Analyzing RANKGEN Across Different Domains

Jimmy Hoang
Princeton University
jhoang@princeton.edu

Jenny Pang
Princeton University
jp54@princeton.edu

Alexis Wu
Princeton University
alexiswu@princeton.edu

Abstract

In this study, our research serves to broaden the analysis and explore the capabilities of RANKGEN (Krishna et al., 2022), a scoring function and decoding method for text generation, through its application to datasets of various domains. First, we reproduce the checkpoints set by the founders of RANKGEN, using the baseline RANKGEN model with GPT2-Medium as the generative language model to replicate their results for RANKGEN's performance on a Wikipedia dataset. Like the authors, we also use MAUVE as our metric for automatic performance evaluation. Then, we investigate the model's generalizability to four specialized domains not present within RANKGEN's training data: government reports, mathematical proofs, code completion, and poetry. Notably, although RANKGEN had not been trained on government reports, it achieved a MAUVE score of 87.1 out of 100, outperforming the authors' reported results from using it to score generations for the Wikipedia dataset. On the other hand, RANKGEN did not generalize well to mathematical proofs, code completion, or poetry, yielding MAUVE scores of 2.5, 0.8, and 42.9, respectively. Mirroring the methodology of the paper's authors, we also engage human evaluators to compare government report texts generated by RANKGEN to those produced by a decoding algorithm employing top-k sampling. We discovered that evaluators preferred RANKGEN generations 70% of the time. Our experiments lay groundwork for studying the generalizability of RANKGEN so that text generation by language models can be further improved in the future.[1][2]

[1]Resources are available at https://github.com/jimmytienhoangy/rankgen

[2]Results can be found at https://drive.google.com/drive/folders/14A9sp4uElweJ08xU385eENKzrmSTKtdW?usp=sharing

1 Introduction

In recent years, large language models (LLMs) have gained widespread traction for their utility in prompt-driven text generation. However, these LLMs can produce outputs that are prone to be repetitive, incoherent, and/or irrelevant to the prompt.

In an effort to address these shortcomings, RANKGEN (Krishna et al., 2022) was developed. RANKGEN is a flexible encoder model that scores and ranks text generations based on individual coherence and consistency with the input sequence (termed as "prefix"). There are four different versions of RANKGEN, each of which were trained on different subsets of the following four domains: PG19, Wikipedia, C4-NewsLike, and C4-WebTextLike. To increase the likelihood for RANKGEN to generalize to other domains, we used the comprehensive RANKGEN model trained on all four of these domains in our replication and experiments outlined below.

To reproduce the RANKGEN developers' results, we replicate their methodology of using RANKGEN in a beam search setup. Using the baseline RANKGEN model and generations from OpenAI's GPT2-Medium LLM, we first evaluate RANKGEN's performance on Wikipedia datapoints, using MAUVE as our automatic metric. We then probe the model's generalizability with four new testing domains not present within RANKGEN's training data: government reports, mathematical proofs, code completions, and poetry. Each of these domains has unique characteristics that can provide us with interesting insights into the current capabilities of RANKGEN.

After generating continuations for prefixes from each of these four chosen domains, the results affirm our initial performance hypotheses. We generally observe better performance with domains that are long-form and follow standard English

structure and conventions while domains that deviate with uncommon notations and syntax perform worse. For comparison, we also use top-k sampling to generate continuations and then compare its MAUVE scores against RANKGEN. For mathematical proofs and code completions, RANKGEN underperforms as expected when compared to more basic decoding algorithms like top-k sampling.

In addition to our automatic metric, we also conduct human evaluation to further gauge the desirability of RANKGEN outputs. We create a survey and recruit fellow collegiate students to participate. By using blind A/B testing with RANKGEN and top-k sampling as our decoding methods for outputting continuations, we ask participants to choose the better generation given a prefix. For each selection, participants must also provide reasons motivating their choice. Our survey's results show that evaluators typically favor RANKGEN's generations over generations from top-k sampling.

We hope that our study can effectively contribute to the existing literature on RANKGEN, highlighting its promising potential to further and improve text generation research.

