Data Speaks, E2E Data-to-Text with T5

Shyamkarthik Rameshubabu
University of California - Berkeley
karthikrbabu@berkeley.edu

Praveen Kasireddy
University of California - Berkeley
praveenreddy@berkeley.edu

Abstract

This study presents an easy-to-use 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. The report covers outcomes and analysis with regards to model complexity, resources, and text-generation techniques. The goal 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., 2016) to achieve encouraging results across many automated metrics. We hope our findings and design work as a foundation to easily extend this 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). 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 many fields including 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, transfer learning accomplishes the data-to-text task.

Experiments capture 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 GPT-2 (Radford et al., 2019) as the pre-trained language model to fine-tune on 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.

3 Problem Description

The data-to-text task is formally defined as generating text T from data D that is encoded via a meaning representation (MR). An assumption made is that content selection is done prior to the D2T task, an assumption also made in the dataset 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

The **Cleaned E2E** dataset introduced in (Dušek et al., 2019) is an automatically cleaned version of the original E2E dataset (Novikova et al., 2016), based out of the restaurant domain. It aims 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 across MRs, and a look at the distribution of token counts.

4.1 Data Sentiment

As an additional EDA step, the **DistilBert** pipeline provided by HuggingFace was applied to compute overall sentiment of each human reference 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 may implicitly prime the model to generate positive text.

5 Implementation

5.1 Pre-Training:

The 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. T5 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 (Kale, 2019). T5 has multiple variants as shown in Table 2, a majority of our work uses `T5-small`.

[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 w.r.t each group. 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

5.2 Tokenization

The T5 vocabulary consists of 32,100 uniquely identifiable sentence pieces, the same tokenizer used in this study is used during model pre-training on the C4 Dataset. As a pre-processing step, the prefix `data_to_text` was included as a special token used to indicate the task. Understanding the spread of token counts was critical in choosing max token length for the encoder and decoder input vectors. These vector dimensions correspond directly to how much information was provided to the model. The improvements worked to minimize model training/validation loss, as well as generate fluent text that performed well on automated metrics.

5.3 Fine-Tuning:

The T5 model uses both the encoder and 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 flattened, linearized strings, shown in Figure 1. The corresponding label shown in Figure 2 is also linearized as textual output. The fine-tuning process builds off the pre-trained model weights, and token embeddings. Experimentation is discussed in section 7.4

5.4 Model Architecture

As shown in the model diagram in Figure 3, our architecture begins with the encoding layer. Inputs to the encoder are data D, to which the D2T tasks' special token is prepended, and inputs to the decoder are T. The T5 tokenizer is applied to both data D and the eventual gold-standard output T to generate sequences of sub-word tokens which have encoded token IDs corresponding to T5's vocabulary. Refer to these sequences as S.

S is passed in batches to the positional embeddings layer that is able to capture the input tokens' ordering. The encoded token IDs are then used to fetch the token embeddings which represent each token and together represent the sequence. During training these input token and positional embeddings are aggregated and passed to the first T5 layer. The 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 tokens.

Fundamentally, the goal of text generation boils down to a language modelling training objective, therefore the objective function used is cross-entropy loss. The network learns the set of weights θ that minimize the cross-entropy loss $l = \sum_{n=|D|+1}^{|S|} \log P_\theta(s_i|s_0, ...s_{i-1})$. The 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

