KDCRec: Knowledge Distillation for Counterfactual Recommendation via Uniform Data

Dugang Liu, Pengxiang Cheng, Zinan Lin, Jinwei Luo, Zhenhua Dong, Xiuqiang He, Weike Pan, and Zhong Ming

Abstract—The bias problems in recommender systems are important challenges. In this paper, we focus on solving the bias problems via uniform data. Previous works have shown that simple modeling with a uniform data can alleviate the bias problems and improve the performance. However, the uniform data is usually few and expensive to collect in a real product. In order to use the valuable uniform data more effectively, we propose a novel and general knowledge distillation framework for counterfactual recommendation with four specific methods, including label-based distillation, feature-based distillation, sample-based distillation and model structure-based distillation. Moreover, we discuss the relation between the proposed framework and the previous works. We then conduct extensive experiments on both public and product datasets to verify the effectiveness of the proposed four methods. In addition, we explore and analyze the performance trends of the proposed methods on some key factors, and the changes in the distribution of the recommendation lists. Finally, we emphasize that counterfactual modeling with uniform data is a rich research area, and list some interesting and promising research topics worthy of further exploration. Note that the source codes are available at https://github.com/dgliu/KDCRec.

Index Terms—Counterfactual, bias, recommender systems, knowledge distillation, uniform data

1 INTRODUCTION

RECOMMENDER Systems as a feedback loop system may suffer from the bias problems such as popularity bias [2], [3], previous model bias [4], [5], [6] and position bias [7], [8]. Previous studies have shown that models and evaluation metrics that ignore the biases do not reflect the true performance of a recommender system, and that explicitly handling of the biases may help improve the performance [6], [7], [9]. Most of the previous works to solve the bias problems of recommender systems can be classified as heuristic-based, counterfactual learning-based [9], [10] and some theoretical tools-based methods [11], [12]. The former mainly makes certain assumptions about the data being missing at random (MNAR) [5], [13], while the latter two mainly use the inverse propensity score (IPS) [10], [14] and the causal inference techniques [11], [12], respectively.

A recent work has shown that a uniform data can alleviate the previous model bias problem [6]. But the uniform data is usually few and expensive to collect in real recommender systems. To collect a uniform data, we must intervene in the system by using a uniform logging policy instead of a stochastic recommendation policy, that is, for each user’s request, we do not use the recommendation model for item delivery, but instead randomly select some items from the available items and rank them with a uniform distribution. The uniform data can then be regarded as a good unbiased agent because it isolates the source of bias at the system level as much as possible. However, the uniform logging policy would hurt the users’ experiences and the revenue of the platform. This means that it is necessary to constrain the uniform data collection within a particularly small traffic (e.g., 1%).

In this article, we focus on how to solve the bias problems in a recommender system with a uniform data. Since biased data and uniform data can be regarded as the treatment group and the control group in the field of causal counterfactual [15], respectively, we call the studied problem as counterfactual recommendation via uniform data. Along the line of [6], in the conference version of this work [1], we conduct empirical studies on a real advertising system and a public dataset to validate the usefulness of the uniform data, where the uniform data is simply combined with the non-uniform data for training models. We have observed that such a simple method can alleviate the bias and improve the performance, which motivates us to study more advanced methods that can make better use of the uniform data. Although there are many ways to extract information or knowledge from a uniform data, in this paper we focus on knowledge distillation because of its simplicity and flexibility.

To use the few and valuable uniform data more effectively, we propose a general knowledge distillation framework for counterfactual recommendation (KDCRec), which enables
uniform data modeling with four methods, i.e., label-based distillation, feature-based distillation, sample-based distillation and model structure-based distillation. Each one is based on a specific concern, i.e., label-based distillation focuses on using the imputed labels as a carrier to provide useful debiasing guidance; feature-based distillation aims to filter out the representative unbiased features; sample-based distillation considers mutual learning and alignment of the information of the uniform and non-uniform data; and model structure-based distillation constrains the training of the models from the perspective of embedded representation.

2 RELATED WORK
Since we study how to apply knowledge distillation techniques for counterfactual recommendation to solve the bias problems, we first review some related works on general knowledge distillation, and then summarize some related work on debiasing recommendation.

2.1 Knowledge Distillation
Hinton's work first proposes the concept of knowledge distillation [16]. By introducing soft targets related to teacher networks as part of the objective function, the training of student networks is guided to achieve knowledge transfer [17]. A series of follow-up works develop different distillation structures (e.g., multiple teachers [18] and cascade distillation [19]) and different forms of knowledge (e.g., alignment of the hidden layers [20] or the relation between the hidden layers [21]). Some recent works are no longer limited to model structure, but consider sample-based knowledge distillation [22], [23]. In this article, we further expand the definition of distillation to include label-based and feature-based forms. The marriage of knowledge distillation and recommender systems has also attracted the attention of the researchers [24], [25], [26]. Most of these works focus on using knowledge distillation to extract some useful knowledge from some auxiliary models to enhance the performance or interpretability of the target recommendation model. In this paper, we focus on using knowledge distillation to solve the bias problems in recommender systems.

2.2 Debiasing Recommendation
Recommender systems generate various bias problems in the feedback loop between the users and system, and how to effectively alleviate these biases is a key issue to be addressed [27]. Based on the availability of a uniform data, the existing methods for debiasing recommendation can be mainly classified into two routes, i.e., debiasing recommendation without uniform data and debiasing recommendation with uniform data.

The former mainly considers the debiasing process directly on the biased log data, and can further be categorized into three classes, including heuristic-based, counterfactual learning-based, and some theoretical tools-based methods. The heuristic-based method links a user's feedback with different specific factors to model the non-random missing data, such as item features [28], [29] and user ratings [5], [30]. The counterfactual learning-based method adopts the inverse propensity score and the doubly robust techniques, and estimates the sample weights on the log data to correct the biased feedback distribution [10], [14], [31]. Moreover, some recent works consider introducing some theoretical tools to integrate and solve the bias problems, such as regularization methods [32], information bottleneck [33], [34], positive-unlabeled learning [35], asymmetric tri-training [36], disentangled representation learning [37], and causal inference techniques [11], [12], [38]. Due to the lack of target knowledge, most methods on this route require more complex designs for a specific bias.

