An Exploratory Study on Model Compression for Text-to-SQL

Shuo Sun[1], Yuze Gao,[1] Yuchen Zhang[1,2], Jian Su[1], Bin Chen[1]
Yingzhan Lin[3], Shuqi Sun[3]
[1]Institute for Infocomm Research (I^{2}R), A*STAR, Singapore
[2]CNRS@CREATE LTD, Singapore
[3]Baidu Inc., China
[1]{Sun_Shuo,Gao_Yuze,Zhang_Yuchen,sujian,bchen}@i2r.a-star.edu.sg,
[2]{linyingzhan01,sunshuqi01}@baidu.com

Abstract

Text-to-SQL translates user queries into SQL statements that can retrieve relevant answers from relational databases. Recent approaches to Text-to-SQL rely on pre-trained language models that are computationally expensive and technically challenging to deploy in real-world applications that require real-time or on-device processing capabilities. In this paper, we perform a focused study on the feasibility of applying recent model compression techniques to sketch-based and sequence-to-sequence Text-to-SQL models. Our results reveal that sketch-based Text-to-SQL models generally have higher inference efficiency and respond better to model compression than sequence-to-sequence models, making them ideal for real-world deployments, especially in use cases with simple SQL statements.

1 Introduction

Text-to-SQL is an important task that has been gaining the attention of researchers over the years. Formally, given a query q and a relational database D, the goal of Text-to-SQL is to build a model f such that $s = f(q, D \mid \theta)$ where θ is a vector of model parameters and s is a predicted SQL statement which we can use to retrieve the answer to q from D.

Text-to-SQL has many potential applications that can improve our standard of living. For example, medical chatbots can convert user queries into SQL statements and then use them to retrieve relevant information from medical knowledge bases. Industry can leverage Text-to-SQL tools to help employees shorten the time needed to write complex SQL queries, thereby improving overall work productivity.

The recent emergence of complex Text-to-SQL datasets containing complicated SQL and cross-table setup has driven researchers to develop huge models that encode various complex relationships between table schema and query with large pre-trained language models such as BERT (Devlin et al., 2019) and T5 (Raffel et al., 2020). These models are usually sequence-to-sequence models that generate SQL statements sequentially or sketch-based models that use classifiers to fill in the slots of SQL templates.

However, despite achieving state-of-the-art performances on benchmark datasets, such models are usually both memory and computationally expensive, making it technically challenging to deploy them in memory-constrained real-world applications that require low inference latency. Therefore, to deploy state-of-the-art Text-to-SQL models in real-world production environments, we must drastically improve the inference time and reduce the number of parameters in these models.

We turn to the field of model compression (Cheng et al., 2017) for solutions that can speed up inference without significantly hurting model performance. Formally, the goal of model compression is to reduce f to a smaller model f' such that $s' = f'(q, D \mid \theta')$. Ideally, we want s' to be the same as s and $\dim(\theta')$ to be much smaller than $\dim(\theta)$.

In this paper, we thoroughly examine the feasibility of using model compression techniques to build faster and more accurate Text-to-SQL models that we can successfully deploy in the real world. For this, we carefully apply a few model compression methods to representative sequence-to-sequence or sketch-based Text-to-SQL models on three datasets: WikiSQL, Spider, and TableQA. The main findings of this paper are: (i) sketch-based models generally respond well to model compression techniques, while sequence-to-sequence models show mixed results, (ii) we observe better speed improvements in Sketch-based models as their slot-filling components are much faster than the decoding components of sequence-to-sequence models. (iii) model compression techniques work poorly on state-of-

Findings of the Association for Computational Linguistics: ACL 2023, pages 11647–11654
July 9-14, 2023

the-art Text-to-SQL models built on pre-trained encoder-decoder language models such as T5.

We hope our findings can empower practitioners to make more informed decisions when selecting Text-to-SQL models and compressing them appropriately for real-world deployments.

