RecBole: Towards a Unified, Comprehensive and Efficient Framework for Recommendation Algorithms

Wayne Xin Zhao[1,2], Shanlei Mu[1,3,#], Yupeng Hou[1,2,#], Zihan Lin[1,3], Yushuo Chen[1,2],
Xingyu Pan[1,3], Kaiyuan Li[4], Yujie Lu[7], Hui Wang[1,3], Changxin Tian[1,3],
Yingqian Min[1,3], Zhichao Feng[4], Xinyan Fan[1,2], Xu Chen[1,2,*], Pengfei Wang[4,*],
Wendi Ji[5], Yaliang Li[6], Xiaoling Wang[5], Ji-Rong Wen[1,2,3]
[1]Beijing Key Laboratory of Big Data Management and Analysis Methods
{[2]Gaoling School of Artificial Intelligence, [3]School of Information} Renmin University of China
[4]Beijing University of Posts and Telecommunications, [5]East China Normal University, [6]Alibaba, [7]Liaoning University

ABSTRACT

In recent years, there are a large number of recommendation algorithms proposed in the literature, from traditional collaborative filtering to deep learning algorithms. However, the concerns about how to standardize open source implementation of recommendation algorithms continually increase in the research community. In the light of this challenge, we propose a unified, comprehensive and efficient recommender system library called *RecBole* (pronounced as [rɛk'boʊlər]), which provides a unified framework to develop and reproduce recommendation algorithms for research purpose. In this library, we implement 73 recommendation models on 28 benchmark datasets, covering the categories of general recommendation, sequential recommendation, context-aware recommendation and knowledge-based recommendation. We implement the RecBole library based on PyTorch, which is one of the most popular deep learning frameworks. Our library is featured in many aspects, including general and extensible data structures, comprehensive benchmark models and datasets, efficient GPU-accelerated execution, and extensive and standard evaluation protocols. We provide a series of auxiliary functions, tools, and scripts to facilitate the use of this library, such as automatic parameter tuning and break-point resume. Such a framework is useful to standardize the implementation and evaluation of recommender systems. The project and documents are released at https://recbole.io/.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Web applications**; **Web mining**.

KEYWORDS

recommendation toolkit, collaborative filtering, recommendation framework

ACM Reference Format:
Wayne Xin Zhao[1,2], Shanlei Mu[1,3,#], Yupeng Hou[1,2,#], Zihan Lin[1,3], Yushuo Chen[1,2],, Xingyu Pan[1,3], Kaiyuan Li[4], Yujie Lu[7], Hui Wang[1,3], Changxin Tian[1,3],, Yingqian Min[1,3], Zhichao Feng[4], Xinyan Fan[1,2], Xu Chen[1,2,*], Pengfei Wang[4,*],, Wendi Ji[5], Yaliang Li[6], Xiaoling Wang[5], Ji-Rong Wen[1,2,3]. 2021. RecBole: Towards a Unified, Comprehensive and Efficient Framework for Recommendation Algorithms. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3459637.3482016

1 INTRODUCTION

In the era of big data, recommender systems are playing a key role in tackling information overload, which largely improve the user experiences in a variety of applications, ranging from e-commerce, video sharing to healthcare assistant and on-line education. The huge business value makes recommender systems become a longstanding research topic, with a large number of new models proposed each year [83]. With the rapid growth of recommendation algorithms, these algorithms are usually developed under different platforms or frameworks. Typically, an experienced researcher often finds it difficult to implement the compared baselines in a unified way or framework. Indeed, many common components or procedures of these recommendation algorithms are duplicate or highly similar, which should be reused or extended. Besides, we are aware that there is an increasing concern about model reproducibility in the research community. Due to some reasons, many published recommendation algorithms still lack public implementations. Even with open source code, many details are implemented inconsistently (e.g., with different loss functions or optimization strategies) by different developers. There is a need to re-consider the implementation of recommendation algorithms in a unified way.

In order to alleviate the above issues, we initiate a project to provide a unified framework for developing recommendation algorithms. We implement an open source recommender system library, called *RecBole* (pronounced as [rɛk'boʊlər]) [1]. Based on this library,

[*] Xu Chen (successcx@gmail.com) and Pengfei Wang (wangpengfei@bupt.edu.cn) are corresponding authors.
[#] Both authors contributed equally to this work.

[1] Bole was a famous Chinese judge of horses in Spring and Autumn period, who was the legendary inventor of equine physiognomy ("*judging a horse's qualities from appearance*"). Bole is frequently associated with the fabled *qianlima* (a Chinese word) "thousand-*li* horse", which was supposedly able to gallop one thousand *li* (approximately 400 km) in a single day. Read more details about Bole at the wikipedia page

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482016

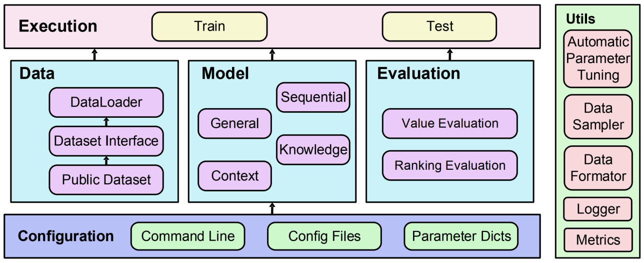


Figure 1: The overall framework of our library RecBole.

we would like to enhance the reproducibility of existing models and ease the developing process of new algorithms. Our work is also useful to standardize the evaluation protocol of recommendation algorithms. Indeed, a considerable number of recommender system libraries have been released in the past decade [14, 15, 59, 64, 74]. These works have largely advanced the progress of open source recommender systems. Many libraries have made continuous improvement with increasingly added features. We have extensively surveyed these libraries and broadly fused their merits into RecBole. To summarize, the key features and capabilities of our RecBole library are summarized in the following five aspects:

• Unified recommendation framework. We adopt PyTorch [46] to develop the entire recommender system library, since it is one of the most popular deep learning frameworks, especially in the research community. As three core components of our library, we design and develop data modules, model modules, and evaluation modules, and encapsulate many common components, functions or procedures shared by different recommendation algorithms. In our library, for reusing existing models, one can easily compare different recommendation algorithms with built-in evaluation protocols via simple yet flexible configuration; for developing new models, one only needs to focus on a small number of interface functions, so that common parts can be reused and implementation details are made transparent to the users.

• General and extensible data structure. For unified algorithm development, we implement the supporting data structures at two levels. At the user level, we introduce *atomic files* to format the input of mainstream recommendation tasks in a flexible way. The proposed atomic files are able to characterize the input of four kinds of mainstream recommendation tasks. At the algorithm level, we introduce a general data structure Interaction to unify the internal data representations tailored to GPU-based environment. The design of Interaction is particularly convenient to develop new algorithms with supporting mechanisms or functions, e.g., fetching the data by referencing feature name. We implement Dataset and DataLoader (two python classes) to automate the entire data flow, which greatly reduces the efforts for developing new models.

• Comprehensive benchmark models and datasets. So far, we have implemented 73 recommendation algorithms, covering the categories of general recommendation, sequential recommendation, context-aware recommendation and knowledge-based recommendation. Besides traditional recommendation algorithms, we incorporate a large number of neural algorithms proposed in recent years. We provide flexible supporting mechanisms via the configuration files or command lines to run, compare and test these algorithms. We also implement rich auxiliary functions to use these models, including automatic parameter tuning and break-point resume. To construct a reusable benchmark, we incorporate 28 commonly used datasets for evaluating recommender systems. With original dataset copies, a user can simply transform the data into a form that can be used in our library with the provided preprocessing tools or scripts. More datasets and methods will be incorporated into our library.

