
Supplementary Materials for AcuRank

Anonymous Author(s)

Affiliation

Address

email

1 Overview

This supplementary material provides additional details and extended results for our AcuRank framework. Specifically:

- **Section 2** describes the mathematical foundations of our Bayesian modeling approach, including the TrueSkill update equations and our efficient approximation of cumulative rank distributions.
- **Section 3** specifies our experimental setup, including hardware specifications, software configurations, datasets, licenses, and modifications to baseline implementations.
- **Section 4** presents comprehensive experimental results, including dataset-wise performance metrics omitted from the main paper due to space constraints. This includes extended results for all baselines (Tables 1-3), full numbers for the TrueSkill-Static variants shown in Figure 2, and additional experiments with **RankVicuna-7B** and **meta-llama/Llama-3.3-70B-Instruct** demonstrating the generalizability of our approach.
- **Section 5** provides a deeper analysis of AcuRank’s adaptive behavior, including query-level compute allocation patterns and a breakdown of performance on easy versus hard queries.
- **Section 6** discusses current limitations and potential future directions.
- **Section 7** lists the exact prompt templates used with RankZephyr, RankVicuna, and Llama-3.3-70B-Instruct for listwise reranking.

A minor typographical error discovered just before the supplementary deadline has been corrected; the change does not affect any reported trend. Concretely, the average number of reranker calls changes (1) for SW-1 with BM25 top-100 / RankGPT (gpt-4-1-mini) from 8.8 → 8.6, and (2) for AcuRank with BM25 top-100 / RankZephyr-7B from 19.2 → 19.7.

2 Supplement on Bayesian modeling and score updates in TrueSkill

2.1 TrueSkill as a Bayesian model

TrueSkill [1] models the latent relevance (originally, “skill”) of each document x_i as a Gaussian random variable, $x_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$, starting with a prior and updating it iteratively via Bayesian inference. Each listwise reranker output, denoted $g(\mathcal{D}'; \mathcal{M})$, is treated as noisy evidence about pairwise preferences among a subset \mathcal{D}' of documents, where \mathcal{M} is the reranking model. Using belief propagation on a factor graph, TrueSkill analytically updates the posterior distribution to

$$p(x_i \mid g(\mathcal{D}'; \mathcal{M})) = \mathcal{N}(\mu'_i, (\sigma'_i)^2).$$

We denote the current update round by t . At $t = 0$, $(\mu_i^{(0)}, \sigma_i^{(0)})$ represents the prior initialized from first-stage retrieval scores. After observing the reranking output, the posterior is updated to $(\mu_i^{(t+1)}, \sigma_i^{(t+1)})$.

32 2.2 Approximating the cumulative rank distribution

33 We approximate the probability that document D_i ranks within the top r positions as

$$P(r_i \leq r) \approx P(x_i > t(r)), \text{ where } \sum_j P(x_j > t(r)) = r.$$

34 The left-hand side is the probability that at most $r - 1$ other documents outrank D_i . The right-hand
 35 side approximates this count by evaluating whether x_i exceeds a threshold $t(r)$ calibrated such
 36 that the expected number of documents exceeding this threshold equals r . This approximation
 37 becomes accurate under two assumptions: (1) the latent scores x_i follow independent Gaussian
 38 distributions with similar variances, and (2) the number of candidates n is sufficiently large, so that
 39 the expected number of exceedances concentrates near its mean by the law of large numbers. This
 40 provides an *efficient closed-form* alternative to the expensive $O(n^2)$ dynamic programming approach
 41 for computing the full ranking distribution over permutations. Instead of enumerating all possible
 42 rankings, we approximate $P(r_i \leq r)$ using a single Gaussian tail probability, $1 - \Phi(\cdot)$.

43 2.3 TrueSkill score update equations

44 The reranking over each batch \mathcal{B} is treated as a *multi-player game*. The observed ranking outcome is
 45 given by $g(\mathcal{B}; \mathcal{M})$. For every adjacent document pair in this order, a win-loss factor is defined to
 46 model their pairwise preference. Applying message passing on the factor graph yields the following
 47 canonical update equations for each document D_i :

$$\mu_i^{(t+1)} = \mu_i^{(t)} + \frac{\sigma_i^2}{c} \lambda, \quad (\sigma_i^{(t+1)})^2 = \sigma_i^2 \left(1 - \frac{\sigma_i^2}{c} v \right),$$

48 where $c = \sqrt{\sum_{D_j \in \mathcal{B}} (\sigma_j^2 + \beta^2)}$. The values λ and v are closed-form functions derived from the win
 49 probability between adjacent items (see Herbrich et al. [1] for a full derivation). Intuitively, when a
 50 document ranks higher than expected, its mean μ_i increases. Unexpected outcomes also reduce σ_i ,
 51 reflecting greater confidence in the document’s estimated relevance.

