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

Wayne Xin Zhao1,2, Shanlei Mu1,3,#, Yupeng Hou1,2,#, Zihan Lin1,3, Yushuo Chen1,2, Xingyu Pan1,3, Kaiyuan Li4, Yujie Lu5, Hui Wang1,3, Changxin Tian1,3, Yingqian Min1,3, Zhichao Feng4, Xinyan Fan1,2, Xu Chen1,2,#, Pengfei Wang7,4,#, Wendi Ji5, Yaliang Li6, Xiaoling Wang5, Ji-Rong Wen1,2,3

1Beijing Key Laboratory of Big Data Management and Analysis Methods
2{Gaoling School of Artificial Intelligence, 3School of Information} Renmin University of China
4Beijing University of Posts and Telecommunications, 5East China Normal University, 6Alibaba, 7Liaoning 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 [rɛkˈboʊlɛ]), 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/.

ACM Reference Format:
Wayne Xin Zhao1,2, Shanlei Mu1,3,#, Yupeng Hou1,2,#, Zihan Lin1,3, Yushuo Chen1,2, Xingyu Pan1,3, Kaiyuan Li4, Yujie Lu5, Hui Wang1,3, Changxin Tian1,3, Yingqian Min1,3, Zhichao Feng4, Xinyan Fan1,2, Xu Chen1,2,#, Pengfei Wang7,4,#, Wendi Ji5, Yaliang Li6, Xiaoling Wang5, Ji-Rong Wen1,2,3. 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 [rɛkˈboʊlɛ]) . Based on this library, 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.

# Both authors contributed equally to this work.

* Xu Chen {successesx@gmail.com} and Pengfei Wang {wangpengfei@bupt.edu.cn} are corresponding authors.
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 which greatly reduces the efforts for developing new models.

- 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 — RECOBOLE

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 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 $\rightarrow$ atomic files $\rightarrow$ Dataset $\rightarrow$ Dataloader $\rightarrow$ Interaction $\rightarrow$ 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.
We introduce and define six atomic file types. 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.

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.

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.

#### 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       | 480,198| 17,770 | 100,480,507   |
| Book-Crossing | 105,284| 340,557| 1,149,780     |
| Jester        | 73,421 | 1,280,969 | 6,442,892   |
| Douban        | 738,701| 1,445,622| 11,557,943   |
| 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          | -      | -      | -             |

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.

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.

#### 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        |
we can see, given a dataset, the user can reuse the same.

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           |

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 (head, tail, relation) 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.

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 calculateLoss() for training and the interface function of predict() 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
Table 6: Implemented 73 recommender models in RecBole on 4 categories.

