`UnifiedGEC`: Integrating Grammatical Error Correction Approaches for Multi-languages with a Unified Framework

Yike Zhao, Xiaoman Wang, Yunshi Lan*, Weining Qian
School of Data Science and Engineering, East China Normal University
Shanghai Engineering Research Center of Big Data Management
`{ykzhao,xmwang}@stu.ecnu.edu.cn, {yslan,wnqian}@dase.ecnu.edu.cn`

Abstract

Grammatical Error Correction is an important research direction in NLP field. Although many models of different architectures and datasets across different languages have been developed to support the research, there is a lack of a comprehensive evaluation on these models, and different architectures make it hard for developers to implement these models on their own. To address this limitation, we present **`UnifiedGEC`**, the first open-source GEC-oriented toolkit, which consists of several core components and reusable modules. In `UnifiedGEC`, we integrate 5 widely-used GEC models and compare their performance on 7 datasets in different languages. Additionally, GEC-related modules such as data augmentation, prompt engineering are also deployed in it. Developers are allowed to implement new models, run and evaluate on existing benchmarks through our framework in a simple way. Code, documents and detailed results of `UnifiedGEC` are available at https://github.com/AnKate/UnifiedGEC.

1 Introduction

Grammatical Error Correction (GEC), aiming to identify and correct grammatical errors in a sentence automatically, is an important research direction in the field of NLP. It has a wide range of applications in real life, including writing assistant, search engine, language learning education (Bryant et al., 2023; Grundkiewicz et al., 2020; Knill et al., 2019), which has attracted much attention in academic and industry fields. Existing GEC tools or commercial products such as Grammarly[1], QuillBot[2], ChatGPT[3] serve the customers with the close techniques. But we notice there is a need of an open and unified framework for more researches on GEC.

Through the investigation, we find there have been many datasets designed for GEC tasks in different languages (Zhao et al., 2018; Yannakoudakis et al., 2011; Ng et al., 2014; Zhang et al., 2022a; Náplava and Straka, 2019; Yamada et al., 2020; Boyd et al., 2014). Meanwhile, a lot of models have been proposed for GEC tasks, categorized into *Seq2Seq* models (Vaswani et al., 2017; Raffel et al., 2023; Zhang et al., 2022b) and *Seq2Edit* models (Omelianchuk et al., 2020; Gu et al., 2019). However, these models are designed with different architectures, and most of them are evaluated on only one or two datasets with different preprocessing pipelines, resulting in the lack of a unified comparison between these models. Furthermore, that makes it difficult for developers to implement these models on their own.

To tackle these issues, we propose `UnifiedGEC`, which is featured with several distinct characteristics: (1) **Modularization**: We decouple GEC methods with different architectures into modularized and reusable components, namely config, data, model, trainer, and evaluation components. We integrate them in a unified framework, which can be adapted to general GEC methods. (2) **Comprehensiveness**: We deploy datasets as well as dataloaders for different languages, models of different architectures, and mainstream evaluators for GEC tasks in our framework, which allows developers to run, evaluate and simply implement models. (3) **Extensibility**: `UnifiedGEC` toolkit provides user-friendly interfaces for various usages. The components in the unified framework are modeled as exchangeable modules, which makes it convenient for developers to develop their methods.

To validate the effectiveness and credibility of `UnifiedGEC` toolkit, we also conduct extensive experiments on GEC tasks via our toolkit, which achieves close results to the prior reports. Fur-

*Corresponding Author.
[1] `https://app.grammarly.com/`
[2] `https://quillbot.com/`
[3] `https://platform.openai.com/docs/api-reference/introduction`

thermore, `UnifiedGEC` toolkit makes it easier to perform research on low-resource GEC, GEC with data augmentation and other cases. In this paper, we demonstrate extensive experiments on GEC tasks. We hope that our toolkit will help developers in GEC field to speed up their development on GEC tasks.

