Efficiency-Effectiveness Reranking FLOPs for LLM-based Rerankers

Zhiyuan Peng[*1], **Ting-Ruen Wei**[1], **Tingyu Song**[2], **Yilun Zhao**[3]
[1]Santa Clara University, [2]Independent Researcher, [3]Yale University

Abstract

Large Language Models (LLMs) have recently been applied to reranking tasks in information retrieval, achieving strong performance. However, their high computational demands often hinder practical deployment. Existing studies evaluate the efficiency of LLM-based rerankers using proxy metrics such as latency, the number of forward passes, input tokens, and output tokens. However, these metrics depend on hardware and running-time choices (*e.g.,* parallel or not, batch size, etc), and often fail to account for model size, making it difficult to interpret and obscuring the evaluation of the efficiency-effectiveness tradeoff. To address this issue, we propose E^{2}R-FLOPs[1] for LLM-based rerankers: RPP (ranking metrics per PetaFLOP), measuring how much ranking quality (e.g., NDCG or MRR) a method achieves per PetaFLOP, and QPP (queries per PetaFLOP), measuring how many queries can be processed per PetaFLOP. Accompanied by the new metrics, an interpretable FLOPs estimator is developed to estimate the FLOPs of an LLM-based reranker even without running any experiments. Based on the proposed metrics, we conduct comprehensive experiments to evaluate a wide range of LLM-based rerankers with different architectures, studying the efficiency-effectiveness trade-off and bringing this issue to the attention of the research community.

1 Introduction

A typical search system balances efficiency and quality with a two-stage pipeline: a lightweight retriever retrieves hundreds of documents from a vast corpus, prioritizing efficiency, and then a more powerful but computationally expensive reranker refines their order. Thanks to the rapid progress of LLMs (Brown et al., 2020; Grattafiori et al., 2024; Anil et al., 2023), LLM-based rerankers have achieved impressive gains in reranking metrics, such as NDCG; however, these gains often come at the cost of substantial computational expense, making them difficult to deploy at scale in production. This underscores the need for evaluation metrics that consider not only reranking quality but also computational efficiency.

Existing studies evaluate the efficiency of LLM-based rerankers using proxies such as latency (Jin et al., 2025), the number of LLM calls (*i.e.,* forward passes) (Zhuang et al., 2024), and input and output token usage (Chen et al., 2025b). However, these metrics lack the computational granularity needed to distinguish differences in internal compute per token or per model call. Specifically, latency is heavily dependent on hardware and runtime choices (GPU vs. CPU, batch size, parallelism), making it an inconsistent basis for comparing algorithms across studies. The number of LLM calls ignores the model size: a single call to a 70B LLM costs orders of magnitude more compute than a call to a 3B model, yet both appear identical under this metric. Similarly, token usage overlooks the model size and is difficult to interpret as the cost of the input token and the output token can be different.

Inspired by the scaling law in LLMs that studies the connection between total compute and performance (Kaplan et al., 2020), we employ floating-point operations (FLOPs) as a fundamental measure of cost for each forward pass or LLM call. The total number of FLOPs required by a model to rerank documents is a hardware-agnostic, intrinsic metric of computational work (Sukthanker et al., 2024). Building on this insight, we introduce **E^{2}R-FLOPs**, **E**fficiency-**E**ffectiveness **R**eranking **FLOPs** for LLM-based rerankers: ranking metrics per PetaFLOP (RPP) for relevance per compute and queries per PetaFLOP (QPP) for hardware-agnostic throughput. The proposed metrics thus enable fair comparisons be-

[*]Correspondence: zpeng@scu.edu
[1]https://github.com/zhiyuanpeng/EER-FLOPs.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track, pages 2782–2791
November 4-9, 2025

tween methods that might utilize different LLMs, reranking algorithms, and running-time choices. Accompanied by the proposed metrics, an interpretable FLOPs estimator was built to estimate the FLOPs of an LLM-based reranker even without running any experiments.

Based on the proposed metrics, we conduct comprehensive experiments to evaluate a wide range of LLM-based rerankers, examining the efficiency-effectiveness trade-off and drawing attention to this issue within the research community. Our key contributions include:

- We derive a closed-form, interpretable formula for the FLOPs of LLM-based rerankers and provide an open-source calculator covering up-to-date models and decoding settings.
- We propose two efficiency-effectiveness metrics: RPP for relevance per compute and QPP for hardware-agnostic throughput.
- We conduct the first large-scale study of the efficiency–effectiveness trade-off in LLM-based rerankers, bring this issue to the attention of the research community.
- All code, data, and the FLOPs estimator are publicly released for reproducible research on computationally efficient rerankingEfficiency-Effectiveness Reranking FLOPs.

2 Related Work

2.1 LLM-based Rerankers

Based on how the documents are compared with each other, LLM-based rerankers can be categorized as pointwise, pairwise and listwise. **Pointwise** methods primarily compute the query-document relevance score by either the likelihood of generating the query conditioned on the document (Ponte and Croft, 2017; Zhuang and Zuccon, 2021; Zhuang et al., 2021; Peng et al., 2024) or the normalized possibility of generating the "Yes" when prompting the LLM whether the query-document pair is relevant or not (Liang et al., 2023; Nogueira et al., 2020). The ranking can be easily accomplished by sorting the relevance score of each document. **Pairwise** methods compare the relevance of a pair of documents to a given query and output the document ID of the more relevant one. To rank a list of documents, sorting (Qin et al., 2024) and sampling (Gienapp et al., 2022; Mikhailiuk et al., 2020) methods are proposed. Sorting uses the pairwise comparison to replace the comparison operation in sorting algorithms, such as bubble sorting and heap sorting. In contrast, sampling methods reduce the number of comparisons by repeatedly drawing random pairs (or small subsets), aggregating wins, and estimating a global ranking. Sorting methods are more efficient for getting the top-K documents as they do not need to compare all the pairs. Setwise (Zhuang et al., 2024) extends the pairwise comparison utilized in heapsort and bubblesort to output the best one from three or more documents in one LLM call and thus reduces the number of LLM calls. To rank a list of documents, setwise build **Listwise** methods directly output a ranked list of document IDs. Most of the existing listwise methods are zero-shot (Ma et al., 2023) or few-shot (Sun et al., 2023; Ma et al., 2023; Pradeep et al., 2023) prompting methods. Recently, researchers (Zhang et al., 2025) have resorted to adopting reinforcement learning to fine-tune LLMs to generate reasoning followed by ranked document IDs to tackle reasoning-intensive tasks like BRIGHT (Su et al., 2025). To get a full rank list, listwise methods usually adopt strategies like sliding-window (Sun et al., 2023) and tournament ranking (Chen et al., 2025b) to get a full rank list with a limited window size.

