International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.4 (2025) pp. 4379-4388
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

Cloud-Native Scalable Learning-to-Rank Architecture for Real-Time Search Optimization

Saraschandra Arveti[1*], Anish Hadkar[2], Mani Teja Nutalapati[3]

[1]Independent Researcher Virginia, USA
* **Corresponding Author Email:** saras.arveti@gmail.com- **ORCID:** 0000-0002-5247

[2]Independent Researcher Washington, D.C., USA
Email: HadkarAnish@gmail.com- **ORCID:** 0000-0002-5247-2250

[3]Independent Researcher Virginia, USA
Email: manitejan16@gmail.com - **ORCID:** 0000-0002-5247-1150

Article Info:

DOI: 10.22399/ijcesen.5118
Received : 05 November 2025
Revised : 25 December 2025
Accepted : 27 December 2025

Keywords

Learning-to-Rank,
Cloud-Native Architecture,
Real-Time Search Optimization,
Scalable Microservices,
Machine Learning Ranking,
Distributed Search Systems

Abstract:

Today, search engines must be capable of delivering extremely relevant results while handling a large number of queries simultaneously in real time. A cloud-native, scalable Learning to Rank (LTR) system can leverage distributed computing, microservices, and on-demand cloud resources for the efficient search optimization task. This approach will incorporate machine learning based ranking models into the search pipeline, thereby enabling the system to re-evaluate and re-order the results in line with user intent, contextual signals, and historical interaction data in a dynamic manner. The proposed architecture utilizes containerized services, stream processing, and scalable storage for real, time feature extraction, model inference, and continuous model updates. By making use of orchestration platforms and serverless components, the system can automatically scale without any human intervention even when workload fluctuations are extreme. It will also maintain low latency and high availability. Data pipelines collect clickstream and behavioral data which are then used to train and periodically retrain the LTR models so that the ranking strategies always reflect the changing preferences of the users. Besides, the design includes facilities for A/B testing, monitoring, and model versioning which give a way to controlled experimentation and performance optimization. Using caching layers and distributed indexing mechanisms results in improved response times and reduced computational overhead.

1. Introduction

Search systems currently play a critical role in providing users with fast and relevant information access as data collections are enormous and continuously increasing. Several recent applications among which e, commerce platforms, digital libraries, enterprise knowledge bases, and multimedia repositories require advanced search engines for delivering accurate and user-oriented results. As both data and queries of users tend to increase, traditional ranking methods lose their capability to balance performance and relevance at the same time. So, the combination of machine learning and system architectures that can be scaled for high performance has become a necessity for improving search quality and responsiveness [1]. Learning to Rank (LTR) from different machine learning approaches is one that is singled out for improving the ranking of search results. Unlike ordinary rule-based ranking, LTR models figure out the best way to order results by harnessing historical data like user clicks, query logs, and interaction patterns. By analyzing query features, document attributes, and user behavior jointly, these models can greatly enhance the relevance of search results [2]. However, the integration of LTR models in live time search queries still remains a major challenge given the latency constraints, providing a model that is scalable, efficient feature extraction, and also constant model updating.

Cloud-native technologies can be very effective tools against these problems because they offer adaptable, scalable, and highly available infrastructures. Cloud-native systems, which rely on microservices, containerization, distributed

storage, and orchestration, can dynamically adjust the size of search systems based on workload demands. This model allows for the deployment of different ranking services independently, which means the development, scaling, and maintenance of system components can be done separately[3]. Also, cloud-based data processing pipelines make it possible to collect and analyze user engagement data continuously, which is then used to train and update ranking models regularly, leading to improved performance. Embedding Learning to Rank techniques in a cloud-native setup is a big help in making real-time search ranking work efficiently. Feature extraction components get relevant signals from distributed data stores; in the meantime, model evaluation components are doing ranking score (re)estimations with the least latency possible. On top of that, distributed caching and indexing contribute to speeding up query execution and reducing the computational load. Such a configuration is capable of maintaining ongoing operation, handling failures, and making changes with minimal disruptions, so it is very suitable for large, scale scenarios where people's demand for speed and relevance keeps increasing.

Traditional search engines only satisfy basic queries whereas modern search engines are capable of understanding the user's intent. They satisfy user needs by offering a relevant and rich set of answers snippets suggestions, and results. Such enhanced functionality can result in a complex system of over a thousand microservices, a number of different stores, hundreds of instances, and a dedicated team of engineers. Owing to this complexity, it is almost impossible to make a change without affecting the overall system. Therefore, the management of the dependencies and the release of the products can become quite a challenge. Today, search engines should be enabling experiments, monitoring, and continuous improvement. Cloud, native platforms allow A/B testing of ranking rules, automate deployment pipelines through scheduling, and system performance monitoring. These features provide developers and researchers a way to upgrade ranking algorithms and other system components without interrupting the live services. In general, a cloud, native scalable Learning to Rank environment constitutes a great platform for tackling the problems of real-time search improvements. Once a machine learning, base ranking is integrated with the distributed cloud infrastructure, organizations will be capable of developing search engines that are not only remarkably fast but also scalable and able to continue providing personalized and relevant results to users at any given time.