Our **main contributions** are as follows:

- We propose the first GEC-oriented toolkit, `UnifiedGEC`, which provides developers with a unified, extensible framework and allows them to implement models simply.
- We implement 5 GEC models and integrate 7 widely-used datasets and 3 mainstream GEC evaluators in our framework, including 2 *Seq2Edit* models, 3 *Seq2Seq* models, 2 Chinese datasets, 2 English datasets and 3 low-resource datasets, so that developers can run and evaluate these models through our framework easily.
- We design a data augmentation module and prompt module for low-resource tasks and research on LLMs.
- We conduct a comprehensive evaluation of integrated models and datasets, providing thorough conclusions and insights for developers in GEC field.

2 Related Works

In the perspectives of boosting text writing and language learning education, there are currently some toolkits or systems designed for different tasks and scenarios, such as OpenNMT (Klein et al., 2017) for machine translation tasks, TextFlint (Gui et al., 2021) for robustness text evaluation, Effidit (Shi et al., 2023) for writing assistant. Inspired by these works, we propose our toolkit `UnifiedGEC` for GEC tasks. To our knowledge, our `UnifiedGEC` is the first GEC-oriented toolkit that integrates GEC models of different architectures and datasets across various languages in a unified framework.

For GEC tasks, a lot of pre-trained models have been proposed and applied (Bryant et al., 2023). These models can be divided into two categories: *Seq2Seq* and *Seq2Edit*. *Seq2Seq* models treat GEC tasks as sequence generation tasks, directly generating tokens according to the context, such as Transformer (Vaswani et al., 2017), T5 (Xue et al., 2021) and SynGEC (Zhang et al., 2022b). Meanwhile, *Seq2Edit* models deal with GEC tasks in the form of sequence labeling tasks, and these models will predict labels of edits for the tokens, such as PIE (Awasthi et al., 2020) and GECToR (Omelianchuk et al., 2020). We have surveyed works in recent years and selected several representative models integrated in our framework. Moreover, there have been related works using data augmentation and LLMs in GEC tasks. We comprehensively integrate some augmentation methods and engineering prompts used in related work in our toolkit (Sottana et al., 2023; Song et al., 2023).

3 Framework Description

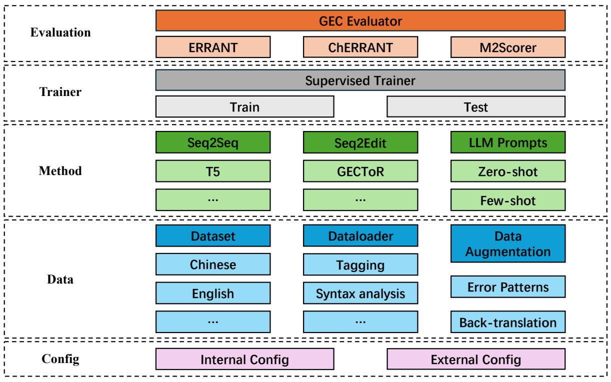

Figure 1: Framework of UnifiedGEC.

`UnifiedGEC` is a GEC-oriented toolkit based on PyTorch (Paszke et al., 2019), which integrates many models and datasets for GEC tasks. It allows developers to run models of different structures under a unified framework with only one single command. Meanwhile, developers are able to implement their own models through extensive and reusable modules integrated in our framework.

As depicted in Figure 1, our framework consists of five components: **Config** component, **Data** component, **Model** component, **Trainer** component, **Evaluation** component. Config component defines training parameters and records parameters of models to construct a complete configuration for the training process, which serves as the most basic part of our framework. Upon the configuration, data component processes data while model component allocates models for training or fine-tuning in the trainer component. Besides, we integrate several mainstream evaluators for GEC tasks in our evaluation component. Next, we will introduce each component in detail.

