# DenoiseRank: Learning to Rank by Diffusion Models

Anonymous ACL submission

## Abstract

Learning to rank (LTR) is one of the core tasks in NLP by supervised algorithmic techniques trained on a dataset with queries and their corresponding labeled relevant items. LTR models have made great progress, but all of them implement the algorithms from discriminative perspective. In this paper, we aim at addressing LTR from a novel perspective, i.e., by a deep generative model. Specifically, we propose a novel denoise rank model, DenoiseRank, which is a denoising diffusion-based model, for the LTR task. Our DenoiseRank noises the relevant labels in the diffusion process and denoises them on the query documents in the reverse process to accurately predict their distribution. Our model is the first to address LTR from generative perspective and is a diffusion method for LTR. Extensive experiments were conducted on benchmark datasets and the results demonstrated the effectiveness of the proposed DenoiseRank model. DenoiseRank provides a benchmark for generative LTR task.

## 1 Introduction

Learning to Rank (LTR) is one of the core tasks in NLP addressed by supervised machine learning techniques that are used to automatically construct ranking models. Its primary goal is to order items (documents, products, answers, etc.) in response to an input query so that the most relevant ones appear higher in a ranked list of results. LTR has been commonly used in a wide spectrum of NLP applications (Song and Ermon, 2019), e.g., recommender systems (Karatzoglou et al., 2013) and question-answering (Agarwal et al., 2012; Ji and Wang, 2013). A large number of LTR algorithms have been proposed and can be classified into two types, tree-based models (Lucchese et al., 2025; Chen et al., 2024; Ke et al., 2017; Burges et al., 2005) and neural-based LTR models (Gu et al., 2020; Pang et al., 2020; Argouarc'h et al., 2024; Jin et al., 2024; Jagerman et al., 2022; Cao et al., 2007). Among these models, those applying self-attention mechanism (Vaswani et al., 2017; Devlin et al., 2019) have typically achieved better performance, as demonstrated by SetRank (Pang et al., 2020) and DASALC (Qin et al., 2021).

Previous LTR models mainly adopt a discriminative approach to LTR (Gu et al., 2020; Pang et al., 2020; Chen et al., 2024; Ke et al., 2017; Argouarc'h et al., 2024; Qin et al., 2021), with no work considering it from a generative perspective. However, a large number of generative models have demonstrated their superior over both traditional and discriminative models for many NLP tasks such as machine translation (Vaswani et al., 2017), text classification (Devlin et al., 2019) and sentiment analysis (Yin and Zhong, 2024; Yuan et al., 2021). Of special interest to us, accordingly, is to explore whether generative models are capable of further enhancing the performance of LTR.

Generative models use the joint probability distribution $P(Y, X)$ (Harshvardhan et al., 2020) to model the data space. This makes them better to capture complex data distributions and more robust for classification and regression tasks (Sehwag et al., 2021). In this paper, we aim to estimate the full distribution of corresponding labels $Y$, i.e., $P(Y|D)$, given a ranked list of documents $D$ in response to an input query, from the perspective of a generative model. Denoising diffusion probabilistic models (DDPMs) (Ho et al., 2020; Nichol and Dhariwal, 2021) as one class of the most effective generative models have demonstrated significant potential in images and videos generation (Li et al., 2025; Rombach et al., 2022; Ho et al., 2022) in recent years. Due to their advantages of distribution generation, diversity representations, and training stability, we aim at investigating whether DDPMs can accurately recover the corresponding labels $Y$ for documents $D$ in response to a given query.

Specifically, we propose a novel denoising rank-

ing model, DenoiseRank [1], to address the LTR task. Our DenoiseRank is a DDPM (Sec.3) model and consists of three main components: diffusion process, reverse process and denoise neural network. In our proposed DenoiseRank, we first gradually inject Gaussian noise into the input corresponding labels $Y$ within a number of timesteps, (based on the Markov process (Sohl-Dickstein et al., 2015)), and consequently $Y$ becomes an isotropic Gaussian distribution. Then, noised labels are fed into the denoise neural network which will output denoised labels. We repeat this process on datasets and optimize the model with a special loss function, in order to learn the conditional probability distribution $P(Y|D)$. Note that the Denoise neural network mainly consists of a custom feedforward network and a Transformer-Encoder network. Finally, during the reverse process, the labels $Y$ are randomly sampled from Gaussian noise and fed into the well-trained Denoise neural network. After enough iterations, accurate labels are obtained. We demonstrate the effectiveness of the proposed DenoiseRank through extensive experimentation (Sec. 4) on the popular LTR dataset (Qin and Liu, 2013; Chapelle and Chang, 2011; Dato et al., 2016) and analyse its characteristics in ablation studies (Appendix. B). Experimental results show that DenoiseRank either outperforms or performs equivalently to recent LTR models (Pang et al., 2020; Qin et al., 2021). Our DenoiseRank can be utilised as a benchmark for future neural ranking models and can be extended to other sequence prediction tasks, such as "Multi-Objective Ranking (MOR) learning" (Gu et al., 2020).

The main contributions of this paper are as follows:

1. We study the problem of LTR from a generative perspective and introduce a diffusion model solution for the first time.
2. We propose a novel DenoiseRank model for LTR, which can be used as a benchmark for future generative neural ranking models.
3. Extensive experiments were conducted on benchmark datasets to demonstrate the effectiveness of DenoiseRank, and optimal performance was achieved against most metrics.
4. We introduce a new metric, $RSD@(K, M)$, to evaluate the ability of the model to produce diverse ranked lists. Our DenoiseRank has been proven to rank documents diversely.

[1]The source code of our model DenoiseRank is publicly downloadable from: https://github.com/yingwang93/DenoiseRank.

## 2 Definition of the LTR Task

We denote a training dataset as query set $Q = \{q_l\}_{l=1}^{L}$ and their corresponding document and label set $\{(D_l, Y_l) \mid q_l\}_{l=1}^{L}$, where $D_l$ is a document list that contains $n$ documents $D_{l,i}$ to be sorted, $i \in [1, n]$, $D_{l,i} \in \mathbb{R}^k$, and $k$ is the dimensions of the feature (Documents are represented and stored by embeddings/features). $Y_l$ is the label list for the corresponding documents list $D_l$, with $Y_{l,i} > 0$ indicating the document $D_{l,i}$ being relevant to the query and $Y_{l,i} = 0$ otherwise. $L$ denotes the total number of queries contained in the dataset.

The goal of LTR is to train a ranking function $f(Q, D)$, which can be used to accurately predict the relevance score of documents. We approximate the ranking function by training a model optimized by the loss $\mathcal{L}(f; Q, D)$. Different models for LTR have different loss formulations, which can be categorized into point-wise (Friedman, 2001), pair-wise (Burges, 2010), and list-wise (Cao et al., 2007) ones. Our DenoiseRank can be a point-wise, pair-wise or even list-wise model dependent on the loss equipped in the model. We make comparisons among the performance of DenoiseRank with point-wise, pair-wise and list-wise losses in the experiments; see Appendix. D.

## 3 Denoise Ranking Model

In this section, we first provide an overview of our proposed DenoiseRank model. We then detail our DenoiseRank model, including the diffusion process, the inverse process and the Denoise neural network. Finally we provide the training and inference algorithm and discuss the advance of our model compare to the others.

### 3.1 Overview of Our DenoiseRank

The overview of our proposed DenoiseRank is provided in Figure 1. As shown in the figure, our DenoiseRank is built based on DDPM and consists of three components: Diffusion Process, reverse process and denoise neural network. It first takes documents $D$ in response to a given query and corresponding labels $Y$ as input, then the Gaussian noise are injected into the labels through diffusion process. Next, noised labels and documents are fed into the denoise neural network to train and optimize them. Finally, in the reverse process, labels from randomly sampling and the corresponding

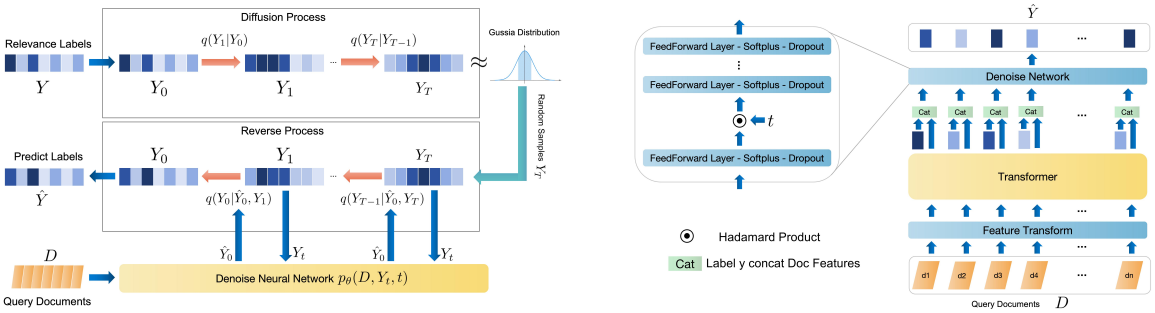


Figure 1: Left is an illustration of the diffusion process and reverse process in DenoiseRank. Right is the architecture of denoise network in DenoiseRank. FeedForward Layer is linear layer, softplus is activation, and dropout is dropout layer. Feature transform is applied when document feature input into transformer block.

documents are fed into the well-trained denoise neural network to predict the ground-truth labels step by step. Note that the Denoise Neural Network in DenoiseRank is implemented by a custom feedforward network and a Transformer-Encoder network, see Figure 1 on the right.

