Data Speaks, E2E Data-to-Text with T5

Anonymous ACL submission

Abstract

We present an easy-to-use HuggingFace model tasked for data-to-text generation by extending the T5 (Raffel et al., 2019) model architecture with a language modeling head. Model variants allow for trade offs between semantic fidelity, accuracy, and text diversity. We report the outcomes of our research with regards to model complexity, resources, and text-generation techniques. The goal here is to operate in a true end-to-end fashion with no reliance on intermediate planning steps. With minimal training data, we leverage the Cleaned E2E dataset (Novikova et al., 2017) to achieve encouraging results across many automated metrics. We hope our research works as a foundation to easily extend our architecture to new datasets and domains.

1 Introduction

Data-to-text (D2T) generation broadly refers to the task of generating fluent text conditional on source content provided in the form of structured data such as tables, graphs, etc. (Kale, 2019). These non-linguistic data points at times can be complex and tough for humans to quickly process, this is where fluent text generation can play a key role. The problem statement can be extended to report generation in fields such as medicine, weather, and finance. In addition to longer form text, D2T can be used to generate short summaries or captions as well.

Traditional approaches to D2T follow a pipeline-based methodology by dividing the problem into several sub-problems (Reiter and Dale, 2000; Gatt and Krahmer, 2018). These stages include document planning (selecting events/data points to be incorporated into the output, establish ordering), micro-planning (choosing words and syntactic structures), and linguistic realization (constructing fluent, well-structured text from the generated outputs from the prior stages). Though very structured, this approach becomes problematic and slow due to collecting and processing the large amount of training data required for each stage.

Advancements in deep learning based language models have shifted research in data-to-text problems towards end-to-end approaches that do away with any and all intermediary steps. This framework simply requires training examples where the inputs are the data points, and the outputs are the gold-standard text. By fine-tuning a pre-trained language model on a domain-specific dataset, we are able to use transfer learning to successfully accomplish the data-to-text task.

In our work, we experiment with a variety of text generation techniques built on top of the core T5 architecture to output accurate text that performs well on automated evaluation metrics. Using a generic architecture, we fine tune the D2T task with a considerably small training dataset to generate text that achieves semantic fidelity and textual diversity.

2 Related Work

Prior research shows multiple studies that are conducted using both pipeline and/or end-to-end approaches. It is proven that pipeline based approaches achieve better semantic faithfulness as noted by (Moryossef et al., 2019). In addition a comparative study done across a variety of end-to-end (E2E) and pipeline approaches with the WebNLG dataset (Gardent et al., 2017) shows that the pipeline approach is significantly better at generalizing to unseen domains (Ferreira et al., 2019). One point to note however, is that all the end-to-end approaches in these studies have been completely trained from scratch on the corresponding task dataset. We hypothesized that using a pre-trained model with strong language generation capabilities raises the performance of end-to-end approaches.

One of the first shared tasks in natural language generation (NLG) aiming to assess the capabilities

of end-to-end, fully data driven NLG systems was conducted by (Dušek et al., 2019) in 2019. They created a novel dataset for the challenge, which is an order-of-magnitude bigger than any previous publicly available dataset for task-oriented NLG. The goal was to highlight that these systems can be trained from pairs of input, meaning representations (MR) and texts, without the need for fine-grained semantic alignments.

DATATUNER, a system from the Amazon Alexa Team, was built using GPT2 as the pre-trained language model to leverage the structured data from the Cleaned E2E dataset.(Harkous et al., 2020) They used a two-stage generation re-ranking approach, combining a fine-tuned language model with a semantic fidelity classifier. Each component is learnt end-to-end without needing dataset specific heuristics, entity delexicalization, or post-processing.

In another data-to-text study, (Kale, 2019) showed that fine-tuning T5 as a single end to-end model can outperform sophisticated, multi-stage pipeline approaches. Using the WebNLG, TotTo, and MultiWoz datsets he proved that T5 outperforms alternatives like BERT (Devlin et al., 2018) and GPT-2 (Radford et al., 2019).

