MIRROR: Measuring, Improving and Reproducing Ranking with Open Retrieval models

Mehak Dhaliwal and Jasmine Lesner
Santa Barbara, California, USA

Abstract

We conduct a replication study of recent advances in zero-shot document ranking with Large Language Models (LLMs), focusing on the Setwise approach introduced at SIGIR 2024. Our results confirm high fidelity in effectiveness metrics (NDCG@10 within ±3%) but reveal efficiency discrepancies, including 33–40% lower token usage for setwise methods. Expanding on the original work, we evaluate performance on the NovelEval-2306 dataset, showing strong ranking capabilities for queries beyond the models' training cutoff. Systematic prompt engineering yields up to 40.7% improvements in NDCG@10 for specific method-dataset pairs, with setwise methods benefiting the most. Experiments with instruction-tuned and conversationally fine-tuned models (Llama 3.1, Llama 2) show consistent gains without added computational cost. These findings validate the original conclusions and highlight strategies for optimizing LLM-based ranking systems through prompt engineering and model selection.

CCS Concepts

• **Information retrieval**; • **Language models**; • **Retrieval models and ranking**; • **Evaluation of retrieval results**;

Keywords

Zero-shot document ranking, Large Language Models (LLMs), Setwise ranking, Pairwise ranking, Listwise ranking, Prompt engineering, TrecDL datasets, BEIR datasets, NDCG@10, Information retrieval, Model efficiency, Query latency, BM25, Computational cost analysis, Reproducibility

1 Introduction

We selected the paper introducing *Setwise*, a novel method for zero-shot document ranking using Large Language Models (LLMs). This approach enhances efficiency by reducing LLM inferences and prompt token usage during ranking while maintaining high effectiveness. The paper, titled *"A Setwise Approach for Effective and Highly Efficient Zero-shot Ranking with Large Language Models"* [10], was presented at the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2024).

We chose this paper because it presents a new methodology that addresses efficiency challenges in zero-shot ranking with LLMs, offering a promising research direction. Previous literature has often lacked a fair and consistent comparison of the effectiveness and efficiency of various techniques. This paper fills that gap by providing a rigorous comparative framework for prompting methods—Pointwise, Pairwise, Listwise, and Setwise—shedding light on their trade-offs. Moreover, with zero-shot and few-shot learning being rapidly evolving fields, this work holds significant relevance.

2 Background

This section summarizes the key algorithms and datasets. Readers should refer to [10] for foundational details essential to understanding our replication and improvements.

2.1 Key Algorithms

Re-ranking algorithms refine the order of documents retrieved by a fast but approximate initial ranking. These algorithms, while more resource-intensive, optimize the top results for greater relevance. They are crucial for surfacing the most relevant content in applications like search engines. **Pointwise**: Scores each document independently for relevance [2, 6, 9]. Variants include: (1) Yes/No Generation (*pointwise.yes_no*): Ranks by the likelihood of "yes." (2) Query Likelihood Modeling (*pointwise.qlm*): Ranks by query-generation likelihood. These methods are efficient but depend on model logits and can suffer from calibration issues. **Listwise**: Generates ranked lists (*listwise.generate*) using sliding windows to re-rank document chunks iteratively [3, 4, 7]. This method is more efficient than Pointwise but relies on coherent list generation. **Pairwise**: Compares document pairs for relevance [5]. Basic approaches (*pairwise.allpairs*) are expensive. Optimized variants use sorting algorithms, e.g., heap sort (*pairwise.heapsort*) or bubble sort (*pairwise.bubblesort*), balancing effectiveness and efficiency. **Setwise**: Processes multiple documents simultaneously for faster sorting (*setwise.heapsort*, *setwise.bubblesort*). It can also refine Listwise rankings using logits (*listwise.likelihood*), achieving strong effectiveness with greater efficiency than Pairwise methods.

2.2 Key Datasets

The authors [10] assess performance of re-ranking using:

- **TrecDL Datasets** [1]: **TrecDL 2019**: 8.8M passages, 503K queries, 43 test queries (~50 GB). **TrecDL 2020**: Similar structure with document and passage ranking tasks (~75 GB).
- **BEIR Datasets** [8]: Includes datasets like Covid, NFCorpus, Touche, DBPedia, SciFact, Signal-1M (RT), News, and Robust04, ranging in size from ~10 MB to ~20 GB.

In this study extend the analysis to the **NovelEval-2306 Dataset** [7], featuring 21 queries with relevance judgments for 20 passages per query.

2.3 Key Metrics

The authors in [10] evaluate **Effectiveness** (using NDCG@10, which assesses ranking quality) and **Efficiency** with:

- Average LLM inferences per query.
- Average prompt tokens per query.
- Average generated tokens per query.
- Average query latency (in seconds).

In this report, we use a metric for our ability to reproduce their published results:

$$\textbf{Discrepancy \%} = \frac{\textbf{Measured} - \textbf{Published}}{\textbf{Published}}$$

The ideal value for this metric is zero. A large positive or negative Discrepancy % suggests a mismatch between our measurements and the published results, indicating potential issues.

To evaluate impact of our algorithm modifications on measured results we use:

$$\textbf{Increase \%} = \frac{\textbf{Modified} - \textbf{Original}}{\textbf{Original}}$$

For NDCG@10, a high positive Increase % is desirable but for query latency, a negative Increase % is ideal.

3 Method

We pursued **three objectives**: (1) Reproduce the results from [10]. (2) Apply the work to additional datasets and LLMs. (3) Explore potential improvements to the work.

The key results in [10] are summarized in two tables. *Table 2* in [10] reports the effectiveness (NDCG@10) and efficiency of various ranking methods tested on the TREC DL 2019 and 2020 datasets, comparing three sizes of the Flan-t5 model (large, xl, and xxl). And *Table 3* in [10] presents the effectiveness of zero-shot ranking methods across multiple BEIR benchmark datasets, again comparing the three model sizes.

We began by cloning the repository at https://github.com/ielab/llm-rankers, as referenced in [10]. To streamline experimentation, we wrote scripts to automate Pyserini commands, collect evaluation logs, and load the results into a Python notebook. While awaiting experimental results, we also imported from [10] both *Table 2* and *Table 3* into the notebook to prepare for comparative analysis.

For all experiments: (1) Initial retrieval used BM25 via the Pyserini library with default settings. (2) Re-ranking was performed on the top 100 documents retrieved by BM25. (3) Evaluation focused on the NDCG@10 metric.

4 Observations: Measured vs. Published

Running experiments we collected over 3,000 metrics, categorized as follows: 17% were directly from [10], 60% were measurements to replicate their results, 23% were new measurements evaluating modifications to their algorithms. Our dataset, along with the Python notebook used for analysis, includes a cached copy of our collected data and many tables and charts. In this section, we present a summary of our efforts to reproduce and improve upon the results in [10].

4.1 TrecDL Metric Discrepancy (Figure 1)

This chart is a match for [10]'s *Table 2* which reports the effectiveness (NDCG@10) and efficiency of various ranking methods tested on the TREC DL 2019 and 2020 datasets. Our chart shows the *Discrepancy %* in each matching value.

Observations

- Substantial negative discrepancies (−96%) in inference counts were observed for pointwise methods, indicating potential differences in implementation for inference counting.
- Setwise methods show consistent negative discrepancies in generated token counts (−33% to −40%), likely due to variations in token generation definitions.
- NDCG@10 discrepancies were minor (mostly within ±3%), supporting successful replication of effectiveness results.
- Pairwise methods displayed varied discrepancy patterns across metrics, reflecting diverse influences in efficiency metrics.
- Patterns of discrepancies were consistent across TREC-DL19 and TREC-DL20 datasets, as well as across model sizes (large, xl, xxl).
- Prompt token discrepancies were notably larger for setwise and pairwise methods, suggesting differences in prompt engineering approaches.

Causes

- An investigation revealed that the −96% discrepancy in pointwise inference counts occurred because [10] used batch size 1 for their paper but had their shared code configured for batch size 32.
- Differences in library versions can lead to changes in how operations are implemented or executed.
- Dependencies (e.g., TensorFlow, PyTorch) and GPU drivers directly influence model performance.
- Pretrained model weights are occasionally updated in public repositories. If downloaded at different times or from different sources, slight differences in initialization could impact token generation and inference.
- Even small updates to datasets, such as corrections or added examples, can influence model outputs.
- Code available in repositories might not exactly match what was used in the original study.

Assessment

- *Challenges:* Large discrepancies in efficiency metrics (inferences, tokens) and systematic patterns of divergence indicate that some implementation details were not fully replicated.
- *Successful Aspects:* The NDCG@10 values, the primary effectiveness metric, show minimal discrepancies, and the relative performance ranking of methods is consistent with the original study.
- *Overall Assessment:* This represents a partial replication success. While effectiveness metrics validate the main conclusions, efficiency results highlight the importance of detailed implementation documentation in ensuring replicability in ML research.

4.2 BEIR NDCG@10 Discrepancy (Figure 2)

This chart matches [10]'s *Table 3* which presents the effectiveness of zero-shot ranking methods across multiple BEIR benchmark datasets, again comparing the three model sizes. Our chart shows the *Discrepancy %* in each matching value.