• Efficient GPU-accelerated execution. We design and implement a number of efficiency optimization techniques that are tailored to the GPU environment. As two major sources of time costs, both the model training and testing are accelerated with GPU-oriented implementations. For model test, a special acceleration strategy is proposed to improve the efficiency of the full ranking for top-K item recommendation. We convert the top-K evaluation for all the users into the computation based on a unified matrix form. With this matrix form, we can utilize the GPU-version topk() function in PyTorch to directly optimize the top-K finding procedure. Furthermore, such a matrix form is particularly convenient for generating the recommendations and computing the evaluation metrics. We empirically show that it significantly reduces the time cost of the straightforward implementation without our acceleration strategy.

• Extensive and standard evaluation protocols. Our library supports a series of widely adopted evaluation protocols for testing and comparing recommendation algorithms. It incorporates the various evaluation settings discussed in [87]. Specially, we implement different combinations of item sorting (i.e., how to sort the items before data splitting) and data splitting (i.e., how to derive the train/validation/test sets) for deriving the evaluation sets. We also consider both full ranking and sample-based ranking, which is recently a controversial issue in the field of recommender system [32]. We encapsulate four basic interfaces (namely *Group*, *Split*, *Order* and *NegSample*) to support the above evaluation protocols, which is flexible to include other evaluation settings. We provide a few commonly used evaluation settings (e.g., ratio-based splitting plus random ordering for dataset splitting), which integrates the alternative settings of the above four factors. Our library provides a possibility to evaluate models under different evaluation settings.

2 THE LIBRARY — RECBOLE

The overall framework of our library RecBole is presented in Figure 1. The bottom part is the configuration module, which helps users to set up the experimental environment (e.g., hyperparameters and running details). The data, model and evaluation modules are built upon the configuration module, which forms the core code of our library. The execution module is responsible for running and evaluating the model based on specific settings of the environment. All the auxiliary functions are collected in the utility module, including automatic parameter tuning, logger and evaluation metrics. In the following, we briefly present the designs of three core modules, and more details can be found in the library documents.

2.1 Data Module

A major development guideline of our library is to make the code highly self-contained and unified. For this purpose, data module is

via the link: https://en.wikipedia.org/wiki/Bo_Le. Here, we make an analogy between identifying qianlima horses and making good recommendations.

Table 1: Collected datasets in our library RecBole.

Dataset	#Users	#Items	#Interactions
MovieLens	-	-	-
Anime	73,515	11,200	7,813,737
Epinions	116,260	41,269	188,478
Yelp	1,968,703	209,393	8,021,122
Netflix	480,189	17,770	100,480,507
Book-Crossing	105,284	340,557	1,149,780
Jester	73,421	101	4,136,360
Douban	738,701	28	2,125,056
Yahoo Music	1,948,882	98,211	11,557,943
KDD2010	-	-	-
Amazon	-	-	-
Pinterest	55,187	9,911	1,445,622
Gowalla	107,092	1,280,969	6,442,892
Last.FM	1,892	17,632	92,834
DIGINETICA	204,789	184,047	993,483
Steam	2,567,538	32,135	7,793,069
Ta-Feng	32,266	23,812	817,741
FourSquare	-	-	-
Tmall	963,923	2,353,207	44,528,127
YOOCHOOSE	9,249,729	52,739	34,154,697
Retailrocket	1,407,580	247,085	2,756,101
LFM-1b	120,322	3,123,496	1,088,161,692
Criteo	-	-	45,850,617
Avazu	-	-	40,428,967
iPinYou	19,731,660	163	24,637,657
Phishing websites	-	-	11,055
Adult	-	-	32,561
MIND	-	-	-

[1] "-" means the dataset is either composed of many small subsets (e.g., Amazon, KDD2010), so that we refer the readers to our website for more detail statistics or the dataset is based on the features (e.g., Criteo, Avazu), instead of users and items.

indeed the most important part that supports the entire library by providing fundamental data structures and functions.

2.1.1 The Overall Data Flow. For extensibility and reusability, our data module designs an elegant data flow that transforms raw data into the model input.

The overall data flow can be described as follows: raw input → atomic files → $\text{Dataset}_{DataFrame}$ → $\text{Dataloader}_{Interaction}$ → algorithms. The implementation of class Dataset is mainly based on the primary data structure of pandas.DataFrame in the library of pandas, and the implementation of class Dataloader is based on a general internal data structure called Interaction.

Our data flow involves two special data forms, which are oriented to users and algorithms, respectively. For data preparation, we introduce and define six *atomic file types* (having the same or similar file format) for unifying the input at the user level. While, for internal data representations, we introduce and implement a flexible data structure Interaction at the algorithm level. The atomic files are able to characterize most forms of the input data required by different recommendation tasks, and the Interaction data structure provides a unified internal data representation for different recommendation algorithms.

Table 2: The functions supported by class Dataset.

Function	Description
_filter_by_inter_num	frequency based user/item filtering
_filter_by_field_value	value based filtering
_remap_ID	map the features to IDs
_fill_nan	missing value imputation
_set_label_by_threshold	generate interaction labels
_normalize	normalize the features
_preload_weight_matrix	initialize embedding tables

Table 3: Summarization of the atomic files.

Suffix	Data types	Content
.INTER	all types	User-item interaction
.USER	all types	User feature
.ITEM	all types	Item feature
.KG	int	Triplets in a knowledge graph
.LINK	int	Item-entity linkage data
.NET	all types	Social graph data

In order to help users transform raw input into atomic files, we have collected more than 28 commonly used datasets and released the corresponding conversion tools, which makes it quite convenient to start with our library. We present the statistics of these datasets in Table 1. During the transformation step from atomic files to class Dataset, we provide many useful functions that support a series of preprocessing steps in recommender systems, such as k-core data filtering and missing value imputation. We present the functions supported by class Dataset in Table 2.

2.1.2 Atomic Files. So far, our library introduces six atomic file types, which are served as basic components for characterizing the input of various recommendation tasks. In the literature, there is a considerable number of recommendation tasks. We try to summarize and unify the most basic input forms for mainstream recommendation tasks. Note that these files are only functionally different while their formats are rather similar. The details of these atomic files are summarized in Table 3.

We identify different files by their suffixes. By summarizing existing recommendation models and datasets, we conclude with four basic data types, i.e., "token" (representing integers or strings), "token sequence", "float" and "float sequence". "token" and "token sequence" are used to represent discrete features such as ID or category, while "float" and "float sequence" are used to represent continuous features, such as price. Atomic files support sparse feature representations, so that the space taken by the atomic files can be largely reduced. Most of atomic files support all the four data types except the .KG and .LINK files. Next, we present the detailed description of each atomic file:

- .INTER is a mandatory file used in all the recommendation tasks. Each line is composed of the user ID (token), item ID (token), user-item rating (float, optional), timestamp (float, optional) and review text (token sequence, optional). Different fields are separated by commas.
- .USER is a user profile file, which includes users' categorical or continuous features. Each line is formatted as user ID (token), feature (token or float), feature (token or float), ..., feature (token or float).
- .ITEM is an item feature file, which describes the item characteristics, and the format is as follows: item ID (token), feature (token or float), feature (token or float), ..., feature (token or float). .USER and .ITEM are used for context-aware recommendation.
- .KG is a knowledge graph file used for knowledge-based recommendation. Each line corresponds to a $\langle head, tail, relation \rangle$ triplet, and the format is as follows: head entity ID (token), tail entity ID (token), relation ID (token).
- .LINK is also used for knowledge-based recommendation. It records the correspondence between the recommender systems items and the knowledge graph entities. The file format is as follows: item ID (token), entity ID (token), which denotes the item-to-entity mapping.
- .NET is a social network file used for social recommendation. The format is as follows: source user ID (token), target user ID (token), weight (float, optional).

