TextBox: A Unified, Modularized, and Extensible Framework for Text Generation

Junyi Li[1,4]†, **Tianyi Tang**[2]†, **Gaole He**[2], **Jinhao Jiang**[3],
Xiaoxuan Hu[2], **Puzhao Xie**[2], **Wayne Xin Zhao**[1,4]*, **Ji-Rong Wen**[1,2,4]
[1]Gaoling School of Artificial Intelligence, Renmin University of China
[2]School of Information, Renmin University of China
[3]University of Electronic Science and Technology of China
[4]Beijing Key Laboratory of Big Data Management and Analysis Methods
{lijunyi,steven_tang}@ruc.edu.cn batmanfly@gmail.com

arXiv:2101.02046v2 [cs.AI] 7 Jan 2021

Abstract

We release an open library, called `TextBox`, which provides a unified, modularized, and extensible text generation framework. TextBox aims to support a broad set of text generation tasks and models. In TextBox, we implements several text generation models on benchmark datasets, covering the categories of VAE, GAN, pre-trained language models, etc. Meanwhile, our library maintains sufficient modularity and extensibility by properly decomposing the model architecture, inference, learning process into highly reusable modules, which allows easily incorporating new models into our framework. It is specially suitable for researchers and practitioners to efficiently reproduce baseline models and develop new models. TextBox is implemented based on PyTorch, and released under Apache License 2.0 at the link https://github.com/RUCAIBox/TextBox.

1 Introduction

Text generation, which has emerged as an important branch of natural language processing (NLP), is often formally referred as natural language generation (NLG). It aims to produce plausible and understandable text in human language from input data (*e.g.,* a sequence, keywords) or machine representation. In the field of text generation, because of incredible performance of deep learning models, many classic tasks have achieved rapid progress, such as machine translation (Vaswani et al., 2017a), dialogue systems (Li et al., 2016), text summarization (See et al., 2017), text paraphrasing (Madnani and Dorr, 2010), and more.

To facilitate the building of text generation models, a few remarkable open-source libraries have been developed (Britz et al., 2017; Klein et al., 2017b; Miller et al., 2017b; Zhu et al., 2018; Hu et al., 2019). These frameworks are mainly designed for some or a small number of specific tasks, particularly machine translation and dialogue systems. They usually focus on a special kind of techniques for text generation such as generative adversarial networks (GAN), or have limitations in covering comprehensive commonly used baseline implementations. Even for an experienced researcher, it is difficult to implement all compared baselines under a unified framework. Therefore, it is highly desirable to re-consider the implementation of text generation algorithms in a unified and modularized way, especially with deep learning.

In order to alleviate the above issues, we initiate a project to provide a unified framework for text generation algorithms. We implement an open source text generation library, called `TextBox`, aiming to enhance the reproducibility of existing models, standardize the implementation and evaluation protocol of text generation algorithms, and ease the development process of new algorithms. Our work is also useful to support several real-world applications in the field of text generation. We have extensively surveyed related text generation libraries and broadly fused their merits into TextBox. The key features and capabilities of our library are summarized in the following three aspects:

• Unified and modularized framework. TextBox is built upon PyTorch (Paszke et al., 2019), which is one of the most popular deep learning framework (especially in the research community). Moreover, it is designed to be highly modularized, by decoupling text generation models into a set of highly reusable modules, including data modules, model modules, evaluation modules, and many common components and functionalities. In our library, it is convenient to compare different text generation algorithms with built-in evaluation protocols via

†Equal contribution.
*Corresponding author.