2 Related Work

Currently, when generating text, LLMs still suffer from issues where greedy decoding and beam search can frequently result in sequences that are repetitive and uninteresting, particularly when they have high probability (Zhang et al., 2021). Although techniques like top-k sampling have been introduced to address these problems, they often result in inconsistencies, hallucinations, or factual errors (Massarelli et al., 2020). Additionally, models tend to focus too much on local context and fail to grasp long-range dependencies (Sun et al., 2021), which can cause coherence and consistency problems in longer texts.

2.1 Overview of RANKGEN

RANKGEN is a 1.2 billion parameter English encoder model presented by Krishna et al. (2022) to improve text generation. This large ranking model "maps human-written prefixes to model-generated continuations of those prefixes." To measure the compatibility between a prefix and a generated text sequence, RANKGEN projects the prefix and sequence into a vector space and calculates a score using a dot product. After computing these scores, RANKGEN uses them to rank the best generations. Because the model considers two sequences, the prefix and the generation, as opposed to other techniques that only examine singular or local tokens, RANKGEN better learns long-distance relationships between prefixes and continuations.

We provide a brief, high-level overview of Krishna et al. (2022)'s methodology and findings to provide background for our work. We encourage readers to visit the original RANKGEN paper for the in-depth implementation details and additional results.

To train RANKGEN, Krishna et al. (2022) employ large-scale contrastive learning with in-batch negative sampling. Each RANKGEN training instance consists of a tuple (p_i, c_i, g_i), where p_i represents a 256-token prefix, c_i represents the corresponding ground-truth continuation varying in length (10 to 128 tokens long), and g_i represents a continuation generated by a language model. The contrastive objective pushes p_i close to c_i and away from both g_i and c_j $(j \neq i)$ in the same minibatch.

After training the model, Krishna et al. (2022) perform experiments with four model variants (each with 1.2B parameters), three of which were trained on solely the Project Gutenberg (PG19) books library dataset and one trained on the union of four domains (PG19, Wikipedia, C4-NewsLike, and C4-WebTextLike). The PG19 dataset is a set of books extracted from the Project Gutenberg library. It primarily contains works of Western literature like novels and short stories as well as reference works and periodicals. The Wikipedia dataset is a set of cleaned Wikipedia page articles stripped of markdown and unwanted sections like references. The C4-NewsLike and C4-WebTextLike datasets are subsets of the C4 dataset that is a colossal, cleaned version of Common Crawl's web crawl corpus. These subsets contain text extracted from online news articles and general websites, respectively.

These experiments compared the MAUVE performance scores of RANKGEN variants to baseline decoding algorithms (i.e. greedy decoding, ancestral sampling, top-k, etc.) with four pretraining datasets of the LM (T5-XXL-C4, GPT2-Medium, GPT2-XL, and T5-XXL-PG19). Krishna et al. (2022) found that the last RANKGEN variant (RANKGEN trained on the union dataset with both INBOOK and GENERATIVE negatives), when combined with beam search, resulted in the best average MAUVE score. The original pa-

per also uses human evaluation to substantiate the MAUVE score results, comparing the best performing RANKGEN variant to nucleus sampling, currently one of the most commonly used decoding algorithms. These evaluations revealed that human evaluators significantly prefer outputs from RANKGEN over nucleus sampling.

2.2 LLMs for Other Domains

Recent advancements in language have spurred research efforts in developing specialized models for domain-specific tasks, including mathematical problem solving, code completion, and poetry generation.

Regarding large language models' text generation abilities for mathematics and problem solving, Google Research's Minerva (Lewkowycz et al., 2022) is a language model evaluated on MATH (Hendrycks et al., 2021b), GSM8k (Cobbe et al., 2021), MMLU-STEM (Hendrycks et al., 2021a), and undergraduate-level, multi-step reasoning problems taken from MIT OpenCourseWare. On these four datasets, the best Minerva model (Minerva 540B) achieved a performance of 50.3%, 78.5%, 75.0%, and 30.8% accuracy, respectively. Researchers cited prevailing errors of the worse-performing 8B and 62B models to be dominated by incorrect calculations and incorrect reasoning.

