Cost-Efficiency Trade-offs for Neural Cascade Rankers in Web Search

Ekaterina Trimbach*
trimbachkatya@gmail.com
EPFL
Lausanne, Switzerland

Badr Abdallaoui
badredda@amazon.com
Amazon
Luxembourg

Paul Missault
pmissaul@amazon.com
Amazon
Luxembourg

Abstract

Web search engines process billions of queries daily, making the balance between computational efficiency and ranking quality crucial. While neural ranking models have shown impressive performance, their computational costs, particularly in feature extraction, pose significant challenges for large-scale deployment. This paper investigates how different configurations of feature selection and document filtering in neural cascade ranking systems influence the trade-off between computational cost and ranking performance.

We propose a two-stage neural cascade architecture where both stages utilize Multi-Layer Perceptrons (MLPs). The first stage processes all documents using a reduced feature set, while the second stage applies a more sophisticated model to only the top-ranked documents. This design allows us to systematically explore the impact of feature selection and document filtering on both computational cost and ranking performance.

Through extensive experiments on three large-scale datasets (Yahoo, Istella, and Microsoft MSLR-WEB30K), we demonstrate significant opportunities for cost reduction with minimal impact on ranking quality. Our results show that optimal cascade configurations can achieve cost reductions in feature extraction of up to 40.37% on the Yahoo dataset and 16% on the MSLR-WEB30K dataset while maintaining nearly identical NDCG@10. Furthermore, we identify clear patterns of diminishing returns in ranking performance as computational resources increase, providing valuable insights for developing resource-efficient ranking systems in large-scale web search environments.

CCS Concepts

• **Information systems** → **Learning to rank**.

Keywords

Neural Cascade Ranking, Web Search, Learning to Rank, Cost-Efficiency Trade-offs, NDCG, Large-scale Information Retrieval

*Work carried out during an internship with Amazon.

WWW Companion '25, April 28-May 2, 2025, Sydney, NSW, Australia

ACM ISBN 979-8-4007-1331-6/2025/04
https://doi.org/10.1145/3701716.3717577

ACM Reference Format:
Ekaterina Trimbach*, Badr Abdallaoui, and Paul Missault. 2025. Cost-Efficiency Trade-offs for Neural Cascade Rankers in Web Search. In *Proceedings of the International Workshop on Resource-Efficient Learning for the Web (REL4Web '25), May 2025, Sydney, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3701716.3717577

1 Introduction

Web search engines face a monumental challenge: they need to handle billions of searches every day while providing accurate results [3]. This requires ranking models that balance speed and accuracy effectively. Even small improvements in efficiency can lead to substantial resource savings at scale [25]. The challenge of balancing efficiency and effectiveness in learning to rank systems has been well-documented in recent literature [18, 19].

Cascade learning has emerged as a promising approach to address this challenge in large-scale web search ranking [16, 24]. This technique employs a multi-stage architecture, where each successive stage applies increasingly complex models to a progressively narrowed subset of documents. The goal is to reduce overall computational costs while maintaining high ranking quality [20].

Recent surveys have highlighted the growing importance of deep learning models in information retrieval [12]. However, the trend towards increasingly complex models to chase marginal gains in accuracy often comes at the cost of significantly increased computational resources and diminishing returns on investment. Moreover, the environmental impact of large-scale machine learning models has become a growing concern [23], further motivating our focus on efficiency.

In this paper, we challenge this maximalist mindset and instead focus on the reverse problem: how can we substantially reduce computational costs while maintaining a level of performance that is practically indistinguishable from state-of-the-art models? Our research shifts the emphasis from pursuing tiny improvements in accuracy to finding an optimal balance between cost-efficiency and ranking quality, a trade-off that has been recognized as crucial in the field [19].

We argue that in many real-world scenarios, the benefits of slight increases in ranking accuracy are outweighed by the substantial computational resources required to achieve them. By focusing on the cost side of the equation, we aim to develop models that are not only highly effective but also economically viable and environmentally sustainable for large-scale deployment.

This approach is particularly relevant in the context of web search engines, where even small reductions in computational cost can translate to significant savings when scaled to billions of queries. Our goal is to demonstrate that it's possible to achieve substantial

cost savings without noticeably hurting performance, thereby providing a more holistic solution to the challenges faced by modern search systems.

In this paper, we present a novel neural-only cascade architecture for web search ranking. Our work builds on recent advancements in efficient neural ranking models [13], while specifically focusing on the cascade architecture. Our approach consists of two stages, both using the same family of models: straightforward, fully-connected Multi-Layer Perceptrons (MLPs). This design allows for optimization within a unified framework. The key distinction between the stages lies in their focus:

(1) The first stage (L_1) processes all documents using a reduced set of features, prioritizing efficiency by minimizing feature extraction costs.
(2) The second stage (L_2) applies a more sophisticated model to a subset of documents selected by a top-k operator, leveraging the full feature set to refine ranking quality.

A central focus of our research is investigating the trade-off between computational cost and ranking performance, with a particular emphasis on the cost of feature extraction for query-document pairs.

