RankFlow: Joint Optimization of Multi-Stage Cascade Ranking Systems as Flows

Jiarui Qin
Shanghai Jiao Tong University
qjr1996@sjtu.edu.cn

Jiachen Zhu
Shanghai Jiao Tong University
gebro13@sjtu.edu.cn

Bo Chen
Huawei Noah's Ark Lab
chenbo116@huawei.com

Zhirong Liu
Huawei Noah's Ark Lab
liuzhirong@huawei.com

Weiwen Liu
Huawei Noah's Ark Lab
liuweiwen8@huawei.com

Ruiming Tang
Huawei Noah's Ark Lab
tangruiming@huawei.com

Rui Zhang
ruizhang.info
rayteam@yeah.net

Yong Yu
Shanghai Jiao Tong University
yyu@sjtu.edu.cn

Weinan Zhang*
Shanghai Jiao Tong University
wnzhang@sjtu.edu.cn

ABSTRACT

Building a multi-stage cascade ranking system is a commonly used solution to balance the efficiency and effectiveness in modern information retrieval (IR) applications, such as recommendation and web search. Despite the popularity in practice, the literature specific on multi-stage cascade ranking systems is relatively scarce. The common practice is to train rankers of each stage independently using the same user feedback data (a.k.a., impression data), disregarding the data flow and the possible interactions between stages. This straightforward solution could lead to a sub-optimal system because of the *sample selection bias* (SSB) issue, which is especially damaging for cascade rankers due to the negative effect accumulated in the multiple stages. Worse still, the interactions between the rankers of each stage are not fully exploited. This paper provides an elaborate analysis of this commonly used solution to reveal its limitations. By studying the essence of cascade ranking, we propose a *joint training* framework named **RankFlow** to alleviate the SSB issue and exploit the interactions between the cascade rankers, which is the first systematic solution for this topic. We propose a paradigm of training cascade rankers that emphasizes the importance of fitting rankers on stage-specific data distributions instead of the unified user feedback distribution. We design the RankFlow framework based on this paradigm: The training data of each stage is generated by its preceding stages while the guidance signals not only come from the logs but its successors. Extensive experiments are conducted on various IR scenarios, including recommendation, web search and advertisement. The results verify the efficacy and superiority of RankFlow.

*Weinan Zhang is the corresponding author.

SIGIR '22, July 11–15, 2022, Madrid, Spain

ACM ISBN 978-1-4503-8732-3/22/07…$15.00
https://doi.org/10.1145/3477495.3532050

CCS CONCEPTS

• **Information systems** → **Information retrieval**.

KEYWORDS

Cascade Ranking Systems, Recommendation, Information Retrieval

ACM Reference Format:
Jiarui Qin, Jiachen Zhu, Bo Chen, Zhirong Liu, Weiwen Liu, Ruiming Tang, Rui Zhang, Yong Yu, and Weinan Zhang. 2022. RankFlow: Joint Optimization of Multi-Stage Cascade Ranking Systems as Flows. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3477495.3532050

1 INTRODUCTION

Balancing the efficiency and effectiveness of ranking models has always been vital to information retrieval (IR) tasks, such as recommendation or web search. Complex IR models [23, 24] have better accuracy but usually suffer from low efficiency, which makes it difficult for online deployment because of the latency constraints [22]. Simple models [16, 27], on the contrary, are limited by their capacity, but their low time complexity is feasible for scoring a large number of documents. To balance the trade-off between efficiency and effectiveness, a widely used solution in the industry is multi-stage cascade ranking systems [30]. There are multiple rankers in the system, from simple models to complex ones. The simple rankers are deployed in the early stages of the retrieval, aiming to quickly filter out the irrelevant documents/items from massive candidates w.r.t. the given query, while the complex rankers are usually placed in the later stages of the retrieval, ranking the documents more accurately. A typical three-stage cascade ranking system is illustrated in Figure 1. There are three stages from the bottom to the top: recall, pre-ranking, and ranking. Each stage is responsible for selecting the top documents in the list it receives and feeding them to the next stage. The recall stage selects from the entire document pool and the selected documents by the ranking stage will be finally displayed to the user. The scale of the different stages shrinks from millions to tens, as shown in Figure 1.

In recent years, various effective IR models, mostly designed for a specific stage, have been proposed – including recall models [6, 16, 40], pre-ranking models [19, 31] and ranking models

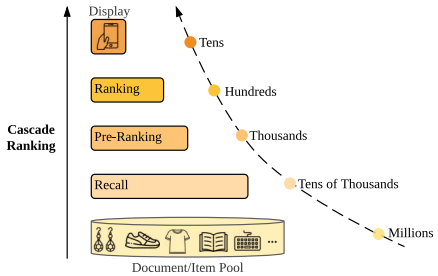

Figure 1: A typical three-stage cascade ranking system.

[2, 12, 13, 20, 21, 38, 39]. Nevertheless, these models only emphasize the performance of a single-stage without providing particular discussions or designs on how they will influence the performance of other stages in a cascade system. Compared to the rapid development of the single-stage models, little research has been done for the cascade ranking architecture, although it is widely used in practice. Early attempts [4, 30, 33, 34] focus on cost-aware learning to manage the trade-off between effectiveness and efficiency, where the cost of computing is introduced to the loss functions. These early research works are mainly proposed for tree-based [33, 34], or linear rankers [30], which may not be the mainstream choices in today's large-scale IR systems. Other cascade learning methods such as [11] try to derive the gradients of the cascade ranking process by using some approximations and relaxations, which may lead to sub-optimal results.

However, most existing works (either single-stage models or cascade models), train rankers independently on the same impression data [1] and thus fail to address the following two challenges. 1) The first one is the **sample selection bias (SSB)** issue [3], which is caused by the inconsistency between the training and the inference data. During training, rankers are trained by the impression data, which only covers the documents that are exposed to the users ("Display" part in Figure 1). While at inference, rankers are required to rank a large amount of unseen data. This means the data distribution during training is far different from that during inference, which will result in a sub-optimal system. It is like forcing a student to take an exam beyond the syllabus. 2) The second challenge is how to exploit the interactions between the rankers. Most existing training methods treat the rankers independently, and each ranker is not aware of the existence of others. However, there could be huge potential if the rankers could interact with each other and achieve better performance without changing the model architectures or sacrificing inference efficiency.

To address the above challenges, we propose a unified joint training framework for multi-stage cascade ranking systems named **RankFlow**. The major paradigm of RankFlow is *fitting each ranker towards its own stage-specific data distribution, with the help of other stages*. In RankFlow, the training data for each ranker is stage-specific, which contributes to reflecting and adapting to its own scoring scope. This idea fits the essence of cascade ranking systems: each stage should select a better set of relevant documents from the set it receives. Specifically, the training data of each stage is generated by the preceding stages. The scale of training documents should be close to the scale of documents at inference time. It helps alleviate the SSB issue because the generated data is not limited to the impression data but covers the exact scale of documents of that stage. Apart from getting training data from the preceding stages, the rankers are further correlated with each other by learning from the succeeding rankers. Normally the rankers of succeeding stages are more complex and capable, which makes them suitable to be teachers. Learning from a teacher could improve the student's capability. RankFlow explores optimizing a ranker by exploiting the interactions with the other rankers in the cascade system. The interactions between rankers include data generation and supervision signals. These interactions among the stages help to solve the two challenges mentioned above.

The joint training procedure of RankFlow could be summarized in two steps: data generation step and learning step. The data generation process prepares training data for each stage based on the preceding rankers. The learning step of a ranker could be divided into two sub-processes: self-learning and tutor-learning. For self-learning, the ranker is trained using the labels from the impression logs and regards the undisplayed samples as negative. It is actually learning to discriminate the documents given by its predecessor. For tutor-learning, the ranker learns the relevance estimations given by its successor. The successor acts like a teacher. Thus the two processes of each stage could form a self-learning flow and a tutor-learning flow. By conducting both flows iteratively, the rankers of the cascade system are trained jointly.