As for code completion, SLANG is a system that combines program analysis and statistical language models, extracting sequences of API calls from a massive input codebase (Raychev et al., 2014). These sequences are then fed to an n-gram or recurrent neural network model to be trained on, learning the probabilities of each sequence. Evaluation results found that the best model was a combination model of RNNME-40 and a tri-gram model, returning the correct completion of a partial program with holes in 69% of test cases.

In the realm of poetry generation, recurrent neural network-based models have been proposed for generating classical Chinese poetry (Zhang and Lapata, 2014). In contrast to the long-document formats of the PG19 and Wikipedia training sets used to train RANKGEN, poetry often demands "a set of structural, phonological, and semantic requirements" (Zhang and Lapata, 2014). We anticipate that these differences may impact RANKGEN's ability to perform well with our poetry dataset. To evaluate their model, Zhang and Lapata (2014) created a corpus of 284,899 classical Chinese poems. The authors then compared their poem generator (RNNPG) to statistical machine translation (SMT) approaches via both BLEU-2 score and human evaluator rankings based on fluency, coherence, meaning, and poeticness. As a summary of their results, RNNPG outperforms other methods (except human-authored poems).

While language models have been trained and developed on domain-specific datasets, our goal is to better understand how well RANKGEN, in particular, is able to approach different domain-specific datasets, analyzing its performance on tasks including solving math problems, completing code, and generating poetry.

3 Datasets

To reproduce RANKGEN's results achieved by Krishna et al. (2022), we evaluate the RANKGEN-base model, trained on all four original datasets, on their publicly available and cleaned Wikipedia dataset. To investigate the generalizability of RANKGEN, we design a new experiment that uses two datasets created for evaluative research papers and two Hugging Face datasets: GovReport (Huang et al., 2021), MetaMathQA (Yu et al., 2023), Python code instructions (Bisht, 2023), and poetry (Noyan, 2021).

3.1 Dataset Construction

Before evaluating RANKGEN on these datasets, we pre-processed the datasets to match the format outlined in the original paper. The GovReport dataset contained long government reports in one column and a summary of the report in the other column. We focused on the column with the raw government reports and used the test split of the dataset, which had 973 reports. Using the NLTK tokenizer, for each report in the dataset, we first tokenized the report into sentences. This was a measure that the original authors took to ensure prefixes and target continuations ended at sentence boundaries to "make the task less reliant on local syntactic patterns" (Krishna et al., 2022). To ensure compatibility with RANKGEN input lengths, we similarly used a max prefix length of 256 tokens and a max target length of 128 tokens, checking the lengths using the GPT2 tokenizer. For example, in one government report, we appended sentences to the prefix until the max length of 256 tokens was reached. Then, for the remainder of the report, we constructed each target by adding sentences until

the max length of 128 tokens was reached, moving onto the next target's construction until the end of the report.

The MetaMathQA and Python code instructions datasets were constructed in a similar way. Both of these datasets had 395k rows and 18.6k rows, respectively, which were both larger than the size of the Wikipedia dataset (7.7k) and GovReport dataset. In the MetaMathQA dataset, we filtered out datapoints with LaTeX code to avoid "overlap" with the Python code dataset. In the Python code instructions dataset, we filtered out datapoints that had a paired input. For example, we removed prefixes like "Compute the sum of the given array." with a paired $[1, 2, 3]$ and kept general prefixes like "Create a Python program to compute the sum of a given array." We set a seed of 484 and randomly sampled 973 rows from the MetaMathQA and Python code instructions datasets to align with the number of datapoints in GovReport.

The poetry dataset contained 573 poems. Due to the short length of the poems, we decreased the max prefix length from 256 tokens to 128 tokens and the max target length from 128 tokens to 64 tokens. We chose a max prefix length of 128 tokens after considering experiments conducted by Krishna et al. (2022) that suggested RANKGEN's robustness to prefix length when ranging from 128 tokens to 500 tokens and examining MAUVE score variation with changes in prefix length. 276 poems remained after processing the dataset.