2 Methodology

2.1 Datasets

Name	Lang	Difficulty	#Questions
WikiSQL	En	Simple	80,654
Spider	En	Complex	9,693
TableQA	Zh	Simple	64,891

Table 1: Statistics of Text-to-SQL datasets

We conduct model compression experiments on several datasets as shown in Table 1:

WikiSQL (Zhong et al., 2017) was extracted from 24,241 Wikipedia tables, with questions manually paraphrased by human annotators.

Spider (Yu et al., 2018) is a complex dataset containing 9,693 question-SQL pairs. The accompanying schemas are annotated by college students, with over 200 databases covering 138 different domains.

TableQA (Sun et al., 2020) is a Chinese text-to-SQL dataset containing 64,891 question-SQL pairs over 6000 tables extracted from online documents such as financial reports or spreadsheets.

Difficulty of datasets WikiSQL and TableQA are considered *simple* datasets because they only contain SQL queries covering the SELECT and WHERE clauses, and each database has only one single table. Contrarily, Spider contains large samples of *complex* SQL instances that connect multiple tables with primary and foreign keys with more advanced clauses such as nested queries, JOIN ON, and ORDER/GROUP BY.

2.2 Baseline Models

Recent deep neural Text-to-SQL models can be broadly classified under two categories: *sequence-to-sequence models* and *sketch-based (also known as slot-filling) models*.

2.2.1 Sequence-to-sequence models

Sequence-to-sequence models are generally made up of an encoder component that converts user query inputs together with database information into a hidden vector and a decoder component that generates SQL statements based on the output hidden vectors from the encoder.

BRIDGE (Lin et al., 2020) encodes input questions and table schema with BERT and LSTM and generates SQL predictions with a pointer-generator decoder (See et al., 2017) supported by a schema-consistency driven search space pruning strategy.

RAT-SQL (Wang et al., 2020a) also encodes input instances with BERT but generates SQL as an abstract syntax tree (AST) with a tree-structured decoder (Yin and Neubig, 2017). It also incorporates a relation-aware self-attention mechanism that further improves schema-linking, schema-encoding, and representation of the encoder.

PICARD (Scholak et al., 2021) is a state-of-the-art algorithm that directly fine-tunes a pre-trained encoder-decoder language model T5 (Raffel et al., 2020) on Text-to-SQL data, and then constrain the decoder to output valid SQL by integrating an incremental parsing strategy to the beam search process.

2.2.2 Sketch-based model

Sketch-based methods also encode user inputs into vectors but only need to fill in slots in SQL sketches rather than generating full SQL statements. Each SQL sketch is a template SQL statement with placeholder slots and the goal of sketch-based models is to predict the best item to go into each slot.

NL2SQL-RULE (Guo and Gao, 2019) is a standard sketch-based model which uses BERT and LSTM to encode input query and database information and predict outputs in slots of SQL sketches.

2.3 Compression Techniques

We follow Sun et al. (2021) and experiment with the following model compression techniques in this study:

Layer Pruning (Sajjad et al., 2022) is a simple yet effective strategy that discards a certain number of layers from transformer-based language models before fine-tuning the pruned models on downstream tasks. We apply the top-layer pruning strategy which deletes the top N encoder or decoder layers before the start of any training.

Knowledge Distillation (Hinton et al., 2015) is a method that compresses deep neural network models by distilling useful knowledge from a larger model (teacher) to a smaller model (student). We follow Jiao et al. (2020) and distill smaller language models from larger ones such as BERT-large, before fine-tuning Text-to-SQL models on those

distilled models. For WikiSQL and Spider, we experiment with the distilled English language models from MiniLM[1] (Wang et al., 2020b), while for TableQA, we use the Chinese TinyBERT models[2].

Token Pruning For PICARD model, We also apply token pruning (Goyal et al., 2020; Kim et al., 2022), which is a different pruning strategy that gradually removes redundant token encodings from the outputs of each encoder layer before feeding the reduced number of tokens to the next encoder layer. We follow Goyal et al. (2020) and implement an attention scoring mechanisms which weights the significance of each token by the sum of attention weights it gets from other tokens. The tokens with the lowest significance scores (based on predetermined thresholds) for each encoder layer are dropped.