The essence of the atomic files is feature-based data frames corresponding to different parts of the task input. They can cover the input of most mainstream recommendation tasks in the literature. In case the atomic files are not sufficient to support new tasks, one can incrementally introduce new atomic files in a flexible way.

2.1.3 *Input Files for Recommendation Tasks.*

Based on the above atomic files, we can utilize a series of file combinations to facilitate five mainstream recommendation tasks, namely *general recommendation*, *context-aware recommendation*, *knowledge-based recommendation*, *sequential recommendation* and *social recommendation*. Currently, we have implemented the supporting mechanisms for the first four kinds of recommendation tasks, while the code for social recommendation is under development.

The correspondence between atomic files and recommendation models are presented in Table 4. A major merit of our input files is that atomic files themselves are not dependent on specific tasks. As we can see, given a dataset, the user can reuse the same .INTER file (without any modification on data files) when switching between different recommendation tasks. Our library reads the configuration file and determines what to do with the data files.

Another note is that Table 4 presents the combination of mandatory atomic files in each task. It is also possible to use additional atomic files besides mandatory files. For example, for sequential recommendation, we may also need to use context features. To support this, one can simply extend the original combination to ⟨.INTER, .USER, .ITEM⟩ as needed.

2.1.4 *The Internal Data Structure INTERACTION.*

As discussed in Section 2.1.1, in our library, INTERACTION is the internal data structural that is fed into the recommendation algorithms.

Table 4: Correspondence between the recommendation task and the atomic files.

Tasks	Mandatory atomic files
General Recommendation	.INTER
Context-aware Recommendation	.INTER, .USER, .ITEM
Knowledge-based Recommendation	.INTER, .KG, .LINK
Sequential Recommendation	.INTER
Social Recommendation	.INTER, .NET

Table 5: The functions that class Interaction supports.

Function	Description
to(device)	transfer tensors to torch.device
cpu	transfer all tensors to CPU
numpy	transfer all tensors to numpy.Array
repeat	repeats along the batch_size dimension
repeat_interleave	repeat elements of a tensor
update	update an object with other Interaction

In order to make it unified and flexible, it is implemented as a new abstract data type based on python.dict, which is a key-value indexed data structure. The keys correspond to *features* from input, which can be conveniently referenced with feature names when writing the recommendation algorithms; and the values correspond to *tensors* (implemented by torch.Tensor), which will be used for the update and computation in learning algorithms. Specially, the value entry for a specific key stores all the corresponding tensor data in a batch or mini-batch.

With such a data structure, our library provides a friendly interface to implement the recommendation algorithms in a batch-based mode. All the details of the transformation from raw input to internal data representations are transparent to the developers. One can implement different algorithms easily based on unified internal data representation Interaction. Besides, the value components are implemented based on torch.Tensor. We wrap many functions of PyTorch to develop a GRU-oriented data structure, which can support batch-based mechanism (e.g., copying a batch of data to GPU). Specially, we summarize the important functions that Interaction supports in Table 5.

2.2 Model Module

Based on the data module, we organize the implementations of recommendation algorithms in a separate model module.

2.2.1 *Unified Implementation Interface.*

By setting up the model module, we can largely decouple the algorithm implementation from other components, which is particularly important to collaborative development of this library. To implement a new model within the four tasks in Table 4, one only needs to follow the required interfaces to connect with input and evaluation modules, while the details of other parts can be ignored. In specific, we utilize the interface function of calculate_loss($\cdot$) for training and the

Table 6: Implemented 73 recommender models in RecBole on 4 categories.

Category	Model	Conference	Year	Typical Evaluation Dataset
General Recommendation	popularity	-	-	-
	ItemKNN [13]	TOIS	2004	ctlg, ccard, ecmrc, EachMovie, MovieLens, skill
	BPR [51]	UAI	2009	Rossmann, Netflix
	SLIMElastic [44]	ICDM	2011	ctlg, ccard, ecmrc,Book-Crossing, MoiveLens, Netflix, Yahoo Music
	FISM [29]	SIGKDD	2013	MovieLens, Netflix, Yahoo Music
	LINE [61, 88]	WWW	2015	NetWork (Wikipedia, Flickr, Youtube, DBLP)
	CDAE [76]	WSDM	2016	MovieLens, Netflix, Yelp
	NeuMF [23]	WWW	2017	MovieLens, Pinterest
	ConvNCF [21]	IJCAI	2017	Yelp, Gowalla
	DMF [79]	IJCAI	2017	MovieLens, Amazon
	NNCF [4]	CIKM	2017	Delicious, MovieLens, Rossmann
	NAIS [22]	TKDE	2018	MovieLens, Pinterest
	SpectralCF [89]	RecSys	2018	MovieLens, HetRec, Amazon
	MultiVAE [37]	WWW	2018	MovieLens, Million Song, Netflix
	MultiDAE [37]	WWW	2018	MovieLens, Million Song, Netflix
	GCMC [63]	SIGKDD	2018	MovieLens, Flixster,Douban, Yahoo Music
	NGCF [72]	SIGIR	2019	Gowalla, Yelp, Amazon
	MacridVAE [41]	NeurIPS	2019	AliShop-7C, MovieLens, Netflix
	EASE [57]	WWW	2019	MovieLens, Million Song, Netflix
	LightGCN [20]	SIGIR	2020	Gowalla, Yelp, Amazon
	DGCF [73]	SIGIR	2020	Gowalla, Yelp, Amazon
	RaCT [39]	ICLR	2020	MovieLens, Million Song, Netflix
	RecVAE [55]	WSDM	2020	MovieLens, Million Song, Netflix
	ENMF [9]	TOIS	2020	Ciao, Epinions, MovieLens
Context-aware recommendation	LR [53]	WWW	2007	Microsoft web search dataset
	FM [49]	ICDM	2010	CML/PKDD Discovery Challenge 2009, Netflix
	DSSM [26]	CIKM	2013	Henceforth
	FFM [28]	RecSys	2016	Criteo, Avazu
	FNN (DNN) [86]	ECIR	2016	iPinYou
	PNN [47]	ICDM	2016	Criteo, iPinYou
	Wide&Deep [11]	RecSys	2016	Google play dataset
	XGBoost [10]	KDD	2016	Allstate, Higgs Boson, Yahoo LTRC, Criteo
	NFM [19]	SIGIR	2017	Frappe,MovieLens
	DeepFM [16]	IJCAI	2017	Criteo, Company
	AFM [77]	IJCAI	2017	Frappe, MoiveLens
	DCN [70]	ADKDD	2017	Criteo
	LightGBM [31]	NIPS	2017	Allstate, Flight Delay, LETOR, KDD10, KDD12
	xDeepFM [36]	SIGKDD	2018	Criteo, Dianping, Bing News
	FwFM [45]	WWW	2018	Criteo, Oath
	DIN [91]	SIGKDD	2018	Amazon, MovieLens, Alibaba
	DIEN [90]	AAAI	2019	Amazon
	AutoInt [56]	CIKM	2019	Criteo, Avazu, KDD Cup 2012, MovieLens
Sequential recommendation	FPMC [52]	WWW	2010	ROSSMANN
	HRM [69]	SIGIR	2015	Ta-Feng, BeiRen, Tmall
	Improved GRU-Rec [60]	DLRS	2016	YOOCHOOSE
	GRU4RecF(+feature embedding) [24]	RecSys	2016	coined VIDXL, CLASS
	Fossil [18]	ICDM	2016	Amazon, Epinions, Foursquare
	NARM [35]	CIKM	2017	YOOCHOOSE, DIGINETICA
	TransRec [17]	RecSys	2017	Amazon, Epinions, Foursquare, Google Local
	STAMP [38]	SIGKDD	2018	YOOCHOOSE, DIGINETICA
	Caser [62]	WSDM	2018	MovieLens, Gowalla, Foursquare, Tmall
	SASRec [30]	ICDM	2018	Amazon, Steam, MovieLens
	KSR [25]	SIGIR	2018	LastFM, MovieLens, Amazon
	SHAN [80]	IJCAI	2018	Tmall, Gowalla
	NPE [43]	IJCAI	2018	Movielens, Online Retail, TasteProfile
	NextItnet [81]	WSDM	2019	YOOCHOOSE, LastFM
	BERT4Rec [58]	CIKM	2019	Amazon, Steam, MovieLens
	SRGNN [75]	AAAI	2019	YOOCHOOSE, DIGINETICA
	GCSAN [78]	IJCAI	2019	DIGINETICA, Retailrocket
	SASRecF(+feature embedding) [84]	IJCAI	2019	-
	FDSA [85]	IJCAI	2019	Amazon, Tmall
	RepeatNet [48]	AAAI	2019	YOOCHOOSE, DIGINETICA, LastFM
	HGN [40]	SIGKDD	2019	MovieLens, Amazon, Goodreads
	S3Rec [92]	CIKM	2020	Meituan, Amazon, Yelp, LastFM
	GRU+KG Embedding	-	-	-
Knowledge-based recommendation	CKE [82]	SIGKDD	2016	MovieLens, IntentBooks
	CFKG [1]	MDPI	2018	Amazon
	RippleNet [65]	CIKM	2018	MovieLens, Book-Crossing, Bing-News
	KTUP [6]	WWW	2019	MovieLens, DBbook2014
	KGAT [71]	SIGKDD	2019	Amazon, LastFM, Yelp2018
	MKR [67]	WWW	2019	MovieLens, Book-Crossing, LastFM, Bing-News
	KGCN [68]	WWW	2019	MovieLens, Book-Crossing, LastFM
	KGNN-LS [66]	SIGKDD	2019	MovieLens, Book-Crossing, LastFM, Dianping-Food

