CRSLab: An Open-Source Toolkit for Building Conversational Recommender System

Kun Zhou[1,3] [†], **Xiaolei Wang**[2] [†], **Yuanhang Zhou**[1,3], **Chenzhan Shang**[1], **Yuan Cheng**[4], **Wayne Xin Zhao**[2,3] [*], **Yaliang Li**[5], and **Ji-Rong Wen**[1,2,3]

[1]School of Information, Renmin University of China
[2]Gaoling School of Artificial Intelligence, Renmin University of China
[3]Beijing Key Laboratory of Big Data Management and Analysis Methods
[4]School of Statistics, Renmin University of China
[5]Alibaba Group

Abstract

In recent years, conversational recommender systems (CRSs) have drawn a wide attention in the research community, which focus on providing high-quality recommendations to users via natural language conversations. However, due to diverse scenarios and data formats, existing studies on CRSs lack unified and standardized implementation or comparison. To tackle this challenge, we release an open-source toolkit **CRSLab**, which provides a unified and extensible framework with highly-decoupled modules to develop CRSs. Based on this framework, we collect 6 commonly used human-annotated CRS datasets and implement 19 models that include advanced techniques such as graph neural networks and pre-training models. Besides, our toolkit provides a series of automatic evaluation protocols and a human-machine interaction interface to evaluate and compare different CRS methods. The project and documents are released at `https://github.com/RUCAIBox/CRSLab`.

1 Introduction

Recent years have witnessed remarkable progress in recommender systems, which aim to present items (*e.g.,* products or movies) of potential interests to users based on their preferences (Sarwar et al., 2001; Rendle et al., 2012). Traditional recommender systems mainly leverage the user historical behavior data (*e.g.,* click or purchase) to estimate user preferences implicitly. Recently, with the rapid development of human-machine conversation techniques (Shang et al., 2015; Zhang et al., 2018a; Lee et al., 2019), conversational recommender systems (CRSs) (Christakopoulou et al., 2016; Sun and Zhang, 2018; Gao et al., 2021) is gaining increasing attention in recent years. It relies on multi-turn natural language conversations to clarify explicit user preferences and generate more appropriate recommendations.

To build an effective CRS, researchers have proposed several datasets (Li et al., 2018; Kang et al., 2019; Liu et al., 2020) and models (Liao et al., 2020; Lei et al., 2020; Xu et al., 2020). However, due to their diverse scenarios (*e.g.,* E-commerce or movie recommendation) and data formats (*e.g.,* historical utterances or interacted items), it is challenging for users to quickly set up reasonable baseline systems and compare their performances.

To alleviate the above issues, we have developed **CRSLab**, an open-source CRS toolkit for research purpose. In CRSLab, we offer a unified and extensible framework to develop CRSs. Based on this toolkit, users are able to quickly train and evaluate CRS models via a few lines of code, and easily design new CRS models using the provided interfaces. To implement the overall framework, we design six highly-decoupled modules (*e.g.,* data module and model module), each module provides clear interfaces for specific functions. Besides, we encapsulate useful procedures and common functions shared by different modules for users to add new datasets or develop new models using our toolkit.

Based on the framework, we integrate comprehensive benchmark datasets and models in CRSLab. So far, we have incorporated 6 commonly used human-annotated datasets and implemented 19 models, including advanced techniques such as graph neural networks (GNN) (Schlichtkrull et al., 2018; Chen et al., 2019) and pre-training models (Devlin et al., 2019; Zhou et al., 2020b). To support these models, we perform necessary preprocessing (*e.g.,* entity linking and word segmentation) on included datasets, and release the processed version. Be-

[†] Equal contribution.
[*] Corresponding author, Email: batmanfly@gmail.com.

sides, CRSLab supports both configuration files and command-line instructions, which facilitate running, comparing and testing models on different datasets.

Furthermore, CRSLab provides a series of automatic evaluation protocols for testing and comparing different methods, covering commonly used metrics in existing works. It makes our work useful for the standardization of evaluation protocols for conversational recommendations. In addition, CRSLab provides a human-machine interactive interface to perform quantitative analysis, which is helpful for users to deploy their systems and converse with the systems via graphical interfaces.

