Decentralized Adaptive Ranking using Transformers

Gregoriadis, Marcel; Stokkink, Quinten; Pouwelse, Johan

DOI
10.1145/3721146.3721945

Publication date
2025

Document Version
Final published version

Published in
EuroMLSys 2025 - Proceedings of the 2025 5th Workshop on Machine Learning and Systems

Citation (APA)
Gregoriadis, M., Stokkink, Q., & Pouwelse, J. (2025). Decentralized Adaptive Ranking using Transformers. In *EuroMLSys 2025 - Proceedings of the 2025 5th Workshop on Machine Learning and Systems* (pp. 12-18). ACM. https://doi.org/10.1145/3721146.3721945

Decentralized Adaptive Ranking using Transformers

Marcel Gregoriadis
Delft University of Technology
Delft, The Netherlands

Quinten Stokkink
Delft University of Technology
Delft, The Netherlands

Johan Pouwelse
Delft University of Technology
Delft, The Netherlands

Abstract

Centralized platforms like TikTok are cause for significant concerns over information control, censorship, and bias. Decentralized systems offer a promising alternative, but their adoption is hindered by the lack of effective relevance ranking of search results. Existing decentralized approaches rely on heuristics that do not adapt to user behavior. This paper presents DART, the first decentralized ranking algorithm to leverage machine learning over users' search activities. DART adapts its ranking function using a Transformer-based learning-to-rank model trained on a real workload from a decentralized file-sharing application. We find that it improves over the best baseline by 19 % on our ranking metric (MRR).

CCS Concepts: • **Computer systems organization → Peer-to-peer architectures**; • **Information systems → Learning to rank**.

Keywords: Decentralized Systems, Information Retrieval, Relevance Ranking, Machine Learning

ACM Reference Format:
Marcel Gregoriadis, Quinten Stokkink, and Johan Pouwelse. 2025. Decentralized Adaptive Ranking using Transformers. In *The 5th Workshop on Machine Learning and Systems (EuroMLSys '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3721146.3721945

1 Introduction

TikTok's video recommendation algorithm can be used for election interference. For instance, Romania has annulled the 2024 presidential elections. In this case, the European Commission considers "TikTok's recommender systems" to be the target of "coordinated inauthentic manipulation or automated exploitation of the service" [6]. At the same time, no decentralized alternatives exist to TikTok, Google, or other Big Tech offerings. We aim to offer an alternative to Big Tech's black-box recommender systems, which determine what we read, see, believe, and vote. To do so, we leverage recent advances in machine learning to learn to recommend, i.e., rank, information in decentralized networks. We present the first such decentralized learning-to-rank algorithm.

We focus on the **problem of decentralized ranking of search results**. This focus is only one part of the greater goal of decentralized search [17]. Decentralized search is characterized by a network of nodes, which locally store only part of the "global" information that is stored among all nodes. When a node requires information that it does not store itself, it must request and rank information from other nodes. However, decentralized ranking itself works best with global information [1]. Thus far, no decentralized search algorithm has mitigated this chicken-and-egg problem sufficiently, to achieve mass popularity. Centralized platforms dominate, as they simply offer a superior experience.

The lack of global information can be overcome by not depending on it for ranking. For this reason, *all* related relevance ranking work in decentralized systems is based on *static* heuristics [9, 10, 16, 18, 23, 30]. This is a problem, as information is not static. New information may be introduced [1] and changes in wordings of descriptions may occur, also known as "semantic drift" [34]. Any ranking must continually adapt to changes in information. We propose to use machine learning to adapt to the dynamic nature of information in decentralized systems.

Recently proposed attention models, known as Transformers [28], are validated for ranking in a non-decentralized context [20, 21] and particularly promising in their application to decentralized ranking. These machine learning models offer light-weight inference that can feasibly be run by nodes with even lower-end hardware. At the same time, these models can deal with the dynamic nature of information in decentralized ranking systems. They can use online learning, i.e., fine-tuning, as new information is introduced.

We propose to make use of Transformers to continuously learn to rank information in a decentralized network. Our contribution is the design, implementation, and evaluation of a **decentralized learning-to-rank algorithm**. We call this algorithm *Decentralized Adaptive Ranking using Transformers (DART)*. Furthermore, we provide the first learning-to-rank dataset compiled from workload on a decentralized system.