### 3.2 The DenoiseRank Model

As shown in Figure 1, given are a list of documents $D = d_1, d_2, ...dn$ in response to a query, and the corresponding list of feedback labels $Y$, $Y = y_1, y_2, ...y_n$. We hope that given $D$, denoiseRank will be able to predict the list of relevance labels $Y$ correctly through the reverse diffusion process. Our model is trained to approximate the distribution $p(Y|D)$. Our goal of training can be formulated as $p_\theta(Y_0|D) := \int p_\theta(Y_{0:T}|D)dY_{1:T}$, where $Y_0$ is the input labels $Y$ noised at timestep 0, $Y_0, ..., Y_T$ is noising data sampled from $Y_0 \sim q(Y_0)$. $p_\theta(Y_{0:T}|D)$ is the reverse diffusion process that we aim to learn the Gaussian transition from a Markov chain, the joint distribution formulated as follows:

$$p_\theta(Y_{0:T}|D) := p(Y_T)\prod_{t=1}^{T} p_\theta(Y_{t-1}|Y_t, D)\,, \tag{1}$$

$$p_\theta(Y_{t-1}|Y_t, D) := \mathcal{N}(Y_{t-1}; \mu_\theta(Y_t, t, D), \textstyle\sum_\theta(Y_t, t, D))\,. \tag{2}$$

For the forward process, we fixed the approximate posterior $q(Y_{1:T}|Y_0, D)$ to a Markov chain that gradually adds Gaussian noise into the labels:

$$q(Y_{1:T}|Y_0) := \prod_{t=1}^{T} q(Y_t|Y_{t-1})\,, \tag{3}$$

$$q(Y_t|Y_{t-1}) := \mathcal{N}(Y_t; \sqrt{1-\beta_t}Y_{t-1}, \beta_t\mathbf{I})\,, \tag{4}$$

where $\beta_1, ..., \beta_T$ is scheduled to control the process of noising data. Let $\alpha_t := 1 - \beta_t$ and $\overline{\alpha}_t := \prod_{s=1}^{t}\alpha_s$, then the posterior can be formulated as:

$$q(Y_t \mid Y_0) = \mathcal{N}(Y_t; \sqrt{\overline{\alpha}_t}Y_0, (1-\overline{\alpha}_t)\mathbf{I})\,. \tag{5}$$

Applying Bayesian theory, the prior probability can be formulated as:

$$q(Y_{t-1} \mid Y_t, Y_0) = \mathcal{N}(Y_{t-1}; \tilde{\mu}(Y_t, Y_0), \tilde{\beta}_t I)\,, \tag{6}$$

where $\tilde{\beta}_t$ and $\tilde{\mu}(Y_t, Y_0)$ are, respectively, as follows:

$$\tilde{\beta}_t := \frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t}\beta_t\,, \tag{7}$$

$$\tilde{\mu}(Y_t, Y_0) := \frac{\sqrt{\overline{\alpha}_{t-1}}\beta_t}{1-\overline{\alpha}_t}Y_0 + \frac{\sqrt{\alpha_t}(1-\overline{\alpha}_{t-1})}{1-\overline{\alpha}_t}Y_t\,. \tag{8}$$

We then train the model to optimize the variational lower bound (VLB) on negative log likelihood:

$$\begin{aligned}\mathcal{L} =& \mathbb{E}_q\left[-\log\, p(Y_T) - \sum_{t\geq 1}\log\frac{p_\theta(Y_{t-1}|Y_t, D)}{q(Y_t|Y_{t-1})}\right] \\ =& \mathbb{E}_q\left[-\log\frac{p_\theta(Y_{0:T}|D)}{q(Y_{1:T}|Y_0)}\right] \geq \mathbb{E}[-\,\log\, p_\theta(Y_0|D)]\,.\end{aligned} \tag{9}$$

Thus we can optimize the $\mathcal{L}$ with stochastic gradient descent during training. Further more, the VLB above can be rewritten as follows:

$$\begin{aligned}\mathcal{L} = \mathbb{E}_q\Bigg[&\underbrace{D_{\mathrm{KL}}\big(q(Y_T|Y_0) \parallel p(Y_T)\big)}_{\mathcal{L}_T} \\ &+ \sum_{t>1}\underbrace{D_{\mathrm{KL}}\big(q(Y_{t-1}|Y_t) \parallel p_\theta(Y_{t-1}|Y_t, D)\big)}_{\mathcal{L}_t} \\ &- \underbrace{\log p_\theta(Y_0|Y_1)}_{\mathcal{L}_0}\Bigg]\,.\end{aligned} \tag{10}$$

While $\mathcal{L}_T$ does depend on $\theta$, it will become zero when data being carefully noised enough and can be ignored when optimizing; $\mathcal{L}_0$ use to evaluate the reconstruct quality, but need complex calculate; $\mathcal{L}_t$ calculated by the KL-diveergence between posterior and model prediction.Following (Ho et al., 2020), we can simplely rewrite the $\mathcal{L}$ and replace by a mean-squared error (MSE) loss as follows:

$$\mathcal{L} = \mathbb{E}_{t,Y_0,\varepsilon}[||\,\epsilon - \epsilon_\theta(D, Y_t, t)\,||^2]\,, \tag{11}$$

where $\epsilon$ and $\epsilon_\theta(D, Y_t, t)$ are noise injected to $Y_t$ and nosie predicted from denoise neural network of DenoiseRank, respectively. According to (Nichol and Dhariwal, 2021), we can further predict $Y_0$ via:

$$Y_0 = \frac{1}{\sqrt{\alpha_t}} \left( Y_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon \right) . \quad (12)$$

Thus, the model can be trained to directly predict $Y_0$ and the loss reformulated as:

$$\mathcal{L} = \mathbb{E}_{t, Y_0, p_\theta}[|| \, Y_0 - p_\theta(D, Y_t, t) \, ||^2] , \quad (13)$$

where $p_\theta(\cdot)$ is the denoising model we aim to train. That means we can also predict $Y_0$ by the other losses of LTR. In this paper, we do ablation experiments on popular LTR losses to investigate the best performance of our models; see Appendix D.

### 3.3 Denoise Neural Network in DenoiseRank

As shown in Figure 1, Our model contains two components, the Self Attention Network and the Denoise Network. In recent years, neural LTR models employ self-attention mechanism and achieve significant performance advances. Since the self-attention mechanism can model the query documents context-aware, we make it as a part of our models. We choose transformer encoder as the basic network, formulated as follows:

$$\mathbf{H} = \mathbf{E}(\mathbf{d_1}, \mathbf{d_2}, ...., \mathbf{d_n}) , \quad (14)$$

where $\mathbf{d_1}, \mathbf{d_2}, ...., \mathbf{d_n}$ are embeddings of all documents in response to a single query, $\mathbf{H}$ are context-wise features of documents calculated by the transformer encoder $\mathbf{E}(\cdot)$. We use a standard transformer encoder architecture, where input document features are computed with advanced self-attention (Vaswani et al., 2017), followed by feed-forward networks and activation functions, and finally layer normalisation and dropout. In order to increase the model capacity and get the optimal performance, we try multi-head attention and multi-blocks architecture in training, $heads \in \{1, 2, 4, 8\}$ and $blocks \in \{3, 4, 5, 6\}$. Unlike the CV diffusion model, the denoising network is a feed-forward network instead of a U-net, formulated as:

$$\hat{Y}_0 = \text{FFN}(\mathbf{H}, Y_t, t) , \quad (15)$$

where t is the time step, $Y_t$ is a corresponding noised label at $t$. FFN$(\cdot)$ is a feedforward network with a multi-tiered architecture (Han et al., 2022). As shown in Figure 1 on the right, firstly, we input $Y_{t,i}$ and $\mathbf{H}_i$) into the first layer of the network after concatenating them to obtain the output $\mathbf{h}^{(l)} = \text{layer}(\mathbf{H}_i, Y_{t,i})$, where $\mathbf{H}_i$ denotes the $i$th document feature vector and $Y_{t,i}$ denotes the corresponding $i$th label. In each denoising layer, it goes through a linear layer, an activation function and a dropout layer respectively, then we do the embedding calculation to get $t_{\text{emb}}$ for timestep $t$. Finally, $\mathbf{h}^{(l)}$ is multiplied by $t_{emb}$ to get the current output. In our experiments, we try multiple denoise layer architectures with the number of layers n $\in \{2, 4, 6, 8\}$. The first layer formulated as:

$$\mathbf{h}^{(1)} = \text{Dropout}(\sigma_{\text{sp}}(\text{Linear}(\mathbf{H}_i, Y_{t,i}))) , \quad (16)$$

$$\mathbf{m}^{(1)} = \mathbf{h}^{(1)} \odot t_{emb} , \quad (17)$$

middle layer formulated as:

$$\mathbf{h}^{(j)} = \text{Dropout}(\sigma_{\text{sp}}(\text{Linear}(\mathbf{m}^{(j-1)}))) , \quad (18)$$

$$\mathbf{m}^{(j)} = \mathbf{h}^{(j)} \odot t_{\text{emb}} , \quad (19)$$

where $j$ denotes the $j$-th denoise layer of FFN$(\cdot)$, $j \in (1, n)$. The output layer is formulated as:

$$\mathbf{h}^{(n)} = \text{Dropout}(\sigma_{\text{sp}}(\text{Linear}(\mathbf{m}^{(n-1)}))) , \quad (20)$$

where $\sigma_{\text{sp}}$ is the softplus activation function, $\mathbf{m}^{(j-1)}$ is the output of the previous layer. The dimension of the last layer $\mathbf{h}^{(n)}$ is 5, each dimension corresponds to the weight of the relevance label, and the final weighted sum computes the predicted label $\hat{Y}_0$, where $\hat{Y}_0 \in [0, 4]$ and is denoted as:

$$\hat{Y}_0 = \mathbf{h}^{(n)} \odot (0, 1, 2, 3, 4) . \quad (21)$$

### 3.4 Diffusion and Training

In the diffusion process, the main task is to add noise to the labels. Specifically, given time steps $t$ and a noise scheduler $\beta$, the noised labels $Y_t$ are obtained. Our goal is to train an approximate model $p_\theta(\cdot)$ that is able to predict the true labels $y_0$ from the noisy labels $y_t$ at $t$ timesteps. We experimented with a variety of noise schedulers, see Appendix B.2. We set the maximum time step T to 1000, and the ablation experiments are referenced in Appendix B.1.