Our contributions are as follows:

• To the best of our knowledge, CRSLab is the first open-source CRS toolkit covering 6 human-annotated datasets and 19 models.

• CRSLab provides a unified and extensible framework consisting of highly-decoupled modules, which helps users run and develop different CRS models.

• CRSLab contains commonly used automatic evaluation metrics and a human-machine interactive interface for users to test CRS performance from different perspectives.

2 Background

As aforementioned, the various scenarios and input formats in earlier works lead to inconvenience when applying existing CRS models on different datasets. By surveying previous CRS works (Sun and Zhang, 2018; Li et al., 2018; Lei et al., 2020), we summarize two basic tasks and an auxiliary task, namely recommendation task, conversation task and policy task.

Given the dialog context (*i.e.,* historical utterances) and other useful side information (*e.g.,* user historical behaviors and knowledge graphs), the recommendation sub-task is defined as predicting user-preferred items (*e.g.,* movies or products), and the conversation sub-task is to generate a proper response for conversing with the user. In existing works, the recommendation and the conversation sub-tasks are considered as the primary objective of the CRS. As a complementary sub-task, policy sub-tasks are proposed by recent works (Sun and Zhang, 2018; Lei et al., 2020; Liu et al., 2020) to better control the overall CRS. It usually focuses on selecting appropriate system actions (*e.g.,* recommendation or conversation) or tracking dialog states (topic prediction or goal tracking) to proactively guide the conversation.

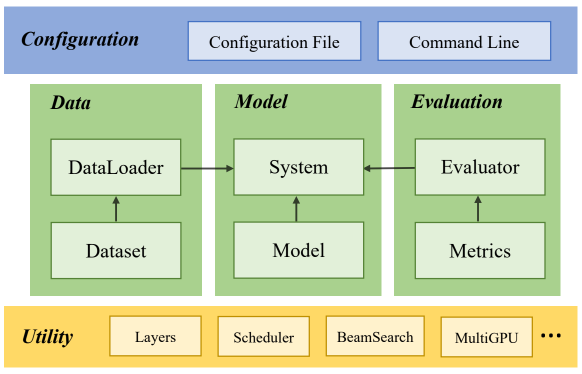


Figure 1: The overall framework of CRSLab.

3 CRSLab

The overall framework of our toolkit CRSLab is presented in Figure 1. The configuration module provides a flexible interface for users to easily set up the experiment environment (*e.g.,* datasets, models and hyperparameters). The data, model and evaluation modules are built upon the configuration module, which forms the core part of our toolkit. The bottom part is the utility module, providing auxiliary functions and interfaces for reuse in other modules (*e.g.,* Layers and Scheduler). In the following part, we briefly present the design of the above modules. More details can be found in the toolkit documents.

3.1 Configuration Module

In CRSLab, we design the configuration module for users to conveniently select or modify the experiment settings (*e.g.,* training data and hyperparameters). Specifically, we design the class Config to store all the configuration settings, which specify the data, model, hyperparameters and other necessary settings of a given experiment. To avoid complicated command line parameters, we transfer most of them into YAML configuration files, while other commonly used ones (*i.e.,* file path and debug mode) are provided as command line instructions. In this way, users can build and evaluate a variety of CRSs with only a few modifications in the configuration files.

3.2 Data Module

For extensibility and reusability, we design an elegant data flow that transforms raw dataset

Dataset	Dialog	Utterance	Domain	Policy Task	Entity KG	Word KG
ReDial (Li et al., 2018)	10,006	182,150	Movie	–	DB	CNet
OpenDialKG (Moon et al., 2019)	13,802	91,209	Movie, Book	Path Generation	DB	CNet
GoRecDial (Kang et al., 2019)	9,125	170,904	Movie	Action Prediction	DB	CNet
DuRecDial (Liu et al., 2020)	10,200	156,000	Movie, Music	Goal Planning	CN-DB	HNet
INSPIRED (Hayati et al., 2020)	1,001	35,811	Movie	Strategy Prediction	DB	CNet
TG-ReDial (Zhou et al., 2020c)	10,000	129,392	Movie	Topic Prediction	CN-DB	HNet

Table 1: The collected datasets in CRSLab. DB and CN-DB stand for the entity-oriented knowledge graph DBpedia and CN-DBpedia, respectively. CNet and HNet stand for the word-oriented knowledge graph ConceptNet and HowNet, respectively.

into the model input as follows: raw public dataset $\longrightarrow$ preprocessed dataset $\longrightarrow$ Dataset $\longrightarrow$ DataLoader $\longrightarrow$ System. Next, we detail the design of each component.