We introduce the concept of learning-to-rank and also give an overview of the related work in decentralized relevance ranking in Section 2, we define our system model and assumptions in Section 3, give the design of DART in Section 4, outline our methodology in Section 5, and present the experimental evaluation in Section 6.

EuroMLSys '25, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1538-9/2025/03
https://doi.org/10.1145/3721146.3721945

2 Background and Related Work

Users tend to focus their attention on the first 2–3 search results [32]. Hence, effective relevance ranking is paramount. Learning-to-rank (LTR) describes a class of machine learning techniques that optimizes the ranking in a list of search results based on patterns learned from past user interactions. Research on the topic dates back to the late 2000s and has since been a crucial technology in information retrieval [4, 15]. Recent approaches have started leveraging the Transformer architecture, enabling context-aware ranking models that improve upon traditional methods by employing self-attention mechanisms [20, 21]. For our model, we build upon the framework presented by Pobrotyn et al. [21]. We consider this state-of-the-art in LTR, as they have demonstrated significant improvements over traditional approaches across standard benchmarks. Compiling feature vectors for query-document pairs is a crucial step in any LTR-based system [13]. Mostly, those vectors are composed of term-based metrics, including formulas like BM25, which get extracted from different elements of the document (e.g., body, title, URL) [24]. In addition, quality scores like PageRank and usage statistics like dwell time also contribute to the feature set. Available LTR datasets, such as WEB30k [24], Istella [8], and Yahoo!'s [5], are curated by companies operating centralized web search engines.

Ranking based solely on term-based metrics ignores a lot of additional information that can serve as strong indicators to assess the relevance of a document to the user and their specific query. Google's PageRank algorithm has inspired a lot of research on decentralized web search [1, 2, 19, 31]. However, its principles do not extend to multimedia content search (i.e., files with no cross-links) on which we focus in this work. Other metrics must be considered as potential indicators of relevance. A strong indicator of relevance is the popularity of a document. In decentralized storage systems, this metric can be extracted from the *resource quantity*, i.e., the number of nodes that provide it [9, 23, 30]. Ranking by resource quantity has been proposed early on for Gnutella [9]; it was implemented for the BitTorrent-based client Tribler [23]; more recently, it has been proposed for IPFS in CASearch [30]. Furthermore, ranking in CASearch relies on bandwidth as an estimator for both content locality and expected transmission speed. CASearch and Tribler also consider the "freshness" of a document, i.e., the time since its inception, in their respective ranking functions. MAAY [18] and UsersRank [19] account for the freshness of user engagement with a document, rather than the document's publication age, within their ranking functions. Tribler uses the number of *leechers* (i.e., peers that actively download the file) to derive a similar metric. MAAY and UsersRank, furthermore, incorporate elements of collaborative filtering. Collaborative filtering is also at the core of G-Rank [10], which examines commonalities between nodes and their behavior.

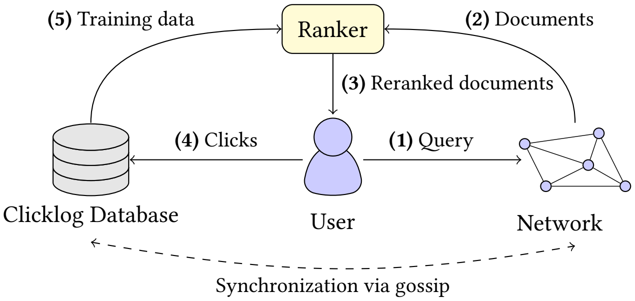


Figure 1. Overview of the system model and its interactions.

3 System Model

This section describes our system model, including its scope and actors, and the assumptions we make. An overview of our system model is illustrated in Figure 1 and it consists of five main steps. A user sends a query (1) to the network, which returns results (2) that are reranked by the ranker (3). The user's clicks (4) are stored in the clicklog database, which provides training data (5) to improve the ranker. Users gossip their clicklogs and store them in their local database.

Our system considers a network of nodes, which form a connected graph. Assuming the graph remains connected, nodes may join or leave the network at any time. We assume nodes engage in decentralized searches, i.e., searches for information that other nodes store. Two main types of information exist in our system: queries and documents.