52 **Side note on implementation.** We employ the Python trueskill library¹. The relevance of
 53 each candidate passage is represented as a trueskill.Rating class. After the listwise reranker
 54 outputs an ordering, we update all ratings by calling trueskill.rate function with the argument
 55 rating_groups= list of ratings, and ranks=[0, 1, ..., n], where n is the number of passages.
 56 The function returns a list of posterior ratings containing (μ, σ) , which serve as the priors for the next
 57 iteration.²

58 2.4 Proportional uncertainty initialization based on retrieval scores

59 We initialize each document’s relevance score as a Gaussian distribution parameterized by its first-
 60 stage retrieval score. Specifically, for each document D_i with retrieval score s_i , we set the mean
 61 and standard deviation as $\mu_i = s_i$ and $\sigma_i = \mu_i/3$. This initialization preserves the *relative scale* of
 62 the retrieval scores while explicitly modeling their uncertainty. A higher retrieval score indicates
 63 greater expected relevance ($\mu_i \uparrow$), but it may also be more prone to overestimation. To reflect this,
 64 we assign a proportionally larger prior uncertainty ($\sigma_i \uparrow$) to such documents. The choice of the 1/3
 65 ratio matches the default setting in the original TrueSkill model. Statistically, this means that the
 66 interval $[\mu_i - \mu_i, \mu_i + \mu_i] = [0, 2\mu_i]$ corresponds to $[\mu_i \pm 3\sigma_i]$, since $3\sigma_i = \mu_i$. This range contains
 67 approximately 99.7% of the Gaussian probability mass. This setup ensures that the prior distribution
 68 is centered on the retrieval score, but still allows for flexible updates. It incorporates the retrieval
 69 signal while leaving room for reranking results to meaningfully revise the belief.

70 2.5 Score refinement (Algorithm 1, Line 5)

71 After each reranker call, we perform the following update procedure:

- 72 1. Convert the returned ranking over the current batch B into a set of pairwise win-loss observations.

¹<https://github.com/sublee/trueskill>

²<https://trueskill.org/#trueskill.TrueSkill.rate>

- 73 2. Run one round of TrueSkill message passing to update each document’s parameters (μ_i, σ_i) in B
74 based on these observations.
- 75 3. Recompute the ranking-uncertainty scores $s_i = P(x_i > t(k))$ to guide selection in the next
76 iteration.
- 77 This iterative posterior refinement progressively sharpens the relevance estimates. As a result, the
78 uncertainty values σ_i naturally decrease over time. Documents whose ranks stabilize early tend to
79 exit the *uncertain* set, allowing us to avoid unnecessary reranker calls and reduce overall computation.

80 3 Implementation Details

81 3.1 Hardware specifications and software configurations

82 **Compute nodes:** Experiments were conducted on either (i) an ASUS ESC8000-E11 server equipped
83 with dual 4th-Gen Intel Xeon processors, 64 CPU threads, 1.1 TB RAM, and eight NVIDIA A6000
84 GPUs (48 GB each), or (ii) a workstation with eight RTX 3090 GPUs (24 GB each).

85 **Inference stack:** All LLM inference was performed using the transformers library, without
86 acceleration backends such as vLLM or FastChat. Greedy decoding was used throughout, with a
87 fixed random seed to ensure reproducibility.

88 **Reproducibility:** We release a complete end-to-end reproduction script, packaged as a .zip archive,
89 which replicates reported experiments.

90 **Token limits:** Following the RankLLM codebase³, all inputs were truncated to a maximum of 4,096
91 tokens.

92 **Default hyperparameters:** Unless otherwise specified, we adopt the default hyperparameter settings
93 provided by the RankLLM implementation.

94 3.2 Datasets and licenses

95 **Abbreviations.** For brevity, we use the following shorthands throughout our tables: DL19–DL23
96 (TREC-DL 2019–2023), DL-H (DL-Hard), COVID (TREC-COVID), NFC (NFCorpus), Signal
97 (Signal-1M), News (TREC-News), R04 (Robust04), Touche (Webis-Touche2020), DBP (DBPedia),
98 and Sci (SciFact).

99 **TREC-DL (2019–2023 tracks).** The Topics and Qrels are in the public domain as U.S. Government
100 work.⁴ The underlying corpus (MS MARCO + ORCAS) is released by Microsoft for **non-commercial**
101 **research purposes only**, with no IP rights to the documents and “as-is” usage at the user’s own risk.⁵

102 **DL-HARD [2].** We use the annotation files and BM25 retrieval result files hosted on GitHub
103 (<https://github.com/grill-lab/DL-Hard>). The repository does not specify a formal license.
104 Following author guidance and consistent with MS MARCO terms (see footnote 5), we use the data
105 strictly for non-commercial academic research.

106 **BEIR [3].** The framework code for BEIR is under the Apache 2.0 license.⁶ Each constituent dataset
107 retains the license of its original source (e.g., public-domain for TREC-COVID). For our experiments,
108 we use pre-computed BM25 and SPLADE top-1000 retrieval runs (included in our anonymized
109 supplementary code), which are released under the BigScience OpenRAIL-M license.

110 **TrueSkill rating algorithm.** We rely on the open-source Python package trueskill (v0.4.5)⁷,
111 distributed under the permissive **BSD 3-Clause License**. The *TrueSkill*TM trademark and the original
112 Bayesian rating system remain the property of Microsoft, who permits the name and algorithm to be
113 used only in Xbox Live titles or *non-commercial* projects.⁸ Our usage complies with these terms, as
114 it is strictly limited to non-profit academic research.

³https://github.com/castorini/rank_llm

⁴<https://trec.nist.gov>

⁵<https://microsoft.github.io/msmarco/>

⁶<https://github.com/beir-cellar/beir>

⁷<https://github.com/sublee/trueskill>

⁸<https://trueskill.org/>

115 3.3 Evaluation protocol

116 All reported metric values are **rounded to one decimal place** for consistency. For each method, we
117 first compute the mean number of reranker calls *within each dataset*. Specifically, we calculate the
118 average across all queries belonging to a given dataset, resulting in one mean value per dataset. We
119 then compute the overall value reported in our tables by taking a simple unweighted average of these
120 14 per-dataset means. This ensures that each dataset contributes equally, regardless of the number of
121 queries it contains. Note that this is a dataset-level **macro average**, not a micro average computed
122 over all individual queries pooled together.