2.4 Evaluation Metrics

We evaluate our experiment results using *Exact set match* (ESM) (Yu et al., 2018). ESM decomposes every pair of predicted and gold SQL queries into sets clauses and then computes the percentage of exact set matches over all pairs (Zhong et al., 2020).

3 Experiment Setup

In most cases, we follow the recommended configurations in corresponding papers. We may adjust the batch sizes and learning rates slightly to fit the experiments on our hardware. We train our models on servers with either NVIDIA GV100 GPU (32GB) or RTX A6000 (45GB) but calculate inference speeds by running models on only CPUs with batch size set to one, which better mimics the situations in the real world. For all datasets, we use their dev sets as the test sets and create new train-dev sets in the ratio of 4 to 1 from the original train set. We early stop our models based on the ESM scores on dev sets and report average test set ESM scores over 5 different runs. Other than PICARD, we use BERT-large for all English datasets and RoBERTa-Zh (Cui et al., 2020) for TableQA.

3.1 Results and Recommendations

3.1.1 Simple datasets

WikiSQL As shown in Figure 1, both layer pruning and knowledge distillation work pretty well for WikiSQL. For example, we can remove 50% of the encoder layers from BRIDGE, while only taking a penalty of only 0.82% drop in Exact Set match (ESM). When only keeping the bottom 6 encoder layers, NL2SQL-RULE can still perform at 0.834 ESM, a 3.65% drop from the original unpruned model. For knowledge distillation, we fine-tuned BRIDGE on two versions of MiniLM (Wang et al., 2020b): L6xH768 and L6xH384. Results show that BRIDGE trained on the MiniLM language models performs slightly worse than the layer pruning method with similar number of layers. However, this is acceptable given the hidden sizes of the MiniLM models are 384 and 768, which are smaller than the hidden size of 1024 for BERT-large.

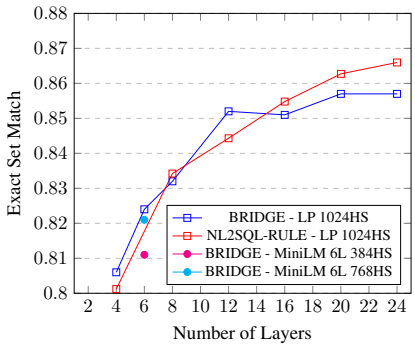

Figure 1: Compression Results on WikiSQL

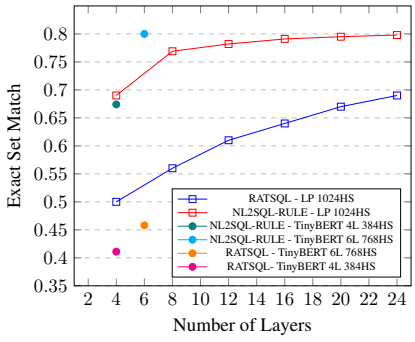

Figure 2: Compression Results on TableQA

TableQA We notice several differences in results between WikiSQL and TableQA. First, the performances of RATSQL on TableQA are significantly lower than those of NL2SQL-RULE. For example, unpruned NL2SQL-RULE achieves an ESM of 0.8 but unpruned RATSQL only achieves 0.69 despite our best efforts. Second, we observe more significant drops in performances when applying layer pruning and knowledge distillation to RATSQL than NL2SQL-RULE. For example, we observe only a 3.63% drop in ESM dropping the first 16

[1] https://github.com/microsoft/unilm/tree/master/minilm

[2] https://github.com/huawei-noah/Pretrained-Language-Model/tree/master/TinyBERT

encoder layers of NL2SQL-RULE but notice an 18.8% drop in the performance of RATSQL with the same configurations. Last but not least, models trained on distilled language models perform slightly worse than the layer pruned models due to their smaller hidden sizes except for NL2SQL-RULE on TinyBERT with 6 layers and 768, which achieves an ESM of 0.80, even higher than that of the unpruned NL2SQL-RULE.

Recommendation: We recommend using slot-filling models when building applications that only deal with simple queries. These models not only perform comparably or even better than sequence-to-sequence models, but also respond better to recent model compression techniques.