Every search consists of a node sending another node a *query* created by the user that operates the node. Whenever a node receives such a query from another node, we assume it answers with results. We call these results *documents*. Further, we assume they have a *title*, along with user-annotated *tags*, and a number of *seeders* and *leechers*. Seeders denote the number of nodes storing the document, while leechers denote the number seeking the document at any given moment. We assume documents can be retrieved from a distributed database based on their title, using traditional text-based matching of query and document title.

After a node receives documents as the results of its query, it engages in ranking of the documents using a local model. The goal of this ranking is to maximize the likelihood that the user *clicks* the first ranked item, whenever the ranked results are displayed in a user's interface. We explicitly note that a click is personal to a user. Therefore, there is no consistent ranking of results for all nodes.

4 Design of DART

We introduce DART, the first self-learning decentralized ranking algorithm. Our model architecture is based on the learning-to-rank framework proposed by Pobrotyn et al. [21],

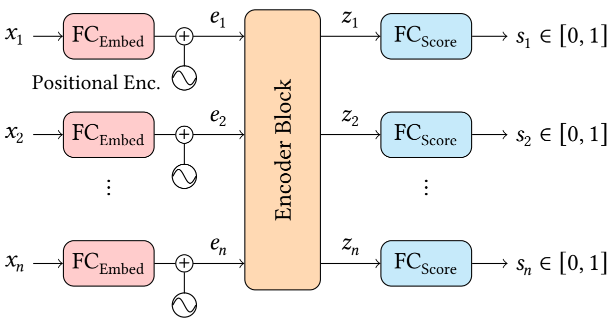

x_1
FC_{Embed}
e_1
z_1
FC_{Score}
$s_1 \in [0, 1]$
Positional Enc.
x_2
FC_{Embed}
e_2
Encoder Block
z_2
FC_{Score}
$s_2 \in [0, 1]$
x_n
FC_{Embed}
e_n
z_n
FC_{Score}
$s_n \in [0, 1]$

Figure 2. Model architecture and data flow.

which we adapt to our decentralized setting. All hyperparameters presented in this section are derived from preliminary experiments designed for optimal performance on our dataset, specifically our first experiment (Section 6.1).

This architecture leverages the self-attention mechanism of transformers to construct a *context-aware* ranker. That is, rather than scoring items independently, the scoring is influenced by the interactions with other items in a given list. We show the model architecture and data flow, which we explain momentarily, in Figure 2. The re-ranking of documents is defined as $R(X)$, where the input

$$X = \{x_1, x_2, \ldots, x_n\} \quad \text{with} \quad x_i \in \mathbb{R}^f$$

denotes the set of n results retrieved for a given query. The ranker processes the input as a sequence of tokens. Each token $x_i \in \mathbb{R}^f$ represents a vector of $f = 31$ features encoding the relevance of a document to a particular query (see Table 1). The set of features is inspired from traditional learning-to-rank datasets [24], as well as extracted from related work, namely Tribler [23], DINX [9], and Panaché [16]. We verified the utility of each feature through an ablation study (see Table 2). The number of features is smaller than that used in classical learning-to-rank datasets (see Table 3). This allows for a smaller model, which reduces computational and memory overhead and fits our target deployment on consumer-grade devices, which may lack dedicated GPUs.

The input tokens are linearly projected through a shared fully connected layer FC_{Embed} to a space of size $d_{\text{FC}} = 24$ and subsequently augmented with fixed positional encoding. As the retrieved result set is already ordered according to some preliminary ranking, we use positional encoding to correct for position bias. This transforms each input token $x_i \in X$ into its embedding form

$$e_i = \text{FC}_{\text{Embed}}(x_i) + \text{PosEnc}(i), \quad e_i \in \mathbb{R}^{d_{\text{FC}}}.$$

The sequence of input embeddings $\{e_1, e_2, \ldots, e_n\}$ passes through a Transformer encoder block with two attention heads, a hidden dimension of 96, and a dropout rate of 0.1. This encoder outputs a sequence of *context-aware* embeddings $\{z_1, z_2, \ldots, z_n\}$ with $z_i \in \mathbb{R}^{d_{\text{FC}}}$.

