MWPToolkit: An open-source framework for deep learning-based Math word problem solvers

Yihuai LAN

Lei WANG

Qiyuan ZHANG

Yunshi LAN

Bing Tian DAI
Singapore Management University, btdai@smu.edu.sg

See next page for additional authors

Author

Yihuai LAN, Lei WANG, Qiyuan ZHANG, Yunshi LAN, Bing Tian DAI, Yan WANG, Dongxiang ZHANG, and Ee-peng LIM

MWPToolkit: An Open-source Framework for Deep Learning-based Math Word Problem Solvers

Yihuai Lan[1][*] **Lei Wang**[2][*] **Qiyuan Zhang**[2][*] **Yunshi Lan**[3][†] **Bing Tian Dai**[2],
Yan Wang[4], **Dongxiang Zhang**[5], **Ee-Peng Lim**[2]

[1] Xihua University, [2]Singapore Management University
[3]East China Normal University, [4] Tencent AI Lab, [5] Zhejiang University

{lei.wang.2019, yslan.2015}@phdcs.smu.edu.sg, {btdai, eplim}@smu.edu.sg
{yifan2250, qiyuanzhang97}@gmail.com, bradenwang@tencent.com, zhangdongxiang@zju.edu.cn

Abstract

Developing automatic Math Word Problem (MWP) solvers has been an interest of NLP researchers since the 1960s. Over the last few years, there are a growing number of datasets and deep learning-based methods proposed for effectively solving MWPs. However, most existing methods are benchmarked solely on one or two datasets, varying in different configurations, which leads to a lack of unified, standardized, fair, and comprehensive comparison between methods. This paper presents MWPToolkit, the first open-source framework for solving MWPs. In MWPToolkit, we decompose the procedure of existing MWP solvers into multiple core components and decouple their models into highly reusable modules. We also provide a hyper-parameter search function to boost the performance. In total, we implement and compare 17 MWP solvers on 4 widely-used single equation generation benchmarks and 2 multiple equations generation benchmarks. These features enable our MWPToolkit to be suitable for researchers to reproduce advanced baseline models and develop new MWP solvers quickly. Code and documents are available at https://github.com/LYH-YF/MWPToolkit.

1 Introduction

Developing automatic Math Word Problems (MWPs) solvers has been an interest of NLP researchers since the 1960s (Feigenbaum and Feldman, 1963; Bobrow, 1964). As shown in Table 1, when solving a MWP, machines need to make inferences based on the given textual problem description and question. It requires machines to translate the natural language text into valid and solvable equations according to context, numbers, and unknown variables in the text and then compute to obtain the numerical values as the answer.

Single Equation Generation:
Paco had 26 salty cookies and 17 sweet cookies. He ate 14 sweet cookies and 9 salty cookies. How many salty cookies did Paco have left?
Equation:
$x = 26 - 9$
Answer:
$x = 17$

Multiple Equations Generation:
Jerome bought 12 CDs. Some cost 7.50\$ each, and the rest cost 6.50 each . How many CDs were bought at each price if he spent 82 dollars?
Equation:
$7.5 \times x + 6.5 \times y = 82,$
$x + y = 12$
Answer:
$x = 4, y = 8$

Table 1: Two examples of Math Word Problems. We display single equation generation and multiple equations generation.

Over the last few years, there are a growing number of datasets (Kushman et al., 2014; Koncel-Kedziorski et al., 2016; Upadhyay and Chang, 2017; Huang et al., 2016; Wang et al., 2017; Qin et al., 2020; Miao et al., 2020; Patel et al., 2021) and deep learning-based methods that have been proposed to solve MWPs, including Seq2Seq (Wang et al., 2017, 2018a; Chiang and Chen, 2019; Li et al., 2019), Seq2Tree (Wang et al., 2019; Liu et al., 2019a; Xie and Sun, 2019; Qin et al., 2020), Graph2Tree (Zhang et al., 2020b; Shen and Jin, 2020), and Pre-trained Language Models (Kim et al., 2020). However, most existing MWP solving methods are evaluated solely on one or two datasets, varying in different settings (i.e., different train-test split and k-fold cross validation), which leads to a lack of unified, standardized, fair, and comprehensive comparison between methods. Moreover, it is time-consuming and complicated to re-implement prior methods as baselines, which leads to the difficulty making a consistent conclusion in term of performance comparison to other methods. Thus

---

[*]Equal contribution.
[†]Corresponding author.

it severely hinders the development of research in MWPs community.