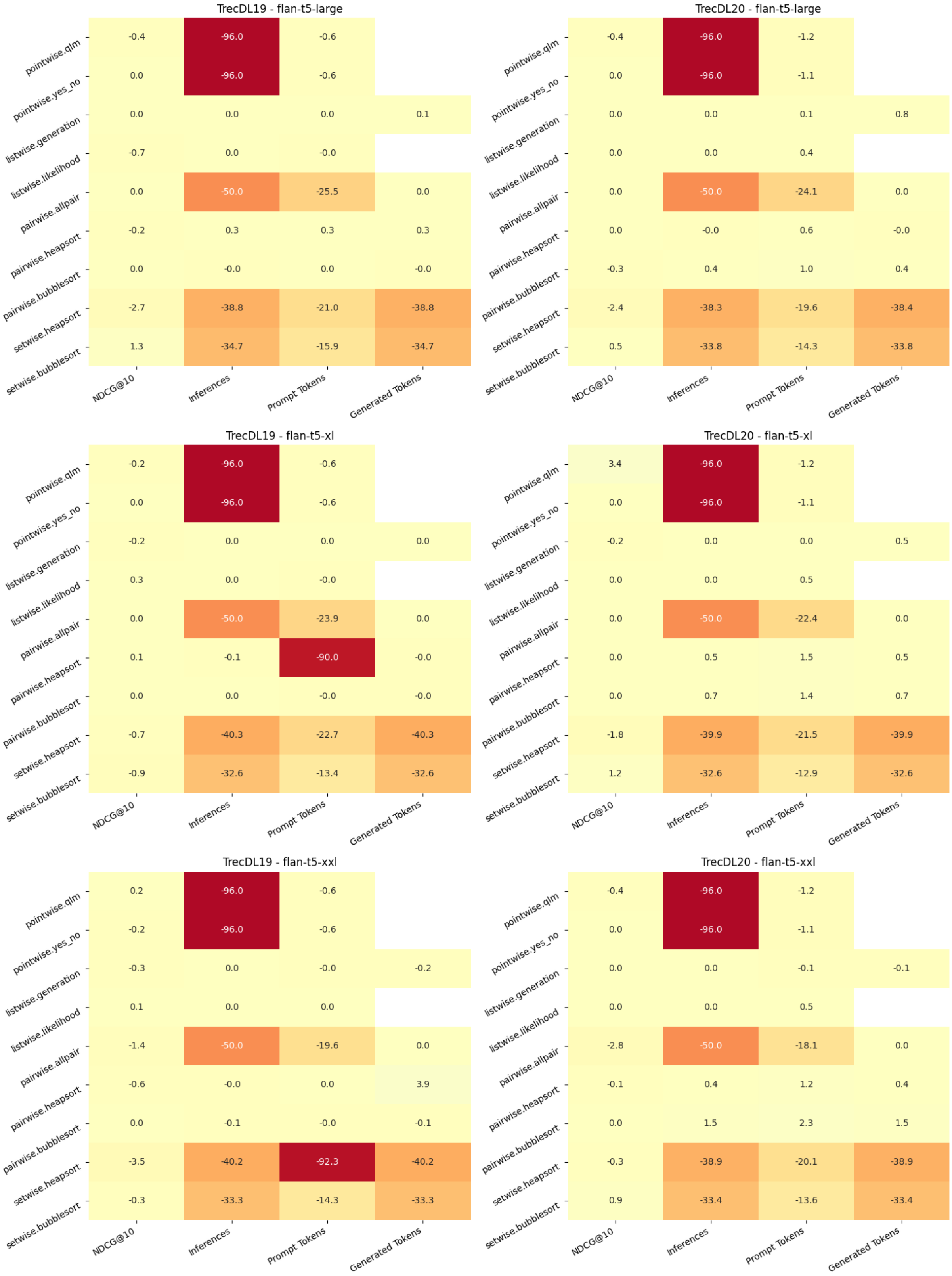


Figure 1: Six heatmaps compare replication results (of Table 2 in [10]) for Flan-T5 models (large, xl, xxl) on TREC-DL19 (left) and TREC-DL20 (right). Rows show model sizes; columns separate datasets. Discrepancies (%) between measured and published values across four metrics (NDCG@10, inferences, prompt tokens, generated tokens).

Observations

- Most discrepancies are very small, typically within ±2%, indicating high agreement between measured and published NDCG@10 values.
- The `flan-t5-xxl` heatmap contains many missing values (white cells), indication incomplete measurements for this model size.
- A notable outlier was observed: −25.5% for the `pairwise.bubblesort` method on the SciFact dataset with `flan-t5-large`.
- Setwise methods tend to show slightly larger discrepancies compared to pointwise and pairwise methods.
- Results are more complete for `flan-t5-large` and `flan-t5-xl`, providing better coverage of datasets and methods.
- Certain datasets, such as Covid and Robust04, exhibit more consistent replication results across methods, while the Signal dataset shows greater variability in discrepancies.

Causes

- The high number of missing values for `flan-t5-xxl` is due to computational resource constraints during the replication effort. Section B explores this further.
- Consistently small discrepancies across most measurements could result from:
  - Minor preprocessing differences between the replication and original experiments.
  - Variations caused by random initialization effects.
  - Differences in hardware or software environments.
- The −25.5% outlier on SciFact may reflect:
  - A possible error in the original reported results.
  - An unaccounted-for implementation detail specific to the `pairwise.bubblesort` method.
  - Dataset-specific preprocessing or handling differences.

Assessment

- *Challenges:*
  - Incomplete replication for the `flan-t5-xxl` model due to missing measurements.
  - A small number of notable outliers, especially the −25.5% discrepancy for `pairwise.bubblesort` on SciFact.
  - Systematic differences in setwise methods could indicate implementation variations not fully captured in the replication.
- *Successful Aspects:*
  - Most discrepancies are within ±5%, demonstrating strong alignment between measured and published values.
  - The general pattern of results is consistent across datasets, methods, and model sizes where measurements are complete.
  - Measured discrepancies are systematic and minor, suggesting acceptable implementation differences rather than fundamental issues.
- *Overall Assessment:* This replication effort can be considered successful. The small discrepancies (typically within ±2%) confirm the published results with high fidelity. The exceptions, such as missing values and outliers, do not undermine the overall conclusions but highlight areas for further investigation and emphasize the importance of detailed documentation in ensuring replicability in machine learning research.

4.3 Replication of Effectiveness and Efficiency Tradeoffs

Figures 3(a) and 3(b) replicate findings from [10] (*Figure 3*) on the impact of two hyperparameters: c, which controls the number of documents compared simultaneously in setwise prompting, and r, the number of sliding window repetitions in listwise prompting.

Figure 3(a) shows the tradeoff between efficiency and effectiveness as c increases. Higher c values reduce query latency but may compromise effectiveness due to document truncation caused by LLM input length limitations. Additionally, the heap sort algorithm consistently outperforms bubble sort in efficiency.

Figure 3(b) reveals a linear relationship between latency and r. Listwise likelihood, which incorporates setwise prompting, is consistently more effective and efficient than listwise generation.

4.4 Replication of Sensitivity to the Initial Ranking

Previous work has highlighted the sensitivity of Listwise and Pairwise re-ranking methods to the initial ranking order ([5, 7]). We evaluated this sensitivity for these methods and the setwise approach using three variations of the BM25 list: (1) original, (2) inverted, and (3) randomly shuffled.

Figure 4 shows results with the Flan-T5-large model. Listwise likelihood (using setwise prompting) demonstrates greater robustness to initial ranking order compared to Listwise generation. Heapsort-based methods (pairwise and setwise) deliver consistent performance across rankings, with setwise slightly outperforming pairwise. Bubblesort-based methods are more affected by initial rankings, though setwise bubblesort is more robust than pairwise. Overall, setwise methods exhibit superior robustness to initial ranking variations.

5 Observations: NovelEval Performance

Table 1 summarizes our NovelEval benchmarks. Consistent with previous findings, all re-ranking methods outperform BM25, confirming that improvements stem from the models' ranking capabilities. Setwise and pairwise prompting achieve the best performance, with setwise prompting being more efficient. This highlights that setwise prompting enables strong ranking even when the models lack prior topic knowledge.

6 Observations: Fine-Tuned Model Performance

We explored the impact of model fine-tuning by comparing base models (Llama 3.1 8B, Llama 2 7B/13B) with their instruction-tuned (Llama 3.1 8B-Instruct) and conversationally fine-tuned (Vicuna 7B/13B) versions. As shown in Figure 5, fine-tuned models consistently improved NDCG@10 without significant latency increase, suggesting fine-tuning enhances ranking capabilities even without explicit ranking optimization.