| Category                          | Model     | Conference | Year | Typical Evaluation Dataset                      |
|-----------------------------------|-----------|------------|------|-------------------------------------------------|
| General Recommendation            | popularity | TOIS       | 2004 | -                                               |
|                                   |ItemKNN    | -          | -    | -                                               |
|                                   |EPR [51]   | UAI        | 2009 | Rossmann, Netflix                               |
|                                   |SLIMElastic [44] | ICDM | 2011 | crlq, cedam, RedisCrossing, MovieLens, Netflix, Yahoo Music |
|                                   |FISM [29]  | SIGKDD     | 2013 | MovieLens, Netflix, Yahoo Music                |
|                                   |LINE [41, 88] | WWW | 2015 | NetWork (Wikipedia, Flickr, Youtube, DBLP)       |
|                                   |CDAE [76]  | WSDM       | 2016 | MovieLens, Netflix, Yelp                       |
|                                   |NeuMF [23] | WWW        | 2017 | MovieLens, Pintrest                           |
|                                   |ConvNCF [21] | IJCAI | 2017 | Yelp, Gowalla                                  |
|                                   |DMF [79]   | IJCAI      | 2017 | MovieLens, Amazon                              |
|                                   |NUCF [4]   | CIKM       | 2017 | Delicious, MovieLens, Rossmann                 |
|                                   |NAIS [22]  | TKDE       | 2018 | MovieLens, Pintrest                           |
|                                   |SpectralKCF [89] | RecSys | 2018 | MovieLens, HetRec, Amazon                      |
|                                   |MultiVAE [37] | WWW | 2018 | MovieLens, Million Song, Netflix               |
|                                   |MultidAE [37] | WWW | 2018 | MovieLens, Million Song, Netflix               |
|                                   |GCMS [63]  | SIGKDD     | 2018 | MovieLens, Flxster,Douhan, Yahoo Music         |
|                                   |NGCF [72]  | SIGIR      | 2019 | Gowalla, Yelp, Amazon                          |
|                                   |MaerlVAE [41] | NeurIPS | 2019 | AliShop-7C, MovieLens, Netflix                 |
|                                   |EASE [57]  | WWW        | 2019 | MovieLens, Million Song, Netflix               |
|                                   |LightGCN [20] | SIGIR | 2020 | Gowalla, Yelp, Amazon                          |
|                                   |DGF [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 | Herencounter                                   |
|                                   |FFM [28]   | RecSys     | 2016 | Criteo, Avazu                                  |
|                                   |FNN (DNN) [36] | ECTR | 2016 | Criteo, PinYou                                 |
|                                   |FNN [47]   | ICDM       | 2016 | Google play dataset                            |
|                                   |Wide2Deep [11] | RecSys | 2016 | Allstate, Higgs Boson, Yahoo LTRC, Criteo      |
|                                   |XGBosst [10] | KDD | 2016 | Frappe,MovieLens                               |
|                                   |NFM [19]   | SIGIR      | 2017 | Frappe,MovieLens                               |
|                                   |DeepFM [16] | IJCAI      | 2017 | Criteo, Company                                |
|                                   |AFM [77]   | SIGIR      | 2017 | Frappe, MovieLens                              |
|                                   |DCN [70]   | ADKD       | 2017 | Criteo                                        |
|                                   |LightGBM [31] | NIPS | 2017 | Allstate, Flight-Delay, LETOR, KDD10, KDD12   |
|                                   |xDeepFM [36] | SIGKDD | 2018 | Criteo, Dianping, Bing News                    |
|                                   |PwFm [45]  | WWW        | 2018 | Criteo, Oath                                   |
|                                   |DIN [91]   | SIGKDD     | 2018 | Amazon, MovieLens, Alibahba                    |
|                                   |DIEN [90]  | AAAI       | 2019 | Amazon                                         |
|                                   |AutoInt [56] | CIKM | 2019 | Criteo, Avazu, KDD Cup 2012, MovieLens        |
| Sequential recommendation         | FFMC [52] | WWW        | 2010 | ROSSMANN                                       |
|                                   |HRM [69]   | SIGIR      | 2015 | Ta-Feng, BeiRen, Tmall                        |
|                                   |Improved GRU-Rec [60] | DLRS | 2016 | YOOCHOOSE                                     |
|                                   |GRU4Rec[feature embedding] [24] | RecSys | 2016 | coined VIDDL, 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 [36] | ICDM | 2018 | Amazon, Steam, MovieLens                      |
|                                   |KSS [25]   | SIGIR      | 2018 | LastFM, MovieLens, Amazon                     |
|                                   |SHM [80]   | IJCAI      | 2018 | Tmall, Gowalla                                |
|                                   |NPE [43]   | IJCAI      | 2018 | MovieLens, Online Retail, TasterProfile        |
|                                   |NextItemfile [81] | WSDM | 2019 | YOOCHOOSE, LastFM                             |
|                                   |BERT4Rec [38] | CIKM | 2019 | Amazon, Steam, MovieLens                      |
|                                   |SIRGN [75] | AAAI       | 2019 | YOOCHOOSE, DIGINETICA                         |
|                                   |GCSAN [78] | IJCAI      | 2019 | DIGINETICA, Retailrocket                      |
|                                   |SASRec[feature embedding] [84] | IJCAI | 2019 | -                                              |
|                                   |FFSA [85]  | CIKM       | 2019 | Amazon, Tmall                                 |
|                                   |RepeatNet [48] | AAAI | 2019 | YOOCHOOSE, DIGINETICA, LastFM                  |
|                                   |HGN [40]   | SIGKDD     | 2019 | MovieLens, Amazon, Goodreads                  |
|                                   |SIMC [92]  | CIKM       | 2020 | Mettiana, Amazon, Yelp, LastFM                 |
| Knowledge-based recommendation    | CKE [82]  | SIGKDD     | 2016 | MovieLens, IntentBooks                        |
|                                   |CFPGK [1]  | MDPI       | -    | Amazon                                         |
|                                   |RippleNet [65] | CIKM | 2018 | MovieLens, Book-Crossing, Bing-News            |
|                                   |KTFUP [6]  | WWW        | 2019 | MovieLens, DBBook2014                         |
|                                   |KGAT [71]  | SIGKDD     | 2019 | Amazon, LastFM, Yelp2018                       |
|                                   |MKR [67]   | WWW        | 2019 | MovieLens, Book-Crossing, LastFM, Bing-News    |
|                                   |KGCN [48]  | WWW        | 2019 | MovieLens, Book-Crossing, LastFM, Bing-News    |
|                                   |KGNN-Li [46] | SIGKDD | 2019 | MovieLens, Book-Crossing, LastFM, Dianping-Food |
we can indicate four types of tuning methods, i.e.,