The latter additionally introduces a uniform data as the target knowledge to guide the training of a debiasing model, and can be divided into the following three categories: 1) use a uniform data to estimate sample weights with some inverse propensity score techniques, or train an imputation model for data augmentation with some doubly robust techniques [10], [31], [39], [40]; 2) design a multi-stage training framework that alternately uses the log data and the uniform data to learn debiasing parameters [41], [42], [43]; 3) use the log data and the uniform data to jointly train the two models and constrain their proximity in some way [1], [44]. Existing works only consider the use of the uniform data from a specific perspective. In this paper, we propose a general and flexible debiasing framework based on the idea of knowledge distillation, which can provide a systematic guidance on how to effectively use the uniform data from four different perspectives. In particular, the main ideas of most existing methods can be linked to the proposed framework. A detailed discussion can be found in Section 3.5. It is worth mentioning that the counterfactual learning method is used in both routes because of its theoretical insight and generalization, but the propensity score suffers from serious high variance and is thus difficult to be accurately estimated in practice.

3 THE PROPOSED FRAMEWORK
In the pilot experiments of the conference version [1], we have observed that training the model on a simply combined uniform data and biased data can alleviate the bias problems. To provide a systematic guide on how to effectively use the uniform data, we propose a general Knowledge Distillation framework for Counterfactual Recommendation in this section, KDCRec for short. Fig. 1 shows the overview of the framework of our KDCRec. In our framework, the uniform data can be modeled with four different methods, including label-based distillation, feature-based distillation, sample-based distillation, and model structure-based distillation. Note that we use a general definition of distillation in the study rather than the past knowledge distillation approaches such as considering the level of sample [22], [23] and model structure [16], [20]. These methods are based on different concerns to mine the potentially useful knowledge from the uniform data, which will be used to improve the learning of the biased data.

Next, we will introduce the four methods in turn as different modules. More specifically, in each module, we will give a formal definition of the corresponding method, and list some practical solutions under the guidance of the definition. Note that this section is an extended version of the framework described in [1]. We omit the introduction of some old strategies in [1] and add some new and more
effective improvement strategies. To further distinguish the new strategy from the old strategy, we will add “NEW” as a mark to the new strategy. Please refer to [1] for an introduction to all the old strategies. Note that in the experiments we have two forms to obtain the desired model. The first is to pre-train a model $M$, on non-uniform data, and then perform debiasing through fine-tuning. The second is to train a model from initialization and debias it during the training process. $M_k$ is used to denote this model to distinguish it from $M$. For ease of description, the main notations are listed in Table 1.

![Diagram](image.png)

**Fig. 1.** Overview of the KDCRec framework. The scale of the biased set $S_t$ is much larger than that of the unbiased set $S_c$. Since the unobserved data is only used in some modules, we distinguish it from $S_c$ and $S_t$ using a different color.

### 3.1 Label-Based Module

Models trained on a non-uniform data $S_c$ tend to produce biased predictions, while predictions from a uniform data $S_t$ are more unbiased. An intuitive idea is that when training a model on $S_c$, the model may receive the imputed labels produced by $S_t$ to correct the bias of its own predictions. Based on this idea, we develop the following formal definition of label-based distillation. Note that on the premise of using the imputed labels, we can also include the labels of $S_t$. We emphasize the use of the imputed labels to avoid confusion with other distillation methods.

**Definition 1 (D1).** A method can be classified as label-based distillation if and only if the training of a non-uniform data $S_c$ can benefit from the imputed labels produced by a uniform data $S_t$.

**Solutions.** We will use the three strategies adopted in our experiments as examples to illustrate how label-based distillation can be realized.

- **Variant 1 of Bridge Strategy (NEW, Bridge-var1 for short).** Let $D$ denote the whole set of data, including the non-uniform data $S_c$, the uniform data $S_t$, and the unobserved data. Recalling the Bridge strategy in [1], we consider a scenario where only the models $M_c$ and $M_t$ are used for joint debiasing training, and then $M_t$ is used for recommendation. Different from the original version [1], we consider introducing a third model $M_k$ to learn from $S_c$ and $S_t$, and make recommendations after debiasing training. Similarly, to make full use of the available knowledge, we also randomly sample an auxiliary set $S_a$ from $D$ as a bridge in each iterative training, and expect the predicted output of $M_c$, $M_t$, and $M_k$ on $S_a$ to be close. Note that most of the samples in $S_a$ are unobserved data because of the data sparsity in recommender systems. Since the knowledge of $S_c$, $S_t$, $M_c$, and $M_t$ is comprehensively considered and balanced, we expect that this strategy can reduce the bias of $M_k$. The final objective function of this strategy is,

$$
\begin{align*}
\min_{W_c, W_t, W_k} & \frac{1}{|S_c|} \sum_{(i,j) \in S_c} \ell(y_{ij}, \hat{y}_{ij}^c) + \frac{1}{|S_t|} \sum_{(i,j) \in S_t} \ell(y_{ij}, \hat{y}_{ij}^t) \\
& + \frac{1}{|S_a|} \sum_{(i,j) \in S_a} \frac{1}{2} \ell(\hat{y}_{ij}^c, \hat{y}_{ij}^t) + \frac{1}{2} \ell(\hat{y}_{ij}^k, \hat{y}_{ij}^t) \\
& + \lambda_c R(W_c) + \lambda_t R(W_t) + \lambda_k R(W_k),
\end{align*}
$$

where $W_c$, $W_t$, and $W_k$ denote the parameters of $M_c$, $M_t$, and $M_k$, respectively, and $\ell(\cdot, \cdot)$ is an arbitrary loss function. And $y_{ij}, \hat{y}_{ij}^c, \hat{y}_{ij}^t$, and $\hat{y}_{ij}^k$ denote the true label, and the predicted labels of $M_c$, $M_t$, and $M_k$ for the sample $(i, j)$, respectively, where $(i, j)$ is associated with user $i$ and item $j$. Note that $R(\cdot)$ is the regularization term, and $\lambda_c$, $\lambda_t$, and $\lambda_k$ are the parameters of the regularization.