The contributions of our paper are summarized as follows:

- We are the first to propose the paradigm of fitting each ranker on its own stage-specific data distribution with the help of other stages for training cascade rankers.
- We propose the RankFlow joint training framework, in which the training data of each stage is generated by its preceding stages while the guidance signals come from not only the logs but also the succeeding ranker. RankFlow does not require new data sources or increase inference complexity to improve the overall performance.
- We provide a detailed analysis on the limitations of training cascade rankers independently on impression data. To our knowledge, it is the first time that the limitations are discussed formally in the research of cascade ranking.

Extensive experiments are conducted based on three real-world datasets, covering the major information retrieval scenarios, including recommendation, web search, and advertising. The improvements of using RankFlow are significant, verifying its efficacy.

2 BACKGROUND

This section will introduce the preliminaries and the basics of a multi-stage cascade ranking system as background knowledge.

2.1 Preliminaries

The goal of a ranking system is to produce a high-quality ranked list of documents corresponding to a given query. We denote a query as $q \in \mathcal{Q}$ and a document $d \in \mathcal{D}$, where $\mathcal{Q}$ and $\mathcal{D}$ are the given query and document set, respectively. The quality of a ranked list is determined by the documents' relevance to the query, which

[1] Impression means page view or display, indicating a user has seen the query-document pair and gives feedback.

is denoted by $y \in \mathcal{Y}$. For explicit feedback problems, $\mathcal{Y}$ usually consists of multiple ratings, while for implicit feedback problems, $\mathcal{Y}$ is usually binary.

Supervised learning is the most widely used technique to train a ranking model. As a supervised learning model, the feature is query-document pair $x = \langle q, d \rangle$ and label y is used as a supervision signal. The feature space is represented as $\mathcal{X} = \mathcal{Q} \times \mathcal{D}$. A ranking model parameterized by θ is denoted as R_θ (R for short). Given a query-document pair x, a ranking model outputs the predicted relevance label as

$$\hat{y}(x) = R_\theta(x). \tag{1}$$

In the probabilistic perspective, a query-document pair x and the corresponding relevance label y could be regarded as a value of random variables X and Y. The joint probabilistic distribution of X and Y is denoted as $T(X, Y)$. Thus the learning of a ranking model could be conducted by minimizing the expected risk as

$$\min_\theta \text{Risk}_\theta(\hat{y}), \text{ where } \text{Risk}_\theta(\hat{y}) = \int \ell(y, \hat{y}(x)) dT(x, y), \tag{2}$$

in which $\ell(y, \hat{y}(x))$ could be point-wise, pair-wise or list-wise loss functions. And (x, y) is sampled from the distribution $T(X, Y)$.

2.2 Multi-Stage Cascade Ranking System

Multi-stage cascade ranking systems are widely used in real-world IR applications such as recommendation and web search. A typical structure consists of three stages: recall, pre-ranking, and ranking. Formally speaking, a multi-stage cascade ranking system consists of multiple rankers, which work in a cascade way. Assume for an S-stage system, we have S rankers: $\{R_{\theta_1}, R_{\theta_2}, \ldots, R_{\theta_S}\}$, each of which might has different parameters and architectures. We use R_i to denote a ranker in stage i for simplicity if there is no ambiguity. The ranking task is selecting the best matched documents w.r.t. the given query. Thus the working procedure of stage i is obtaining the documents with top scores and feeding them to the next stage, which is formulated as,

$$\begin{aligned} \mathcal{D}_{i+1}^q = &\{d_j\}_{j=1}^{K_i},\ R_i(\langle q, d_j \rangle) \geq \text{Top-}K_i(\mathcal{H}_i), \\ &\mathcal{H}_i = \{R_i(\langle q, d \rangle)\}, \forall d \in \mathcal{D}_i^q, \end{aligned} \tag{3}$$

where $\mathcal{D}_{i+1}^q$ is the resulted document list given by R_i to the succeeding stage $i+1$ for the given query q. $\mathcal{D}_i^q$ is the received document list from the preceding stage $i-1$, K_i is the truncated length of the ranking list for stage i. $\mathcal{H}_i$ represents the set of ranking scores on all the query-document pairs of stage i and Top-K_i represents the K_i-th largest score. $R_i(\langle q, d \rangle)$ is the ranking score of q and d. $\mathcal{D}_1^q = \mathcal{D}$ is the whole document set, and $\mathcal{D}_{S+1}^q$ is the final ranking list that will be displayed to the user.

In a more general view, Eq. (3) could be written as

$$\mathcal{X}_{i+1} = \text{Rule}_{\theta_i, K_i}(\mathcal{X}_i), \tag{4}$$

where $\mathcal{X}_i = \{\langle q, d \rangle, \forall q \in \mathcal{Q}, d \in \mathcal{D}_i^q\}$, meaning the data received by the stage i. The scoring and ranking process to get the top documents could be regarded as a selection rule determined by the ranker's parameters θ_i and the system truncation parameter K_i. Then we could have that $\mathcal{X}_{i+1} \subset \mathcal{X}_i$, indicating the cascade ranking is a data selection process. The involving data for each stage is shrinking. As the input data size (i.e., number of documents) shrinks, the early stages usually utilize rankers with simpler architectures and fewer parameters, while more complex rankers are deployed for the later stages. For the early stages stages like recall, the dot product model is often used [6, 16] as the maximum inner product search (MIPS) could be conducted very quickly. As for later stages like ranking, complex models [25, 38, 39] are deployed to output a more sophisticated ranking list.

3 METHODOLOGY

In this section, we first introduce our paradigm of training a multi-stage cascade ranking system. Then the detailed design of RankFlow is presented. Finally, we discuss the limitations of the widely used training procedure in the view of RankFlow.

3.1 The Paradigm of Training Cascade Rankers

We propose that the paradigm of training cascade rankers is to fit the stage-specific data distributions for different stages accordingly, with the help of other stages. To reveal the paradigm, we could start with the ideal situation.

3.1.1 Ideal Situation. In the ideal case, the optimization object for ranker R_i at stage i is

$$\min_{\theta_i} \text{Risk}_{\theta_i}(\hat{y}), \text{ where } \text{Risk}_{\theta_i}(\hat{y}) = \int \ell(y, \hat{y}(x)) dT(x, y). \tag{5}$$

In Eq. (5), $(x, y) \sim T(X, Y)$. $T(X, Y)$ is the distribution to generate the query-document pair x and the corresponding relevance label y [2]. For the ideal case, $(x, y) \sim T(X, Y) = \Pr(X, Y)$, where $\Pr(X, Y)$ is the underlying ground-truth distribution of data.

However, the ideal case is hard to fulfill because of the following two reasons: 1) The ground-truth data distribution $\Pr(X, Y)$ is hard to depict because the entire feature space $\mathcal{X}$ is huge, and $Y|X$ is unknown in most of the cases (the data that is not displayed to the users). 2) It is challenging for a ranker to fit the entire data distribution.

As the essence of a cascade ranking system is selecting a better set of relevant documents from the set each ranker receives, which is shown in Eq. (3), the scoring scope of each stage is nested with each other (Eq. (4)). Thus, we argue that ranker R_i of stage i does not necessarily need to fit the entire $\Pr(X, Y)$ distribution very well. Instead it could focus on fitting the data that is filtered by the preceding stages. As shown in Figure 2, each stage i could only focus on the data distribution $\Pr_i(X, Y)$ of its own, instead of targeting all the stages to the same data distribution. The current training method is fitting each stage to $\Pr(X, Y, O = 1)$, where O represents whether the data is displayed to the users and has the ground-truth relevance. The limitations of the widely used method are discussed in detail in Section 3.5.