Table 7: Example evaluation settings.

Notation	Explanation
RO_RS	Random Ordering + Ratio-based Splitting
TO_LS	Temporal Ordering + Leave-one-out Splitting
RO_LS	Random Ordering + Leave-one-out Splitting
TO_RS	Temporal Ordering + Ratio-based Splitting
full	Full ranking with all item candidates
uni$N$	One positive item is paired with N negative items

interface function of predict$(\cdot)$ for testing. To implement a model, what a user needs to do is to implement these important interface functions, without considering other details. These interface functions are indeed general to various recommendation algorithms, so that we can implement various algorithms in a highly unified way. Such a design mode enables quick development of new algorithms. Besides, our model module further encapsulates many important model implementation details, such as the learning strategy. For code reuse, we implement several commonly used loss functions (e.g., BPR loss, margin-based loss, and regularization-based loss), neural components (e.g., MLP, multi-head attention, and graph neural network) and initialization methods (e.g., Xavier's normal and uniform initialization) as individual components, which can be directly used when building complex models or algorithms.

2.2.2 Implemented Models. Until now, we have implemented 73 recommendation models in the four categories of general recommendation, sequential recommendation, context-aware recommendation and knowledge-based recommendation. We refer the readers to Table 6 for more details on these models. When selecting the models to be implemented, we have carefully surveyed the recent literature and selected the commonly used recommendation models and their associated variants (which may not receive high citations) in our library. We mainly focus on the recently proposed neural methods, while also keep some classic traditional methods such as ItemKNN and FM. In the future, more methods will also be incorporated in regular update. For all the implemented models, we have tested their performance on two or four selected datasets, and invited a code reviewer to examine the correctness of the implementation.

2.2.3 Rich Auxiliary Functions. In order to better use the models in our library, we also implement a series of useful functions. A particularly useful function is automatic parameter tuning. The user is allowed to provide a parameter set for searching an optimal value leading to the best performance. Given a set of parameter values, we can indicate four types of tuning methods, i.e., "*Grid Search*", "*Random Search*", "*Tree of Parzen Estimators (TPE)*" and "*Adaptive TPE*". The tuning procedure is implemented based on the library of hyperopt [5]. Besides, we add the functions of model saving and loading to store and reuse the learned models, respectively. Our library also supports the resume of model learning from a previously stored break point. In the training process, one can print and monitor the change of the loss value and apply training tricks such as early-stopping. These tiny tricks largely improve the usage experiences with our library.

2.3 Evaluation Module

The function of evaluation module is to implement commonly used evaluation protocols for recommender systems. Since different models can be compared under the same evaluation module, our library is useful to standardize the evaluation of recommender systems.

2.3.1 Evaluation Metrics. Our library supports both value-based and ranking-based evaluation metrics. The value-based metrics (for rating prediction) include Root Mean Square Error (RMSE) and Mean Average Error (MAE), measuring the prediction difference between the true and predicted values. The ranking-based metrics (for top-K item recommendation) include the most widely used ranking-aware metrics, such as Recall@K, Precision@K, NDCG, and MRR, measuring the ranking performance of the generated recommendation lists by an algorithm.

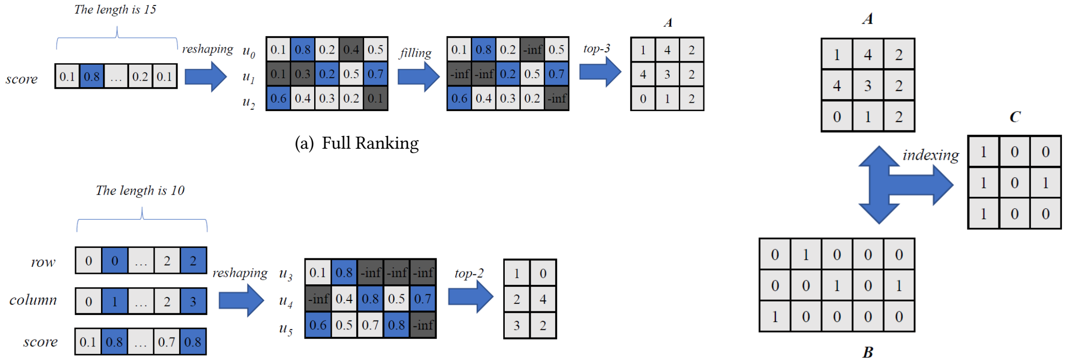
2.3.2 Evaluation Settings. In recent years, there are more and more concerns on the appropriate evaluation of recommender systems [32, 87]. Basically speaking, the divergence mainly lies in the ranking-based evaluation for top-K item recommendation. Note that the focus of our library is not to identify the most suitable evaluation protocols. Instead, we aim to provide most of the widely adopted evaluation protocols (even the most critical ones) in the literature. Our library provides a possibility to compare the performance of various models under different evaluation protocols.