2.2 FLOPs Calculation

Several FLOPs profilers exist for deep learning models. Still, most are limited to standard forward passes and do not support token-level generation with KV-cache, which is essential for accurate LLM inference estimation. PyPAPI[2] measures CPU-level FLOPs for general Python code but is not designed for PyTorch or GPU workloads. ptflops[3] and fvcore[4] compute FLOPs by running a model's "forward" function, but do not support autoregressive decoding. DeepSpeed's FLOPs profiler (Rasley et al., 2020) and calflops[5] both support the FLOPs of the decoding process, but they also require full forward execution. All existing tools require model execution and lack closed-form support for generation-aware FLOPs estimation. For

[2] https://github.com/flozz/pypapi
[3] https://github.com/sovrasov/flops-counter.pytorch
[4] https://github.com/facebookresearch/fvcore
[5] https://github.com/MrYxJ/calculate-flops.pytorch

reranking, prior studies utilize coarse FLOP estimates, *e.g.,* double the total parameter count (Shao et al., 2025) or open-source tooling (Abdallah et al., 2025), which lack interpretability regarding the specific facts that impact the FLOPs count. Our work differs from theirs in that our FLOPs estimator is well-interpretable, and we propose new metrics to comprehensively evaluate the efficiency and effectiveness of LLM-based rerankers.

3 Method

We first introduce the metrics we designed to measure the efficiency-effectiveness tradeoff of LLM-based rerankers. Then we elaborate the FLOPs estimator that estimates the number of FLOPs needed for one LLM call. To rank a set of documents for a given LLM-based reranker, the number of LLM calls can be estimated, allowing for the estimation of total FLOPs, which can then be compared with those of different LLM-based rerankers.

3.1 Metrics

We report two FLOPs-normalized metrics to compare different LLM-based rerankers, thereby effectively evaluating the effectiveness-efficiency tradeoff without being tied to a specific hardware.

3.1.1 Ranking metrics per PetaFLOP (RPP)

$$\text{RPP} = \frac{m(q)}{C_{\text{q}}/10^{15}} \tag{1}$$

where $m(q)$ can be *any* ranking metric for query q (*e.g.,* NDCG, MRR, MAP). RPP therefore expresses *ranking metrics per peta-FLOP*; a higher value indicates better ranking quality for a fixed compute budget.

3.1.2 Queries per PetaFLOP (QPP)

$$\text{QPP} = \frac{1}{AVG_{C_{\text{q}}}/10^{15}} \tag{2}$$

QPP measures *throughput*: how many queries can be processed with one peta-FLOP. Together, RPP and QPP trace a method's efficiency–effectiveness frontier: RPP weights quality per compute, while QPP captures raw FLOPs-normalized throughput.

3.2 FLOPs Estimator

We parameterize a Transformer with four hyperparameters: the number of layers n_{layer}, the residual-stream width d_{model}, the hidden size of the feed-forward block d_{ff}, and the dimension of attention output d_{attn} which is the dimension of `Q`, `K`, `V` projections before splitting into multiple heads and by default $d_{\text{attn}} = d_{\text{model}}$. Because decoder-only and encoder–decoder designs dominate LLM-based rerankers, we derive estimates for both. To keep the analysis general yet concrete, we adopt the baseline decoder-only configuration of Kaplan et al. (2020) and the T5 encoder–decoder architecture (Raffel et al., 2020). In a typical reranking call, the model receives a prompt (the context, denoted `ctx`) and produces an output sequence (`opt`). The prompt concatenates a task-specific prefix p, the query q, and a list of w documents, resulting in a length of n_{ctx}. The generated sequence has length n_{opt}.

3.2.1 Decoder-only

Following Kaplan et al. (2020), we ignore subleading costs such as nonlinearities, biases, and layer-normalization. The number of attention and feedforward relevant parameters N_{dec} is:

$$N_{\text{dec}} \approx 2d_{\text{model}}\, n_{\text{layer}}\, (2d_{\text{attn}} + d_{\text{ff}}) \tag{3}$$

Given an LLM with KV cache enabled, we now compute the FLOPs of generating a sequence named `opt` (short for "output"), consisting of n_{opt} tokens, conditioned on a prompt `ctx` (short for "context") of length n_{ctx}. The context includes a task-specific prompt p, a query q, and a list of w documents. Each token within `ctx` requires $2N_{\text{dec}} + 4n_{\text{layer}}\, n_{\text{ctx}} d_{\text{attn}}$ FLOPs, where $2N_{\text{dec}}$ comes from the fact that each parameter in N_{dec} has one addition and one multiplication operation and $4n_{\text{layer}}\, n_{\text{ctx}} d_{\text{attn}}$ is taken by the basic multi-head attention operation (Kaplan et al., 2020). The total FLOPs for n_{ctx} tokens $C(\text{ctx})$ is:

$$C\,(\text{ctx}) = 2N_{\text{dec}} n_{\text{ctx}} + 4n_{\text{layer}}\, n_{\text{ctx}}^2 d_{\text{attn}} \tag{4}$$

When generating token opt_i, the total sequence length seen by LLM is $n_{\text{ctx}} + (i-1)$ and the FLOPs for token i is:

$$C\,(\text{opt}_i) = 2N_{\text{dec}} + 4n_{\text{layer}}\, [n_{\text{ctx}} + (i-1)]\, d_{\text{attn}} \tag{5}$$