123 3.4 Sliding windows baseline

124 The sliding windows baselines rerank the top-100 candidates using fixed-size windows of size 20 and
125 a stride of 10. This configuration results in *nine* overlapping windows per query. When the list length
126 is not divisible by 20, we form a final, smaller window containing the remaining documents. This
127 ensures full coverage of all retrieved candidates.

128 3.5 TourRank baseline

129 We adapted the public TourRank implementation⁹ to handle candidate sets with fewer than 100
130 passages. The original code assumes $|\mathcal{D}| = 100$ and executes five tournament stages. However,
131 for some queries in certain datasets (e.g., NFCorpus), the query text is extremely short, so BM25
132 retrieves fewer than 100 candidate documents ($|\mathcal{D}| < 100$). To handle such cases, we added the
133 simple stage-skip rule below and a one-line boundary check to the original grouping function, in
134 order to avoid redundant inference and out-of-range errors. Our modifications are twofold:

135 **Stage skip rule.** We skip Stages 1 through 4 when $|\mathcal{D}|$ is less than or equal to 50, 20, 10, and 5,
136 respectively. When $|\mathcal{D}| \leq 2$, we return the BM25 order directly.

137 **Boundary check (grouping).** We retain the original index modulo grouping logic from the public
138 implementation and insert a simple boundary check (if `idx < len(D)`) to prevent `IndexError`.

139 4 Extended experimental results

140 4.1 Average number of reranker calls per dataset

141 We report the number of reranker calls per dataset with NDCG@10 scores, which correspond to
142 the Tables and Figures reported in the main paper. **Table 1** reports the NDCG@10 scores for each
143 dataset, along with the average number of reranker calls. These results complement **Table 1** in
144 the main paper by providing dataset-level breakdowns. Additionally, **Table 3** reports the complete
145 per-dataset results for TrueSkill-Static with different configurations shown in **Figure 2**. **Table 2**
146 summarizes dataset-wise results for the SPLADE++ED first-stage retrieval results and results with
147 RankGPT-4.1-mini as a reranker, including the average number of passes, corresponding to **Table 2**
148 in the main paper. Finally, Table 4 reports the *per-dataset* average number of reranker calls and the
149 resulting NDCG@10 for each design-choice variant, corresponding with **Table 3** in our main paper.

150 A key observation from these extended results is that, unlike static approaches such as sliding windows
151 or TourRank, our method exhibits varying numbers of calls across different datasets. This adaptive
152 behavior reflects our uncertainty-aware framework’s ability to allocate computational resources based
153 on the inherent difficulty and characteristics of each dataset.

154 4.2 Generalization across rerankers

155 To evaluate the generalizability of AcuRank across different reranking models and scales, we conduct
156 comprehensive experiments using both RankVicuna-7B [4] and Llama-3.3-70B-Instruct [5] models.

157 **Evaluation with RankVicuna-7B.** We first test AcuRank using **RankVicuna-7B** (huggingface
158 identifier of `castorini/rank_vicuna_7b_v1`) on BM25 top-100 retrieval results. As shown in

⁹<https://github.com/chenyiqun/TourRank>

Table 1: Full results corresponding to Table 1 of the main paper, with annotated avg_calls per dataset. Only SW-1 and TourRank-1 are shown because these variants used fixed computation per query; the counts for SW-N or TourRank-N for $N > 1$ can be obtained by multiplying the reported numbers by N . Slightly lower values on NFCorpus, DBPedia, and SciFact reflect queries that retrieve fewer than 100 documents with BM25, which proportionally reduces the average number of calls.