The Paradigm. The paradigm of RankFlow is to fit each ranker R_i of stage i to its own data distribution $\Pr_i(X, Y)$, which is consistent with its inference scope. As the scope of $\Pr_i(X, Y)$ could be much smaller than that of the entire data distribution, this paradigm has two good properties: 1) $\Pr_i(X, Y)$ is easier to depict. 2) It is easier for a ranker R_i to fit $\Pr_i(X, Y)$ than $\Pr(X, Y)$.

[2] In this paper, we may regard click to represent relevant, and non-click to represent irrelevant. The bias between relevance and click is not in the scope of this paper.

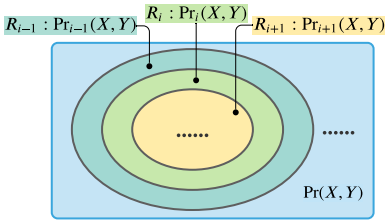

Figure 2: Illustration of the training paradigm of RankFlow: ranker R_i should be trained on $\Pr_i(X, Y)$.

We use an analogy to make our points clearer: Suppose we have multiple students who are going to take exams of different subjects or different syllabuses. The current solution is to prepare a unified cheat sheet that cannot cover the syllabuses well. RankFlow, on the contrary, will prepare each student a suitable cheat sheet w.r.t. her own syllabus. The "student" is a ranker, the "syllabus" is the scale of documents in inference, and the "cheat sheet" represents the training data distribution.

3.2 Approximating Stage-Specific Data Distributions

The stage-specific data distribution $\Pr_i(X, Y)$ could be approximated using two components: $\Pr_i(X)$ and $\Pr_i(Y \mid X)$ as

$$\begin{aligned}\Pr_i(X, Y) &= \Pr_i(Y \mid X) \Pr_i(X) \\ &= (\Pr_i(Y \mid X, O = 1) + \Pr_i(Y \mid X, O = 0)) \cdot \Pr_i(X),\end{aligned} \tag{6}$$

where O represents whether the query-doc pair x is displayed to the users. Thus, to generate data (x, y) from $\Pr_i(X, Y)$ could be regarded as two following steps: generating x using $\Pr_i(X)$, and generating y from $\Pr_i(Y \mid X)$.

3.2.1 $\Pr_i(X)$ *Approximation.* To approximate the X distribution for stage i, the most straightforward way is to use the preceding stages to generate the data that are closest to the data scale of inference. In other words, $\Pr_i(X)$ is determined by stages $1, \ldots, (i-1)$. The preceding stages perform as a data selector to R_i. To generate data with proper scales w.r.t. different stages, we just need to follow the cascade ranking procedure from the bottom to the top.

3.2.2 $\Pr_i(Y \mid X)$ *Approximation.* As for $\Pr_i(Y \mid X, O = k)$ we have two ways to estimate. The first one uses the real relevant label for the impression data ($k = 1$) and the negative relevance label for the unseen data ($k = 0$). The second way is to utilize the model of the succeeding stage. The succeeding stage $i + 1$ generally has a more advanced yet complex model with better capacity. Thus the succeeding stage $i + 1$ could act as a "teacher," and its prediction on the query-doc pair x could be utilized as a supervision signal. So the relevance distribution $\Pr_i(Y \mid X, O = k)$ could be regarded as determined by the succeeding stage. Using the succeeding stage as a teacher could provide the unseen data with an estimated relevance and help the ranker in the current stage i learn better.

In our method, each stage gets its own training data from its preceding stages. Moreover, the supervision signal not only comes from the user feedback logs but also from the succeeding stage with better modeling capacity.

3.3 RankFlow: Overall Framework

The overall framework of RankFlow is shown in Figure 3. The entire procedure of RankFlow could be divided into two phases: independent training and joint training.

3.3.1 Independent Training. In independent training phase, each stage i is trained on the impression dataset D_{imp} using ordinary loss functions such as binary cross entropy loss. This dataset could be regard as sampled from the distribution $\Pr(X, Y, O = 1)$. For the recall stage, specifically, we use the impression data with the random negative samples. The independent training process works as a warmup phase essentially. As the joint training process described in Section 3.2 requires other stages to help with the currently trained stage, they should have basic ranking abilities. As shown in Figure 3, after the independent training, the initial ranker R_i^{init} becomes R_i^{warm}.

3.3.2 Joint Training. In the joint training phase, the first step is to generate the stage-specific data for each stage using Eq. (3). After we get the dataset $\mathcal{X}_i$ generated by preceding stages, obtaining the corresponding label $\mathcal{Y}_i$ is the key point. According to different sources of the labels, joint training is conducted iteratively on two flows: *self-learning flow* and *tutor-learning flow*. We incorporate the labels from the impression logs for the self-learning flow. For the observed query-doc pair x, we use its accurate label while directly using negative (zero) labels for the unobserved ones. The self-learning of R_i is essentially learning to discriminate the documents generated by its predecessors. As shown in Figure 3, for the tutor-learning flow, the supervision signals come from the succeeding ranker R_{i+1}. In this setting, the ranker R_i is the student and R_{i+1} becomes a teacher. The training process lets the student learn to rank the documents like its teacher. In this way, the teacher's capacity will be transferred to the student without changing the student's architecture. For each stage i, the ranker R_i has these two tasks: self-learning and tutor-learning. These two tasks form the corresponding flows, which are conducted iteratively.

3.4 RankFlow: Training Details

In this part, we describe the detailed training method of a stage i in RankFlow framework, which is illustrated in the right plot (Phase II(i)) of Figure 3.

3.4.1 Data Generation. The first step of training ranker R_i is to obtain its training set [3], which is generated by the preceding stage $i - 1$. Following Eq. (3) and Eq. (4) we get the training data $\mathcal{X}_i$ for R_i. They could be regarded as sampled from $\Pr_i(X)$ as $x_j \sim \Pr_i(X)$, $\forall x_j \in \mathcal{X}_i$, where $\Pr_i(X)$ is determined by the preceding stages. The top K_i documents from R_{i-1} will form the training set for R_i. The exception is R_1 (recall model) because it has no preceding stage. So we use the impression data with the random negative samples as the training data for it to cover more documents.

3.4.2 Self-Learning Flow. We use the relevance label from the impression logs for the self-learning flow. As illustrated in Figure 3, the $\Pr_i(Y \mid X)$ distribution is determined by the impression logs and the negative assumption for the undisplayed data. Specifically,

[3]The training dataset here only contains the x part, y is generated using two different ways, which will be discussed in the following sections.

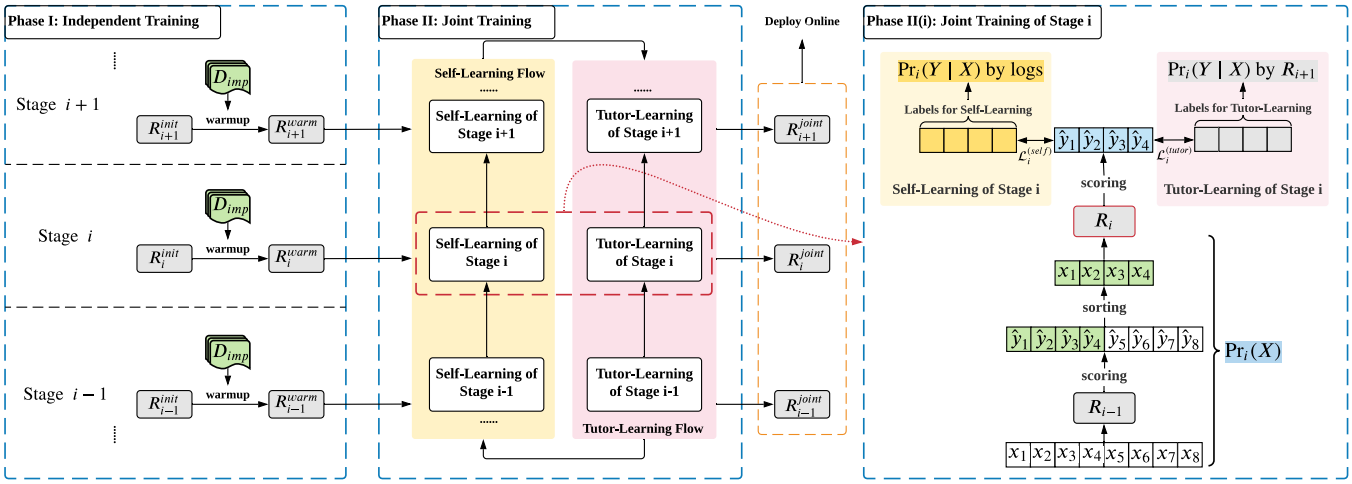

Figure 3: Overall illustration of the RankFlow. Phase I is independent training which acts as a warmup phase. Phase II is the major joint training process. Each ranker has two learning tasks: self-learning and tutor-learning. The two tasks form the corresponding flows. In each flow, the stages are trained bottom-up and the two flows are conducted iteratively. The details of self-learning and tutor-learning for stage i is shown in Phase II(i).