This 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 mod-

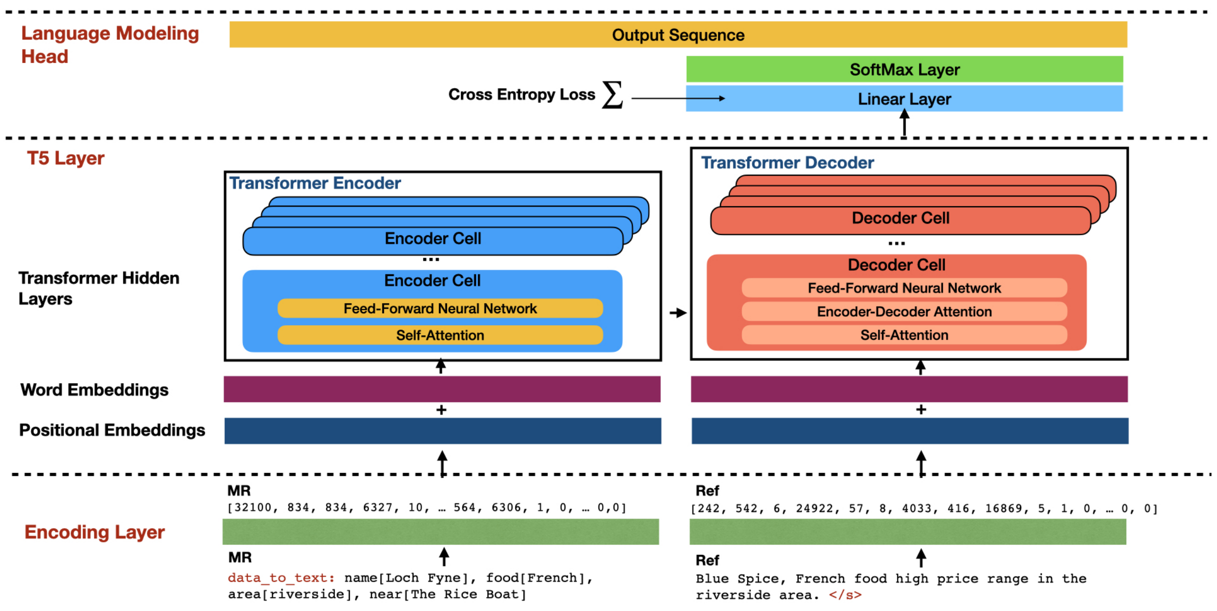

Figure 3: Model architecture for fine-tuning.

eling was conducted on GCP `c2-standard-8 (8 vCPUs, 32 GB memory)` instances. Quickly realizing the need to increase compute resources, all further work was shifted to AWS Cloud using 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 required. Elastic Book Store (EBS) and Simple Storage Service (S3) were used for saving trained models and selective outputs.

6 Metrics

Previous shared tasks in related fields (Bojar et al., 2017b; Chen et al., 2015) guided the choice of metrics to measure word-overlap between system output and references. The following metrics were computed for each experiment run: BLEU (Papineni et al., 2002), measures n-gram precision. METEOR (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 (Lin, 2004), calculates recall for the longest common sub-sequence. CIDEr (Vedantam et al., 2004), is based on TF-IDF scoring of n-grams.

7 Experiments

7.1 Overview

After locking in the model architecture, there were multiple avenues taken to improve model performance. 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 this 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. The multi-phased approach to experimentation is explained in the following sections.

7.2 Model Size

After successfully running the initial suite of experiments on T5-Small, the winning model was run 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 the choice of `encoder_max_len` and `decoder_max_len`. While increasing the batch size and other relevant parameters, the model ran into out of memory issues on our custom GPU setup. Unfortunately, we were unable to get the infrastructure working on AWS, GCP, or Google Collab. This will be the next set of experiments 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], ...`

Various pre-processing techniques were applied to build modified datasets, 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

7.4 Training Parameters

Multiple experiments with different model training parameters were run to fine-tune T5 with the dataset. Goal was to reduce loss and improve accuracy on training and validation data. Some key hyper-parameters include:

- 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, over 75 experiments were run with different text generation techniques. For each technique the respective hyper-parameters were iterated upon to identify the top models based on automated metrics. Some key categories include:

- 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

The baseline model, as shown in Table 5, 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 attempts to help the model identify data relationships, additional tags for each existing tag per MR was added; however this experiment resulted in degraded model metrics. When treating `data_to_text` as a special token there was a strong positive effect on the model performance.

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. In comparison to optimizers that are stochastic in nature or those that enable moving averages, the method used in RMSProp was more effective.

Experiments on various encoder and decoder input lengths determine the number of slots in each input tensor that will contain a 768 dimensional token embedding. EDA showed that token counts averaged around 43 tokens, however the experiments revealed that choosing a length well above the average, closer to the upper bound actually lead to better results. This behavior was evident for both the MR encoder input, and human reference decoder input. Lastly, experiments showed that choosing learning rate as 0.001 was the best option

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

for this task, allowing the model to learn the most through each gradient step.

During generation parameter experiments, many times the output text was abruptly cut off. This was attributed to insufficient `max_length` and allowing `early_stopping` to occur. By setting `early_stopping = False` and increasing the `max_length`, the issue of cut off sentences was avoided. One detriment to increasing the max length value from 80 to 220 was that it increased the output generation time by 25x. In some cases the model started to hallucinate and generated irrelevant text, this was addressed by reducing the `max_length` as needed.

Experiments confirmed that beam search based approaches produced text with the best semantic fidelity, however over time the results showed signs of sounding very templated. 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, a chance exists that the output may not line up with the expected gold-standard text. Experiments showed that automated metric scores took a slight hit, this is a trade-off that is made for generating more diverse text.

8 Conclusion

This study evaluated end-to-end data-to-text generation using the pre-trained T5 model. It was shown that fine tuning the data-to-text task with limited training data achieves comparable or at times better results compared to pipeline based D2T approaches. The model design makes a trade-off between semantic fidelity and textual diversity. Our work is publicly available for continued research and further development on this task.