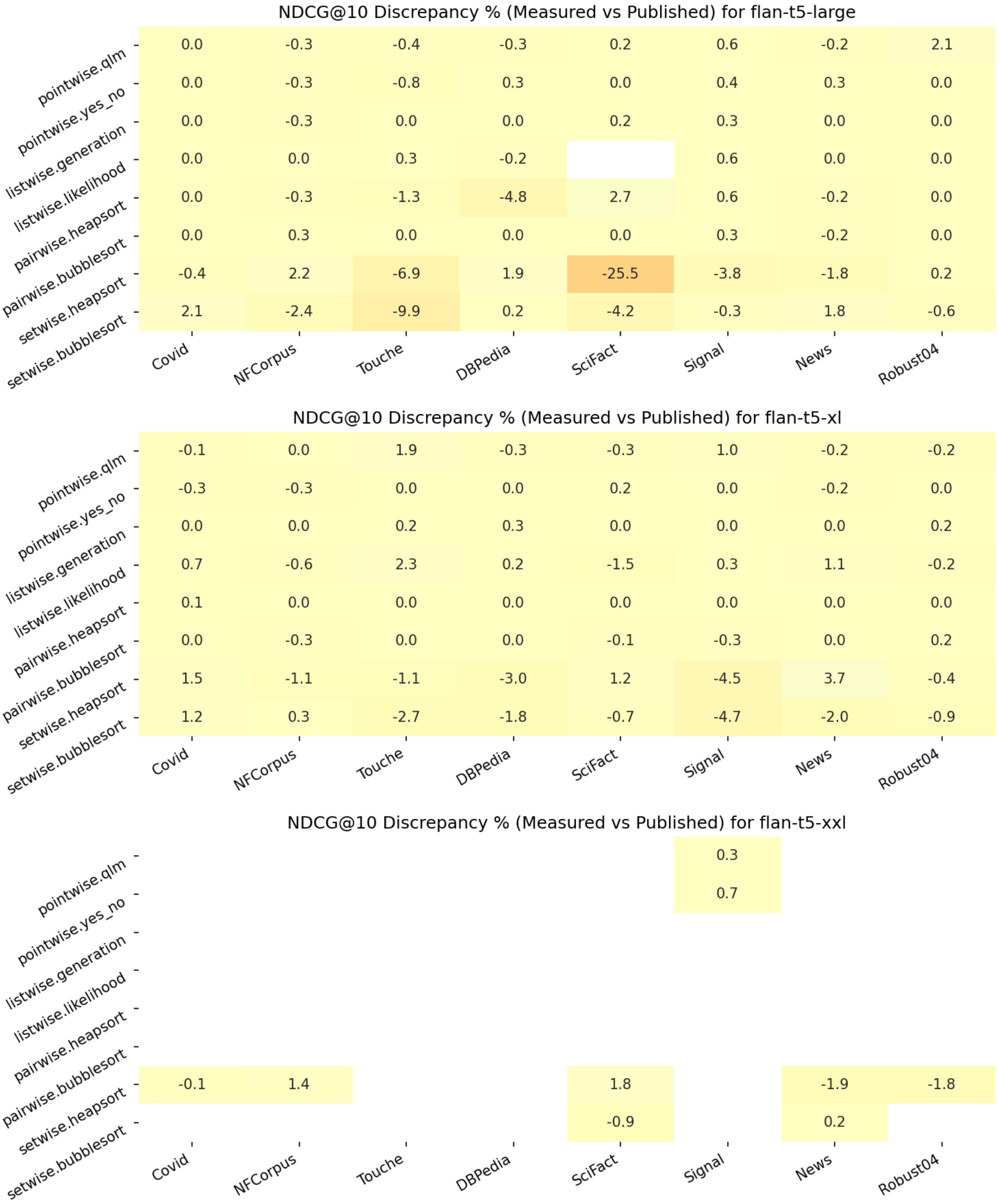
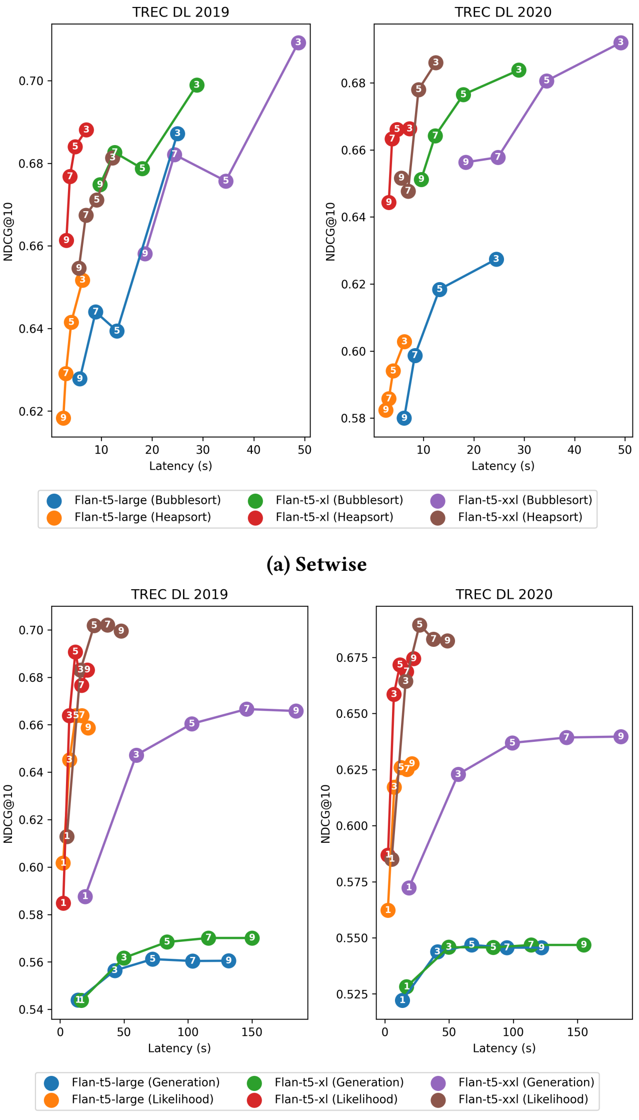


Figure 2: The figure shows three heatmaps comparing replication attempts of NDCG@10 results (from Table 3 in [10]) across eight BEIR datasets. Heatmaps are grouped by Flan-T5 model size (large/xl/xxl). Rows represent ranking methods, columns represent datasets. White cells indicate missing data.



(a) Setwise

(b) Listwise

Figure 3: Effectiveness and efficiency tradeoffs across methods. (a) Setwise: scatter plot numbers indicate documents compared (c) at each step of the sorting algorithm. (b) Listwise: scatter plot numbers indicate sliding window repetitions (r).

7 Observations: Modified vs Original

We explored **prompt tuning** the pointwise, pairwise and setwise algorithms as shown in Figure 9, Figure 10, Figure11, and Figure 12. In this section we compare versions of pointwise, pairwise and setwise that use our modified prompts to the original versions using the **Increase %** metric from Section 2.3.

Table 1: Effectiveness and efficiency results obtained on NovelEval dataset. The best NDCG@10 results are highlighted in boldface.

	Methods	NDCG@10	#Inferences	Pro. tokens	Gen. tokens	Latency (s)
	BM25	0.684	-	-	-	-
flan-t5-large	pointwise.qlm	0.691	4.0	15503.9	-	0.6
	pointwise.yes_no	0.738	4.0	16394	-	0.7
	listwise.generation	0.776	242.1	123971.1	1230.8	32.6
	listwise.likelihood	0.797	242.1	114045.0	-	12.8
	pairwise.allpair	0.802	4853.6	3000465.5	48536.2	426.6
	pairwise.heapsort	0.811	211.6	130628.5	2115.7	17.8
	pairwise.bubblesort	0.804	658.7	406968.7	6587.1	56.5
	setwise.heapsort	0.806	72.4	41609.2	361.9	5.8
	setwise.bubblesort	**0.844**	306.7	175563.7	1533.6	30.0
flan-t5-xl	pointwise.qlm	0.696	4.0	15503.9	-	1.2
	pointwise.yes_no	0.770	4.0	16394	-	1.3
	listwise.generation	0.777	242.1	123975.4	1227.8	36.7
	listwise.likelihood	0.814	242.1	114064.0	-	12.5
	pairwise.allpair	**0.818**	4853.6	3000465.5	48536.2	521.7
	pairwise.heapsort	0.815	208.7	128694.6	2086.7	22.4
	pairwise.bubblesort	0.813	549.4	339230.2	5493.8	79.2
	setwise.heapsort	0.789	72.9	41947.1	364.5	7.5
	setwise.bubblesort	0.808	311.8	178694.5	1558.8	31.6
flan-t5-xxl	pointwise.qlm	0.684	4.0	15503.9	-	3.2
	pointwise.yes_no	0.784	4.0	16394	-	3.4
	listwise.generation	0.816	242.1	123975.6	1228.2	60.0
	listwise.likelihood	**0.825**	242.1	114073.4	-	32.5
	pairwise.allpair	0.797	4853.6	3000465.5	48536.2	944.1
	pairwise.heapsort	0.778	213.9	132073.9	2139.0	42.4
	pairwise.bubblesort	0.806	718.1	443344.9	7181.4	138.4
	setwise.heapsort	0.808	72.9	41988.6	364.3	14.3
	setwise.bubblesort	0.818	302.8	173491.1	1513.8	59.6

7.1 BEIR NDCG@10 Increase (Figure 6)

Observations

- Setwise methods show consistently larger positive changes, particularly on Touche and SciFact datasets.
- A significant improvement of 40.7% is observed for pairwise.bubblesort on SciFact using the Flan-T5-large model.
- Pointwise methods often exhibit modest or negative changes, indicating relative insensitivity to prompt modifications.
- Datasets such as Touche and SciFact demonstrate higher sensitivity to prompt variations, leading to notable improvements.
- The xl model displays more stable and consistent changes compared to the large model, highlighting greater robustness.
- Certain method-dataset combinations achieve improvements exceeding 10%, emphasizing the impact of prompt changes.
- Flan-T5-large generally yields more pronounced improvements than xl, suggesting size-specific dynamics.