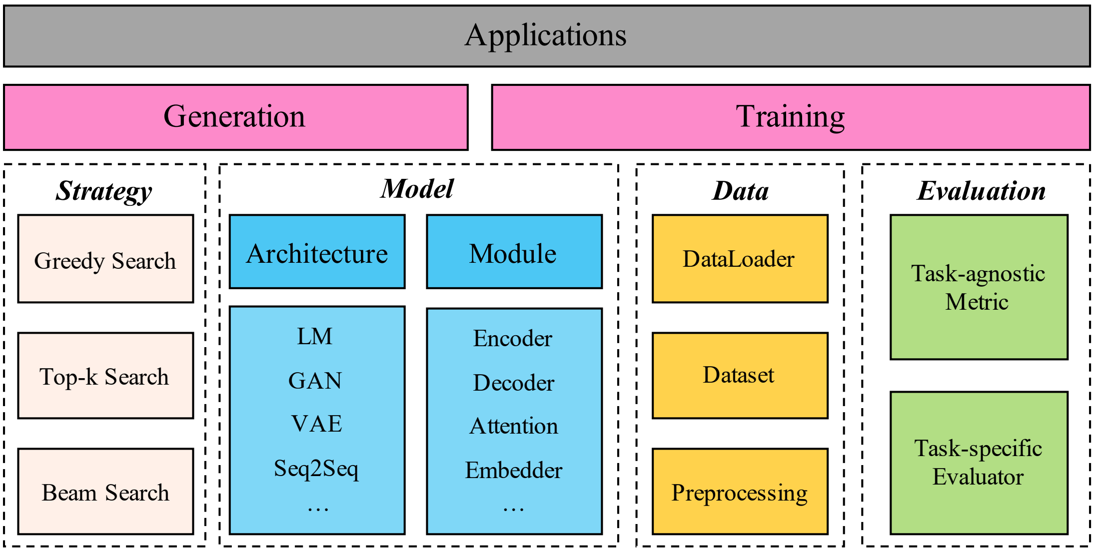

Figure 1: The illustration of the main functionalities and modules in our library TextBox.

simple yet flexible configuration, or develop new text generation models at a highly conceptual level by plugging in or swapping out modules.

• Comprehensive models, benchmark datasets and standardized evaluations. TextBox contains a wide range of text generation models, covering the categories of variational auto-encoder (VAE), generative adversarial networks (GAN), recurrent neural network (RNN), Transformer based models, and pre-trained language models (PLM). We provide flexible supporting mechanisms via the configuration file or command line to run, compare and test these traditional and state-of-the-art algorithms. Based on these models, we implement two major text generation tasks, including unconditional text generation tasks and conditional text generation tasks (*e.g.,* text summarization and machine translation). To construct a reusable benchmark, we incorporate many commonly used datasets with regards to different text generation tasks for evaluation. Our library supports a series of widely adopted evaluation protocols for testing and comparing text generation algorithms, such as perplexity, negative log-likelihood, BLEU, and ROUGE.

• Extensible and flexible framework. TextBox provides convenient interfaces of various common functions or modules in text generation models, *e.g.,* RNN-based encoder-decoder, Transformer-based encoder-decoder and pre-trained language models. Within our library, users are convenient to choose different API interfaces for building and evaluating their own models. Besides, the interfaces of our library are fully compatible with the PyTorch interface which allows seamless integration of user-customized modules, and enables users to integrate external components as needed.

2 Architecture and Design

Figure 1 presents the illustration of the main functionalities and modules in our library TextBox. The configuration module at the bottom helps users set up the experimental environment (*e.g.,* hyper-parameters and running details). Built upon the configuration module, the data, model, and evaluation modules form the core elements of our library. In the following, we describe the detailed structure of these three modules.

2.1 Data Module

A major design principle of our library is to support different text generation tasks. For this purpose, data module is the fundamental part to provide various data structures and functions adapting to different tasks.

For extensibility and reusability, our data module designs a unified data flow feeding input text into the models. The data flow can be described as: input text → `Dataset` → `DataLoader` → models. The class `Dataset` involves two special data structures, *i.e.,* single sequence and paired sequence, which are oriented to unconditional and conditional text generation tasks, respectively, shown in Table 1. The single sequence structure requires users to preprocess input text into one sequence per line in input files, while the paired sequence structure requires users to separate the

Task		Dataset	Train	Dev	Test	Structure
Unconditional		COCO	10,000	10,000	10,000	Single Sequence
		EMNLP	268,586	10,000	10,000	
		IMDB	80,000	10,000	10,000	
Conditional	Translation	IWSLT14 De→EN	153,348	6,970	6,750	Paired Sequence
		WMT14 En→De	3,621,184	36,652	2,931	
	Summarization	GigaWord	3,803,957	189,651	1,936	

Table 1: Benchmarks used in our library TextBox.

source text and target text into two files with one sequence per line in each file. These two data structures are utilized according to the provided hyper-parameter, `task_type`, such as *unconditional*, *translation* and *summarization*. The implementation of `Dataset` contains many common data preprocessing functionalities, such as converting text into lowercase and word tokenization using NLTK[1]. And the class `Dataloader` is based on the above two data structures, which is responsible for organizing the data stream.

In order to compare different generation models in our framework, we have collected some commonly used benchmarks for text generation tasks, which makes it quite convenient for users to start with our library. The statistics of these benchmark datasets for different tasks in our library are presented in Table 1.