To encourage the development of this field, we present `MWPToolkit`, the first open-source framework for deep learning-based MWP solvers. To unify MWP methods into `MWPToolkit`, we design the framework of MWP solvers as an architecture with multiple core components: config, data, model and evaluation. We further decouple the components into highly reusable modules and deploy them into `MWPToolkit`. Thus, it is easily extensible and convenient to develop new models by combing existing modules and replacing individual modules with proposed ones. Besides, we also develop a hyper-parameter search function for all methods developed in `MWPToolkit`, which helps mitigate the negative impact caused by sub-optimal hyper-parameters.

`MWPToolkit` includes comprehensive benchmark datasets and models. So far, we have incorporated 6 widely-used MWP datasets and 17 models. The datasets contain 4 datasets that are single equation generation and 2 datasets that are multiple equation generation. The models include Seq2seq, Seq2tree, Graph2tree, and commonly-used non-pretrained (AttSeq2Seq (Bahdanau et al., 2014), LSTMVAE (Zhang et al., 2016), and Transformer (Vaswani et al., 2017)) and pretrained models (GPT-2 (Radford et al., 2019), BERTGen (Devlin et al., 2018), and RoBERTaGen (Liu et al., 2019b)). Currently, our framework supports built-in evaluation protocols including equation accuracy and answer accuracy for two types of generation. In our `MWPToolkit`, users can run, compare, and test models on MWP tasks under the same setting with simple configuration files and command lines.

To ensure that our re-implementations in `MWPToolkit` are correct and the experiments by our framework are reliable, we set the same hyper-parameters as the ones in original papers and ensure the re-implemented result should be approximate to the reported result. In this paper, we provide a set of results of 17 models on 6 datasets with the same k-fold cross-validation setting after the built-in hyper-parameter search. We hope the community can benefit from the results of this comprehensive comparison, better understand existing MWP methods, and easily develop new and powerful models by utilizing our `MWPToolkit`.
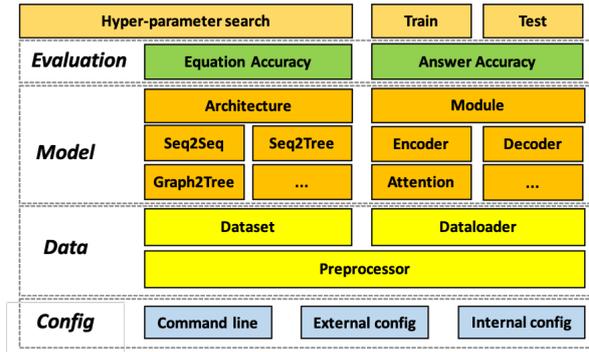

Figure 1: The overall framework of `MWPToolkit`.

2 `MWPToolkit` Framework

The overall framework of our `MWPToolkit` is presented in Figure 1, including the config component, data component, model component, evaluation component, and execution from bottom to top. The config component is used to set up the experimental configuration, which supports the following components. The data component preprocess different datasets into a unified form used in the subsequent components. The model component is responsible for model construction. After determining specific evaluation metrics in the evaluation component, the execution part is used to train with a given group of hyper-parameters or hyper-parameter searching and evaluate models with a specific setting, i.e., train-test split or k-fold cross-validation. Note that in our `MWPToolkit`, users can use the given random seed to reproduce results completely. In the following part, we present the details of config, data, model, evaluation components and execution part.

2.1 Config Component

Config component serves as the core human-system interaction component in which the developer can specify experimental configurations. The configuration part of our framework consists of *Command lines*, *External config* and *Internal config*. The default configuration is defined in internal configuration file. Users can flexibly and simply use the command lines to modify the major settings of the experiment. They can also have more customized settings with external configuration files, which is beneficial for the duplication of research results.

2.2 Data Component

Any raw dataset in the data module follows the predefined data flow to convert raw data into a unified

Dataset	Language	Task	# Examples	# Multi-Equ	Hard Set	Reference
MAWPS-s	en	Single equation generation	1,987	-	-	(Koncel-Kedziorski et al., 2016)
Draw1K	en	Multiple equations generation	1,000	745	-	(Upadhyay and Chang, 2017)
Math23K	zh	Single equation generation	23,162	-	-	(Wang et al., 2017)
HMWP	zh	Multiple equations generation	5,491	1,789	-	(Qin et al., 2020)
ASDiv-a	en	Single equation generation	1,238	-	-	(Miao et al., 2020)
SVAMP	en	Single equation generation	3,138	-	1,000	(Patel et al., 2021)

Table 2: The collected datasets in MWPToolkit. "# Multi-Equ" stands for the number of examples, the targets of which are multiple equations. "Hard Set" means an external challenging or adversarial test set.

format of data as input for the the following model component: raw data $\mapsto$ *Preprocessor* $\mapsto$ *Dataset* $\mapsto$ *Dataloader* $\mapsto$ processed data.