For top-K item recommendation, the implemented evaluation settings cover various settings of our earlier work in [87], where we have studied the influence of different evaluation protocols on the performance comparison of models. In particular, we mainly consider the combinations between item sorting (i.e., how to sort the items before data splitting) and data splitting (i.e., how to derive the train/validation/test sets) for constructing evaluation sets. We also consider both full ranking and sampling-based ranking, which is recently a controversial issue in the field of recommender system [32]. We summarize the supporting evaluation settings by our library in Table 7.

In order to facilitate various evaluation settings, we encapsulate the related functions into four major parts, namely *Group*, *Split*, *Order* and *NegSample*. With these implementations, we can effectively support different evaluation protocols, which is also an appealing feature to use our library.

2.3.3 Acceleration Strategy for Top-K Evaluation. Computing Top-K evaluation metrics is usually time consuming. The basic reason lies in that one need to exhaustively estimate the score for each user-item pair. Since the method of score estimation varies across different models, it is not easy to optimize the entire evaluation procedure in a general way. Therefore, we mainly focus on the step of selecting and generating top K items given the ranking scores.

A problem is that different users have a varying number of ground-truth items in test set (resulting in different-sized user-by-item matrices), which is not suitable for parallel GPU computation in a unified manner. Our approach is to consider all the items, including the ones in the training set (called *training items*). Given n users and m items for consideration, when performing full ranking, we can obtain a $n \times m$ matrix $\boldsymbol{D}$ consisting of the confidence scores from a model over the entire item set. When performing sample-based ranking, we create an $n \times m$ matrix $\boldsymbol{D}$, initializing all elements



(a) Full Ranking

(b) Sample-based Ranking

(c) Indexing

Figure 2: Illustration of the proposed acceleration strategy for top-K item evaluation. Here, $u_0, \cdots, u_5$ denote six users; black, blue and grey boxes denote training items, test items and other candidate items, respectively.

to negative infinity. Then, we fill matrix $\boldsymbol{D}$ with the confidence scores over sampled items. This step is called *reshaping*. When performing full ranking with all item candidates, we provide an option to mask the score of training items. If the user choose to mask, the matrix $\boldsymbol{D}$ obtained in the above step cannot be directly used for top-K prediction. Our solution is to set the scores of training items to negative infinity, and perform the full ranking over the entire item set without removing training items. This step is called *filling*. In this way, all the users correspond to equal-sized evaluation matrices (i.e., $n \times m$) for subsequent computation in full ranking and sample-based ranking and the following steps are the same for both cases.

Then, we utilize the GPU-version topk() function provided by PyTorch to find the top K items with the highest scores for users. The GPU-version topk() function has been specially optimized based on CUDA, which is very efficient in our case. This step is called *topk-finding*. With the topk() function, we can obtain a matrix $\boldsymbol{A}$ with size $n \times K$, which records the original index of the selected top K items. We further generate a binary matrix $\boldsymbol{B}$ of size $n \times m$ to indicate the existence of an item in the test set (blue boxes in Figure 2(a)) and Figure 2(b)). Next, we use each row of matrix $\boldsymbol{A}$ to index the same row in matrix $\boldsymbol{B}$ and obtain a binary matrix $\boldsymbol{C}$ of size $n \times K$, which can be implemented efficiently through gather() function provided by PyTorch. We take the case of full ranking as an example in Figure 2(c). This step is called *indexing*. Finally, we concatenate the matrix $\boldsymbol{C}$ of all the batches. The generated result consists of zeros and ones, which is particularly convenient for computing evaluation metrics. As will be shown next, such an acceleration strategy is able to improve the efficiency for both full ranking and sample-based ranking item recommendation.

2.3.4 Efficiency and Scalability. In this part, we empirically analyze the efficiency improvement yielded by our acceleration strategy and the scalability of the evaluation architecture. Specifically, the classic BPR model [51] is selected for efficiency analysis, since it is one of the most commonly used baselines for top-K recommendation. Besides, its model architecture is pretty simple without the influence of other factors, which is suitable for efficiency analysis. We compare its performance *with* and *without* the acceleration strategy in our implementation. We measure the model performance by the total time that (1) it generates a recommendation list of top ten items for users and (2) computes the metrics (NDCG@10 and Recall@10) over the recommendation list, on all the users. To further analyze the model efficiency on datasets of varying sizes, we use three Movielens datasets [2] (i.e., Movielens-100k, Movielens-1M, and MovieLens-10M) to conduct the experiments. We split one original dataset into train, validation and test sets with a ratio of 8 : 1 : 1. We only count the time for generating top ten recommendations (with full ranking) on the test set. We average the time of ten runs of different implementations. Our experiments are performed on a linux PC with CPU (Intel(R) Xeon(R) Silver 4216, 16 cores, 32 threads, 2.10GHz) and GPU (Nvidia RTX 3090 24G). The results of efficiency comparison are shown in Table 8. From the result we can see that by applying the acceleration strategy, we can significantly speed up the evaluation process. In particular, on the largest dataset MovieLens-10M, the accelerated model can perform the full ranking about two seconds, which indicates that our implementation is rather efficient. Currently, we only compare the entire time with all the acceleration techniques. As future work, we will analyze the contribution of each specific technique in detail. Apart from the superiority of efficiency, the evaluation of *Recbole* is also flexible and extendable. As shown in Figure 3, the *evaluator* is decoupled

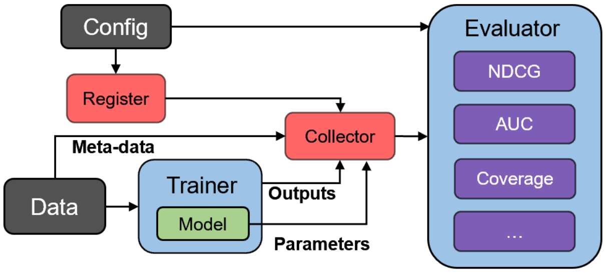
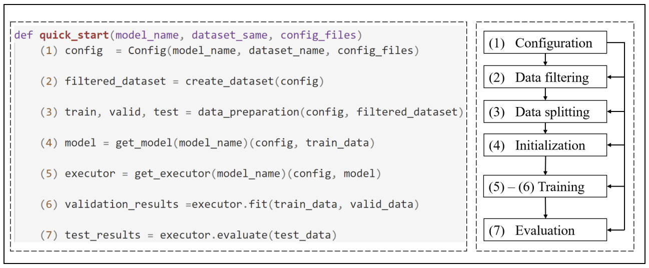


Figure 3: Architecture and dataflow of the evaluation model.

[2] https://grouplens.org/datasets/movielens/

Table 8: Time cost comparison (in second) on different-sized of Movielens datasets with the acceleration strategy or not. BPR^{acc} denotes the model with acceleration strategy.

Model	Dataset		
	MovieLens-100k	MovieLens-1M	MovieLens-10M
BPR	0.245s	2.478s	29.900s
BPR^{acc}	0.009s	0.090s	2.210s



```
def quick_start(model_name, dataset_same, config_files)
    (1) config  = Config(model_name, dataset_name, config_files)

    (2) filtered_dataset = create_dataset(config)

    (3) train, valid, test = data_preparation(config, filtered_dataset)

    (4) model = get_model(model_name)(config, train_data)

    (5) executor = get_executor(model_name)(config, model)

    (6) validation_results =executor.fit(train_data, valid_data)

    (7) test_results = executor.evaluate(test_data)
```

Figure 4: An illustrative usage flow of our library.

with model and data, and all the required resources for calculating metrics are well-wrapped by a *collector*. In this way, it is flexible to develop other customized metrics with these unified interfaces: implement new metrics and sign them in the *register* (see Figure 3).