Causes

- Larger improvements for setwise methods imply that original prompts were suboptimal for these techniques, allowing room for optimization.
- Dataset-specific sensitivities may reflect task or domain characteristics affecting performance.
- The xl model's stability suggests that larger models have more robust representations, making them less affected by prompt variations.
- Dramatic improvements for SciFact could indicate that original prompts overlooked domain-specific needs, making changes particularly effective.

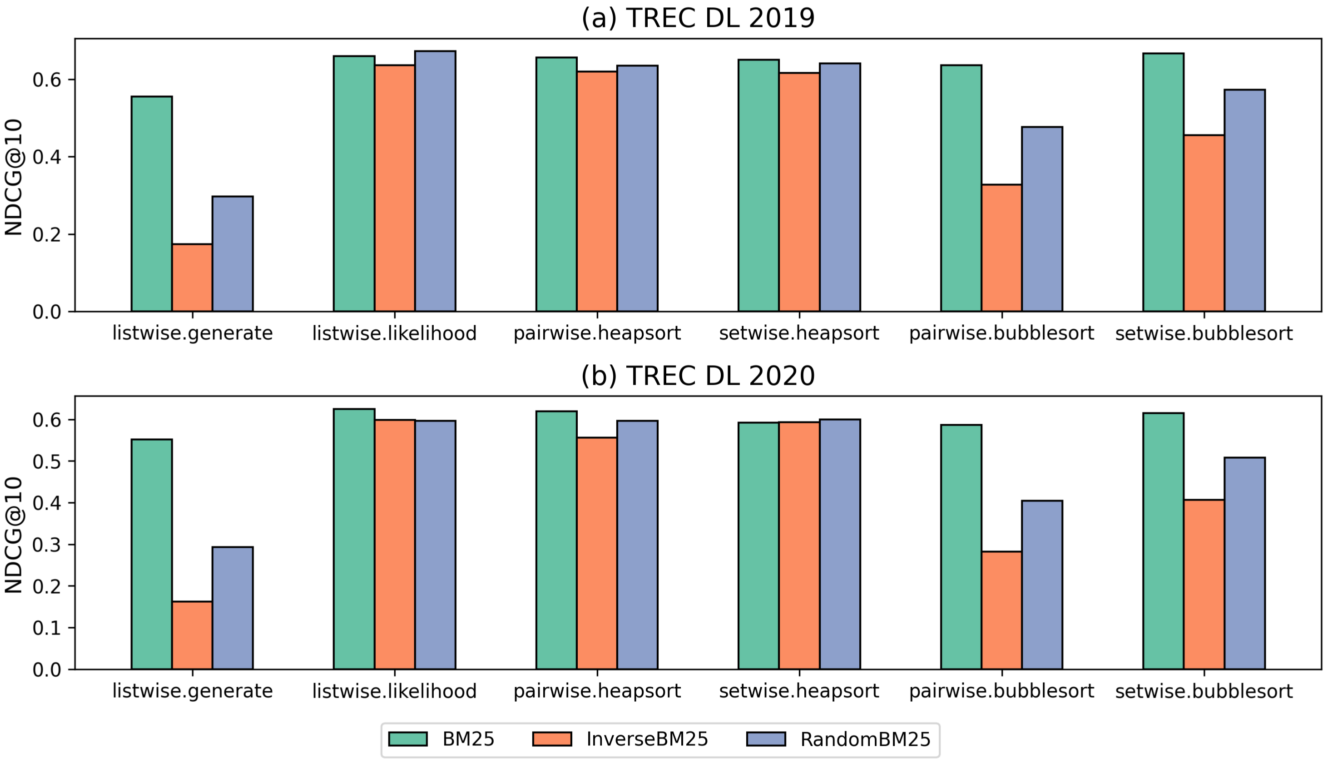


Figure 4: Sensitivity to the initial ranking for the Flan-T5-large model.

- Pointwise methods' modest changes suggest less dependence on or responsiveness to prompt engineering compared to other methods.

Evaluation of Modifications

- *Challenges:*
  - Performance gains are inconsistent, with some methods and datasets showing reduced performance.
  - Many changes are minor (under 5%), questioning the modifications' overall impact.
  - Divergent behavior between large and xl models raises concerns about generalizability across architectures.
  - Results may not extrapolate to untested datasets or models, limiting the changes' broader applicability.
- *Successful Aspects:*
  - Substantial gains exceeding 10% in key scenarios, with a 40.7% improvement for setwise.heapsort on SciFact, demonstrate the modifications' effectiveness.
  - Over all datasets, setwise methods appears to have benefited the most from the modified prompts.
  - Consistent positive outcomes for the xl model imply robustness and scalability of improvements.
  - Improvements span multiple datasets for certain methods, showing generalizability across domains.
  - Worst-case declines are limited to −7%, avoiding catastrophic performance degradations.

Assessment: The prompt modifications show clear benefits, particularly for setwise methods. Key evidence includes:

(1) Substantial improvements, including a 40% gain for pairwise.bubblesort, outweigh modest degradations.
(2) The xl model's consistent results suggest the changes scale well with larger architectures.
(3) Gains for newer methods indicate potential for further optimization through prompt engineering.
(4) Limited degradations (no worse than −7%) ensure that the modifications do not introduce significant trade-offs.

While the our prompt modifications are not universally effective, they yield substantial benefits in specific method-dataset combinations, demonstrating significant potential without severe downsides.

7.2 BEIR Metric Increase (Figures 7 and 8)

Observations

- *Model Size Matters:* flan-t5-large and flan-t5-xl models often display different, sometimes opposite patterns of changes in metrics.
- *Metric Correlations:* Changes in inference counts often correlate with changes in generated tokens, suggesting an interdependence between these metrics.
- *Dataset Sensitivity:* Certain datasets, such as Robust04 and SciFact, exhibit more dramatic changes, indicating varying levels of impact based on dataset characteristics.

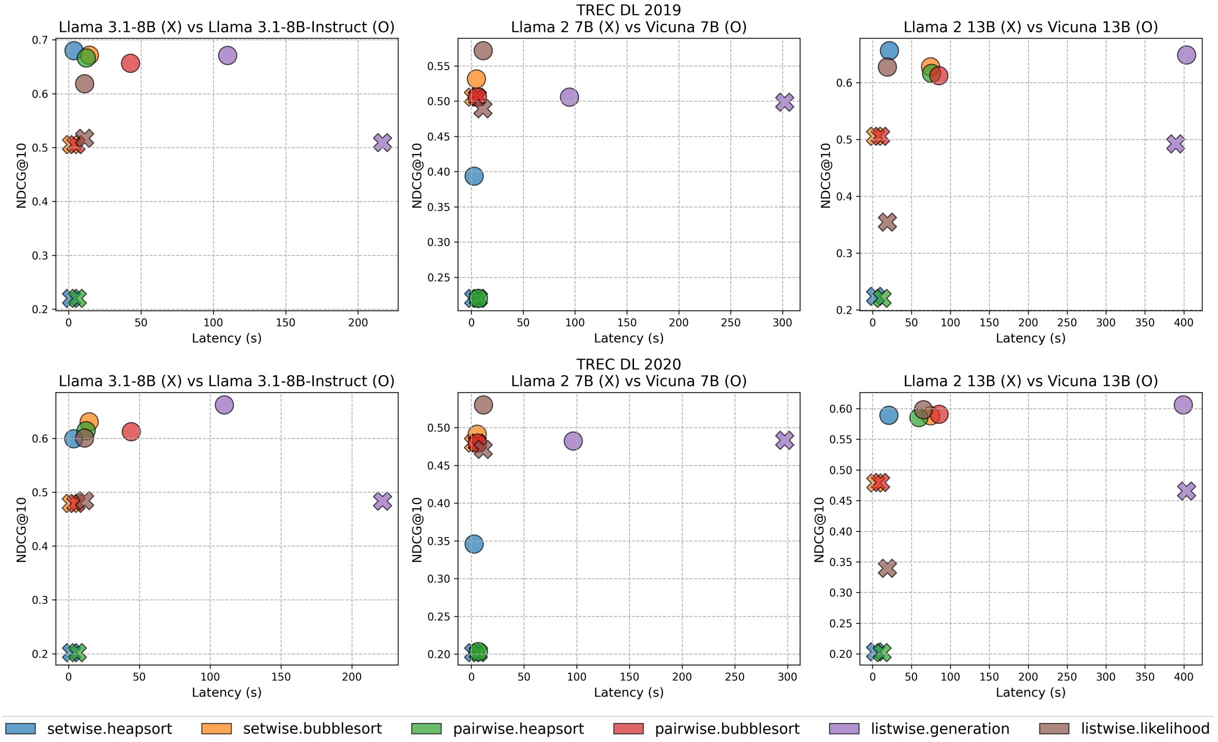


Figure 5: Effectiveness (NDCG@10) and efficiency (Latency (s)) comparison of base models (pictured with 'X') and their fine-tuned counterparts (pictured with 'O').