The FLOPs of generating n_{opt} tokens is computed by summing over all the n_{opt} tokens:

$$\begin{aligned} C(\text{opt}) &= 2N_{\text{dec}} n_{\text{opt}} \\ &\quad + 2n_{\text{layer}}\, d_{\text{attn}} \left[2n_{\text{opt}} n_{\text{ctx}} + n_{\text{opt}}(n_{\text{opt}} - 1)\right] \end{aligned} \tag{6}$$

	Multi-head Attention		Grouped-query Attention	
Ops	Parameters	FLOPs per Token	Parameters	FLOPs per Token
Atten: QKV	$n_{\text{layer}} d_{\text{model}} 3 d_{\text{attn}}$	$2 n_{\text{layer}} d_{\text{model}} 3 d_{\text{attn}}$	$n_{\text{layer}} d_{\text{model}} \left(1 + 2 n_{\text{kv}} / n_q\right) d_{\text{attn}}$	$2 n_{\text{layer}} d_{\text{model}} \left(1 + 2 n_{\text{KV}} / n_Q\right) d_{\text{attn}}$
Atten: O	$n_{\text{layer}} d_{\text{attn}} d_{\text{model}}$	$2 n_{\text{layer}} d_{\text{attn}} d_{\text{model}}$	$n_{\text{layer}} d_{\text{attn}} d_{\text{model}}$	$2 n_{\text{layer}} d_{\text{attn}} d_{\text{model}}$
Atten: Mask	-	$4 n_{\text{layer}} d_{\text{attn}} d_{\text{model}}$	–	$4 n_{\text{layer}} n_{\text{ctx}} (n_{\text{KV}} / n_Q) d_{\text{attn}}$
Feedforward	$n_{\text{layer}} 2 d_{\text{model}} d_{\text{dff}}$	$2 n_{\text{layer}} 2 d_{\text{model}} d_{\text{dff}}$	$n_{\text{layer}}\ 2 d_{\text{model}} d_{\text{dff}}$	$2 n_{\text{layer}}\ 2 d_{\text{model}} d_{\text{dff}}$

Table 1: FLOP count for the attention mechanism for multi-head attention and grouped-query attention

The total FLOPs of taking prompt `ctx` and generating `opt` is:

$$C(\text{ctx} + \text{opt}) = C(\text{ctx}) + C(\text{opt}) \tag{7}$$

For LLM-based reranker, n_{ctx} consists of task-specific prompt p, query q, and a list of w documents. By approximating the length of each document as the average document length L_{doc}, n_{ctx} can be estimated as:

$$n_{\text{ctx}} = n_p + n_q + w l_{\text{doc}} \tag{8}$$

Suppose n_{Q} represents the number of heads for `Q` and n_{KV} denotes the number of heads for `K` and `V`. Compared to multi-head attention, the number of parameters and the FLOPs per token changed accordingly, as shown in Table 1. The equations are rewritten as:

$$N_{\text{dec}} \approx 2 d_{\text{model}}\, n_{\text{layer}} \left((1 + \frac{n_{\text{KV}}}{n_{\text{Q}}}) d_{\text{attn}} + d_{\text{ff}} \right) \tag{9}$$

$$C\left(\text{ctx}\right) = 2 N_{\text{dec}} n_{\text{ctx}} + 4 n_{\text{layer}}\, n_{\text{ctx}}^2 \frac{n_{\text{KV}}}{n_{\text{Q}}} d_{\text{attn}} \tag{10}$$

$$\begin{aligned} C(\text{opt}) = {} & 2 N_{\text{dec}} n_{\text{opt}} + 2 n_{\text{layer}} \frac{n_{\text{KV}}}{n_{\text{Q}}} d_{\text{attn}} \cdot 2 n_{\text{opt}} n_{\text{ctx}} \\ & + 2 n_{\text{layer}} \frac{n_{\text{KV}}}{n_{\text{Q}}} d_{\text{attn}} \cdot n_{\text{opt}} (n_{\text{opt}} - 1) \end{aligned} \tag{11}$$

For models with MoE, only the parameter count of the "Feedforward" component in Table 1 needs to be adjusted. Suppose there are n_{expert} experts, each with intermediate size $d_{\text{dff-MoE}}$. Then the number of "Feedforward" parameters is $n_{\text{layer}} \cdot 2 d_{\text{model}} \cdot n_{\text{expert}} d_{\text{dff-MoE}}$. Equivalently, substituting d_{ff} with $n_{\text{expert}} d_{\text{dff-MoE}}$ yields the updated N_{dec} for MoE models. Because N_{dec} appears in both $C(\text{ctx})$ and $C(\text{opt})$, these expressions are automatically updated once N_{dec} is replaced. For instance, Qwen1.5-MoE-A2.7B[6] has one expert activated for each token, and for each token, there are four additional experts selected from a pool of 60 experts. Thus, the number of the parameters of "Feedforward" is $n_{\text{layer}} 2 d_{\text{model}} 5 d_{\text{dff-MoE}}$. Qwen1.5-MoE-A2.7B adopts intermediate size 5632 for each shared expert and 1408 for each of the remaining 60 experts, so $d_{\text{dff-MoE}} = (5632 + 1408 * 4)/5 = 2252.8$ on average.