2.2 Model module

To support a variety of models, we set up the model module by decoupling the algorithm implementation from other components and extracting a set of frequently used modules, *e.g.,* `encoder`, `decoder`. TextBox allows flexible combinations among these modules following the required interface to connect with input and evaluation modules. Based on this abstract design, it is convenient to switch between different text generation tasks, and change from one modeling paradigm to another by simply plugging in or swapping out modules.

In addition to modularized design, our library also includes a large number of text generation baseline models for reproducibility. At the first released version, we have implemented several baseline models within four categories of text generation models, namely VAE-based, GAN-based, RNN-based, and Transformer-based models, corresponding to different generation tasks. For example, GAN-based models consist of `generator` and `discriminator`, and VAE-based models contain `encoder` and `decoder`. We summarize all the implemented models in Table 2. For all the implemented models, we test their performance on corresponding benchmarks, and invite a code reviewer to examine the correctness of the implementation. Overall, the extensible and comprehensive model modules can be beneficial for fast exploration of new algorithms for a specific task, and quick comparison between different models.

In specific, for each model, we utilize two interface functions, *i.e.,* `calculate_loss` and `generate`, for training and testing, respectively. These functions are general to various text generation algorithms, so that we can implement various algorithms in a highly unified way. Such a design also enables quick development of new models.

In order to improve the quality of generation results, we also implement a series of generation strategies when generating text, such as greedy search, top-k search and beam search. Users are allowed to switch between different generation strategies leading to better performance through setting a hyper-parameter, *i.e.,* `decoding_strategy`. Besides, we add the functions of model saving and loading to store and reuse the learned models, respectively. In the training process, one can print and monitor the change of the loss value and apply training tricks such as warm-up and 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 text generation. It is important that different models should be compared under the unified evaluate protocols, which is useful to standardize the evaluation of text generation.

[1]https://www.nltk.org/

Category	Model	Reference
VAE	LSTM-VAE	(Bowman et al., 2016)
	CNN-VAE	(Yang et al., 2017)
	Hybrid-VAE	(Semeniuta et al., 2017)
GAN	SeqGAN	(Yu et al., 2017)
	TextGAN	(Zhang et al., 2017)
	RankGAN	(Lin et al., 2017)
	MaliGAN	(Che et al., 2017)
	LeakGAN	(Guo et al., 2018)
	MaskGAN	(Fedus et al., 2018)
Seq2Seq	RNN	(Sutskever et al., 2014)
	Transformer	(Vaswani et al., 2017b)
	GPT-2	(Radford et al., 2019)
	XLNet	(Yang et al., 2019)
	BERT2BERT	(Rothe et al., 2020)
	BART	(Lewis et al., 2020)

Table 2: Implemented models in our library TextBox.

Our library supports both logit-based and word-based evaluation metrics. The logit-based metrics (for unconditional text generation task) include negative log-likelihood[2] (NLL) and perplexity[3] (PPL), measuring how well the probability distribution or a probability model predicts a sample compared with the ground-truth. The word-based metrics (for both unconditional and conditional text generation tasks) include the most widely used generation metrics, such as BLEU (Papineni et al., 2002), and ROUGE (Lin, 2004), measuring the ratios of the overlapping n-grams between the generated and real samples. Besides, to evaluate the diversity of generated samples, we also take into account the Self-BLEU (Zhu et al., 2018) metric in text generation. In summary, users can choose different evaluation protocols towards a specific generation task by setting the hyper-parameter, *i.e.,* `metrics`.

In practice, as the model may generate many text pieces, evaluation efficiency is an important concern. Hence, we integrate efficient computing package, `fastBLEU` (Alihosseini et al., 2019), to compute evaluation scores. Compared with other package, `fastBLEU` adopts the multi-threaded C++ implementation.

3 System Usage

In this section, we show a detailed guideline to use our system library. Users can run the existing models or add their own models as needed.