3 Problem Description

The D2T task is formally defined as generating text T from data D that is encoded via a meaning representation (MR). We assume that content selection is done prior to the D2T task, an assumption also made in the dataset we used. Therefore, the text T should have semantic fidelity by conveying all the input data, and only the input data. (Harkous et al., 2020)

4 Data Set

We are using the **Cleaned E2E** dataset introduced in (Dušek et al., 2019). It is an automatically cleaned version of the original E2E dataset (Dušek et al., 2020), aiming to eliminate omissions and hallucinations in the human reference text (gold-standard) by fixing the corresponding meaning representations. In addition the training and validation sets are filtered so that they don't overlap with the test set. The data is presented in a format called **meaning representations (MR)**, as shown in Figure 1. These text inputs are atomic data points where each MR has 3-8 tag-value pairs with corresponding values. The goal is to generate a **reference text (REF)** as shown in Figure 2. These text blurbs should capture the data points from the respective MR into a fluent sentence.

```
'name[The Eagle], eatType[coffee shop],
food[Japanese], priceRange[less than £20], customer
rating[low], area[riverside], familyFriendly[yes],
near[Burger King]'
```

Figure 1: Example of a meaning representation.

```
'The Eagle is a low rated coffee shop near Burger
King and the riverside that is family friendly
and is less than £20 for Japanese food.'
```

Figure 2: Example of a human reference.

Table 1 presents high level data points gathered from EDA on the Cleaned E2E dataset. For greater understanding please refer to the Appendix that includes histograms and bar charts which present a more in depth view of the various tags included in the MR's, and a look at the distribution of token counts.

4.1 Data Sentiment

As an additional EDA step, we applied the **DistilBert** pipeline provided by HuggingFace to compute the overall sentiment of each human references in both the training and validation data. As shown in Table 1, on average outputs have **62% positive sentiment** and **38% negative sentiment**. The sentiment classification pipeline predicts positive or negative classification with **95% confidence**. This is a strong signal that fine tuning with the E2E Cleaned dataset implicitly primes our model to generate positive text. We must be cognisant of this as our dataset, which is comprised of restaurant descriptions, has an inclination to be positive.

5 Implementation

5.1 Pre-Training:

Our data-to-text generation model builds on the pre-trained T5: Text-to-Text Transfer Transformer (Raffel et al., 2019), a multi-layer, sequence-to-sequence language model. This model was pre-trained using the C4 [1] dataset in a multitask fashion, with an unsupervised "span masking" objective as well as on tasks such as supervised translation, summarization, classification, and question answering

[1] C4 is a colossal, cleaned version of Common Crawl's web crawl corpus.

Split	Size	Tok Count (mean)	Uniq_Tok_MR	Uniq_Tok_Ref	(+) Senti.	(-) Senti.
Train	33,525	42	112	3,674	61.5%	38.5%
Validation	4,299	45	79	1,392	61.3%	38.7%
Test	4,693	-	-	-	-	-

Table 1: **Cleaned E2E** dataset overview. **Tok Count** refers to the MRs. **Uniq_Tok_MR** and **Uniq_Tok_Ref** refer to number of unique tokens in w.r.t each group. We ensured no analysis was performed on the **Test** data.

T5 Variant	Parameter Count
T5-Small	60 M
T5-Base	220 M
T5-Large	770 M
T5-3B	3 B
T5-11B	11 B

Table 2: T5 variants, and their trainable model parameter counts

tasks (Kale, 2019). T5 has multiple variants as shown in Table 2, we have done a majority of our work using `t5-small`.

5.2 Fine-Tuning:

The T5 model uses both the encoder and the decoder of the original transformer, thereby framing tasks as sequence-to-sequence problems. The D2T task is cast in this text-to-text framework by representing the structured data as a flattened, linearized strings, shown in Figure 1. The corresponding label shown in Figure 2 is also linearized as textual output. We build off the pre-trained model weights, and token embeddings to fine-tune our task. As discussed in greater detail in Model Architecture as well as in Experiments, we iterated by tuning the following model parameters for fine-tuning.