We display the statistics of all built-in datasets in Table 2. As we can see, raw datasets vary in formats and features, so we first preprocess these raw datasets and convert them to a unified format. In *Preprocessor*, we first tokenize input text by a tokenizer, extract numbers from the tokenized text by some simple rules, and record extracted numbers and map them into position-aware special tokens (a.k.a, number mapping). To avoid infinite generation space in target, we convert equations into equation templates by replacing numbers with position-aware special tokens from number mapping. We add another special token $< bridge >$ for the multiple equations generation task to convert the equation forest to a tree. Hence it can be treated as the single equation generation task. Note that different models require us to prepare different data formats and features. For example, Bert-based MWP models use WordPiece embeddings (Wu et al., 2016) instead of word embeddings. For another example, Graph2tree models utilize external information, like the results of the dependency parser, to construct the graph. Hence we customize the preparation of data preprocessing after basic preprocessing. Users can add a new dataset to our framework by referring to our processing step.

We design the *Dataset* module to do data preparation. The design of *AbstractDataset* is to include some shared attributes and basic functions. Any specific dataset class or user customized dataset class can inherit *AbstractDataset* with few modifications.

After the *Dataset* module, *DataLoader* module selects features from the processed data to form tensor data (PyTorch), which can be directly used in the model component. *AbstractDataLoader* class, including common attributes and basic functions, allows users to easily create new *DataLoaders* for new models.

2.3 Model Component

We organize the implementations of MWP solving methods in the model component. The objective of the model component is to disentangle model implementation from data processing, evaluation, execution, and other parts, which benefits users to focus on the model itself. We unify the implementation of a model. Specifically, we provide three interface functions for loss calculation, prediction, and test, respectively. When users deploy or add a new model with MWPToolkit, they can simply focus on these interface functions without considering other parts. Such a design enables users to develop new algorithms easily and quickly. Besides, the commonly-used components of the implemented models have been decoupled and shared across different models for code re-usage.

We have carefully surveyed the recent literatures and selected the commonly-used MWP solving models in our library. As the first released version, we have implemented 17 MWP solving models in the four categories: *Seq2seq*, *Seq2tree*, *Graph2tree*, and *Pretrained Language Models*. In the future, more methods will be added into our toolkit as the regular update, like MathDQN (Wang et al., 2018b), EPT (Kim et al., 2020), KAS2T (Wu et al., 2020), and NumS2T (Wu et al., 2021). We summarize all the implemented models in Table 3.

For all the implemented models in MWPToolkit, we ensure that these re-implementations are correct and that the experiments by our framework are reliable. We set the same hyper-parameters as the ones in original papers and ensure the re-implemented result should be approximate to the reported result. The detailed performance comparison between our re-implementation and original results is shown in Table 4 and Table 5.

Type	Model	Encoder	Decoder	Pretrained Model	Reference
Seq2Seq	DNS	GRU	LSTM	-	(Wang et al., 2017)
	MathEN	BiLSTM	LSTM	-	(Wang et al., 2018b)
	Saligned	BiLSTM	LSTM	-	(Chiang and Chen, 2019)
	GroupATT	BiLSTM	LSTM	-	(Li et al., 2019)
	AttSeq2Seq	LSTM	LSTM	-	(Bahdanau et al., 2014)
	LSTMVAE	LSTM	LSTM	-	(Zhang et al., 2016)
	Transformer	Transformer	Transformer	-	(Vaswani et al., 2017)
Seq2Tree	TRNN	BiLSTM	LSTM	-	(Wang et al., 2019)
	AST-Dec	BiLSTM	TreeLSTM	-	(Liu et al., 2019a)
	GTS	GRU	TreeDecoder	-	(Xie and Sun, 2019)
	SAU-Solver	GRU	TreeDecoder	-	(Qin et al., 2020)
	TSN	GRU	TreeDecoder	-	(Zhang et al., 2020a)
Graph2Tree	Graph2Tree	LSTM+GCN	TreeDecoder	-	(Zhang et al., 2020b)
	MulltiE&D	GRU+GCN	GRU	-	(Shen and Jin, 2020)
Pretrained based	BERTGen	BERT	Transformer	BERT	(Devlin et al., 2018)
	RoBERTaGen	RoBERTa	Transformer	RoBERTa	(Liu et al., 2019b)
	GPT-2	-	Transformer	GPT-2	(Radford et al., 2019)

Table 3: The implemented models in *MWPToolkit*. Currently, the toolkit includes four types of models: Seq2Seq, Seq2Tree, Graph2Tree, and Pretrained models.