- **Variant 2 of Bridge Strategy (NEW, Bridge-var2 for short).** When $M_t$ contains serious bias problems, the incorrect knowledge provided by $M_t$ may weaken the effectiveness of the Bridge-var1 strategy. For this reason, we consider only using the knowledge in $M_t$ to guide $M_k$. However, due to the small scale of $S_t$, $M_t$ may make inaccurate prediction for $S_a$. This means that directly constraining the alignment of $M_k$ and $M_t$ on label may lead to incorrect training. In order to address this issue, we propose to introduce some additional weights for each sample ($S_c$ and $S_t$) based on the similarity of the parameters in $M_t$ and $M_k$, where a higher similarity means a higher weight. The final objective function of this strategy is,

**TABLE 1**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>the whole set of data, i.e., $D = S_c \cup S_t \cup$ unobserved data</td>
</tr>
<tr>
<td>$S_c$</td>
<td>the uniform data</td>
</tr>
<tr>
<td>$S_t$</td>
<td>the non-uniform data</td>
</tr>
<tr>
<td>$M_c$</td>
<td>a model pre-trained on $S_c$ in a supervised manner</td>
</tr>
<tr>
<td>$M_t$</td>
<td>a model pre-trained on $S_t$ in a supervised manner</td>
</tr>
<tr>
<td>$M_k$</td>
<td>an initialized model trained on $S_c \cup S_t$</td>
</tr>
<tr>
<td>$(i,j)$</td>
<td>a sample associated with user $i$ and item $j$</td>
</tr>
<tr>
<td>$y_{ij}$</td>
<td>the true label for the sample $(i,j)$</td>
</tr>
<tr>
<td>$\hat{y}_{ij}^c$</td>
<td>the predicted label of $M_c$ for the sample $(i,j)$</td>
</tr>
<tr>
<td>$\hat{y}_{ij}^t$</td>
<td>the predicted label of $M_t$ for the sample $(i,j)$</td>
</tr>
<tr>
<td>$\hat{y}_{ij}^k$</td>
<td>the predicted label of $M_k$ for the sample $(i,j)$</td>
</tr>
<tr>
<td>$W_c$</td>
<td>model parameters</td>
</tr>
<tr>
<td>$\ell$</td>
<td>an arbitrary loss function</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>the parameters of the regularization</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>a tunable parameter that controls the importance of loss</td>
</tr>
<tr>
<td>$\tau$</td>
<td>temperature parameter</td>
</tr>
</tbody>
</table>
The bias of $S_t$ may be reflected in the labels, resulting in models trained on these labels being biased. For example, when generating samples for modeling, all the observed positive feedback are usually labeled as 1, and all the observed negative feedback are labeled as -1. However, in real applications, they should fit a preference distribution. With $S_t$, we expect to be able to better infer the true distribution of the labels on $S_c$ and then refine them. Recalling the Refine strategy in [1], we use the imputed label generated by $M_t$ to refine the label of $S_t$. However, previous works show that a combined data is more effective than $S_t$ alone [1]. Instead of obtaining a model $M_t$ pre-trained on $S_t$ [1], we want to obtain a model $M'_t$ pre-trained on a small combined data. More specifically, we first randomly sample a small subset $S'_c$ from $S_c$, and then obtain a small combined data $S_t \cup S'_c$ and a remaining biased data $S_t \setminus S'_c$. We then use $M'_t$ to predict all the samples in $S_t \setminus S'_c$. These imputed labels are combined with the original labels of $S_t \setminus S'_c$, and then the feature-wise influence [50] on validation loss. Sample-wise influence has been used in previous works to estimate the propensity score of a sample in $S_c$ where $S_t$ is used as the validation set [39]. If a change in a sample in $S_c$ leads to a large change in validation loss, the sample has greater influence and is considered to be closer to be unbiased, otherwise it has less influence and is more biased. Similarly, in this strategy, we use $S_t$ as the validation set and use feature-wise influence to guide the training of a feature selection model built using $S_t$ where $S_t$ and $S_c$ are used together to compute the influence loss in the trained network. If the selection of a feature leads to a greater change in the verification loss, the feature has a greater influence and is retained. Otherwise, the feature should be filtered. This strategy is shown in Fig. 3. And finally, we obtain a stable feature set $S_{oc}$ for model training.

**Pre-Feature Selection Strategy (NEW, PFS for short).** Due to the unbiased nature of a uniform data, performing a feature selection process on it is more likely to get the desired stable feature index set. We first adopt a classic feature selection method on $S_t$ to obtain the stable feature index set. Note that we use the Lasso method [47] for feature selection in the experiments. This stable feature index set is then used to perform pre-feature selection on $S_c$ to remove pseudo-correlated features that are not conducive to model training. Finally, we train a debiased recommendation model on a stable set $S_{oc}$. This strategy is illustrated in Fig. 2.

**Influential Feature Selection Strategy (NEW, IFS for short).** The influence function (IF) is an important concept in robust statistics [48], which can be extended to measure the sample-wise influence [49] and the feature-wise influence [50] on validation loss. Sample-wise influence has been used in previous works to estimate the propensity score of a sample in $S_c$ where $S_t$ is used as the validation set [39]. If a change in a sample in $S_c$ leads to a large change in validation loss, the sample has greater influence and is considered to be closer to be unbiased, otherwise it has less influence and is more biased. Similarly, in this strategy, we use $S_t$ as the validation set and use feature-wise influence to guide the training of a feature selection model built using $S_c$ where $S_t$ and $S_c$ are used together to compute the influence loss in the trained network. If the selection of a feature leads to a greater change in the verification loss, the feature has a greater influence and is retained. Otherwise, the feature should be filtered. This strategy is shown in Fig. 3. And finally, we obtain a stable feature set $S_{oc}$ for model training.
3.3 Sample-Based Module

In a real recommender system with a stochastic logging policy, the probability of an item being recommended is different, and the probability of a user making a choice is also different. This means that model $M_i$ may treat some items and users unfairly, because the samples in $S_i$ lack support for these items and users. This unfairness can be corrected to some extent by directly considering the samples in $S_i$ during the training process of $M_i$. Because the uniform logging policy corresponding to $S_i$ increases the probability of the less popular items being selected, and $M_i$ can weigh this difference between $S_i$ and $S_t$. Based on this idea, we develop the following formal definition of sample-based distillation.