**Training.** We perform hundreds of epocs of training for DenoiseRank as follows. First, the timestep $t$ is randomly sampled for each query list. Based on the time step $t$ and the noise scheduler $\beta$, label vector $Y_0$ will gradually become Gaussian noise $Y_T$ at $T$ (Eq.5). Second, $Y_t$ and document list $D$ are fed into the model $p_\theta(\cdot)$, which will output the

**Algorithm 1:** Training

**Input** : Docs: $D = \{d_1, d_2, \ldots, d_n\}$
Truth labels: $Y_0 = \{y_1, \ldots, y_n\}$
Timesteps: $t \in [0, T]$
Noise schedule: $\beta_t$
Base model: $p_\theta(\cdot)$
Training epochs: $K$
**Output** : Well-trained Model: $p_\theta(\cdot)$
**for** epoch $\leftarrow 1$ **to** $K$ **do**
  $Y_t \leftarrow q(Y_t|Y_0)$ // Eq.(22)
  $\hat{Y}_0 \leftarrow p_\theta(D, Y_t, t)$ // Prediction
  $\mathcal{L}(\hat{Y}_0, Y_0)$ // Compute loss
  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$ // Update
**end**
**return** $p_\theta$

**Algorithm 2:** Inference

**Input** : Docs: $D = \{d_1, d_2, \ldots, d_n\}$
Noised labels: $Y_t \sim \mathcal{N}(0, \mathbf{I})$
Max Diffusion steps: $T$
Noise schedule: $\beta_t$
Trained model: $p_\theta(\cdot)$
**Output** : Denoised labels: $\hat{Y}_0$
**for** $t \leftarrow T$ 1 **do**
  $\hat{Y}_0 = p_\theta(D, Y_t, t)$ // Prediction
  $q(Y_{t-1}|\hat{Y}_0, Y_t) \rightarrow Y_{t-1}$ // Eq.(23)
**end**
**return** $\hat{Y}_0$

denoised labels $\hat{Y}_0$. Finally, the loss function is calculated and the model parameters are adjusted. The above process is repeated until the target epoc and the model converge; see Algorithm 1. Our goal is to make $\hat{Y}_0$ accurately equal to $Y_0$ through extensive training. Training process can be briefly formulated as:

$$Y_t \leftarrow q(Y_t|Y_0), \hat{Y}_0 = p_\theta(D, Y_t, t)\,. \tag{22}$$

### 3.5 Reverse and Inferencing

The main task in the reverse process is labels denoising. Starting from a given noisy label $Y_t$ and time step $t$, denoising yields the next time step labels $Y_{t-1}$. The goal of inference is to obtain the predicted labels $\hat{Y}_0$ that can correctly rank the query document $D$. We need to predict $\hat{Y}_0$ at each step and then compute $Y_{t-1}$, see Eq.2,12.

**Inference.** First, the noised labels $Y_t$ is sampled from the Gaussian distribution $\mathcal{N}(0, I)$ in the max timestep $T$. Then, $Y_t$ and the document list $D$ are fed into the model $P_\theta$ and the denoised labels $\hat{Y}_0$ are output. According to equation 2,12, we can approximate $Y_{t-1}$ from $Y_t$ and $\hat{Y}_0$. Finally, repeat the above denoising process until $t = 0$, and obtain $y_0$, the predicted relevance labels, see Algorithm 2. Inferencing process can be brief formulated as:

$$\hat{Y}_0 = p_\theta(D, Y_t, t), \;\; Y_{t-1} \leftarrow q(Y_{t-1}|\hat{Y}_0, Y_t)\,. \tag{23}$$

### 3.6 Discussions

Our DenoiseRank differs from previous diffusion based generative models in at least the following aspects: (1) Our model is the first one to address LTR task by generative diffusion model to accurately rank by fitting the conditional distribution $P(Y|D)$. (2) Previous diffusion models use the U-net as the denoise network (especially in CV areas (Rombach et al., 2022; Ho et al., 2022)), but we use a novel network consists of feedforward network and transformer. (3) Compare to the diffusion models in sequence recommendation(Li et al., 2023), we utilise the transformer to calculate context-wise features and denoise labels by the feedforward network.

## 4 Experiments

### 4.1 Research Questions

The remainder of this paper is guided by the following research questions: (1) Does our DenoiseRank outperform state-of-the-art LTR models? (2) How do the hyperparameters and each design choice of the DenoiseRank affect its performance? (3) What about the diversity of the rank result of DenoiseRank compared to other LTR models ?

### 4.2 Datasets and Metric

**Datasets.** Experiments were conducted on three famous datasets for LTR, including the Microsoft Web30k (Qin and Liu, 2013), Yahoo! LETOR (Chapelle and Chang, 2011), and Istella LETOR (Dato et al., 2016). The queries and documents in these datasets were obtained from real search engines. Each dataset contains a large number of query-documents. In addition, labels are represented on a 5-level scale from 0 to 4 (the most relevant). Each document is represented by multidimensional features such as BM25 score of the page section. We use pre-partitioned train/test data from each dataset for training and testing.

**Metrics.** Normalised Discounted Cumulative Gain (NDCG) is used to evaluate all models' performance. We report the values of the metric at positions 1, 5, and 10, i.e., NDCG@1, NDCG@5, and NDCG@10. It is worth noting that we use

NDCG@10 as the evaluation criterion to select the best model. Experimental results against other metrics are shown in Appendix E.

### 4.3 Comparison Models

We evaluate DenoiseRank against the advanced and recent baselines; see Table 1. There are two categories of baselines, tree-based and neural-based.

**Tree-based.** $\mathrm{MART}_{\mathrm{GBM}}$ (Ke et al., 2017) is one of the LambdaMART (Wu et al., 2010) implementations provided by Microsoft and is one of the best tree-based methods. $\mathrm{MART}_{\mathrm{RankLib}}$ is part of the RankLib library and is another implementation of LambdaMART (Wu et al., 2010).

**Neural-based.** DLCM (Ai et al., 2018) employs the RNN network to capture the local ranking context and is trained with an attention-based loss function, which makes it more effective. $\mathrm{SetRank}_{\mathrm{re}}$ (Pang et al., 2020) is an improved version of SetRank, where documents are reranked before they are entered. NeuralNDCG (Pobrotyn and Białobrzeski, 2021) is a neural LTR model that addresses the mismatch between optimisation objective and evaluation criterion of traditional model by a novel loss function. DASALC (Qin et al., 2021) is a baseline provided by Google, which is based on the self-attention and multiple optimisation components to outperform even the tree-based models. Rankformer (Buyl et al., 2023) is one of the recent baselines, which uses list-wise labels to capture contextual information and also uses an novel implicit feedback component. However, implicit feedback component of Rankformer is not enabled in the experiments as it is not part of our study.

We report evaluation results for $\mathrm{MART}_{\mathrm{GBM}}$, $\mathrm{MART}_{\mathrm{RankLib}}$, DLCM, and Rankformer in the same runtime environment as DenoiseRank. The results for NeuralNDCG, $\mathrm{SetRank}_{\mathrm{re}}$ and DASALC are those reported in their original papers. The DenoiseRank model parameters and training setups are presented in Appendix A.

### 4.4 Comparison Result

The results are shown in Table 1, and we conclude that 1) DenoiseRank achieves better or competitive performance compared to other discriminative models, especially on the Web30k and Yahoo datasets, which proves the effectiveness of our models. 2) Compared to advanced tree-based models, DenoiseRank consistently leads in performance on the Web30k dataset; on the Yahoo dataset, NDCG@10 performs better, and @1 and @5 are closely. 3) DenoiseRank achieves an overall lead over advanced neural LTR baselines. 4) DenoiseRank performs better on the Web30k and Yahoo datasets, and @10 achieves a lead on the Istella dataset compared to other neural LTR models.

### 4.5 Design Choice and Hyperparameters

We carry out ablation studies on DenoiseRank and investigate the best design choice of it on different datasets. The configurations of DenoiseRank are in Table 2. We further introduce hyperparameters and design choice in Appendix. A and B.

**Noise schedule.** Noise schedule is the dynamic parameter $\overline{\alpha}_t$ which controls the noise ratio each step during diffusion. We choose 4 types of schedule to evaluate our DenoiseRank, including Linear, TruncatedLinear, Sqrt and Cosine. As shown in Figure 6 and Table 5, we summarize that: (1) Difference schedule affect the performance of our DenoiseRank (2) TruncatedLinear is the better choice because it results in more reliable performance. More introduction is found in Appendix B.2.