[2]https://en.wikipedia.org/wiki/Likelihood_function
[3]https://en.wikipedia.org/wiki/Perplexity

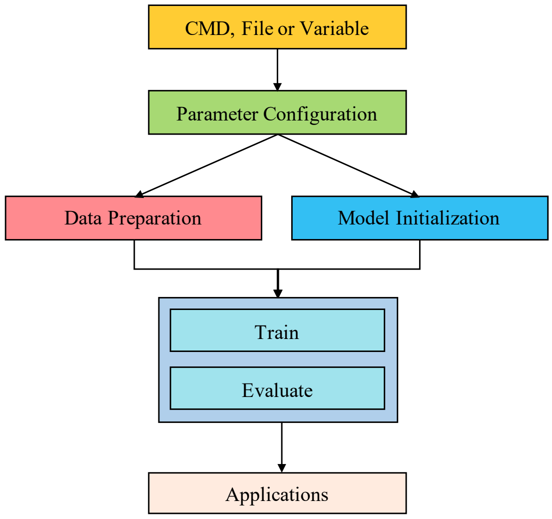

Figure 2: An illustractive usage flow of our library.

3.1 Running existing models

To run an existing model within TextBox, users need to specify the dataset, model, and task by setting hyper-parameters, *e.g.,* `dataset`, `model` and `task`. And then experiments can be run with a simple command-line interface:

```
python run_textbox.py \
  --model=GPT2 --dataset=COCO \
  --task=unconditional
```

TextBox mainly provides two kinds of YAML configuration files, *i.e.,* dataset configuration and model configuration, which allow running many experiments without modifying source code. It also supports users to modify the YAML configuration file and include it in the command line, which is useful for some specifically defined parameters. The above case shows an example that runs GPT-2 (Radford et al., 2019) model on COCO dataset (Lin et al., 2015) for the unconditional text generation task. TextBox is designed to be run on different hardware devices. By default, CUDA devices will be used if users set the hyper-parameter `use_gpu` as *True*, or otherwise CPU will be used. Users can determine the ID of used CUDA devices by setting hyper-parameter `gpu_id`.

Based on the configuration, we provide the auxiliary function to split the dataset into train, validation and test sets according to the provided hyper-parameter `split_ratio`, or load the pre-splitted dataset. Moreover, TextBox also allows users to load and re-train the saved model for speeding up reproduction, rather than training from scratch.

Figure 2 presents a general usage flow when running a model in our library. The running procedure relies on some experimental configuration, obtained from the files, command line or parameter dictionaries. The dataset and model are prepared and initialized according to the configured settings, and the execution module is responsible for training and evaluating models.

3.2 Implementing a New Model

With the unified `Data` and `Evaluation` modules, one needs to implement a specific `Model` class and three mandatory functions as follows:

• `__init__()` function. In this function, the user performs parameters initialization, global variable definition and so on. It is worth noting that, the imported new model should be a sub-class of the abstract model class defined in our library. One can reuse the modules (*e.g.,* Transformer) and layers (*e.g.,* Highway net) already existing in our library for convenience. A configuration file is preferable to conduct further flexible adjustment.

• `calculate_loss()` function. This function calculates the training loss to be optimized and validation loss to avoid overfitting. Based on the returned training loss, our library will automatically involve different optimization methods to learn the model parameters according to pre-defined configuration.

• `generate()` function. This function is employed to generate output text based on input text or free text. Our library also provides several generation strategies, such as beam search and top-k search, for users to improve generation results.

In order to implement user-customized modules, one can reuse functions and classes inherent from our basic modules, or override original functions and add new functions.

4 Performance Evaluation

In order to evaluate TextBox, we implement various text generation models, and compare their performance on unconditional and conditional text generation tasks.

4.1 Unconditional Text Generation

Following previous work, we adopt COCO (Lin et al., 2015) and EMNLP2017 WMT News (Chatterjee et al., 2017) datasets for comparing the performance of five traditional and state-of-the-art models, *i.e.,* LSTM-VAE, SeqGAN, RankGAN, MaliGAN, and GPT-2 models, in the unconditional text generation tasks.

In our experiments, we follow the parameter configurations described in their original papers. Note that the BLEU-n and Self-BLEU-n metrics in our library employ the one-hot weights (*e.g.,* $(0, 0, 0, 1)$ for BLEU-4) instead of average weights, since we consider that one-hot weights can reflect the overlapping n-grams more realistically. Besides, we adopt NLL loss to measure how well the generation probability distribution is, which is computed as follows:

$$\text{NLL} = -\mathbb{E}_{Y_{1:T} \sim P_{real}} \left[\sum_{t=1}^{T} \log(G(y_t | Y_{1:t-1})) \right],$$

where T denotes the length of sequence, P_{real} and G denote the distribution of real data and the text generation model, respectively.

These results are shown in Table 3. In our experiments, these models implemented in our library have the comparable performance compared with the results reported in the original papers. Moreover, the pretrained language model, *i.e.,* GPT-2, achieves consistent and remarkable performance in both COCO and EMNLP datasets. These results are compatible with our expectations.