The main Key Contributions are:

- Design of a Cloud Native Learning to Rank Framework: Put forward a scalable system design that embeds machine learning ranking models in a distributed cloud based search setup.
- Real Time Feature Processing and Model Inference: Offers ways to streamline feature retrieval, minimize the delay in model evaluation, and keep the ranking continuously updated even in high throughput scenarios.
- Scalable and Adaptive Search Optimization: This paper explains the role of cloud native tools like micro services, containers and distributed data pipes in boosting the scalability, reliability and performance of contemporary search engines.

2. Literature Review

Learning to Rank (LTR) has become an extremely crucial technique to increase the relevance of the search results in the most advanced information retrieval systems. Earlier ranking methods either relied on manually crafted heuristics or very simple statistical models, so they inherently couldn’t capture the complex query-document relationships. With the rise of machine learning, LTR has become the predominant means through which ranking functions are automatically learned from the training data and user interactions. The pioneering work in this field mainly exploited user behavior to improve search ranking. In [5], the authors presented a technique for leveraging the clickthrough data to get the most out of search engines and demonstrated that user implicit feedback is an excellent source for training ranking algorithms. This paper really emphasized the significance of using real user interactions to enhance search relevance since, in this way, users don't have to provide manual judgments which are usually low in quantity.

Subsequently, a great deal of work on ranking algorithms was done. In [6], pairwise ranking methods were replaced by listwise ones that consider the whole list of search results when training the model. The authors found that listwise approaches better model ranking relations, and as a result, there are often more successful retrievals. Moreover, [7] introduced new ideas by describing the top-ranking methods such as RankNet, LambdaRank, and LambdaMART. Because of their ability to efficiently optimize ranking metrics like Normalized Discounted Cumulative Gain (NDCG), these algorithms have become the leading ones in the field of commercial search engines. Along with the unprecedented expansion of deep learning, the concept of leveraging neural network architectures

for ranking problems began to attract widespread attention among researchers. As an illustration, [8] designed an attention-based deep neural network for learning to rank with the aim of demonstrating that attention mechanisms are not only able to interpret complex feature interactions but can also significantly improve ranking performance. Meanwhile, from a statistical perspective, [9] investigated the advantages of different learning to rank (LTR) techniques and observed that machine learning-based ranking models outperform traditional ranking models significantly in almost all scenarios of search.

LTR techniques have been widely implemented in specialized domains such as digital libraries and expert search. For example, [10] investigated the usage of LTR techniques for expert finding in academic digital libraries. They demonstrated that ranking models can be extremely effective in analyzing publication data and citation information in order to identify domain experts. Their research mainly addressed the potential of LTR systems to facilitate knowledge discovery in scholarly databases. Besides ranking algorithms, several researchers have realized that semantic representations and embeddings can be good means for improving retrieval performance. [11] proposed a unified traceability recovery method that is based on word embeddings and is not only simple to execute but also very effective in finding semantic relations between source code and natural language. Later, [12] took a pointer from the idea of word embeddings learned for the traceability recovery task and proceeded further on the adaptation of word embedding to the contextual information which is novel and fresh. The results from these two works suggest that embedding, based representations not only can complement ranking systems in terms of capturing deeper semantic relations between queries and documents, but also can be the basis of new types of ranking models, such as neural ranking models.

Research also explored the structural aspects of ranking systems. For instance, [13] elaborated the learning-to-rank methods in information retrieval with details, highlighting major aspects of feature engineering, model selection, and evaluation metrics in ranking systems. In recent years, [14, 15] have developed a graph neural network (GNN) based ranking method which makes use of graph structures to improve resource ranking. Their work is a clear demonstration of the increasing trend of combining complex deep learning techniques with ranking frameworks. In fact, the literature depicts the journey of learning to rank methods starting from simple pairwise models up to the use of advanced deep learning and graph-based models. These breakthroughs serve as a strong foundation for the development of scalable and intelligent search systems. Integration of these ranking methods within cloud-native architectures will not only facilitate scalability and real-time processing but also improve search efficiency in large-scale information retrieval scenarios.

3. Methodology

3.1 Cloud-Native Learning-to-Rank System Architecture

The first step in the proposed approach is to develop a cloud-native Learning to Rank (LTR) system capable of supporting large-scale and highly efficient real-time ranking optimizations. The proposed system is a microservices-based one where various components related to query processing, feature generation, ranking model prediction, and data storage are completely independent but still communicate via minimal APIs. Such a compartmentalized structure is very good for scalability, providing great flexibility and making the roll-out of the system in cloud-based distributed environments even easier. System has five main stages: Data gathering Feature generation Model training [16-18] Ranking prediction Monitoring The user's queries are initially processed by a query router service, which then distributes the requests to the search servers. These servers are tasked with retrieving candidate documents that reside in the distributed indexes contained in high-capacity storage systems such as cloud databases or distributed file systems. The documents that are pulled out are subsequently referred to the feature computation service which calculates a bunch of features, e.g. textual similarity, document popularity, click, through rate, and contextual relevance. These features are then passed to the Learning, to, Rank model that has been trained to output a ranking score for every single document. A document's ranking score is obtained by passing its feature vector through a ranking function, which is the output of a training process:

$$Score(q,d) = f\big(\emptyset(q,d)\big) \quad (1)$$

The relationship between the query and document, extracted feature vectors and ranking model can be expressed in two ways:

1. In a linear fashion
2. By representing each variable, i.e., q=1, d=2, ϕ(q,d)=3, and f=4, using their respective weights.

$$Score\ (q,d) = \sum_{i=1}^{n} w_i x_i \quad (2)$$

Where

- xi represents the i-th feature
- wi represents the learned weight of the feature

This architecture ensures fault tolerance, scalability, and high availability while supporting real-time ranking [19].