[6] https://huggingface.co/Qwen/Qwen1.5-MoE-A2.7B

3.2.2 Encoder-Decoder

Decoder-only LLMs, such as GPT (Radford et al., 2019), do not include encoder-decoder attention and therefore share a similar structure with the encoder component of encoder-decoder models. The main difference lies in the attention masking strategy, which, however, does not affect the FLOPs required to process the prompt. Although decoder-only models are designed to compute attention only over previous tokens, in practice, they compute full self-attention (*e.g.,* $Q_{\text{prompt}} \times K_{\text{prompt}}$) and apply a causal mask to prevent attending to future tokens. So, for encoder-decoder LLMs, the FLOPs of consuming prompt `ctx` is the same as that of encoder-only LLMs:

$$C\left(\text{ctx}\right) = 2 N_{\text{enc}} n_{\text{ctx}} + 4 n_{\text{layer}}\, n_{\text{ctx}}^2 d_{\text{attn}} \tag{12}$$

Where N_{enc} is same as N_{dec} in Equation 3. The decoder employs a different attention mechanism from that of the encoder, utilizing an encoder-decoder attention mechanism followed by self-attention. In an encoder–decoder model, each decoder layer must, once per prompt, project the encoder outputs to cross-attention keys and values. This setup cost is calculated as:

$$C_{\text{cross-KV}} = 4\, n_{\text{layer}}\, n_{\text{ctx}}\, d_{\text{model}}\, d_{\text{attn}} \tag{13}$$

Even, it has two attentions, when generating token opt_i, it only goes through two `Q` projections, two `O` one `K` projection, and one `V` projection, as the left `Q` and `K` projections are for prompt "ntx", so

$$N_{\text{dec}} \approx 2 d_{\text{model}}\, n_{\text{layer}}\, \left(3 d_{\text{attn}} + d_{\text{ff}}\right) \tag{14}$$

The total sequence length seen by self-attention is $(i-1)$ and the attention operation takes $4 n_{\text{layer}}\, (i -$

$1)d_{\text{attn}}$ FLOPs. The sequence length seen by the following encoder-decoder attention is fixed as n_{ctx}, which requires $4n_{\text{layer}}\, n_{\text{ctx}} d_{\text{attn}}$ for computing attention. The total attention FLOPs is $4n_{\text{layer}}\,(n_{\text{ctx}} + i - 1)d_{\text{attn}}$ which is the same as that of decoder-only models as shown in the right part of equation 5 and thus the total FLOPs at generating token opt_i is also same as equation 5 and the only difference is that the value of N_{dec} is different. Similary, the $C(\text{opt})$ is same as equation 6 but with a different value of N_{dec}. The cost of encoder-decoder model is:

$$C(\text{ctx}{+}\text{opt}) = C(\text{ctx}){+}C_{\text{cross-KV}}{+}C(\text{opt}) \quad (15)$$

4 Experiment Setup

Following the setwise setup in Zhuang et al. (2024), we utilize the Flan-T5 (Chung et al., 2024) as the backbone for most of the LLM-based rerankers except for IRL (Chen et al., 2025a) and Tourrank (Chen et al., 2025b) as these two methods require a longer context than Flan-T5's input limitation allows. For IRL and Tourrank, we employ the Llama-3.1-8B-Instruct[7] model. To compare our estimated FLOPs with those reported by open-source packages, we also include Qwen2.5 (Yang et al., 2025) (3B[8], 7B[9], 14B[10]), which implements group query attention instead of multi-head attention. For all LLM-based rerankers, we report their performance using the new metrics on the TREC-DL19, TREC-DL20 (Craswell et al., 2020), and two other datasets from BEIR (Thakur et al., 2021). The top 100 documents are retrieved using Pyserini's BM25 (Lin et al., 2021).

We utilize DeepSpeed's FLOPs profiler (Rasley et al., 2020) and calflops to compute the measured FLOPs and get identical results, so we only report one kind of measured FLOPs. We also present the FLOPs of BM25 in Appendix A.1.

5 Experimental Results and Analysis

We conduct extensive experiments to study four key research questions. Our results and analysis are as follows:

[7] https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

[8] https://huggingface.co/Qwen/Qwen2.5-3B-Instruct

[9] https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

[10] https://huggingface.co/Qwen/Qwen2.5-14B-Instruct

Q1: Can E²R-FLOPs overcome the limitations of existing efficiency proxies? Existing efficiency proxies for LLM-based rerankers, such as latency (Jin et al., 2025), number of LLM calls (*i.e.,*, forward passes) (Zhuang et al., 2024), and token counts (Chen et al., 2025b), are weak surrogates for actual compute. *Latency* is confounded by hardware, parallelism, etc.; the same algorithm can appear faster or slower across different platforms. *LLM-call counts* discard model size and sequence length, for instance, one call on a 70B model is orders of magnitude more expensive than one call on a 3B model, yet both count as "1." *Token counts* ignore model size and the prefill-decode asymmetry (quadratic attention during prefill versus near-linear growth during decoding), so equal token totals can yield very different FLOPs.

These limitations surface empirically in Table 2. With Flan-T5-large on DL19, `pointwise.yes_no` and `pointwise.qlm` each use 100 calls but differ in RPP (72.67 vs. 61.89). Fewer calls do not imply proportional gains. On DL19 (Flan-T5-large), `setwise.heapsort` (125.4 calls) vs `setwise.bubblesort` (460.5 calls) yields RPP 26.8 vs 7.45. Holding the call count at 100, scaling the backbone from Flan-T5-large to -xl to -xxl collapses RPP/QPP (72.67→18.06→4.50; 111.1→27.78→6.99), reflecting the sharp FLOPs growth from wider/deeper attention. Similar patterns are observed across different LLMs and datasets (Appendix A.2)

FLOPs, in contrast, is an intrinsic, hardware-agnostic measure of work. Normalizing reranking metrics and throughput by FLOPs (RPP/QPP) enables fair, interpretable comparisons across different LLMs. Additionally, it aligns with trends in measured FLOPs and latency (Section 3, Figure 1, Figure 2).

Q2: What are the LLM-based reranker performances under the E²R-FLOPs? Table 2 reports the efficiency–effectiveness trade-offs of a broad set of LLM-based rerankers under the proposed RPP and QPP metrics. Overall, these systems perform poorly once computation is taken into account. Across all LLMs and methods, TourRank with Llama-3.1-8B-Instruct achieves the highest NDCG on both DL19 and DL20, albeit at the cost of almost the lowest RPP and QPP. Beyond this result, several additional insights emerge.

Pointwise methods dominate the RPP and QPP metrics across different LLMs and datasets.