**Definition 3 (D3).** A method can be classified as sample-based distillation if and only if a uniform data $S_t$ is directly applied to help learning on all the samples without generating some imputed labels.

**Solutions.** We will use the three strategies adopted in our experiments as examples to illustrate how sample-based distillation can be realized.

- **Causal Embedding Strategy (CausE for short).** The causal embedding method [44] first considers the scenario of training $M_c$ and $M_i$ simultaneously. It designs an additional alignment term to explicitly represent the learning of $M_c$ for $M_i$. Causal embedding defines this alignment term as the pairwise difference between the parameters of $M_c$ and $M_i$, which is then included in the object function to be minimized. When the value of the alignment term becomes small, it means that $M_c$ learns the causal information contained in $S_t$, which helps correct the bias in learning on $S_i$. Note that it is difficult to dynamically optimize the differences between all the parameters of two complex models such as neural networks, so we use two low-rank models to implement this strategy in our experiments. The final objective function is,

$$\min_{W_c, W_i} \frac{1}{|S_t|} \sum_{(i,j) \in S_t} \ell(y_{ij}, \hat{y}_{ij}^t) + \frac{1}{|S_i|} \sum_{(i,j) \in S_i} \ell(y_{ij}, \hat{y}_{ij}^i) + \lambda_c R(W_c) + \lambda_i R(W_i) + \lambda_{c}^{\text{CausE}} |W_i - W_c|_F^2,$$

where $\lambda_{c}^{\text{CausE}}$ is the regularization parameter for the alignment term of $M_c$ and $M_i$.

- **Weighted Sample Strategy (NEW, WeightS for short).** How to effectively combine the sample in $S_c$ and $S_i$ to train the model $M_i$? Inspired by modeling of heterogeneous implicit feedback [51], we add a confidence parameter to each sample of $S_c$ and $S_i$ to indicate whether it is unbiased. Naturally, the confidence of the samples in $S_i$ is set to 1, and the confidence of the samples in $S_c$ has two schemes to be used. The first scheme is a global setting, i.e., we set a confidence value in advance for all the samples of $S_c$. The second scheme is a local setting, i.e., each sample of $S_c$ has a confidence value that needs to be learned by $M_k$. From the experimental results in [1], we find that the local setting does not perform well on a large-scale dataset. One possible reason is that as the scale of $S_c$ increases, it becomes more difficult to accurately learn a large number of weights without referring to other information. We suggest introducing $M_i$ and designing local weights according to the similarity of the parameters in $M_i$ and $M_k$. When $M_i$ and $M_k$ have similar patterns on some samples, due to the unbiased nature of $M_i$, we can think that these samples are more likely to be unbiased (i.e., greater weight). The confidence of each sample is related to the corresponding loss function. The final objective function of this strategy is,

$$\begin{align*}
\min_{W_k} & \frac{1}{|S_c|} \sum_{(i,j) \in S_c} \alpha_{ij} \ell(y_{ij}, \hat{y}_{ij}^c) + \frac{1}{|S_i|} \sum_{(i,j) \in S_i} \ell(y_{ij}, \hat{y}_{ij}^i) + \lambda_k R(W_k),
\end{align*}$$

where $\alpha_{ij}$ is a parameter used to control the confidence that we believe the sample $(i,j)$ is unbiased. When considering the global setting, $\alpha_{ij}$ shares a parameter value that we preset for all the samples in $S_c$, but in the local setting, $\alpha_{ij} = \text{sigmoid}(\frac{\cos(W_i^j, W_k^c) + \cos(W_c^j, W_k^c)}{2})$.

- **Delayed Combination Strategy (Delay for short).** Instead of introducing a confidence parameter, we propose a strategy called delayed combination. This strategy directly applies the data of $S_i$ and $S_t$ to the training of $M_k$ in an alternative manner. Specifically, in the $S_i$ step of each iteration, $M_k$ is trained on the data of $s$ batches from $S_i$. In the $S_t$ step, we randomly sample one batch of data from $S_t$ to train $M_k$. We repeat these two steps until all the data of $S_i$ are used. The batch ratio is set to $s:1$, which can better ensure that $M_k$ learns the information of $S_c$ and the correction under the guidance of $S_t$. The final objective function of this strategy is,

$$\begin{align*}
\min_{W_k} & \frac{1}{|S_t|} \sum_{(i,j) \in S_t} \ell(y_{ij}, \hat{y}_{ij}^t) + \lambda_i R(W_i), \quad S_t \text{ step}, \\
\min_{W_k} & \frac{1}{|S_c|} \sum_{(i,j) \in S_c} \ell(y_{ij}, \hat{y}_{ij}^c) + \lambda_c R(W_c), \quad S_c \text{ step},
\end{align*}$$

3.4 Model Structure-Based Module

Finally, we return to the model itself through considering how to directly use the pre-trained model $M_i$ to help the learning of $M_c$. This is the most commonly adopted distillation strategy in existing works [16]. In order to help $M_c$ with the guidance from $M_i$, we assume that some embedded representations of $M_i$ correspond to some embedded representations of $M_c$. We constrain the selected embedded representations in $M_c$ to be similar to their corresponding
Previous works have shown that $M$ will have a similar pattern to $M_i$ and thus may benefit from it. Note that the selected embedded representations of $M_i$ and $M_t$ do not necessarily have the same index. For example, suppose $A$ is a 4-layer network and $B$ is an 8-layer network, we may specify that each layer of $A$ corresponds to an even layer of $B$, namely 2, 4, 6 and 8. Based on this idea, we develop the following formal definition of model structure-based distillation. For the sake of discussion, as shown in Fig. 4, we classify all the embedded representations into three types with different functions.

**Definition 4 (D4).** A method can be classified as model structure-based distillation if and only if instead of using the labels and data, the embedded representation trained on a uniform data $S_t$ is used to help the learning of a non-uniform data $S_c$.

**Solutions.** We will use the three strategies adopted in our experiments as examples to illustrate how model structure-based distillation can be realized.