- Batch Size
- Epochs
- Train/ Validation Step Size
- Learning Rate
- Training Optimizer
- Vector Dimensions

5.3 Tokenization

The T5 vocabulary consists of 32,100 uniquely identifiable sentence pieces. In alignment with the T5 model, we must use the appropriate tokenizer which was also used during model pre-training on the C4 Dataset mentioned in (Raffel et al., 2019). As a pre-processing step we chose to include the prefix `data_to_text` as a special token that is used to indicate our task. Understanding the spread of token counts was critical in choosing max token length for the encoder and decoder input vectors. These vector dimensions corresponded directly to how much information was provided to the model as we worked on minimizing model training/validation loss, as well as generating fluent text that performed well on automated metrics.

5.4 Model Architecture

The T5 sequence-to-sequence model uses both the encoder and the decoder of the traditional transformer model, this captures both token embeddings and positional embeddings which are learned at each layer. We provide a model diagram in Figure 3.

Inputs to the encoder are data D, to which we prepend our tasks' special token, and inputs to the decoder are T. We apply the T5 tokenizer to both the D and the eventual gold-standard output T to generate sequences of sub-word tokens which have encoded token IDs which correspond to T5's vocabulary. We will refer to these sequences as S

We then pass S in batches to the positional embeddings layer that is able to capture the input tokens' ordering. We then use the encoded tokens IDs to fetch the token embeddings to represent each token and ultimately sequence. During training these input token embeddings and positional embeddings are aggregated and passed to the first T5 layer. Our architecture consists of 6 encoder cells and 6 decoder cells, each apply multi-headed attention. The final layer of T5 normalizes using LayerNorm (Ba et al., 2016) before passing the intermediary output to the feed forward language-modelling head. At this point, the softmax calculation is applied on the feed-forward layer's output to generate probability distributions with respect to all potential output