The function of evaluation module is to implement commonly used
ranking-based evaluation metrics. The value-based metrics (for
top-K item recommendation) 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
include 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 de-

rival 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, Or-
der 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 × m matrix D consisting of the confidence scores
from a model over the entire item set. When performing sample-

-based ranking, we create an n × m matrix D, initializing all elements
to negative infinity. Then, we fill matrix 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 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
**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 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).

$^2$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\textsuperscript{acc} denotes the model with acceleration strategy.

| Model  | MovieLens-100k | MovieLens-1M | MovieLens-10M |
|--------|----------------|--------------|---------------|
| BPR    | 0.245s         | 2.478s       | 29.900s       |
| BPR\textsuperscript{acc} | 0.009s         | 0.090s       | 2.210s        |

Figure 4: An illustrative usage flow of our library.

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)`.

(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 “\texttt{\_\_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 “\texttt{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 “\texttt{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.

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.
[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, Zhihun 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, Christophe Freudenthaler, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In UAI 2009, Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 21-23, 2009. 452–461.

[50] Steffen Rendle. 2012. Factorization Machines with libFM. In Proceedings of the 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010. 995–1000.

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

[52] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting clicks: estimating the click-through rate for new ads. In Proceedings of the 10th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007. 521–530.

[53] Ilya Shelbin, 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.

[54] Xiaogang Song, Chencen Shi, Zhujiao Xia, Zhijun Duan, Xuewen 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.

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

[56] Fei Sun, Jian Liu, Jian Wu, Changhui Pei, Xiao Lin, Wenwu Ou, and Feng 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. 1441–1451.

[57] Zihao Sun, Di Yu, Hui Fang, Je Yang, Qihang Qu, Ji Zhang, and Cong Geng. 2020. Are We Evaluating Rigorously? Benchmarking Recommendation for Re-producible Evaluation and Fair Comparison. In Proceedings of the 14th ACM Conference on Recommender Systems.

[58] Yong Xu, Xuan Tan, Ching Xue, and Yong Liu. 2016. Improved Recurrent Neural Networks for Session-based Recommendations. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRecSys 2016, Boston, MA, USA, September 15, 2016. 17–22.

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

[60] Ziyu Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequential 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.

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

[62] 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 ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020. 109–118.

[63] 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. 3307–3313.

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

[65] Tingting Zhang, Pengpeng Zhao, Yanchi Liu, Victor S. Sheng, Jiajie Xu, Fuzhen Zhang, Jinhua 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, Melbourne, Australia, August 19-25, 2019. 3119–3125.

[66] Tingting Zhang, Pengpeng Zhao, Yanchi Liu, Victor S. Sheng, Jiajie Xu, Fuzhen Zhang, Jinhua 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 18-19, 2019. 3940–3946.

[67] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tai. 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. ACM Comput. Surv. 52, 1, Article 3 (Feb. 2019), 38 pages. https://doi.org/10.1145/3285029

[68] Weinan Zhang. 2018. An Empirical Study on User Response Prediction. In Advances in Information Retrieval – 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. 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.