- *Method-Specific Patterns:*
  - Pointwise methods show minimal changes in inference counts, highlighting stability in efficiency.
  - Pairwise methods often exhibit larger changes across all metrics, reflecting higher sensitivity to prompt modifications.
  - Setwise methods show mixed results, with some instances of dramatic efficiency gains.
- *Efficiency-Effectiveness Tradeoff:* Some methods achieve simultaneous NDCG@10 improvements and reductions in computational costs, pointing to potential optimization opportunities.

Causes

- *Model Size Differences:* Divergent patterns between flan-t5-large and flan-t5-xl may reflect:
  - Optimal prompting strategies varying by model capacity.
  - Greater robustness of larger models (e.g., flan-t5-xl) to prompt variations.
- *Dataset-Specific Variations:* Differences across datasets could be due to:
  - Domain-specific language characteristics influencing prompt effectiveness.
  - Varying complexity in the underlying ranking tasks.
- *Method-Specific Patterns:* Discrepancies might arise from:
  - Differences in prompt dependency across ranking methods.
  - Variations in optimization opportunities depending on architectural choices.

Evaluation of Modifications

- *Challenges:*
  - Improvements are inconsistent across datasets and models, making generalization difficult.
  - Some significant degradations occur in efficiency metrics, raising concerns about reliability.
- *Successful Aspects:*
  - Several methods show simultaneous effectiveness and efficiency gains, demonstrating potential for improvement.
  - Dramatic improvements (e.g., > 30%) are observed in some cases, highlighting areas of opportunity.
  - Optimization potential is evident in ranking methods, supporting further exploration.
  - No catastrophic failures suggest the approach is fundamentally sound.
  - Some consistent patterns across datasets and methods provide actionable insights for refinement.

Overall Assessment: The results show promise but also highlight risks and limitations. Dataset and model-specific optimization appears more effective than generalized prompt modifications. Inconsistencies across datasets and models indicate that a one-size-fits-all approach is unlikely to succeed. A nuanced strategy that accounts for dataset characteristics and model capacity is more practical. While degradation risks are present, the potential for significant

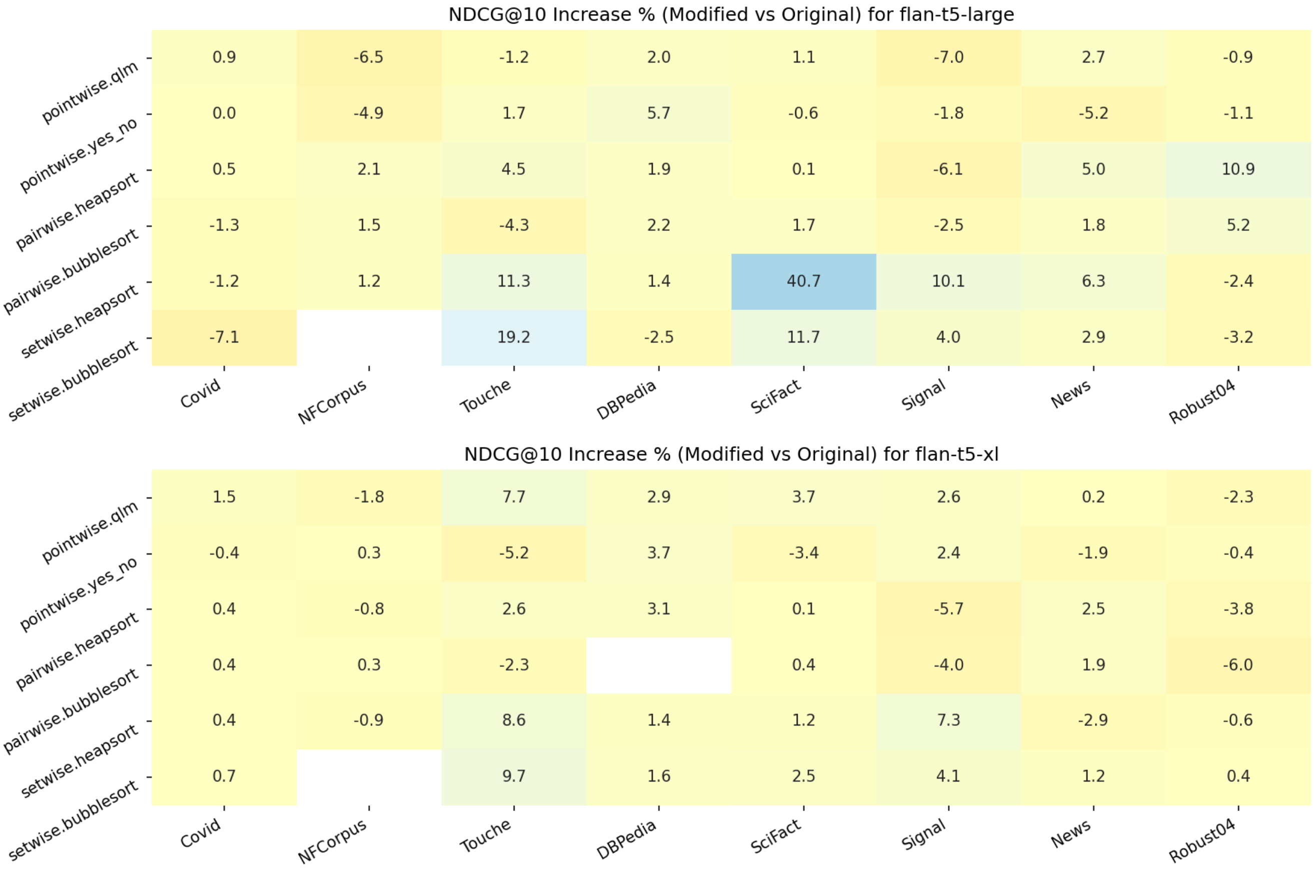


Figure 6: Heatmaps show percentage changes in NDCG@10 scores when prompts for re-ranking methods were modified. Rows represent methods, columns are BEIR datasets, and each heatmap corresponds to a Flan-T5 model size (large/xl).

improvements justifies further investigation when modifications are carefully matched to the use case.

8 Conclusion

This study contributes to zero-shot document ranking with Large Language Models (LLMs) in three key ways.

First, our replication of [10] confirms the effectiveness of their approach while revealing discrepancies in efficiency metrics. We aligned closely with the original study on ranking performance (NDCG@10 within ±3%) but observed notable differences in efficiency, including a 96% reduction in inference counts for pointwise methods (which we now know is due to differences in batch size 1 vs 32) and a 33%-40% decrease in token usage for setwise methods. These findings underscore the need for precise implementation documentation in machine learning research.

Second, our experiments on the NovelEval dataset show the robustness of LLM-based ranking methods when addressing queries with information beyond the models' training cutoff dates. All methods outperformed the BM25 baseline, with setwise and pairwise approaches excelling. This result highlights the strong generalization ability of LLMs, even for unfamiliar topics.

Third, our exploration of prompt engineering and model fine-tuning demonstrates substantial potential for improvement. Prompt modifications led to up to 40.7% gains in NDCG@10 for certain method-dataset combinations, especially benefiting setwise methods. Fine-tuning experiments across models like Llama 3.1 and Llama 2 showed consistent gains without major computational costs. These results suggest that tailored optimization strategies outperform one-size-fits-all approaches.

These findings offer guidance for both research and practical applications. Researchers should prioritize comprehensive documentation and hardware transparency to ensure reproducibility. Practitioners can achieve significant performance improvements through prompt engineering and model selection, even with limited computational resources. However, our work also highlights challenges in generalizing improvements across datasets and models, reinforcing the need for context-specific optimizations.

References

[1] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. 2020. Overview of the TREC 2019 Deep Learning Track. In *Proceedings of the Twenty-Eighth Text REtrieval Conference (TREC 2019).*

[2] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al.