Data Preprocessing Since raw datasets vary in formats and features, we preprocess them to support unified interfaces in data modules. Based on the task description in Section 2, we first preprocess CRS datasets to match the input and output formats. Exactly, we organize the dialog contexts and side information as the input, and extract the recommended items, dialog actions and responses as the output of recommendation, policy and conversation tasks, respectively. To support some advanced models (*e.g.,* graph neural networks and pre-training models), we incorporate useful side data (*e.g.,* knowledge graph) and conduct specific data preprocessing (*e.g.,* entity linking and BPE segmentation).

As shown in Table 1, we collect 6 commonly used human-annotated datasets and release the preprocessed version with side data in CRSLab. Besides, we also release the pre-trained word embeddings and other associated files, which ease the use of integrated datasets and reduce the time cost.

Dataset Class To decouple the implementation of data preparation in CRSLab, we design the class Dataset for integrating the model-agnostic data processing functions, while the rest functions are implemented within the class DataLoader. In this way, Dataset only focuses on processing the input data into a unified format (*i.e.,* a list of python.dict), without considering specific models. In CRSLab, we design the class BaseDataset which includes some common attributes (*e.g.,* configurations and data paths) and basic functions (*e.g.,* data loading) of Dataset. Users can inherit BaseDataset with very few modifications to introduce new datasets.

DataLoader Class To support different input formats, DataLoader further transforms data from the Dataset module to support various models. It focuses on selecting input data from the processed data to form tensor data (*i.e.,* torch.Tensor) in a batch or mini-batch, which can be directly used for training or testing. To implement it, we design the class BaseDataLoader to integrate commonly used attributes and functions, and inherit it to produce new DataLoader for corresponding models.

3.3 Model Module

Based on the task descriptions and the above data modules, we reorganize the implementations of different CRSs in the model module. Existing works either integrate specific models to accomplish the recommendation, conversation and policy task, respectively (Chen et al., 2019; Zhou et al., 2020a), or only focus on one of the above tasks (Kang and McAuley, 2018; Hayati et al., 2020). Therefore, we divide commonly used models into four categories, namely CRS models (containing several sub-models to complete the corresponding tasks), recommendation models, conversation models and policy models. Besides, we also consider some classic heuristic methods (*e.g.,* Popularity and PMI) and several popular models which can be utilized to solve one of the above tasks, such as TextCNN (Kim, 2014) and BERT (Devlin et al., 2019). As illustrated in Table 2, we have implemented 19 models in the first released version, including some advanced models such as graph neural networks and pre-training models.

For implementation of these models, we develop the model class to provide functions and interfaces of specific models for corresponding tasks. In the model class, we focus on providing a basic structure and highly-decoupled useful functions or procedures for further development. In detail, we unify the basic attributes and func-

Category	Model	GNN	PTM	Reference
CRS model	ReDial	×	×	(Li et al., 2018)
	KBRD	√	×	(Chen et al., 2019)
	KGSF	√	×	(Zhou et al., 2020a)
	TG-ReDial	×	√	(Zhou et al., 2020c)
Recommendation model	Popularity	×	×	–
	GRU4Rec	×	×	(Hidasi et al., 2016)
	SASRec	×	×	(Kang and McAuley, 2018)
	TextCNN	×	×	(Kim, 2014)
	R-GCN	√	×	(Schlichtkrull et al., 2018)
	BERT	×	√	(Devlin et al., 2019)
Conversation model	HERD	×	×	(Serban et al., 2016)
	Transformer	×	×	(Vaswani et al., 2017)
	GPT-2	×	√	(Radford et al., 2019)
	INSPIRED	×	√	(Hayati et al., 2020)
Policy model	PMI	×	×	–
	MGCG	×	×	(Liu et al., 2020)
	Conv-BERT	×	√	(Zhou et al., 2020c)
	Topic-BERT	×	√	(Zhou et al., 2020c)
	Profile-BERT	×	√	(Zhou et al., 2020c)