Method	TREC-DL						BEIR								Avg. # Calls	
	DL19	DL20	DL21	DL22	DL23	DL-H	COVID	NFC	Signal	News	R04	Touche	DBP	Scif		
First-stage retrieval: <i>BM25 top 100</i> Reranker: <i>RankZephyr-7B</i>																
BM25 (no rerank)	50.6	48.0	44.6	26.9	26.2	30.4	59.5	32.2	33.0	39.5	40.7	44.2	31.8	67.9	41.1	0.0
↪ avg_calls	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	0.0
SW-1	74.0	70.2	69.5	51.5	44.5	38.6	84.1	36.8	32.0	52.3	54.0	32.4	44.5	75.5	54.3	8.8
↪ avg_calls	9.0	9.0	9.0	9.0	9.0	9.0	9.0	6.2	9.0	9.0	9.0	9.0	9.0	9.0	8.8	8.8
TourRank-1	74.2	68.2	69.6	51.1	45.2	38.1	81.8	36.5	30.7	51.9	54.5	31.2	43.2	71.3	53.4	12.7
↪ avg_calls	13.0	13.0	13.0	13.0	13.0	13.0	13.0	9.3	13.0	13.0	13.0	13.0	13.0	13.0	12.7	12.7
AcuRank-9	73.3	71.4	70.1	50.1	45.3	39.5	83.2	36.8	30.8	53.0	56.0	37.1	44.7	73.3	54.6	8.8
↪ avg_calls	9.0	8.9	9.0	9.0	9.0	9.0	9.0	6.7	8.9	9.0	8.9	9.0	8.9	8.9	8.8	8.8
AcuRank	74.2	71.8	70.3	52.0	47.0	39.4	85.3	37.2	31.8	53.9	56.6	36.5	46.0	75.3	55.5	19.7
↪ avg_calls	18.2	16.3	15.5	18.7	17.4	16.3	21.8	23.3	30.4	18.5	17.8	19.4	21.5	20.1	19.7	19.7
AcuRank-H	74.6	70.8	70.5	52.2	47.3	40.4	85.8	37.4	32.1	53.7	56.8	37.5	46.0	75.4	55.7	41.7
↪ avg_calls	36.9	35.3	29.6	40.0	36.6	32.9	43.9	51.5	58.8	38.9	37.6	44.2	48.4	49.0	41.7	41.7
AcuRank-HH	74.7	71.8	70.6	51.9	47.0	40.0	86.1	37.5	31.3	54.4	57.8	36.1	46.0	75.4	55.8	57.2
↪ avg_calls	53.3	46.6	43.4	60.6	60.2	48.9	60.2	59.8	75.2	52.2	50.6	57.0	68.6	63.9	57.2	57.2
First-stage retrieval: <i>BM25 top 1000</i> Reranker: <i>RankZephyr-7B</i>																
BM25	50.6	48.0	44.6	26.9	26.2	30.4	59.5	32.2	33.0	39.5	40.7	44.2	31.8	67.9	41.1	0.0
↪ avg_calls	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	0.0
SW-1	75.1	78.8	71.5	57.5	49.5	40.9	80.7	38.0	28.9	51.0	48.0	30.9	48.0	76.4	56.2	94.6
↪ avg_calls	99.0	99.0	99.0	99.0	99.0	99.0	99.0	40.1	99.0	99.0	99.0	99.0	98.4	97.0	94.6	94.6
AcuRank	76.7	75.3	73.1	59.3	53.5	41.0	85.0	37.0	30.7	56.5	63.2	36.2	48.8	76.0	58.0	68.4
↪ avg_calls	67.8	65.6	67.6	74.4	72.0	68.8	72.8	40.6	82.2	68.0	68.7	70.4	74.2	64.7	68.4	68.4

Table 2: Results in NDCG@10 on BEIR with two alternative first-stage / reranker configurations, corresponding to Table 2 of our main paper. For every method that performs reranking, the dataset-wise average number of passes is given in the immediately following row (prefixed by ↪).

Method	COVID	NFC	Signal	News	R04	Touche	DBP	Scif	Avg. # Calls
First-stage retrieval: <i>SPLADE++ED top-100</i> Reranker: <i>RankZephyr-7B</i>									
SPLADE++ED	71.1	34.5	29.6	39.4	45.8	24.4	44.1	69.9	44.8 0.0
SW-1	85.2	37.5	29.7	50.1	62.0	28.9	49.3	75.7	52.3 9.0
↪ avg_calls	9.0	9.0	9.0	9.0	9.0	9.0	9.0	9.0	9.0 9.0
AcuRank	85.5	37.7	27.2	51.9	63.1	30.2	50.0	75.6	52.7 8.9
↪ avg_calls	9.3	7.6	10.0	8.9	8.8	9.1	9.1	8.3	8.9 8.9
First-stage retrieval: <i>BM25 top-100</i> Reranker: <i>RankGPT (gpt-4.1-mini)</i>									
BM25	59.5	32.2	33.0	39.5	40.7	44.2	31.8	67.9	43.6 0.0
SW-1	84.5	37.9	33.3	49.5	61.5	35.7	46.3	78.7	53.4 8.6
↪ avg_calls	9.0	6.2	9.0	9.0	9.0	9.0	9.0	9.0	8.6 8.6
AcuRank	86.4	38.3	34.8	52.1	63.0	31.9	46.2	77.2	53.7 20.8
↪ avg_calls	20.9	25.0	22.0	18.3	18.8	19.4	20.0	21.7	20.8 20.8

Table 5, while RankVicuna achieves lower absolute NDCG@10 scores than RankZephyr, AcuRank maintains its effectiveness pattern. Specifically, AcuRank-9 achieves +0.7 improvement over SW-1 that uses same number of calls (8.8). Meanwhile, AcuRank achieves +1.8 NDCG@10 improvement over SW-1, and +1.2 improvement over SW-3 despite using fewer calls (19.5 vs 26.4), and +1.5 improvement over TourRank-2 using similar number of calls (25.5 vs 26.4). In summary, AcuRank consistently outperforms both sliding windows and TourRank baselines while requiring fewer reranker calls.

Evaluation with Llama-3.3 70B as a zero-shot reranker. To test whether AcuRank continues to yield benefits for larger, purely zero-shot rerankers, we replaced our 7B parameter RankZephyr

Table 3: Dataset-wise results in NDCG@10 of *TrueSkill-Static* with different configurations c (BM25 top-100 + RankZephyr-7B), corresponding to Figure 2 of the main paper. TS- c is represented as TS- N , where $N = \sum_i c_i$, for short. The last two columns are the macro average over 14 datasets and the macro mean number of reranker calls.