for an observed query-doc pair x_j, the accurate relevance label is used. For an unobserved query-doc pair x_j, y_j is set to be negative. The loss function here could be the widely used learning to rank loss. Here we use the point-wise loss as

$$\mathcal{L}_i^{(self)} = \frac{1}{|\mathcal{X}_i|}\sum_{j=1}^{|\mathcal{X}_i|} -y_j \log(R_i(x_j)) - (1-y_j)\log(1-R_i(x_j)). \quad (7)$$

The self-learning flow consists of all the training process from stage 1 to stage S by solving $\arg\min_{\theta_i} \mathcal{L}_i^{(self)}$, $\forall i \in \{1,\ldots,S\}$.

In the self-learning flow, the information flows bottom-up because the preceding stage tells the current stage what data it should learn to discriminate. From [9, 15] we can find that the negative samples are vital for training an accurate ranker. The procedure of the self-learning flow actually regards the preceding stage as a data generator. This generator will provide high-quality negative samples with a proper scale.

3.4.3 Tutor-learning Flow. For the tutor-learning task, the ranker R_i aims to learn the scores given by its successor R_{i+1}. The loss function $\mathcal{L}_i^{(tutor)}$ for stage i is

$$\begin{aligned}
\mathcal{L}_i^{(tutor)} &= \alpha \cdot \ell_{\text{ranking}} + (1-\alpha)\cdot \ell_{\text{mse}}.\\
\ell_{\text{ranking}} &= -\frac{1}{|\mathcal{Q}|}\sum_q \log\left(\sigma\left(\frac{\sum_{d_p} R_i(\langle q, d_p\rangle)}{K_i} - \frac{\sum_{d_n} R_i(\langle q, d_n\rangle)}{(K_{i-1}-K_i)}\right)\right),\\
\ell_{\text{mse}} &= \frac{1}{|\mathcal{X}_i|}\sum_{j=1}^{|\mathcal{X}_i|}(R_{i+1}(x_j) - R_i(x_j))^2,
\end{aligned} \quad (8)$$

where ℓ_{ranking} is the ranking loss and ℓ_{mse} is the mean squared error. They are combined by the weight parameter α. $\sigma(t) = \frac{1}{1+\exp(-t)}$ is the sigmoid function. The ℓ_{mse} is utilized to learn the teacher's ranking score in a point-wise manner. Training dataset $\mathcal{X}_i$ could be divided into the query part and the document part. For each query q in $\mathcal{Q}$, in stage i, it has a document list whose length is K_{i-1}, and we use L_q^i to denote the list. To learn the ranking of the teacher model R_{i+1}, we rank the documents in L_q^i according to the scores given by R_{i+1}. d_p in Eq. (8) represents the document that is among the top-K_i choices by R_{i+1}, and d_n represents the document that is not. The ranking loss is essentially the "pair-wise" loss between the average score of the top-K_i documents and the rest of the documents in L_q^i. By incorporating the ℓ_{ranking} term, the student model R_i is learning the order of the documents given by its teacher R_{i+1} or, equivalently speaking, learning to generate a better top-K_i list for R_{i+1}.

The exception for tutor-learning flow is the final stage S because it does not have a succeeding ranker as the teacher. Thus the tutor-learning flow is conducted from R_1 to R_{S-1}.

The entire RankFlow training procedure is summarized in the Algorithm 1. The key point of RankFlow is generating data for each stage (line 4 and 6) and conducting the self-learning flow (line 5) and the tutor-learning flow (line 7) iteratively until the performance of the final stage converges. The rankers after the joint training R_i^{joint} are ready to be deployed.

3.5 Discussions on Existing Training Methods

This section discusses the limitations of the current existing training methods for multi-stage cascade ranking systems.

3.5.1 Independent Training on Impression Data. Because the impression data is easy to collect (users feedback logs). The existing widely used solution is to use the impression data, i.e., $(x, y) \sim \Pr(X, Y, O = 1)$, where O represents if the data is displayed to the user. This solution implicitly assumes that

$$\Pr(X, Y) \propto \Pr(X, Y, O = 1), \quad (9)$$

and uses $\Pr(X, Y, O = 1)$ to replace $\Pr(X, Y)$. But it does not hold for most of the cases [35]. We firstly express both sides in Eq. (9)

Algorithm 1 Training procedure of RankFlow

Require: Impression dataset D_{imp}, S rankers R_i^{init}, $i = 1, \ldots, S$. Cascade length for each stage $K_1, \ldots, K_S$.

Ensure: The rankers that are trained by RankFlow as R_i^{joint}, $i = 1, \ldots, S$.

1: Initialize the parameters of all the rankers $R_1^{init}, \ldots, R_S^{init}$.
2: Conduct the independent training process as described in Section 3.3.1 and get the warmed-up rankers R_i^{warm}.
3: **repeat**
4: Generate the training set $\mathcal{X}_i$ for all the stages using Eq. (3).
5: For every stage 1,...,S, conduct self-learning using Eq. (7).
6: Re-generate the training set $\mathcal{X}_i$ for all the stages using Eq. (3) as the rankers are updated.
7: For stage $1, \ldots, S-1$, conduct tutor-learning using Eq. (8).
8: **until** performance of the final stage coverges

respectively as

$$\begin{aligned} \Pr(X, Y) &= \Pr(Y \mid X) \Pr(X) \\ \Pr(X, Y, O = 1) &= \Pr(Y \mid X, O = 1) \Pr(X, O = 1). \end{aligned} \tag{10}$$

As the relevance Y is solely dependent on query-doc feature X and impression does not affect it, we could have $\Pr(Y \mid X, O = 1) = \Pr(Y \mid X)$. Furthermore, $\Pr(X, O = 1) = \Pr(X) \Pr(O = 1 \mid X)$. The correctness of Eq. (9) is based on whether $\Pr(O = 1 \mid X)$ is a constant as

$$\Pr(O = 1 \mid X) = \text{Uniform}(X). \tag{11}$$

However, Eq. (11) does not hold except for a random display strategy. In most cases, the probability of a document being displayed to a user is relevant to the query-doc feature X. Thus the existing solution has inherent limitations. To be specific, using data $(x, y) \sim \Pr(X, Y, O = 1)$ to train each stage will suffer from **sample selection bias (SSB)** [3]. The impression data distribution is inconsistent with the inference data distribution because impression data only covers a limited part of documents.

