened considerably (up to 14×). There is no public data available regarding the CO_2 emissions and energy consumption of gpt-3.5-turbo and gpt-4, but it can be assumed that the levels

are higher—or at least comparable—than those of Qwen2.5-72B-Instruct, given that gpt-3 has already 2.5× more parameters than Qwen2.5-72B-Instruct.

5 Conclusion

In this paper, we have introduced FIRE-LLM, a novel information retrieval pipeline that optimizes LLM use for IR tasks. Our approach combines conventional BM25 retrieval with two strategic LLM stages: a filtering phase and a final listwise reranking phase. This architecture achieves competitive performance while substantially reducing the computational costs compared to existing methods.

Our main findings are threefold. First, we demonstrated that direct reranking of large passage sets led to degraded performance and reliability issues, thus highlighting the necessity for intermediate filtering strategies. Second, we showed that our filtering approach enabled single-pass listwise reranking, thereby eliminating the need for computationally expensive sliding window techniques. Third, we provided comprehensive empirical evidence that our method could perform as well as state-of-the-art approaches like RankGPT while only requiring two LLM inferences instead of multiple sequential passes, hence substantially reducing the runtime and CO_2 emissions. The efficiency gains increased with the number of candidate passages, which means that our approach would be very suitable for large-scale information retrieval scenarios.

However, our FIRE-LLM pipeline has several limitations that warrant mention. As is the case for any LLM-based method, our approach is highly sensitive to the choice of LLM backbone, and the performance can vary considerably depending on the model used. The filtering stage adds even more impact on the LLM choice, as it relies on the model's context length, and its ability to process a high number of passages at once. Moreover, our method still requires two sequential LLM calls, which, despite being more efficient than sliding window approaches, cannot be parallelized and may limit scalability in high-throughput applications. The volume of passages processed by the LMMs is still an issue for real-time applications, as the LLM inference time rapidly increases with the number of passages, even for a single pass.

Looking ahead, we intend to explore advanced optimization techniques with the aim of further reducing the inference costs. We plan to investigate KV-cache mechanisms and especially KV-cache reuse strategies that could even further enhance the LLM utilization efficiency. Through our experiments, we observed that passages were often reused multiple times for different queries, and recomputing them each time was a massive computation expense. New recent approaches explore ways to reuse the KV-cache of passages already computed by the LLM [1,14]. These techniques are promising for reducing the computational overhead of the filtering and reranking stages, potentially making LLM-based retrieval pipelines even more practical for real-world deployment scenarios.

Acknowledgments. This project was provided with computing HPC/AI and storage resources by GENCI at IDRIS thanks to the grant 20XX-AD011014995R1 on the supercomputer Jean Zay's A100 partition.

References

1. An, Y., Cheng, Y., Park, S.J., Jiang, J.: Hyperrag: Enhancing quality-efficiency tradeoffs in retrieval-augmented generation with reranker kv-cache reuse (2025), https://arxiv.org/abs/2504.02921
2. Chen, S., Gutierrez, B.J., Su, Y.: Attention in large language models yields efficient zero-shot re-rankers. In: The Thirteenth International Conference on Learning Representations (2025), https://openreview.net/forum?id=yzloNYH3QN
3. Courty, B., Schmidt, V., Luccioni, S., Goyal-Kamal, MarionCoutarel, Feld, B., Lecourt, J., LiamConnell, Saboni, A., Inimaz, supatomic, Léval, M., Blanche, L., Cruveiller, A., ouminasara, Zhao, F., Joshi, A., Bogroff, A., de Lavoreille, H., Laskaris, N., Abati, E., Blank, D., Wang, Z., Catovic, A., Alencon, M., Stęchły, M., Bauer, C., de Araújo, L.O.N., JPW, MinervaBooks: mlco2/codecarbon: v2.4.1 (May 2024). https://doi.org/10.5281/zenodo.11171501, https://doi.org/10.5281/zenodo.11171501
4. Craswell, N., Mitra, B., Yilmaz, E., Campos, D.: Overview of the trec 2020 deep learning track. In: TREC (2020)
5. Lin, J., Ma, X., Lin, S.C., Yang, J.H., Pradeep, R., Nogueira, R.: Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In: Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021). pp. 2356–2362 (2021)
6. Ma, X., Zhang, X., Pradeep, R., Lin, J.: Zero-Shot Listwise Document Reranking with a Large Language Model (2023)
7. Meng, C., Arabzadeh, N., Askari, A., Aliannejadi, M., de Rijke, M.: Ranked List Truncation for Large Language Model-based Re-Ranking. In: Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 141–151 (2024)
8. Nouriinanloo, B., Lamothe, M.: Re-Ranking Step by Step: Investigating Pre-Filtering for Re-Ranking with Large Language Models (2024)
9. Qin, Z., Jagerman, R., Hui, K., Zhuang, H., Wu, J., Yan, L., Shen, J., Liu, T., Liu, J., Metzler, D., Wang, X., Bendersky, M.: Large language models are effective text rankers with pairwise ranking prompting. In: Duh, K., Gomez, H., Bethard, S. (eds.) Findings of the Association for Computational Linguistics: NAACL 2024. pp. 1504–1518. Association for Computational Linguistics, Mexico City, Mexico (Jun 2024). https://doi.org/10.18653/v1/2024.findings-naacl.97, https://aclanthology.org/2024.findings-naacl.97/
10. Rathee, M., MacAvaney, S., Anand, A.: Guiding retrieval using llm-based listwise rankers. In: Advances in Information Retrieval: 47th European Conference on Information Retrieval, ECIR 2025, Lucca, Italy, April 6–10, 2025, Proceedings, Part I. p. 230–246. Springer-Verlag, Berlin, Heidelberg (2025). https://doi.org/10.1007/978-3-031-88708-6_15, https://doi.org/10.1007/978-3-031-88708-6_15
11. Robertson, S., Zaragoza, H.: The probabilistic relevance framework: Bm25 and beyond. Foundations and Trends in Information Retrieval **3**, 333–389 (01 2009). https://doi.org/10.1561/1500000019
12. Sun, W., Yan, L., Ma, X., Wang, S., Ren, P., Chen, Z., Yin, D., Ren, Z.: Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. pp. 14918–14937. Association for Computational Linguistics (2023)

13. Team, Q.: Qwen2.5: A party of foundation models (September 2024), https://qwenlm.github.io/blog/qwen2.5/
14. Yang, J., Hou, B., Wei, W., Bao, Y., Chang, S.: Kvlink: Accelerating large language models via efficient kv cache reuse (2025), https://arxiv.org/abs/2502.16002
15. Zhu, Y., Yuan, H., Wang, S., Liu, J., Liu, W., Deng, C., Chen, H., Liu, Z., Dou, Z., Wen, J.R.: Large language models for information retrieval: A survey (2024), https://arxiv.org/abs/2308.07107
16. Zhuang, H., Qin, Z., Hui, K., Wu, J., Yan, L., Wang, X., Bendersky, M.: Beyond yes and no: Improving zero-shot LLM rankers via scoring fine-grained relevance labels. In: Duh, K., Gomez, H., Bethard, S. (eds.) Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers). pp. 358–370. Association for Computational Linguistics, Mexico City, Mexico (Jun 2024). https://doi.org/10.18653/v1/2024.naacl-short.31, https://aclanthology.org/2024.naacl-short.31/
17. Zhuang, S., Zhuang, H., Koopman, B., Zuccon, G.: A setwise approach for effective and highly efficient zero-shot ranking with large language models. In: Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. p. 38–47. SIGIR '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3626772.3657813, https://doi.org/10.1145/3626772.3657813