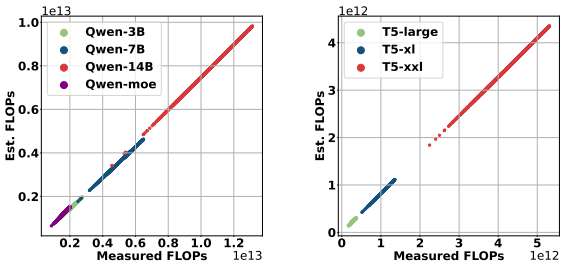
		TREC DL 2019							TREC DL 2020						
	Methods	**NDCG**	**#LLM**	**In**	**Out**	**#FLOPs**	**RPP**	**QPP**	**NDCG**	**#LLM**	**In**	**Out**	**#FLOPs**	**RPP**	**QPP**
	BM25	.506	-	-	-	-	-	-	.480	-	-	-	-	-	-
Flan-t5-large	pointwise.qlm	.557	100	152.12	0	0.009	61.89	**111.1***	.567	100	152.85	0	0.009	63.0	**111.1***
	pointwise.yes_no	.654	100	161.12	0	0.009	**72.67***	**111.1***	.615	100	161.85	0	0.009	**68.33***	**111.1***
	listwise.generation	.561	245	486.21	10.54	0.076	7.38	13.16	.547	245	488.28	10.04	0.076	7.2	13.16
	listwise.likelihood	.669	245	384.49	0	0.058	11.53	17.24	**.626**	245	388.61	0	0.058	10.79	17.24
	pairwise.allpair	.666	9900	304.48	5	1.865	0.36	0.536	.622	9900	304.47	5	1.865	0.33	0.536
	pairwise.heapsort	.657	230.3	455.72	10	0.066	9.95	15.15	.619	226.8	459.62	10	0.066	9.38	15.15
	pairwise.bubblesort	.636	844.2	451.77	10	0.242	2.63	4.132	.589	778.5	459.03	10	0.227	2.59	4.405
	setwise.heapsort	.670	125.4	322.65	5	0.025	26.80	40.0	.618	124.2	325.5	5	0.025	24.72	40.0
	setwise.bubblesort	**.678**	460.5	320.90	5	0.091	7.45	10.99	.624	457.4	325.63	5	0.092	6.78	10.87
Flan-t5-xl	pointwise.qlm	.542	100	152.12	0	0.034	15.94	**29.41**	.542	100	152.85	0	0.034	15.94	**29.41**
	pointwise.yes_no	.650	100	161.12	0	0.036	**18.06**	27.78	.636	100	161.85	0	0.036	**17.67**	27.78
	listwise.generation	.569	245	486.38	11.87	0.282	2.02	3.546	.547	245	489.04	11.49	0.283	1.93	3.534
	listwise.likelihood	.689	245	385.49	0	0.216	3.19	4.629	.672	245	388.97	0	0.218	3.08	4.587
	pairwise.allpair	**.713**	9900	298.33	5	6.826	0.10	0.146	.682	9900	297.93	5	6.817	0.10	0.147
	pairwise.heapsort	.705	241.9	455.26	10	0.259	2.72	3.861	**.692**	244.3	455.76	10	0.262	2.64	3.817
	pairwise.bubblesort	.683	886.9	451.42	10	0.942	0.73	1.061	.662	863.9	457.18	10	0.930	0.71	1.075
	setwise.heapsort	.693	129.5	321.74	5	0.096	7.22	10.42	.678	127.8	325.27	5	0.096	7.06	10.42
	setwise.bubblesort	.705	446.9	335.53	5	0.346	2.04	2.890	.676	463.5	326.32	5	0.349	1.94	2.865
Flan-t5-xxl	pointwise.qlm	.506	100	152.12	0	0.135	3.75	**7.407**	.492	100	152.85	0	0.136	3.62	**7.352**
	pointwise.yes_no	.644	100	161.12	0	0.143	**4.50**	6.993	.632	100	161.85	0	0.144	**4.39**	6.944
	listwise.generation	.662	245	487.08	11.53	1.105	0.60	0.904	.637	245	489.60	11.05	1.110	0.57	0.901
	listwise.likelihood	.701	245	385.87	0	0.851	0.82	1.175	.690	245	389.73	0	0.860	0.8	1.162
	pairwise.allpair	.699	9900	282.32	5	25.510	0.03	0.039	.688	9900	282.32	5	25.510	0.03	0.039
	pairwise.heapsort	.708	239.4	456.98	10	1.010	0.70	0.990	**.699**	240.5	458.26	10	1.017	0.69	0.983
	pairwise.bubblesort	.679	870.5	453.06	10	3.642	0.19	0.275	.681	842.9	459.56	10	3.577	0.19	0.279
	setwise.heapsort	.706	130.1	323.43	5	0.383	1.84	2.610	.688	128.1	325.01	5	0.379	1.82	2.638
	setwise.bubblesort	**.711**	468.3	321.94	5	1.375	0.52	0.727	.686	467.9	326.37	5	1.393	0.49	0.717
L3.1	IRL	.649	2	4469.12	0	0.096	**6.76**	**10.42**	.639	2	4556.31	0.0	0.098	**6.52**	**10.20**
	Tourrank	**.757***	130	1651.62	27.91	2.274	0.33	0.440	**.777**	130	1659.93	26.79	2.284	0.34	0.438

Table 2: Results on TREC DL. All the methods re-rank the top 100 documents retrieved by BM25. #LLM represents the average number of LLM calls per query for reranking 100 documents. "In" and "Out" denote the average input tokens and output tokens per LLM call. #FLOPs is the estimated FLOPs per query for reranking 100 documents. Bold value is the best within each LLM, and starred value is the best across different LLMs. "L3.1" represent Llama-3.1-8B-Instruct model. We report NDCG@10 for the NDCG metric.

`pointwise.yes_no` of Flan-T5-large yields the highest RPP of 72.67 (DL19) and 68.3 (DL20) and achieves the maximum QPP of 111 queries/PetoFLOPs. The variant `pointwise.qlm` obtains a similar QPP metric but 5∼10 worse RPP points. These methods obtain 10% to 30% relative NDCG gains over the baseline BM25 with negligible FLOPs consumption compared with other LLM-based rerankers.