Our main contributions in this paper are as follows:

(1) We propose and evaluate a novel neural cascade architecture for web search ranking, where both stages utilize Multi-Layer Perceptrons (MLPs). This unified framework enables efficient optimization of feature extraction costs while maintaining ranking quality.
(2) We provide a comprehensive analysis of the cost-performance trade-offs in our cascade model, focusing specifically on the impact of feature extraction costs on overall system efficiency.

To investigate our approach, we conducted experiments using three large-scale web search datasets: Yahoo [7], Istella [10], and Microsoft MSLR-WEB30K [22]. We explored various combinations of feature subsets for (L_1) and different values of k (the number of documents passed from (L_1) to (L_2)), analyzing their impact on both ranking quality (measured by NDCG@10) and computational cost.

Our work extends previous research on cascade models [8, 11] by focusing exclusively on neural network architectures and providing a comprehensive comparison with single-stage neural rankers. This approach allows us to leverage the flexibility of neural networks while addressing the efficiency challenges in large-scale web search ranking.

The remainder of this paper is organized as follows: We first review related work in cascade learning and neural ranking models. We then describe our neural cascade architecture and experimental methodology in detail. Next, we present our results, including our analysis of efficiency trade-offs. Finally, we discuss the implications of our findings and suggest directions for future research.

2 Related Work

Cascade ranking models have evolved significantly in information retrieval, progressing from traditional decision tree-based approaches to neural network-based models. Early work in this field introduced tree-based cascade models using boosting algorithms [24], which considered efficiency but were limited in end-to-end optimization. Subsequent research advanced this concept with globally optimized cascade architectures for tree-based models, importantly including cost analysis and addressing the effectiveness-efficiency trade-off [11].

The adoption of neural networks has opened new avenues for cascade ranking. Recent developments include an Adaptive Neural Ranking Framework with a novel loss function [27]. Addressing the often-overlooked problem of optimal result list truncation, Choppy was introduced as a Transformer-based model [1]. This approach focuses on determining the optimal number of results to return, balancing overall relevance with the user cost of processing more results. Choppy uses a multi-head attention mechanism to directly optimize any user-defined IR metric, improving upon previous state-of-the-art methods in ranked list truncation. Theoretical insights on two-stage ranking systems have also emerged, emphasizing the importance of joint optimization in retrieval and reranking stages [28].

However, while tree-based approaches have considered efficiency, and some neural models have addressed specific aspects like list truncation, there remains a gap in the comprehensive analysis of cost-efficiency trade-offs in neural cascade ranking models. Our work aims to fill this gap by proposing a novel neural-only cascade architecture that allows for end-to-end optimization. We focus on quantifying cost gains and examining the relationship between performance degradation and efficiency improvements in neural models. By investigating the impact of feature selection and document filtering on both computational efficiency and ranking quality, we provide a holistic understanding of the trade-offs in designing neural cascade ranking systems. This analysis is crucial for developing practical, resource-efficient models for large-scale web search environments, extending the efficiency considerations from tree-based models to the neural domain.

3 Neural Ranking Cascade Model

3.1 Learning-to-Rank background

Consider a search query Q that retrieves n documents. Each pair of query-document (Q, D_i) is represented with a one-dimensional feature vector X_i of length l_x. Typically, the goal of Learning-to-Rank is to learn a scoring function $s : \mathbb{R}^{l_x} \longrightarrow \mathbb{R}$ that produces a vector of scores $s(X_i)$. The individual documents i are then ranked in the descending order of $s_i = s(X_i)$ scores. The quality of the ranking is assessed by a utility function U evaluated at a depth k, where U captures the relevance or usefulness of the top-k ranked documents to the user.

In this study, we focus on the Normalized Discounted Cumulative Gain (NDCG) as our primary evaluation metric [14]:

$$\text{NDCG@}k = \frac{1}{Z_k} \sum_{i=1}^{k} \frac{2^{y_i} - 1}{\log_2(i+1)} \tag{1}$$

where:

- y_i is the relevance label of the i-th document. This is typically a graded scale, where higher values indicate greater relevance.

- Z_k is a normalization constant, calculated as the DCG of the ideal ranking for that query up to position k. The ideal ranking is obtained by ordering documents in descending order of their relevance labels y_i. This ensures that the perfect ranking for each query will have an NDCG of 1.0, allowing for meaningful comparisons across queries with different numbers of relevant documents.

Documents are ordered according to the scores s_i produced by our ranking model. The use of $2^{y_i} - 1$ as the gain function gives exponentially increasing importance to higher relevance levels, which aligns with typical user behavior in web search scenarios.

NDCG is particularly suitable for our study as it takes into account both the relevance and the position of documents in the ranked list, uses a graded relevance scale, and discounts the contribution of lower-ranked documents. This aligns well with user behavior in web search scenarios where top results are more likely to be examined.

It's important to note that while we focus on NDCG in this study, the principles and methodologies we develop are generalizable to other ranking metrics. The trade-offs between computational cost and ranking quality that we explore are fundamental to cascade learning and are not specific to NDCG. Researchers and practitioners could apply our approach using other metrics relevant to their specific use case or domain.