4 Replication

RANKGEN has three model sizes: base, large, and XL. In consideration of time constraints and limited computing resources, we chose to reproduce Krishna et al. (2022)'s MAUVE results for their RANKGEN-base model that was trained on all four domains (PG19, Wikipedia, C4-NewsLike and C4-WebTextLike) and used the GPT2-Medium generator language model when evaluating on the Wikipedia dataset. RANKGEN-base has 110.2 million parameters compared to RANKGEN-large's 342.3 million and RANKGEN-XL's 1.2 billion parameters. Like RANKGEN-large, RANKGEN-base has a larger mini-batch size of 4096 than RANKGEN-XL, which has a mini-batch size in the range of 256-1536. We chose to replicate the authors' use of GPT2-Medium as the generator language model because, in comparison to GPT2-XL, the authors found that GPT2-Medium had less than a 1% difference in MAUVE scores. Additionally, to increase probability of generalizability to other domains in our experiments, we chose to use the RANKGEN model trained on all four original domains in our replication instead of the RANKGEN model that was only trained on the PG19 dataset. For the RANKGEN setup, when given a prefix, 10 samples are generated where each sample has a length of 20 tokens. With a beam size of 2, the top 2 samples are concatenated to the prefix and generation continues. When running RANKGEN-base on the Wikipedia dataset, for 800 datapoints, it took approximately 3 hours on the Google Colab T4 GPU. For 7713 datapoints, it took 29 hours in total. We also utilized top-k sampling as opposed to RANKGEN on the Wikipedia dataset to replicate Krishna et al. (2022)'s results that showed RANKGEN outperforming this decoding algorithm. We chose top-k sampling over the other decoding methods used by Krishna et al. (2022) because of its higher average MAUVE scores.

5 Experiments

In the original paper, the authors only evaluated RANKGEN's performance on the PG19 and Wikipedia datasets, but these two domains were already present within the training data. Thus, our goal was to answer two research questions: 1) "To what extent does RANKGEN generalize to unseen, specialized domains?" and 2) "Does RANKGEN outperform other decoding strategies for these domains?"

5.1 GovReport

Given a snippet of a government report, how well can RANKGEN produce continuations that are similar to the original report's content? The GovReport dataset includes reports written by governmental research agencies, such as the Congressional Research Service and the United States Government Accountability Office (Huang et al., 2021). Given that there may be more policy or bureaucratic-related vocabulary used in this specialized dataset, we wanted to observe if RANKGEN's performance was still comparable when generating continuations for it. We hypothesized that RANKGEN would still perform well because the reports are of long-content form and typically follow standard English syntax, but its performance may be limited by the fact that government reports were an unseen domain in the training set.

5.2 MetaMathQA

Given a math question to solve and the relevant background information to solve the problem, how well does RANKGEN complete the proof? Mathematical proofs are a domain with sequential data, which adds another layer of complexity to the gold-standard generated output. Compared to the training domains, instead of just needing to stay relevant to the overall topic, the generated text must now also follow a certain logical order. The queries included in the MetaMathQA dataset are also considerably shorter than the Wikipedia text and GovReport reports, but experiments by Krishna et al. (2022) ruled that RANKGEN's performance is robust to prefix length. We hypothesized that RANKGEN's performance in this domain would be inhibited by the lack of sequential training data.

5.3 Python Code Instructions

Given instructions to program a task in Python, how well does RANKGEN return valid Python code that can be run to carry out the task? Code completion is another domain with sequential data but also involves syntax that is very different from standard English. We hypothesized that RANKGEN performance would falter in this regard, due to the absence of code in its training data and its overall vastly different structure from standard English.

5.4 Poetry

Given poetry, how well does RANKGEN continue the style and characteristics of the work? Poetry often does not abide by standard English grammar rules, as poets such as Shakespeare can create new words to suit their needs and are not required to follow a uniform sentence structure. The messages underlying poetry are often up to interpretation as well, which can be an obstacle for RANKGEN when generating logical and continuous suffixes. Thus, we hypothesized that RANKGEN would underperform in comparison to a "more easily" understood dataset like GovReport because of the freedom and artistic liberties that poets usually write with in their work.