3 USAGE EXAMPLES OF THE LIBRARY

In this section, we show how to use our library with code examples. We detail the usage description in two parts, namely running existing models in our library and implementing new models based on the interfaces provided in our library.

3.1 Running Existing Models

The contained models in our library can be run with either fixed parameters or auto-tuned parameters.

3.1.1 Model Running with Fixed Parameters. Figure 4 presents a general procedure for running existing models in our library. To begin with, one needs to download and format the raw public dataset based on our provided utils. The running procedure relies on some experimental configuration, which can be obtained from the files, command line or parameter dictionaries. The dataset and model are prepared according to the configured parameters and settings, and the execution module is responsible for training and evaluating the models. The detailed steps are given as follows.

(i) Formatting the dataset. A dataset is firstly selected by a user, and then it is formatted based on the scripts, which can generate required atomic files for different datasets. This procedure takes the following code: atomic_file=PreProcess(dataset).

(ii) Generating the configuration. In our library, the experiment configurations can be generated in different ways. One can write a configuration file, and then read this file in the main function as in line (1) of Figure 4. Another way for configuration is to include parameters in the command line, which is useful for specially focused parameters. At last, one can also directly write the parameter dictionaries in the code.

(iii) Filtering and splitting the dataset. We provide rich auxiliary functions to filter and split the datasets. For example, one can filter the dataset by keeping the users/items with at least K interactions, removing the data occurred in some fixed time period. Different filtering methods can be applied in a unified function (line (2) of Figure 4). When splitting the dataset, one can indicate ratio-based method or leave-one-out method. Then, one can use line (3) of Figure 4 to generate the training, validation and testing sets.

(iv) Loading the model. The next step is to build a recommender model. Given a target model in mind, the user can obtain a model instance according to line (4) of Figure 4.

(v) Training and evaluation. Once the dataset and model are prepared, the user can, at last, train and evaluate the model based on line (5) of Figure 4.

3.1.2 Parameter Tuning. Our library is featured in the capability of automatic parameter (or hyper-parameter) tuning. One can readily optimize a given model according to the provided hyper-parameter range. The general steps are given as follows.

(i) Setting the parameter range. The users are allowed to provide candidate parameter values in the file “hyper.test”. In this file, each line is formatted as: parameter=[value 1,value 2,...value *n*]. Instead of a fixed value, the users can empirically indicate a value set, which will be explored in the following tuning steps.

(ii) Setting the tuning method. Our parameter tuning function is implemented based on the library hyperopt. Given a set of parameter values, we can indicate four types of tuning methods, i.e., “*Grid Search*”, “*Random Search*”, “*Tree of Parzen Estimators (TPE)*” and “*Adaptive TPE*”. The tuning method is invoked by the following code: hy = HyperTuning(objective, tuning_method, range_file), where the parameter range file is used to indicate parameter values.

(iii) Starting the tuning process. The user can start the running process by the following code: hy.run(). With the tuning ranges and method, our library will run the model iteratively, and finally output and save the optimal parameters and the corresponding model performance.

3.2 Implementing a New Model

Based on RecBole, it is convenient to implement a new model by instantiating three functions as follows:

(i) Implementing the “__init__()” function. In this function, the user performs parameter initialization, global variable definition and so on. The new model should be a sub-class of the abstract model class provided in our library. Until now, we have implemented the abstract classes for general recommendation, knowledge-based recommendation, sequential recommendation and context-aware recommendation.

(ii) Implementing the “calculate_loss()” function. This function calculates the loss to be optimized by the new model. Based on the return value of this function, the library will automatically invoke different optimization methods to learn the model according to the pre-set configurations.

(iii) Implementing the “predict()” function. This function is used to predict a score from the input data (e.g., the rating given a user-item pair). This function can be used to compute the loss or derive the item ranking during the model testing phase.

Table 9: Comparison with existing recommender system libraries.

Library	Language	#Models	#Datasets	#Fork	#Star	#Issues	Release time	Neural	PT
MyMediaLite ([14])	C#	61	5	199	477	451	2010	No	manual
LibFM ([50])	C++	1	-	415	1400	32	2014	No	manual
LibRec ([15])	Java	93	11	1000+	3009	252	2014	No	manual
RankSys ([8])	Java	8	-	58	259	38	2016	No	manual
Crab ([7])	Python	2	4	381	1122	75	2011	No	manual
Surprise ([27])	Python	11	3	888	4989	333	2015	No	manual
LightFM ([34])	Python	1	2	610	3741	425	2015	No	manual
Case Recommender ([12])	Python	27	-	75	354	24	2015	No	manual
Recommenders ([3])	Tensorflow	31	5	1900+	10000+	602	2018	Yes	automatic
Cornac ([54])	Tensorflow	42	14	75	397	58	2018	Yes	automatic
NeuRec ([74])	Tensorflow	33	3	199	816	29	2019	Yes	manual
Elliot ([2])	Tensorflow	50	-	19	108	8	2021	Yes	automatic
Spotlight ([33])	PyTorch	8	5	389	2552	109	2017	Yes	automatic
DaisyRec ([59])	PyTorch	20	14	59	354	8	2019	Yes	automatic
ReChorus ([64])	PyTorch	12	2	47	214	16	2020	Yes	manual
Beta-recsys ([42])	PyTorch	22	21	25	75	120	2020	Yes	manual
RecBole	**PyTorch**	**73**	**28**	**193**	**1179**	**163**	**2020**	**Yes**	**automatic**

[1] Neural means the libarary support deep recommender models, and PT denotes parameter tuning. The statistics were collected on the date of Aug 21, 2021.

4 COMPARISON WITH EXISTING LIBRARIES

In recent years, a considerable number of open source recommender system libraries have been released for research purpose. We summarize and compare the characteristics of existing recommender system libraries in Table 9, from which, we can see: the programming language of these libraries gradually evolves from C/C++/JAVA to Python/Tensorflow/PyTorch. From the model perspective, recent libraries mostly support neural recommender models, which agrees with the advance trend in the recommendation domain. In our framework, we select PyTorch as the basic deep learning framework for development, due to its friendly features like easy debugging, compatible with numpy and etc.

RecBole provides the most comprehensive models and benchmark datasets among existing libraries, which can better free the users from the heavy model re-programming work. In addition to reproduce the existing models, we aim to ease the developing process of new algorithms. We design general and extensible underlying data structures to support the unified development framework. By providing a series of useful tools, functions and scripts (e.g., automatic parameter tuning), our library is particularly convenient to be used for scientific research.

At last, we believe that implementation is only the first step for open source recommendation library, as more efforts are needed to maintain and update the library according to users’ feedbacks and suggestions. Our team is working hard to respond to the GitHub issues and fix possible bugs (134 issues were solved until Aug 21, 2021). After release, our library has received much attention from the users. To the date of publication, it is ranked at the third and fourth places based on the number of received stars for the topic of “recommender system” and “recommendation system”, respectively.

5 CONCLUSION

In this paper, we have released a new recommender system library called RecBole. So far, we have implemented 73 recommendation algorithms on 28 commonly used datasets. We design general and extensible data structures to offer a unified development framework for new recommendation algorithms. We also support extensive and standard evaluation protocols to compare and test different recommendation algorithms. Besides, our library is implemented in a GPU-accelerated way, involving a series of optimization techniques for achieving efficient execution. The RecBole library is expected to improve the reproducibility of recommendation models, ease the developing process of new algorithms, and set up a benchmark framework for the field of recommender system. In the future, we will make continuous efforts to add more datasets and models. We will also consider adding more utils for facilitating the usage of our library, such as result visualization and algorithm debugging.

