Evolved Transformer for Conditional Text Generation

Yuri Sala
Politecnico di Torino
yuri.sala@studenti.polito.it

Giovanni Tangredi
Politecnico di Torino
s276086@studenti.polito.it

Luca Viscanti
Politecnico di Torino
luca.viscanti@studenti.polito.it

Abstract

This work is about analyzing and improving a conditional transformer language model based on a previous work, ***Conditional Transformer Language for Conditional Text Generation (CTRL)****. We implemented the model and fine-tuned it with Common Objects in Context (COCO) dataset properly adapted for our purpose, then we performed some studies on the model structure and tried to improve the results and the performances. To achieve these goals, we substitute the original Transformer with new Evolved Transformer.*

1. Introduction

In the previous years, the text generation has been based on the Transformer architecture [7] that is considered the state of the art for this purpose. The CTRL model [3] is also based on transformer, and uses it for conditional text generation. The focus of this task is to diversifying the content of this generation, creating different sentences influenced by external conditions, such as the context, the topic or the emotion. The solution proposed by CTRL introduces the use of control codes, that allow to predict which words are more suitable for context and also to drive the structure of the generated text, for example it can be applied in goal-oriented conversation and question and answer.

CTRL is a conditional language model that is always conditioned on a control code c and learns the distribution $p(x|c)$, as shown in the formula (2), instead of normal language learning modeling that learn a distribution conditioned only by previous words $p(x)$, see the formula (1). The conditional distribution can still be decomposed using the chain rule of probability, and trained with a loss that take control code into account, as shown in the formula (3).

$$p(x) = \prod_{i=1}^{n} p(x_i|x_{<i}) \tag{1}$$

$$p(x|c) = \prod_{i=1}^{n} p(x_i|x_{<i}, c) \tag{2}$$

$$\mathcal{L}(D) = -\sum_{k=1}^{|D|} \log p_0(x_i^k|x_{<i}^k, c^k) \tag{3}$$

CTRL learn the distribution by training on a set of raw sentences where usually the control codes are at the start of the sentence, for other structure, such as Q&A, more then one control code could be required, for more detailed examples rely on the paper of Keskar *et al*. [3].

In our work, we implemented a reduced version of CTRL using the Huggingface Transformers interface [8], we also add some new control codes using the Common Objects in Context (COCO) dataset and evaluated the generated sentences with BLEU [4] metrics relying on *Texygen* of Zhu *et al*. [9]. After this, we proposed a modified version of CTRL using Evolved Transoformer, following the idea of So, Liang and Le [6] that should improve the generation phase especially for small size models. All the code for implementation is available at the following repository on GitHub. https://github.com/giovannitangredi/MLProject

2. Implementation details

2.1. Dataset

Common Objects in Context (COCO) is a large-scale object detection, segmentation and captioning dataset. The whole dataset is divided by categories grouped in supercategories, for our purpose we only used annotations from images and we considered supercategories as control codes.

This lead us to add 12 new control codes: ***Person, Vehicle, Indoor, Outdoor, Appliance, Kitchen, Electronic, Furniture, Food, Accessory, Animal, Sports***. Each sentence of the dataset is composed by the starting word that is the control code (supercategory to which the annotation belongs), followed by the relative annotation. This approach allowed us to train our model with more than a million of sentences.

2.2. Transformer

The Transformer [7] is a neural recurrent sequence transduction model based on an encoder-decoder structure. The overall model is composed of a stack of 12 layers: one encoder made of 6 layers and one decoder made of 6 layers, as shown on Figure [1].