3.2 Real-Time Feature Processing Framework

The Learning-to-Rank system depends heavily upon good quality features for determining the degree of relevance of items in its ranking process. To this end, a feature processing pipeline which operates in real-time has been put in place using a distributed streaming framework (e.g., Kafka) to allow for the processing of large amounts of user interaction data [20].

There are several types of features collected by the system including, but not limited to, query-related features (e.g., query length, query type), document-related features (e.g., keyword frequency, document popularity) and user-related features (e.g., click-through rate, dwell time).

Term relevance can typically be determined by either the TF-IDF weighting scheme between each query and document [21].

$$TFIDF(t,d) = TF(t,d) * IDF(t) \quad (3)$$

$$IDF(t) = log\frac{N}{df_t} \quad (4)$$

Where

- TF(t,d), the frequency of term t in document d
- N, the number of total documents
- dft, the number of documents, which contain term t

One other similarity measure that is commonly used is cosine similarity:

$$Sim(q,d) = \frac{q.d}{||q||||d||} \quad (5)$$

This similarity score helps evaluate the semantic closeness between the query vector and the document vector.

The feature processing framework delivers the models with up-to-date information from which a decision can be made in real-time about what rank to give to a document.

3.3 Training Models and Inference at Ranking Time

Models used in learning to rank are based on historical user search logs or labeled data. This type of training is directed toward improving the ranking of documents, so that the documents which are the most relevant to the user's query are at the top of the list of retrieved documents. Training of a ranking model is carried out using pairwise or listwise methods.

When a ranking model is trained through pairwise comparison, it learns to compare two documents based upon their rankings. Thus, the rank of a document can be incorporated into their comparison through the loss function that is used with a pairwise ranking model. The function for pairwise ranking can be expressed as follows:

$$L = \sum_{(i,j)} \log(1 + e^{-(s_i - s_j)} \quad (6)$$

Where

- si and sj represent predicted scores for documents i and j

Another important ranking evaluation metric is Normalized Discounted Cumulative Gain (NDCG):

$$DCG = \sum_{i=1}^{n} \frac{2^{reli} - 1}{log_2\,(i+1)} \quad (7)$$

$$NDCG = \frac{DCG}{IDCG} \quad (8)$$

Where

- reli is the relevance score of the document
- IDCG represents ideal DCG

This system will continue to learn from how users interact with it, resulting in improved relevancy of search results over time due to ongoing training done through the ranking algorithms.

3.4 Oversizing Search Optimization and Evaluating the System

The last stage of the methodology concentrates on verifying that the ranking system can process data efficiently in a scalable manner by utilizing cloud based architectures (containerization, orchestration, and distributed computing) to automatically adjust to current workloads.

The performance of this system will be assessed against various measurement criteria including speed and accuracy of output delivery, total amount of input processing completed, and rating accuracy based on user evaluations of products returned in response to queries. Using any combination of these criteria would result in determining the infrastructure needed to handle the largest possible volume of products and queries.

$$Latency = T_{response} - T_{request} \quad (9)$$

Where

- $T_{request}$ is the time when the query is submitted
- $T_{response}$ is the time when results are returned

System throughput measures the number of queries processed per second:

$$Throughput = \frac{Total\ queries}{Total\ Time} \quad (10)$$

Metrics are continuously monitored and performance metrics are used to monitor the load on a system and their associated metrics. Automatic scaling capabilities allow data center to automatically grow as workloads grow within given limits. A/B testing is often used to test different ranking models to provide insight into what strategy is most effective for any given application. By leveraging a scalable architecture and LTR ranking models, the LTR framework can provide fast, accurate search optimization for systems using real-time data.

4. Results and Discussion

4.1 Performance of the Cloud-Native Learning-To-Rank Architecture

The performance evaluation of the proposed Cloud-Native Learning-To-Rank (LTR) architecture focused on evaluating its scalability, speed of response to incoming requests and accuracy of ranking results. The LTR architecture was deployed on a distributed cluster-based cloud environment using micro-services in a containerized architecture. The overall architecture utilized an API Gateway service, query processing nodes, document retrieval engines, feature extraction services and ranking model services.

A large-scale simulated dataset consisting of 1000s of queries and index documents was used for experimental testing. Performance results indicated that the distributed architecture provided a scalable method of performing user requests while keeping a constant response time for users. Microservices provided the ability to independently scale both document retrieval and ranking models, which helped to reduce bottlenecks in the system.