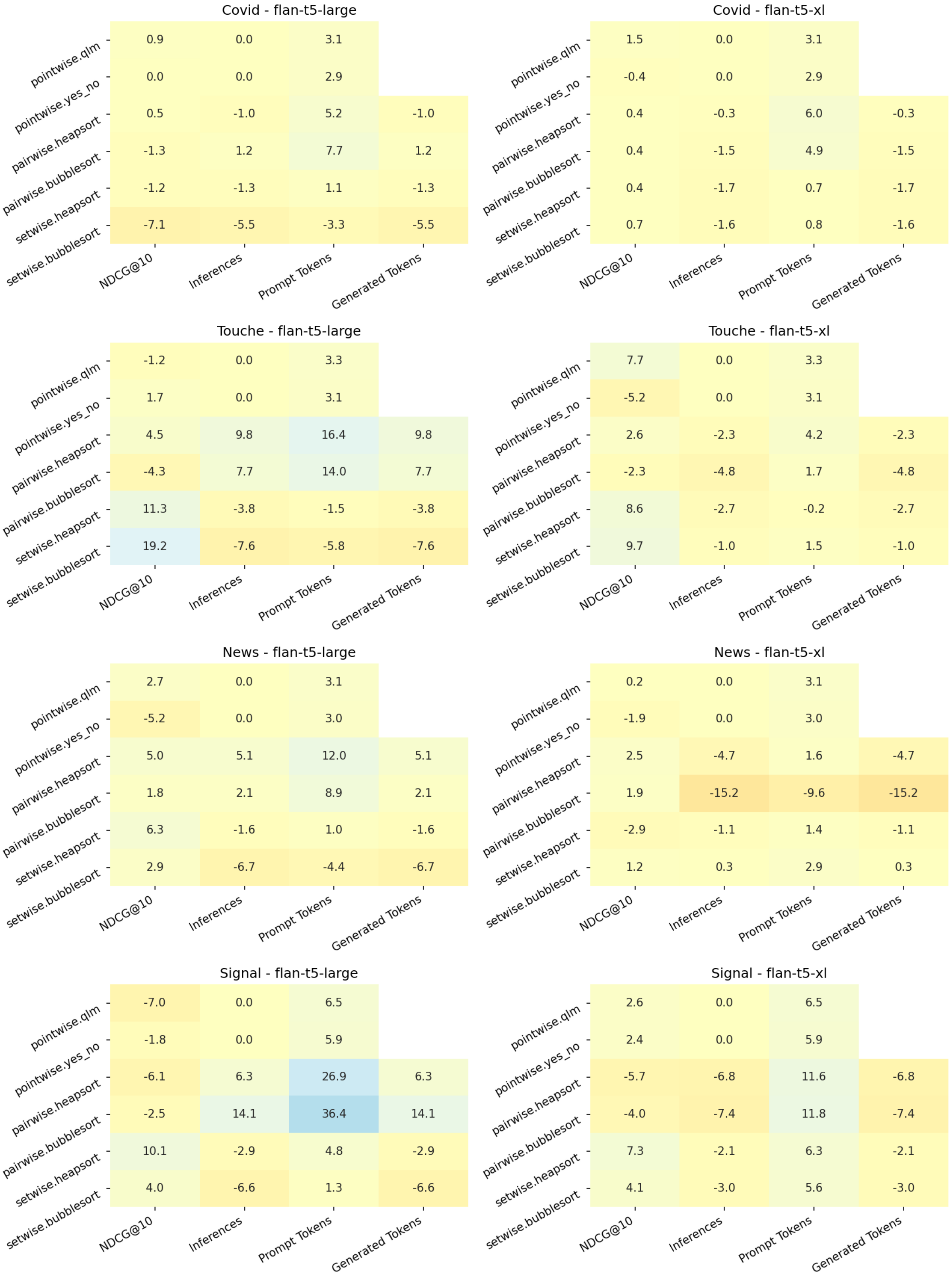


Figure 7: Eight paired heatmaps show the effect of prompt changes on reranking methods. Each pair compares flan-t5-large (left) and flan-t5-xl (right) across four BEIR datasets (Covid, Touche, News, Signal). Heatmaps display percentage changes in NDCG@10, inferences, prompt tokens, and generated tokens, based on our replication baseline.

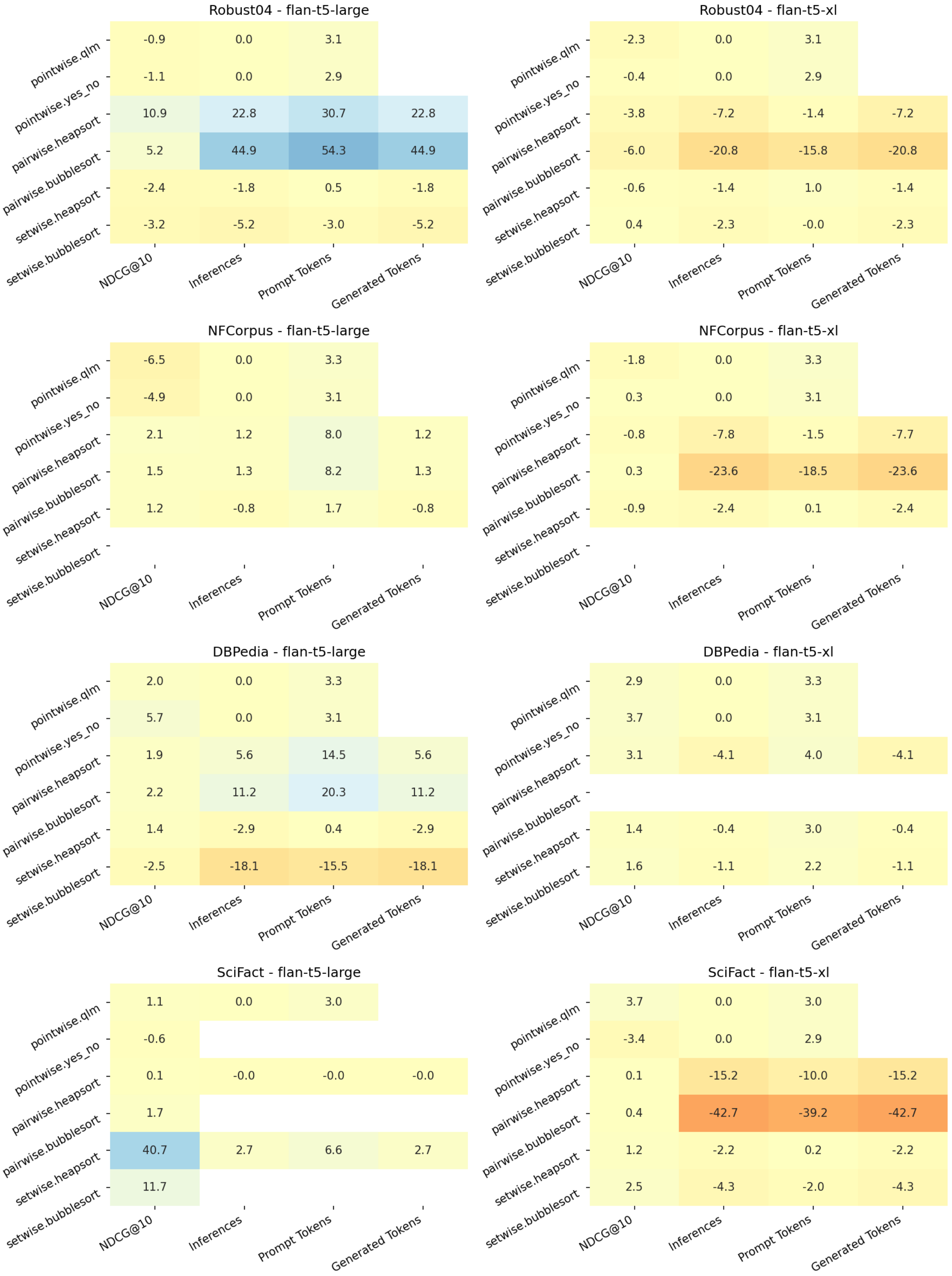


Figure 8: Eight paired heatmaps show the effect of prompt changes on reranking methods. Each pair compares flan-t5-large (left) and flan-t5-xl (right) across four BEIR datasets (Robust04, NFCorpus, DBPedia, SciFact). Heatmaps display percentage changes in NDCG@10, inferences, prompt tokens, and generated tokens, based on our replication baseline.

2022. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110* (2022).

[3] Xueguang Ma, Xinyu Zhang, Ronak Pradeep, and Jimmy Lin. 2023. Zero-shot listwise document reranking with a large language model. *arXiv preprint arXiv:2305.02156* (2023).

[4] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. Rankvicuna: Zero-shot listwise document reranking with open-source large language models. *arXiv preprint arXiv:2309.15088* (2023).

[5] Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, et al. 2023. Large language models are effective text rankers with pairwise ranking prompting. *arXiv preprint arXiv:2306.17563* (2023).

[6] Devendra Singh Sachan, Mike Lewis, Mandar Joshi, Armen Aghajanyan, Wen-tau Yih, Joelle Pineau, and Luke Zettlemoyer. 2022. Improving passage retrieval with zero-shot question generation. *arXiv preprint arXiv:2204.07496* (2022).

[7] Weiwei Sun, Lingyong Yan, Xinyu Ma, Pengjie Ren, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agent. *ArXiv* abs/2304.09542 (2023).

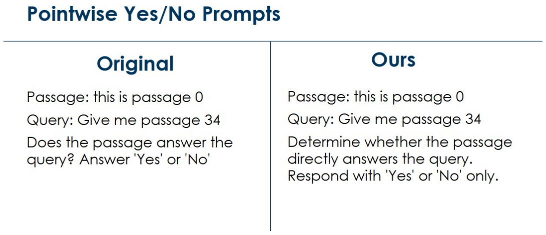
[8] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Advances in Neural Information Processing Systems (NeurIPS)*. https://openreview.net/pdf?id=wCu6T5xFjeJ

[9] Shengyao Zhuang, Bing Liu, Bevan Koopman, and Guido Zuccon. 2023. Open-source large language models are strong zero-shot query likelihood models for document ranking. *arXiv preprint arXiv:2310.13243* (2023).

[10] Shengyao Zhuang, Honglei Zhuang, Bevan Koopman, and Guido Zuccon. 2024. 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 (SIGIR '24)*.

A APPENDIX: Relation to Class Material

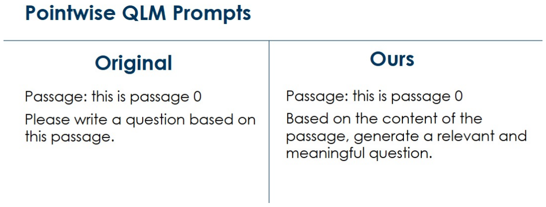
The lecture slides on ranking and learning significantly complement the project's exploration of LLM-based document ranking. The coverage of loss function design strategies – pointwise, pairwise, and listwise approaches – parallels the ranking methods discussed in our project. Both highlight how these fundamental approaches can be applied whether using traditional machine learning or modern LLMs. The slides' theoretical foundation helps contextualize our project's practical implementation of zero-shot LLM ranking within the broader field of learning-to-rank methodologies.