3.5.2 Practical Improvements. Some improvements are common in real-world applications to alleviate the SSB issue. A widely used improvement for training a better recall model is to generate unobserved samples via random negative sampling [15]. These unobserved negative samples are combined with the observed samples to form a training set. We could have that

$$\Pr(X, Y) = \Pr(X, Y, O = 1) + \Pr(X, Y, O = 0), \tag{12}$$

where

$$\Pr(X, Y, O = 0) = \underbrace{\Pr(Y|X, O = 0)}_{Y \text{always negative}} \underbrace{\Pr(X, O = 0)}_{\text{random sampling}} \tag{13}$$

The random negative sampling is essentially approximating the unobserved distribution $\Pr(X, Y, O = 0)$ by setting the rules that query-doc pair data X is sampled from uniform distribution and the relevance label Y is always negative. This approach is an effective solution to boost the performance of the recall stage, but it is not a systematic solution for all the stages in a cascade ranking system. RankFlow, on the contrary, uses stage-specific data distribution to provide (negative) samples, which is more suitable for each stage itself. Furthermore, the succeeding ranker is utilized as another source of supervision signal.

4 EXPERIMENTS

In this section, we present the details of the experiments with the corresponding analysis. The implementation code of RankFlow is available [4]. Four research questions (RQs) lead our discussions:

- **RQ1:** Is the overall performance of RankFlow superior to the baselines?
- **RQ2:** Are the self-learning and tutor-learning both effective and essential? What if we replace a simple ranking model to a complex one instead of using RankFlow?
- **RQ3:** What is the influence of some crucial hyperparameters?
- **RQ4:** What is the training efficiency of RankFlow?

4.1 Experimental Settings

4.1.1 Datasets. To verify the effectiveness of RankFlow, we conduct experiments on three widely used datasets from recommendation and web search scenarios: MovieLens-1M (ML-1M), TianGong-ST, and Tmall. Besides, we also have done simulation experiments of the advertising task (ranking with bid price) based on the datasets.

Recommendation and advertising tasks can be regarded as a generalized information retrieval problem, where the query is the user profile/representation, and the documents are items. For the advertising task, the final list is ranked based on the combination of estimated relevance and bid price as $R_S(x_i) \times \text{bid}(x_i)$, which is a common form [18]. As the public datasets normally do not have the impression-level bid price, we generate the bid prices from a log-normal distribution [7]. The charging policy is the widely-used generalized second price (GSP) mechanism [8] on Cost-Per-Click (CPC) advertisement.

- **ML-1M** [5] contains users ratings on movies with 6,040 users and 3,706 items. As the implicit feedback is more common in practice, we set ratings no less than four as positive relevance and others as negative.
- **TianGong-ST** [6] is a dataset that contains searching logs of clicking records on each given query, which has 40,596 queries and 314,459 documents.
- **Tmall** [7] is provided by Alibaba Group, which contains user behavior history on Tmall e-commerce platform from May 2015 to November 2015. It contains 424,170 users and 1,090,390 items.

All the above datasets are split into train, validation, and test sets using timestamps [24, 25].

4.1.2 Compared Methods. We compare RankFlow with two different training procedures:

- **Independent** represents training the rankers of each stage independently on the impression dataset. For the recall stage, we use random negative sampling to enhance the coverage of documents as described in Section 3.5.2.

[4] https://github.com/qinjr/RankFlow
[5] https://grouplens.org/datasets/movielens/1m/
[6] http://www.thuir.cn/tiangong-st/
[7] https://tianchi.aliyun.com/dataset/dataDetail?dataId=42

- **ICC** [11] is the most recently proposed joint training method of cascade ranking. The score for a query-document pair involves all the stages, but the training data is still based on the impression logs. We choose the "ICC" mode of the framework because that is consistent with our cascade setting.

As for the specific rankers, we choose DSSM [16], and YoutubeDNN [6] for recall stage as they are both dual-tower models [36] which outputs query and document vector representations. At the inference time, we use maximum inner product search to accomplish the recall stage using FAISS [8]. For other stages, we utilize well-known models including FM [27], DeepFM [12], COLD [31], WDL [5], PNN [26], DIN [39] and DIEN [38].

4.1.3 Evaluation Metrics. Precision@K (P@K), recall@K (R@K), and F1@K are used to measure the recall models' performance. Because for the recall stage, we care more about the number of relevant documents returned than the detailed ranking orders. Hit ratio (HR@K), normalized discounted cumulative gain (NDCG@K), and mean average precision (MAP@K) are used to measure the ranking performance of other stages. The calculation of the above metrics can be found in [17]. Furthermore, we use effective cost per mile (eCPM) to measure the advertising income performance [29].

4.2 Performance and Analysis (RQ1)

4.2.1 On Recommendation and Web Search Datasets. The overall performances on three datasets are shown in Table 1, 2 and 3. For each dataset, we conduct experiments for two groups of different ranking systems to verify the compatibility of RankFlow as a framework. The main results of each group of experiments are shown in the "Cascade List Generation" column because it is the actual inference process for a multi-stage cascade ranking system: generating the ranking list for each query bottom-up. The "Impression Data" column is the testing results of a single model on the impression dataset after the independent training. This column is used to reflect that the later stage models usually have better performance due to the more complex structures.

The blue shaded area represents the final performance of the entire cascade system because the document lists given by the ranking stage will be displayed to the users. We could verify that RankFlow significantly improves ranking performance to a large extent on all three datasets. As for the gray shaded area is the performance of other stages (recall & pre-ranking). If the system is trained in the RankFlow framework, the performances of all the stages increase a lot compared to the independent training procedure, and ICC training. From the tables, we could verify that RankFlow could improve the performance apparently, and the improvements are maintained when the datasets or the rankers in the cascade systems are changed.

4.2.2 On Advertising Datasets. The results of the advertising experiments are shown in Table 4, 5 and 6 to simulate paid web search and paid recommendation. In the simulation experiment, the final display list is jointly determined by the model predictions and bid prices [18]. From the tables, we could find the performances of RankFlow still exceed the baselines significantly, which means more relevant documents are displayed to users (only the clicked documents will be charged because we use the Cost-Per-Click (CPC) mode. This group of experiments on the advertising task demonstrates the robustness of RankFlow. Although the final ranking is affected by bid prices and the training framework has no sense of it, the ranking performance, in terms of both clicking metrics (NDCG, MAP, CTR) and profit metrics (eCPM), still outperforms the baselines. Significant tests are conducted for the all the results in Section 4.2.

[8] https://faiss.ai

4.3 Ablation Study (RQ2)

In this section, we investigate the effectiveness of different components of RankFlow.

4.3.1 Removing the Flows. We test the ranking performance without the tutor-learning (w/o Tutor-L) and without the self-learning flow (w/o Self-L). As the scales of different metrics vary a lot, we choose to plot the relative performance gain of the two ablation models compared to the full framework, as shown in Figure 4.

The figure shows that the ranking performance decreases by 11% to 75% compared to the full framework if the self-learning flow is removed. If the tutor-learning flow is removed, the performance drops by 10% to 13% on three datasets.

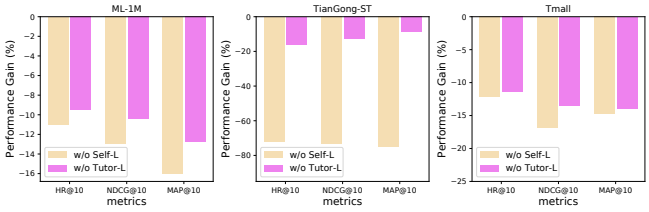

Figure 4: Ablation study of tutor-learning and self-learning. The ranker groups are the same with the "Ranking System 1" of each datasets as in Table 1,2 and 3.

These results have shown two facts: (1) Both components are important to RankFlow. The ranking quality will drop if any of the flows are removed. (2) Among the two flows, self-learning flow is more important to the performance because the metrics drop more if it is removed compared to the tutor-leaning flow. This fact demonstrates that learning to discriminate the data from the preceding stage is more important than satisfying the succeeding stage by learning the teacher's relevance estimations.