2.4 Evaluation Component

Our toolkit standardizes the evaluation of MWP solving models with *Equation Accuracy* and *Answer Accuracy* for single equation generation or multiple equations generations. Equations accuracy is computed by measuring the exact match of predicted equations and ground-truth equations. For answer accuracy, we first check the validation of predicted equations. The answer accuracy is 0 if the predicted equations are invalid or unsolvable. We then calculate the answer using our encapsulated calculation module and compare it with the ground-truth answer. If their difference is less than $1e-5$, we regard it as 1 and 0 otherwise.

2.5 Execution

On the top of above components, we implement training and testing paradigms, where two options are provided. One is to follow the standard train-test splitting if the splitting data is given in the original dataset. Another is to conduct k-fold cross-validation. To improve the performance, we also implement a series of hyper-parameter search strategies, such as beam search, greedy search, sampling strategy.

3 Usage of **MWPToolkit**

This section shows how users run the existing models and incorporate new models with our toolkit.

3.1 Running Existing Models

Figure 2 shows the procedure of running an existing model in *MWPToolkit*. Firstly, users need a con-

figuration file to set up the experimental environment. In the configuration file, users should specify an existing model, a dataset, a task, and other hyper-parameters regarding the model and training. The class *Configure()* loads all information on configuration for the subsequent steps. Then, the toolkit preprocesses data and organizes the dataset by calling the function *create_dataset()*. Based on the processed dataset, users can use the function *create_dataloader()* to convert data to the tensor format for training, validation, and test with the specified batch size and other hyper-parameters like the maximum length of the input sequence. Later, the function *get_model()* is used to get access to the model that the users would like to run. Next, users can employ the function *get_trainer()* to build an executable MWP solver based on the dataloader, model, and specified task obtained in previous steps. Eventually, users run the function *trainer.fit()* to start the training and evaluation.

3.2 Developing New MWP Sovlers

`MWPToolkit` is an extensible and easy-to-use framework. It is convenient for users to add a new MWP solving model or a new benchmark dataset into `MWPToolkit` by filling up the specified interfaces. In the following, we present the details of how to add a new dataset and model.

3.2.1 Add a New Dataset

To add a new dataset, users need to inherit the abstract class *AbstractDataset* and are required to fill in the functions: *__init__()*, *_load_data()*,
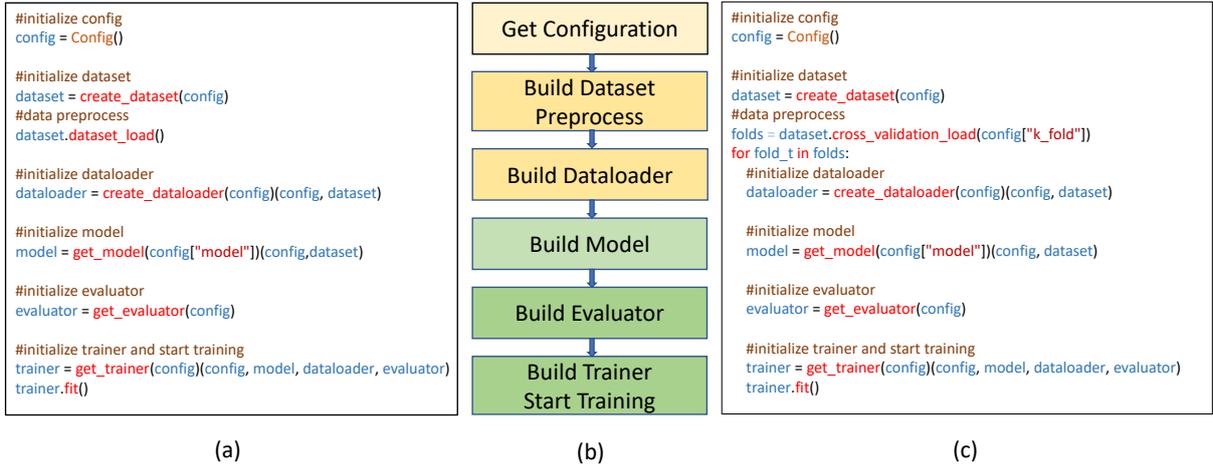
(a)
```
#initialize config
config = Config()

#initialize dataset
dataset = create_dataset(config)
#data preprocess
dataset.dataset_load()

#initialize dataloader
dataloader = create_dataloader(config)(config, dataset)

#initialize model
model = get_model(config["model"])(config,dataset)

#initialize evaluator
evaluator = get_evaluator(config)

#initialize trainer and start training
trainer = get_trainer(config)(config, model, dataloader, evaluator)
trainer.fit()
```