Pointwise Yes/No Prompts

Original	Ours
Passage: this is passage 0 Query: Give me passage 34 Does the passage answer the query? Answer 'Yes' or 'No'	Passage: this is passage 0 Query: Give me passage 34 Determine whether the passage directly answers the query. Respond with 'Yes' or 'No' only.

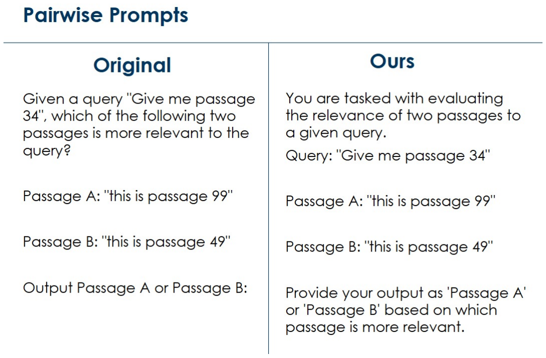
Figure 9



Pointwise QLM Prompts

Original	Ours
Passage: this is passage 0 Please write a question based on this passage.	Passage: this is passage 0 Based on the content of the passage, generate a relevant and meaningful question.

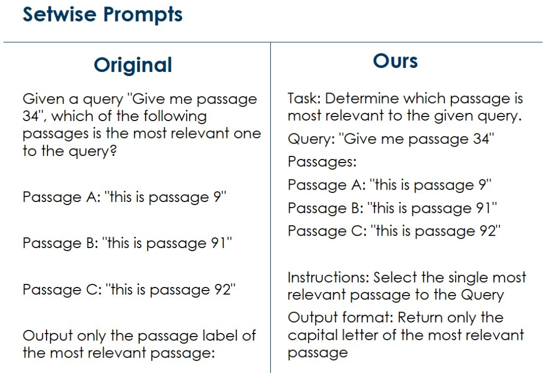
Figure 10



Pairwise Prompts

Original	Ours
Given a query "Give me passage 34", which of the following two passages is more relevant to the query?	You are tasked with evaluating the relevance of two passages to a given query. Query: "Give me passage 34"
Passage A: "this is passage 99"	Passage A: "this is passage 99"
Passage B: "this is passage 49"	Passage B: "this is passage 49"
Output Passage A or Passage B:	Provide your output as 'Passage A' or 'Passage B' based on which passage is more relevant.

Figure 11



Setwise Prompts

Original	Ours
Given a query "Give me passage 34", which of the following passages is the most relevant one to the query?	Task: Determine which passage is most relevant to the given query. Query: "Give me passage 34" Passages:
Passage A: "this is passage 9"	Passage A: "this is passage 9"
Passage B: "this is passage 91"	Passage B: "this is passage 91"
Passage C: "this is passage 92"	Passage C: "this is passage 92"
Output only the passage label of the most relevant passage:	Instructions: Select the single most relevant passage to the Query Output format: Return only the capital letter of the most relevant passage

Figure 12

B APPENDIX: GPU Minutes (Figure 13)

Observations

- *Efficiency Tiers:* Pointwise experiments methods were the fastest (3-22 minutes), followed by listwise methods (20-100 minutes), with pairwise methods being the slowest (often exceeding 200 minutes).
- *Model Size Impact:*
  - Substantial computation increase for complex methods as model size grew.
  - It appears there is minimal computation increase from large to xl for simple methods however to fully use our resources we ran two XL experiments per GPU and four large model experiments per GPU.
  - Very limited xxl experiments were completed. Experiments with xxl models frequently crashed due to insufficient GPU memory despite having the entire 24GB GPU.
- *Dataset Impact:*
  - DBPedia was consistently the most time-consuming dataset.
  - News datasets were generally the fastest to process.
  - There was a 5-10x variation in computation time for the same method across different datasets.
- *Setwise vs Pairwise:* Setwise methods showed moderate efficiency improvements over pairwise methods.

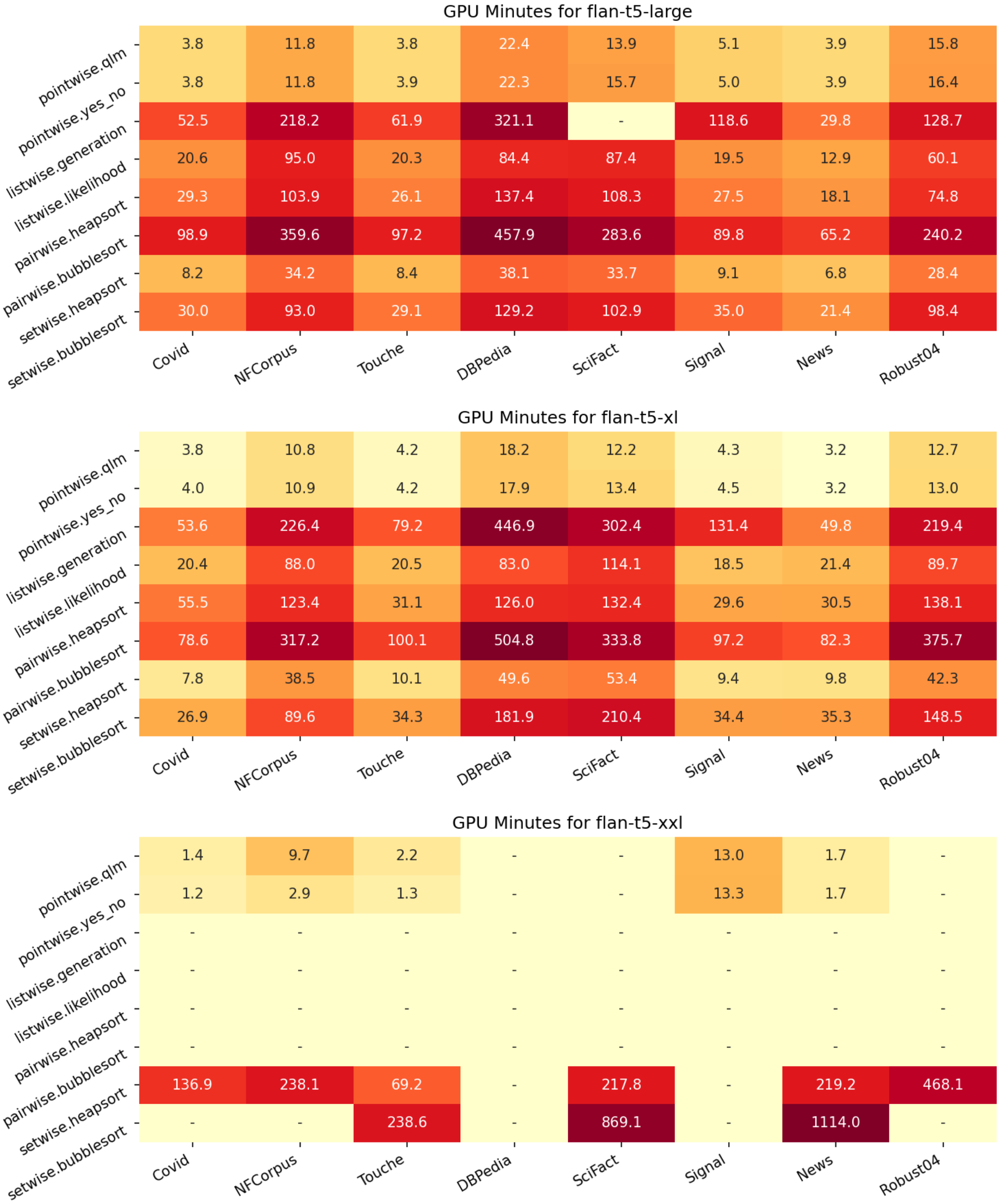
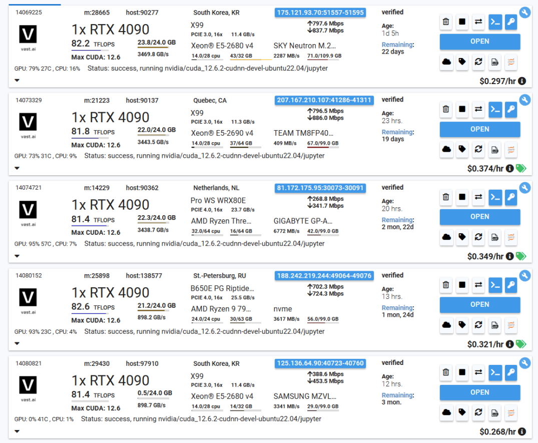


Figure 13: GPU computation time (in minutes) required to evaluate different re-ranking methods across eight BEIR datasets. The heatmaps are organized by Flan-T5 model size (large/xl/xxl from top to bottom). Each cell represents the total GPU minutes required for a specific method-dataset combination. Color intensity corresponds to computational demand, with darker reds indicating longer processing times. Missing values (denoted by "-") indicate experiments that weren't completed. The layout corresponds to *Table 3* in [10], allowing direct comparison between computational costs and reported NDCG@10 performance.