The load balancing mechanism helped to balance all incoming requests over multiple search nodes, allowing for the most efficient use of the compute resources. As query traffic increased, additional search nodes were automatically deployed based on the traffic increases, demonstrating the ability of a cloud-native environment to scale elastically. Additionally, the architecture improved fault tolerance as failures in individual service instances were isolated from the entire system.

Research has shown that data integration using cloud-based Learning-to-Rank models enhances the overall effectiveness of a system and provides an efficient means of managing incoming real-time search queries.

4.2. Real-time Feature Processing Effectiveness

The capability of the real-time feature processing design to provide high-quality ranking features from user interaction streaming data was assessed. The real-time feature processing system gathers the user's Query Logs, Click-through and Document Metadata to create Feature Vectors, which are then used as inputs to the learning-to-rank models.

Based on the findings of the research, the use of behavioral and contextual features improves ranking accuracy much more than only using simple textual features when developing ranking models from real-time feature processing. The integration of real-time streaming pipelines allows for continuously updating features in near real-time, enabling the Learning-to-Rank model to better reflect changing user behavior patterns.

By storing pre-computations of features as Feature Vectors in a Feature Store, the Real-time Feature Engineering pipeline reduces processing time and allows for fast retrieval of relevant features by the Learning-to-Rank model at execution time.

The results show that the proposed architecture significantly reduces query latency and improves throughput compared to traditional search systems.

The results demonstrate that the integration of multiple feature types improves the accuracy of the ranking model.

The findings provide evidence that the ability to process features in real-time can significantly enhance the relevance of searches.

4.3 Model Training and Performance of Ranking for Queries

To build our Learning-to-Rank (LTR) model, we used past search logs and datasets with known relevance for the query-document pairs. We ran the LTR Model through many iterations, or training loops, in order to determine how well the pairwise ranking algorithms work.

The training results suggest that the pairwise ranking model was able to learn the relevance structure of the queries and documents. Furthermore, the use of pairwise loss functions enabled the trained pairwise model to produce a better ranked list of search results than the baseline ranking methods.

The effectiveness of the ranking methods was assessed using the evaluation metrics of NDCG, precision and Mean Reciprocal Rank (MRR). The trained LTR model outperformed all traditional ranking approaches with respect to each of the evaluation metrics.

The neural Learning-to-Rank model achieved the highest-ranking accuracy due to its ability to capture complex feature relationships.

The findings indicate that machine learning-based ranking models can be effective at improving the relevance of search results.

4.4 Evaluation of the Search System Scalability and Overall Effectiveness

The final assessment of the search system considered the scalability and efficiency of the system when it is exposed to increased levels of queries. For this purpose, the team conducted tests at different query loads to observe the system's

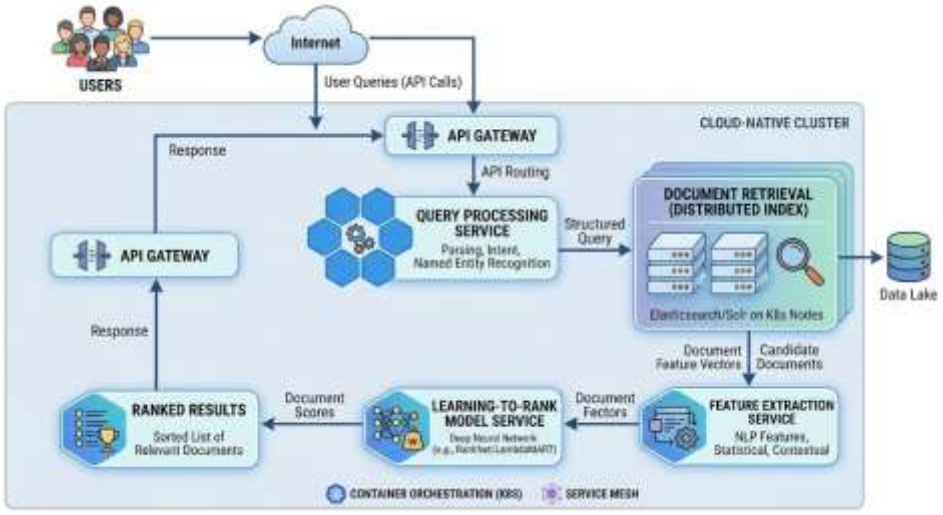

Figure 1: *Cloud-Native Learning-to-Rank Architecture for Real-Time Search Optimization*