- **Feature Embedding Strategy** (FeatE for short). Feature embedding are the embedded representations that are directly connected to the users and items. In a neural network, it is usually the result of a one-hot coding after a lookup operation; and in a low-rank model, it is the users’ preference vector $U_i$ and the items’ attribute vector $V_j$. As a special example, we think that the feature embedding of the autoencoder refers to the items’ attribute vector. According to the different definitions of the two modules to which they belong, CausE and FeatE do not bring much useful information to the student network [16]. But, by introducing some softmax and temperature operations to relax the label, training the student network to keep the same output as the teacher network on a soft label will result in a significant improvement [16]. We follow a similar setup in this strategy. Note that in the experiments we must also use deep neural networks to implement this strategy. The final objective function of this strategy is,

\[
\min_{W_c} \frac{1}{|S_c|} \sum_{(i,j) \in S_c} \ell(y_{ij}, \hat{y}_{ij}^c) + \lambda_c R(W_c)
\]

where $\hat{y}_{ij}^c$ is the output of $M_c$ and $M_t$ on their respective designated hint layers.

- **Soft Label Strategy.** Previous works have shown that training the student network to mimic the output of the teacher network on hard-labeled objectives does not bring much useful information to the student network [16]. But, by introducing some softmax and temperature operations to relax the label, training the student network to keep the same output as the teacher network on a soft label will result in a significant improvement [16]. We follow a similar setup in this strategy. The final objective function of this strategy is,

\[
\min_{W_c} \frac{\alpha}{|D|} \sum_{(i,j) \in D} \ell\left(\text{softmax}\left(\frac{\hat{y}_{ij}}{\tau}\right), \text{softmax}\left(\frac{\hat{y}_{ij}^c}{\tau}\right)\right)
\]

where $\tau$ is a temperature parameter, and $\alpha$ is a tunable parameter that controls the importance of the soft labels.

**3.5 Summary and Remarks**

Based on the above description, we can see that different strategies exhibit their own characteristics about how to make use of $S_t$. One possible confusion is between CausE and FeatE. Although they both operate on the feature embeddings, they are different in terms of the available training samples and training methods. According to the different definitions of the two modules to which they belong, CausE cannot use some imputed labels, while FeatE does not have this restriction. In addition, the feature embedding in CausE is used to constrain the alignment of $M_i$ and $M_t$, so as part of the additional loss term to measure the distance between the two. FeatE aims to use $M_i$’s feature embedding to help $M_t$’s training, so it is more flexible in usage. For example, we can use the proposed Initialization and Concatenation operations.
Although we introduce the four distillation methods in different modules, their are closely related. This means that we can design new strategies with different combinations of the four distillation methods, such as the combination of sample-based distillation and label-based distillation. Note that most of the works in debiasing recommendations with uniform data can be incorporated into our framework. For example, label-based distillation includes a direct method for learning an imitation model and its variants, which is a module that most methods can be equipped with. Sample-based distillation includes the IPS techniques [9], [10], and the combination of sample-based and label-based distillation includes doubly robust techniques [14], [52], which are the ideas adopted by most methods. Moreover, they are also related to the types of knowledge (instance, feature and model) and strategies (adaptive, collective and integrative) in transfer learning [17], [53].

In addition, we must keep in mind the different considerations when using these four distillation methods. Although label-based and sample-based distillations are easy to implement, they need to consider the potential factors on the label and sample that may affect the model, such as the differences in sample size and label distributions. The difference in label distributions is passed on to the distributions of the predicted labels, so that the strategy of directly using the predicted labels may lead to poor results. The difference in data size means that $M_i$ in a rough strategy can almost ignore the guided information from $S_i$. Feature-based distillation relies on the effectiveness of the method used to filter out the causal and stable features. However, the current research in this direction is still not sufficient. Model structure-based distillation requires only the model itself without regarding to other potential factors. But it is not easy to design an effective distillation structure or to select some good embedded representations.

4 Empirical Evaluation

In this section, we conduct experiments with the aim of answering the following four key questions.

- **RQ1:** How do the proposed methods perform against the baselines in an unbiased evaluation?
- **RQ2:** How do some key factors affect the performance of the proposed methods?
- **RQ3:** How do some specific factors affect the performance of the proposed new strategies?
- **RQ4:** What impact does the proposed methods have on the item distribution of the recommendation lists?

4.1 Experiment Setup

4.1.1 Datasets

To evaluate the performance of the proposed framework in an ideal unbiased scenario, the selected dataset must have a uniform subset for training and test. We consider the following datasets in the experiments, where the statistics are described in Table 2.

- **Yahoo! R3 [5]:** This dataset contains ratings collected from two different sources on Yahoo! Music services, involving 15,400 users and 1,000 songs. The Yahoo! user set consists of ratings supplied by users during normal interactions, i.e., users pick and rate items as they wish. This can be considered as a stochastic logging policy by following [9], [10], and thus the user set is biased. The Yahoo! random set consists of ratings collected during an online survey, when each of the first 5,400 users is asked to provide ratings on ten songs. The random set is different because the songs are randomly selected by the system instead of by the users themselves. The random set corresponds to a uniform logging policy and can be considered as the ground truth without bias. We binarize the ratings based on a threshold $\epsilon = 3$. Hence, a rating $y_{ij} > \epsilon$ is considered as a positive feedback (i.e., label $y_{ij} = 1$), otherwise, it is considered as a negative feedback (i.e., label $y_{ij} = -1$). The Yahoo! user set is used as a training set in a biased environment ($S_u$). For the Yahoo! random set, we randomly split the user-item interactions into three subsets: 5% for training in an unbiased environment ($S_u$), 5% for validation to tune the hyper-parameters ($S_{va}$), and the rest 90% for test ($S_t$).

- **Product:** This is a large-scale dataset for CTR prediction, which includes three weeks of users’ click records from a real-world advertising system. The first two weeks’ samples are used for training and the next week’s samples for test. The records containing the exposed items clicked by a user are considered as the positive feedback (i.e., label $y_{ij} = 1$), and the records containing the exposed but not clicked items are randomly sampled as the negative feedback (i.e., label $y_{ij} = -1$). There exists two policies in this dataset: non-uniform policy and uniform policy which are defined in Section 3.1. We can thus separate this dataset into two parts, i.e., a uniform data and a non-uniform data. Note that in order to reduce the influence of position bias in the uniform data to ensure better unbiasedness, we only filter samples at positions 1 and 2. The non-uniform data contains around 29 million records and 2.8 million users, which is directly used as a training set named as $S_u$. Next, we randomly split the uniform data into three subsets by the same way as that of Yahoo! R3, i.e., 5% as training set ($S_u$), 5% as validation set ($S_{va}$), and the rest as test set ($S_t$).