Table 2: The implemented models in CRSLab. The CRS models integrate several sub-models to complete the overall conversational recommendation process, while recommendation, policy and conversation models only focus on one individual task. GNN and PTM stand for the graph neural networks and pre-training models, respectively.

tions of various models (*e.g.,* parameter initialization and model loading) into the class BaseModel. A user can inherit BaseModel and implement a few functions to design and develop new models. In this way, we re-implement the above models in our toolkit and unify the implementation of commonly used layers and components into the Utility Module for future usage. For all the implemented models, we have tested their performance on several datasets, and invited a code reviewer to examine the correctness of implementation. In the future, more methods will be incorporated along with regular updates.

3.4 System Module

In the system module, we build, train and evaluate contained models for accomplishing the overall conversational recommendation task. We aim to integrate the dataloader, model and evaluator modules into a complete system. To support flexible architectures for CRSs at the system level, we devise the system class with several functions for various usage. In detail, we design the class BaseSystem to unify the structures and interfaces, where we develop basic attributes and functions in the process. Among them, we develop the __init__() function to set up the required dataloader, contained models and evaluation protocols, which can be implemented by users for building new systems. Besides, we also implement a series of useful functions, such as optimizer initialization, learning ratio adjustment and early stop strategy. These functions and tiny tricks ease the developing process of a new system and greatly improve the user experience with our toolkit.

Based on the above settings, we design the fit() and step() functions in BaseSystem. The fit() function is used to train the whole system and then conduct evaluation, in which users need to devise the overall training process of all the models in the system, including data distribution, training orders and so on. In the step() function, users implement the detailed learning process for specific models, and functions within the corresponding models can be utilized to optimize model parameters.

3.5 Evaluation Module

The evaluation module implements the evaluation protocols for CRSs. In CRSLab, we implement some commonly used automatic evaluation metrics, and design a human-machine interactive interface for users to perform an end-to-end qualitative analysis.

Automatic Evaluation Since the CRS task is divided into three sub-tasks, we develop automatic evaluation metrics for each one. All the supported metrics are summarized in Table 3.

For the recommendation task, following exist-

Category	Metrics
Recommendation Metrics	Hit@{1,10,50}, MRR@{1,10,50}, NDCG@{1,10,50}
Conversation Metrics	Perplexity, BLEU-{1,2,3,4}, Embedding Average/Extreme/Greedy, Distinct-{1,2,3,4}
Policy Metrics	Accuracy, Hit@{1,3,5}

Table 3: The implemented automatic evaluation metrics in CRSLab.

ing CRSs (Sun and Zhang, 2018; Zhang et al., 2018b), we develop ranking-based metrics (*i.e.,* Hit, MRR and NDCG) to measure the ranking performance of the generated recommendation lists. For the conversation task, CRSLab supports both relevance-based (*i.e.,* BLEU (Papineni et al., 2002) and embedding-based metrics (Liu et al., 2016)) and diversity-based evaluation metrics (*i.e.,* Distinct-{1,2,3,4} (Li et al., 2016)). For the policy task, we implement commonly used metrics (*i.e.,* Accuracy and Hit@K) for evaluation.

Similarly, we design the class BaseEvaluator with common attributes and functions. Then, we inherit this class to implement RecEvaluator, ConvEvaluator and PolicyEvaluator for evaluating recommendation, conversation and policy tasks, respectively. Note that we implement the report() function in these classes. With this, users can print and monitor the performance of models evaluating on validation or test set.

Human-Machine Interaction Interface To evaluate CRSs qualitatively, CRSLab offers a human-machine interaction interface to help perform an end-to-end evaluation. The human-machine interaction interface is implemented within the system module, where the interaction strategy can be easily adapted to a specific policy model. In this way, a user can converse with a CRS or diagnose the system, which provides a direct approach to evaluating the overall performance of a CRS. Besides, the interaction interface enables users to correct errors by modifying intermediate results.