In Neural LTR approaches, the function s is a parameterized neural network optimized using a loss function to ensure that the output scores s_i align well with the true relevance y_i of the documents, producing a ranking that matches the desired order. This comparison is either made pointwise (i.e. $s(X_i)$ with y_i), pairwise (i.e. $(s(X_i), s(X_j))$ with (y_i, y_j) for $i, j \in I$), or listwise (i.e. $s(X_i)_{i\in I}$ with $(y_i)_{i\in I}$) [16]. Common loss functions include Mean Squared Error for pointwise approaches [9], RankNet for pairwise approaches [5], and ListNet for listwise approaches [6]. Recent trends focus on directly optimizing ranking metrics like NDCG through smooth approximations [21] or lambda gradients [26]. In this work, we do not assume any specific type of comparison. However, we assume that the scoring happens in a pointwise manner. In other words, at inference time, for a query Q, the score s_i depends only on the feature values of (Q, d_i), and not on other (Q, d_j) for $j \neq i$.

3.2 Neural Cascade

Instead of one neural scoring function, we consider two scoring functions chained together by a top-k operator. Figure 1 illustrates the proposed architecture of a neural cascade.

Assume the training data consists of N queries, and each query consists of a variable number of documents. Without loss of generality, we assume that all queries retrieve of a fixed number of documents n.

Assume that a given query retrieves n documents $\mathcal{D} = \{D_1, \ldots, D_n\}$. Each document D_i is scored independently by Neural Ranker 1 (L_1), and gets the output scores $S_1 = \{S_{11}, \ldots, S_{1n}\}$. The n documents are then filtered out by the top-k operator to keep only the documents whose scores fall into the top-k scores. We then get a new set of documents $\mathcal{D}' = \{D'_1, \ldots, D'_k\} \subseteq \mathcal{D}$. We typically consider $k \ll n$. The new set passes through the second Neural Ranker 2 (L_2) and gets scores $S_2 = \{S_{21}, \ldots, S_{2k}\}$. For simplicity, we make the following assumptions:

- The relative order produced by L_1 is dropped and not used by the second model.
- The documents relevance labels used for L_1 and L_2 are identical.
- The input feature values for L_1 and L_2 are identical.

For Neural Ranker L_1, we denote the output set $Y_1 = \{y_{11}, \ldots, y_{1n}\}$, and for Neural Ranker L_2, $Y_2 = \{y_{21}, \ldots, y_{2k}\}$, $\subseteq Y_1$, for which the labels correspond to the top-k selected documents. Scores are then computed by minimizing the loss.

3.3 Loss Optimization Strategies

In our cascade architecture, we have two loss functions: $\mathcal{L}_1$ for the first stage (L_1) and $\mathcal{L}_2$ for the second stage (L_2). We explored two main strategies for optimizing these losses: simultaneous (joint) optimization and independent optimization.

3.3.1 Simultaneous Optimization. In the simultaneous optimization approach, we combine the losses of both stages into a single objective function:

$$\mathcal{L} = \mathcal{L}_1(S_1, Y_1) + \mathcal{L}_2(S_2, Y_2)$$

This combined loss is then optimized in a single training process. The idea behind this approach is to allow the two stages to co-adapt, potentially leading to a more cohesive and effective overall ranking system.

To handle the non-differentiability of the top-k operator between L_1 and L_2, we employ the Straight-Through Estimator method [2]:

- In the forward pass, we apply the top-k operation to select documents to be passed to L_2.
- In the backward pass, we treat the top-k operation as an identity function, allowing gradients to flow from L_2 to L_1. The parameters of s_2 are first updated. We then let the gradient pass through the top-k as it is, i.e. it acts as an identity function during backward pass which replicates the gradient of the above layer and passes it as it is to the below layer without any modification. We pass the gradient through the top-k operation as if it were a continuous function. Then the backward pass continues through s_1.

3.3.2 Independent Optimization. In the Independent optimization approach, we optimize $\mathcal{L}_1$ and $\mathcal{L}_2$ independently:

We independently train the L_1 model by optimizing $\mathcal{L}_1(S_1, Y_1)$ and we train the L_2 model by optimizing $\mathcal{L}_2(S_2, Y_2)$ on the same training set. This approach is simpler to implement and allows each stage to be optimized without interference from the other stage.

3.3.3 Comparison of Approaches. Interestingly, our experiments did not show a clear superiority of the simultaneous optimization approach over the independent approach. Despite the theoretical appeal of joint optimization, we found that:

The independent approach performed comparably with the simultaneous approach in terms of the final ranking quality. The independent approach was often more stable during training and easier to tune. In some cases, the independent approach even slightly outperformed the simultaneous approach. We hypothesize that this