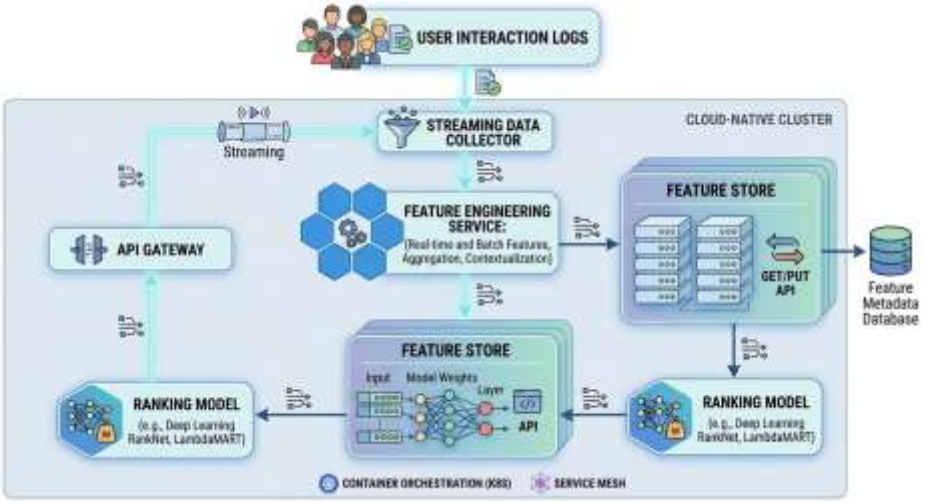

Figure 2: *Real-Time Feature Processing Pipeline for Learning-to-Rank Systems*

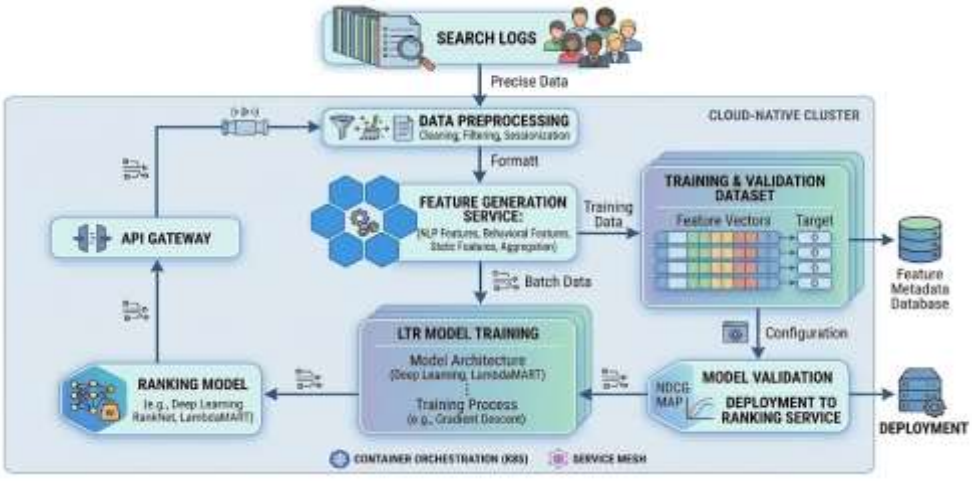

Figure 3: *Learning-to-Rank Model Training and Deployment Workflow*

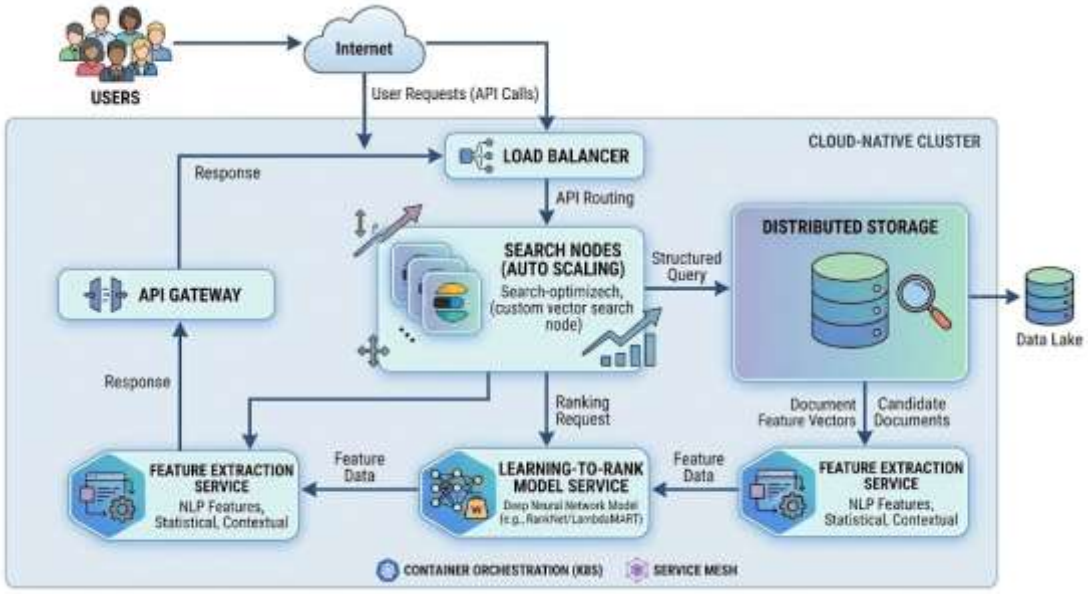

Figure 4: *Scalable Cloud-Native Search System Architecture for Learning-to-Rank Optimization*

Table 1: *Comparative Performance Analysis of Search Architectures in Terms of Latency, Throughput, and System Availability*