6 ACKNOWLEDGMENT

This work was partially supported by the National Natural Science Foundation of China under Grant No. 61872369, 61802029 and 61972155, Beijing Academy of Artificial Intelligence (BAAI) under Grant No. BAAI2020ZJ0301, and Beijing Outstanding Young Scientist Program under Grant No. BJJWZYJH012019100020098. We also sincerely thank non-author team members (Chen Yang, Zheng Gong, Chenzhan Shang, Zihan Song, Ze Zhang, Jingsen Zhang, Lanling Xu, Zhen Tian and Wenjing Yue) for testing our library.

REFERENCES

[1] Qingyao Ai, Vahid Azizi, Xu Chen, and Yongfeng Zhang. 2018. Learning Heterogeneous Knowledge Base Embeddings for Explainable Recommendation. *Algorithms* 11, 9 (2018), 137.

[2] Vito Walter Anelli, Alejandro Bellogín, Antonio Ferrara, Daniele Malitesta, Felice Antonio Merra, Claudio Pomo, Francesco Maria Donini, and Tommaso Di Noia. 2021. Elliot: A Comprehensive and Rigorous Framework for Reproducible Recommender Systems Evaluation. In *SIGIR*. ACM, 2405–2414.

[3] Andreas Argyriou, Miguel González-Fierro, and Le Zhang. 2020. Microsoft Recommenders: Best Practices for Production-Ready Recommendation Systems. In *Companion of The 2020 Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*. 50–51.

[4] Ting Bai, Ji-Rong Wen, Jun Zhang, and Wayne Xin Zhao. 2017. A Neural Collaborative Filtering Model with Interaction-based Neighborhood. In *CIKM*. ACM, 1979–1982.

[5] James Bergstra, Daniel Yamins, and David D. Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013 (JMLR Workshop and Conference Proceedings, Vol. 28)*. JMLR.org, 115–123.

[6] Yixin Cao, Xiang Wang, Xiangnan He, Zikun Hu, and Tat-Seng Chua. 2019. Unifying Knowledge Graph Learning and Recommendation: Towards a Better Understanding of User Preferences. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. 151–161.

[7] Marcel Caraciolo, Bruno Melo, and Ricardo Caspirro. 2011. Crab: A recommendation engine framework for python. *Jarrodmillman Com* (2011).

[8] Pablo Castells, Neil J. Hurley, and Saul Vargas. 2015. Novelty and Diversity in Recommender Systems. In *Recommender Systems Handbook*. 881–918.

[9] Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. 2020. Efficient Neural Matrix Factorization without Sampling for Recommendation. *ACM Trans. Inf. Syst.* 38, 2 (2020), 14:1–14:28.

[10] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *KDD*. ACM, 785–794.

[11] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS@RecSys 2016, Boston, MA, USA, September 15, 2016*. 7–10.

[12] Arthur F. Da Costa, Eduardo P. Fressato, Fernando S. Aguiar Neto, Marcelo G. Manzato, and Ricardo J. G. B. Campello. 2018. Case recommender: a flexible and extensible python framework for recommender systems. In *Proceedings of the 12th ACM Conference on Recommender Systems, RecSys 2018, Vancouver, BC, Canada, October 2-7, 2018*. 494–495.

[13] Mukund Deshpande and George Karypis. 2004. Item-based top-*N* recommendation algorithms. *ACM Trans. Inf. Syst.* 22, 1 (2004), 143–177.

[14] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. MyMediaLite: A Free Recommender System Library. In *5th ACM International Conference on Recommender Systems (RecSys 2011)* (Chicago, USA).

[15] Guibing Guo, Jie Zhang, Zhu Sun, and Neil Yorke-Smith. 2015. LibRec: A Java Library for Recommender Systems. In *Posters, Demos, Late-breaking Results and Workshop Proceedings of the 23rd Conference on User Modeling, Adaptation, and Personalization (UMAP 2015), Dublin, Ireland, June 29 - July 3, 2015*.

[16] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. 1725–1731.

[17] Ruining He, Wang-Cheng Kang, and Julian J. McAuley. 2017. Translation-based Recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017, Como, Italy, August 27-31, 2017*. 161–169.

[18] Ruining He and Julian J. McAuley. 2016. Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation. In *ICDM*. IEEE Computer Society, 191–200.

[19] Xiangnan He and Tat-Seng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*. 355–364.

[20] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*. 639–648.

[21] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. 2018. Outer Product-based Neural Collaborative Filtering. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. 2227–2233.

[22] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. 2018. NAIS: Neural Attentive Item Similarity Model for Recommendation. *IEEE Trans. Knowl. Data Eng.* 30, 12 (2018), 2354–2366.

[23] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*. 173–182.

[24] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. 2016. Parallel Recurrent Neural Network Architectures for Feature-rich Session-based Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*. 241–248.

[25] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y. Chang. 2018. Improving Sequential Recommendation with Knowledge-Enhanced Memory Networks. In *SIGIR*. ACM, 505–514.

[26] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*. 2333–2338.

[27] Nicolas Hug. 2020. Surprise: A Python library for recommender systems. *Journal of Open Source Software* 5, 52 (2020), 2174.

[28] Yu-Chin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware Factorization Machines for CTR Prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*. 43–50.

[29] Santosh Kabbur, Xia Ning, and George Karypis. 2013. FISM: factored item similarity models for top-N recommender systems. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*. 659–667.

[30] Wang-Cheng Kang and Julian J. McAuley. 2018. Self-Attentive Sequential Recommendation. In *ICDM*. IEEE Computer Society, 197–206.

[31] 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. In *NIPS*. 3146–3154.

[32] Walid Krichene and Steffen Rendle. 2020. On Sampled Metrics for Item Recommendation. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. 1748–1757.

[33] Kula and Maciej. 2017. Spotlight. https://github.com/maciejkula/spotlight.

[34] Maciej Kula. 2015. Metadata Embeddings for User and Item Cold-start Recommendations. In *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015. (CEUR Workshop Proceedings, Vol. 1448)*. CEUR-WS.org, 14–21.

[35] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural Attentive Session-based Recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*. 1419–1428.

[36] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. 1754–1763.

[37] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. 2018. Variational Autoencoders for Collaborative Filtering. In *WWW*. ACM, 689–698.

[38] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. STAMP: Short-Term Attention/Memory Priority Model for Session-based Recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. 1831–1839.

[39] Sam Lobel, Chunyuan Li, Jianfeng Gao, and Lawrence Carin. 2020. RaCT: Toward Amortized Ranking-Critical Training For Collaborative Filtering. In *ICLR*. OpenReview.net.

[40] Chen Ma, Peng Kang, and Xue Liu. 2019. Hierarchical Gating Networks for Sequential Recommendation. In *KDD*. ACM, 825–833.

[41] Jianxin Ma, Chang Zhou, Peng Cui, Hongxia Yang, and Wenwu Zhu. 2019. Learning Disentangled Representations for Recommendation. In *NeurIPS*. 5712–5723.