Scaling up hurts efficiency far more than it helps effectiveness. Most LLM-based rerankers gain NDCG when moving from Flan-T5-large to Flan-T5-xl, yet see only marginal improvement from xl to xxl. For example, setwise. Heapsort rises from 0.670 to 0.693 and then to 0.706 in NDCG. Meanwhile, efficiency collapses: RPP plunges from 26.8 to 7.22 and then to 1.84, while QPP falls from 40.0 to 10.42 and finally to 2.61. In short, scaling boosts quality slowly but sacrifices RPP and QPP on a much larger scale.

Pairwise and listwise methods are intensely FLOP-hungry. Allpair sorting, although it delivers the highest NDCG on Flan-T5-xl (0.713), issues 9,900 LLM calls per query; its RPP collapses to around 0.10, and it processes barely 0.15 queries per petaFLOPs, making large-scale deployment impractical. Heapsort- and bubblesort-based variants cut the number of calls by roughly 90%, yet remain about orders of magnitude less efficient than pointwise methods on both RPP and QPP.



(a) Decoder **(b)** Encoder-Decoder

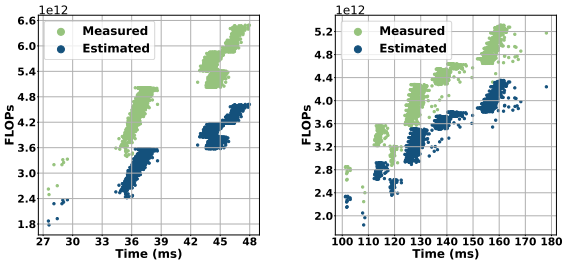
Figure 1: Linear trends between estimated and measured FLOPs for decoder (left) and encoder-decoder (right) models of various sizes on DL19. The same is observed for the DL20 dataset.

Q3: Do the estimated FLOPs reflect the measured FLOPs? Figure 1 shows the relationship between the estimated and measured FLOPs on

DL19 for models of various sizes. The comparison contains both decoder-only and encoder-decoder architectures, providing a comprehensive view of scaling trends. In both cases, the estimated and measured FLOP counts scale with the model size, reflecting the expected rise in computational cost with increasing model parameters. The linear pattern across models illustrates that the estimated FLOPs correlate linearly with the measured FLOPs and are consistent across model families and architectural types, suggesting that our FLOPs estimator is accurate and reliable. The close alignment between the two quantities provides empirical justification for the FLOPs estimator described in Section 3, affirming its reliability as a proxy when real measurements are unavailable.

Q4: How does latency relate to the FLOP counts? Figure 2 shows the relationship between latency and FLOP counts for two representative models: Qwen-7B (a decoder-only architecture) and Flan-T5-XXL (an encoder-decoder architecture), evaluated on the DL19 dataset. For both models, we observe that latency increases in accordance with the number of FLOPs. Importantly, the estimated FLOP counts exhibit a correlation with latency that closely mirrors the relationship between measured FLOP counts and latency. The Pearson correlation coefficients between latency and estimated FLOP counts are 0.88 for Qwen-7B and 0.94 for Flan-T5-XXL. This alignment indicates that our FLOPs estimator approximates computational cost accurately and can serve as a reliable predictor of real-world latency trends. This means that the estimator can be used to anticipate inference time without requiring direct hardware profiling, which is particularly useful when comparing models in a platform-agnostic setting or during early-stage architecture design.

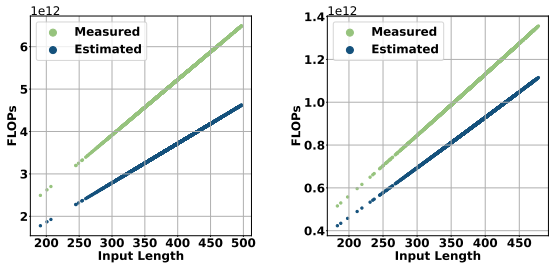
Q5: What is the relationship between prompt length and FLOPs? Figure 3 shows the relationship between prompt length and FLOPs. As expected, both the estimated and actual FLOPs increase with longer prompts, reflecting the greater computational cost required to process more input tokens. Notably, the estimated FLOPs exhibit a strong correlation with prompt length, which closely mirrors the pattern observed in the actual FLOPs. The Pearson correlation coefficient between prompt length and FLOP counts is 1. As the prompt becomes longer, the estimator reliably tracks the resulting increase in computation,



(a) Decoder **(b)** Encoder-Decoder

Figure 2: Latency in milliseconds increases with FLOPs on Qwen-7B (left) and Flan-T5-XXL (right). The Pearson correlation coefficients between latency and estimated FLOP counts are 0.88 for Qwen-7B and 0.94 for Flan-T5-XXL.

consistent with what is observed empirically. This result provides additional validation for the robustness of our FLOPs estimator, demonstrating its ability to respond to changes in input length in a manner consistent with measured FLOPs.



(a) Decoder **(b)** Encoder-Decoder

Figure 3: FLOPs increases with prompt length for Qwen-7B (left) and Flan-T5-XL (right) on the DL19 dataset.

6 Conclusion and Future Works

Due to the limitations of existing metrics in evaluating the efficiency-effectiveness tradeoff of large language models as rerankers, we propose two metrics, RPP and QPP, to quantify the model performance. In addition, we provide a calculator based on a closed-form and interpretable formula to compute the FLOPs, and validate this estimation through experiments on existing decoder-only and encoder-decoder model architectures. The estimated FLOP count exhibits a strong linear correlation with the actual measured values, allowing it to approximate real-world computational cost even without model execution. Future work includes conducting a linear regression between the measured and estimated FLOP counts to refine our estimation and adapting to more advanced architectures.

Limitations

The FLOP estimation relies on model architecture specifications and assumes consistent implementation across different frameworks, which may not hold in practice due to library-level optimizations or kernel differences. Although the estimator shows strong linear correlation with actual FLOP measurements, the approximation may be less accurate for models with more advanced architectures in the future. While FLOPs offer a more stable proxy than latency or token counts, they do not capture other real-world constraints such as memory bandwidth, energy consumption, or inference-time variability under dynamic system loads.