**Max diffusion timesteps.** Diffusion timesteps $T$ control the speed of noising corresponding labels, noise becomes more subtle as max timestep increases. We choose 5 max timesteps, including 1,000, 800, 600, 400 and 200. The result shown in Figure 5 and Table 4 denotes that: (1) The performance is benefitial from carefully nosing after increasing the max diffusion timesteps (2) Training on different datasets need different diffusion timesteps, for example, the best choice is T=600 on istella datasets. Appendix B.1 shows the detailed discussion.

**The number of denoise network layers.** Denoise network is an important part of our model to predict ground-truth labels, and its layer count may affect the ranking result of DenoiseRank. Thus we choose 4 types of layers to investigate it, and results are shown in Figure 7 and Table 6 denotes that: (1) The number of Layers significantly affects the performance of DenoiseRank (2) the best choice is 2, 4, 4 on Web30K, Yahoo!, Istella datasets respectively. (See Appendix B.3)

**Self attentions.** The performance of DenoiseRank is significantly improved as Transformer becomes a part of our model, as shown in Table 7 and Figure 8. The self-attention mechanism makes it

Table 1: NDCG@K performance comparison on benchmark datasets. Best performance is bolded. $^*$ and $^\dagger$ denote statistically significant improvements over the best tree-based and neural models respectively. Last row is relative difference of DenoiseRank over the best comparison models.

| Method | Microsoft Web30K | | | Yahoo! | | | Istella | | |
|---|---|---|---|---|---|---|---|---|---|
| | @1 | @5 | @10 | @1 | @5 | @10 | @1 | @5 | @10 |
| $\text{MART}_{\text{RL}}$ | 45.35 | 44.59 | 46.46 | 68.52 | 70.27 | 74.58 | 65.71 | 61.18 | 65.91 |
| $\text{MART}_{\text{GBM}}$ | 50.73 | 49.67 | 51.46 | **71.90** | **74.20** | 78.01 | **74.95** | **71.20** | **76.05** |
| DLCM | 46.31 | 45.01 | 46.90 | 67.71 | 69.91 | 74.29 | 65.57 | 61.95 | 66.80 |
| $\text{SetRank}_{re}$ | 45.91 | 45.15 | 46.96 | 68.22 | 70.29 | 74.53 | 67.60 | 63.45 | 68.34 |
| NeuralNDCG | – | 51.45 | 53.49 | – | 66.02 | 71.02 | – | – | – |
| DASALC | 50.95 | 50.92 | 52.88 | 70.98 | 73.76 | 77.66 | 72.77 | 70.06 | 75.30 |
| Rankformer | 49.61 | 49.23 | 51.27 | 70.18 | 73.02 | 77.58 | 68.11 | 68.20 | 75.03 |
| DenoiseRank | **51.87**$^{*\dagger}$ | **52.52**$^{*\dagger}$ | **54.60**$^{*\dagger}$ | 71.37$^\dagger$ | 74.06$^\dagger$ | **78.42**$^{*\dagger}$ | 70.00 | 69.30 | 75.82$^\dagger$ |
| *Relative Diff* | (+1.8%) | (+2.1%) | (+2.1%) | (-0.7%) | (-1.6%) | (+0.5%) | (-7.1%) | (-2.7%) | (-1.4%) |

*Note*: $\text{MART}_{\text{RL}}$ and $\text{MART}_{\text{GBM}}$ denotes Ranklib and GBM version of the LambdaMART respectively.

Table 2: Recommended configurations of DenoiseRank from ablation study.

| **Design** | **Web30K** | **Yahoo** | **Istella** |
|---|---|---|---|
| Noise Schedule | TruncL | TruncL | TruncL |
| Max Diffusion Steps | 1000 | 1000 | 600 |
| Denoising Layers | 2 | 4 | 8 |
| Self-Attention | ✓ | ✓ | ✓ |
| Loss Function | ListNet | MSE | MSE |

*Note*: All configurations use the same base architecture. ✓ denotes the inclusion of self-attention modules. TruncL denotes Truncate Linear schedule.

passible to recalculate documents feature context-wise, and DenoiseRank can learn about the relation between documents. (See Appendix B.4)

**Learning Rate.** We train our model with different learning rates $\in 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$, and user AdamW optimizer. In most situations, $10^{-4}$ is a good choice, while training on the web30k dataset, performance is a little better when learning rate is $10^{-3}$.

### 4.6 Diversity of Ranking Results

As our DenoiseRank is a diffusion based model, it is able to produce diverse ranked lists of documents in response to the same query while may still keep the high standard of the NDCG Performance, compared to traditional LTR models that always produce the same ranked list of documents in response to the same query. Such diverse ranked lists of documents allow documents with the same ground-truth labels have the same chances to be ranked in the top-K position in the ranked lists.

In this study, we verify the diversity of the ranking sequences produced by DenoiseRank. In the inference stage, $Y_T$, which is sampled at random from Gaussian noise, introduces uncertainty when making predictions. To address this, we perform multiple inferences for the same query and analyze the ranking diversity. We introduce the RSD@(K,M) metric (see Eq. 24), which denotes the number of different sequences on the top $K$ ranking of the same query among $M$ times inference. (Note that RSD (Ranking Sequence Diversity) is absolutely different to traditional diversity metrics; see Appendix. C) Using $K \in \{1, 5, 10, 20\}, M = 10$, we compare RSD@(K,M) of DenoiseRank to Rankformer, which has a similar architecture to ours but addresses LTR task from a traditional discriminative perspective without uncertainty.

As shown in Figure 2, the ranking results inferred by DenoiseRank are diverse, while those inferred by Rankformer remain singleton. Referring to Table 8, we find that the ranked lists generated by DenoiseRank are various in response to the same test query 10 times while the NDCG metric remains excellent. In contrast, Rankformer keeps the same ranked list and the NDCG is also unchanged. More detailed analysis can be found in the Appendix. C.

## 5 Related Work

Over the last 20 years, LTR has usually been studied from the perspective of discriminative methods (Friedman, 2001; Burges, 2010; Cao et al., 2007). These studies can be categorised as tree-based and neural network-based.

Tree-based models show competitive performance (Lucchese et al., 2025), e.g., LambdaMART (Ke et al., 2017; Wu et al., 2010), but poor performance when data is sparse and not easy

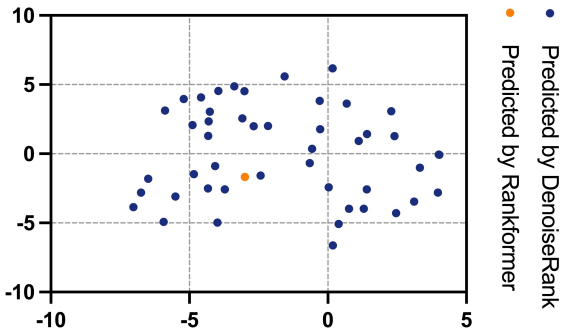


Figure 2: A t-SNE plot shows the diverse ranking sequences on the top 20 predicted in the inference stage of a single query randomly selected. The blue points denote the ranking sequences inferred by DenoiseRank using 100 different $Y_T$ values from Gaussian noise. The orange yellow represents the other sequences predicted by Rankformer in 100 attempts. Testing was conducted on the MS Web30K dataset.

to be scalable (Qin et al., 2021). Other studies advocate the use of neural networks to train LTR models, e.g., RankNet (Burges et al., 2005). The advantage of neural networks based models is that they are easy to be scalable, but are prone to be overfitting, and the feed-forward layer treats documents in isolation and ignore documents' correlation. Some studies introduce attention mechanisms such as RNN and attention to LTR and achieve significant performance, e.g., SetRank (Pang et al., 2020) and DASALC (Qin et al., 2021). However, all these models are discriminative. In this work, we introduce generative method to LTR, using high-capacity networks and self-attention mechanisms.

In addition to model design, there are LTR studies using unbiased estimation (Luo et al., 2024) and innovative loss functions (Pobrotyn and Białobrzeski, 2021). In recent years, there have been studies on obtaining implicit feedback and reducing the bias in realistic feedback scores by designing click models, e.g. Rankformer (Buyl et al., 2023), InfoRank (Jin et al., 2024). Unbiased LTR achieved significant results, but this is far different from our study, which focuses on generative LTR model design. Loss functions have been the focus of LTR research, and traditionally there are three types, point-wise (Friedman, 2001), pair-wise (Burges, 2010) and list-wise (Cao et al., 2007). In recent years, there have also been studies proposing loss functions such as ApproxNDCG (Bruch et al., 2019), LambdaLoss (Jagerman et al., 2022), NeuralNDCG (Pobrotyn and Białobrzeski, 2021). In this study, the MSE loss is used, which is more suitable for diffusion models.

Traditional LTR studies have used discriminative models, which are also commonly used in classification and regression studies. Generative models, represented by VAE (Kingma et al., 2013), GAN (Goodfellow et al., 2014), etc., can model the data distribution (Zhou et al., 2023; Liu et al., 2021) and better solve the problems of data sparsity, overfitting and noise sensitivity, etc. In recent years, there have been researches on the use of generative methods in classification and regression studies (Han et al., 2022). Unfortunately, there is no research on LTR using generative models. Diffusion Models (Sohl-Dickstein et al., 2015) have shown great potential in recent years, with models represented by DDPMs (Ho et al., 2020; Nichol and Dhariwal, 2021) being applied to multimodal generation (Zhang et al., 2025; Song and Ermon, 2019; Ho et al., 2022; Song and Ermon, 2020). Some studies have applied diffusion models to recommender systems (Li et al., 2023), but not yet to LTR. This study is the first to address LTR through a generative approach and lays the foundation for subsequent studies on LTR through generative models.