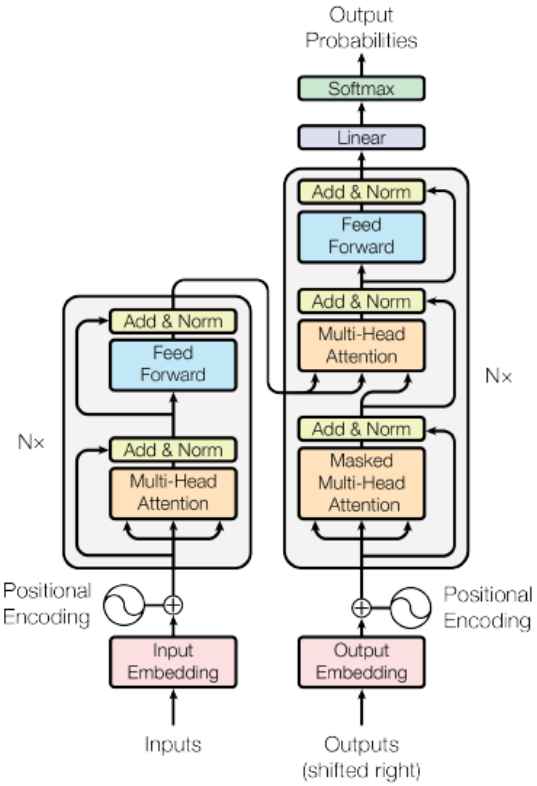

Figure 1: The Transformer - model architecture.

The encoder is composed by 6 identical layers, and has two sub-layer. The first is a multi-head self-attention mechanism, and the second is a position-wise fully connected feed forward network.

CTRL model only uses the encoder layer, in particular is made of 8 consecutive encoder block, that leads up to 48 total layers.

Before the data are sent to the encoder layers, both models use a positional encoding based on sinusoidal function. After the data are processed by all the encoding layers, a softmax function is applied to get the final probability distribution over the all dictionary.

2.3. Evolved transformer

Although the Transformer architecture has reached very good results on the text generation task, nowadays the research for a better model is not closed yet. Different approaches are applied to find the best improvement, one of them is the Neural Architecture Search (NAS) that is widely used cause is very effective. The work of David R.So, Chen Liang et Quoc V.Le [6] is focused on finding a transformer-based architecture such that can guarantee better performance than the already implemented model. They applied *tournament selection* [2] as Real *et al*. did [5], warm start it with the Transformer and eventually they ended up with an evolved structure that they called 'Evolved Transformer'.

At a big model size, the Evolved Transformer establishes a new state-of-the-art BLEU score but it is also effective at smaller sizes, achieving the same quality as the original "big" Transformer (same results but with less parameters).

The two architectures are quite similar to each other, both composed by stacks of encoder and decoder layers. CTRL model, and also us on our work, make use of the encoder part only so we focus our description on that and his differences with respect to the base version. The evolved encoder block is made up by 2 transformer layers such that the first one has relevant variations, instead the latter portion is identical to the old one. The main differences in the upper part, as visible on Figure [2], are the replacement of the first multi-head attention layer with a Gated Linear Unit (GLU) [1] and the modified feed-forward network. Especially noticeable in the feed-forward part, are the two branches using two different convolutions that are then combined into an another wide depth-wise separable convolution. Our proposal is to employ that evolved transformer block onto the CTRL model.

2.4. Texygen

Texygen [9] is a benchmarking platform used for text generation models. It has some predefined models that can be used for obtain metrics, but it also have a complete set of metrics that can be directly computed. For our work we mostly used the source code for getting inspiration about BLEU, SELF-BLEU and POS-BLEU metrics, in particular Texygen has a complete implementation of BLEU and SELF-BLEU. Then we adapted the Texygen BLEU code for the purpose to calculate POS-BLEU metrics. All the Texygen source code can be found at https://github.com/geek-ai/Texygen/tree/master/utils/metrics.