Table 1. Features of a Document-Query Pair

ID	Description
0	BM25 score [25]
1–5	Term frequency (TF): min, max, mean, sum, and variance [26]
6–10	Inverse document frequency (IDF): min, max, mean, sum, and variance [26]
11–15	TF*IDF: min, max, mean, sum, and variance [26]
16	Cosine similarity of the TF*IDF 5-tuple
17	Number of query terms in the document title
18	Ratio of query terms in the document title
19	Number of characters in the document title
20	Number of terms in the document title
21	Number of terms in the query
22	Query matches document title exactly
23	Ratio of query terms matching the document title
24	Number of nodes storing the doc. (seeders) [23]
25	Number of nodes querying the doc. (leechers) [23]
26	Number of times the doc. has been clicked [9]
27	Number of times the document was selected when one of the document's terms was also part of the query terms (hit count) [16]
28	Document rank in the result, before re-ranking
29	Number of user-annotated document tags [23]
30	Freshness (time since document creation) [23, 30]

Table 2. Ablation Study

Method	Feat. IDs	MRR	Δ
DART (base)	–	**0.380**	–
w/o term-based features	0–23	0.373	-0.007
w/o number of seeders	24	0.368	-0.012
w/o number of leechers	25	0.374	-0.006
w/o click count	26	0.374	-0.006
w/o query hit count	27	0.379	-0.001
w/o pos	28	0.368	-0.012
w/o tag count	29	0.375	-0.005
w/o freshness	30	0.377	-0.003

To obtain ranking scores, the tokens are independently processed through another shared fully connected layer FC_{Score}, again of size d_{FC}. The output of this layer is activated by a sigmoid activation function to normalize the scores:

$$s_i = \sigma(\text{FC}_{\text{Score}}(z_i)), \quad s_i \in [0, 1].$$

The ranking model is optimized using the NeuralNDCG loss function, which directly approximates the NDCG metric used for ranking performance evaluations, and has been demonstrated to work best with this model architecture [22].

Further, the model undergoes training for up to 100 epochs, incorporating an adaptive learning rate and early stopping with a patience of 5 to prevent overfitting. In total, the model comprises 8065 parameters, with a memory footprint of just 31.5 kB.

Table 3. Learning-to-Rank Datasets

Dataset	Queries	Users	Features
WEB30k [24]	30 000	not published	136
Istella [8]	33 018	not published	220
Yahoo! [5]	29 921	not published	700
Tribler	**9068**	**948**	**31**

5 Method

This section gives the method by which we perform our experimental evaluation in Section 6. In our evaluation, we compare the performance of DART with the state-of-the-art of decentralized ranking algorithms (see Table 4) in three experiments with increasing realism. Furthermore, we add our own variation of DINX, which leverages the number of seeders as a popularity metric alternative to click counts, dubbed DINX-s. We re-rank documents in our dataset using each algorithm and assess its performance using Mean Reciprocal Rank (MRR), a standard metric for ranking evaluation [29, 33]. We explain this metric in Section 5.2.

Table 4. Baselines of Ranking Algorithms for Decentralized Multimedia Search (CD = Context-Dependent)

Algorithm	Year	Ranking Strategy	CD
DINX [9]	2001	Order by click counts	●
DINX-s		Order by seeders	○
Panaché [16]	2002	Order by keyword hit count	●
MAAY [18]	2006	Personalized to user's interest in queried terms, doc. popularity and relevance	●
Tribler [23]	2008	Heuristic based on title, seeders, leechers, freshness	○
G-Rank [10]	2023	User and doc. similarity	●
DART	**2025**	**Machine learning**	●

5.1 Dataset

Available datasets from centralized (web) search engines like AOL [12] or Bing [24] lack key attributes specific to decentralized applications, such as seeders and leechers. Furthermore, decentralized content has a different structure, e.g., the absence of semantic identifiers (URLs). Document titles also tend to follow domain-specific naming conventions, often including metadata such as release years, quality indicators, and encoding formats. Given these distinct attributes, publicly available datasets prove inadequate. This makes the construction of a dedicated dataset essential for the realistic evaluation of our algorithm in decentralized settings. Therefore, we created our own dataset based on workload from a real decentralized application, *Tribler*.

Tribler is a fully decentralized file-sharing application [23] with integrated search and anonymized file sharing. Tribler has been part of an ongoing research project since 2005 and has over 40 000 monthly active users [27]. Tribler uses gossip for search, content discovery, and node discovery. We deployed a crawler to initially discover identifiers, seeders, and leechers, of documents in the public Tribler network. We later enriched this data with the document's title, creation date, and number of user-annotated tags, again by querying the network. Using a temporary access-restricted API (to disallow potential abuse) we obtain search result lists, including the query, the ordered result list, and the clicked result position. Our code, as well as our dataset of compiled feature vectors, is available on GitHub [11].