C APPENDIX: GPU Resources

In their paper [10] state: “We carried out the efficiency evaluations on a local GPU workstation equipped with an AMD Ryzen Threadripper PRO 3955WX 16-Core CPU, a NVIDIA RTX A6000 GPU with 49GB of memory, and 128GB of DDR4 RAM.”

Our research was carried out using NVIDIA A100s and RTX 4090 GPUs. Figure 14 shows the hardware we used to reproduce results and Figure 15 shows the hardware we used to measure the impact of our prompt modifications.



14069225 m:28665 host:90277 South Korea, KR 175.121.93.70:51557-51595 verified
1x RTX 4090
82.2 TFLOPS 23.8/24.0 GB 3469.8 GB/s Max CUDA: 12.6
X99 PCIE 3.0, 16x 11.4 GB/s
Xeon® E5-2680 v4 14.0/28 cpu 43/32 GB
↑797.6 Mbps ↓837.7 Mbps
SKY Neutron M.2... 2287 MB/s 71.0/109.9 GB
Age: 1d 5h
Remaining: 22 days
OPEN
GPU: 79% 27C , CPU: 16% Status: success, running nvidia/cuda_12.6.2-cudnn-devel-ubuntu22.04/jupyter
$0.297/hr

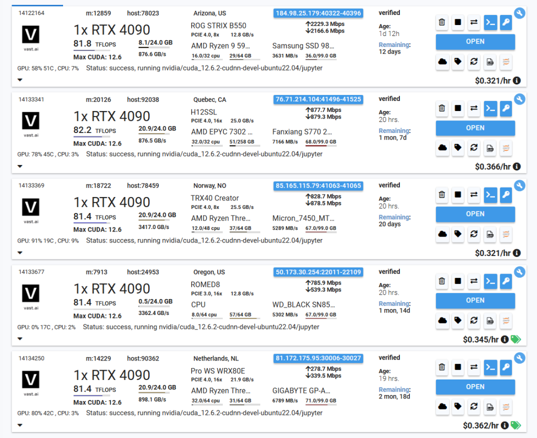
14073329 m:21223 host:90137 Quebec, CA 207.167.210.107:41286-41311 verified
1x RTX 4090
81.8 TFLOPS 22.0/24.0 GB 3443.5 GB/s Max CUDA: 12.6
X99 PCIE 3.0, 16x 11.3 GB/s
Xeon® E5-2690 v4 14.0/28 cpu 37/64 GB
↑796.5 Mbps ↓886.0 Mbps
TEAM TM8FP40... 409 MB/s 67.0/99.0 GB
Age: 23 hrs.
Remaining: 19 days
OPEN
GPU: 73% 31C , CPU: 9% Status: success, running nvidia/cuda_12.6.2-cudnn-devel-ubuntu22.04/jupyter
$0.374/hr

14074721 m:14229 host:90362 Netherlands, NL 81.172.175.95:30073-30091 verified
1x RTX 4090
81.4 TFLOPS 22.3/24.0 GB 3438.7 GB/s Max CUDA: 12.6
Pro WS WRX80E PCIE 4.0, 16x 23.7 GB/s
AMD Ryzen Thre... 32.0/64 cpu 16/64 GB
↑268.8 Mbps ↓341.7 Mbps
GIGABYTE GP-A... 6772 MB/s 42.0/99.0 GB
Age: 20 hrs.
Remaining: 2 mon, 22d
OPEN
GPU: 95% 57C , CPU: 7% Status: success, running nvidia/cuda_12.6.2-cudnn-devel-ubuntu22.04/jupyter
$0.349/hr

14080152 m:25898 host:138577 St.-Petersburg, RU 188.242.219.244:49064-49076 verified
1x RTX 4090
82.6 TFLOPS 21.2/24.0 GB 898.2 GB/s Max CUDA: 12.6
B650E PG Riptide... PCIE 4.0, 16x 25.5 GB/s
AMD Ryzen 9 79... 24.0/24 cpu 30/63 GB
↑702.3 Mbps ↓724.3 Mbps
nvme 3617 MB/s 56.0/99.0 GB
Age: 13 hrs.
Remaining: 1 mon, 24d
OPEN
GPU: 93% 23C , CPU: 4% Status: success, running nvidia/cuda_12.6.2-cudnn-devel-ubuntu22.04/jupyter
$0.321/hr

14080821 m:29430 host:97910 South Korea, KR 125.136.64.90:40723-40760 verified
1x RTX 4090
81.4 TFLOPS 0.5/24.0 GB 898.7 GB/s Max CUDA: 12.6
X99 PCIE 3.0, 16x 11.4 GB/s
Xeon® E5-2680 v4 14.0/28 cpu 14/32 GB
↑388.6 Mbps ↓453.5 Mbps
SAMSUNG MZVL... 3341 MB/s 29.0/99.0 GB
Age: 12 hrs.
Remaining: 3 mon.
OPEN
GPU: 0% 41C , CPU: 1% Status: success, running nvidia/cuda_12.6.2-cudnn-devel-ubuntu22.04/jupyter
$0.268/hr

Figure 14: Hardware configurations of RTX 4090 GPU instances used for baseline replication experiments. Each instance features 24GB VRAM, with varying CPU configurations including Intel Xeon E5-2680/2690 v4 and AMD Ryzen processors. The systems achieve between 81.4-82.6 TFLOPS of computing performance.



14122164 m:12859 host:78023 Arizona, US 184.98.25.179:40322-40396 verified
1x RTX 4090
81.8 TFLOPS 8.1/24.0 GB 876.6 GB/s Max CUDA: 12.6
ROG STRIX B550 PCIE 4.0, 8x 12.8 GB/s
AMD Ryzen 9 59... 16.0/32 cpu 29/64 GB
↑2229.3 Mbps ↓2166.6 Mbps
Samsung SSD 98... 3631 MB/s 36.0/99.0 GB
Age: 1d 12h
Remaining: 12 days
OPEN
GPU: 58% 51C , CPU: 7% Status: success, running nvidia/cuda_12.6.2-cudnn-devel-ubuntu22.04/jupyter
$0.321/hr

14133341 m:20126 host:92038 Quebec, CA 76.71.214.104:41496-41525 verified
1x RTX 4090
82.2 TFLOPS 20.9/24.0 GB 876.5 GB/s Max CUDA: 12.6
H12SSL PCIE 4.0, 16x 25.0 GB/s
AMD EPYC 7302 ... 32.0/32 cpu 51/258 GB
↑877.7 Mbps ↓879.3 Mbps
Fanxiang S770 2... 7166 MB/s 68.0/99.0 GB
Age: 20 hrs.
Remaining: 1 mon, 7d
OPEN
GPU: 78% 45C , CPU: 3% Status: success, running nvidia/cuda_12.6.2-cudnn-devel-ubuntu22.04/jupyter
$0.366/hr

14133369 m:18722 host:78459 Norway, NO 85.165.115.79:41063-41065 verified
1x RTX 4090
81.4 TFLOPS 20.9/24.0 GB 3417.0 GB/s Max CUDA: 12.6
TRX40 Creator PCIE 4.0, 8x 25.5 GB/s
AMD Ryzen Thre... 12.0/48 cpu 37/64 GB
↑828.7 Mbps ↓878.5 Mbps
Micron_7450_MT... 5289 MB/s 67.0/99.0 GB
Age: 20 hrs.
Remaining: 20 days
OPEN
GPU: 91% 19C , CPU: 9% Status: success, running nvidia/cuda_12.6.2-cudnn-devel-ubuntu22.04/jupyter
$0.321/hr

14133677 m:7913 host:24953 Oregon, US 50.173.30.254:22011-22109 verified
1x RTX 4090
81.4 TFLOPS 0.5/24.0 GB 3362.4 GB/s Max CUDA: 12.6
ROMED8 PCIE 3.0, 16x 12.8 GB/s
CPU 8.0/64 cpu 57/64 GB
↑785.9 Mbps ↓539.3 Mbps
WD_BLACK SN85... 5302 MB/s 67.0/99.0 GB
Age: 20 hrs.
Remaining: 1 mon, 14d
OPEN
GPU: 0% 17C , CPU: 2% Status: success, running nvidia/cuda_12.6.2-cudnn-devel-ubuntu22.04/jupyter
$0.345/hr

14134250 m:14229 host:90362 Netherlands, NL 81.172.175.95:30006-30027 verified
1x RTX 4090
81.4 TFLOPS 20.9/24.0 GB 898.1 GB/s Max CUDA: 12.6
Pro WS WRX80E PCIE 4.0, 16x 21.9 GB/s
AMD Ryzen Thre... 32.0/64 cpu 31/64 GB
↑278.7 Mbps ↓339.5 Mbps
GIGABYTE GP-A... 6789 MB/s 71.0/99.0 GB
Age: 19 hrs.
Remaining: 2 mon, 18d
OPEN
GPU: 80% 42C , CPU: 3% Status: success, running nvidia/cuda_12.6.2-cudnn-devel-ubuntu22.04/jupyter
$0.362/hr

Figure 15: Hardware specifications of the NVIDIA RTX 4090 GPU instances used for experiments with modifications. The systems feature AMD processors (including Ryzen 9, EPYC 7302, and Threadripper) paired with 24GB VRAM GPUs delivering 81.4-82.2 TFLOPS of computing performance.