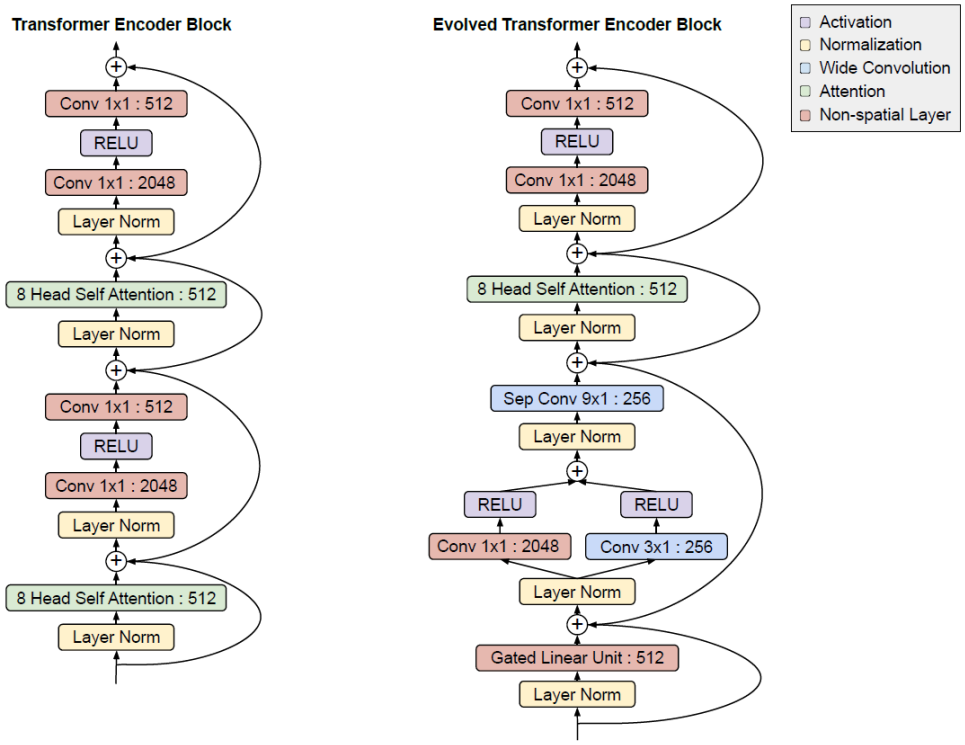

Figure 2: Differences between Transformer and Evolved Transformer architecture cells.

2.4.1 BLEU

BLEU (*Bilingual Evaluation Understudy Score*) [4] is one of the most used metric for evaluating a generated sentence given a set of references sentences. Specifically BLEU counts the number of matching N-grams, a set of N words from the candidate, and take the best result. N-gram scores can be individual or cumulative.

- **Individual:** this score is the evaluation of just matching N-grams of a specific order, such as single words (1-gram) or word pairs (2-gram or bigram).

- **Cumulative:** this scores is the evaluation of all the individual gram scores for 1 to N and weighting them by calculating the weighted geometric mean with given weights.

The metrics we calculated are based on the cumulative N-gram score in particular we refer it in the paper as *Metric-N* and the weights used are $1/N$ for each individual gram, for example in BLEU-3 we use the BLEU metric with 3-gram cumulative score.

2.4.2 SELF-BLEU

SELF-BLEU is a variation of BLEU when the references are also the set generated sentences (non including the candidate one) so that the diversity of the generated text can be measured.

2.4.3 POS-BLEU

POS-BLEU is also a variation when instead of normal text we use the part-of-speech tags for the evaluation of the BLEU metric, for our work we use the universal pos-tags available on NLTK. All the NLTK documentation is available at : https://www.nltk.org/api/nltk.html.

3. Our approach

First of all we tried to replicate the work of Keskar *et al.* [3] using the Huggingface Tranformers interface [8]. We decided to downsize the model to only 10 encoder layers due to memory usage and time spent. We converted a pre-trained checkpoint from Saleforce repository, available at the following link https://github.com/salesforce/ctrl and we applied it to our new model. Then, starting from that, we perform a fine-tuning over the

COCO annotation dataset, explain in detail in the following paragraph 3.1.

After having evaluated some metrics on this intermediate step, as shown in the paragraph 3.2, we looked for some variations that we could apply onto the CTRL architecture.

Our attention was addressed to four different parts:

- the positional encoding
- the encoder layers
- the loss function
- the final sampling

Searching on the web we found out that the positional encoding and the sampling were already implemented in the best way. Regarding the loss function, the Huggingface interface uses an internal function (a Cross Entropy Loss) that is easily accessible from the model outputs so we left it as it is. Eventually our focus fell down on the main components of the network: the encoder layers. The main idea was to improve the performance of the single layer and we ended up using the evolved transformer encoder block, see details in the paragraph 3.3.

This is a fast overall review on our approach that is better described in the next sections.