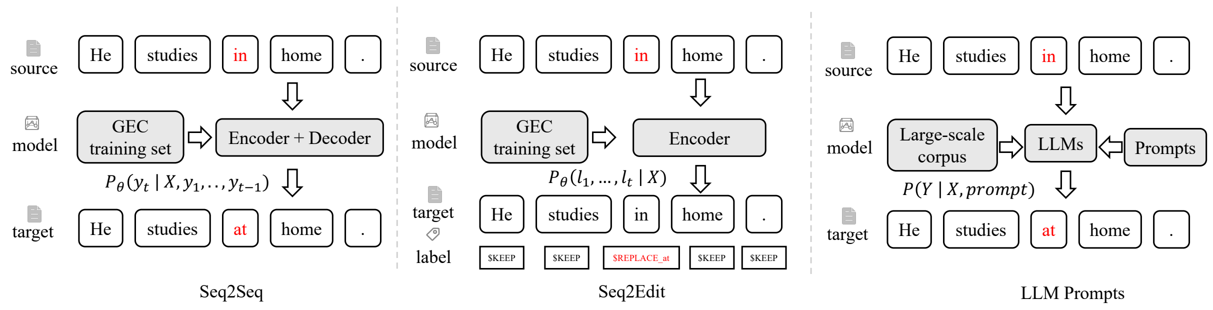

Figure 2: Illustration of different deployed method paradigms with examples. Noting that both *Seq2Seq* and *Seq2Edit* paradigms aim to fine-tune the models on the GEC training data set while the *LLM prompting* paradigm aims to conduct inference on an LLM.

3.1 Config Component

The config component is the preliminary component in which developers may customize the settings of datasets or models through related configuration files. There are two kinds of configuration files in our framework: *External Config* and *Internal Config*. The internal configuration file defines default configuration such as names of the model and dataset, learning rate and number of training epochs, while the external configuration file allows developers to modify some more detailed settings such as the parameters of a model.

3.2 Data Component

The data component plays a crucial role in our framework as it provides a unified approach for handling GEC datasets across diverse languages via various methods. As shown in Table 1, there are 7 GEC datasets integrated in our framework, including 2 Chinese datasets, 2 English datasets and 3 datasets in other languages. All the datasets deployed in our framework are open-sourced. Specifically, for MuCGEC[4] (Zhang et al., 2022a), we deploy its development set of as the test set in our framework because its official test set has not been published yet. Due to the different sources of datasets, unexpected variance my be involved for training. To bridge the gap, we integrate these preprocessing steps in our framework. This results in a standard and basic preprocessing pipeline, including *Dataset*, *Dataloader* and *Data Augmentation* modules.

Dataset module: Raw GEC data is initially read from a JASON file and converted into a *Dataset* class, which supports data splitting, noise removal and simple tokenization.

[4]https://github.com/HillZhang1999/MuCGEC/

DataLoader module: After reading data, *Dataloader* module takes charge of preparing the data into batches in the configured format. Specifically, *Dataloader* module converts data into tensors based on the requirement of different model architectures. For example, the dataloader converts data into tagged text for GECToR (Omelianchuk et al., 2020) model, and the dataloader extracts the syntactic information of the dataset as additional input for SynGEC (Zhang et al., 2022b). We also provide *Abstract Dataloader* so that developers may inherit the abstract class and implement their dataloader class.

Data Augmentation module: We integrate *Data Augmentation* module in our framework, which can be launched via a single command line, which supports to augment training data with limited raw data. There are two methods implemented in this module, *error patterns* (Zhao et al., 2019; Ehsan and Faili, 2013), which will directly inject errors to original data to generate more sentence pairs, and *back-translation*, which generates parallel corpus through a bridge language (Madnani et al., 2012; Zhou et al., 2019).