[42] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, Iadh Ounis, Siwei Liu, Yaxiong Wu, Xi Wang, Shangsong Liang, Yucheng Liang, Guangtao Zeng, Junhua Liang, and Qiang Zhang. 2020. BETA-Rec: Build, Evaluate and Tune Automated Recommender Systems. In *RecSys 2020: Fourteenth ACM Conference on Recommender Systems, Virtual Event, Brazil, September 22-26, 2020*. 588–590.

[43] ThaiBinh Nguyen and Atsuhiro Takasu. 2018. NPE: Neural Personalized Embedding for Collaborative Filtering. In *IJCAI*. ijcai.org, 1583–1589.

[44] Xia Ning and George Karypis. 2011. SLIM: Sparse Linear Methods for Top-N Recommender Systems. In *ICDM*. IEEE Computer Society, 497–506.

[45] Junwei Pan, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun, and Quan Lu. 2018. Field-weighted Factorization Machines for Click-Through Rate Prediction in Display Advertising. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*. 1349–1357.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems.* 8026–8037.

[47] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-Based Neural Networks for User Response Prediction. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain.* 1149–1154.

[48] Pengjie Ren, Zhumin Chen, Jing Li, Zhaochun Ren, Jun Ma, and Maarten de Rijke. 2019. RepeatNet: A Repeat Aware Neural Recommendation Machine for Session-Based Recommendation. In *AAAI.* AAAI Press, 4806–4813.

[49] Steffen Rendle. 2010. Factorization Machines. In *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010.* 995–1000.

[50] Steffen Rendle. 2012. Factorization Machines with libFM. *ACM Trans. Intell. Syst. Technol.* 3, 3, Article 57 (May 2012), 22 pages.

[51] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009.* 452–461.

[52] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized Markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010.* 811–820.

[53] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007.* 521–530.

[54] Aghiles Salah, Quoc-Tuan Truong, and Hady W. Lauw. 2020. Cornac: A Comparative Framework for Multimodal Recommender Systems. *Journal of Machine Learning Research* 21, 95 (2020), 1–5.

[55] Ilya Shenbin, Anton Alekseev, Elena Tutubalina, Valentin Malykh, and Sergey I. Nikolenko. 2020. RecVAE: A New Variational Autoencoder for Top-N Recommendations with Implicit Feedback. In *WSDM.* ACM, 528–536.

[56] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019.* 1161–1170.

[57] Harald Steck. 2019. Embarrassingly Shallow Autoencoders for Sparse Data. In *WWW.* ACM, 3251–3257.

[58] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019.* 1441–1450.

[59] Zhu Sun, Di Yu, Hui Fang, Jie Yang, Xinghua Qu, Jie Zhang, and Cong Geng. 2020. Are We Evaluating Rigorously? Benchmarking Recommendation for Reproducible Evaluation and Fair Comparison. In *Proceedings of the 14th ACM Conference on Recommender Systems.*

[60] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved Recurrent Neural Networks for Session-based Recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS@RecSys 2016, Boston, MA, USA, September 15, 2016.* 17–22.

[61] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW.* ACM, 1067–1077.

[62] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018.* 565–573.

[63] Rianne van den Berg, Thomas N. Kipf, and Max Welling. 2017. Graph Convolutional Matrix Completion. *CoRR* abs/1706.02263 (2017). arXiv:1706.02263

[64] Chenyang Wang, Min Zhang, Weizhi Ma, Yiqun Liu, and Shaoping Ma. 2020. Make It a Chorus: Knowledge- and Time-aware Item Modeling for Sequential Recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020.* 109–118.

[65] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2018. RippleNet: Propagating User Preferences on the Knowledge Graph for Recommender Systems. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018.* 417–426.

[66] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. 2019. Knowledge-aware Graph Neural Networks with Label Smoothness Regularization for Recommender Systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019.* 968–977.

[67] Hongwei Wang, Fuzheng Zhang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2019. Multi-Task Feature Learning for Knowledge Graph Enhanced Recommendation. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019.* 2000–2010.

[68] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. 2019. Knowledge Graph Convolutional Networks for Recommender Systems. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019.* 3307–3313.

[69] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2015. Learning Hierarchical Representation Model for NextBasket Recommendation. In *SIGIR.* ACM, 403–412.

[70] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17, Halifax, NS, Canada, August 13 - 17, 2017.* 12:1–12:7.

[71] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge Graph Attention Network for Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019.* 950–958.

[72] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019.* 165–174.

[73] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled Graph Collaborative Filtering. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020.* 1001–1010.

[74] Bin Wu, Zhongchuan Sun, Xiangnan He, Xiang Wang, and Jonathan Staniforth. 2017. NeuRec. https://github.com/wubinzzu/NeuRec.

[75] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-Based Recommendation with Graph Neural Networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.* 346–353.

[76] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative Denoising Auto-Encoders for Top-N Recommender Systems. In *WSDM.* ACM, 153–162.

[77] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. 2017. Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017.* 3119–3125.

[78] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S. Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. 2019. Graph Contextualized Self-Attention Network for Session-based Recommendation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019.* 3940–3946.

[79] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep Matrix Factorization Models for Recommender Systems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017.* 3203–3209.

[80] Haochao Ying, Fuzhen Zhuang, Fuzheng Zhang, Yanchi Liu, Guandong Xu, Xing Xie, Hui Xiong, and Jian Wu. 2018. Sequential Recommender System based on Hierarchical Attention Networks. In *IJCAI.* ijcai.org, 3926–3932.

[81] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose, and Xiangnan He. 2019. A Simple Convolutional Generative Network for Next Item Recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019.* 582–590.

[82] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016.* 353–362.

[83] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Comput. Surv.* 52, 1, Article 5 (Feb. 2019), 38 pages. https://doi.org/10.1145/3285029

[84] Tingting Zhang, Pengpeng Zhao, Yanchi Liu, Victor S. Sheng, Jiajie Xu, Deqing Wang, Guanfeng Liu, and Xiaofang Zhou. 2019. Feature-level Deeper Self-Attention Network for Sequential Recommendation. In *IJCAI.* ijcai.org, 4320–4326.

[85] Tingting Zhang, Pengpeng Zhao, Yanchi Liu, Victor S. Sheng, Jiajie Xu, Deqing Wang, Guanfeng Liu, and Xiaofang Zhou. 2019. Feature-level Deeper Self-Attention Network for Sequential Recommendation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019.* 4320–4326.

[86] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep Learning over Multi-field Categorical Data - - A Case Study on User Response Prediction. In *Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings.* 45–57.

[87] Wayne Xin Zhao, Junhua Chen, Pengfei Wang, Qi Gu, and Ji-Rong Wen. 2020. Revisiting Alternative Experimental Settings for Evaluating Top-N Item Recommendation Algorithms. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*. ACM, 2329–2332.

[88] Wayne Xin Zhao, Jin Huang, and Ji-Rong Wen. 2016. Learning distributed representations for recommender systems with a network embedding approach. In *Asia information retrieval symposium*. Springer, 224–236.

[89] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S. Yu. 2018. Spectral collaborative filtering. In *Proceedings of the 12th ACM Conference on Recommender Systems, RecSys 2018, Vancouver, BC, Canada, October 2-7, 2018*. 311–319.

[90] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep Interest Evolution Network for Click-Through Rate Prediction. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. 5941–5948.

[91] Guorui Zhou, Xiaoqiang Zhu, Chengru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep Interest Network for Click-Through Rate Prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. 1059–1068.

[92] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. 2020. S3-Rec: Self-Supervised Learning for Sequential Recommendation with Mutual Information Maximization. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*. ACM, 1893–1902.