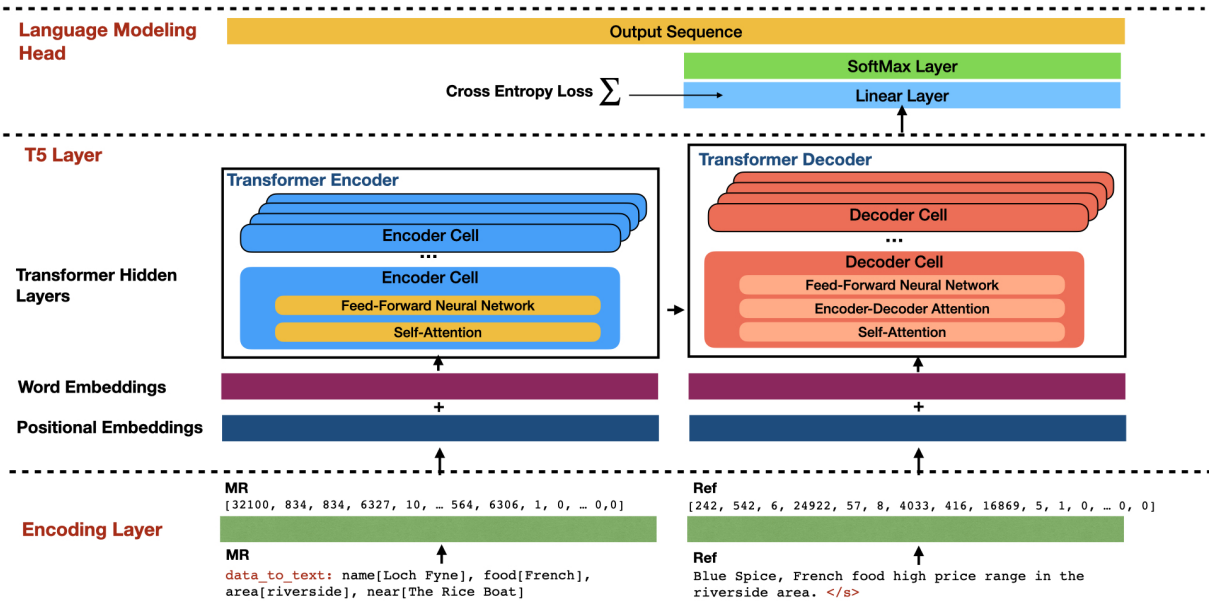

Figure 3: Model architecture for fine-tuning.

tokens.

Fundamentally the goal of text generation boils down to a language modelling training objective, therefore our objective function is cross-entropy loss. We want our network to learn the set of weights θ that minimize the cross-entropy loss $l = \sum_{n=|D|+1}^{|S|} \log P_\theta(s_i|s_0, ...s_{i-1})$. Our task is to generate text given the data, therefore cross-entropy loss is computed for the text following the prepended special token `data_to_text`.

5.5 Infrastructure

Our research was structured to run on public clouds such as Google Cloud Platform (GCP) and Amazon Web Services (AWS), leveraging their high speed deep learning focused infrastructure. Early EDA, data pre-processing, and initial modeling was conducted on GCP `c2-standard-8 (8 vCPUs, 32 GB memory)` instances. We immediately realized the need to increase compute resources; all further work was conducted on AWS Cloud using their deep learning machine images running on the Elastic Computing service (EC2). Infrastructure was set up using AWS `g4dn.2xlarge(32GiB RAM, 1 NVIDIA T4 GPU)` and `g4dn.8xlarge(128GiB RAM, 1 NVIDIA T4 GPUs)` instances which provided the level of compute we needed. We took advantage of the Elastic Book Store (EBS) and Simple Storage Service (S3) as mechanisms for saving trained models and selective outputs.

6 Metrics

Following previous shared tasks in related fields (Bojar et al., 2017b; Chen et al., 2015), we selected a range of metrics to measure word-overlap between system output and references. We compute the following metrics for each experiment run: BLEU, METEOR, ROUGE-L, and CIDEr.

BLEU (B) (Papineni et al., 2002), measures n-gram precision. METEOR (M) (Lavie and Agarwal, 2007), is based on the harmonic mean of uni-gram precision and recall while accounting for stem and synonymy matching. ROUGE-L (R) (Lin, 2004), calculates recall for the longest common sub-sequence. CIDEr (C) (Vedantam et al., 2004), is based on TF-IDF scoring of n-grams. We used the official metric evaluation scripts of the E2E challenge provided **here**.

7 Experiments

7.1 Overview

After locking in our model architecture, we searched for various avenues in which we could improve the performance of our model. The T5 architecture has multiple variants that differ in size, as shown in Table 2. We have conducted experiments using a couple of these variants. Due to the

nature of our task, evaluating success is split into both minimizing model cross-entropy loss during training and in tuning the text generation strategy to improve upon automated-evaluation metrics. We took a multi-phased approach to experimentation as explained in the following sections.

7.2 Model Size

After successfully running the initial suite of experiments on T5-Small, we ran the winning model on T5-Base. In the initial runs with T5-Base the model generated very poor output due to insufficient batch size and limited vector lengths due to our choice of `encoder_max_len` and `decoder_max_len`. When we increased the batch size and other relevant parameters, the model ran into out of memory issues on our custom GPU setup. We unfortunately were not able to get the infrastructure working on AWS, GCP, or Google Collab. This will be the first thing we experiment more on as we continue our research.

All the other experiments referenced are conducted using T5-Small.

7.3 Data Pre-Processing

Each MR is a sequence of tag-content pairs in the following manner.

```
tag[content], tag[content], ...
```

We applied various pre-processing techniques to build modified datasets, surprisingly these experiments did not lead to steady improvement across metrics. Refer to Table 5.