3.2 Complex dataset

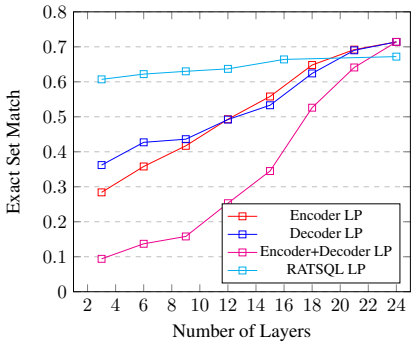

Figure 3: Compression Results on Spider

Spider As PICARD was trained on a 3 billion parameters pre-trained language model with an encoder and a decoder of similar size, we show three sets of results by applying layer pruning on 1) the encoder, 2) the decoder, and 3) both the encoder and decoder.

As seen in Figure 3, the layer pruning strategy does not work as well on PICARD. At around six layers, PICARD loses around 49.9% and 40.3% of its original performance for encoder-only and decoder-only pruning settings respectively. For the encoder+decoder pruning strategy, we observe similar levels of performance when discarding the same number of transformer layers as the other two configurations. For example, dropping 3 layers each from the encoder and decoder gets us 0.641 ESM, compared to 0.624 when dropping 6 decoder layers and 0.648 when dropping 6 encoder layers. On the other hand, RATSQL demonstrates better compression results on Spider, maintaining 92.6% of original performance while keeping on six encoder layers, contrary to the results on TableQA.

Token pruning We follow the implementation of Goyal et al. (2020) and apply token pruning to PICARD. We plot the ESM performance of a token-pruned model against the number of retained tokens in Figure 4. As seen in the plots, although we can remove an average of 286 tokens from the top six encoder layers, we are only able to discard an average of 41 tokens from the bottom six layers. For example, we see a sharp drop in ESM performance by just pruning around 40 tokens from the 3rd encoder layer. Similarly, we also observe steady drop in ESM performance when pruning more than 100 tokens from encoder layers 15 and 18. Our final model achieves an ESM of 0.527 (26.3% drop in performance) while only seeing a 5.2% improvement in inference speed when applying token pruning to the encoder of T5. As we cannot significantly prune the number of tokens in each encoder layer without severely hurting model performance, we conclude token pruning is also not effective on the PICARD model.

Recommendation: Our results suggest that both layer and token pruning are not effective on PICARD and we would get better compression performances on sequence-to-sequence models like RATSQL, which has a much bigger encoder than decoder in terms of model size.

3.3 Discussion

The main difference between recent sequence-to-sequence and sketch-based models is related to how we generate the SQL statements. Compared to the lightweight slot-filling classifiers in sketch-based models, recent sequence-to-sequence model decoders rely heavily on grammar-guided decoding processes which requires navigating through a huge search space and requires an even longer inference time than the encoders. For example, 76.62% and 87.14% of the inference time are spent in the decoding step for BRIDGE and RATSQL, while most of the inference time in NL2SQL-RULE is spent on the encoder. Considering the speed, compression effectiveness, and performance, sketch-based models would be better choices if we get similar performances on benchmark datasets.

4 Conclusion

This paper investigates whether we can use model compression to improve the inference efficiency of recent Text-to-SQL models that rely heavily on large pre-trained language models. Our results