5.5 Automatic Evaluation Metrics

In similar fashion to Krishna et al. (2022), we use MAUVE as our automatic evaluation metric to reproduce their RANKGEN performance on the Wikipedia dataset as well as to compare the performance of RANKGEN on our four experimental testing datasets. MAUVE measures the similarity between human-written text and generated text, and compared to other automatic metrics, has been found to better correlate with human judgment (Pillutla et al., 2021).

5.6 top-k sampling

top-k sampling is one of Krishna et al. (2022)'s baseline decoding methods. As mentioned previously in the Replication section, for a comparison metric against RANKGEN's performance, we chose top-k sampling over the other baseline decoding methods in Krishna et al. (2022) because of its higher average MAUVE scores. In top-k sampling, at each timestep, the next token or word is randomly sampled from the top k most probable ones (Fan et al., 2018). Mirroring the authors, we similarly use $k = 40$ and average the MAUVE scores for top-k sampling over five runs in our experiments. Generated text was truncated at sentence boundaries to ensure lower variation in MAUVE scores (Krishna et al., 2022). MAUVE scores of RANKGEN and top-k generations were compared to determine whether RANKGEN still outperforms top-k sampling when evaluated on our four chosen datasets from unseen specialized domains.

5.7 Human Evaluation Metrics

In addition to automatic metrics, we surveyed Princeton and other college-level students to determine if RANKGEN's text generation was preferred compared to generations from top-k sampling. Due to the expensive nature of human evaluation, we only compared RANKGEN to top-k sampling on the GovReport dataset.

From the 973 GovReport top-k sampling generations, we randomly sampled 10 to be used in blind A/B testing. We then found the corresponding 10 generations in the RANKGEN outputs by prefix matching. For each prefix, the corresponding RANKGEN and top-k generated text were randomly shuffled to be either option A or B.

Human evaluators were asked to choose the continuation of the prefix that they preferred more and select among a list of reasons to explain each of their selections. The list of possible reasons to explain their choices were taken from Krishna et al. (2022) when they compiled their participant's explanations. We received 12 responses to our survey.

MAUVE scores

Dataset	RANKGEN-base	top-k
Wikipedia	86.8	73.8
GovReport	87.1	76.7
MetaMathQA	2.5	8.7
Python code	0.8	1.4
Poetry	42.9	28.8

Table 1: MAUVE scores from reproducing Krishna et al. (2022)'s Wikipedia results and evaluating the generalizability of RANKGEN on government reports, mathematical proofs, Python code completion, and poetry.

6 Results and Discussion

6.1 RANKGEN vs. top-k sampling

In Table 1, we show the results of our MAUVE scoring on the RANKGEN outputs of the Wikipedia, GovReport, MetaMathQA, Python code completion, and poetry datasets. Our Wikipedia MAUVE score was 86.8 versus Krishna et al. (2022)'s 83.8. Our higher MAUVE score can be attributed to differences in the generated beam search outputs, but overall, our reproduced results are consistent with those of Krishna et al. (2022) as RANKGEN outperforms the top-k sampling decoding algorithm at a similar magnitude in terms of MAUVE scores.

GovReport achieved a very high MAUVE score of 87.1, outperforming top-k sampling. This indicates that RANKGEN generalizes well to other long-form English content, even if the specialized domain of government reports was not included in the training data.

On the other hand, we observed that RANKGEN performed extremely poorly, with MAUVE scores of near 0, when tasked to generate math proofs and Python code completion. This suggests that the ground-truth and generated text were not similar at all. Top-k sampling outperformed RANKGEN on these two datasets.

RANKGEN performed better on the poetry dataset compared to the math and code datasets, with a MAUVE score of 42.9, outperforming top-k sampling by a large magnitude. However, this score is still significantly lower compared to the Wikipedia and GovReport MAUVE scores, suggesting that the ground-truth and generated text were somewhat similar but RANKGEN could not learn how to generate high-quality text in the creative poetry domain.