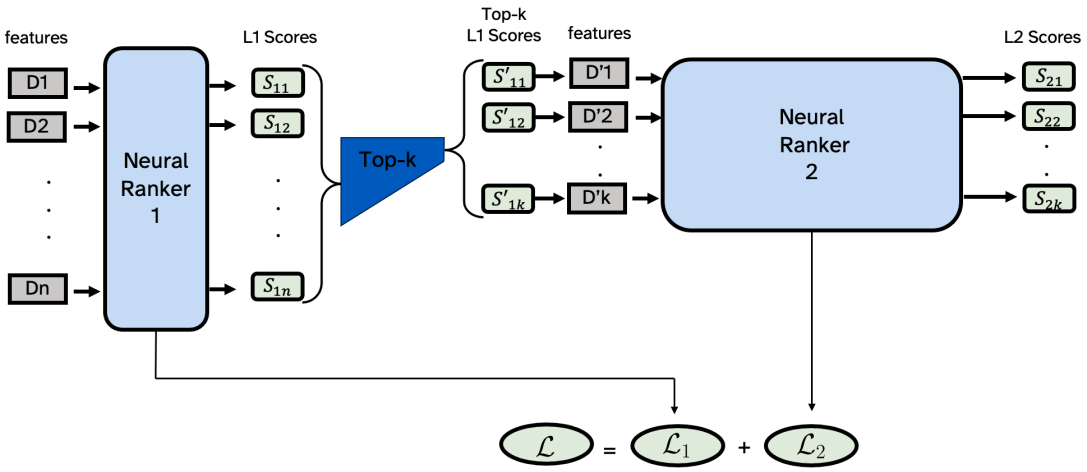

Figure 1: Neural Cascade illustration

lack of improvement from joint optimization could be due to several factors:

The simplicity of our datasets may not necessitate the complex co-adaptation that joint optimization enables. The top-k operation may create a bottleneck that limits the effectiveness of gradient flow between L_2 and L_1. The independent optimization of each stage may allow for better specialization to their respective tasks (recall for L_1, precision for L_2). Given these findings, we primarily focus on the independent optimization approach in our subsequent experiments and analysis. This choice simplifies our training process without sacrificing performance and allows us to more easily explore different configurations of feature subsets and k values for our cost-efficiency trade-offs.

3.4 Cost Definition

For our cost model, we adopt a framework that focuses exclusively on feature extraction costs in the context of cascade learning for web search ranking [8, 11]. This framework does not account for the cost of prediction execution for the learned models, as this cost is considered negligible compared to feature extraction. This assumption is particularly valid in our case, as the neural networks used in both stages of the cascade are relatively small, making their inference costs minimal.

For our two-stage cascade system, the computational cost for the given query is calculated as:

$$C = \sum_{i=1}^{n} \left(\sum_{d \in D} \text{cost}(f_i, d) \right) + \sum_{i=n+1}^{N} \left(\sum_{d \in \text{top-k}} \text{cost}(f_i, d) \right) \quad (2)$$

where:

- D is the set of all documents for a given query
- n is the number of features used by the first stage (L_1) of the cascade
- N is the total number of available features
- k is the number of documents passed from L_1 to L_2
- $\text{cost}(f_i, d)$ represents the computational cost of extracting feature f_i for document d

This cost function consists of two main components:

(1) The cost of extracting the first n features for all documents
(2) The cost of extracting the remaining $N - n$ features for only the top-k documents

This formulation assumes that once a feature is extracted, any later stage can use it without additional cost. By adopting this established cost model, we can systematically evaluate the trade-offs between computational efficiency and ranking effectiveness in our cascade learning approach. Our goal is to identify configurations that minimize cost while maintaining high ranking quality, as measured by metrics such as NDCG@10.

By explicitly modeling these costs, we can make informed decisions about the trade-offs between computational efficiency and ranking effectiveness in our cascade learning approach. Our goal is to find configurations that minimize cost while maintaining high ranking quality, as measured by metrics such as NDCG@10.

4 Experiments and Evaluation

Our experimental evaluation investigates how different configurations of feature selection in the first stage (L_1) and document filtering between stages (L_1 and L_2) impact the trade-off between computational cost and ranking performance. Specifically, we explore how reducing the number of features in L_1 and limiting the number of documents passed to L_2 affect both efficiency and ranking quality. This analysis provides insights into optimizing cascade architectures for large-scale web search systems.

4.1 Evaluation Metrics

In our experiments, we use NDCG@m as the primary metric for evaluating ranking performance, where m specifies the cutoff point for evaluation (e.g., NDCG@10). While the choice of $m = 10$ is common in web search evaluation, it is not fixed and can be tailored to the needs of specific applications. A key consideration is that

the number of documents passed from the first stage (L_1) to the second stage (L_2), denoted as k, must satisfy the condition $k \geq m$. This ensures that L_2 receives enough documents to perform meaningful ranking refinements. Although our experiments focus on NDCG@10, the methodology is flexible and can be applied to other values of m, as long as the constraint $k \geq m$ is maintained.

4.2 Datasets

We conducted experiments on three widely used learning-to-rank datasets: Yahoo [7], Istella [10], and Microsoft MSLR-WEB30K [22]. Table 1 summarizes the key characteristics of these datasets. The depth and relevance distributions for each dataset are illustrated in Figures 5, 6, and 7 in the Appendix.

Table 1: Dataset Characteristics

Dataset	Query-Doc. Pairs	Features	Queries	Relevance Scale
Yahoo	700,220	700	28,786	0-4
MSLR-WEB30K	3,700,000	136	30,000	0-4
Istella-S LETOR	3,408,630	220	10,000	0-4

These datasets present diverse characteristics in terms of feature distributions, query depths, and relevance patterns, allowing us to validate our cascade approach across different web search scenarios. The variety in dataset characteristics significantly influences the optimal configuration of our cascade model, particularly in determining appropriate values for the number of features (n) to be used for L_1 and number of documents (k) to pass between stages.