References

Abdelrahman Abdallah, Jamshid Mozafari, Bhawna Piryani, and Adam Jatowt. 2025. ASRank: Zero-shot re-ranking with answer scent for document retrieval. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2950–2970, Albuquerque, New Mexico. Association for Computational Linguistics.

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, and et al. 2023. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Shijie Chen, Bernal Jimenez Gutierrez, and Yu Su. 2025a. Attention in large language models yields efficient zero-shot re-rankers. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Yiqun Chen, Qi Liu, Yi Zhang, Weiwei Sun, Xinyu Ma, Wei Yang, Daiting Shi, Jiaxin Mao, and Dawei Yin. 2025b. Tourrank: Utilizing large language models for documents ranking with a tournament-inspired strategy. In *Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025- 2 May 2025*, pages 1638–1652. ACM.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, and 16 others. 2024. Scaling instruction-finetuned language models. *J. Mach. Learn. Res.*, 25:70:1–70:53.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2020. Overview of the trec 2019 deep learning track. *arXiv preprint arXiv:2003.07820*.

Lukas Gienapp, Maik Fröbe, Matthias Hagen, and Martin Potthast. 2022. Sparse pairwise re-ranking with pre-trained transformers. In *ICTIR '22: The 2022 ACM SIGIR International Conference on the Theory of Information Retrieval, Madrid, Spain, July 11 - 12, 2022*, pages 72–80. ACM.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Can Jin, Hongwu Peng, Anxiang Zhang, Nuo Chen, Jiahui Zhao, Xi Xie, Kuangzheng Li, Shuya Feng, Kai Zhong, Caiwen Ding, and Dimitris N. Metaxas. 2025. Rankflow: A multi-role collaborative reranking workflow utilizing large language models. In *Companion Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025 - 2 May 2025*, pages 2484–2493. ACM.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *CoRR*, abs/2001.08361.

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 Ré, Diana Acosta-Navas, Drew A. Hudson, and 31 others. 2023. Holistic evaluation of language models. *Trans. Mach. Learn. Res.*, 2023.

Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2356–2362.

Xueguang Ma, Xinyu Zhang, Ronak Pradeep, and Jimmy Lin. 2023. Zero-shot listwise document reranking with a large language model. *CoRR*, abs/2305.02156.

Aliaksei Mikhailiuk, Clifford Wilmot, María Pérez-Ortiz, Dingcheng Yue, and Rafal K. Mantiuk. 2020. Active sampling for pairwise comparisons via approximate message passing and information gain maximization. In *25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021*, pages 2559–2566. IEEE.

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document ranking with a pre-trained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 708–718. Association for Computational Linguistics.

Zhiyuan Peng, Xuyang Wu, Qifan Wang, Sravanthi Rajanala, and Yi Fang. 2024. Q-PEFT: query-dependent parameter efficient fine-tuning for text reranking with large language models. *CoRR*, abs/2404.04522.

Jay M. Ponte and W. Bruce Croft. 2017. A language modeling approach to information retrieval. *SIGIR Forum*, 51(2):202–208.

Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. Rankvicuna: Zero-shot listwise document reranking with open-source large language models. *CoRR*, abs/2309.15088.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, and Michael Bendersky. 2024. Large language models are effective text rankers with pairwise ranking prompting. In *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 1504–1518. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 3505–3506. ACM.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Rulin Shao, Rui Qiao, Varsha Kishore, Niklas Muennighoff, Xi Victoria Lin, Daniela Rus, Bryan Kian Hsiang Low, Sewon Min, Wen-tau Yih, Pang Wei Koh, and Luke Zettlemoyer. 2025. Reasonir: Training retrievers for reasoning tasks. *CoRR*, abs/2504.20595.

Hongjin Su, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han-yu Wang, Haisu Liu, Quan Shi, Zachary S. Siegel, Michael Tang, Ruoxi Sun, Jinsung Yoon, Sercan Ö. Arik, Danqi Chen, and Tao Yu. 2025. BRIGHT: A realistic and challenging benchmark for reasoning-intensive retrieval. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Rhea Sukthanker, Arber Zela, Benedikt Staffler, Aaron Klein, Lennart Purucker, Jörg K. H. Franke, and Frank Hutter. 2024. Hw-gpt-bench: Hardware-aware architecture benchmark for language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. 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, EMNLP 2023, Singapore, December 6-10, 2023*, pages 14918–14937. Association for Computational Linguistics.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *CoRR*, abs/2104.08663.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Le Zhang, Bo Wang, Xipeng Qiu, Siva Reddy, and Aishwarya Agrawal. 2025. Rearank: Reasoning re-ranking agent via reinforcement learning. *arXiv preprint arXiv:2505.20046*.

Shengyao Zhuang, Hang Li, and Guido Zuccon. 2021. Deep query likelihood model for information retrieval. In *Advances in Information Retrieval - 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 - April 1, 2021, Proceedings, Part II*, volume 12657 of *Lecture Notes in Computer Science*, pages 463–470. Springer.

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 2024, Washington DC, USA, July 14-18, 2024*, pages 38–47. ACM.

Shengyao Zhuang and Guido Zuccon. 2021. TILDE: term independent likelihood model for passage reranking. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 1483–1492. ACM.