4.3.2 Replacing a Ranker to a More Complex One. We further investigate what will happen if we deploy the rankers of a succeeding stage directly into the preceding stage. For instance, we change a cascade system that contains DSSM+FM+DeepFM to DSSM+DeepFM+DeepFM, which means the pre-ranking model is updated to a more advanced one. We compare the new system with the original one (DSSM+FM+DeepFM) that is trained with or without RankFlow. The results are shown in Figure 5, which are of the pre-ranking stage. We show the relative performance gain w.r.t. the original ranker trained independently. "Rank as Pre-Rank" means that the pre-ranking model is replaced by the ranking model (as our examples above). The figure shows that deploying a more complex ranking stage model into the pre-ranking stage could benefit its performance. However, the performance gain is even larger

Table 1: Performance on ML-1M.

Group	Ranking System 1: DSSM+FM+DeepFM							Ranking System 2: DSSM+WDL+PNN						
Stage	Model	Impression Data		Cascade List Generation				Model	Impression Data		Cascade List Generation			
Recall	DSSM	AUC	LogLoss	Method	P@100	R@100	F1@100	DSSM	AUC	LogLoss	Method	P@100	R@100	F1@100
		0.7564	0.3908	Independent	0.0224	0.3224	0.0418		0.7564	0.3908	Independent	0.0224	0.3224	0.0418
				ICC	0.0224	0.3226	0.0419				ICC	0.0224	0.3227	0.0419
				RankFlow	**0.0226**	**0.3256**	**0.0423**				RankFlow	**0.0226**	**0.3258**	**0.0423**
Pre-Ranking	FM	AUC	LogLoss	Method	HR@50	NDCG@50	MAP@50	WDL	AUC	LogLoss	Method	HR@50	NDCG@50	MAP@50
		0.7753	0.3972	Independent	0.0271	0.2191	0.0897		0.7646	0.3962	Independent	0.0253	0.2097	0.0861
				ICC	0.0274	0.2203	0.0897				ICC	0.0252	0.2093	0.0858
				RankFlow	**0.0313**	**0.2634**	**0.1236**				RankFlow	**0.0308**	**0.2537**	**0.1142**
Ranking	DeepFM	AUC	LogLoss	Method	HR@10	NDCG@10	MAP@10	PNN	AUC	LogLoss	Method	HR@10	NDCG@10	MAP@10
		0.7815	0.3883	Independent	0.0421	0.1606	0.1066		0.7719	0.4612	Independent	0.0472	0.1798	0.1208
				ICC	0.0423	0.1611	0.1066				ICC	0.0472	0.1799	0.1209
				RankFlow	**0.0473**	**0.1845**	**0.1269**				RankFlow	**0.0489**	**0.1848**	**0.1236**

Table 2: Performance on TianGong-ST.

Group	Ranking System 1: DSSM+FM+DeepFM							Ranking System 2: DSSM+DeepFM+PNN						
Stage	Model	Impression Data		Cascade List Generation				Model	Impression Data		Cascade List Generation			
Recall	DSSM	AUC	LogLoss	Method	P@100	R@100	F1@100	DSSM	AUC	LogLoss	Method	P@100	R@100	F1@100
		0.6785	1.1909	Independent	0.0012	0.0359	0.0022		0.6785	1.1909	Independent	0.0012	0.0359	0.0022
				ICC	0.0012	0.0363	0.0023				ICC	0.0011	0.0343	0.0021
				RankFlow	**0.0015**	**0.0460**	**0.0029**				RankFlow	**0.0015**	**0.0466**	**0.0029**
Pre-Ranking	FM	AUC	LogLoss	Method	HR@50	NDCG@50	MAP@50	DeepFM	AUC	LogLoss	Method	HR@50	NDCG@50	MAP@50
		0.7183	0.6017	Independent	0.0010	0.0113	0.0030		0.7286	0.5807	Independent	0.0010	0.0122	0.0037
				ICC	0.0010	0.0119	0.0032				ICC	0.0010	0.0118	0.0036
				RankFlow	**0.0014**	**0.0179**	**0.0065**				RankFlow	**0.0022**	**0.0423**	**0.0289**
Ranking	DeepFM	AUC	LogLoss	Method	HR@10	NDCG@10	MAP@10	PNN	AUC	LogLoss	Method	HR@10	NDCG@10	MAP@10
		0.7286	0.5807	Independent	0.0007	0.0033	0.0021		0.7347	0.5337	Independent	0.0011	0.0036	0.0015
				ICC	0.0008	0.0034	0.0021				ICC	0.0010	0.0033	0.0014
				RankFlow	**0.0025**	**0.0117**	**0.0080**				RankFlow	**0.0016**	**0.0063**	**0.0037**

Table 3: Performance on Tmall.

Group	Ranking System 1: DSSM+DeepFM+DIN							Ranking System 2: YoutubeDNN+COLD+DIEN						
Stage	Model	Impression Data		Cascade List Generation				Model	Impression Data		Cascade List Generation			
Recall	DSSM	AUC	LogLoss	Method	P@200	R@200	F1@200	YoutubeDNN	AUC	LogLoss	Method	P@200	R@200	F1@200
		0.8776	0.5142	Independent	0.0036	0.0371	0.0066		0.884	0.4991	Independent	0.0046	0.0465	0.0084
				ICC	0.0031	0.0306	0.0056				ICC	0.0046	0.0457	0.0083
				RankFlow	**0.0039**	**0.0392**	**0.0071**				RankFlow	**0.0048**	**0.0469**	**0.0087**
Pre-Ranking	DeepFM	AUC	LogLoss	Method	HR@50	NDCG@50	MAP@50	COLD	AUC	LogLoss	Method	HR@50	NDCG@50	MAP@50
		0.8961	0.5028	Independent	0.0072	0.0766	0.0334		0.8885	0.4993	Independent	0.0085	0.0914	0.0429
				ICC	0.0055	0.0599	0.0259				ICC	0.0073	0.0690	0.0269
				RankFlow	**0.0081**	**0.0892**	**0.0428**				RankFlow	**0.0093**	**0.0948**	**0.0457**
Ranking	DIN	AUC	LogLoss	Method	HR@10	NDCG@10	MAP@10	DIEN	AUC	LogLoss	Method	HR@10	NDCG@10	MAP@10
		0.8987	0.467	Independent	0.0147	0.0607	0.0432		0.8973	0.4547	Independent	0.0162	0.0658	0.0466
				ICC	0.0102	0.0422	0.0292				ICC	0.0138	0.0549	0.0377
				RankFlow	**0.0156**	**0.0718**	**0.0489**				RankFlow	**0.0189**	**0.0757**	**0.0501**

Table 4: Ads Performance on ML-1M (w/ Bid).

Group	Ranking System 1: DSSM+FM+DeepFM				Ranking System 2: DSSM+WDL+PNN			
Method	NDCG@10	MAP@10	CTR	eCPM	NDCG@10	MAP@10	CTR	eCPM
Independent	0.1148	0.0747	0.0298	622.0	0.1027	0.0670	0.0259	524.3
ICC	0.1169	0.0769	0.0299	622.7	0.1035	0.0678	0.0258	536.2
RankFlow	**0.1612**	**0.1082**	**0.0466**	**883.6**	**0.1562**	**0.1054**	**0.0398**	**824.6**

Table 5: Ads Performance on TianGong-ST (w/ Bid).

Group	Ranking System 1: DSSM+FM+DeepFM				Ranking System 2: DSSM+DeepFM+PNN			
Method	NDCG@10	MAP@10	CTR	eCPM	NDCG@10	MAP@10	CTR	eCPM
Independent	0.0039	0.0024	0.0009	17.34	0.0051	0.0030	0.0012	23.33
ICC	0.0042	0.0026	0.0010	17.67	0.0044	0.0027	0.0010	24.60
RankFlow	**0.0076**	**0.0046**	**0.0018**	**32.79**	**0.0061**	**0.0035**	**0.0015**	**28.55**

if the original pre-ranking model could be trained under the RankFlow framework. Not only would the performance be better, but the inference complexity is unharmed. Thus training in RankFlow is more feasible for online deployment than directly changing the model to a complex one if the online computation power restriction is tight.