Method	TREC-DL						BEIR								Avg. # Calls	
	DL19	DL20	DL21	DL22	DL23	DL-H	COVID	NFC	Signal	News	R04	Touche	DBP	Scif		
TS-10 [5-2-2-1]	74.4	71.2	69.9	50.7	45.7	39.8	82.6	37.0	31.9	52.5	55.8	37.2	45.4	74.6	54.9	9.8
TS-14 [5-3-3-3]	75.0	71.8	70.2	51.1	45.6	40.5	83.6	37.0	31.3	53.8	56.2	36.7	45.7	74.8	55.2	13.8
TS-25 [5-5-5-5]	74.8	71.8	70.3	51.3	45.7	40.4	83.5	37.1	32.0	53.1	56.7	36.8	45.6	75.1	55.3	24.5
TS-25 [5-4-4-4-4]	75.0	71.6	70.5	51.3	46.0	39.9	84.2	37.1	32.0	53.2	56.5	36.0	45.6	75.1	55.3	24.5
TS-35 [5-3-3-3-3-3-3-3]	75.5	72.0	70.7	51.5	45.8	40.1	84.4	37.0	31.1	53.9	56.1	35.9	45.3	75.8	55.4	34.4

Table 4: Dataset-wise NDCG@10 for the **AcuRank** design-choice ablation, corresponding to Table 3 of our main paper. For every variant the second line (prefixed by \hookrightarrow) reports the dataset-specific average number of passes.

Variant	TREC-DL						BEIR								Avg. # Calls	
	DL19	DL20	DL21	DL22	DL23	DL-H	COVID	NFC	Signal	News	R04	Touche	DBP	Scif		
AcuRank (default)	74.2	71.8	70.3	52.0	47.0	39.4	85.3	37.2	31.8	53.9	56.6	36.5	46.0	75.3	55.5	19.7
\hookrightarrow avg_calls	18.2	16.3	15.5	18.7	17.4	16.3	21.8	23.3	30.4	18.5	17.8	19.4	21.5	20.1	19.7	19.7
No 1st-stage init	74.1	70.8	70.9	52.0	47.3	38.8	84.3	36.8	30.5	52.1	55.3	33.5	44.9	75.8	54.8	13.4
\hookrightarrow avg_calls	12.5	12.7	12.3	12.5	13.3	12.9	12.9	9.6	21.4	13.8	13.3	13.0	14.1	13.4	13.4	13.4
Random chunking	74.2	71.2	70.8	51.8	47.2	37.6	87.0	37.3	31.6	53.9	56.8	34.5	46.0	74.8	55.3	22.6
\hookrightarrow avg_calls	19.9	17.4	17.4	19.6	20.7	18.1	25.7	30.4	34.4	19.6	19.6	23.4	23.7	26.8	22.6	22.6
Top- k stability stop	75.2	69.5	70.0	51.4	46.5	40.1	84.7	37.0	31.7	52.5	56.1	37.6	45.4	74.5	55.2	22.7
\hookrightarrow avg_calls	22.3	20.3	18.0	22.8	20.7	20.1	23.7	21.6	29.3	22.1	22.2	24.4	24.5	25.3	22.7	22.7

model with **Llama-3.3 70B Instruct**¹⁰ [5] and reran the pipeline. Resource constraints limited us to three representative benchmarks: DL23, TREC-COVID, and TREC-NEWS. Results are reported in Table 6. Even for this stronger backbone AcuRank extracts an additional **+1.6** average NDCG@10 over the single-pass baseline, while keeping the number of calls bounded. This confirms that our framework continues to provide a favorable accuracy–efficiency trade-off as model capacity scales.

5 Extended analysis on adaptive computation allocation

Following Section 5.3 of the main paper (“Adaptive allocation behavior”) and Figure 3, we provide a more fine-grained query-level analysis.

5.1 Correlations with WIG

This is an extended analysis of Figure 3 in the main paper. On designing the experiment, we pool all datasets containing fewer than 100 queries to obtain reliable correlations. This includes every benchmark except DBPedia-Entity, SciFact, NFCorpus, and Robust04. The resulting subset contains 612 queries drawn from 14 datasets. For each query, we compute the *Weighted Information Gain* (WIG) [6] over the BM25 top-100 retrieval scores. We then measure two Spearman correlations: (i) between WIG and the number of calls made by AcuRank to satisfy its stopping criterion, and (ii) between WIG and the final NDCG achieved by AcuRank. As shown in Table 7, both correlations are negative.

- **Calls versus WIG.** Lower WIG, which reflects more ambiguous top-100 lists, leads to more reranker calls. This confirms that AcuRank allocates additional computation when the candidate set is uncertain.
- **Calls versus NDCG.** Queries that result in lower final NDCG tend to receive more reranker calls. This indicates that AcuRank naturally focuses effort on harder queries.

¹⁰meta-llama/Llama-3.3-70B-Instruct

Table 5: Results in NDCG@10 using BM25 top-100 first-stage retrieval with **RankVicuna-7B** reranker. For every variant the second line (prefixed by \hookrightarrow) reports the dataset-specific average number of passes. Lower call counts on some datasets (NFCorpus, DBPedia, SciFact) reflect queries retrieving fewer than 100 documents with BM25.