## 6 Conclusion

This study aims at addressing the LTR task. Previous studies address the LTR task from a discriminative perspective, do not modeling the data well and ignore the latent relationships among documents. In contrast, to our knowledge we are the first to address the task via a generative model. We propose the novel DenoiseRank model, which is a diffusion-based LTR model for the task. Specifically, our DenoiseRank noises the relevant labels in the diffusion process and denoises them on the query documents in the reverse process to accurately predict the labels of the documents in response to the input query. Experimental results demonstrate the advantages of our DenoiseRank, including excellent retrieval performance and diversity of ranked lists. We also propose a new evaluation metric to evaluate the performance of the generative model in terms of the diversity of the ranked lists. We believe that our work makes an important contribution to advance research on neural-based LTR models and paves the way for future research into generative models for LTR. As to future work, we intend to include interactive data such as clicks on documents into the model to improve the performance.

## 7 Limitations

This study advances LTR task from a generative perspective but has notable limitations. Interactive feedback information such as clicks received from human-beings is not included in the training datasets, but such information does help to improve the performance of the model. We inject Gaussian noises during the diffusion process in our model. However, other kinds of noise distributions rather than Gaussian may be much more helpful to model the data for the diffusion process.

## References


Arvind Agarwal, Hema Raghavan, Karthik Subbian, Prem Melville, Richard D Lawrence, David C Gondek, and James Fan. 2012. Learning to rank for robust question answering. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 833–842.

Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. 2018. Learning a deep listwise context model for ranking refinement. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 135–144.

Elouan Argouarc'h, François Desbouvries, Eric Barat, and Eiji Kawasaki. 2024. Generative vs. discriminative modeling under the lens of uncertainty quantification. *arXiv preprint arXiv:2406.09172*.

Sebastian Bruch, Masrour Zoghi, Michael Bendersky, and Marc Najork. 2019. Revisiting approximate metric optimization in the age of deep neural networks. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, pages 1241–1244.

Chris 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*, pages 89–96.

Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81.

Maarten Buyl, Paul Missault, and Pierre-Antoine Sondag. 2023. Rankformer: Listwise learning-to-rank using listwide labels. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3762–3773.

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

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

Qi Chen, Xiubo Geng, Corby Rosset, Carolyn Buractaon, Jingwen Lu, Tao Shen, Kun Zhou, Chenyan Xiong, Yeyun Gong, Paul Bennett, and 1 others. 2024. Ms marco web search: A large-scale information-rich web dataset with millions of real click labels. In *Companion Proceedings of the ACM Web Conference 2024*, pages 292–301.

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):1–31.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems*, 27.

Yulong Gu, Zhuoye Ding, Shuaiqiang Wang, Lixin Zou, Yiding Liu, and Dawei Yin. 2020. Deep multifaceted transformers for multi-objective ranking in large-scale e-commerce recommender systems. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 2493–2500.

Xizewen Han, Huangjie Zheng, and Mingyuan Zhou. 2022. Card: Classification and regression diffusion models. *Advances in Neural Information Processing Systems*, 35:18100–18115.

GM Harshvardhan, Mahendra Kumar Gourisaria, Manjusha Pandey, and Siddharth Swarup Rautaray. 2020. A comprehensive survey and analysis of generative models in machine learning. *Computer Science Review*, 38:100285.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. 2022. Video diffusion models. *Advances in Neural Information Processing Systems*, 35:8633–8646.

Rolf Jagerman, Zhen Qin, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2022. On optimizing top-k metrics for neural ranking models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2303–2307.

Zongcheng Ji and Bin Wang. 2013. Learning to rank for question routing in community question answering. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2363–2368.

Jiarui Jin, Zexue He, Mengyue Yang, Weinan Zhang, Yong Yu, Jun Wang, and Julian McAuley. 2024. Inforank: Unbiased learning-to-rank via conditional mutual information minimization. In *Proceedings of the ACM Web Conference 2024*, pages 1350–1361.

Alexandros Karatzoglou, Linas Baltrunas, and Yue Shi. 2013. Learning to rank for recommender systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 493–494.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.

Diederik P Kingma, Max Welling, and 1 others. 2013. Auto-encoding variational bayes.

Jia Li, Lijie Hu, Jingfeng Zhang, Tianhang Zheng, Hua Zhang, and Di Wang. 2025. Fair text-to-image diffusion via fair mapping. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 26256–26264.

Zihao Li, Aixin Sun, and Chenliang Li. 2023. Diffurec: A diffusion model for sequential recommendation. *ACM Transactions on Information Systems*, 42(3):1–28.

Shiao Liu, Xingyu Zhou, Yuling Jiao, and Jian Huang. 2021. Wasserstein generative learning of conditional distribution. *arXiv preprint arXiv:2112.10039*.

Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, and Alberto Veneri. 2025. Explainable, effective, and efficient learning-to-rank models using ilmart. *ACM Transactions on Information Systems*.

Dan Luo, Lixin Zou, Qingyao Ai, Zhiyu Chen, Chenliang Li, Dawei Yin, and Brian D Davison. 2024. Unbiased learning-to-rank needs unconfounded propensity estimation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1535–1545.

Alexander Quinn Nichol and Prafulla Dhariwal. 2021. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR.

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*, pages 499–508.

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

Tao Qin and Tie-Yan Liu. 2013. Introducing letor 4.0 datasets. *arXiv preprint arXiv:1306.2597*.

Tao Qin, Tie-Yan Liu, and Hang Li. 2010. A general approximation framework for direct optimization of information retrieval measures. *Information retrieval*, 13:375–397.

Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2021. Are neural rankers still outperformed by gradient boosted decision trees? In *International conference on learning representations*.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695.

Vikash Sehwag, Saeed Mahloujifar, Tinashe Handina, Sihui Dai, Chong Xiang, Mung Chiang, and Prateek Mittal. 2021. Robust learning meets generative models: Can proxy distributions improve adversarial robustness? *arXiv preprint arXiv:2104.09425*.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. pmlr.

Yang Song and Stefano Ermon. 2019. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32.

Yang Song and Stefano Ermon. 2020. Improved techniques for training score-based generative models. *Advances in neural information processing systems*, 33:12438–12448.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2018. The lambdaloss framework for ranking metric optimization. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 1313–1322.

Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Information Retrieval*, 13:254–270.

Shuo Yin and Guoqiang Zhong. 2024. Textgt: A double-view graph transformer on text for aspect-based sentiment analysis. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 19404–19412.

Ziqi Yuan, Wei Li, Hua Xu, and Wenmeng Yu. 2021. Transformer-based feature reconstruction network for robust multimodal sentiment analysis. In *Proceedings of the 29th ACM international conference on multimedia*, pages 4400–4407.

Jiaqing Zhang, Mingxiang Cao, Xue Yang, Kai Jiang, and Yunsong Li. 2025. Diffclip: Few-shot language-driven multimodal classifier. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 22443–22451.

Xingyu Zhou, Yuling Jiao, Jin Liu, and Jian Huang. 2023. A deep generative approach to conditional sampling. *Journal of the American Statistical Association*, 118(543):1837–1848.

## A Hyperparameters

**DenoiseRank hyperparameters** We set the hyperparameters of the model, including: dropout $\in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$, denoise net hiddenSize $\in \{64, 128, 256, 512\}$ for linearLayer, denoise Layers $\in \{2, 4, 6, 8\}$, transformer blocks $\in \{3, 4, 5, 6\}$, self-attention heads $\in \{1, 2, 4, 5, 8\}$. In the diffusion configuration, noise schedule $\in \{TruncatedLinear, Linear, Cosine, Sqrt\}$, max diffusion timesteps $\in \{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000\}$. We have experimented and tuned different datasets and the results can be found in Tables 4 , 5. We introduce the experiments result in the following paragraph and Appendix. B.

**DenoiseRank training settings** First, we use the AdamW optimiser and set LearningRate $\in \{0.1, 0.01, 0.001, 0.0001\}$ and batchsize=128. Second, the training epoc is set to 200. For training, we evaluate every 10 epochs on the test dataset using NDCG@10 as a benchmark. Store the optimal model and the results evaluated by the metrics. Finally, we run the model on a single NVIDIA GeForce RTX 3090. The performance with different learning rate is shown in Figure 3 and Table 3. We find that:

1. A learning rate of $10^{-3}$ is optimal for training DenoiseRank on the MS Web30K dataset, with $10^{-4}$ being the next best option.

2. For the Yahoo! and Istella datasets, $10^{-4}$ is the better learning rate with which to train DenoiseRank; $10^{-3}$ provides an approximate result.

3. In most situations, learning rates of $10^{-1}$ and $10^{-2}$ result in poor performance, which suggests that our DenoiseRank needs subtle optimisation.

**Convergence** DenoiseRank is a new LTR model consider the task from generative perspective, combine with Diffusion model, which need a lot of timesteps in diffusion and reverse process. Thus we investigate the convergence speed on training process in experiments on the runtime enviroment we mention above. We alse compare our model to Rankformer, which has the similiar model architecture. We use the best hyperparameter and design choice of them and which can make a best ranking performence.