3.1. Fine-tuning

Fine-tuning is one of the methods to improve the performance of the neural network. It is a technique that takes weights of a neural network previously trained and use it as initialization for the new model being trained on new data from the same domain. In our case, we already had available the model trained by Keskar *et al*. [3] using raw text that is typically collected for training large language models, associated with control codes like *Horror, Books, Wikipedia* and more. We decided to create new control codes as mention above using the COCO dataset, so we performed the training using the huggingface library thanks to the work of Wolf *et al*. [8] with a lot of new sentences for each new control code. We divided the total amount of sentences in 12 different files, and in each file there were random sentences that belong to different control codes. Since there were so much sentences (over 1 million), we decided to train just for one epoch.

Here we report the detailed settings used for the training: the tokenizer for input sentences is the pre-trained one downloaded from Huggingface library (we added a special padding tokens '$\sim$' needed when the input sentences have different length); we trained with a batch size of 15; the model has some dropout layer with probability 0.2; as suggested on Saleforce script we used the Adagrad Optimizer with learning rate = 0.01.

3.2. Metrics

We decided to use the approach also used by Texygen [9] to calculate all the metrics needed. The idea was to use the cumulative way with N-grams value in the range from 1 to 4 which are the the most used range when calculate BLEU and similar scores. As references we extracted 14330 annotations from the COCO validation set, then to obtain the candidates we generated 1000 random prompts.

To score the metrics we decided to calculate the sentence metric from the candidates and then calculate the arithmetic mean, so we obtained the average score for the given set of generated sentences. We repeated this operation for each N-gram and each metric as shown it the following formula (4).

$$\text{Metric} = \sum_{i=1}^{N}(M_i)/N \tag{4}$$

Where M_i is the score for a single candidate and N the total number of candidates.

Obtain the SELF-BLEU scores is simple because we just need to iterate to all the candidates and for each candidate we calculate the BLEU score with the rest of other candidates as references.

For the POS-BLEU we decided to use the Universal Part-of-Speech Tagset given by NLTK because is the most viable and intuitive for English. In case another languages is used the results may be inconsistent and even errors could arise. We converted both references and candidates to POS tokens, than we run the same function used for calculate BLEU scores.

For all the details about implementation consult the code on the github: https://github.com/giovannitangredi/MLProject/blob/main/Metrics.py.

3.3. Evolved Transformer

After having fine-tuned the pre-trained model and having seen that it get some good results, we switched our attention to improve the model architecture. As previously described we looked for different ways to perform a variation that could lead to better outputs.

CTRL network is based on a stack of encoder layers (48) that are derived from the Transformer encoder block. Our idea was to replace these layers with the Evolved Transformer ones because in the work of So, Liang and Le [6] they showed that this new implementation can perform better result, especially on BLEU metric. Following their work we tried to implement a new network starting from the Huggingface classes. For the implementation we left unchanged the overall structure that is based on a "pretrainedModel" class that is inherited by a "CTRLModel" class. Our modifications was only on the encoder class that we tested in two

different ways.

In the first implementation we tried to replicate as much as possible the real Evolved Transformer block with their own classes for multi-head attention layers, scaled dot-product attention and point-wise feed forward network. Indeed that approach lead us to have a very high weight network that uses a lot of GPU memory and takes far more time to be trained than the previous one.

So, to avoid such a bad behaviour, we changed our approach to a simpler one, using standard pytorch networks. Instead of using their classes we employed a standard multi-head attention (taken from pytorch.nn with dropout probability of 0.1) and a feed forward network composed by a couple of linear transformations with an intermediate ReLU activation functions and a dropout layer with probability of 0.2. Thanks to these simplifications we got a network that is comparable with the standard one in term of memory usage.

Once we ended the implementation of our model we started to perform some training. We didn't have a checkpoint to start with, so we started our training from scratch (with standard initialization of the weights as in the Huggingface model). We trained our network for more than one epoch and we tried two different learning rate to see how much it influences the outputs. We perform the training with similar settings as previously described for fine-tuning. We only decreased the batch size to 10 and used learning rate of 0.1 for the third epoch.