- Randomized MR arrangement
- Removed tags from each MR
- Linearized data with additional tags

  ```
  MR = <name> name=[Zizzi];
  <area> area=[riverside]
  <eatType> eatType=[coffee
  shop]
  ```

- Treating `data_to_text` as a special tag by adding it to the T5 vocab vs. not

7.4 Training Parameters

To fine-tune T5 with our dataset we ran multiple experiments with different model training parameters. Goal was to reduce loss and improve accuracy on training and validation data.

- Batch Size
  - Values in the range of 30-120
- Epoch Count
- Train/ Validate Step Size
- Learning Rate
  - Values in the range of 0.0001 to 0.001
  - CustomScheduler, varies based on epoch
- Optimizer
  - Adam
  - RMSProp,
  - Adamax,
  - Adagrad,
  - SGD,
- Vector Dimensions
  - Values in the range of 60-180

Refer to Table 3 for a summary of the top 5 training parameter focused experiments.

7.5 Generation Parameters

Using the winning training parameters, we ran over 75 experiments with different text generation techniques. For each technique we played with the value of corresponding hyper-parameters to compute metrics and identify top models.

- Number of Beams
- Sampling + Temperature
- Top K
- Top P
- Top P + Top K

Refer to Table 4 for a summary of the top 5 generation parameter focused experiments.

7.6 Results & Error Analysis

Starting with input data, we conducted multiple methods of pre-processing as different experiments to understand the effect these changes would have on the model performance. As shown in Table 5, the baseline model is run with Cleaned E2E dataset as is; the MRs are kept in a fixed order, with tag-content pairs, and no special tokens added to the T5 vocab. Removing the tags from MRs as well as shuffling the tag-content pairs within each MR had no significant effect on the model performance. In another experiment we attempted to give the model more confidence in data relationships by providing additional tags for each existing tag that per MR, here we observed slightly degraded model metrics. When we treated `data_to_text` as a special token we saw a strong positive effect on the model performance. From these experiments we learned that treating our task prefix as a special

t_acc	t_loss	v_acc	v_loss	opt	lr	enc_maxlen	dec_maxlen	spl_tok
0.9811	**0.2536**	**0.9834**	**0.2369**	**rmsprop**	**0.001**	**90**	**120**	**yes**
0.9811	0.2532	0.9829	0.2376	rmsprop	0.001	60	120	yes
0.9810	0.2536	0.9833	0.2377	rmsprop	0.001	60	120	no
0.9811	0.2535	0.9833	0.2380	rmsprop	0.001	120	120	no
0.9811	0.2533	0.9835	0.2383	rmsprop	0.001	90	120	no

Table 3: Summary of the top 5 performing training experiments

Experiment	BLEU	ROUGE_L	METEOR	CIDEr
T5Small-220MaxLength	33.36	51.80	38.12	2.0134
T5Small-300MaxLength	33.36	51.80	38.12	2.0134
T5Small-No-SplToken	33.28	52.53	38.15	2.1109
T5Small-Yes-SplToken	33.01	52.45	38.02	2.0646
T5Small-7Beams	33.00	51.85	37.95	1.9917

Table 4: Summary of the top 5 performing text generation experiments

token was beneficial and we continued to keep this change as we explored other avenues.

As shown in Table 3, choosing RMSProp as the training optimizer achieved the highest accuracy and lowest validation loss compared to other optimizers. In Tensorflow, RMSProp uses *momentum_decay_factor* as a factor in the step calculation which explicitly takes a fraction of the previous step into account when calculating the next step size. This approach seemed to be stronger in comparison to other optimizers that enable moving averages or are stochastic in nature.

Experiments for various encoder/decoder lengths essentially determine the number of slots in each input tensor that will contain a 128 dimensional token embedding. Our EDA showed that token counts averaged around 43 tokens, however our experiments showed that choosing a length well above the average, closer to the upper bound actually lead to better results. We noticed this for both the MR encoder input, and human reference decoder input. Lastly, as we tuned the learning rates, 0.001, came across as the best option for our task allowing for the model to learn the most through each gradient step.