5.2 Evaluation Metric

Numerous metrics have been developed to evaluate the quality of result ranking in information retrieval systems. The most commonly used metrics [4, 21, 33] are Normalized Discounted Cumulative Gain (NDCG) [14], Mean Average Precision (MAP) [3], and Mean Reciprocal Rank (MRR) [29].

NDCG considers the full list of results and measures its alignment with the ideal ranking. This metric, however, is not a good fit for decentralized workloads. Each clicklog contains exactly one relevant document, with all others considered irrelevant. Thus, an "ideal" ranking can be any arbitrary order of results as long as the relevant document is found at position 1. This is different from, e.g., WEB30k [24], where documents are graded with multiple judgment levels. As pointed out by Cao et al. [4], when relevance judgments are binary (either *relevant* or *irrelevant*), the appropriate metric is MAP. In our special case, because our dataset always determines exactly one relevant document per query, MAP simplifies to MRR. We calculate MRR by averaging the Reciprocal Rank (RR) across all evaluated clicklogs. RR is calculated as

$$\text{RR} = \frac{1}{\text{rank of relevant document}}.$$

Hence, its value ranges from 0 to 1, where a score of 1 indicates optimal ranking.

Furthermore, we report Recall@k for a more intuitive and easily interpretable evaluation. Recall@k quantifies the percentage of times that the desired document appears within the top-k presented results.

6 Experimental Evaluation

Our first experiment partitions the *entire* dataset into a context and test set. Our second experiment examines ranking performance when the context is smaller, i.e., clicklogs are more scarce. Finally, our third experiment simulates a decentralized system aligned with our envisioned system model (see Section 3). It, therefore, provides the most realistic evaluation of DART and the baseline algorithms.

Recall that DART and other baselines depend on context, i.e., a set of observed clicklogs before application, for their performance. Henceforth, we refer to the context used for our evaluations as $C_{\text{ctx}} \subset D$, where D denotes all $|D| = 9068$ clicklogs available in our dataset. Similarly, $C_{\text{test}} \subset D$ denotes the set of clicklogs used for testing and evaluation. In all instances, C_{ctx} and C_{test} are disjoint. For evaluations of Tribler and DINX-s, C_{ctx} is ignored.

6.1 Ranking Performance

Our first experiment demonstrates the ranking performance of each algorithm on a random sample of 10 % of the entire dataset, where the remainder is considered as context. Explicitly, we partition D, such that

$$|C_{\text{test}}| = \frac{|D|}{10} \quad \text{and} \quad C_{\text{ctx}} = D \setminus C_{\text{test}}.$$

While Tribler and DINX-s are stateless algorithms that do not operate on context, the other algorithms rely on context. That includes DART, which derives its training and validation set from it.

As we show in Table 5, DART significantly outperforms the second-best algorithm, Tribler, which corresponds to the dataset's original ranking. To illustrate this improvement, Figure 3 compares the distribution of clicked positions for Tribler and DART. As desired, the distribution shifts to the left, indicating that users are more likely to click results ranked closer to the top. The average position for relevant documents improved significantly, from 22 to 15. Furthermore, we note a high variance across all algorithms. We postulate that this is caused by the high variance in result set sizes within our dataset.

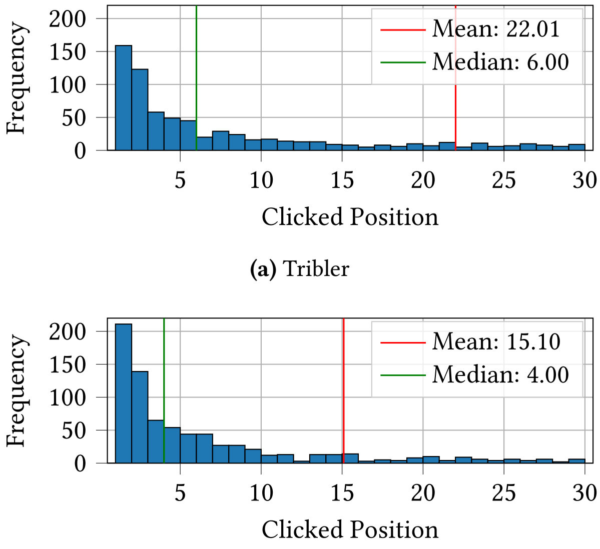
While DINX, Panaché, MAAY, and G-Rank show lower MRR and Recall than the competition, their ability to surpass random ranking indicates that their metrics capture relevance at least to some extent. We postulate that with a bigger data set or more homogeneous data, these algorithms would perform better. Our following experiment will shed more light on it.