		TREC-COVID							Robust04						
	Methods	**NDCG**	**#LLM**	**In**	**Out**	**#FLOPs**	**RPP**	**QPP**	**NDCG**	**#LLM**	**In**	**Out**	**#FLOPs**	**RPP**	**QPP**
	BM25	.595	-	-	-	-	-	-	.407	-	-	-	-	-	-
Flan-t5-large	pointwise.qlm	0.664	100	160.78	0	0.010	66.40	100.00	0.439	100	163.23	0	0.010	43.90	100.00
	pointwise.yes_no	0.664	100	169.78	0	0.010	66.40	100.00	0.456	100	172.23	0	0.010	45.60	100.00
	listwise.generation	0.692	245	511.55	5.15	0.080	8.65	12.50	0.441	245	512.39	4.99	0.080	5.51	12.50
	listwise.likelihood	0.756	245	475.11	0	0.073	10.36	13.70	0.475	245	476.64	0	0.073	6.51	13.70
	pairwise.heapsort	0.761	241.32	630.77	10.00	0.099	7.69	10.10	0.402	182.36	634.95	10.00	0.075	5.36	13.33
	pairwise.bubblesort	0.714	880.18	630.84	10.00	0.361	1.98	2.77	0.439	528.48	634.06	10.00	0.218	2.01	4.59
	setwise.heapsort	0.768	129.62	449.59	5.00	0.037	20.76	27.03	0.462	120.84	452.73	5.00	0.034	13.59	29.41
	setwise.bubblesort	0.761	468.42	450.22	5.00	0.133	5.72	7.52	0.497	462.36	452.86	5.00	0.132	3.77	7.58
Flan-t5-xl	pointwise.qlm	0.679	100	160.78	0	0.036	18.86	27.78	0.427	100	163.23	0	0.037	11.54	27.03
	pointwise.yes_no	0.698	100	169.78	0	0.038	18.37	26.32	0.479	100	172.23	0	0.039	12.28	25.64
	listwise.generation	0.65	245	511.44	5.08	0.293	2.22	3.41	0.475	245	511.99	4.93	0.293	1.62	3.41
	listwise.likelihood	0.736	245	474.00	0	0.268	2.75	3.73	0.526	245	476.40	0	0.269	1.96	3.72
	pairwise.heapsort	0.778	252.28	629.13	10.00	0.377	2.06	2.65	0.55	241.30	635.46	10.00	0.364	1.51	2.75
	pairwise.bubblesort	0.763	914.34	628.55	10.00	1.365	0.56	0.73	0.553	771.66	634.93	10.00	1.164	0.48	0.86
	setwise.heapsort	0.757	135.8	449.53	5.00	0.142	5.33	7.04	0.52	129.38	452.73	5.00	0.136	3.82	7.35
	setwise.bubblesort	0.756	468.96	449.38	5.00	0.491	1.54	2.04	0.537	452.44	452.86	5.00	0.477	1.13	2.10
Flan-t5-xxl	pointwise.qlm	0.707	100	160.78	0	0.143	4.94	6.99	0.44	100	163.23	0	0.146	3.01	6.85
	pointwise.yes_no	0.691	100	169.78	0	0.152	4.55	6.58	0.515	100	172.23	0	0.154	3.34	6.49
	listwise.generation	0.664	245	511.44	5.19	1.147	0.58	0.87	0.495	245	511.98	5.01	1.148	0.43	0.87
	listwise.likelihood	0.749	245	474.73	0	1.052	0.71	0.95	0.518	245	476.43	0	1.056	0.49	0.95
	pairwise.heapsort	0.738	243.08	629.60	10.00	1.416	0.52	0.71	0.543	244	635.42	10.00	1.434	0.38	0.70
	pairwise.bubblesort	0.733	887.98	630.01	10.00	5.175	0.14	0.19	0.55	859.26	635.05	10.00	5.049	0.11	0.20
	setwise.heapsort	0.752	135.66	449.48	5.00	0.557	1.35	1.80	0.513	132.12	452.72	5.00	0.546	0.94	1.83
	setwise.bubblesort	0.768	470.88	449.89	5.00	1.935	0.40	0.52	0.534	456.23	452.86	5.00	1.887	0.28	0.53

Table 3: Results on TREC-COVID and Robust04. All the methods re-rank BM25 top 100 documents. #LLM represents average number of LLM call per query for reranking 100 documents. "In" and "Out" denote the average input tokens and output tokens per LLM call. #FLOPs is the estimated FLOPs per query for reranking 100 documents. Bold value is the best within each LLM and stared value is the best across different LLMs. "L3.1" represent Llama-3.1-8B-Instruct model. We report NDCG@10 for the NDCG metric.

A Appendix

A.1 BM25

$$\sum_{i=1}^{n} \text{IDF}(q_i) \frac{f(q_i, D)(k+1)}{f(q_i, D) + k\left(1 - b + b\frac{|D|}{\text{avgdl}}\right)} \quad (16)$$

BM25 (Robertson et al., 2009) computes a relevance score between a query and a document with Equation 16 where IDF denotes the inverse document frequency, $f(q_i, D)$ represents the frequency of the i^{th} query token in the document, $|D|$ is the length of the document, avgdl is the average document length in the corpus, and k and b are hyperparameters. Assuming that the term frequencies and the inverse document frequencies are precomputed and do not contribute to the runtime FLOP count, the upper bound on the FLOPs required for BM25 scoring can be estimated as:

$$C(\text{BM25}) = 11 \cdot L_Q \cdot N_D \quad (17)$$

where L_Q is the length of the query and N_D is the number of documents, which is 100 for reranking top-100 documents in our experiments. This represents an upper bound because it assumes that every query token appears in every document. In cases where a query token does not appear in a document, the corresponding numerator becomes zero, resulting in zero FLOPs for that term-document pair instead of 11.

A.2 BEIR results

We conduct extra experiments on two more datasets from BEIR: TREC-COVID and Robust04 to strengthen generalizability. The results are shown in Table 3. We observe the same efficiency–effectiveness patterns as in Table 2. Pointwise rerankers consistently deliver the strongest RPP and the highest QPP, whereas stronger models get the better performance at the cost of RPP and QPP. On TREC-COVID (Flan-T5-large), `setwise.heapsort` vs. `setwise.bubblesort` yields RPP 20.76 vs. 5.72 (QPP 27.03 vs. 7.52); on Robust04 the gap is 13.59 vs. 3.77 (29.41 vs. 7.58). Scaling the backbone collapses FLOPs-normalized efficiency: `pointwise.yes_no` QPP falls 100.00→26.32→6.58 on TREC-COVID (RPP 66.40→18.37→4.55) and 100.00→25.64→6.49 on Robust04 (RPP 45.60→12.28→3.34), while NDCG gains remain modest. For both Table 2 and Table 3, `setwise.heapsort` achieves top-tier performance while maintaining good efficiency.