All three datasets use a 5-point relevance scale (0-4), where 0 typically represents irrelevant documents and 4 represents highly relevant documents. This consistent relevance scale across datasets allows for meaningful comparisons of our model's performance. Also, for training all datasets divided on test, train and validation by different queries with 20% of queries devoted to test.

Feature Cost Assignment. For feature costs, we use the labeled costs provided in the Yahoo! dataset, which range from 1 to 200. For the MSLR-WEB30K and Istella datasets, which do not provide explicit cost information, we assign relative cost labels based on the Yahoo! S1 cost set and our domain expertise, following established methodologies [8, 11]. This ensures a consistent framework for evaluating the computational implications of different feature sets across all datasets.

4.3 Experimental Setup and Methodology

As previously discussed, a cascade approach provides an effective solution to the cost-efficiency trade-off in ranking models. For such an analysis, it is important to select the L_1 and L_2 models with sufficient difference between them to allow for meaningful comparisons. In our experiments, we used a consistent architecture across all three datasets (Yahoo, MSLR-WEB30K, and Istella) to ensure fair comparisons and reproducibility.

For the first stage (L_1), we employed a neural network with one hidden layer of 16 units and a dropout rate of 0.1. This lightweight architecture ensures efficient processing of all documents while using a reduced feature set. For the second stage (L_2), we used larger neural network with one hidden layer of 256 units and a dropout rate of 0.5. This architecture allows for more refined ranking of the top-k documents selected by L_1. In both stages, the input dimension x corresponds to the number of features used (e.g., 700 for Yahoo, 136 for MSLR-WEB30K, and 220 for Istella), and the output is a single relevance score ranging from 0 to 1.

For training both components of the cascade, we employ the ListNet [6] loss function, which is based on the softmax cross entropy. This loss function is particularly well-suited for listwise learning-to-rank approaches. Following the formulation in [4], we detail the construction of this loss function.

To compute the loss, the relevance labels y_i and the model scores s_i of document D_i are projected onto the probability simplex to form the ground-truth distribution P_y and the score distribution P_s as follows:

$$P_y(D_i) = \frac{y_i}{\sum_{j=1}^{n} y_j}, \quad P_s(D_i) = \frac{e^{s_i}}{\sum_{j=1}^{n} e^{s_j}} \tag{3}$$

These probabilities can be interpreted as encoding the likelihood of document D_i appearing at the top of the ranked list, referred to as the "top one" probability.

Given these two distributions, the loss is defined as their distance measured by cross entropy:

$$\mathcal{L} = H(P_y, P_s) \triangleq -\sum_{i=1}^{n} P_y(D_i) \log P_s(D_i) \tag{4}$$

This can be expanded to:

$$\mathcal{L} = -\sum_{i=1}^{n} y_i \log \left(\frac{e^{s_i}}{\sum_{j=1}^{n} e^{s_j}} \right) \tag{5}$$

This formulation allows the model to learn to rank documents by aligning its scores with the true relevance order, rather than predicting absolute relevance values. The use of softmax ensures that the model outputs are transformed into a proper probability distribution, while the cross-entropy loss encourages the predicted distribution to match the ground truth distribution.

4.4 Feature Selection and Cost Analysis

To define the subset of features for the first neural network (L_1), we employed a Random Forest classifier [15, 17] to calculate the importance of each feature. Feature importance scores were computed using the training dataset, ranking features based on their contribution to model performance. This approach allowed us to select the most informative features for L_1, enabling dimensionality reduction without significantly impacting ranking quality.

Next, for each top-n subset of features, where n belongs to $\{10, 20, 30, 40, \ldots, \text{total number of features}\}$, we trained an L_1 model using only the selected top-n features.

Subsequently, we evaluated the cascade performance for k values within $\{10, 11, 12, \ldots, \text{max query depth}\}$. Finally, we performed an exhaustive evaluation of all combinations of n and k across the grid defined by $\{10, 20, 30, 40, \ldots, \text{total number of features}\} \times \{10, 11, 12, \ldots, \text{maximum query depth}\}$.

5 Results and Analysis

In this section, we compare the performance of the cascade architecture (illustrated in Figure 1) against a single large neural network with the same architecture as the second stage (L_2) of the cascade. This comparison is designed to showcase the cost benefits of the cascade approach while maintaining comparable ranking performance. Specifically, the single large neural network serves as our baseline, representing a traditional, non-cascade ranking system that processes all documents using the full feature set. Since the neural networks used in both the cascade and the single large neural network are relatively small, we neglect the cost of prediction for both systems, as it is negligible compared to the cost of feature extraction. This means the cost comparison focuses exclusively on feature extraction, which is the dominant factor in computational expense.

For every pair of parameters (n, k), we evaluate the NDCG@10 of the cascade architecture and calculate the feature extraction cost using Equation 2. Here, n represents the number of features used in the first stage (L_1), and k represents the number of documents passed from L_1 to L_2. The results are visualized in Figures 2, 3, and 4, where each point corresponds to a specific (n, k) configuration. These graphs illustrate the trade-off between feature extraction cost and NDCG@10 performance, comparing the cascade architecture against the baseline performance of the single large neural network.