During experimentation with generation parameters, we noticed that in the generated text many times would get cut off, and abruptly end. We attributed this to insufficient `max_length` and allowing `early_stopping` to occur. By setting `early_stopping = False` and increasing the `max_length` parameter we avoided cutting off sentences in the generated text. One detriment to increasing the max length value from 80 to 220 was that it increased the output generation time by 25x. We also found some cases of hallucination, but were able to address this by reducing the `max_length` as needed.

Through our experiments we confirmed that **beam search** based experiments produced text with the best semantic fidelity, however over time the results showed signs of sounding very templated. i.e. The model may be memorizing a structure, and generating outputs that consistently follow that pattern. Other experiments using **Sampling and Temperature** or **Top K/P** generated text that felt more *human*. Due to the probabilistic nature of these generation techniques, the correct output may not always be generated and therefore automated metric scores may take a hit. This is a trade-off for getting text that is more diverse, while still being mostly accurate.

8 Conclusion

In this study we evaluated end-to-end data-to-text generation using the pre-trained T5 model. We found that fine tuning the data-to-text task with limited training data achieves comparable or at times better results compared to pipeline based D2T approach. At generation time, the model design must make the trade-offs between semantic fidelity and textual diversity, this should be considered when choosing the most appropriate generation setting for any given application. Our work is publicly available for continued research and further development for this task.

Experiment	BLEU	ROUGE_L	METEOR	CIDEr
Baseline	18.18	44.58	25.65	0.9547
D2T_Spl_Tok	**26.30**	**47.27**	**31.67**	**1.5382**
No_Tags	18.49	45.11	26.59	0.9961
Shuffle_MRs	17.55	44.16	25.81	0.9303
Extra_Tags	16.43	37.46	22.60	0.5259

Table 5: Summary of the top performing pre-processing experiments

Acknowledgments

We'd like to thank our instructors, Professor Daniel Cer, Professor Mark Butler, Professor Michael Tamir, Professor Paul Spiegelhalter and Professor Joachim Rahmfeld. These individuals have aided us heavily in bouncing ideas, developing our problem statement, and research as a whole.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. 1607.06450. arXiv.

Ondrej Bojar, Yvette Graham, and Amir Kamran. 2017b. Results of the wmt17 metrics shared task. pages 489–513. Association for Computational Linguistics.

Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollar, and C. Lawrence Zitnick. 2015. Microsoft coco captions: Data collection and evaluation server. Microsoft.

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

Ondřej Dušek, David M. Howcroft, and Verena Rieser. 2019. Semantic noise matters for neural natural language generation. In *Proc. of the 12th International Conference on Natural Language Generation*, pages 421–426, Tokyo, Japan. Association for Computational Linguistics.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2020. Evaluating the State-of-the-Art of End-to-End Natural Language Generation: The E2E NLG Challenge. *Computer Speech & Language*, 59:123–156.

Thiago Castro Ferreira, Chris van der Lee, Emiel van Miltenburg, and Emiel Krahmer. 2019. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. 1904.03396. arXiv.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. Creating training corpora for nlg micro-planners. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 179–188. Association for Computational Linguistics.

Albert Gatt and Emiel Krahmer. 2018. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, page 61:65–170.

Hamza Harkous, Isabel Groves, and Amir Saffari. 2020. Have your text and use it too! end-to-end neural data-to-text generation with semantic fidelity. 2004.06577. Amazon Alexa, arXiv.

Mihir Kale. 2019. Text-to-text pre-training for data-to-text tasks. 2005.10433. Google, arXiv.

Alon Lavie and Abhaya Agarwal. 2007. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *In Proceedings of the Second Workshop on Statistical Machine Translation*, page 228–231. Association for Computational Linguistics.