6.2 Examining RANKGEN's Generated Outputs

To give intuition behind RANKGEN's MAUVE scores, we provide sample outputs from each of the four datasets in the Appendix. In Table 4, we compare the ground-truth with an example of RANKGEN's output for the GovReport dataset. As shown, the generated output from RANKGEN stays relevant to the prefix, is coherent, and is not repetitive. This level of quality is consistent throughout other GovReport RANKGEN generations as well.

In Table 5, we compare the ground-truth with an example of RANKGEN's output for the MetaMathQA dataset. In the generated output, RANKGEN is unable to solve the given math problem and instead loses focus of it, beginning to make many unnecessary assumptions. In many of the other examples, RANKGEN proposes additional math questions, disregards the given information and draws information elsewhere, or gives an incorrect answer with an incorrect proof.

In Table 6, we compare the ground-truth with an example of RANKGEN's output for the Python code instructions dataset. In the generated output, RANKGEN is unable to follow and provide proper Python code to answer the given instructions. In many other outputs as well, the question is repeated several times, the code is minimal and incorrect, or the problem is just discussed until the max generation length is reached.

In Table 7, we compare the ground-truth with an example of RANKGEN's output for the poetry dataset. In this example and many other generated outputs, RANKGEN generates a passage that contains some words relevant to the prefix, but the continuation is nonsensical and syntactically/structurally incorrect.

6.3 Human Evaluation Results

Out of the 10 questions, 9 had a majority of evaluator responses in favor of RANKGEN's generation. The remaining question was a tie. On average, a participant preferred the RANKGEN generation over the generation from top-k sampling for 70% of the questions with a standard deviation of 21% and variation of 44%. Furthermore, 58.33% of participants preferred the RANKGEN generation for any given question on average with a standard deviation of 14% and variation of 23%.

To gauge the motivations behind participant pref-

Reasons relating the prefix with the generation (70.55%)	
More topically relevant to the prefix	22.70%
Better continuity / flow / chronology	38.04%
Does not contradict prefix	1.83%
Stylistically closer to prefix	7.98%
Reasons related only to the generated text (29.45%)	
Better commonsense understanding	7.98%
Less repetitive	9.19%
More grammatical	6.13%
Less contradictions	0.00%
More coherent / other	6.13%

Table 2: Distribution of reasons given by our human evaluators (college, mainly Princeton, students) for choosing RANKGEN generations over generations from top-k sampling. Percentages are computed over 163 total reasons.

Reasons relating the prefix with the generation (59.76%)	
More topically relevant to the prefix	18.29%
Better continuity / flow /chronology	26.83%
Does not contradict prefix	2.44%
Stylistically closer to prefix	12.20%
Reasons related only to the generated text (40.24%)	
Better commonsense understanding	10.98%
Less repetitive	4.88%
More grammatical	13.41%
Less contradictions	0.00%
More coherent / other	10.97%

Table 3: Distribution of reasons given by our human evaluators (college, mainly Princeton, students) for choosing generations from top-k sampling over RANKGEN generations. Percentages are computed over 82 total reasons.

erences, we also asked participants to provide reasons for their choices. In Table 2, we see that 70.55% of participant answers that preferred a RANKGEN generation to a top-k sampling generation were motivated by reasons relating the prefix to the generation, such as the generation's relevance to the prefix and continuity, while 29.45% were motivated by reasons relating only to the generated text such as its standalone grammatical correctness and conciseness. In Table 3, we see that 59.76% of participant answers that preferred a generation from top-k sampling over a RANKGEN generation were motivated by reasons relating the prefix while 40.24% were motivated by reasons relating only to the generated text. The major takeaway from these two tables is that if a RANKGEN generation was preferred, it was more likely to have been because it was a better continuation in relation to the prefix, which is the fundamental goal of RANKGEN.