Additionally, we highlight several optimal (n, k) configurations in Tables 2, 3, and 4. The baseline performance, corresponding to the single large neural network, is marked with a star (∗). These tables demonstrate that the cascade architecture achieves results close to the baseline in terms of NDCG@10 while significantly reducing computational costs.

Tables 2, 3, and 4 present detailed results for each dataset, showing various configurations and their corresponding performance metrics. Notably, each dataset exhibits different optimal configurations, suggesting that the cascade approach can adapt to specific dataset characteristics while maintaining efficiency gains.

Table 2: Results comparing various configurations for the Yahoo dataset with the L_2 baseline (∗).

n	k	NDCG@10	Cost Reduction (%)	NDCG Drop (%)
50	17	0.770 250	40.3746	0.28
70	16	0.770 217	39.9480	0.28
130	13	0.770 544	37.9608	0.24
40	20	0.770 031	35.5336	0.31
30	23	0.770 300	30.4346	0.27
200	12	0.770 337	30.3693	0.27
240	12	0.771 161	26.5741	0.16
700*	–	0.772 409	0.0000	0.00

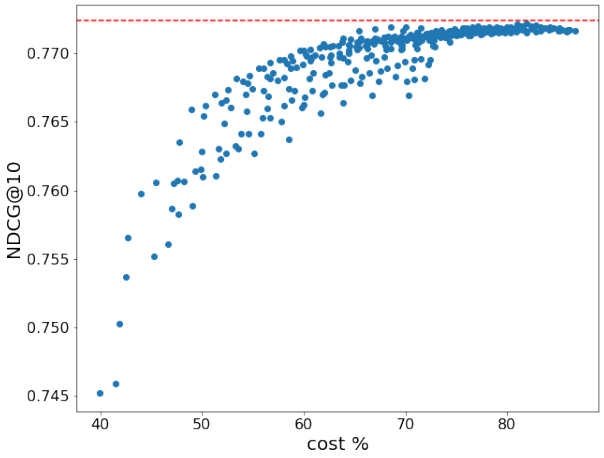

Figure 2: Dependency between cost and NDCG for all pairs of n and k for Yahoo dataset. The red line represents the baseline performance of L_2 model.

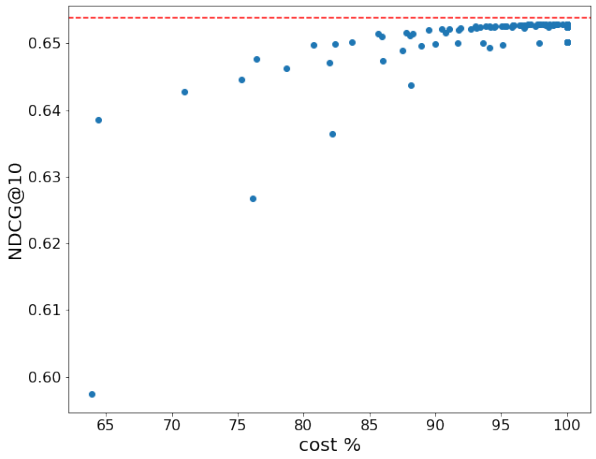

Figure 3: Dependency between cost and NDCG for all pairs of n and k for MSLR-WEB30K dataset.The red line represents the baseline performance of L_2 model.

6 Conclusion and Future Work

Our neural cascade architecture demonstrates significant potential for balancing computational efficiency and ranking quality in large-scale web search systems. By systematically exploring the trade-offs between feature selection in the first stage (L_1) and document filtering between stages (L_1 and L_2), we have shown that substantial cost reductions can be achieved with minimal impact on ranking performance. Our experiments across three datasets (Yahoo, MSLR-WEB30K, and Istella) reveal that optimal configurations

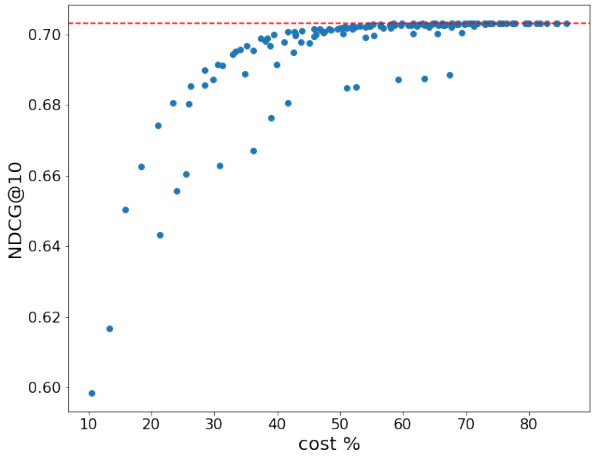

Figure 4: Dependency between cost and NDCG for all pairs of n and k for Istella dataset.The red line represents the baseline performance of L_2 model.