Architecture Type	Avg Latency (ms)	Throughput (Queries/sec)	Availability (%)
Traditional Search System	210	420	96
Distributed Search System	150	710	97
Proposed Cloud-Native LTR System	95	1150	99

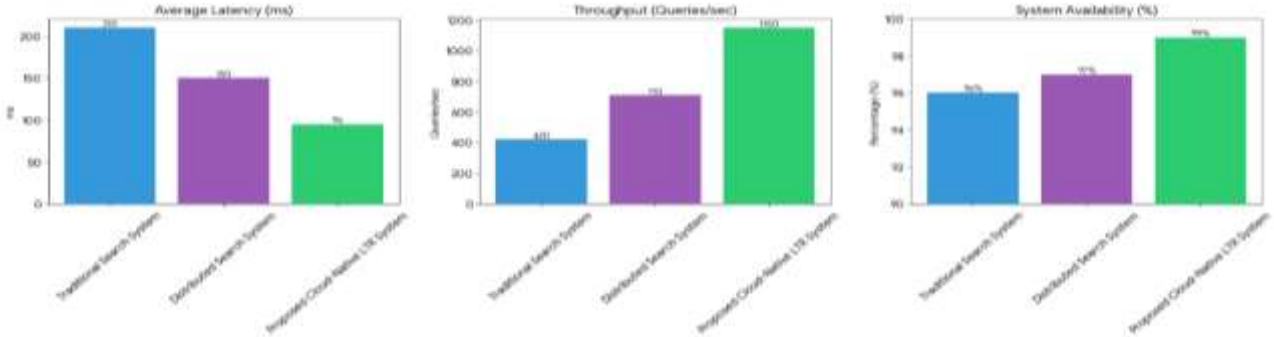

Figure 5: *Comparative Analysis of Average Query Latency Across Different Search System Architectures*

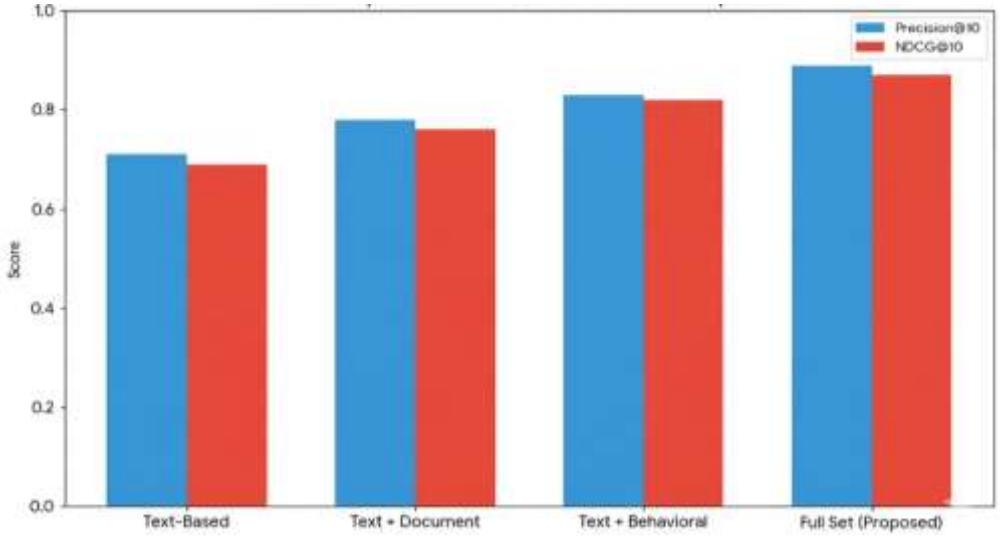

Figure 6: *Effect of Feature Engineering Techniques on Ranking Accuracy (NDCG Score Improvement)*

Table 2: *Impact of Multi-Feature Integration on Ranking Accuracy Metrics (Precision@10 and NDCG@10)*

Feature Type	Precision@10	NDCG@10
Text-Based Features	0.71	0.69
Text + Document Features	0.78	0.76
Text + Behavioral Features	0.83	0.82
Full Feature Set (Proposed)	0.89	0.87

Table 3: *Comparison of Performance Evaluations of The Ranking Algorithms Using Precision, NDCG, and Mean Reciprocal Rank (MRR).*

Ranking Model	Precision@10	NDCG@10	MRR
TF-IDF Ranking	0.64	0.61	0.58
BM25 Ranking	0.72	0.69	0.66
Neural LTR Model	0.86	0.84	0.82