6.2 Impact of Context Size on Performance

Tribler and DINX-s are stateless, meaning they rank documents based solely on static features such as document title, seeders, and freshness. DINX, as well as Panaché, MAAY, and G-Rank, on the other hand, maintain a state that is updated with each observed clicklog. Their ranking decision is influenced by this data, and moreover, by the amount of this data. As an ML-based algorithm, this also holds true for DART. The number of available clicklogs may vary between peers, e.g., depending on their uptime in the network. We are therefore interested in an analysis of the relationship between ranking performance and the number of observed clicklogs.

Table 5. Ranking Performance (Rec.=Recall)

Algorithm	MRR (± SD)	Rec.@1	Rec.@5	Rec.@10
Random	*0.18 ± 0.26*	0.15	0.31	0.42
G-Rank	0.25 ± 0.32	0.22	0.41	0.52
MAAY	0.27 ± 0.34	0.26	0.43	0.52
Panaché	0.28 ± 0.35	0.25	0.43	0.53
DINX	0.28 ± 0.34	0.26	0.46	0.55
DINX-s	0.31 ± 0.36	0.28	0.49	0.61
Tribler	0.32 ± 0.35	0.31	0.50	0.61
DART	**0.38 ± 0.37**	**0.38**	**0.61**	**0.73**



(a) Tribler

(b) DART

Figure 3. Comparison of clicked position distributions for Tribler and DART, truncated at position 30.

For this experiment, we evaluated different context sizes, setting

$$|C_{\text{ctx}}| = i \quad \text{for} \quad i \in [0, 1, \ldots, |D| - 100].$$

In every iteration, we also randomly sampled

$$C_{\text{test}} \subset D \setminus C_{\text{ctx}}, \quad \text{such that} \quad |C_{\text{test}}| = 100.$$

The compilation of large sets of clicklogs into the feature vectors needed for DART is computationally expensive. For practical reasons, therefore, we do not evaluate DART for every step in the interval but instead increase the size by

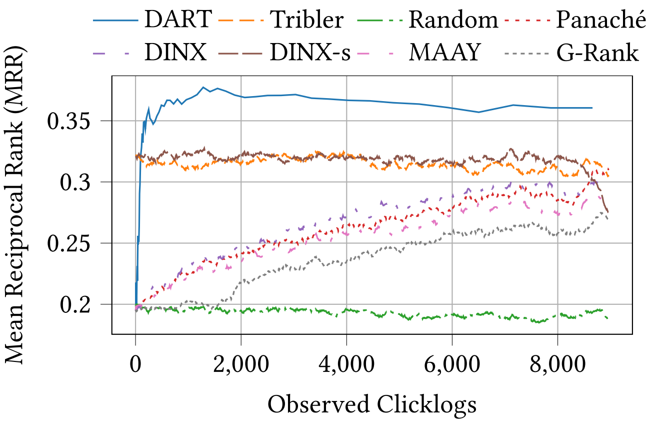


Figure 4. Ranking performance with increasing context size. Adaptive smoothing has been applied.

roughly 10 % each time (i.e., $i \leftarrow \lfloor 1.1 \cdot i \rfloor + 1$). We show the results of this experiment in Figure 4. Due to the high variance in the data, we present a smoothed representation using a moving average[1] to make the trend more apparent. As expected, Tribler and DINX-s are unaffected by the number of observed clicklogs and demonstrate stable performance from the onset. DART, however, quickly overtakes both baselines at only 56 observed clicklogs. Other algorithms' MRR increases steadily but at a much slower pace. That trend suggests that the peak for those algorithms is still ahead, with some algorithms likely surpassing Tribler and DINX-s after around 10 thousand clicklogs.