For end-to-end evaluation, users first need to set up the background of a simulated user (*e.g.,* interaction history and user profile), then freely chat with the CRS through the interface. During a conversation, the dialog history and the output of each component (*e.g.,* the recommended items and generated responses) are stored within a dictionary (*i.e.,* python.dict), helping users get a better understanding of how the system works.

3.6 Utility Module

To facilitate the usage of our toolkit, we design the utility module to include a series of useful functions (*e.g.,* layers() and scheduler()). We collect commonly used functions in various models (*e.g.,* CNN, RNN and Transformer layers) to constitute Layers, which can be easily used to develop new CRSs. Besides, we also decouple commonly used functions or procedures in other modules to form the utility file (*i.e.,* utils.py) for reuse.

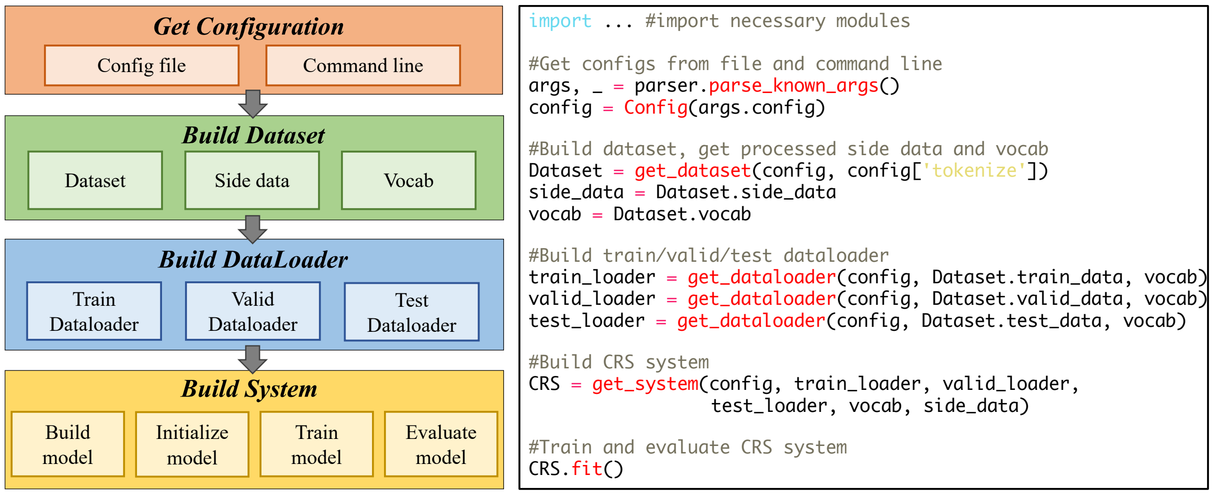
Another particularly useful function is scheduler(), which provides a set of strategies for training large-scale models, such as warming-up strategy and weight decay. In addition, we also implement other functions to enhance user experiences, such as BeamSearch() to improve the inference performance, MultiGPU() for parallel training, logger() to print and monitor the running process, save_model() and load_model() to store and reuse the pre-trained models.

4 System Demonstration

In this section, we show how to use our CRSLab with code examples. We detail the usage descriptions in two parts, namely running an existing CRSs in our toolkit and implementing a new CRS based on the interfaces provided in our toolkit.

4.1 Running an Existing CRS

Our CRSLab enables quickly building a CRS with a few lines of code. Figure 2 presents a general procedure for running an existing CRS in our toolkit. To begin with, the whole procedure relies on the configurations to prepare the dataset and build the system. In the configurations, the user selects a dataset to use and specifies the tokenizer. Then, the Dataset class will automatically download the dataset and perform necessary processesing steps (*e.g.,* tokenization and converting tokens to IDs) based on the configurations. This procedure is executed by the function get_dataset(). Based on the processed datasets, users can use the function get_dataloader() to generate training, validation and test sets, in which the configurations specify the batch size and other necessary parameters for data processing. After that, the function get_system() can be adopted to leverage the prepared data for building a CRS. Similarly, the configurations specify the hyperparameters of



```
import ... #import necessary modules

#Get configs from file and command line
args, _ = parser.parse_known_args()
config = Config(args.config)

#Build dataset, get processed side data and vocab
Dataset = get_dataset(config, config['tokenize'])
side_data = Dataset.side_data
vocab = Dataset.vocab

#Build train/valid/test dataloader
train_loader = get_dataloader(config, Dataset.train_data, vocab)
valid_loader = get_dataloader(config, Dataset.valid_data, vocab)
test_loader = get_dataloader(config, Dataset.test_data, vocab)

#Build CRS system
CRS = get_system(config, train_loader, valid_loader,
                 test_loader, vocab, side_data)

#Train and evaluate CRS system
CRS.fit()
```