Table 6: Ads Performance on Tmall (w/ Bid).

Group	Ranking System 1: DSSM+DeepFM+DIN				Ranking System 2: YoutubeDNN+COLD+DIEN			
Method	NDCG@10	MAP@10	CTR	eCPM	NDCG@10	MAP@10	CTR	eCPM
Independent	0.0317	0.0207	0.0075	156.98	0.0366	0.0238	0.0089	184.13
ICC	0.0237	0.0152	0.0057	117.29	0.0308	0.0200	0.0074	153.15
RankFlow	**0.0391**	**0.0266**	**0.0094**	**198.72**	**0.0402**	**0.0294**	**0.0102**	**201.34**

4.4 Hyperparameters Study (RQ3)

We study the influence of some crucial hyperparameters in this section.

4.4.1 Number of stages. In the previous experiments, the total number of stages are all three: recall, pre-ranking, and ranking. To verify the robustness of RankFlow w.r.t. the number of stages, we

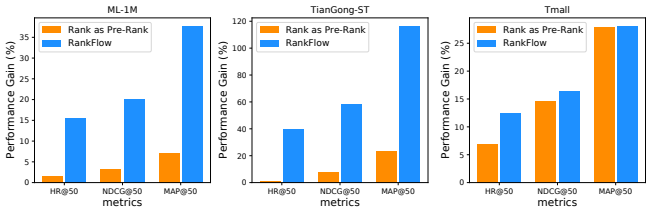

Figure 5: Relative performance gain against the original pre-ranking model. "Rank as Pre-Rank": use the model of ranking stage in pre-ranking stage. "RankFlow": original pre-ranking model that is trained in RankFlow. The result is of the pre-ranking stage.

conduct extensive experiments of two and four stages on ML-1M. The results are shown in Table 7 and Table 8.

We use DSSM and DeepFM in recall and ranking stages for the two-stage experiment, respectively. For the four-stage experiment, we have the recall, pre-ranking, ranking, and re-ranking stages. As for the re-ranking stage, for simplicity, we do not use the ordinary re-ranking models, which will take the initial ranking list as input. DeepFM still uses the point-wise scoring function because this section is only used to verify the robustness of RankFlow. The results demonstrate that RankFlow will steadily outperform the baselines regardless of the different cascade ranking structures.

Table 7: Two-Stage Performance on ML-1M.

Stage	Model	Cascade List Generation			
Recall	DSSM	Method	P@100	R@100	F1@100
		Independent	0.0224	0.3224	0.0418
		ICC	0.0224	0.3227	0.0419
		RankFlow	**0.0226**	**0.3257**	**0.0423**
Ranking	DeepFM	Method	HR@10	NDCG@10	MAP@10
		Independent	0.0421	0.1606	0.1066
		ICC	0.0424	0.1611	0.1065
		RankFlow	**0.0446**	**0.1711**	**0.1154**

Table 8: Four-Stage Performance on ML-1M.

Stage	Model	Cascade List Generation			
Recall	DSSM	Method	P@100	R@100	F1@100
		Independent	0.0224	0.3224	0.0418
		ICC	0.0224	0.3226	0.0419
		RankFlow	**0.0226**	**0.3257**	**0.0423**
Pre-Ranking	WDL	Method	HR@50	NDCG@50	MAP@50
		Independent	0.0253	0.2097	0.0860
		ICC	0.0252	0.2093	0.0859
		RankFlow	**0.0308**	**0.2544**	**0.1146**
Ranking	PNN	Method	HR@20	NDCG@20	MAP@20
		Independent	0.0404	0.2140	0.1203
		ICC	0.0404	0.2141	0.1203
		RankFlow	**0.0405**	**0.2177**	**0.1229**
Re-Ranking	DeepFM	Method	HR@10	NDCG@10	MAP@10
		Independent	0.0447	0.1682	0.1113
		ICC	0.0447	0.1684	0.1115
		RankFlow	**0.0449**	**0.1689**	**0.1124**

4.4.2 Loss weight α. The influence of the loss weight α in Eq. (8) is also investigated in this section. For the three datasets, we test the final ranking quality of RankFlow using different αs from the set $\{0, 0.25, 0.5, 0.75, 1\}$.

As shown in Figure 6, we can find that there exists fluctuation in the values of HR, NDCG, and MAP of different α choices. The metrics first rise and then fall, showing there is an optimal selection of α. Recall that α is the weight of the ranking loss function, the results indicate that there is a balance of the ranking loss and the point-wise MSE loss.

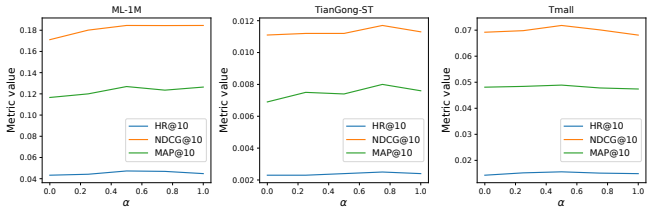

Figure 6: Performance on different α values of Eq. (8).

4.5 Training Efficiency (RQ4)

In this section, we present the wall time comparison between the independent training and RankFlow training. The experiments are conducted based on the same hardware settings with an NVIDIA GeForce GTX 1080Ti GPU processor and 128GB memory. The results are shown in Table 9, the training time of the three datasets are corresponding to the "Ranking System 1" in Table 1,2 and 3, respectively. The "Independent" column is the total independent training time of all three stages for each dataset. As the independent training is the first phase of RankFlow, we compare it with the joint training phase, which is the actual addition of time consumption introduced by RankFlow. From Table 9 we observe that the time consumption caused by the joint training phase of RankFlow is acceptable because it is in the same order of magnitudes as the independent training. Although the time consumption is increased, the training efficiency could be improved by some techniques such as distributed training. The inference time, which is the key factor of a real-world application, is not changed as the model architectures remain the same. And the performance gain is significant as shown in Section 4.2.

Table 9: Wall Time of RankFlow Training (in minutes). "∼" means approximation. Rel.Inc: relative increment of time.

Dataset	Independent	RankFlow (Phase II)	Rel.Inc
ML-1M	∼8	∼12	150%
TianGong-ST	∼29	∼31	107%
Tmall	∼550	∼244	44%

5 RELATED WORKS

Cascade ranking. The idea of multi-stage cascade ranking is proposed in [30] which is an excellent solution to balance the efficiency and effectiveness of a ranking system. Early-day research on cascade ranking systems is mainly about assigning different rankers

to each stage to achieve the desired trade-off collectively. Modeling the cost of each ranker in a cascade system is a major research direction, Wang et al. [30] propose to generalize the AdaRank algorithm by incorporating a cost model to construct a cascade where each weak learner becomes a single stage within the system. Chen et al. [4] use normalization loss functions to represent the cost of each stage and optimizes the ranker performance under the restrictions of computational cost. Other cost-aware methods includes [33, 34]. In recent years, cascade ranking models based on deep learning have been proposed. Gallagher et al. [11] try to derive the gradients of the cascade rankers and use an end-to-end method to optimize them directly, Fan et al. [9] propose to unify the multiple stages to one single stage with hard negative sampling. Fei et al. [10] propose to share features across different stages. [14] is proposed to jointly utilize multiple models for the recall stage and learning to aggregate the recalled items from different recall channels. These cascade ranking methods mainly use the impression data thus will still suffer from the sample selection bias. Furthermore, in the research of cascade rankers, no previous work specifically discusses or analyzes the limitations of the current independent training frameworks. Our work propose to train rankers on their specific data distributions which could alleviates the SSB issue and further exploits the interactions between each stage.