6.3 Decentralized Network Simulation

We finally demonstrate DART's performance in a simulated decentralized setting. When users perform searches, queries and clicked documents often relate to previously issued searches. Losing this correlation poses a challenge for DINX, Panaché, MAAY, and G-Rank, which rely on exact term match statistics. DART is more flexible as it is able to generalize from abstract user engagement patterns rather than depending on exact term matches. Nonetheless, it remains sensitive to extreme diversity in its context data. Recall, for example, that the term-based click counters of DINX and Panaché are included in DART's feature set (feature ID 26 and 27 in Table 1). In this experiment, we simulate a decentralized network by preserving the original user mappings and chronology of the clicklogs in our dataset. This approach ensures the most realistic evaluation of DART.

Specifically, for this experiment, we filtered our dataset to include only users with at least 10 clicklogs that have at least 30 documents in their result set, respectively. This amounted to a total of 112 users with varying amounts of clicklogs. Consistent with prior experiments, we partition each user's set of clicklogs into 90 % context and 10 % for testing. Contrary to prior experiments, however, we respect the original chronology of clicklogs, such that the test set corresponds to 10 % of each user's most recent queries.

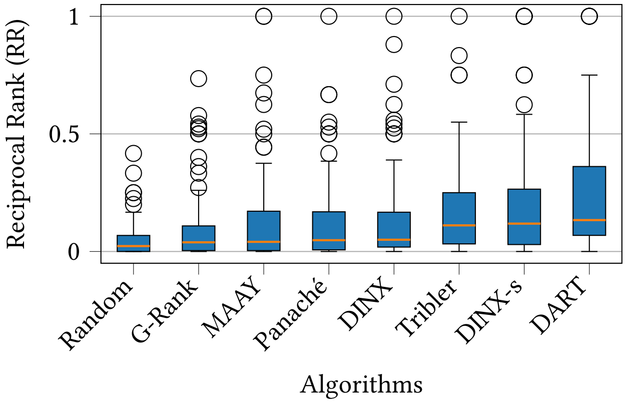


Figure 5. Ranking performance aggregated over 112 peers.

Given our assumption of full clicklog gossip, each user is exposed to the complete set of context clicklogs, meaning they all operate on the same dataset. Thus, for DART, we simplify the experiment by training just one model. Finally, we test the model against each user's test set individually. Our baselines, accordingly, make their rankings on the users' test sets on the basis of all context clicklogs. This led to the results shown in Figure 5. We observe that DART outperforms Tribler ($p = 0.12$), DINX-s ($p = 0.36$), and our other baselines ($p < 0.0009$), in terms of RR.

7 Conclusion

Decentralization of any algorithm is challenging and decentralized relevance ranking is a known difficult problem. AI may underpin the revival of the peer-to-peer movement, as Big Tech further gains dominance. We established the baseline for such decentralized ranking using transformers. However, significant gains remain beyond our baseline, we believe. With our privacy-preserving dataset and open-source prototype, we aim to contribute to further advancing decentralized search. However, deployment of fully decentralized machine learning whilst *also* offering privacy and spam resilience remains unsolved. In 2012, we successfully deployed decentralized SGD inside Tribler [7]. We are working on deploying DART. We believe that within a few years, the problems of scalability, privacy, attack-resilience, and continuous learning of deployed decentralized machine learning will be mitigated.

Acknowledgments

This work was funded by Dutch national NWO/TKI science grant BLOCK.2019.004.

[1] We applied a moving avg. over a window of 200 data points. Fort DART, in particular, we used a window size that decays logarithmically from 20 to 1.

References

[1] Karl Aberer and Jie Wu. 2003. A framework for decentralized ranking in web information retrieval. In *Asia-Pacific Web Conference*. Springer, 213–226.

[2] Reaz Ahmed, Md Faizul Bari, Rakibul Haque, Raouf Boutaba, and Bertrand Mathieu. 2014. DEWS: A decentralized engine for Web search. In *10th International Conference on Network and Service Management (CNSM) and Workshop*. IEEE, 254–259.

[3] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*. Vol. 463. ACM press New York.

[4] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*. 129–136.

[5] Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. In *Proceedings of the learning to rank challenge*. PMLR, 1–24.

[6] European Commission. 2024. Commission opens formal proceedings against TikTok on election risks under the Digital Services Act. Press release IP/24/6487. Available from https://ec.europa.eu/commission/presscorner/detail/en/ip_24_6487.

[7] Kornél Csernai and Márk Jelasity. 2012. Distributed machine learning using the tribler platform.

[8] Domenico Dato, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2016. Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. *ACM Transactions on Information Systems (TOIS)* 35, 2 (2016), 1–31.