Method	TREC-DL						BEIR										Avg. # Calls
	DL19	DL20	DL21	DL22	DL23	DL-H	COVID	NFC	Signal	News	R04	Touche	DBP	Scif			
First-stage retrieval: <i>BM25 top-100</i> Reranker: <i>RankVicuna-7B</i>																	
BM25	50.6	48.0	44.6	26.9	26.2	30.4	59.5	32.2	33.0	39.5	40.7	44.2	31.8	67.9	41.1	0.0	
↪ avg_calls	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	0.0	
SW-1	66.4	62.5	60.4	40.9	38.2	35.1	77.4	33.0	33.7	45.2	46.9	33.7	43.6	69.5	49.0	8.8	
↪ avg_calls	9.0	9.0	9.0	9.0	9.0	9.0	9.0	6.2	9.0	9.0	9.0	9.0	9.0	9.0	8.8	8.8	
SW-2	67.4	63.2	60.5	40.8	38.1	36.2	78.6	33.3	33.5	45.4	47.1	33.7	43.5	70.6	49.4	17.6	
↪ avg_calls	18.0	18.0	18.0	18.0	18.0	18.0	18.0	12.4	18.0	18.0	18.0	18.0	18.0	18.0	17.6	17.6	
SW-3	67.5	63.2	60.6	41.0	38.2	35.9	78.7	33.5	33.9	45.6	47.2	34.8	43.9	69.9	49.6	26.4	
↪ avg_calls	27.0	27.0	27.0	27.0	27.0	27.0	27.0	18.6	27.0	27.0	27.0	27.0	26.9	27.0	26.4	26.4	
SW-4	67.5	63.2	60.6	41.0	37.3	36.2	78.7	33.4	33.8	45.9	47.4	34.4	43.8	70.3	49.5	35.2	
↪ avg_calls	36.0	36.0	36.0	36.0	36.0	36.0	36.0	24.9	36.0	36.0	36.0	36.0	35.9	36.0	35.2	35.2	
TourRank-1	65.7	63.4	61.2	42.5	35.4	35.5	76.7	31.6	31.6	43.9	46.7	29.2	39.8	62.2	47.5	12.7	
↪ avg_calls	13.0	13.0	13.0	13.0	13.0	13.0	13.0	9.3	13.0	13.0	13.0	13.0	13.0	13.0	12.7	12.7	
TourRank-2	67.1	64.5	61.1	40.8	36.8	34.7	78.4	31.2	34.3	47.6	48.5	33.5	44.0	67.4	49.3	25.5	
↪ avg_calls	26.0	26.0	26.0	26.0	26.0	26.0	26.0	18.7	26.0	26.0	26.0	26.0	25.94	26.0	25.5	25.5	
TourRank-5	67.1	64.5	62.2	42.7	38.1	35.1	78.5	32.5	32.8	46.3	48.9	28.3	42.8	66.5	49.0	63.7	
↪ avg_calls	65.0	65.0	65.0	65.0	65.0	65.0	65.0	46.7	65.0	65.0	65.0	65.0	64.9	65.0	63.7	63.7	
AcuRank-9	66.4	63.4	60.9	41.1	37.7	35.8	77.9	34.4	33.1	49.2	47.5	35.2	42.6	70.8	49.7	8.8	
↪ avg_calls	9.0	9.0	9.0	8.9	8.9	9.0	8.9	6.7	9.0	9.0	8.9	8.9	8.9	8.9	8.8	8.8	
AcuRank	66.4	65.5	61.4	43.0	38.5	36.9	80.7	35.3	33.7	49.7	49.6	35.4	44.4	71.3	50.8	19.5	
↪ avg_calls	18.2	16.4	15.7	18.3	17.7	17.5	22.4	25.1	18.5	18.1	18.7	23.3	20.1	22.8	19.5	19.5	
AcuRank-H	67.4	65.8	61.7	43.3	39.0	36.5	81.1	35.1	33.5	49.9	50.1	34.2	44.6	71.8	51.0	25.1	
↪ avg_calls	23.7	21.7	21.8	23.9	23.6	23.4	28.7	31.0	23.4	22.8	23.5	30.2	25.0	28.7	25.1	25.1	

Table 6: Results in NDCG@10 for AcuRank with the zero-shot **Llama-3.3-70B-Instruct** model. Averages are computed across the three datasets shown. Without any task-specific fine-tuning, AcuRank improves quality while keeping the number of reranker calls within a reasonable range.

Method	DL23	TREC-COVID	News	Avg.	# Calls
BM25 (no rerank)	26.2	59.5	39.5	41.7	0
SW-1	47.0	84.6	50.1	60.6	9.0
AcuRank	47.7	86.1	53.0	62.2	19.4

5.2 Comparison with fixed sliding windows

We further compare AcuRank (with an average of 19.7 calls across 18 datasets) with a Sliding Windows baseline using a similar compute budget (SW-2, 18 calls). For three challenging datasets (Touche, TREC-COVID, and DL-Hard), we split the queries into *Easy* ($\text{NDCG} \geq \mu$) and *Hard* ($\text{NDCG} < \mu$), where μ is the dataset-level mean NDCG produced by SW-2. Table 8 shows a consistent pattern:

- **Avg. # Reranker Calls: Hard > Easy.** AcuRank spends more calls on the *Hard* bucket (e.g. +6.2 on TREC-COVID) and sometimes even saves calls on the *Easy* bucket (e.g. DL-HARD).
- **NDCG Gains: Hard > Easy.** The additional computation translates into larger NDCG improvements on Hard queries (e.g. +7.0 on TOUCHE, +1.7 on TREC-COVID), while improvements on Easy queries are smaller than those for the Hard bucket queries.

These observations corroborate the claim that AcuRank *adaptively* directs compute to the most uncertain and therefore most difficult queries, yielding a superior accuracy and efficiency trade-off compared with fixed-schedule baselines.

Table 7: **Corpus-wide query-level correlations** ($n = 612$). Negative ρ for “# Calls” indicates that AcuRank issues *more* reranker calls on queries predicted as harder (lower WIG) or where its own NDCG is lower.