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*

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

Conference on Natural Language Generation, pages 421–426, Tokyo, Japan. Association for Computational Linguistics.

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, Oliver Lemon, and Verena Rieser. 2016. Crowd-sourcing nlg data: Pictures elicit better data. Edinburgh, EH14 4AS, UK. Interaction Lab, Heriot-Watt University.

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

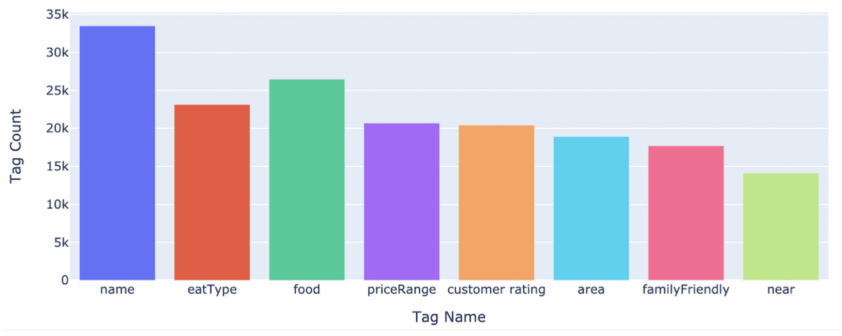

Figure 4: Bar Chart showing the spread of tag counts in the training data.

B MR Token Histogram - Train

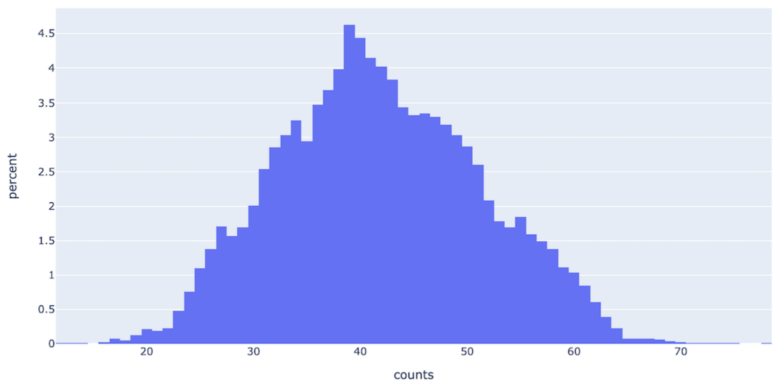

Figure 5: Histogram displays the length of meaning representations after applying T5 tokenization.

C MR Token Cumulative Histogram - Train

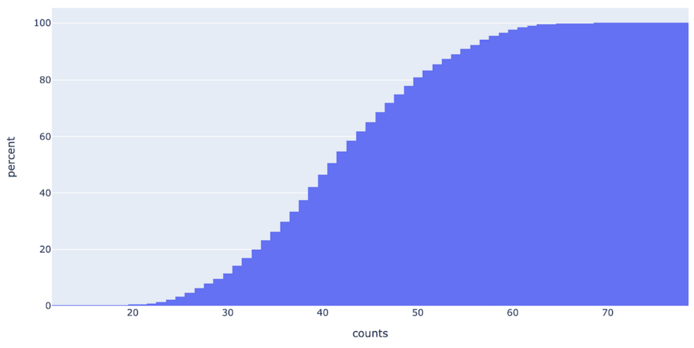

Figure 6: Cumulative Histogram displaying training data coverage based on token counts.

D MR Token Histogram - Validation

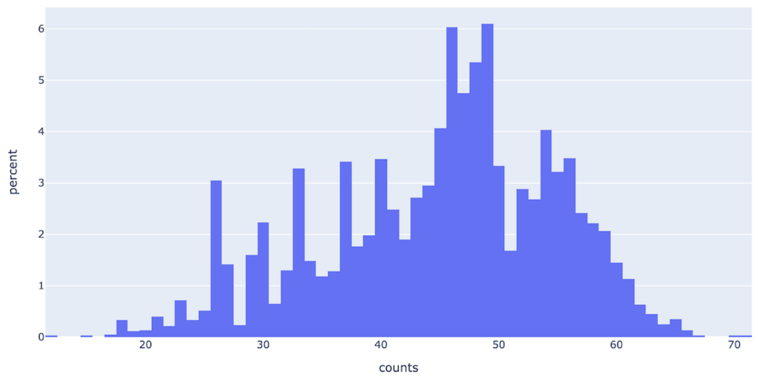

Figure 7: Histogram displays the length of meaning representations after applying T5 tokenization.