Table 3: Results comparing various configurations for the MSLR-WEB30K dataset with the L_2 baseline ($*$).

n	k	NDCG@10	Cost Reduction (%)	NDCG Drop (%)
30	12	0.650 135	16.3234	0.57
10	15	0.652 457	5.8094	0.21
120	13	0.652 656	2.4093	0.18
70	14	0.652 787	4.0899	0.16
136*	–	0.653 840	0.0000	0.00

Table 4: Results comparing various configurations for the Istella-S LETOR dataset with the L_2 baseline ($*$).

n	k	NDCG@10	Cost Reduction (%)	NDCG Drop (%)
30	30	0.700 713	58.3652	0.34
40	30	0.700 659	57.3135	0.35
50	25	0.700 107	57.0731	0.43
20	35	0.700 929	56.1218	0.31
60	25	0.700 533	52.6176	0.37
218*	–	0.703 130	0.0000	0.00

can reduce feature extraction costs by up to 40.37% while maintaining nearly identical NDCG@10 scores. These findings highlight the effectiveness of cascade architectures in resource-constrained environments.

Future work could build on these results by exploring dynamic cascade configurations that adapt to query complexity and user intent, as well as cost-aware feature selection strategies that optimize the trade-off between feature importance and extraction cost. Additionally, investigating end-to-end differentiable cascade architectures could further enhance the flexibility and performance of neural ranking systems. By addressing these directions, we aim to develop even more efficient and adaptable ranking models for real-world web search applications.

References

[1] Dara Bahri, Yi Tay, Che Zheng, Donald Metzler, and Andrew Tomkins. 2020. Choppy: Cut transformer for ranked list truncation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1513–1516.

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[3] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems* 30, 1-7 (1998), 107–117.

[4] Sebastian Bruch, Masrour Zoghi, and Thorsten Joachims. 2020. An Analysis of the Softmax Cross-Entropy Loss for Learning-to-Rank with Binary Relevance. *arXiv preprint arXiv:2006.12186* (2020).

[5] Christopher J.C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to Rank Using Gradient Descent. In *Proceedings of the 22nd International Conference on Machine Learning*. ACM, 89–96.

[6] Zhe Cao, Tao Qin, Tie-Yan Liu, Jinfeng 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.

[7] Olivier Chapelle, Yi Chang, and Tie-Yan Liu. 2011. The Yahoo! learning to rank challenge. *Journal of Machine Learning Research* 14 (2011), 1–24.

[8] Ruey-Cheng Chen, Leif Azzopardi, and Falk Scholer. 2017. Efficient cost-aware cascade ranking in multi-stage retrieval. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 445–454.

[9] David Cossock and Tong Zhang. 2006. Subset ranking using regression. In *International Conference on Computational Learning Theory*. Springer, 605–619.

[10] 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.

[11] Luke Gallagher, Ruey-Cheng Chen, Roi Blanco, and J Shane Culpepper. 2019. Joint optimization of cascade ranking models. In *Proceedings of the twelfth ACM international conference on web search and data mining*. 15–23.

[12] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W Bruce Croft, and Xueqi Cheng. 2020. A deep look into neural ranking models for information retrieval. *Information Processing & Management* 57, 6 (2020), 102067.

[13] Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. 2021. Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 113–122.

[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] Miron B Kursa and Witold R Rudnicki. 2011. The all relevant feature selection using random forest. *arXiv preprint arXiv:1106.5112* (2011).

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

[17] Gilles Louppe, Louis Wehenkel, Antonio Sutera, and Pierre Geurts. 2013. Understanding variable importances in forests of randomized trees. *Advances in neural information processing systems* 26 (2013).

[18] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, and Nicola Tonellotto. 2023. On the efficiency-effectiveness trade-offs in learning to rank. *Information Processing & Management* 60, 2 (2023), 103134.

[19] Joel Mackenzie, J Shane Culpepper, Roi Blanco, Alistair Moffat, and Falk Scholer. 2018. Efficiency-effectiveness tradeoffs in learning to rank. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1093–1102.

[20] Irina Matveeva, Chris Burges, Timo Burkard, Andy Laucius, and Leon Wong. 2006. High accuracy retrieval with multiple nested ranker. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. 437–444.

[21] Tao Qin and Tie-Yan Liu. 2010. A general approximation framework for direct optimization of information retrieval measures. *Information retrieval* 13, 4 (2010), 375–397.

[22] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 datasets. *arXiv preprint arXiv:1306.2597* (2013).

[23] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243* (2019).
[24] Lidan Wang, Jimmy Lin, and Donald Metzler. 2011. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. 105–114.
[25] Lidan Wang, Jiwei Mao, Yinzhi Guo, Zhaochun Ren, and Xue Han. 2018. A survey of efficiency optimization in web search systems. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–32.
[26] Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2018. The LambdaLoss framework for ranking metric optimization. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (2018), 1313–1322.
[27] Yunli Wang, Zhiqiang Wang, Jian Yang, Shiyang Wen, Dongying Kong, Han Li, and Kun Gai. 2023. Adaptive Neural Ranking Framework: Toward Maximized Business Goal for Cascade Ranking Systems. *arXiv preprint arXiv:2310.10462* (2023).
[28] Hamed Zamani, Michael Bendersky, Donald Metzler, Honglei Zhuang, and Xuanhui Wang. 2022. Stochastic retrieval-conditioned reranking. In *Proceedings of the 2022 ACM SIGIR International Conference on Theory of Information Retrieval*. 81–91.