4. Results

4.1. Fine-tuning

The results after fine-tuning are quite sufficient and satisfactory as we expected regarding the new control codes and relative sentences. As mentioned above in the paragraph 3.2, we perform the analysis with BLEU metrics, and the outcome are shown in the Table [1]. Using part of the annotations from the COCO validation set as reference set, all the values are better compared to the not pre-trained model, especially the BLEU-4 and SELF-BLEU-3. Given the length of the training time and the number of sentence to train, we decided to fine-tune it for only one epoch and just perform the comparison. The comparison is performed between this fine-tuned model and a new model trained only with our COCO dataset without starting it from a checkpoint, we still performed just one epoch to compare the results.

In addition to this quantitative result, the qualitative analysis also shows good generated sentences, as in the examples shown below:

Kitchen *A knife* and fork cut into four bowls of food.

Vehicle *A car* is parked near a kite in the grass.

Sports *A woman* with headphones standing alone on a suitboard.

Sports *A person* riding skis across a snow covered field.

Kitchen *A person* making a stack of pancakes in their hands.

Food *A person* holding a bowl of fruits with water.

In these examples the word in red is the control code, and the blue part is the starting input. *Kitchen*, *Sports*, *Food* and *Vehicle* are some of the new control codes. We can also notice that using different control codes with the same input the model adapts the generation to match the context of the control code.

Using the old model not fine-tuned without the new categories, the generation, as expected, is very bad: The main

Kitchen *A knife* and a knife.

problem of the fine-tuning is related to the previous categories, which now are less effective because the neural network tends to forget the previous learned information upon learning new information. This is the *catastrophic forgetting* problem, which it caused worse results with respect to the previous control codes, and better results with the new ones. In the following example a demonstration of this problem is shown using the same old control code, *"Wikipedia"*. The comparison is made between the sentence generated with the fine-tuned model (the first one) and the one generated with the CTRL original model (the second one).

Wikipedia *A car* is parked on a street with all luggage.

Wikipedia *A car* bomb exploded in the city of Kirkuk, killing at least 12 people and injuring more than 30 others. - January 3 – The Iraqi government announced that it would not allow a referendum on Kurdish independence from Iraq. ...

4.2. Evolved Transformer

On the contrary of the fine-tuned model, the new Evolved CTRL does not guarantees good output results. This worsening is easily noticeable by looking at the metrics scores listed in the Table [2]. In that table we compare the results of the training over 3 consecutive epochs. The scores are always lower than the ones of the standard CTRL listed in Table [1]. Not only they are lower than the fine-tuned model but also a bit lower than the ones achieved with only an epoch of training on a empty model (see Table [3]).

Furthermore the standard BLEU scores get lower from the first epoch to the second. On the other way we can see

Metric	Not Fine-tuned	Fine-tuned
BLEU-1	0.912417	0.977311
BLEU-2	0.800933	0.904347
BLEU-3	0.656049	0.789426
BLEU-4	0.496392	0.640745
SELF-BLEU-1	0.913938	0.943371
SELF-BLEU-2	0.694236	0.790118
SELF-BLEU-3	0.436293	0.590173
SELF-BLEU-4	0.251645	0.401788
POS-BLEU-1	0.958004	0.999487
POS-BLEU-2	0.956272	0.999129
POS-BLEU-3	0.953015	0.997969
POS-BLEU-4	0.94336	0.994107

Table 1: BLEU metrics comparison.

an improvement on SELF-BLEU. After analyzing these results we thought that the problem was on the learning rate so we tried to change it and, in particular, to increase it to 0.1 (for the first two epoch we used 0.01) because we were at starting steps of the training procedure. Differently from our prevision the scores went worse than before, probably cause the high increasing (10X before). We suddenly noticed a big decreasing of the metrics scores.

Metric	Epoch 1	Epoch 2	Epoch 3
BLEU-1	0.832407	0.787189	0.663274
BLEU-2	0.489352	0.432798	0.570655
BLEU-3	0.339345	0.283155	0.488831
BLEU-4	0.232121	0.187552	0.398599
SELF-BLEU-1	0.867994	0.933815	0.905392
SELF-BLEU-2	0.548223	0.757587	0.653678
SELF-BLEU-3	0.275747	0.514904	0.39477
SELF-BLEU-4	0.130936	0.295935	0.245933
POS-BLEU-1	0.969576	0.941479	0.683651
POS-BLEU-2	0.961107	0.942358	0.682104
POS-BLEU-3	0.951158	0.937750	0.677338
POS-BLEU-4	0.925324	0.917216	0.662195

Table 2: BLEU metrics obtained from training.