As shown in Figure. 4, we summarize that:

1. On the MS Web30K datasets, both DenoiseRank and Rankformer can converge after 50 epocs of training.

2. On the Yahoo! datasets, DenoiseRank converge after 130 epocs, while rankformer is more slow and coverage after 200 epoc.

3. We speculate it is because: first, documents in Yahoo! have higher dimension of feature (700 dimensions per document) than those in MS Web30K (136 dimensions per document), so model need more epoc to fit them; second, our DenoiseRank address LTR task from generative perspective and comine with Diffusion model, it can fit high dimensional feature more effective

## B Ablation Study

We have done ablation studies on DenoiseRank, including maximum diffusion timesteps, noise scheduler, the number of denoise network layers and effectiveness of self-attentions; see Table [4,5,6, 7].

Table 3: NDCG@K performance of DenoiseRank with different learning rates on Microsoft Web30K, Yahoo!, and Istella datasets. Best performance per column in bold.

| LR | Web30K | | | Yahoo! | | | Istella | | |
|---|---|---|---|---|---|---|---|---|---|
| | @1 | @5 | @10 | @1 | @5 | @10 | @1 | @5 | @10 |
| 0.1 | 47.32 | 48.27 | 50.42 | 39.17 | 49.34 | 57.84 | 5.70 | 7.38 | 9.87 |
| 0.01 | 46.48 | 46.15 | 47.89 | 41.31 | 51.24 | 59.34 | 43.39 | 43.80 | 48.89 |
| 0.001 | **51.87** | **52.52** | **54.60** | 69.51 | 72.18 | 76.65 | 68.42 | 68.20 | 74.85 |
| 0.0001 | 51.01 | 51.86 | 54.10 | **71.27** | **73.96** | **78.40** | **69.14** | **69.09** | **75.63** |

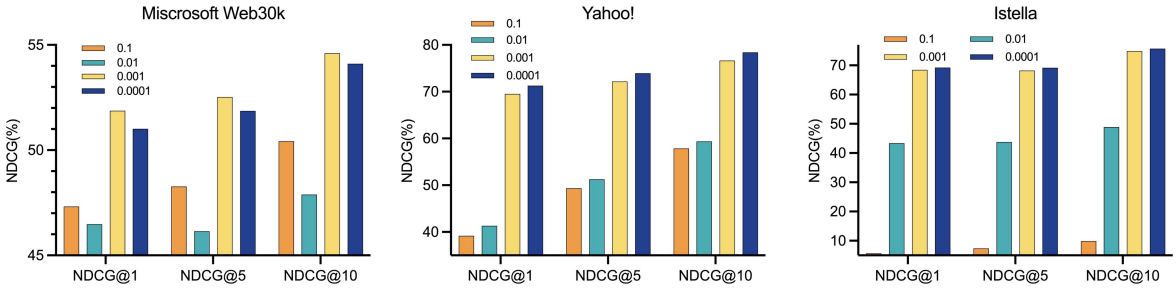


Figure 3: NDCG@K of DenoiseRank with different learning rates on Miscrosoft Web30k, Yahoo! and Istella datasets.

### B.1 Maximum Diffusion Timesteps

In the diffusion process, the maximum diffusion timestep ($T$) refers to how many iterations are required to change from the original sample to the Gaussian noise. The larger the maximum time step, the smaller the sample change per iteration, and conversely, the larger the sample change. The original DDPM uses a maximum time step of $T = 1000$ (Ho et al., 2020)(Nichol and Dhariwal, 2021). As different maximum time steps can have an impact on the model performance (Li et al., 2023). We evaluate the model performance in the case of time step $t = [1000, 800, 600, 400, 200]$ respectively. As shown in Figure. 5 and Table 4:

1. The model performs better as the maximum time step increases, suggesting that slow noise addition is more beneficial for model learning.

2. The model performance is more dependent on long time steps on the web30k dataset.

3. The performance of the model is not always optimal for long time steps. The model performs optimally on the Istella dataset at $T = 600$. This means that we can reduce the time step appropriately to speed up training and inference.

### B.2 Noise Scheduler

The noise scheduler is the way in which the $\overline{\alpha}_t$ changes during diffusion, where $\overline{\alpha}_t := \prod_{s=1}^{t} \alpha_s$, see eq. 4. The rate of change of $\overline{\alpha}_t$ varies in different noise-adding schemes, e.g., truncated linear has a large change before $\frac{T}{2}$ and a small change after $\frac{T}{2}$, whereas Cosine has a relatively balanced change(Li et al., 2023).

In order to evaluate the performance of DenoiseRank under different noise schedules, we try different choices, including Truncated Linear, Linear, Cosine, Sqrt. The results (see Figure.6 and Table.5) show that:

1. TruncatedLinear performs better than the other schedules overall, but there is not a big difference.

2. the performance of the different noise schedules varies greatly on the web30k datasets, i.e. TruncatedLinear > Sqrt > Linear > Cosine.

3. on the yahoo and istella datasets, there is not much difference in the reliability of the ranking, and on the istella dataset, sqrt even performs slightly better than TruncatedLinear.

### B.3 The Number of Denoise Network Layers

As shown in Figure. 1 on the right, the denoising network of DenoiseRank is a feed-forward archi-

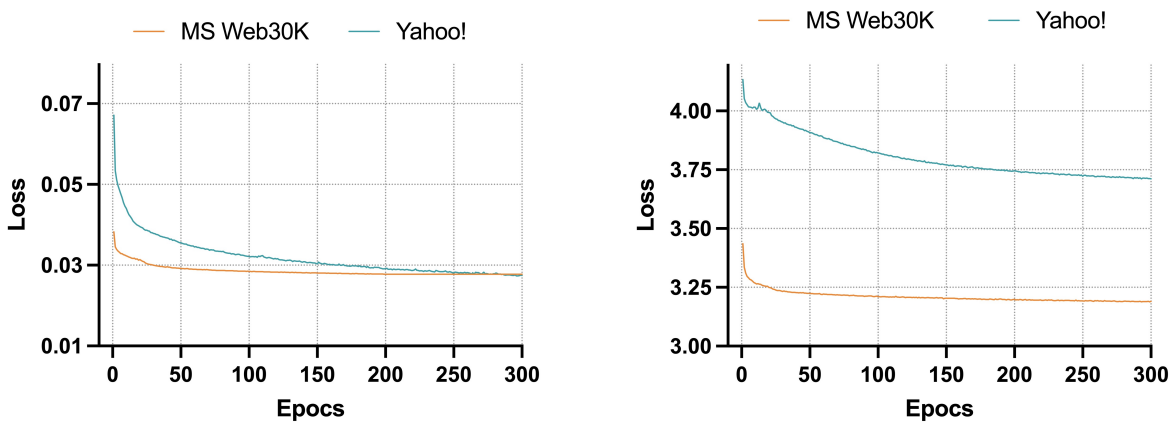


Figure 4: Curve of training loss of DenoiseRank(Left) and Rankformer(Right) on MS Web30K and Yahoo! datasets among 300 epocs.

Table 4: NDCG@K performance with different diffusion timesteps on Microsoft Web30K, Yahoo!, and Istella datasets. Best performance per column in bold.

| Timesteps | Web30K | | | Yahoo! | | | Istella | | |
|---|---|---|---|---|---|---|---|---|---|
| | @1 | @5 | @10 | @1 | @5 | @10 | @1 | @5 | @10 |
| 1000 | **51.87** | **52.52** | **54.60** | **71.27** | 73.96 | **78.40** | 69.14 | 69.09 | 75.63 |
| 800 | 51.49 | 52.03 | 54.10 | 70.84 | 73.70 | 78.22 | 69.25 | 69.13 | 75.67 |
| 600 | 50.90 | 51.58 | 53.53 | 70.82 | 73.74 | 78.25 | 69.09 | **69.21** | **75.68** |
| 400 | 50.52 | 49.82 | 51.66 | 70.87 | **73.99** | 78.35 | **69.47** | 68.97 | 75.52 |
| 200 | 49.87 | 49.33 | 51.11 | 70.71 | 73.82 | 78.23 | 69.41 | 68.77 | 75.35 |

tecture. The input and output layers of the denoising network are required, and the hidden layers in between can be dynamically adjusted (Han et al., 2022). Different hidden layers can affect the performance of the model. To obtain the optimal model structure, we investigate the effect of different layers of the denoising network. The number of layers includes $[2, 4, 6, 8]$, e.g. $layer = 2$ means that only the input and output layers are included and there is no hidden layer.

The results (see Figure.7 and Table.6) show that:

1. There is a significant difference between different layers on model performance.

2. On the web30k dataset, layers=2 performs the best, followed by layers=4, and the performance decreases instead as the layers increase.

3. On the Yahoo dataset, the model performs significantly better than 6 and 8 when the layers are 2 and 4.

4. On the istella dataset, the number of layers has no significant effect on model performance.

### B.4 Self Attentions

In recent studies on learning-to-rank (Pang et al., 2020)(Qin et al., 2021)(Buyl et al., 2023), the self-attention mechanism has been shown to significantly improve ranking results. To evaluate the effectiveness of self-attention (SA) in DenoiseRank, we conducted experiments with and without Transformer, and the results (see Figure.8 and Table.7) show that SA significantly improves the model performance, especially on the MS Web30k and Istella datasets.

## C Diversity

In inference stage, DenoiseRank randomly samples $Y_T$ from Gaussian noise, causing uncertainty to result of the ranking. Different Gaussian noise labels $Y_T$ may have a different ranking sequence. Compared to DenoiseRank, traditional LTR models may have been trained to rank certainly in various attempts given the same input document features.