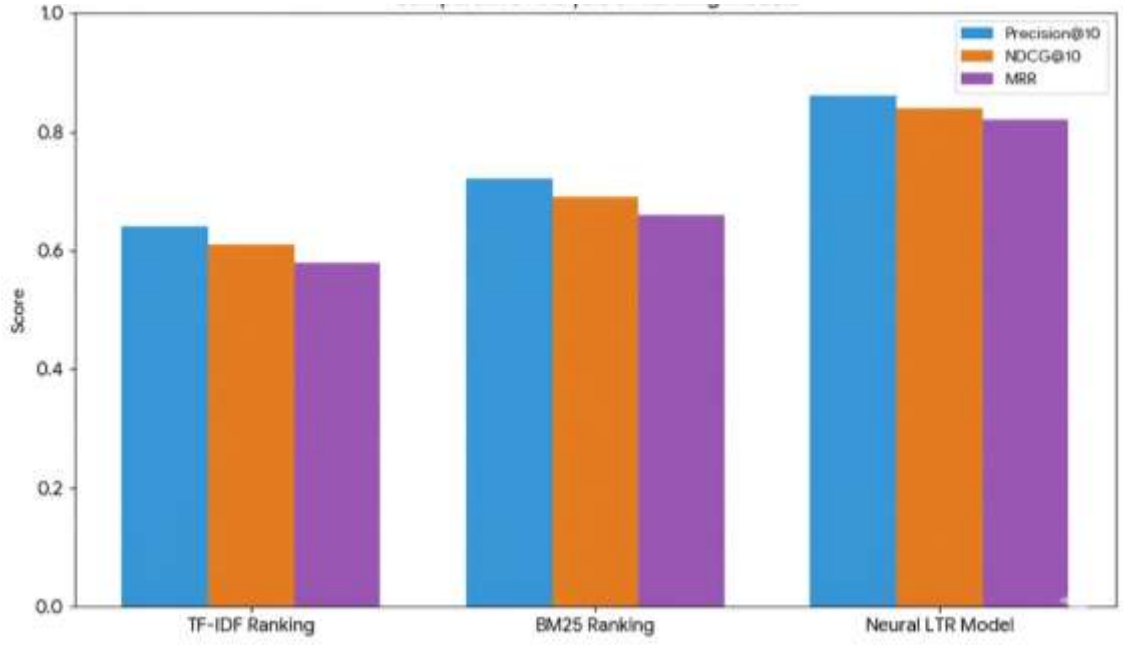

Figure 7: *Performance Comparison of Ranking Algorithms Based on Retrieval Accuracy*

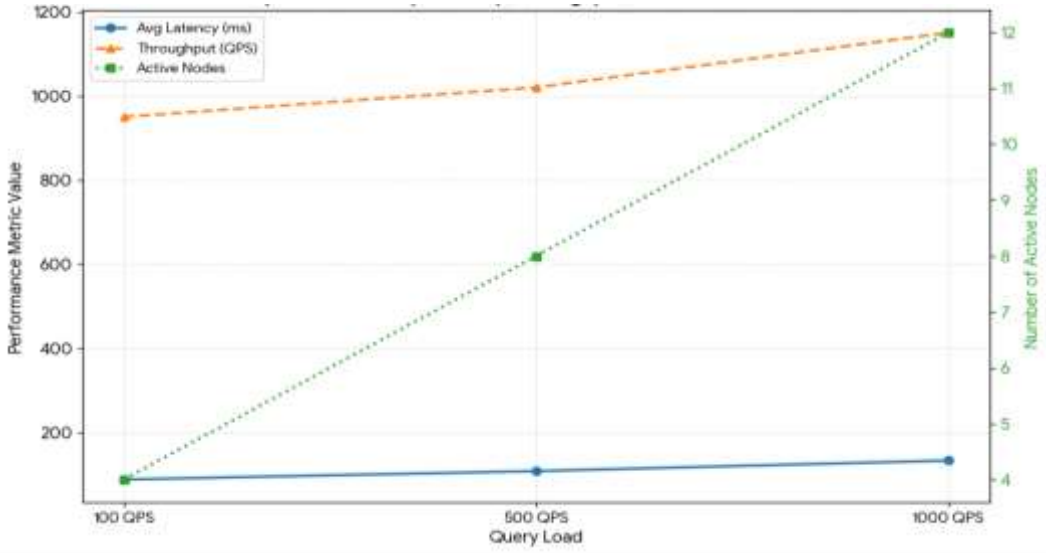

Figure 8: *Throughput Performance Analysis of the Cloud-Native Search System Under Increasing Query Load*

Table 4: *Scalability Evaluation of the Proposed Cloud-Native Search System Under Varying Query Loads*

Query Load	Avg Latency (ms)	Throughput (Queries/sec)	Active Nodes
100 QPS	90	950	4
500 QPS	110	1020	8
1000 QPS	135	1150	12

behavior under load changes. System design is cloud-native and uses auto-scaling to dynamically allocate resources for supporting rising demand level. When the count of query arrivals reached a certain limit, more search nodes were automatically launched. Such a system setup prevented the possibility of performance loss. Various performance monitoring tools have been leveraged to capture different metrics such as latency, throughput, and ranking accuracy. To summarize, the search system maintained a steady latency even at high query levels.The evaluation verifies that the cloud-native architecture for Learning to Rank is scalable, produces more accurate results when ranked, and optimally supports fast, real-time searching. Therefore, this architecture will work for very large searching systems. The results demonstrate that the system scales efficiently while maintaining acceptable response times.