4.1.2 Evaluation Metrics

We employ five evaluation metrics that are widely used in recommender systems, including the area under the ROC curve (AUC-ROC), precision at $k$ (Prec@k), recall at $k$ (Recall@k), average precision (AP), and normalized discounted cumulative gain (NDCG). These metrics are calculated based on the predicted labels and the ground truth without bias. We binarize the ratings based on a threshold $\epsilon = 3$. Hence, a rating $y_{ij} > \epsilon$ is considered as a positive feedback (i.e., label $y_{ij} = 1$), otherwise, it is considered as a negative feedback (i.e., label $y_{ij} = -1$). The Yahoo! user set is used as a training set in a biased environment ($S_u$). For the Yahoo! random set, we randomly split the user-item interactions into three subsets: 5% for training in an unbiased environment ($S_u$), 5% for validation to tune the hyper-parameters ($S_{va}$), and the rest 90% for test ($S_t$).

### Table 2

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Feedback</th>
<th>P/N</th>
<th>#Feedback</th>
<th>P/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yahoo! R3</td>
<td>$S_u$</td>
<td>311,704</td>
<td>67.02%</td>
<td>29,255,580</td>
</tr>
<tr>
<td></td>
<td>$S_t$</td>
<td>2,700</td>
<td>9.36%</td>
<td>20,751</td>
</tr>
<tr>
<td></td>
<td>$S_{vo}$</td>
<td>2,700</td>
<td>8.74%</td>
<td>20,751</td>
</tr>
<tr>
<td></td>
<td>$S_{vc}$</td>
<td>48,600</td>
<td>9.71%</td>
<td>373,522</td>
</tr>
</tbody>
</table>

P/N represents the ratio between the numbers of positive and negative feedback.
We first consider \( \tau \in \{2, 5, 10, 20\} \). The tuning history and optimal parameter file of each method can be found in the link of the source codes.

### 4.2 RQ1: Comparison Results

The comparison results are shown in Table 3. Because an additional baseline MLP is required in feature-based distillation as described in Section 4.1.3, we list its results separately. In addition, since Yahoo! R3 does not provide feature information, we only report the results on Product. As shown in the tables, our methods perform better than all the compared methods in most cases. More specifically, we have the following observations: 1) The sample combination of \( S_c \) and \( S_t \) improves the performance in all cases. The method using only \( S_c \) has an uncompetitive result on Yahoo! R3, but an improved AUC on Product, which is expected because Product contains a large uniform data \( S_t \). The propensity-based method achieves superior performance on Yahoo except on AUC, but the results on Product are degraded on all metrics. One reason is that \( S_c \) and \( S_t \) of Product have a close ratio between the positive and negative feedback, which weakens the effect of the propensity scores. 2) The trends on AUC and other metrics are consistent in most cases. 3) The improvements brought by all the proposed strategies vary in different model implementations and different data scales. It means that each strategy's ability to use \( S_t \) depends on distinct scenarios. In general, when the data scale is small, the label-based strategy can maintain better performance, followed by the sample-based strategy and the model-based strategy (feature-based strategies are excluded due to different baselines for comparison). When the data scale is large, the difference between different strategies will be reduced, especially when the neural network model is used as the skeleton. We will conduct in-depth study on some strategies separately in the future. In addition, we also verify that the proposed new strategies are better than or supplementary to the original strategies, and the comparison results between them can be found in the appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2022.3199585.

### 4.3 RQ2: Impact of Key Factors

In this section, we explore and analyze some key factors that may affect the performance of the proposed framework. We only report experimental results on Yahoo! R3. We first consider the difference in P/N value between \( S_t \) and \( S_c \). The difference in P/N value between \( S_t \) and \( S_c \) may be caused by some bias problems such as selection bias. Different degrees of difference obviously have different effects on the proposed strategies. To simulate the differences, we first set the size of the subset sampled from \( S_c \) to N (for example, N = 100,000 for Yahoo! R3). We control the proportion of positive samples included in the subset to 10\%, 30\%, 50\%, 70\% and 90\%, which correspond to 5 different P/N value scenarios (i.e., the P/N value of the biased subset is \( \frac{1}{2} \), \( \frac{2}{3} \), \( \frac{3}{4} \), \( \frac{4}{5} \) and \( \frac{5}{6} \)). Finally, we use the new biased subset and \( S_c \) to retrain the proposed framework. The results are shown in Figs. 5a and 5b.

When using MF to implement the various proposed strategies, as the proportion of positive samples in the biased subset increases, the performance shows a trend of first increasing...
and then decreasing, and most of them reach a peak when the proportion is 50%. When the proportion of positive samples is 10%, the P/N value of the biased subset is close to the P/N value of $S_t$, which damages the performance of the proposed strategies. Because in this paper we only consider the feedback information, small differences in label information lead to a strategy not being able to extract useful knowledge from $S_t$ well. However, when the difference between the positive sample ratio of the biased data and $S_t$ is too large (i.e., the right side of Fig. 5a), the role of these useful knowledge will be reduced to a certain extent. Note that existing IPS methods are more susceptible to this effect due to excessive correction of the sample distribution. We can observe similar results when using AE to implement the various suggested strategies, as shown in Fig. 5b. An interesting observation is that when the proportion of positive samples in the biased subset is too different from $S_t$, the strategies implemented using AE have better robustness, and some strategies still maintain a growing trend.

Next, we consider another key factor affecting the performance of the proposed framework, i.e., the size of $S_t$. Since the core of the proposed framework is to learn some useful unbiased knowledge from $S_t$, different sizes of $S_t$ obviously have different effects on the proposed strategy. To simulate...