Other techniques for cascade ranking. Some practical techniques are used to train some stages in cascade rankers. Random negative sampling and hard negative sampling are used to optimize the performance of the recall stage [15]. Using random negative samples could alleviate the SSB issue because the model is trained in a broader range of data. Knowledge distillation is well-studied for ranking model [28, 32, 41], and there are also methods using it to optimize the candidate generation (a.k.a., recall) stage [37].

6 CONCLUSIONS

This paper proposes RankFlow as a general joint training framework of multi-stage cascade ranking systems for information retrieval. In RankFlow, each stage is trained on the specific dataset produced by its predecessors. While training, each stage has to conduct two sub-tasks: self-learning based on the labels from logs and tutor-learning by regarding the succeeding ranker as a teacher. The proposed method significantly outperforms baselines while the time complexity of the inference is not compromised. Furthermore, we thoroughly analyze and discuss the limitations of current training methods of cascade rankers from the data perspective. We plan to give a more in-depth theoretical analysis on the properties of cascade ranking systems in the future work, and try to deploy the RankFlow framework online to serve the real-world traffic.

ACKNOWLEDGEMENT

The SJTU team is supported by "New Generation of AI 2030" Major Project (2018AAA0100900) and National Natural Science Foundation of China (62177033). The work is also sponsored by Huawei Innovation Research Program. The author Jiarui Qin is supported by Wu Wen Jun Honorary Doctoral Scholarship, AI Institute, SJTU. We thank MindSpore [1] for the partial support of this work, which is a new deep learning computing framework.

REFERENCES

[1] 2020. MindSpore. https://www.mindspore.cn/
[2] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.
[3] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. 2020. Bias and debias in recommender system: A survey and future directions. *arXiv preprint arXiv:2010.03240* (2020).
[4] Ruey-Cheng Chen, Luke Gallagher, Roi Blanco, and J Shane Culpepper. 2017. Efficient cost-aware cascade ranking in multi-stage retrieval. In *SIGIR*. 445–454.
[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *DLRS*. 7–10.
[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.
[7] Ying Cui, Ruofei Zhang, Wei Li, and Jianchang Mao. 2011. Bid landscape forecasting in online ad exchange marketplace. In *KDD*.
[8] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. 2007. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American economic review* 97, 1 (2007), 242–259.
[9] Miao Fan, Jiacheng Guo, Shuai Zhu, Shuo Miao, Mingming Sun, and Ping Li. 2019. MOBIUS: towards the next generation of query-ad matching in baidu's sponsored search. In *SIGKDD*. 2509–2517.
[10] Hongliang Fei, Jingyuan Zhang, Xingxuan Zhou, Junhao Zhao, Xinyang Qi, and Ping Li. 2021. GemNN: Gating-enhanced Multi-task Neural Networks with Feature Interaction Learning for CTR Prediction. In *SIGIR*. 2166–2171.
[11] Luke Gallagher, Ruey-Cheng Chen, Roi Blanco, and J Shane Culpepper. 2019. Joint optimization of cascade ranking models. In *WSDM*. 15–23.
[12] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. Deepfm: a factorization-machine based neural network for ctr prediction. In *IJCAI*.
[13] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *ICLR*.
[14] Jiri Hron, Karl Krauth, Michael Jordan, and Niki Kilbertus. 2021. On component interactions in two-stage recommender systems. *Advances in Neural Information Processing Systems* 34 (2021).
[15] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2553–2561.
[16] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM, 2333–2338.
[17] Tie-Yan Liu. 2011. Learning to rank for information retrieval. (2011).
[18] Xiangyu Liu, Chuan Yu, Zhilin Zhang, Zhenzhe Zheng, Yu Rong, Hongtao Lv, Da Huo, Yiqing Wang, Dagui Chen, Jian Xu, et al. 2021. Neural Auction: End-to-End Learning of Auction Mechanisms for E-Commerce Advertising. *arXiv preprint arXiv:2106.03593* (2021).
[19] Xu Ma, Pengjie Wang, Hui Zhao, Shaoguo Liu, Chuhan Zhao, Wei Lin, Kuang-Chih Lee, Jian Xu, and Bo Zheng. 2021. Towards a Better Tradeoff between Effectiveness and Efficiency in Pre-Ranking: A Learnable Feature Selection based Approach. *arXiv preprint arXiv:2105.07706* (2021).
[20] Liang Pang, Jun Xu, Qingyao Ai, Yanyan Lan, Xueqi Cheng, and Jirong Wen. 2020. Setrank: Learning a permutation-invariant ranking model for information retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 499–508.
[21] 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.
[22] Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Practice on long sequential user behavior modeling for click-through rate prediction. In *KDD*. 2671–2679.
[23] Pi Qi, Xiaoqiang Zhu, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, and Kun Gai. 2020. Search-based User Interest Modeling with Lifelong Sequential Behavior Data for Click-Through Rate Prediction. In *CIKM*.
[24] Jiarui Qin, Weinan Zhang, Rong Su, Zhirong Liu, Weiwen Liu, Ruiming Tang, Xiuqiang He, and Yong Yu. 2021. Retrieval & Interaction Machine for Tabular Data Prediction. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1379–1389.
[25] Jiarui Qin, W. Zhang, Xin Wu, Jiarui Jin, Yuchen Fang, and Y. Yu. 2020. User Behavior Retrieval for Click-Through Rate Prediction. In *SIGIR*.
[26] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *ICDM*.
[27] Steffen Rendle. 2010. Factorization machines. In *ICDM*.
[28] Jiaxi Tang and Ke Wang. 2018. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *Proceedings of the 24th*

ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2289–2298.

[29] Jun Wang and Shuai Yuan. 2013. Real-time bidding: A new frontier of computational advertising research. In *CIKM Tutorial*.

[30] 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.

[31] Zhe Wang, Liqin Zhao, Biye Jiang, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2020. Cold: Towards the next generation of pre-ranking system. *arXiv preprint arXiv:2007.16122* (2020).

[32] Chen Xu, Quan Li, Junfeng Ge, Jinyang Gao, Xiaoyong Yang, Changhua Pei, Fei Sun, Jian Wu, Hanxiao Sun, and Wenwu Ou. 2020. Privileged features distillation at Taobao recommendations. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2590–2598.

[33] Zhixiang Xu, Matt Kusner, Kilian Weinberger, and Minmin Chen. 2013. Cost-sensitive tree of classifiers. In *International conference on machine learning*. PMLR, 133–141.

[34] Zhixiang Xu, Matt J Kusner, Kilian Q Weinberger, Minmin Chen, and Olivier Chapelle. 2014. Classifier cascades and trees for minimizing feature evaluation cost. *The Journal of Machine Learning Research* 15, 1 (2014), 2113–2144.

[35] Bowen Yuan, Jui-Yang Hsia, Meng-Yuan Yang, Hong Zhu, Chih-Yao Chang, Zhenhua Dong, and Chih-Jen Lin. 2019. Improving ad click prediction by considering non-displayed events. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 329–338.

[36] Weinan Zhang, Jiarui Qin, Wei Guo, Ruiming Tang, and Xiuqiang He. 2021. Deep Learning for Click-Through Rate Estimation. In *IJCAI*.

[37] Zhong Zhao, Yanmei Fu, Hanming Liang, Li Ma, Guangyao Zhao, and Hongwei Jiang. 2021. Distillation based Multi-task Learning: A Candidate Generation Model for Improving Reading Duration. *arXiv preprint arXiv:2102.07142* (2021).

[38] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *AAAI*, Vol. 33. 5941–5948.

[39] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *KDD*.

[40] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning Tree-based Deep Model for Recommender Systems. In *KDD*.

[41] Jieming Zhu, Jinyang Liu, Weiqi Li, Jincai Lai, Xiuqiang He, Liang Chen, and Zibin Zheng. 2020. Ensembled CTR Prediction via Knowledge Distillation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2941–2958.