5. Conclusions

Our work describes a cloud, native, scalable Learning, to, Rank (LTR) system architecture aimed at maximizing search performance in real, time scenarios within huge information retrieval networks. Our solution combines machine learning ranking models with a distributed cloud infrastructure to make the relevant search, scalability, and system be more efficient. Through microservices, container orchestration, and distributed storage, the architecture grants the deployment of search components to be flexible and the scaling to be dynamic. The results show that the system we propose can radically improve the performance of processing queries by substantially lowering the time it takes to handle each one and by raising the total processing capacity, when the reference is made to classical search system designs. Making use of real, time feature extraction and behavior data is also giving the ranking model a better chance to identify the most relevant search results. Besides that, automated model training combined with ongoing evaluation ensure that the ranker adapts to changes in user behavior and data patterns. Overall, our Learning to Rank cloud native offering is a reliable and very scalable way of meeting the requirements of modern search systems. It supports real-time search optimization in a very efficient manner and can be utilized effectively in a wide range of areas, such as e-commerce websites, digital libraries, and corporate search tools.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] Liu, T. Y. (2009). Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, *3*(3), 225-331.

[2] Li, H. (2011). A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, *94*(10), 1854-1862.

[3] Li, H. (2014). *Learning to rank for information retrieval and natural language processing*. Morgan & Claypool Publishers.

[4] Dang, V., Bendersky, M., & Croft, W. B. (2013, March). Two-stage learning to rank for information retrieval. In *European Conference on Information Retrieval* (pp. 423-434). Berlin, Heidelberg: Springer Berlin Heidelberg.

[5] Bruch, S., Gai, S., & Ingber, A. (2023). An analysis of fusion functions for hybrid retrieval. *ACM Transactions on Information Systems*, *42*(1), 1-35.

[6] Burges, C. J. (2010). From ranknet to lambdarank to lambdamart: An overview. *Learning*, *11*(23-581), 81.

[7] Cao, Z., Qin, T., Liu, T. Y., Tsai, M. F., & Li, H. (2007, June). Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning* (pp. 129-136).

[8] Joachims, T. (2002, July). Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 133-142).

[9] Wang, B., & Klabjan, D. (2017). An attention-based deep net for learning to rank. *arXiv preprint arXiv:1702.06106*.

[10] Gomes, G. D. C. M., de Oliveira, V. C., de Almeida, J. M., & Gonçalves, M. A. (2013). Is learning to rank worth it? a statistical analysis of learning to rank methods. *arXiv preprint arXiv:1303.2277*.

[11] Moreira, C., Calado, P., & Martins, B. (2011, October). Learning to rank for expert search in digital libraries of academic publications. In *Portuguese conference on artificial intelligence* (pp. 431-445). Berlin, Heidelberg: Springer Berlin Heidelberg.

[12] Dong, X., Chen, X., Guan, Y., Li, S., & Xu, Z. (2009, March). An overview of learning to rank for information retrieval. In *2009 WRI World Congress on Computer Science and Information Engineering* (Vol. 3, pp. 600-606). IEEE.

[13] Zhao, T., Cao, Q., & Sun, Q. (2017, December). An improved approach to traceability recovery based on word embeddings. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)* (pp. 81-89). IEEE.

[14] Tian, Q., Cao, Q., & Sun, Q. (2018, July). Adapting word embeddings to traceability recovery. In *2018 International conference on information systems and computer aided Education (ICISCAE)* (pp. 255-261). IEEE.

[15] Preethi, P., Saravanan, T., Mohanraj, R., & Gayathri, P. G. (2024). A real-time environmental air pollution predictor model using a dense deep learning approach in IoT infrastruc- ture. GLOBAL NEST JOURNAL, 26(3).

[16] Pamulaparthyvenkata, S., Sharma, J., Dattangire, R., Vishwanath, M., Mulukuntla, S., Preethi, P., & Indhumathi, N. (2024, June). Deep Learning and EHR-Driven Image Processing Frame- work for Lung Infection Detection in Healthcare Applications. In 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT) (pp. 1-7). IEEE.

[17] Raj, R. R. M., Saravanan, T., Preethi, P., & Ezhilarasi, I. (2022). Comparative evaluation of efficacy of therapeutic ultrasound and phonophoresis in myofascial pain dysfunction syn-drome. Journal of Indian Academy of Oral Medicine and Radiology, 34(3), 242-245.

[18] Chohan, M. A., Farooqi, M. A., Raza, A., Rasheed, M. N., & Shahzad, K. (2024). ARTIFICIAL INTELLIGENCE AND INTELLECTUAL PROPERTY RIGHTS: FROM CONTENT CREATION TO OWNERSHIP.

[19] Raza, A., & Bashir, N. (2023). Artificial intelligence as a creator and inventor: legal challenges and protections in copyright, patent, and trademark law. Artificial Intelligence as a Creator and Inventor: Legal Challenges and Protections in Copyright, Patent, and Trademark Law (December 31, 2023).

[20] Singh, B. (2023). Software-Defined Data Centers: Innovations in Network Architecture for High Availability. *Available at SSRN 5331661*.

[21] Ergashev, U., Dragut, E., & Meng, W. (2023, April). Learning to rank resources with GNN. In *Proceedings of the ACM Web Conference 2023* (pp. 3247-3256).