7 Conclusion and Future Work

We replicated Krishna et al. (2022)'s RANKGEN performance on the Wikipedia dataset and saw consistent results. Then, we investigated RANKGEN's generalizability to other domains by testing the model on four different datasets. As shown by high MAUVE and human evaluation scores, RANKGEN generalized well to government reports, a domain that it was not trained on. This suggests that RANKGEN can generalize well to other English long-form content outside of the initial training domains. Low MAUVE scores and an analysis of RANKGEN's generated outputs for the MetaMathQA, Python code instructions, and poetry datasets indicate that RANKGEN does not generalize well to these three domains because of vastly different styles and structures.

Future work could involve training RANKGEN on long-form math proofs, code outputs, or poetry and investigating whether this will improve the quality of the generated output.

We also suggest that a follow-up human evaluation involving a larger number of evaluators be conducted to draw more comprehensive conclusions. In addition to undergraduate-level students, these surveys could be extended to graduate-level students, professors, and other experts to incorporate feedback from domain professionals. Furthermore, it may be valuable to have experts from outside of the field of Computer Science and language model sector participate. It would be useful to have specialists in domains like public policy, mathematics, and/or English provide their thoughts as human evaluators on the RANKGEN text generations.

Limitations

After processing the poetry dataset, only 276 out of the original 573 datapoints remained, which prevents a fair comparison to the other larger datasets that we tested RANKGEN on.

We could not use the RANKGEN-XL model, given the limitations in our computational power and resources as well as time, which most likely would have produced higher quality generations and led to stronger insights.

We were unable to recruit a sufficient number of students to participate in our survey for the results to be statistically significant. The length of the survey was a detracting factor, but the length was inevitable given the long format of the GovReport datapoints.

Ethics Statement

As noted in the original paper, RANKGEN, similar to other current text generation techniques, may output factually incorrect information. Though we used the GovReport dataset to analyze RANKGEN's ability to generate text with characteristics including being coherent with the prefix, grammatically correct, commonsense, and topically relevant, we encourage that users exercise strong caution when examining RANKGEN outputs for real-world purposes. Before any direct deployment of the system, particularly to any of the four new domains we presented (government reports, mathematics, code generation, and poetry), we propose that RANKGEN be trained on domain-specific datasets and analyzed for performance.

Acknowledgements

We would first like to thank Professor Karthik Narasimhan for his valuable support and feedback as our project mentor and course instructor. We would also like to thank our fellow COS484 classmates, the graduate teaching assistants, and undergraduate course assistants who provided us with useful suggestions and advice for our project. Lastly, we would like to thank our human evaluators for filling out our survey. We could not have done this project without all of your support.

References

Tarun Bisht. 2023. Hugging face dataset: Python code instructions 18k alpaca.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization.

Kalpesh Krishna, Yapei Chang, John Wieting, and Mohit Iyyer. 2022. Rankgen: Improving text generation with large ranking models.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models.

Luca Massarelli, Fabio Petroni, Aleksandra Piktus, Myle Ott, Tim Rocktäschel, Vassilis Plachouras, Fabrizio Silvestri, and Sebastian Riedel. 2020. How decoding strategies affect the verifiability of generated text. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 223–235, Online. Association for Computational Linguistics.

Merve Noyan. 2021. Hugging face dataset: Poetry.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. 2021. Mauve: Measuring the gap between neural text and human text using divergence frontiers.

Veselin Raychev, Martin Vechev, and Eran Yahav. 2014. Code completion with statistical language models. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, page 419–428, New York, NY, USA. Association for Computing Machinery.

Simeng Sun, Kalpesh Krishna, Andrew Mattarella-Micke, and Mohit Iyyer. 2021. Do long-range language models actually use long-range context? In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 807–822, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models.

Hugh Zhang, Daniel Duckworth, Daphne Ippolito, and Arvind Neelakantan. 2021. Trading off diversity and quality in natural language generation. In *Proceedings of the Workshop on Human Evaluation of NLP Systems (HumEval)*, pages 25–33.

Xingxing Zhang and Mirella Lapata. 2014. Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680, Doha, Qatar. Association for Computational Linguistics.

Appendix

Continued on the following page.