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

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019. Step-by-step: Separating planning from realization in neural data-to-text generation. A long paper in NAACL-2019.

Jekaterina Novikova, Ondrej Dušek, and Verena Rieser. 2017. The E2E dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, Saarbrücken, Germany. ArXiv:1706.09254.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *In Proceedings of the 40th annual meeting on association for computational linguistics*, page 311–318. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. OpenAI.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou,

Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. 1910.10683. arXiv.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge, UK.

Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. 2004. Cider: Consensus-based image description evaluation. In *In Proceedings of the IEEE conference on computer vision and pattern recognition*, page 4566–4575. IEEE conference on computer vision and pattern recognition.

9 Appendix

A Tag Counts - Train

Figure 4: Bar Chart showing the spread of tag counts in the training data.

B MR Token Histogram - Train

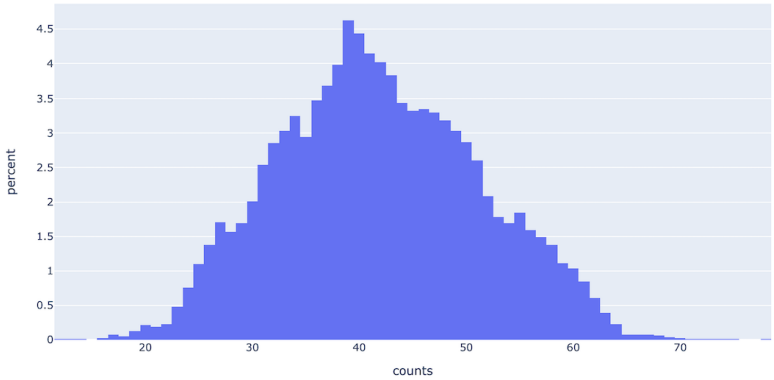

Figure 5: Histogram displays the length of meaning representations after applying T5 tokenization.

C MR Token Cumulative Histogram - Train

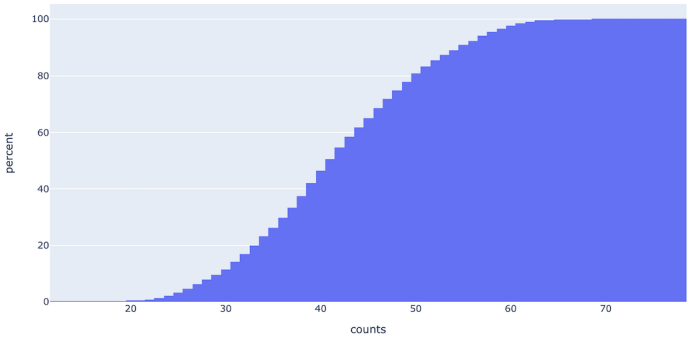

Figure 6: Cumulative Histogram displaying training data coverage based on token counts.

D MR Token Histogram - Validation

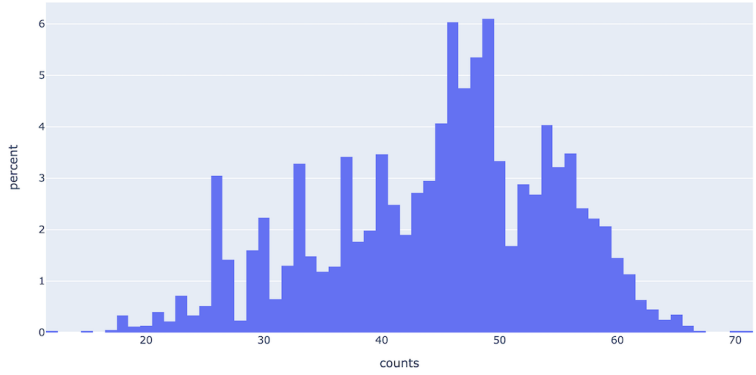

Figure 7: Histogram displays the length of meaning representations after applying T5 tokenization.