(b)
Get Configuration

Build Dataset
Preprocess

Build Dataloader

Build Model

Build Evaluator

Build Trainer
Start Training

(c)
```
#initialize config
config = Config()

#initialize dataset
dataset = create_dataset(config)
#data preprocess
folds = dataset.cross_validation_load(config["k_fold"])
for fold_t in folds:
    #initialize dataloader
    dataloader = create_dataloader(config)(config, dataset)

    #initialize model
    model = get_model(config["model"])(config, dataset)

    #initialize evaluator
    evaluator = get_evaluator(config)

    #initialize trainer and start training
    trainer = get_trainer(config)(config, model, dataloader, evaluator)
    trainer.fit()
```

Figure 2: Examples of how to use our `MWPToolkit`. Figure (a) illustrates the code of running models using the train-test split. Figure (b) is about the usage flow of the toolkit. Figure (c) shows the code of running models using k-fold cross-validation.

_preprocess(), and *_build_vocab()*. *__init__()* is used to set up parameters of the dataset. *_load_data()* is used to load the entire raw dataset or split training, validation, and test sets. The function *_preprocess()* is used to process the raw dataset and prepare the processed dataset for later usage in other modules. *_build_vocab()* is used to build shared or separate vocabulary dictionaries for the encoder and decoder. The users can fill in the above required implemented functions to create a customized dataset class quickly.

3.2.2 Add a New Model

To add a new model, users need to complete three functions in a new model class: *__init__()*, *calculate_loss()*, and *model_test()*. *__init__()* is used to build the model and initialize the parameters. *calculate_loss()* is used to calculate the loss for training based on the model prediction and ground-truth equations. *model_test()* prepares suitable evaluation protocols and is executed for evaluation of model performance on a specified dataset and task.

4 Performance Comparison

To evaluate the models in `MWPToolkit`, we conduct extensive experiments to compare 17 MWP solving models on 4 widely-used single equation generation benchmark datasets and 2 multiple equations generation benchmarks. In our experiments, if models have been evaluated on certain datasets, we run models with the parameter configurations described in their original papers. Otherwise, we run hyper-parameter search to search a group of hyper-parameters for these models. In the following sections, we discuss the detailed performance

comparison.

4.1 Single Equation Generation

Table 4 displays the results of models on single equation generation datasets. We include four datasets for the single equation generation task, i.e., Math23k, MAWPS-s, ASDiv-a, and SVAMP. We can see from Table 4. We report three types of results for a model on a dataset. The first two columns are equation accuracy (Equ. Acc) and answer accuracy (Ans. Acc). The third column is the results reported in the original papers under their settings, such as train-test split (* means train-test split) or 5-fold cross-validation. Note that for any models, the results based on 5-fold cross-validation are less than those based on train-test split because the number of training examples of 5-fold cross-validation is smaller than those in train-test split. As shown in table 4, answer accuracy in the k-fold cross-validation setting by our `MWPToolkit` are either better than original answer accuracy or close to them. Through our experiments, we can observe that Graph2Tree and RoBERTaGen are the most effective baselines, which means it is potential for researchers to develop better models based on these two model categories.