Figure 2: An illustrative usage flow of our CRSLab.

models and set up the training and evaluation procedures. Finally, users can start the running process by the function System.fit().

4.2 Developing a New CRS

Based on our toolkit, it is convenient to implement a new CRS with the provided interfaces. Users only need to inherit a few basic classes and implement some interface functions. In this part, we will introduce the detailed implementation process of adding a new dataset and model, respectively.

4.2.1 New Dataset

To add a new dataset, users need to inherit BaseDataset to design a new Dataset class for preparing the dataset into a unified format. In Dataset, the following functions are required to be implemented: __init__(), _load_data() and _data_preprocess().

In __init__(), users set up parameters and links for downloading data. In _load_data(), the training, validation, test data and other side data are loaded from corresponding files. If users follow our naming protocol, all they need is to reuse the functions from the Dataset class. The function _data_preprocess() is to prepare the loaded data, and we have integrated useful functions in the utility module to ease the implementation.

4.2.2 New Model

To add a new model, users should inherit BaseModel to design a new Model class, in which they need to implement the build_model() and forward() functions. In build_model(), users build the model, initialize the parameters and set up the loss function, while in forward() users use the model to predict the result or calculate the loss given the input data. Indeed, users can leverage the encapsulated layers and functions from Layers or the utility files to implement these two functions.

4.3 Performance Evaluation

To evaluate CRSLab, we train and test various implemented models on the TG-ReDial dataset (Zhou et al., 2020c), and compare their performance on recommendation, conversation and policy tasks. In our experiments, we have tuned the hyperparameters of these models to achieve their best performance on this dataset. Due to the space limit, we present the results in our GitHub page [1]. As we can see, our toolkit provides a possibility to compare the performance of various CRS models under different evaluation protocols. Among them, GNN-based models and pretraining methods achieve consistent and remarkable performance on the above tasks. These results are compatible with our expectations.

5 Conclusion

In this paper, we released a toolkit called **CRSLab**, which is the first open-source conversational recommender systems (CRSs) toolkit for research purpose. In CRSLab, we offered a unified and extensible framework with highly-decoupled modules to develop CRSs. Based on this framework, we have incorporated 6 datasets and implemented 19 models in our toolkit. Besides, we also provided extensive automatic evaluation pro-

[1]https://github.com/RUCAIBox/CRSLab

tocols and a human-machine interactive interface in CRSLab, to help evaluate and compare different CRSs. For demonstration, we illustrated how to run or implement a CRS using our toolkit.

With this toolkit, we expect to help users quickly run existing CRSs, ease the development of new CRSs, and set up a benchmark framework for the research of CRSs. In the future, we will make continuous efforts to add more datasets and models. We will also consider adding more utilities for improving the usage of our toolkit, such as result visualization and algorithm debugging.

Acknowledgement

This work was partially supported by the National Natural Science Foundation of China under Grant No. 61872369 and 61832017, Beijing Academy of Artificial Intelligence (BAAI) under Grant No. BAAI2020ZJ0301, Beijing Outstanding Young Scientist Program under Grant No. BJJWZYJH012019100020098, the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China under Grant No.18XNLG22 and 19XNQ047. Xin Zhao is the corresponding author.

References

Qibin Chen, Junyang Lin, Yichang Zhang, Ming Ding, Yukuo Cen, Hongxia Yang, and Jie Tang. 2019. Towards knowledge-based recommender dialog system. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 1803–1813.