7 Appendix

7.1 Data Analysis

In this section, we present additional graphs illustrating the depth of relevance and the distribution of scores. The query depth refers to the number of documents associated with a given query. Furthermore, we display the distribution across the training, testing, and validation sets. As mentioned earlier, the test set comprises 20% of the entire dataset.

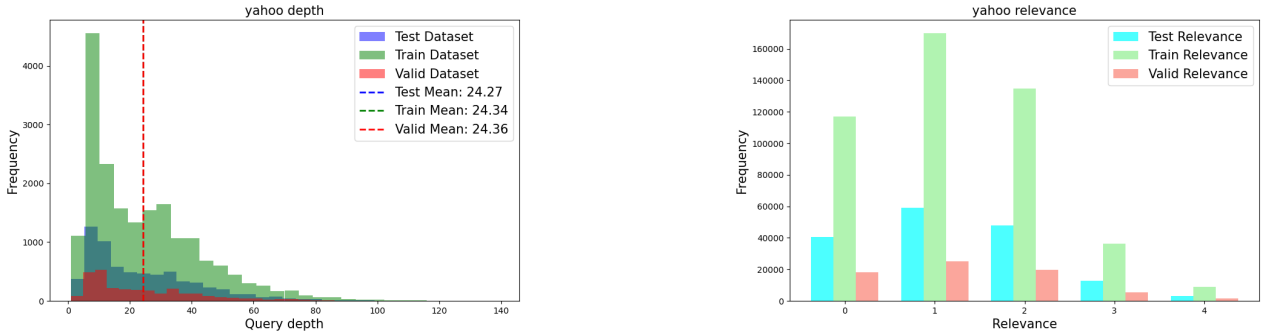

Figure 5: Yahoo dataset: (left) query depth distribution, (right) relevance distribution

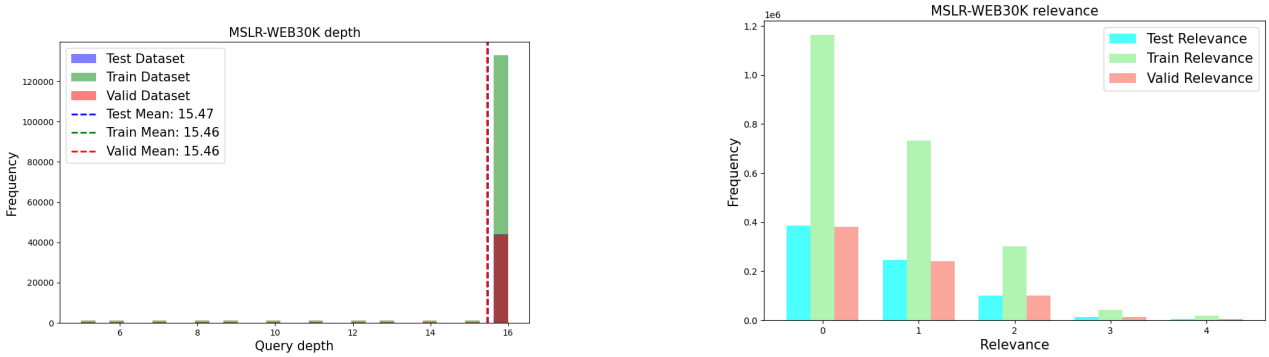

Figure 6: MSLR-WEB30K dataset: (left) query depth distribution, (right) relevance distribution

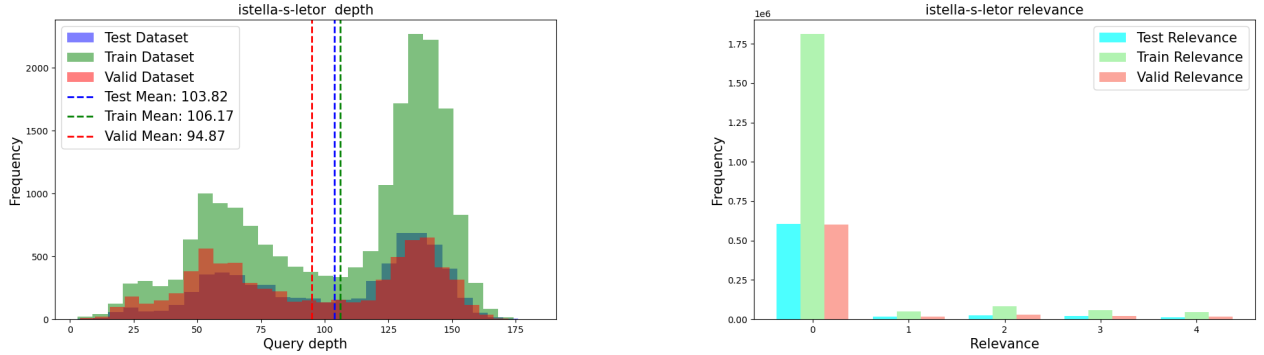

Figure 7: Istella-S LETOR dataset: (left) query depth distribution, (right) relevance distribution