4.2 Multiple Equations Generation

We add another special token $< bridge >$ to convert equation forest to a tree for the multiple equations generation task. Hence it can be treated as the single equation generation task. We apply the 17 models on two multiple equation generation datasets, i.e., DRAW1K and HMWP. Their results are shown in Table 5. As we can observe in ta-

Model	Datasets												
	Math23K			MAWPS-s			ASDiv-a			SVAMP		
	Equ. Acc	Ans. Acc	OA Acc	Equ. Acc	Ans. Acc	OA Acc	Equ. Acc	Ans. Acc	OA Acc	Equ. Acc	Ans. Acc	OA Acc
DNS	57.1	67.5	58.1	78.9	86.3	59.5	63.0	66.2	-	22.1	24.2	-
MathEN	66.7	69.5	66.7*	85.9	86.4	69.2	64.3	64.7	-	21.8	25.0	-
Saligned	59.1	69.0	65.8	**86.0**	86.3	-	66.0	67.9	-	23.9	26.1	-
GroupAtt	56.7	66.6	66.9	84.7	85.3	76.1	59.5	61.0	-	19.2	21.5	-
AttSeq	57.1	68.7	59.6	79.4	87.0	79.7	64.2	68.3	55.5	23.0	25.4	24.2
LSTMVAE	59.0	70.0	-	79.8	88.2	-	64.0	68.7	-	23.2	25.9	-
Transformer	52.3	61.5	62.3*	77.9	85.6	-	57.2	59.3	-	18.4	20.7	-
TRNN	65.0	68.1	66.9*	**86.0**	86.5	66.8	68.9	69.3	-	22.6	26.1	-
AST-Dec	57.5	67.7	69.0*	84.1	84.8	-	54.5	56.0	-	21.9	24.7	-
GTS	63.4	74.2	74.3	83.5	84.1	82.6	67.7	69.9	**71.4**	25.6	29.1	30.8
SAU-Solver	64.6	75.1	74.8	83.4	84.0	-	68.5	71.2	-	27.1	29.7	-
TSN	63.8	74.4	75.1	84.0	84.7	**84.4**	68.5	71.0	-	25.7	29.0	-
Graph2Tree	64.9	75.3	75.5	84.9	85.6	83.7	**72.4**	75.3	-	**31.6**	**35.0**	**36.5**
MultiE&D	**65.5**	76.5	**76.9**	83.2	84.1	-	70.5	72.6	-	29.3	32.4	-
BERTGen	64.8	76.6	-	79.0	86.9	-	68.7	71.5	-	22.2	24.8	-
RoBERTaGen	65.2	**76.9**	-	80.8	**88.4**	-	68.7	72.1	-	27.9	30.3	-
GPT-2	63.8	74.3	-	75.4	75.9	-	59.9	61.4	-	22.5	25.7	-

Table 4: Performance comparisons of different methods on single equation generation task. "Equ. Acc" is equation accuracy. "Ans. Acc" stands for answer accuracy. "OA Acc" means original answer accuracy in previous papers. "*" means train-test split.

Model	Datasets					
	Draw1K			HMWP		
	Equ. Acc	Ans. Acc	OA Acc	Equ. Acc	Ans. Acc	OA Acc
DNS	35.8	36.8	-	24.0	32.7	-
MathEN	38.2	39.5	-	32.4	43.7	-
Saligned	36.7	37.8	-	31.0	41.8	-
GroupAtt	30.4	31.4	-	25.2	33.2	-
AttSeq	39.7	41.2	-	32.9	44.7	-
LSTMVAE	**40.9**	**42.3**	-	33.6	45.9	-
Transformer	27.1	28.3	-	24.4	32.4	-
TRNN	27.4	28.9	-	27.2	36.8	-
AST-Dec	26.0	26.7	-	24.9	32.0	-
GTS	38.6	39.9	-	33.7	44.6	-
SAU-Solver	38.4	39.2	-	33.1	43.7	**44.8**
TSN	39.3	40.4	-	34.3	44.9	-
Graph2Tree	39.8	41.0	-	34.4	45.1	-
MultiE&D	38.1	39.2	-	34.6	45.3	-
BERTGen	33.9	35.0	-	29.2	39.5	-
RoBERTaGen	34.2	34.9	-	30.6	41.0	-
GPT-2	30.7	31.5	-	**36.3**	**49.0**	-

Table 5: Performance comparison of different methods on multiple equations generation task. "Equ. Acc" is equation accuracy. "Ans. Acc" stands for answer accuracy. "OA Acc" means original answer accuracy in previous papers.

ble 5, to our surprise, LSTMVAE achieves the best performance on Draw1K, and GPT-2 achieves the best performance on HMWP. Most researchers focus on improving performance on the single equation generation task, while few researchers develop models on the multiple equation generation task in the MWP solving community. We hope the results shown in table 5 can help researchers develop more powerful and effective models for solving MWPs with multiple equations as their generation targets.