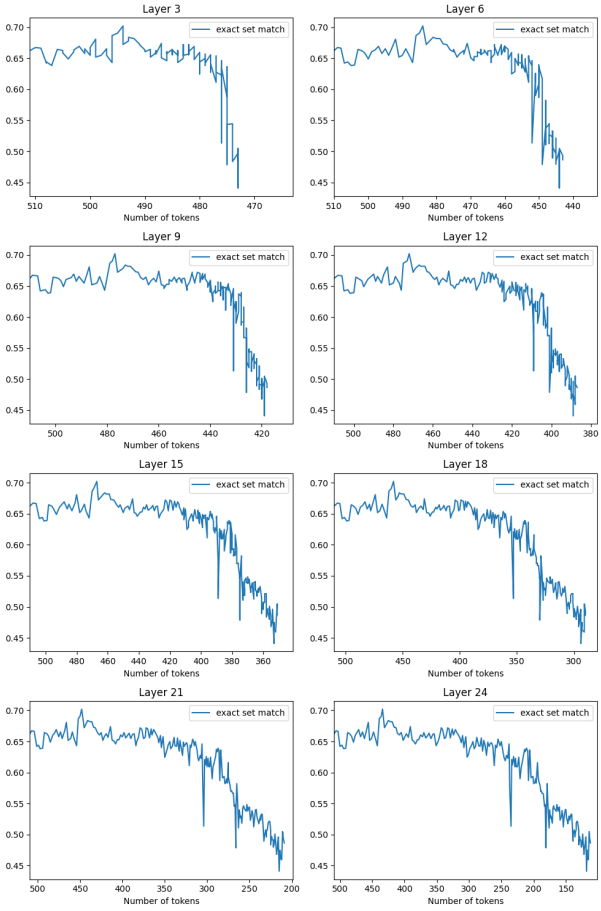

Figure 4: ESM score against number of retained tokens for PICARD on spider dataset

show that on simple Text-to-SQL datasets, we can deploy simple strategies such as layer pruning to obtain a 5-6x speedup without significantly hurting model performances. We also observe that sketch-based models generally respond better to model compression than sequence-to-sequence models. However, we are not able to effectively compress PICARD on the spider dataset and we would tackle this problem as a future work.

Limitations

There are several limitations to this paper. First, due to time and space constraints, we are unable to experiment with other interesting model compression techniques such as neural architecture search and quantization. We also have to select only a small subset of baseline Text-to-SQL models to represent the performances on each of the datasets. We are also aware of the existence of RYANSQL (Choi et al., 2021), a sketch-based model for the Spider dataset. However, we are not able to reproduce the baseline results to the best of our efforts and have to exclude them from our analysis. Therefore, it is important to be aware of these potential limitations and biases when using our results for real-world deployments.

Acknowledgments

This research is partially supported by the programme DesCartes funded by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme.

References

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv e-prints*, pages arXiv–1710.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2021. RYANSQL: Recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases. *Computational Linguistics*, 47(2):309–332.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Shijin Wang, and Guoping Hu. 2020. Revisiting pre-trained models for Chinese natural language processing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 657–668, Online. Association for Computational Linguistics.

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

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pages 3690–3699. PMLR.

Tong Guo and Huilin Gao. 2019. Content enhanced bert-based text-to-sql generation. *arXiv preprint arXiv:1910.07179*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. Tinybert: Distilling bert for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174.

Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. 2022. Learned token pruning for transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 784–794, New York, NY, USA. Association for Computing Machinery.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2022. On the effect of dropping layers of pre-trained transformer models. *Comput. Speech Lang.*, 77(C).

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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 (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Ningyuan Sun, Xuefeng Yang, and Yunfeng Liu. 2020. Tableqa: a large-scale chinese text-to-sql dataset for table-aware sql generation. *arXiv preprint arXiv:2006.06434*.

Shuo Sun, Ahmed El-Kishky, Vishrav Chaudhary, James Cross, Lucia Specia, and Francisco Guzmán. 2021. Classification-based quality estimation: Small and efficient models for real-world applications. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5865–5875, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020a. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020b. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-SQL with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 396–411, Online. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
After the conclusion

☑ A2. Did you discuss any potential risks of your work?
In the limitations section

☑ A3. Do the abstract and introduction summarize the paper's main claims?
In the abstract

☑ A4. Have you used AI writing assistants when working on this paper?
Grammarly

B ☐ Did you use or create scientific artifacts?

Not applicable. Left blank.

☐ B1. Did you cite the creators of artifacts you used?
Not applicable. Left blank.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Not applicable. Left blank.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Not applicable. Left blank.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Not applicable. Left blank.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Not applicable. Left blank.

C ☑ Did you run computational experiments?

section 3

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
section 3

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
section 3

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
section 3

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
section 3

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Not applicable. Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Not applicable. Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Not applicable. Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Not applicable. Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Not applicable. Left blank.