In real-world information retrieval, the diverse ranked list of items in different search sceneries can be meaningful. In some situations, e.g., shopping retrieval on the e-Commerce website, we want items with the same relevance scores to have a fair chance to rank higher. Unfortunately, previous LTR

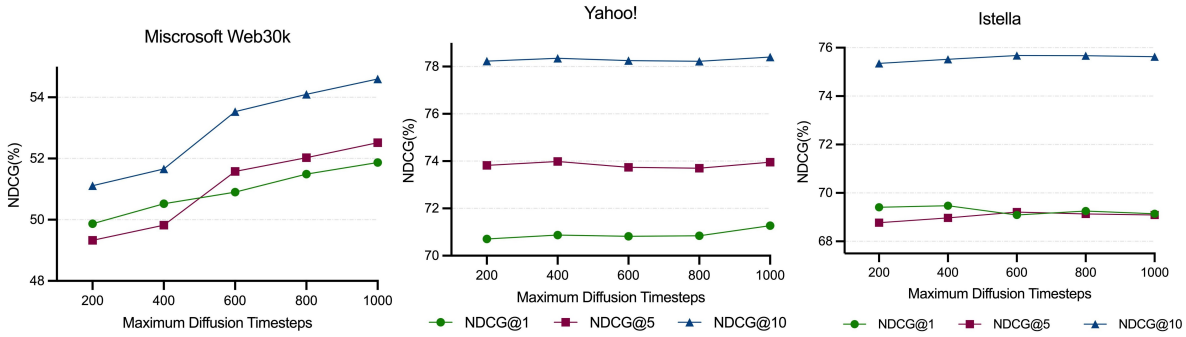


Figure 5: NDCG@K of DenoiseRank at different noise schedule on Miscrosoft Web30k, Yahoo! and Istella datasets.

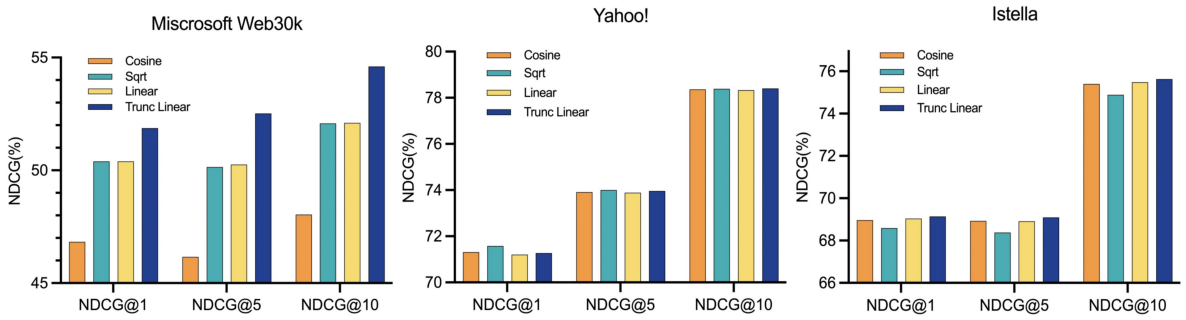


Figure 6: NDCG@K of DenoiseRank at different noise schedule on Miscrosoft Web30k, Yahoo! and Istella datasets.

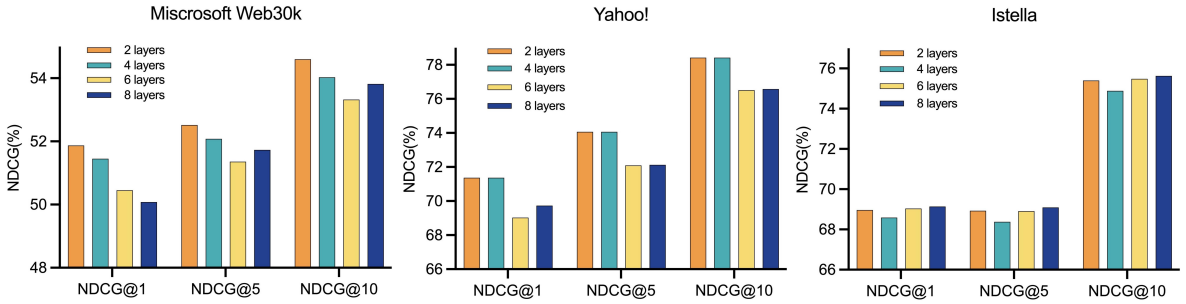


Figure 7: NDCG@K of DenoiseRank at different denoise-net design on Miscrosoft Web30k, Yahoo! and Istella datasets.

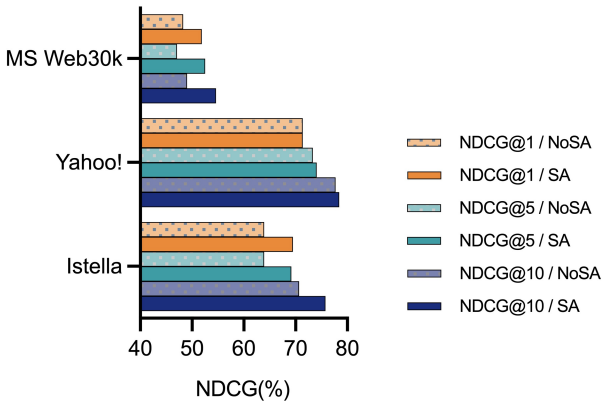


Figure 8: NDCG@K of DenoiseRank without and with self attention on Miscrosoft Web30k, Yahoo! and Istella datasets.

models did not consider uncertainty for ranking and may not rank items diversely.

In this study, we denote diversity in LTR task as: given a query $Q$ and the corresponding documents $D$, run inference by LTR model $M$ times, the number of different ranking sequences is the diversity. For instance, ranking sequence of items $[a, b, c, d, e, f]$ and $[a, b, d, c, f, e]$ are inferenced at different times and causing diversity.

In order to evaluate diversity of our DenoiseRank, we are the first time to introduce a new metric RSD (Ranking Sequence Diversity), formulated as:

$$RSD@(K, M) = \frac{N}{M}, \tag{24}$$

where $K$ denotes the top K corresponding documents of the ranking results, $N$ denotes the number of different sequences of items in $M$ times

Table 5: NDCG@K performance with different noise schedulers on Microsoft Web30K, Yahoo!, and Istella datasets. Best performance per column in bold.

| Scheduler | Web30K | | | Yahoo! | | | Istella | | |
|---|---|---|---|---|---|---|---|---|---|
| | @1 | @5 | @10 | @1 | @5 | @10 | @1 | @5 | @10 |
| TruncatedLinear | **51.87** | **52.52** | **54.60** | 71.27 | 73.96 | **78.40** | **69.14** | **69.09** | **75.63** |
| Linear | 50.40 | 50.25 | 52.10 | 71.20 | 73.89 | 78.33 | 69.04 | 68.91 | 75.48 |
| Cosine | 46.83 | 46.16 | 48.04 | 71.31 | 73.91 | 78.36 | 68.59 | 68.38 | 74.89 |
| Sqrt | 50.40 | 50.15 | 52.08 | **71.58** | **74.00** | 78.39 | 68.97 | 68.93 | 75.40 |

Table 6: NDCG@K Performance with different denoise network depths on Microsoft Web30K, Yahoo!, and Istella datasets. Best performance per column in bold.

| Layers | Web30K | | | Yahoo! | | | Istella | | |
|---|---|---|---|---|---|---|---|---|---|
| | @1 | @5 | @10 | @1 | @5 | @10 | @1 | @5 | @10 |
| 2 | **51.87** | **52.52** | **54.60** | 71.37 | 74.06 | 78.42 | 69.00 | 69.10 | 75.69 |
| 4 | 51.45 | 52.08 | 54.03 | **71.37** | **74.06** | **78.42** | **69.54** | 69.14 | 75.65 |
| 6 | 50.45 | 51.36 | 53.32 | 69.03 | 72.09 | 76.51 | 69.41 | 69.01 | 75.65 |
| 8 | 50.08 | 51.73 | 53.82 | 69.73 | 72.13 | 76.58 | 69.46 | **69.17** | **75.80** |

Table 7: NDCG@K performance with/without self-attention on Microsoft Web30K, Yahoo!, and Istella datasets. Best performance per column in bold. ↑ denotes significant improvements.

| Self-Attention | Web30K | | | Yahoo! | | | Istella | | |
|---|---|---|---|---|---|---|---|---|---|
| | @1 | @5 | @10 | @1 | @5 | @10 | @1 | @5 | @10 |
| Without | 48.25 | 47.08 | 49.00 | 71.37 | 73.35 | 77.70 | 63.91 | 63.91 | 70.65 |
| With | **51.87**$^{\uparrow}$ | **52.52**$^{\uparrow}$ | **54.60**$^{\uparrow}$ | **71.37**$^{\uparrow}$ | **74.06**$^{\uparrow}$ | **78.42**$^{\uparrow}$ | **69.46**$^{\uparrow}$ | **69.17**$^{\uparrow}$ | **75.80**$^{\uparrow}$ |

inferred and $N \in [1, M]$. In LTR task, we hope that $RSD@(K, M)$ increases, while $NDCG@K$ does not significantly decrease. There are various metrics that are relevant to ranking diversity, including $Coverage$, $ERR@K$, $\text{Precition} - \text{IA}@K$, $DIVERSITY@K$, $\alpha$-NDCG@K etc. However, those metrics are quite different to our $RSD@(K, M)$ in at least the following aspects: (1) $RSD$ considers $M$ times inference and $M$ sequences of items, while traditional diversity metrics focus on single-ranking sequence of items. (2) $RSD$ focus on the diversity of the sequences, while traditional diversity metrics consider the similarity between items in a single sequence. For instance, $DIVERSITY@K$ consider the similarity of pairwise documents among the top K documents in the result sequence. The higher the metric, the more dissimilar the pair.