4.2 Conditional Text Generation

In this section, we show the performance on the test data for conditional text generation task. Due to space limits, we only select the typical conditional text generation task, *i.e.,* machine translation, to compare the attention-based RNN model and Transformer model using three generation strategies, *i.e.,* top-k, greedy, and beam search. The greedy strategy considers the most probable token at each generation step, the top-k search strategy means sorting by probability and zero-ing out the probabilities for anything below the k-th token, and beam search (Vijayakumar et al., 2018) strategy selects the top scoring B candidates from the set of all possible one token extensions of its beams, where B is the beam size.

In our experiments, we adopt the benchmark machine translation dataset, IWSLT2014 German-to-English (Cettolo et al., 2014). RNN model is based on GRU variant with two layers, and Transformer model adopts the base configuration, *i.e.,* six encoder and decoder layers. The beam size for generation is set to 5. Besides BLEU-n, we also provide the average BLEU metric with average weights $(1/4, 1/4, 1/4, 1/4)$ to compare these

Datasets	Models	NLL	BLEU-2	BLEU-3	BLEU-4	Self-BLEU-2	Self-BLEU-3	Self-BLEU-4
COCO	LSTM-VAE	33.02	80.46	51.50	25.89	89.18	61.58	32.69
	SeqGAN	30.56	80.15	49.88	24.95	84.45	54.26	27.42
	RankGAN	31.07	77.36	45.05	21.46	83.13	50.62	23.79
	MailGAN	31.50	80.08	49.52	24.03	84.85	55.32	28.28
	GPT-2	26.82	75.51	58.87	38.22	92.78	75.47	51.74
EMNLP	LSTM-VAE	142.23	58.81	19.70	5.57	72.79	27.04	7.85
	SeqGAN	142.22	63.90	20.89	5.64	70.97	25.56	7.05
	RankGAN	142.27	61.28	19.81	5.58	67.71	23.15	6.63
	MailGAN	149.93	45.00	12.69	3.16	65.10	20.55	5.41
	GPT-2	88.00	55.88	21.65	5.34	75.67	36.71	12.67

Table 3: Performance comparisons of different methods for unconditional text generation under two datasets. *The above results were obtained from our TextBox in preliminary experiments. However, these algorithms were implemented and tuned based on our understanding and experiences, which may not achieve their optimal performance. If you could yield a better result for some specific algorithm, please kindly let us know. We will update this table after the results are verified.

Model	Metric	Top-k	Greedy	Beam
RNN +Attention	**BLEU-2**	26.68	33.74	35.68
	BLEU-3	16.95	23.03	24.94
	BLEU-4	10.85	15.79	17.42
	BLEU	19.66	26.23	28.23
Transformer	**BLEU-2**	30.96	35.48	36.88
	BLEU-3	20.83	24.76	26.10
	BLEU-4	14.16	17.41	18.54
	BLEU	23.91	28.10	29.49

Table 4: Performance comparisons of different generation strategies for translation from German to English.

two sequence-to-sequence models. These results are presented in Table 4.

As we can see from Table 4, Transformer model outperforms the RNN model by a clear margin. Additionally, the adopted beam search generation strategy brings much improvement into the both text generation models compared with the simple greedy and top-k strategies.

The results of all implemented models in other datasets can be acquired from our GitHub page[4].

5 Related Work

Text generation has received much attention from the research community. Several toolkits have been released focusing on one or a few specific tasks or techniques. For example, Tensor2Tensor (Vaswani et al., 2018), MarianNMT (Junczys-Dowmunt et al., 2018) and OpenNMT (Klein et al., 2017a) are designed for machine translation task, while ParlAI (Miller et al., 2017a) and Plato (Papangelis et al., 2020) specialized for dialog research in this field. There are two text generation libraries closely related to our library, including Texygen (Zhu et al., 2018) and Texar (Hu et al., 2019) focusing on GAN technique and high modularization, respectively.

Compared with them, TextBox covers more text generation tasks and models, which is useful for reproducibility. Besides, we also implemente standardized evaluations to commpare different models. Also, our library provides various common modules for convenience. It has a proper focus on text generation field, and provide a comprehensive set of modules and functionalities.