3.3 Model Component

The model component consists of a variety of GEC models, which can be flexibly called to perform fine-tuning or inference. In this component, we introduce a unified paradigm for implementation. Each model should be pre-defined in terms of their initialized parameters, loss definition, and propagation procedures for training as well as testing. Hence, developers could simply focus on the development of the GEC model, trying different combinations and architectures for solving GEC tasks. In addition, since the basic modules such as Transformer are reusable to implement a new model, our

	Module	Component
Data	FCE (Yannakoudakis et al., 2011)	English dataset
	CoNLL14 (Ng et al., 2014)	English dataset
	NLPCC18 (Zhao et al., 2018)	Chinese dataset
	MuCGEC (Zhang et al., 2022a)	Chinese dataset
	COWSL2H (Yamada et al., 2020)	Spanish dataset
	Falko-MERLIN (Boyd et al., 2014)	French dataset
	AKCES-GEC (Náplava and Straka, 2019)	Czech dataset
Model	Transformer (Vaswani et al., 2017)	Transformer Encoder + Transformer Decoder
	T5 (Xue et al., 2021)	Pre-trained Encoder + Pre-trained Decoder
	SynGEC (Zhang et al., 2022b)	syntax analysis, DepGCN + Transformer Decoder
	Lev-T (Gu et al., 2019)	Transformer Encoder + Transformer Decoder
	GECToR (Omelianchuk et al., 2020; Zhang et al., 2022a)	tagging, Pre-trained Encoder
Data Augmentation	Error patterns (Ehsan and Faili, 2013; Zhao et al., 2019)	–
	Back-translation (Madnani et al., 2012; Zhou et al., 2019)	Pre-trained T5 model
LLM Prompts	prompts (Fang et al., 2023)	Chinese/English prompts for zero-shot/few-shot

Table 1: Deployed datasets and methods in `UnifiedGEC`, associated with their basic components. Here, "Lev-T" is the abbreviation of "Levenshtein Transformer".

`UnifiedGEC` also encapsulates these basic modules in a folder such that it is convenient for developers to build their models by reusing these basic modules. According to the paradigms of the methods, we categorize the deployed methods as *Seq2Seq*, and *Seq2Edit* and *LLM Prompts* paradigms.

Seq2Seq paradigm: This line of methods follow the autoregressive principle as shown in Figure 2, where the correct sentence is generated token by token. In detail, we have implemented Transformer (Vaswani et al., 2017), (m)T5 (Xue et al., 2021) and SynGEC (Zhang et al., 2022b) as the representative models of this type due to their good perforamnce on GEC tasks.

Seq2Edit paradigm: This line of methods have non-autoregressive architectures as shown in Figure 2. Instead of predicting the correct token, they first identify the operation labels, including keeping, substitution and deletion, then the model predicts the operation for each token in parallel. We implement Transformer(Vaswani et al., 2017) and Levenshtein Transformer(Gu et al., 2019) as baseline models for Seq2Edit models.

LLM Prompts paradigm: Due to the impressive capability of the Large Language Models (LLMs) in general tasks, we include LLM-based methods for solving GEC tasks and a *Prompt* module is integrated in our model component. We provide prompts for directly generating correct sentences in English and Chinese. In-Context Learning (Dong et al., 2023) is also supported in our *Prompt* module. Models integrated in our framework are shown in Table 1.

Our framework also provides developers with a unified interface through which they can either load any models or write their own instructions to prompt LLMs to correct erroneous sentences. Developers can specify the backbone model and dataset they use and the number of examples for in-context learning through command lines, and then our framework will extract demonstrations from the specified dataset randomly. In this way, the developers are able to test the capabilities of LLMs on various GEC datasets.

3.4 Trainer Component

To facilitate a more standard training and inference pipeline, we introduce a *Supervised Trainer* as the trainer component in our framework, which is mainly designed for GEC tasks. This component controls the training procedure with some tunable parameters such as learning rate and number of epochs. Developers can simply adjust the configuration file to modify related training setups. Furthermore, developers can choose to either conduct a full training process from scratch or load a pre-trained checkpoint to perform inference directly, based on their needs.

Similair as other components, we also provide an *Abstract Trainer* so that developers can inherit the class and implement their own trainer if there is any.