Konstantina Christakopoulou, Filip Radlinski, and Katja Hofmann. 2016. Towards conversational 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*, pages 815–824.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Chongming Gao, Wenqiang Lei, Xiangnan He, Maarten de Rijke, and Tat-Seng Chua. 2021. Advances and challenges in conversational recommender systems: A survey. *CoRR*, abs/2101.09459.

Shirley Anugrah Hayati, Dongyeop Kang, Qingxiaoyang Zhu, Weiyan Shi, and Zhou Yu. 2020. INSPIRED: toward sociable recommendation dialog systems. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 8142–8152.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Dongyeop Kang, Anusha Balakrishnan, Pararth Shah, Paul Crook, Y-Lan Boureau, and Jason Weston. 2019. Recommendation as a communication game: Self-supervised bot-play for goal-oriented dialogue. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 1951–1961.

Wang-Cheng Kang and Julian J. McAuley. 2018. Self-attentive sequential recommendation. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*, pages 197–206.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751.

Sungjin Lee, Qi Zhu, Ryuichi Takanobu, Zheng Zhang, Yaoqin Zhang, Xiang Li, Jinchao Li, Baolin Peng, Xiujun Li, Minlie Huang, and Jianfeng Gao. 2019. Convlab: Multi-domain end-to-end dialog system platform. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28 - August 2, 2019, Volume 3: System Demonstrations*, pages 64–69.

Wenqiang Lei, Xiangnan He, Yisong Miao, Qingyun Wu, Richang Hong, Min-Yen Kan, and Tat-Seng Chua. 2020. Estimation-action-reflection: Towards deep interaction between conversational and recommender systems. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, pages 304–312.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A diversity-promoting objective function for neural conversation models. In

NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016, pages 110–119.

Raymond Li, Samira Ebrahimi Kahou, Hannes Schulz, Vincent Michalski, Laurent Charlin, and Chris Pal. 2018. Towards deep conversational recommendations. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 9748–9758.

Lizi Liao, Ryuichi Takanobu, Yunshan Ma, Xun Yang, Minlie Huang, and Tat-Seng Chua. 2020. Topic-guided relational conversational recommender in multi-domain. *IEEE Transactions on Knowledge and Data Engineering*.

Chia-Wei Liu, Ryan Lowe, Iulian Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. 2016. How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2122–2132.

Zeming Liu, Haifeng Wang, Zheng-Yu Niu, Hua Wu, Wanxiang Che, and Ting Liu. 2020. Towards conversational recommendation over multi-type dialogs. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 1036–1049.

Seungwhan Moon, Pararth Shah, Anuj Kumar, and Rajen Subba. 2019. Opendialkg: Explainable conversational reasoning with attention-based walks over knowledge graphs. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 845–854.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*, pages 311–318.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295.

Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, pages 593–607.

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 3776–3784.

Lifeng Shang, Zhengdong Lu, and Hang Li. 2015. Neural responding machine for short-text conversation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1577–1586.

Yueming Sun and Yi Zhang. 2018. Conversational recommender system. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, pages 235–244.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008.

Hu Xu, Seungwhan Moon, Honglei Liu, Bing Liu, Pararth Shah, and Philip S. Yu. 2020. User memory reasoning for conversational recommendation. In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 5288–5308.

Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. 2018a. Personalizing dialogue agents: I have a dog, do you have pets too? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 2204–2213.

Yongfeng Zhang, Xu Chen, Qingyao Ai, Liu Yang, and W. Bruce Croft. 2018b. Towards conversational search and recommendation: System ask, user respond. In *Proceedings of the 27th ACM International Conference on Information and Knowledge*

Management, CIKM 2018, Torino, Italy, October 22-26, 2018, pages 177–186.

Kun Zhou, Wayne Xin Zhao, Shuqing Bian, Yuanhang Zhou, Ji-Rong Wen, and Jingsong Yu. 2020a. Improving conversational recommender systems via knowledge graph based semantic fusion. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 1006–1014.

Kun Zhou, Wayne Xin Zhao, Hui Wang, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. 2020b. Leveraging historical interaction data for improving conversational recommender system. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 2349–2352.

Kun Zhou, Yuanhang Zhou, Wayne Xin Zhao, Xiaoke Wang, and Ji-Rong Wen. 2020c. Towards topic-guided conversational recommender system. In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 4128–4139.