Variable pair	Spearman ρ	p -value
# Calls vs. WIG	−0.24	9.3×10^{-10}
# Calls vs. NDCG (Ours)	−0.36	2.7×10^{-20}

Table 8: Per-query analysis comparing the Sliding-Window baseline (SW-2) with AcuRank. Queries are split into *Easy* and *Hard* halves using the baseline’s own mean NDCG. AcuRank allocates more computation to **hard** queries, which often results in larger NDCG gains, while occasionally reducing the number of calls for easy queries.

Dataset	Subset	$ Q $	Ranking Quality (NDCG)			Avg. # Reranker Calls		
			SW-2	AcuRank	Δ	SW-2	AcuRank	Δ
Touche	Easy ($\geq \mu$)	23	50.5	52.7	+2.2	18.0	18.1	+0.1
	Hard ($< \mu$)	26	15.2	22.2	+7.0	18.0	20.5	+2.5
	All ($\mu = 31.8$)	49	31.8	36.5	+4.7	18.0	19.4	+1.4
TREC-COVID	Easy ($\geq \mu$)	30	96.1	96.7	+0.6	18.0	20.2	+2.2
	Hard ($< \mu$)	20	66.7	68.4	+1.7	18.0	24.2	+6.2
	All ($\mu = 84.4$)	50	84.4	85.4	+1.0	18.0	21.8	+3.8
DL-Hard	Easy ($\geq \mu$)	25	65.7	64.6	−1.1	18.0	15.8	−2.2
	Hard ($< \mu$)	25	11.2	14.3	+3.1	18.0	16.8	−1.2
	All ($\mu = 38.4$)	50	38.4	39.4	+1.0	18.0	16.3	−1.7

204 6 Limitations and Future Work

205 **Latency optimization.** Although AcuRank reduces the total number of reranker calls, further latency
206 improvements may be achieved through parallelization. Unlike sequential sliding windows, our
207 use of disjoint candidate groups enables batched inference across groups within the same iteration.
208 In addition, cross-query computation sharing could significantly reduce latency when processing
209 multiple queries concurrently.

210 **Hyperparameter adaptation.** Our fixed global hyperparameters (μ , ϵ , and stopping criteria) perform
211 well overall, but may be suboptimal for certain domains. Query-aware or domain-specific tuning
212 could improve effectiveness. Automatic adaptation methods represent a promising direction for future
213 work.

214 **Potential method enhancements.** While AcuRank demonstrates strong performance, several
215 promising directions remain for future research. Our current approach uses thresholded cumu-
216 lative rank probabilities to guide reranking decisions. However, the full rank probability distri-
217 bution offers a richer source of uncertainty information. For example, *rank entropy*, defined as
218 $\mathbb{H}(r_i) = -\sum_{r=1}^n \mathbb{P}(r_i = r) \log \mathbb{P}(r_i = r)$, quantifies the dispersion in a document’s predicted
219 rank and could support more expressive uncertainty-based filtering strategies.¹¹ In addition, our
220 current grouping strategy for uncertain documents is based on simple heuristics. Clustering-based
221 or similarity-aware grouping methods may improve reranking efficiency by better capturing inter-
222 document structure. Finally, modern LLMs may expose token-level or generation-level confidence
223 scores. These signals could be integrated to refine document-level uncertainty estimates or modulate
224 the strength of score updates during reranking.

¹¹The pointwise and cumulative rank probabilities are mutually derivable. The cumulative probability is the sum of pointwise probabilities, $\mathbb{P}(r_i \leq r) = \sum_{j=1}^r \mathbb{P}(r_i = j)$, while the pointwise probability is given by the difference between adjacent cumulative terms, $\mathbb{P}(r_i = r) = \mathbb{P}(r_i \leq r) - \mathbb{P}(r_i \leq r - 1)$.

7 Example input/output prompt format for listwise reranking

We provide concrete prompt examples for RankZephyr [7], RankVicuna [4], and zero-shot Llama-3.3-70B-Instruct reranking models [5].