Same analysis and conclusions can be verified by looking at the generated sentences that are quite different from these who belongs to the COCO dataset. We noticed different behaviour over the three epochs: after the first one the generated sentences are, on average, very long and highly senseless but sometimes appears also a text that is comparable with the dataset. The outputs after the second epoch are similar to the previous one but they are longer and rarely appears a very short phrase. Instead, having changed the learning rate, the third epoch lead to wide changes in the outputs. Now the generated sentences are very short, in most of the cases too much short, that seems the network doesn't generate any words but only replicates the input and adds some useless word. Some examples taken from all epochs are shown below.

Person *a person* holding basic standing meal middle on.

Appliance *A man making* his which three cut their park sauce their straight one py middle on.

Generated text from epoch 1.

Outdoor *A bicyclist waiting* enclosed one luggage having types bottle sleeping having past case one the corner after cheese, ...

Generated text from epoch 2.

Indoor *A living room* with wine

Animal *A group of* zebra eating

Sports *Two dogs* playing in car

Generated text from epoch 3.

Metric	CTRL original model	Evolved model
BLEU-1	0.912417	0.832407
BLEU-2	0.800933	0.489352
BLEU-3	0.656049	0.339345
BLEU-4	0.496392	0.23212
SELF-BLEU-1	0.913938	0.867994
SELF-BLEU-2	0.694236	0.548223
SELF-BLEU-3	0.436293	0.275747
SELF-BLEU-4	0.251645	0.130936
POS-BLEU-1	0.958004	0.969576
POS-BLEU-2	0.956272	0.961107
POS-BLEU-3	0.953015	0.951158
POS-BLEU-4	0.94336	0.925324

Table 3: Difference between the original CTRL model and our model with Evolved Transformer.

5. Conclusion

During our work we focused on two main parts: we have tried to replicate the conditional text generation model and performed some fine-tuning on it with new dataset and new control codes. Then we tried to improve the text generation with the use of the Evolved Transformer, and tested with BLEU metrics if the model was actually better.

About the training with new dataset, this allow us to understand the benefits and the problem of fine-tuning. Surely it leads to worse performance with the old control codes, due to the *catastrofic forgetting*, but our main goal was to adapt the network to the new dataset without any constrain on previous learned knowledges. We manage to obtain good metric scores and also to generate nice and well-composed sentences, so we can state that we reached quite good results.

Sadly, our implementation of the new network with the Evolved Transformer blocks did not go as good as we expected. It gets worse results either about metrics either on sequences generation, that is the most valuable objective of the network. We tried to figure out the reason of such a bad behavior and we got these conclusion:

a possible error that we made is implementing a simplified version of the evolved transformer encoder.

Another consideration is more general and it's about the way we modified the network. Is it possible that we changed too much the whole structure, replacing all layers. Such a big modification may deserve more accurate studies or further optimization onto other parts of the overall network.

References

[1] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks., 2017. International Conference on Machine Learning. 2

[2] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms., 1991. In Foundations of Genetic Algorithms. 2

[3] N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher. Ctrl: a conditional transformer language model for controllable generation, 2019. 1, 3, 4

[4] T. W. Kishore Papineni, Salim Roukos and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation, 2002. Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002, pp. 311-318. 1, 3

[5] E. Real, A. Aggarawal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search., 2019. International Conference on Learning Representations. 2

[6] D. R. So, C. Liang, and Q. V. Le. The evolved transformer, 2019. 1, 2, 4

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin. Attention is all you need, 2017. 1, 2

[8] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. 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, Online, Oct. 2020. Association for Computational Linguistics. 1, 3, 4

[9] Y. Zhu, S. Lu, L. Zheng, J. Guo, W. Zhang, J. Wang, and Y. Yu. Texygen: A benchmarking platform for text generation models. *SIGIR*, 2018. 1, 2, 4