### TABLE 3
Comparison Results of Unbiased Evaluation

<table>
<thead>
<tr>
<th>Module</th>
<th>Strategy</th>
<th>Low Rank (MF)</th>
<th>Neural Nets (AE)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>AUC</td>
<td>NDCG</td>
</tr>
<tr>
<td>Baseline</td>
<td>Base ($S_t$)</td>
<td>+0.00%</td>
<td>+0.00%</td>
</tr>
<tr>
<td></td>
<td>Uniform ($S_t$)</td>
<td>-21.76%</td>
<td>-15.46%</td>
</tr>
<tr>
<td></td>
<td>Combine (L)</td>
<td>+0.27%</td>
<td>+3.78%</td>
</tr>
<tr>
<td></td>
<td>Propensity (L)</td>
<td>-0.78%</td>
<td>+59.11%</td>
</tr>
<tr>
<td>Label</td>
<td>Refine-var (L)</td>
<td>+2.08%</td>
<td>+98.28%</td>
</tr>
<tr>
<td></td>
<td>Bridge-var1 (L)</td>
<td>+0.80%</td>
<td>+21.65%</td>
</tr>
<tr>
<td></td>
<td>Bridge-var2 (L)</td>
<td>+1.06%</td>
<td>+47.08%</td>
</tr>
<tr>
<td>Sample</td>
<td>Delay (L)</td>
<td>+0.74%</td>
<td>+19.59%</td>
</tr>
<tr>
<td></td>
<td>Cause (L)</td>
<td>+0.00%</td>
<td>-1.03%</td>
</tr>
<tr>
<td></td>
<td>Weights (1)</td>
<td>+0.77%</td>
<td>+36.08%</td>
</tr>
<tr>
<td></td>
<td>Weights (g)</td>
<td>+0.55%</td>
<td>+19.93%</td>
</tr>
<tr>
<td>Model</td>
<td>FeatEa (L)</td>
<td>+0.60%</td>
<td>+25.77%</td>
</tr>
<tr>
<td>Structure</td>
<td>FeatEc (L)</td>
<td>+0.33%</td>
<td>+8.25%</td>
</tr>
<tr>
<td></td>
<td>Hint (L)</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Soft Label (L)</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

*Note that since these strategies rely on deep networks, we use the deep version of the base strategy as a reference to report the results, which is Deep AutoEncoder.*
the different sizes of $S_t$, we randomly sample a subset of $S_t$ according to the ratio of 20%, 40%, 60% and 80%, which corresponds to four different size scenarios. Finally, we retrain the proposed framework using $S_t$ and the new unbiased subset. The results are shown in Figs. 5c and 5d.

When using MF to implement the various proposed strategies, the performance improves as the scale of $S_t$ increases. A small number of strategies show slight declines on the right side of Fig. 5c. This may mean that these strategies have reached the upper limit of their use of $S_t$. Note that the existing IPS method cannot obtain a large gain from the scale of $S_t$. This may be because the scale of $S_t$ does not cause a large change in its own distribution. Similarly, we can observe similar results when using AE to implement these strategies, as shown in Fig. 5d. The difference from Fig. 5c is that there is no longer a significantly better strategy. This may be because a smaller-scale uniform data in the neural network limits the performance improvement.

In our experiments, the original number of $S_t$ is 2,700, and the corresponding number on the x-axis of Figs. 5c and 5d is 540 to 2,160.

4.4 RQ3: Analysis Results of the Proposed New Strategies

Next, we analyze and discuss some specific factors for the proposed new strategies. As described in the Bridge-var2 strategy and WeightS-l strategy (in Sections 3.1 and 3.3), we use the form of Sigmoid mode to construct the similarity of model $M_k$ and model $M_l$ in parameters, i.e., $\text{sigmoid} \left( \frac{\cos(W_i^k, W_i^l) + \cos(W_j^k, W_j^l)}{2} \right)$. Naturally, these two strategies may have different performance when using different modes to construct similarity. In the experiments we consider three modes, including Sigmoid, Exp and Norm. Exp mode can be described as $\exp \left( \frac{\cos(W_i^k, W_i^l) + \cos(W_j^k, W_j^l)}{2} \right)$, and Norm mode can be described as $\frac{\cos(W_i^k, W_i^l) + \cos(W_j^k, W_j^l)}{2}$. The results are shown in Table 4, from which we observe that Sigmoid mode and Exp mode are better choices under different conditions.

For the Refine-var strategy, the two key factors are the number of samples from $S_c$ and the mode of sampling. In the experiments, we first fix the number of samples from $S_c$ to $[200, 400, \ldots, 2400, 2600, 2700]$, and then retrain and evaluate the model. Note that we constrain the upper limit of the number of biased subsets to be the same as the number of $S_t$ (i.e., 2,700), because combining with a too large biased subset may damage the unbiased knowledge of $S_t$. We show the results in Fig. 6. We use the dotted line to represent the result of the original Refine strategy, i.e., only use $S_t$ to train the

TABLE 4

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Mode</th>
<th>Low Rank (MF) AUC</th>
<th>Neural Nets (AE) AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bridge-var2</td>
<td>Sigmoid</td>
<td>0.7331</td>
<td>0.7226</td>
</tr>
<tr>
<td></td>
<td>Exp</td>
<td>0.7314</td>
<td>0.7249</td>
</tr>
<tr>
<td></td>
<td>Norm</td>
<td>0.6744</td>
<td>0.7221</td>
</tr>
<tr>
<td>WeightS-l</td>
<td>Sigmoid</td>
<td>0.7352</td>
<td>0.7307</td>
</tr>
<tr>
<td></td>
<td>Exp</td>
<td>0.7359</td>
<td>0.7247</td>
</tr>
<tr>
<td></td>
<td>Norm</td>
<td>0.6849</td>
<td>0.7306</td>
</tr>
</tbody>
</table>
imputation model. We observe that as the number of samples from $S_c$ increases, the performance of the Refine-var strategy increases. In addition, we also notice that this upward trend is not linear, i.e., the performance may remain flat within a certain number of biased subsets.

As described in Refine-var strategy, we use a random mode in the previous experiments to sample a biased subset from $S_c$. We then control the sampling modes, including In-user, Out-user, Head-item and Tail-item modes. In the In-user mode, we only sample from the overlapping user sets between $S_c$ and $S_t$, while the Out-user mode is the opposite. We sort the items in descending order according to their popularity, and divide the items into head items and tail items at a ratio of 3:7. In the Head-item mode, we constrain to sample only the samples containing the head items from $S_c$, while the Tail-item mode is the opposite. The results are shown in Table 5. We observe that except for the Tail-item mode, the other four modes have similar results. This means that enhancing or supplementing the user information in $S_t$ and enhancing the information of popular items in $S_c$ are beneficial to mining some unbiased knowledge.