[9] Blaise Gassend, Thomer M Gil, and Bin Song. 2001. DINX: A decentralized search engine. (2001).

[10] Andrew Gold and Johan Pouwelse. 2023. G-Rank: Unsupervised Continuous Learn-to-Rank for Edge Devices in a P2P Network. *arXiv preprint arXiv:2301.12530* (2023).

[11] Marcel Gregoriadis. 2025. DART: Research Repository. GitHub repository: https://github.com/mg98/DART. Accessed: 11-02-2025.

[12] Qian Guo, Wei Chen, and Huaiyu Wan. 2021. AOL4PS: A large-scale data set for personalized search. *Data Intelligence* 3, 4 (2021), 548–567.

[13] Xinzhi Han and Sen Lei. 2018. Feature selection and model comparison on microsoft learning-to-rank data sets. *arXiv preprint arXiv:1803.05127* (2018).

[14] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[15] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

[16] Tim Lu, Shan Sinha, and Ajay Sudan. 2002. *Panache: A scalable distributed index for keyword search.* Technical Report. Citeseer.

[17] Petru Neague, Marcel Gregoriadis, and Johan Pouwelse. 2024. De-DSI: Decentralised Differentiable Search Index. In *Proceedings of the 4th Workshop on Machine Learning and Systems*. 134–143.

[18] Frédéric Dang Ngoc, Joaquín Keller, and Gwendal Simon. 2006. MAAY: a decentralized personalized search system. In *International Symposium on Applications and the Internet (SAINT'06)*. IEEE, 8–pp.

[19] Athanasios Papagelis and Christos Zaroliagis. 2012. A collaborative decentralized approach to web search. *IEEE transactions on systems, man, and cybernetics-part a: systems and humans* 42, 5 (2012), 1271–1290.

[20] Changhua Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, Junfeng Ge, Wenwu Ou, et al. 2019. Personalized re-ranking for recommendation. In *Proceedings of the 13th ACM conference on recommender systems*. 3–11.

[21] Przemysław Pobrotyn, Tomasz Bartczak, Mikołaj Synowiec, Radosław Białobrzeski, and Jarosław Bojar. 2020. Context-aware learning to rank with self-attention. *arXiv preprint arXiv:2005.10084* (2020).

[22] Przemysław Pobrotyn and Radosław Białobrzeski. 2021. Neuralndcg: Direct optimisation of a ranking metric via differentiable relaxation of sorting. *arXiv preprint arXiv:2102.07831* (2021).

[23] Johan A Pouwelse, Pawel Garbacki, Jun Wang, Arno Bakker, Jie Yang, Alexandru Iosup, Dick HJ Epema, Marcel Reinders, Maarten R Van Steen, and Henk J Sips. 2008. TRIBLER: a social-based peer-to-peer system. *Concurrency and computation: Practice and experience* 20, 2 (2008), 127–138.

[24] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. *CoRR* abs/1306.2597 (2013). http://arxiv.org/abs/1306.2597

[25] Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*. Springer, 232–241.

[26] Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* 28, 1 (1972), 11–21.

[27] Tribler. [n. d.]. Release dashboard | Tribler. https://release.tribler.org/dashboard/. [Accessed 11-02-2025].

[28] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[29] Ellen M Voorhees et al. 1999. The trec-8 question answering track report.. In *Trec*, Vol. 99. 77–82.

[30] Feng Wang and Yanjun Wu. 2020. Keyword search technology in content addressable storage system. In *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 728–735.

[31] Jie Wu and Karl Aberer. 2004. Using siterank for decentralized computation of web document ranking. In *Adaptive Hypermedia and Adaptive Web-Based Systems: Third International Conference, AH 2004, Eindhoven, The Netherlands, August 23-26, 2004. Proceedings 3*. Springer, 265–274.

[32] Xinwei Wu, Hechang Chen, Jiashu Zhao, Li He, Dawei Yin, and Yi Chang. 2021. Unbiased learning to rank in feeds recommendation. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 490–498.

[33] Jun Xu and Hang Li. 2007. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. 391–398.

[34] Lu Yu, Bartlomiej Twardowski, Xialei Liu, Luis Herranz, Kai Wang, Yongmei Cheng, Shangling Jui, and Joost van de Weijer. 2020. Semantic drift compensation for class-incremental learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 6982–6991.