Prefix	Ground-truth continuation	RANKGEN continuation
GovReport, ... Table 1 shows seven effects commonly associated with climate change that DOD has documented. According to a 2010 National Research Council report on making informed decisions about climate change and our October 2009 report on climate change adaptation, most decision makers need a basic set of information to understand and make choices about how to adapt to the effects of climate change. This set of information includes information and analysis about observed climate conditions, information about observed climate effects and vulnerabilities, and projections of what climate change might mean for the local area.	In November 2015, we found that in order for climate information to be useful, it must be tailored to meet the needs of each decision maker, such as an engineer responsible for building a bridge in a specific location, a county planner responsible for managing development over a larger region, or a federal official managing a national-scale program. Agencies across the federal government collect and manage many types of climate information, including observational records from satellites and weather monitoring stations on temperature and precipitation, among other things; projections from complex climate models; and tools to make this information more meaningful to decision makers.	This information is needed to understand and make a decision about what actions to take or what measures to take as climate change increases. One way DOD has demonstrated its ability to understand and make informed decisions about climate change is by creating several interactive online climate change resource databases, the DOD Climate Change Database (http://www.drccd.mil) and the DOD Climate Change Research Project (http://www.drccr.mil), which provide this information at no cost and for no consideration.

Table 4: Comparing RANKGEN generations from the GovReport dataset to the ground-truth.

Prefix	Ground-truth continuation	RANKGEN continuation
MetaMathQA, Carl caused a car accident that resulted in $40,000 worth of property damage and $70,000 worth of medical bills. If Carl's insurance company will pay 80% of the costs, leaving Carl to pay the remaining 20%, how much does Carl personally owe?	The total cost of the car accident is $40,000 + $70,000 = $110,000. Carl's insurance company will pay 80% of the costs, so they will cover 80% of $110,000 = $88,000. This means Carl is responsible for the remaining 20% of the costs, which is 20% of $110,000 = $22,000. Therefore, Carl personally owes $22,000. #### 22000 The answer is: 22000	Let's assume Carl is not insured by any insurance company. Let's assume that Carl only lives in the Chicago area for a short period of time. Carl has a good credit score and is currently unemployed. What is Carl's insurance claim history and what does this tell us about Carl's ability to pay his insurance claim? Let's assume that Carl is a current employee of a non-bank financial company and has an auto insurance policy.

Table 5: Comparing RANKGEN generations from the MetaMathQA dataset to the ground-truth.

Prefix	Ground-truth continuation	RANKGEN continuation

Python code instructions, Create a Python script to find the maximum and minimum values for each sublist in a list	def findMinMax(list):
	min_list = []
	max_list = []
	
	for sublist in list:
	min_list.append(min(sublist))
	max_list.append(max(sublist))
	return min_list, max_list
	
	list = [[8, 4], [6, 2, 3], [1, 9, 7]]
	min_list, max_list = find
		(or a tuple of lists, or tuple of lists) #!/usr/bin/env python """ A simple script to find the maximum and minimum values for each sublist in a list or a tuple of lists """ # Find the maximum value from first list list = list [- 1] print'The maximum value is {}'. format (list) print'The minimum value is {}'. format (list) return max (list) # Find the minimum value from first list list = list [- 1] print'The minimum value is {}'.

Table 6: Comparing RANKGEN generations from the Python code instructions dataset to the ground-truth.

Prefix	Ground-truth continuation	RANKGEN continuation
Poetry, Let the bird of loudest lay On the sole Arabian tree Herald sad and trumpet be, To whose sound chaste wings obey. But thou shrieking harbinger, Foul precurrer of the fiend, Augur of the fever's end, To this troop come thou not near. From this session interdict Every fowl of tyrant wing, Save the eagle, feather'd king; Keep the obsequy so strict.	Let the priest in surplice white, That defunctive music can, Be the death-divining swan, Lest the requiem lack his right.	Every bird of tyrician wing,ortium The fowl of the tyranny wing;DeliveryDate These steeds of the bird of thy wing,ocally Thy fount and your lord.

Table 7: Comparing RANKGEN generations from the poetry dataset to the ground-truth.