3.5 Evaluation Component

In the evaluation component, we implement a *GEC Evaluator* which can switch between different evaluation metrics based on the language of the dataset. For Chinese datasets, we use ChERRANT (Zhang et al., 2022a), and for other languages, we use ERRANT (Bryant et al., 2017). Following the majority of prior studies, our `UnifiedGEC` toolkit calculates average precision, recall and $F_{0.5}$ as the final evaluation results, known as *Micro PRF*.

In addition, we still integrate traditional evaluation metrics, known as *Macro PRF*, including mentioned ChERRANT and ERRANT, and M2Scorer (Dahlmeier and Ng, 2012) for specific datasets such as FCE (Yannakoudakis et al., 2011). These diverse evaluation methods enable a more comprehensive comparison of GEC models.

4 Usage

With our framework, developers are allowed to run existing models on integrated datasets and add a new model or a new dataset in a customized manner. This section illustrates the detailed usage and workflow of `UnifiedGEC`.

4.1 Basic Usage

Developers can simply run our toolkit through the following command:

```
$ python run_gectoolkit.py -m model_name -d
    dataset_name
```

Then, *Config* class will load *internal config* and *external config* to construct a complete configuration file that includes necessary information such as the language of the dataset and hyper-parameters of the model. Based on the configuration file, the *Dataset* class as well as the *Dataloader* class will be initialized to process the data. Subsequently, our `UnifiedGEC` toolkit initializes the model specified by developers and loads the pre-trained checkpoint if there is any. Next, it initializes the evaluation module and the *Trainer* class will be built upon the configuration file. Once everything is ready, the training process starts.

To set the configuration in a fast way, developers are allowed to modify detailed configurations through command lines, for example:

```
$ python run_gectoolkit.py -m model_name -d
    dataset_name --learning_rate 1e-5
```

During the process of initialization, our framework will parse arguments in the command line and overwrite original ones in *external config*. All the parameters in configuration files are allowed to tune through command lines, such as learning rate in *internal config* and dropout possibility in *external config*.

To implement a new model, we provide abstract options for models, so developers are able to add customized model modules by inheriting the *Model* class and defining their own functions. It is worth noting that possible parameters of the model are suggested to be stored in a configuration file in *properties* directory, as well as other required files such as a vocabulary or a configuration for the tokenizer, which makes it easy for developers to modify and manage these configuration files with a little efforts.

To add a new dataset, developers can simply incorporate new JSON files of datasets into the framework and create a corresponding configuration file in *properties* directory. Then, they can either call the existing components or implement their own *Dataset* class to process specific datasets in different languages.

4.2 Extended Usage

In addition to the aforementioned basic usage, we also deployed some commonly used functions in `UnifiedGEC` for extended usage.

To augment limited training data, developers can use the specified data augmentation strategy in experiments, as the *Data Augmentation* module can collaborate with any GEC models:

```
$ python run_gectoolkit.py -m model_name -d
    dataset_name --augment translation
```

`UnifiedGEC` toolkit will automatically detect whether the file of augmented data exists. If it does not exist, our framework will execute the corresponding augmentation function specified by developer's command line, and generate new data files in the *dataset* directory. Then our framework will directly use augmented data instead of generating them again. Eventually, the training process will then be conducted on the augmented data.

To conduct LLM prompting, developers can launch the *Prompt* module in the same way:

```
$ python run_gectoolkit.py -m model_name -d
    dataset_name --use_llm --example_num 4
```

When flag *use_llm* is detected, our framework will use provided prompts in *Prompt* module for the LLM specified by developers. Developers are able to specify the LLM through *model_name* parameter, such as *Qwen/Qwen1.5-14B-chat*. The argument *example_num* indicates the number of in-context learning examples. When the value of this