5 Related Work

In NLP community, there have been a number of toolkits that managed to summarize the existing methods and establish a unified framework for a certain task, such as OpenNMT (Klein et al., 2017)

and TexBox (Li et al., 2021) for text generation tasks, ExplainaBoard (Liu et al., 2021) for evaluating interpretable models, Photon (Zeng et al., 2020) for text-to-SQL tasks, and Huggingface's Transformers (Wolf et al., 2020) for model pretraining. To our best knowledge, there is no such a unified and comprehensive framework for MWPs solving task. Therefore, we release MWPToolkit, which includes a considerable number of benchmark datasets and deep learning-based solvers.

Recently, a large number of new MWP solving methods have been proposed, including graph neural networks based (Li et al., 2020), template based (Lee et al., 2021), neural symbolic (Qin et al., 2021), pre-trained based (Liang et al., 2021), multilingual pre-trained based methods (Tan et al., 2021), and solvers using external information and signals (Liang and Zhang; Wu et al., 2021). In addition, weakly supervised learning for MWP solving (Hong et al., 2021; Chatterjee et al., 2021) and supervised learning for geometric problem solving (Lu et al., 2021; Chen et al., 2021) have recently attracted much researchers' attention. More work on math word and geometric problem solving can be found in the survey paper (Zhang et al., 2019). We will update more above methods to the toolkit in the future.

6 Conclusion

This paper presented an extensible, modularized, and easy-to-use toolkit, MWPToolkit, the first open-source framework for solving MWPs. In our MWPToolkit, we decompose the procedure of existing MWP methods into multiple components and

decouple their models into highly reusable modules. We also provide a hyper-parameter search function for a fairer comparison. Furthermore, we implement and compare 17 MWP solving models, including Seq2Seq, Seq2tree, Graph2Tree, and commonly-used non-pretrained models and pretrained models, on 4 widely-used single equation generation benchmark datasets and 2 multiple equations generation benchmarks. These features enable our `MWPToolkit` to be suitable for researchers to reproduce reliable baseline models and develop new MWP solving methods quickly.

In the future, we will continue to add more benchmark datasets, the latest published MWP solvers, and commonly-used models into `MWPToolkit` as the regular update. We welcome more researchers and engineers to join, develop, maintain, and improve this toolkit to push forward the development of the research on MWPs solving.

Acknowledgement

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

D. Bobrow. 1964. Natural language input for a computer problem solving system. pages 146–226.

Oishik Chatterjee, Aashish Waikar, Vishwajeet Kumar, Ganesh Ramakrishnan, and Kavi Arya. 2021. A weakly supervised model for solving math word problems. *arXiv preprint arXiv:2104.06722*.

Jiaqi Chen, Jianheng Tang, Jinghui Qin, Xiaodan Liang, Lingbo Liu, Eric P Xing, and Liang Lin. 2021. Geoqa: A geometric question answering benchmark towards multimodal numerical reasoning. *arXiv preprint arXiv:2105.14517*.

Ting-Rui Chiang and Yun-Nung Chen. 2019. Semantically-aligned equation generation for solving and reasoning math word problems. In *NAACL-HLT (1)*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Edward A. Feigenbaum and Julian Feldman. 1963. *Computers and Thought*. McGraw-Hill, Inc., New York, NY, USA.

Yining Hong, Qing Li, Daniel Ciao, Siyuan Huang, and Song-Chun Zhu. 2021. Learning by fixing: Solving math word problems with weak supervision. In *AAAI Conference on Artificial Intelligence*.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. 2016. How well do computers solve math word problems? large-scale dataset construction and evaluation.

Bugeun Kim, Kyung Seo Ki, Donggeon Lee, and Gahgene Gweon. 2020. Point to the expression: Solving algebraic word problems using the expression-pointer transformer model. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3768–3779.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. 2017. Opennmt: Opensource toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *NAACL*, pages 1152–1157.

Nate Kushman, Luke Zettlemoyer, Regina Barzilay, and Yoav Artzi. 2014. Learning to automatically solve algebra word problems. In *ACL*, pages 271–281.

Donggeon Lee, Kyung Seo Ki, Bugeun Kim, and Gahgene Gweon. 2021. Tm-generation model: a template-based method for automatically solving mathematical word problems. *The Journal of Supercomputing*, pages 1–17.

Jierui Li, Lei Wang, Jipeng Zhang, Yan Wang, Bing Tian Dai, and Dongxiang Zhang. 2019. Modeling intra-relation in math word problems with different functional multi-head attentions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6162–6167.