We conducted experiments on MS Web30K datasets to investigate the ranking sqeuence deiversity of DenoiseRank and Rankformer which has the similar architecture. We set $K \in \{1, 5, 10, 20\}$ and $M = 10$. Then, given a well-trained DenoiseRank model and Rankformer model, we evaluate them on the same test file of Web30K datasets. The result of NDCG@K and RSD@(K,M) shown in Table. 8, and we find that:

1. Among 10 times inferences, the $RSD$ is 0.11, 0.16, 0.28, 0.64 in the top 1,5,10,20 positions, respectively, showing the ability to rank sequence diversity of DenoiseRank and the Gaussian sampling for $Y_T$ provides uncertainty to rank.

2. Performance of NDCG@K remains excellent and even slightly increases after repeat inference, which means that our DenoiseRank can produce diverse ranked lists while guarantees reliability of ranking result.

3. Rankformer did not present the ability to rank in different order, the $RSD$ is 0.1 regardless of the K poisition. It proved our extrapolate that traditional LTR models do not inject uncertainty which results in a static ranking sequence.

According to the above analysis, our DenoiseRank can be applied to areas requiring diverse ranking sequences of items. Our novel metric, RSD, can also be used to evaluate the ranking diversity ability of models in other areas.

Table 8: NDCG@K and RSD@(K,M) performance of DenoiseRank and Rankformer on Microsoft Web30K datasets.

| Model | M | RSD@(K,M) | | | | NDCG@K | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 5 | 10 | 20 | 1 | 5 | 10 | 20 |
| Rankformer | 1 | – | – | – | – | 49.62 | 49.30 | 51.42 | 54.29 |
| | 10 | 0.1 | 0.1 | 0.1 | 0.1 | 49.62 | 49.30 | 51.42 | 54.29 |
| DenoiseRank | 1 | – | – | – | – | 51.48 | 52.46 | 54.47 | 57.49 |
| | 10 | 0.11 | 0.16 | 0.28 | 0.64 | 51.73 | 52.52 | 54.47 | 57.45 |

## D Loss Functions

DenoiseRank employs MSE as the loss function in order to predict $Y_0$ at every timestep, see Eq.13. MSE is also the original loss in DDPMs. Defining suitable ranking losses is an important branch of LTR studies, and there are many versions of loss functions that significantly improve the effectiveness of models. To align with this, we evaluate the performance of DenoiseRank with different losses for ranking. Thus, in this study, we try to find an optimal loss function for DenoiseRank on different datasets.

We consider the following loss functions:

1. RMSE: a typical point-wise loss: $L_{\text{RMSE}}(Y, \hat{Y}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$.

2. RankNet(Burges et al., 2005): a classic pair-wise loss: $L_{\text{RankNet}}(Y, \hat{Y}) = \sum_{Y_i > Y_j} \log_e(1 + e^{\hat{Y}_j - \hat{Y}_i})$.

3. NDCGLoss$_{2++}$(Wang et al., 2018): a NDCG metric-driven loss functions based on the lambdaLoss probabilistic framework:

$$L_{\text{NDCGLoss2++}}(Y, \hat{Y}) = -\sum_{Y_i > Y_j} \log_2 \sum_{\pi} \left(\frac{1}{1 + e^{-\sigma(\hat{Y}_i - \hat{Y}_j)}}\right)^{(\rho_{ij} + \mu\delta_{ij})|G_i - G_j|} H(\pi|\hat{Y}),$$

where $G_i = \frac{2^{y_i}-1}{\text{maxDCG}}$, $\rho_{ij} = |\frac{1}{Di} - \frac{1}{D_j}|$, $\delta_{ij} = |\frac{1}{D_{|i-j|}} - \frac{1}{D_{|i-j|+1}}|$, $D_i = \log_2(1+i)$, and $H(\pi|\hat{Y})$ is a hard assignment distribution of permutations.

4. ApproxNDCG(Qin et al., 2010)(Bruch et al., 2019): a loss that designed to be approximation of NDCG metrics, $L_{\text{ApproxNDCG}}(Y, \hat{Y}) = \frac{1}{Z}\sum_{i=1}^{n}\frac{G(Y_i)}{\log_2(1+\pi(i))}$, where $Z = -DCG(\pi^*, Y)$, $G(Y_i) = 2^{Y_i} - 1$ and $\pi(i) = \frac{1}{2} + \sum_j \text{sigmoid}(\frac{\hat{Y}_j - \hat{Y}_i}{T})$, $T$ is a smooth parameter.

5. ListNet(Cao et al., 2007): a classic list-wise loss: $L_{\text{ListNet}}(Y, \hat{Y}) = -\sum_{i=1}^{n} Y_i \log_e \frac{e^{\hat{Y}_i}}{\sum_j e^{\hat{Y}_j}}$.

6. MSE (Ho et al., 2020)(Nichol and Dhariwal, 2021): a loss function use in DDPMs to predict $x_0$ or $\epsilon$, here we formulate it as $L_{\text{MSE}}(Y, \hat{Y}) = \mathbb{E}[|| Y - \hat{Y} ||^2]$

We report the results based on the best NDCG@10 for different losses. For different loss functions, we use AdamW optimizer and scan learning rate $\in 0.01, 0.001, 0.0001$. We try to find the best performance of every loss and report the results based on the NDCG@10. The results are shown in Table. 9, we find that:

1. DenoiseRank, when trained with MSE, RMSE and ListNet, achieves first-tier performance and is far superior to the rest.

2. Though ApproxNDCG improves the performance of neural LTR models in the original papers, it does not seem to work well on DenoisRank, which is implemented from a generative perspective.

3. DenoiseRank, when trained with ListNet, performs the best on the Web30K dataset. However, for the Yahoo! and Istella datasets, training with MSE loss is the best choice.

## E Other Metrics

In order to evaluate our denoiseRank fully, we use another 4 types of ranking metrics, including Expected Reciprocal Rank (ERR), Mean Average Precision (MAP), Mean Reciprocal Rank(MRR) and Precision. We reported results at ranks 1,3,5,10,20 and the total rank (denoted as "ALL"). The results are shown in Table. 10, and the results further confirm the effectiveness of our models.

Table 9: NDCG@K performance of DenoiseRank with different loss functions on Microsoft Web30K, Yahoo!, and Istella datasets. Best performance per column in bold.

| Loss | Web30K | | | Yahoo! | | | Istella | | |
|---|---|---|---|---|---|---|---|---|---|
| | @1 | @5 | @10 | @1 | @5 | @10 | @1 | @5 | @10 |
| RMSE | 50.48 | 51.41 | 53.43 | 70.64 | 73.30 | 77.84 | 69.26 | **69.32** | 75.80 |
| RankNet | 43.66 | 45.84 | 48.56 | 56.65 | 66.96 | 73.18 | 51.11 | 57.25 | 65.83 |
| $\text{NDCGLoss}_{2++}$ | 43.01 | 47.68 | 50.57 | 66.48 | 72.18 | 76.98 | 56.00 | 59.96 | 67.55 |
| ApproxNDCG | 24.46 | 29.21 | 33.72 | 60.24 | 64.67 | 70.64 | 33.01 | 42.21 | 52.48 |
| ListNet | **51.87** | **52.52** | **54.60** | 70.81 | 73.82 | 78.35 | 68.75 | 69.05 | 75.58 |
| MSE | 51.20 | 51.73 | 53.77 | **71.37** | **74.06** | **78.42** | **69.4**6 | 69.17 | **75.80** |

Table 10: ERR, MRR, MAP and precision of DenoiseRank on Microsoft Web30K, Yahoo!, and Istella datasets.

| Metric | K | Dataset | | | Metric | K | Dataset | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Web30K | Yahoo! | Istella | | | Web30K | Yahoo! | Istella |
| ERR | 1 | 26.53 | 34.41 | 61.53 | MAP | 1 | 78.07 | 87.13 | 94.64 |
| | 5 | 36.77 | 43.90 | 73.79 | | 5 | 81.55 | 89.16 | 95.19 |
| | 10 | 38.55 | 45.34 | 74.34 | | 10 | 78.76 | 87.91 | 93.14 |
| | 20 | 39.28 | 45.72 | 74.40 | | 20 | 74.89 | 86.72 | 90.27 |
| | ALL | 39.57 | 45.78 | 74.40 | | ALL | 63.91 | 85.75 | 88.36 |
| MRR | 1 | 78.07 | 87.13 | 94.64 | Precision | 1 | 78.07 | 87.13 | 94.64 |
| | 5 | 84.36 | 90.56 | 96.77 | | 5 | 72.86 | 83.59 | 89.38 |
| | 10 | 84.64 | 90.69 | 96.79 | | 10 | 69.25 | 81.25 | 80.44 |
| | 20 | 84.73 | 90.70 | 96.79 | | 20 | 64.17 | 78.81 | 55.41 |
| | ALL | 84.75 | 90.71 | 96.79 | | ALL | 44.97 | 75.30 | 12.81 |