Models	CoNLL14 (EN)						NLPCC18 (ZH)					
	full data			10% of data	w/ EP	w/ BT	full data			10% of data	w/ EP	w/ BT
	R	P	$F_{0.5}$	$F_{0.5}$	$F_{0.5}(\Delta)$	$F_{0.5}(\Delta)$	R	P	$F_{0.5}$	$F_{0.5}$	$F_{0.5}(\Delta)$	$F_{0.5}(\Delta)$
Levenshtein Transformer	12.6	13.5	13.3	9.5	6.4($\downarrow$ 3.1)	12.5($\uparrow$ 3.0)	8.5	12.6	10.7	6.0	4.9($\downarrow$ 1.1)	5.9($\downarrow$ 0.1)
GECToR	21.7	52.3	40.8	14.2	15.1($\uparrow$ 0.9)	16.7($\uparrow$ 2.5)	20.9	30.9	28.2	17.4	19.9($\uparrow$ 2.5)	19.4($\uparrow$ 2.0)
Transformer	15.5	24.1	21.7	12.6	14.5($\uparrow$ 1.9)	16.6($\uparrow$ 4.0)	20.8	22.3	22.0	9.5	9.9($\uparrow$ 0.4)	10.4($\uparrow$ 0.9)
T5-large	39.5	36.6	37.1	31.7	32.0($\uparrow$ 0.3)	32.2($\uparrow$ 0.5)	21.1	32.5	29.4	26.3	27.0($\uparrow$ 0.7)	21.1($\downarrow$ 5.2)
SynGEC	51.8	50.6	50.9	47.7	48.2($\uparrow$ 0.5)	47.7(−)	36.8	36.0	36.2	32.4	34.9($\uparrow$ 2.5)	34.6($\uparrow$ 2.2)
zero-shot + LLM	49.1	48.8	48.8	–	–	–	38.3	24.7	26.6	–	–	–
few-shot + LLM	50.2	50.4	50.4	–	–	–	39.8	24.8	26.8	–	–	–

Table 2: Results of GEC models on CoNLL14 and NLPCC18 datasets implemented via `UnifiedGEC`. Here, "EP" and "BT" are abbreviations of error patterns and back-translation augmentation methods, respectively. The top section includes the setups with full and partial training data. The bottom section includes the setups with zero/few-shot data.

parameter is not 0, prompts designed for in-context learning will be used.

5 Evaluation

To validate our `unifiedGEC` toolkit, we conduct numerous experiments to evaluate 5 models in our toolkit on 7 GEC datasets of different languages. Furthermore, we also conduct experiments to evaluate our *data augmentation* module and *prompt* module. To be consistent with the evaluation settings in the original papers, different evaluation metrics are employed across various datasets in our experiments. We use M^2Scorer (Dahlmeier and Ng, 2012) for FCE, CoNLL14, NLPCC18, while ChERRANT (Zhang et al., 2022a) is utilized on MuCGEC. For other languages, we employ ERRANT (Felice et al., 2016; Bryant et al., 2017).

The results are shown in Table 2[5]. As we can see, most of the models integrated in our framework perform comparably to levels demonstrated in the original papers. The performance of GECToR on CoNLL14 dataset is slightly lower than that in the original paper, as we did not conduct cold epoch training. The performance of Transformer may also be lower than that in other papers because we only implemented the basic greedy decoding strategy instead of beam searching. According to Table 2, it can be observed that models tend to have better performances on Chinese and English datasets than on datasets of other languages. We believe this is because the amount of data in other languages is relatively limited in the GEC field, and most of the models we used have not been fine-tuned on corresponding languages. As an exception, the T5 model integrated in our framework performs well on all datasets, as we implement a multilingual version in our framework.

For evaluation of *data augmentation* module, we extract 10% data from the original datasets to simulate low-resource setting, and then conduct tests on two data augmentation methods. It is evident that our proposed approaches are effective in improving the performance of models in most cases.

For *prompt* module, we evaluate the performance of zero-shot and few-shot prompts. In the few-shot experiment, we randomly select examples from the training set and try different sets of number of examples. For Chinese dataset, we choose Qwen1.5-14B-chat[6](Bai et al., 2023) as an example, while for English dataset, we use LLaMA2-7B-Chat[7](Touvron et al., 2023), and we demonstrate the best performance achieved in our evaluation. Due to the page limitation, we put all the experimental results on our GitHub page[8].