7.1 RankZephyr [7] prompt example

```
<|system|>
You are RankLLM, an intelligent assistant that can rank passages based on their
relevancy to the query.
</s>
<|user|>
I will provide you with 20 passages, each indicated by a numerical identifier []. Rank
the passages based on their relevance to the search query: PGE 2 promotes
intestinal tumor growth by altering the expression of tumor suppressing and DNA
repair genes..

[1] Title: Prostaglandin E2 promotes intestinal tumor growth via DNA methylation
Content: Although aberrant DNA methylation is considered to be one of the key ways by
which tumor-suppressor and DNA-repair genes are silenced during tumor initiation
and progression, the mechanisms underlying DNA methylation alterations in cancer
remain unclear. Here we show that prostaglandin E2 (PGE2) silences certain tumor-
suppressor and DNA-repair genes through DNA methylation to promote tumor growth.
These findings uncover a previously unrecognized role for PGE2 in the promotion of
tumor progression.
[2] Title: Mole ... Patients were treated within clinical trials testing ...
...
[20] Title: DNA Damage and Repair Modify DNA Methylation and Chromatin Domain of the
Targeted Locus
Content: We characterize the changes in chromatin structure, DNA methylation and
transcription during and after homologous DNA repair (HR). We find that HR modifies
the DNA methylation pattern of the repaired segment. HR also alters local histone
H3 methylation as well as chromatin structure by inducing DNA-chromatin loops
connecting the 5' and 3' ends of the repaired gene.

Search Query: PGE 2 promotes intestinal tumor growth by altering the expression of tumor
suppressing and DNA repair genes..
Rank the 20 passages above based on their relevance to the search query. All the
passages should be included and listed using identifiers, in descending order of
relevance. The output format should be [] > [], e.g., [2] > [1]. Only respond with
the ranking results; do not say any word or explain.
</s>
<|assistant|>
-----
Example Output:
[1] > [3] > [6] > [11] > [5] > [8] > [20] > [2] > [18] > [15] > [16] > [19] > [10] >
[12] > [13] > [4] > [17] > [7] > [9] > [14]
```

```

<s>[INST] <<SYS>>
You are RankLLM, an intelligent assistant that can rank passages based on their
relevancy to the query.
<</SYS>>

I will provide you with 20 passages, each indicated by a numerical identifier []. Rank
the passages based on their relevance to the search query: what does prenatal care
include.

[1] Pregnancy and prenatal care go hand in hand. During the first trimester, prenatal
care includes blood tests, a physical exam, conversations about lifestyle and more.
Prenatal care is an important part of a healthy pregnancy. Whether you choose a
family physician, obstetrician, midwife or group prenatal care, here's what to
expect during the first few prenatal appointments.
[2] Pregnancy and prenatal care go hand in hand. During the first trimester, prenatal
care includes blood tests, a physical exam, conversations about lifestyle and more.
Prenatal care is an important part of a healthy pregnancy. Whether you choose a
family physician, obstetrician, midwife or group prenatal ...
[3] Prenatal appointments typically ...
[4] Routine prenatal screening ...
[5] Standard tests include ...
...
[19] What is Prenatal Care: Prenatal care is the health care that both the woman and the
baby receive before giving birth. This is more than just a few doctor's visits and
an ultrasound or two.
[20] Medicaid coverage for prenatal care provides another opportunity to illustrate the
four pathways' differences and how each state determines what services to include.
(1) Right away, we know that the emergency pathway will not pay for prenatal care
because preventive services are non-urgent by definition.

Search Query: what does prenatal care include.
Rank the 20 passages above based on their relevance to the search query. All the
passages should be included and listed using identifiers, in descending order of
relevance. The output format should be [] > [], e.g., [2] > [1]. Only respond with
the ranking results, do not say any word or explain. [/INST]
-----
Example Output:
[1] > [2] > [3] > [5] > [6] > [19] > [8] > [15] > [16] > [14] > [17] > [4] > [12] > [13]
> [9] > [10] > [18] > [7] > [11] > [20]

```

273 7.3 Llama-3.3-70B-Instruct (zero-shot) prompt example

```

273 <|begin_of_text|><|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023
Today Date: 26 Jul 2024

You are RankLLM, an intelligent assistant that can rank passages based on their
    relevancy to the query.<|eot_id|><|start_header_id|>user<|end_header_id|>

I will provide you with 20 passages, each indicated by a numerical identifier [].
Rank the passages based on their relevance to the search query:
How much impact do masks have on preventing the spread of the COVID-19?.

[1] Title: Preparedness and response to COVID-19 in Saudi Arabia: Building on MERS
    experience
    Content: Nearly four months have passed since the emergence of SARS-CoV-2, ...
[2] Title: The effect of community masking on transmission rates ...
[3] ...
...
274 [19] Title: Comparative hydrophobicity of fabrics used for improvised masks ...
[20] Title: An exploration of how fake news is taking over social media and putting
    public health at risk.
    Content: Recent statistics show that almost 1/4 million people have died ...

Search Query: How much impact do masks have on preventing the spread of the COVID-19?.

Rank the 20 passages above based on their relevance to the search query.
All the passages should be included and listed using identifiers, in descending order of
    relevance.
The output format should be [] > [], e.g., [2] > [1].
Only respond with the ranking results; do not say any word or explain.<|eot_id|><|
    start_header_id|>assistant<|end_header_id|>

[/INST]
-----

Example Output:
[5] > [12] > [7] > [16] > [19] > [15] > [10] > [6] > [9] > [8]
> [14] > [4] > [3] > [2] > [1] > [18] > [17] > [13] > [11] > [20]

```

275

276 References

- 277 [1] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill™: A bayesian skill rating system.
278 *Advances in Neural Information Processing Systems*, 19, 2006.
- 279 [2] Iain Mackie, Jeffrey Dalton, and Andrew Yates. How deep is your learning: the dl-hard annotated
280 deep learning dataset. In *Proceedings of the 44th International ACM SIGIR Conference on*
281 *Research and Development in Information Retrieval*, pages 2335–2341, 2021.
- 282 [3] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. Beir:
283 A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-*
284 *fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*
285 *(Round 2)*, 2021.
- 286 [4] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. Rankvicuna: Zero-shot listwise
287 document reranking with open-source large language models. *arXiv preprint arXiv:2309.15088*,
288 2023.
- 289 [5] Meta AI. Llama 3.3 70B Instruct model card. [https://huggingface.co/meta-llama/](https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct)
290 [Llama-3.3-70B-Instruct](https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct), 2024. Model release date: 6 Dec 2024. Accessed 23 May 2025.

- 291 [6] Yun Zhou and W Bruce Croft. Query performance prediction in web search environments.
292 In *Proceedings of the 30th annual international ACM SIGIR conference on Research and*
293 *development in information retrieval*, pages 543–550, 2007.
- 294 [7] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. Rankzephyr: Effective and robust
295 zero-shot listwise reranking is a breeze! *arXiv preprint arXiv:2312.02724*, 2023.