Junyi Li, Tianyi Tang, Gaole He, Jinhao Jiang, Xiaoxuan Hu, Puzhao Xie, Zhipeng Chen, Zhuohao Yu, Wayne Xin Zhao, and Ji-Rong Wen. 2021. TextBox: A unified, modularized, and extensible framework for text generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 30–39. Association for Computational Linguistics.

Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng Zhong. 2020. Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. *arXiv preprint arXiv:2004.13781*.

Zhenwen Liang, Jipeng Zhang, Jie Shao, and Xiangliang Zhang. 2021. Mwp-bert: A strong baseline for math word problems. *arXiv preprint arXiv:2107.13435*.

Zhenwen Liang and Xiangliang Zhang. Solving math word problems with teacher supervision.

Pengfei Liu, Jinlan Fu, Yang Xiao, Weizhe Yuan, Shuaicheng Chang, Junqi Dai, Yixin Liu, Zihuiwen Ye, and Graham Neubig. 2021. Explainaboard: An explainable leaderboard for nlp. *arXiv preprint arXiv:2104.06387*.

Qianying Liu, Wenyv Guan, Sujian Li, and Daisuke Kawahara. 2019a. Tree-structured decoding for solving math word problems. 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)*, pages 2370–2379.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. 2021. Inter-gps: Interpretable geometry problem solving with formal language and symbolic reasoning. *arXiv preprint arXiv:2105.04165*.

Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing english math word problem solvers. In *ACL*.

Arkil Patel, S. Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? In *NAACL*.

Jinghui Qin, Xiaodan Liang, Yining Hong, Jianheng Tang, and Liang Lin. 2021. Neural-symbolic solver for math word problems with auxiliary tasks. *arXiv preprint arXiv:2107.01431*.

Jinghui Qin, Lihui Lin, Xiaodan Liang, Rumin Zhang, and Liang Lin. 2020. Semantically-aligned universal tree-structured solver for math word problems. *arXiv preprint arXiv:2010.06823*.

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

Yibin Shen and Cheqing Jin. 2020. Solving math word problems with multi-encoders and multi-decoders. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2924–2934.

Minghuan Tan, Lei Wang, Lingxiao Jiang, and Jing Jiang. 2021. Investigating math word problems using pretrained multilingual language models. *arXiv preprint arXiv:2105.08928*.

Shyam Upadhyay and Ming-Wei Chang. 2017. Annotating derivations: A new evaluation strategy and dataset for algebra word problems. In *EACL*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018a. Translating a math word problem to an expression tree. *arXiv preprint arXiv:1811.05632*.

Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018b. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7144–7151.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854.

Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

Qinzhuo Wu, Qi Zhang, Jinlan Fu, and Xuan-Jing Huang. 2020. A knowledge-aware sequence-to-tree network for math word problem solving. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7137–7146.

Qinzhuo Wu, Qi Zhang, Zhongyu Wei, and Xuanjing Huang. 2021. Math word problem solving with explicit numerical values. In *ACL*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *IJCAI*, pages 5299–5305.

Jichuan Zeng, Xi Victoria Lin, Caiming Xiong, Richard Socher, Michael R Lyu, Irwin King, and Steven CH Hoi. 2020. Photon: A robust cross-domain text-to-sql system. *arXiv preprint arXiv:2007.15280*.

Biao Zhang, Deyi Xiong, Jinsong Su, Hong Duan, and Min Zhang. 2016. Variational neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 521–530, Austin, Texas. Association for Computational Linguistics.

Dongxiang Zhang, Lei Wang, Luming Zhang, Bing Tian Dai, and Heng Tao Shen. 2019. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE transactions on pattern analysis and machine intelligence*, 42(9):2287–2305.

Jipeng Zhang, Ka Wei LEE, Ee-Peng Lim, Wei Qin, Lei Wang, Jie Shao, Qianru Sun, et al. 2020a. Teacher-student networks with multiple decoders for solving math word problem.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020b. Graph-to-tree learning for solving math word problems. Association for Computational Linguistics.

Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Kaiyuan Li, Yushuo Chen, Yujie Lu, Hui Wang, Changxin Tian, Xingyu Pan, et al. 2020. Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms. *arXiv preprint arXiv:2011.01731*.

Kun Zhou, Xiaolei Wang, Yuanhang Zhou, Chenzhan Shang, Yuan Cheng, Wayne Xin Zhao, Yaliang Li, and Ji-Rong Wen. 2021. Crslab: An open-source toolkit for building conversational recommender system. *arXiv preprint arXiv:2101.00939*.