6 Conclusions

We propose a developer-friendly, modularized and GEC-oriented toolkit `UnifiedGEC` in this paper. In our `UnifiedGEC`, we provide developers with multiple extensive modules so that they can implement their own models easily. We also integrate models of different architectures and datasets across various languages, which allows developers to evaluate their models simply. These features enable our our framework to help developers handle GEC tasks easily. Furthermore, we conduct a comprehensive evaluation on integrated models, which provides developers in GEC field with thorough conclusion and insights.

[5]We put results of experiments on more datasets on GitHub page.

[6]`https://huggingface.co/Qwen/Qwen1.5-14B-Chat`

[7]`https://huggingface.co/meta-llama/Llama-2-7b-chat`

[8]`https://github.com/AnKate/UnifiedGEC`

In the future, we will continue improving our framework and adding more GEC-related models, datasets and modules to the toolkit.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments. This work is supported by the Young Scientists Project (No. 62206097) and the Funds for International Cooperation and Exchange of the National Natural Science Foundation of China (No. W2421085) .

References

Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, and Vihari Piratla. 2020. Parallel iterative edit models for local sequence transduction. *Preprint*, arXiv:1910.02893.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Adriane Boyd, Jirka Hana, Lionel Nicolas, Detmar Meurers, Katrin Wisniewski, Andrea Abel, Karin Schöne, Barbora Štindlová, and Chiara Vettori. 2014. The MERLIN corpus: Learner language and the CEFR. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 1281–1288, Reykjavik, Iceland. European Language Resources Association (ELRA).

Christopher Bryant, Mariano Felice, and Ted Briscoe. 2017. Automatic annotation and evaluation of error types for grammatical error correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 793–805, Vancouver, Canada. Association for Computational Linguistics.

Christopher Bryant, Zheng Yuan, Muhammad Reza Qorib, Hannan Cao, Hwee Tou Ng, and Ted Briscoe. 2023. Grammatical error correction: A survey of the state of the art. *Computational Linguistics*, 49(3):643–701.

Daniel Dahlmeier and Hwee Tou Ng. 2012. Better evaluation for grammatical error correction. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 568–572.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. 2023. A survey on in-context learning. *Preprint*, arXiv:2301.00234.

Nava Ehsan and Heshaam Faili. 2013. Grammatical and context-sensitive error correction using a statistical machine translation framework. *Software: Practice and Experience*, 43(2):187–206.

Tao Fang, Shu Yang, Kaixin Lan, Derek F. Wong, Jinpeng Hu, Lidia S. Chao, and Yue Zhang. 2023. Is chatgpt a highly fluent grammatical error correction system? a comprehensive evaluation. *Preprint*, arXiv:2304.01746.

Mariano Felice, Christopher Bryant, and Ted Briscoe. 2016. Automatic extraction of learner errors in ESL sentences using linguistically enhanced alignments. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 825–835, Osaka, Japan. The COLING 2016 Organizing Committee.

Roman Grundkiewicz, Christopher Bryant, and Mariano Felice. 2020. A crash course in automatic grammatical error correction. In *Proceedings of the 28th International Conference on Computational Linguistics: Tutorial Abstracts*, pages 33–38, Barcelona, Spain (Online). International Committee for Computational Linguistics.

Jiatao Gu, Changhan Wang, and Jake Zhao. 2019. Levenshtein transformer. *Preprint*, arXiv:1905.11006.