[4]https://github.com/RUCAIBox/TextBox

6 Conclusion

This paper presented a unified, modularized, and extensible text generation library, called `TextBox`. So far, we have implemented 15 text generation models, including VAE-based, GAN-based, RNN-based Transformer-based models and pretrained language models, and 6 benchmark datasets for unconditional and conditional text generation tasks. Moreover, Our library is modularized to easily plug in or swap out components, and extensible to support seamless incorporation of other external modules. In the future, we would further add more models and datasets and consider more functions for covering more text generation tasks.

References

Danial Alihosseini, Ehsan Montahaei, and Mahdieh Soleymani Baghshah. 2019. Jointly measuring diversity and quality in text generation models. In *Proceedings of the Workshop on Methods for Optimizing and Evaluating Neural Language Generation*, pages 90–98, Minneapolis, Minnesota. Association for Computational Linguistics.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, pages 10–21.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451, Copenhagen, Denmark. Association for Computational Linguistics.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*, volume 57.

Rajen Chatterjee, Matteo Negri, Marco Turchi, Marcello Federico, Lucia Specia, and Frédéric Blain. 2017. Guiding neural machine translation decoding with external knowledge. In *Proceedings of the Second Conference on Machine Translation, Volume 1: Research Papers*, pages 157–168, Copenhagen, Denmark. Association for Computational Linguistics.

Tong Che, Yanran Li, Ruixiang Zhang, R. Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. 2017. Maximum-likelihood augmented discrete generative adversarial networks. *CoRR*, abs/1702.07983.

William Fedus, Ian J. Goodfellow, and Andrew M. Dai. 2018. Maskgan: Better text generation via filling in the _______. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Long text generation via adversarial training with leaked information. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 5141–5148.

Zhiting Hu, Haoran Shi, Bowen Tan, Wentao Wang, Zichao Yang, Tiancheng Zhao, Junxian He, Lianhui Qin, Di Wang, Xuezhe Ma, Zhengzhong Liu, Xiaodan Liang, Wanrong Zhu, Devendra Singh Sachan, and Eric P. Xing. 2019. Texar: A modularized, versatile, and extensible toolkit for text generation. 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 159–164. Association for Computational Linguistics.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. 2018. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017a. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada. Association for Computational Linguistics.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017b. Opennmt: Open-source toolkit for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, System Demonstrations*, pages 67–72. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880.

Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. 2016. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1192–1202. The Association for Computational Linguistics.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Kevin Lin, Dianqi Li, Xiaodong He, Ming-Ting Sun, and Zhengyou Zhang. 2017. Adversarial ranking for language generation. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3155–3165.

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. 2015. Microsoft coco: Common objects in context.

Nitin Madnani and Bonnie J. Dorr. 2010. Generating phrasal and sentential paraphrases: A survey of datadriven methods. *Comput. Linguistics*, 36(3):341–387.

Alexander Miller, Will Feng, Dhruv Batra, Antoine Bordes, Adam Fisch, Jiasen Lu, Devi Parikh, and Jason Weston. 2017a. ParlAI: A dialog research software platform. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 79–84, Copenhagen, Denmark. Association for Computational Linguistics.

Alexander H. Miller, Will Feng, Dhruv Batra, Antoine Bordes, Adam Fisch, Jiasen Lu, Devi Parikh, and Jason Weston. 2017b. Parlai: A dialog research software platform. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017 - System Demonstrations*, pages 79–84. Association for Computational Linguistics.

Alexandros Papangelis, Mahdi Namazifar, Chandra Khatri, Yi-Chia Wang, Piero Molino, and Gokhan Tur. 2020. Plato dialogue system: A flexible conversational ai research platform.

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. ACL.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

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.

Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. 2020. Leveraging pre-trained checkpoints for sequence generation tasks. *Trans. Assoc. Comput. Linguistics*, 8:264–280.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1073–1083. Association for Computational Linguistics.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. 2017. A hybrid convolutional variational autoencoder for text generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 627–637.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. 2018. Tensor2Tensor for neural machine translation. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, pages 193–199, Boston, MA. Association for Machine Translation in the Americas.

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

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

Ashwin K. Vijayakumar, Michael Cogswell, Ramprasaath R. Selvaraju, Qing Sun, Stefan Lee, David J. Crandall, and Dhruv Batra. 2018. Diverse beam search for improved description of complex scenes. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 7371–7379. AAAI Press.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764.

Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. 2017. Improved variational autoencoders for text modeling using dilated convolutions. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 3881–3890.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 2852–2858.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. 2017. Adversarial feature matching for text generation. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 4006–4015.

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. 2018. Texygen: A benchmarking platform for text generation models. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, pages 1097–1100. ACM.