### 4.5 RQ4: Item Distribution of the Recommendation Lists

Finally, we conduct a preliminary experiment from the perspective of item distribution to analyze how the proposed strategy can effectively improve the debiasing performance. We sort the items in Yahoo! R3 in descending order according to the corresponding popularity, and then regard the top 20% of the items as popular items, and the rest as unpopular items. We show the distribution of $S_t$ in Fig. 7a, from which we can find that although the probability of the popular and unpopular items being recommended is indeed even (e.g., popular items account for 20% of the total items, and the probability of being recommended also accounts for 20%), but the utility (i.e., the probability of hit divided by the probability of being recommended) brought by popular items is still higher. We only report the results of using MF as a skeleton model on Yahoo! R3.

Combining Figs. 7b and 7c we can observe: 1) To approximate the uniform distribution of $S_{va}$, MF trained on $S_c$, which suffers from the popularity bias, tends to capture the global distribution of popular and unpopular items similar to Fig. 7a (i.e., the ratio of 2:8). But unreasonably displaying too many unpopular items may not bring much benefit. 2) Uniform trained on $S_t$ can avoid being misled by the global distribution and capture the property that popular items have higher utility. Therefore, in order to better sort popular items in $S_{va}$, it retains a slightly more proportion of popular items than MF. However, due to the limitation of the data size, it cannot further accurately identify users’ preferences on popular and unpopular items. Combine is a trade-off between those two, and its effectiveness is shown in Table 3. 3) The proposed strategies also play a role in adjusting the proportion of popular items. But different from the baseline, they can maintain a good utility on both the adjusted two parts of the item, especially on the unpopular items. We have also observed that as the degree of adjustment increases, i.e., the proportion of popular items is higher, the two best strategies in Table 3 (i.e., Refine and Bridge-var2) can have a significant utility advantage in both parts of the items. The above results show that the proposed strategy can effectively integrate the information of the log data and uniform data, which can more accurately learn users’ preferences on unpopular items, and maintain high utility on popular items. We believe this will produce a more ideal recommendation policy.

### TABLE 5

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Mode</th>
<th>AUC (MF)</th>
<th>AUC (AE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Rank</td>
<td>Random</td>
<td>0.7426</td>
<td>0.7283</td>
</tr>
<tr>
<td></td>
<td>In-user</td>
<td>0.7432</td>
<td>0.7277</td>
</tr>
<tr>
<td>Refine-var</td>
<td>Out-user</td>
<td>0.7419</td>
<td>0.7285</td>
</tr>
<tr>
<td></td>
<td>Head-item</td>
<td>0.7435</td>
<td>0.7282</td>
</tr>
<tr>
<td></td>
<td>Tail-item</td>
<td>0.7357</td>
<td>0.7170</td>
</tr>
</tbody>
</table>

---

**Fig. 6.** The results of Refine-var strategy when sampling different sample sizes from $S_c$.

**Fig. 7.** Item distribution and corresponding utility of different strategies on Yahoo! R3.

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on August 28, 2023 at 05:40:28 UTC from IEEE Xplore. Restrictions apply.
5 Future Directions

Counterfactual recommendation via a uniform data is still a rich research field. In this section, we discuss some interesting and promising future directions: 1) Label-Based Module. The distribution difference between $S_i$ and $S_l$ is a key factor affecting this module. It is necessary to design strategies that are more robust to different degrees of difference, such as voting using multiple imputation models. 2) Feature-Based Module. The current approach only makes use of the feature information in each sample to learn the stable features, while the label in each sample from $S_l$ is more stable and unbiased. So how to filter out the stable features with both labels and features in $S_l$ is another interesting research question. 3) Sample-Based Module. The difference between the data size of $S_l$ and that of $S_i$ is a challenge for sample-based methods. One promising direction is to use the information in $S_l$ to filter out a more unbiased subset from $S_{op}$ or use the information in $S_l$ to perform data augmentation on $S_{op}$. Instead of using the label information, another promising direction is that we can consider modeling the preference ranking relation between $S_l$ and $S_i$. 4) Model Structure-Based Module. The current distillation structure selection methods are based on enumeration or empirical methods. How to effectively design a good distillation structure is another promising direction, for which AutoML has the potential to find a reasonable model structure based on $S_i$.

In addition, there are also many other directions closely related to the framework. For example, the visualization or interpretation of the useful information (or knowledge) learned from $S_l$; further exploration of the results at a micro level, i.e., the impact on each user or each item; and the relation between the size of $S_l$ and the performance of the model. In addition, we would like to further investigate the trade-off of training on $S_i$ introduced by $S_l$ and gain more theoretical insights into why it is effective. These theoretical insights can also inspire us to design better distillation strategies.

6 Conclusion

In this work, motivated by the observation that simply modeling with a uniform data can alleviate the bias problems, we propose a novel and general knowledge distillation framework for counterfactual recommendation via uniform data, i.e., KDCRec, including label-based, feature-based, sample-based and model structure-based distillations. We conduct extensive experiments on both public and product datasets, demonstrating that the proposed four methods can achieve better performance over the baseline models. In addition, we also explore and analyze the performance changes of the proposed methods on some key factors. Finally, we discuss some promising directions worthy of further exploration.

References


Weike Pan received the PhD degree in computer science and engineering from the Hong Kong University of Science and Technology, Kowloon, Hong Kong, China, in 2012. He is currently an associate professor with the National Engineering Laboratory for Big Data System Computing Technology and the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include transfer learning, federated learning, recommender systems and machine learning. He has published more than 60 refereed international conference and journal papers. He was the recipient of the ACM TiS 2016 Best Paper Award and SDM 2013 Nominee of Best Paper Award.

Zhong Ming received the PhD degree in computer science and technology from Sun Yat-Sen University, Guangzhou, China, in 2003. He is currently a professor with the National Engineering Laboratory for Big Data System Computing Technology and the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include software engineering and web intelligence. He has published more than 200 refereed international conference and journal papers (including more than 35 ACM/IEEE Transactions papers). He was the recipient of the ACM TiS 2016 Best Paper Award and some other best paper awards.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.