Tao Gui, Xiao Wang, Qi Zhang, Qin Liu, Yicheng Zou, Xin Zhou, Rui Zheng, Chong Zhang, Qinzhuo Wu, Jiacheng Ye, Zexiong Pang, Yongxin Zhang, Zhengyan Li, Ruotian Ma, Zichu Fei, Ruijian Cai, Jun Zhao, Xingwu Hu, Zhiheng Yan, Yiding Tan, Yuan Hu, Qiyuan Bian, Zhihua Liu, Bolin Zhu, Shan Qin, Xiaoyu Xing, Jinlan Fu, Yue Zhang, Minlong Peng, Xiaoqing Zheng, Yaqian Zhou, Zhongyu Wei, Xipeng Qiu, and Xuanjing Huang. 2021. Textflint: Unified multilingual robustness evaluation toolkit for natural language processing. *Preprint*, arXiv:2103.11441.

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

Kate M Knill, Mark JF Gales, PP Manakul, and AP Caines. 2019. Automatic grammatical error detection of non-native spoken learner english. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8127–8131. IEEE.

Nitin Madnani, Joel Tetreault, and Martin Chodorow. 2012. Exploring grammatical error correction with not-so-crummy machine translation. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*, pages 44–53.

Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. 2014. The CoNLL-2014 shared task on grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14, Baltimore, Maryland. Association for Computational Linguistics.

Jakub Náplava and Milan Straka. 2019. Grammatical error correction in low-resource scenarios. *Preprint*, arXiv:1910.00353.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. GECToR – grammatical error correction: Tag, not rewrite. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 163–170, Seattle, WA, USA → Online. Association for Computational Linguistics.

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, Zach 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. *Preprint*, arXiv:1912.01703.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer. *Preprint*, arXiv:1910.10683.

Shuming Shi, Enbo Zhao, Wei Bi, Deng Cai, Leyang Cui, Xinting Huang, Haiyun Jiang, Duyu Tang, Kaiqiang Song, Longyue Wang, et al. 2023. Effidit: An assistant for improving writing efficiency. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 508–515.

Yixiao Song, Kalpesh Krishna, Rajesh Bhatt, Kevin Gimpel, and Mohit Iyyer. 2023. Gee! grammar error explanation with large language models. *Preprint*, arXiv:2311.09517.

Andrea Sottana, Bin Liang, Kai Zou, and Zheng Yuan. 2023. Evaluation metrics in the era of gpt-4: Reliably evaluating large language models on sequence to sequence tasks. *Preprint*, arXiv:2310.13800.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

Aaron Yamada, Sam Davidson, Paloma Fernández-Mira, Agustina Carando, Kenji Sagae, and Claudia Sánchez-Gutiérrez. 2020. Cows-l2h: A corpus of spanish learner writing. *Research in Corpus Linguistics*, 8(1):17–32.

Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. 2011. A new dataset and method for automatically grading ESOL texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 180–189, Portland, Oregon, USA. Association for Computational Linguistics.

Yue Zhang, Zhenghua Li, Zuyi Bao, Jiacheng Li, Bo Zhang, Chen Li, Fei Huang, and Min Zhang. 2022a. MuCGEC: a multi-reference multi-source evaluation dataset for Chinese grammatical error correction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3118–3130, Seattle, United States. Association for Computational Linguistics.

Yue Zhang, Bo Zhang, Zhenghua Li, Zuyi Bao, Chen Li, and Min Zhang. 2022b. Syngec: Syntax-enhanced grammatical error correction with a tailored gec-oriented parser. *Preprint*, arXiv:2210.12484.

Wei Zhao, Liang Wang, Kewei Shen, Ruoyu Jia, and Jingming Liu. 2019. Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and*

Short Papers), pages 156–165, Minneapolis, Minnesota. Association for Computational Linguistics.

Yuanyuan Zhao, Nan Jiang, Weiwei Sun, and Xiaojun Wan. 2018. Overview of the nlpcc 2018 shared task: Grammatical error correction. In *Natural Language Processing and Chinese Computing*, pages 439–445, Cham. Springer International Publishing.

Wangchunshu Zhou, Tao Ge, Chang Mu, Ke Xu, Furu Wei, and Ming Zhou. 2019. Improving grammatical error correction with machine translation pairs. *arXiv preprint arXiv:1911.02825*.